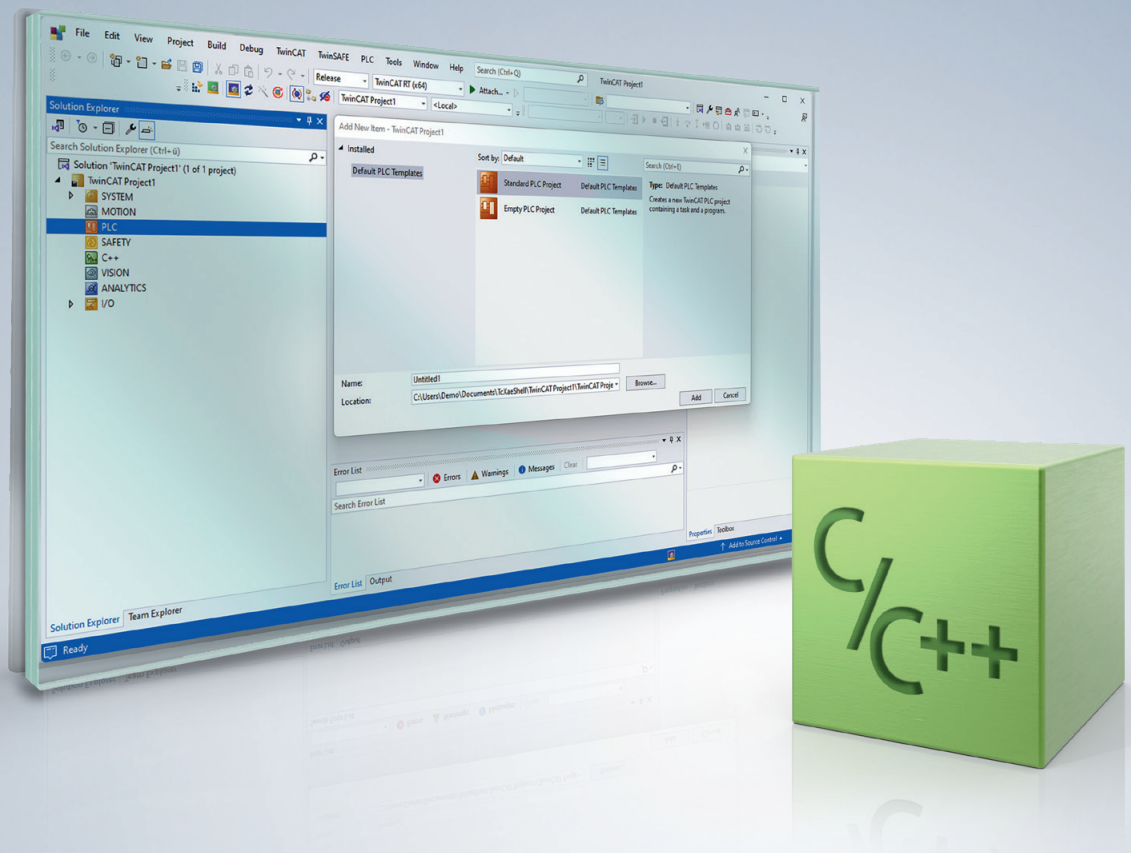
BECKHOFF New Automation Technology

Handbuch | DE

TwinCAT 3

C/C++



22.10.2025 | Version: 1.18.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® und XTS® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Kennzeichnungen führen.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie, lizenziert durch die Beckhoff Automation GmbH, Deutschland.

Copyright

Fremdmarken

In dieser Dokumentation können Marken Dritter verwendet werden. Die zugehörigen Markenvermerke finden Sie unter: https://www.beckhoff.com/trademarks.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

1.4 Ausgabestände der Dokumentation

Version	Änderung
1.18.x	**Neu:** Arbeiten mit TwinCAT Task Dumps [▶ 86] Installation [▶ 21] Online Change per ADS [▶ 245] Third-party components [▶ 359]

2 Übersicht

Dieses Kapitel behandelt die Implementierung von TwinCAT 3 in C/C++. Die wichtigsten Kapitel sind:

- Grundlagen
  Welche Plattformen werden unterstützt? Sind weitere Programme nötig, um TwinCAT 3 C++ Module zu implementieren?
  Die Antworten finden Sie in Anforderungen [▶ 21] und Vorbereitung [▶ 23], es gelten folgende Limitierungen [▶ 157].
- Schnellstart [▶ 62]
  Dies ist ein „weniger als fünf Minuten in Anspruch nehmendes Beispiel“ um einen einfachen, zyklisch ausgeführten Zähler in C++ zu implementieren. Der Zählerwert wird überwacht und überschrieben, es sind Debugging-Möglichkeiten beschrieben, usw.
- MODULE [▶ 41]
  Modularisierung ist die Grundphilosophie von TwinCAT 3. Insbesondere für C++ Module ist ein Verständnis des modularen Konzepts von TwinCAT 3 die Grundvoraussetzung.
  Grundkenntnisse zur Architektur von TwinCAT-Modulen sind erforderlich.
- Assistenten [▶ 89]
  Dokumentation der visuellen Komponenten der TwinCAT C++ Umgebung.
  Hierzu gehören Tools zum Erstellen von Projekten und Tools für die Bearbeitung von Modulen und die Konfiguration von Modulinstanzen.
- Programmierreferenz [▶ 145]
  Dieses Kapitel enthält ausführliche Informationen zur Programmierung in TwinCAT C++. Hier findet man Schnittstellen und andere TwinCAT-Funktionen für ADS-Kommunikation und Hilfsmethoden.
- Das „How to ...? [▶ 227]“ Kapitel enthält nützliche Tipps für das Arbeiten mit TwinCAT C++.
- Beispiele
  Einige Schnittstellen und deren Verwendung sind ausführlich in Form eines ausführbaren Programms beschrieben, das als Download mit Quellcode und Solution zur Verfügung gestellt wird.

3 Einleitung

Die Methode, klassische Automatisierungsgeräte wie speicherprogrammierbare Steuerungen (SPS) und numerische Steuerungen (NC) als Software auf leistungsstarker Standard-Hardware nachzubilden, ist seit vielen Jahren der Stand der Technik und wird nun von vielen Herstellern praktiziert.

Es gibt viele Vorteile, aber der wichtigste ist sicher, dass die Software weitgehend hardwareunabhängig ist. Das heißt zum Einen, dass die Leistungsfähigkeit der Hardware speziell an die Anwendung angepasst werden kann und zum Anderen, dass man automatisch von ihrer Weiterentwicklung profitieren kann.

Dies gilt insbesondere für PC-Hardware, deren Leistung noch immer dramatisch schnell ansteigt. Die relative Unabhängigkeit von einem Lieferanten, die aus dieser Trennung von Software und Hardware resultiert, ist für den Nutzer auch sehr wichtig.

Da SPS und Bewegungssteuerung - und möglicherweise andere Automatisierungskomponenten - bei dieser Methode unabhängige Logikbausteine bleiben, gibt es im Vergleich zur klassischen Automatisierungstechnologie nur wenige Änderungen in der Anwendungsarchitektur.

Die SPS bestimmt die logischen Abläufe der Maschine und überträgt der Bewegungssteuerung die Implementierung bestimmter Achsfunktionen. Dank der verbesserten Leistung der Steuerungen und der Möglichkeit, höhere Programmiersprachen (IEC 61131-3) zu verwenden, können auf diese Weise auch komplexe Maschinen automatisiert werden.

Modularisierung

Um die Komplexität moderner Maschinen zu meistern und gleichzeitig den dafür nötigen Engineering-Aufwand zu senken, haben viele Hersteller begonnen, ihre Maschinen zu modularisieren. Individuelle Funktionen, Baugruppen oder Maschineneinheiten werden dabei als Module betrachtet, die untereinander so unabhängig wie möglich sind und über einheitliche Schnittstellen in das Gesamtsystem eingebettet werden.

Eine Maschine ist danach idealerweise hierarchisch strukturiert, wobei Module der niedrigsten Ebene einfachste, ständig wiederverwendbare Grundelemente repräsentieren. Miteinander verbunden bilden sie immer komplexere Maschineneinheiten, bis auf höchster Ebene die gesamte Maschine entsteht. Für Steuerungssysteme bei Maschinenmodularisierung gibt es unterschiedliche Herangehensweisen. Sie können grob in eine dezentralisierte und eine zentralisierte Vorgehensweise unterteilt werden.

Beim lokalen Ansatz hat jede Maschine ihre eigene Steuerung, die die SPS- und möglicherweise auch die Bewegungsfunktionen des Moduls übernimmt.

Die individuellen Module können getrennt voneinander in Betrieb genommen und gewartet werden. Sie können relativ unabhängig voneinander skaliert werden. Die notwendigen Interaktionen zwischen den Steuerungen werden über Kommunikationsnetzwerke (Feldbusse oder Ethernet) koordiniert und über geeignete Profile standardisiert.

Der zentralisierte Ansatz konzentriert die Steuerungsfunktionen aller Module in einer gemeinsamen Steuerung und nutzt die Rechenintelligenz der lokalen I/O-Geräte nur sehr wenig. Interaktionen können in der zentralen Steuerungseinheit viel direkter ausgeführt werden, da die Kommunikationswege viel kürzer sind. Es treten keine Totzeiten auf und die Steuerungshardware wird viel besser ausgenutzt, was die Gesamtkosten senkt.

Allerdings hat die zentralisierte Methode auch den Nachteil, dass eine Modularisierung der Steuerungssoftware nicht automatisch notwendig ist. Die Möglichkeit, auf alle Informationen anderer Programmteile in der zentralen Steuerung zugreifen zu können, bietet keinen Anreiz zum Aufbau der Steuerungssoftware aus Modulen, die in anderen Anwendungen wiederverwendet werden können. Da es zwischen den Steuerungseinheiten keinen Kommunikationskanal gibt, bleiben die Entwicklung eines geeigneten Profils und die Standardisierung der Steuerungseinheiten häufig auf der Strecke.

Das Beste aus beiden Welten

Die ideale Steuerung für modulare Maschinen nutzt Elemente aus dezentralisierter und zentralisierter Steuerungsarchitektur. Eine zentrale, leistungsstarke und möglichst allgemeine Computerplattform dient „wie üblich“ als Steuerungshardware.

Die Vorteile einer zentralisierten Steuerungstechnologie:

- geringe Gesamtkosten
- verfügbar
- schnelles, modulares Feldbussystem (Stichwort EtherCAT)
- und die Möglichkeit, auf alle Informationen im System ohne Kommunikationsverlust zugreifen zu können

sind entscheidende Argumente.

Die oben aufgeführten Vorteile einer dezentralisierten Herangehensweise können in der zentralisierten Steuerung durch geeignete Modularisierung der Steuerungssoftware umgesetzt werden.

Anstelle ein großes, komplexes SPS-Programm und eine NC mit vielen Achsen ablaufen zu lassen, können viele kleine „Steuerungen“ in einer gemeinsamen Laufzeit nebeneinander auf der gleichen Hardware relativ unabhängig voneinander koexistieren. Die einzelnen Steuerungsmodule sind in sich abgeschlossen und stellen der Umgebung ihre Funktionen über Standard-Schnittstellen zur Verfügung oder nutzen entsprechende Funktionen anderer Module oder der Laufzeit.

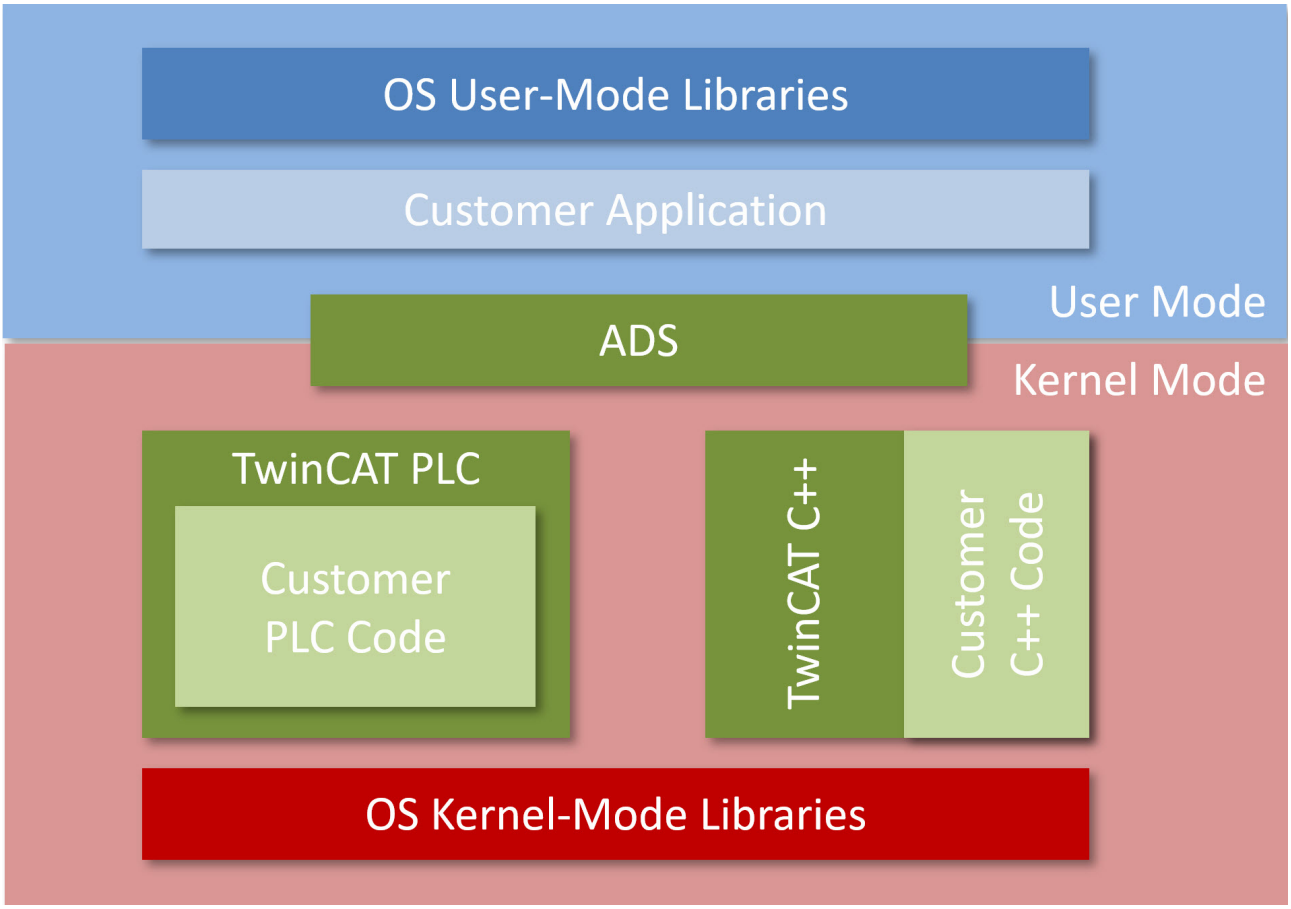
Durch die Definition dieser Schnittstellen und die Standardisierung der entsprechenden Parameter und Prozessdaten entsteht ein signifikantes Profil. Da die einzelnen Module in eine Laufzeit implementiert sind, sind auch direkte Aufrufe anderer Module möglich - wiederum über entsprechende Standardschnittstellen. Die Modularisierung kann so in sinnvollen Grenzen stattfinden, ohne dass Kommunikationsverluste auftreten.

Während der Entwicklung oder Inbetriebnahme einzelner Maschinenmodule können die entsprechenden Steuerungsmodule erstellt und auf beliebiger Steuerungshardware mit geeigneter Laufzeit getestet werden. Fehlende Anbindungen an andere Module können in dieser Phase emuliert werden. Bei der kompletten Maschine werden sie dann zusammen auf der zentralen Laufzeit instanziiert, die lediglich so dimensioniert sein muss, dass der Bedarf an Ressourcen für alle instanziierten Module (Speicher, Tasks und Rechenleistung) erfüllt wird.

TwinCAT 3 Runtime

TwinCAT Runtime bietet eine Softwareumgebung in der die TwinCAT Module geladen, implementiert und verwaltet werden. Sie bietet weitere Basisfunktionen, sodass die Systemressourcen (Speicher, Tasks, Feldbus und Hardwarezugang, usw.) genutzt werden können. Die einzelnen Module müssen nicht mit demselben Compiler erstellt worden sein. Damit können sie voneinander unabhängig sein und von unterschiedlichen Anbietern stammen.

Beim Start der Laufzeit werden automatisch einige Systemmodule geladen, sodass ihre Eigenschaften anderen Modulen zur Verfügung stehen. Allerdings erfolgt der Zugriff auf die Eigenschaften der Systemmodule auf die gleiche Weise wie auf die Eigenschaften normaler Module, sodass es für die Module keine Rolle spielt, ob die jeweilige Eigenschaft von einem Systemmodul oder einem normalen Modul zur Verfügung gestellt wird.

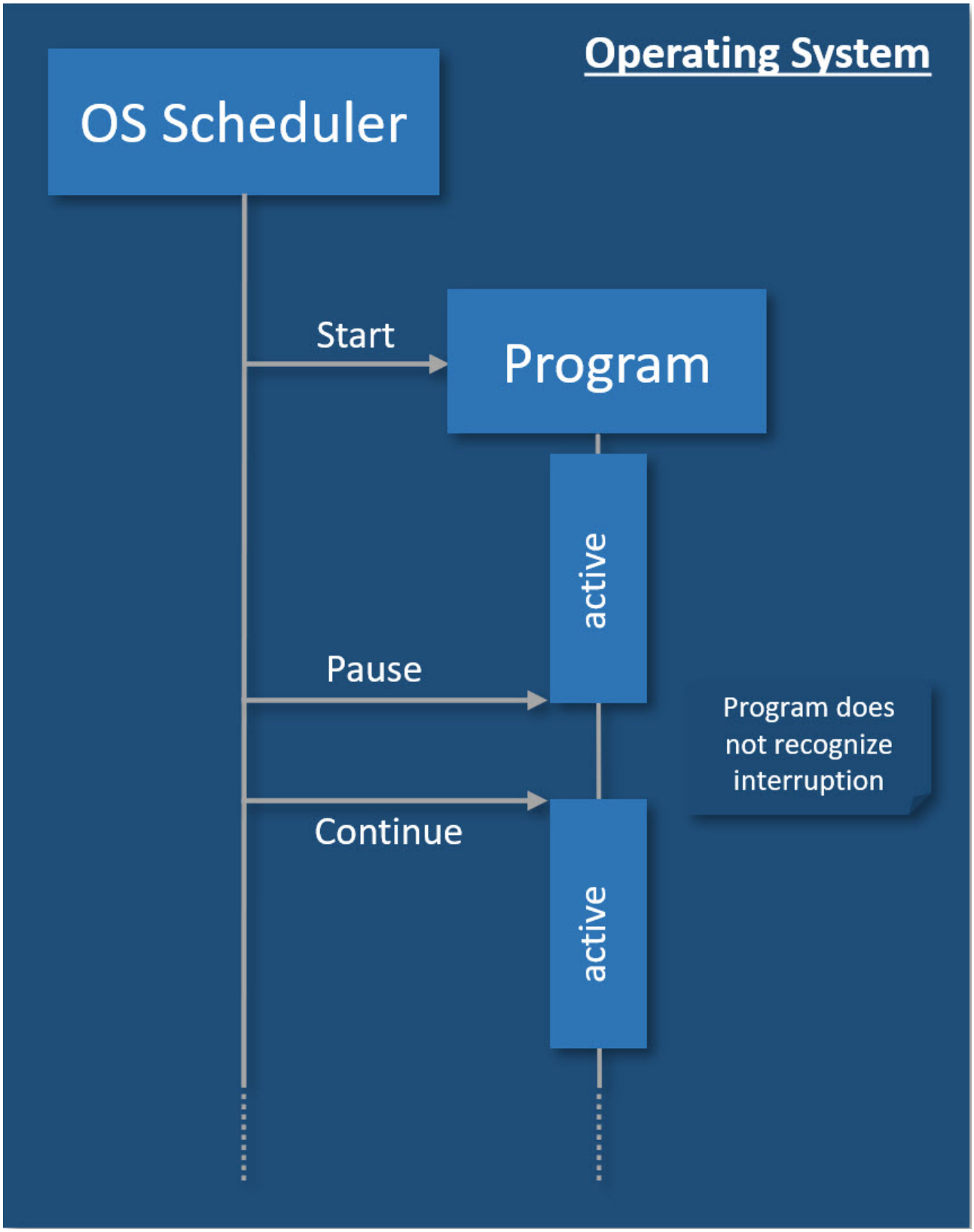


Im Gegensatz zur SPS, wo das benutzerdefinierte Programm in einer Laufzeitumgebung ausgeführt wird, sind TwinCAT C++ Module nicht in so einer gehosteten Umgebung. Dadurch werden TwinCAT C++ Module (.tmx) als Kernel-Module ausgeführt, die von TwinCAT geladen werden.

3.1 Unterschied Usermode- zur Echtzeit-Programmierung

Dieser Artikel soll die Unterschiede der Programmierkonzepte beschreiben, die zwischen einer normalen Usermode-Programmierung in einer Programmiersprache wie C++, C# oder Java zu der Echtzeit-Programmierung in TwinCAT bestehen.

Dieser Artikel bezieht sich dabei insbesondere auf die TwinCAT-Echtzeit-Programmierung mit TwinCAT C++, da gerade hier auf der einen Seite wesentliche Vorkenntnisse der C++ Programmierung eingebracht werden können, auf der anderen Seite die Ablaufeigenschaften des TwinCAT Echtzeitsystems Berücksichtigung finden müssen.

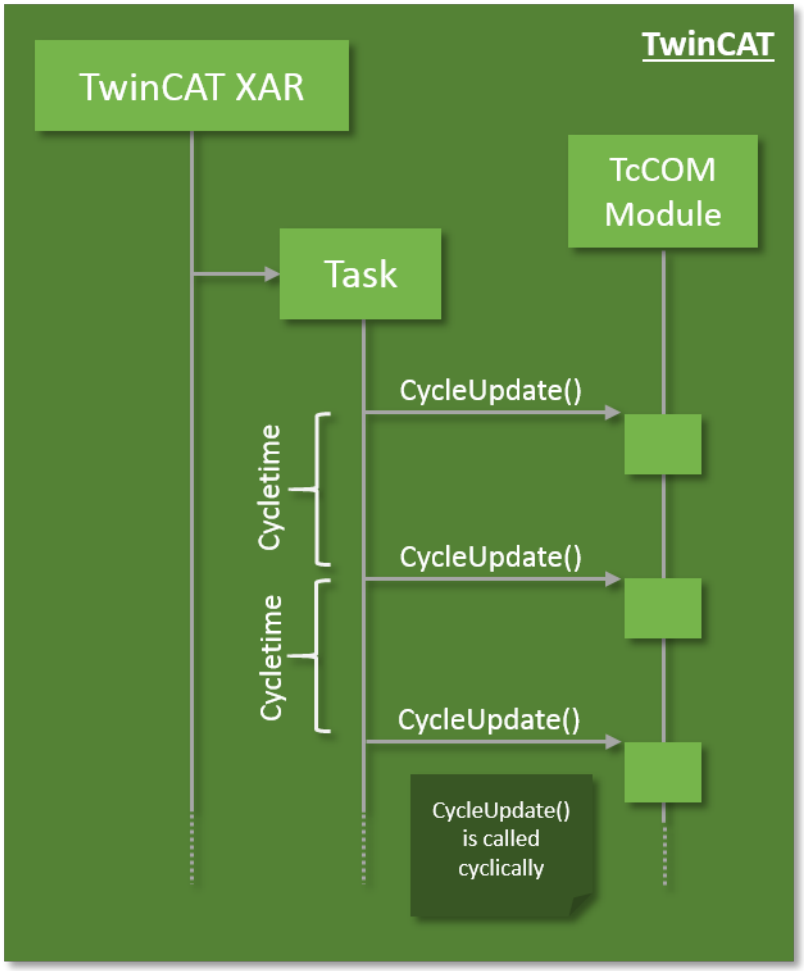


Beim klassischen Usermode-Programmieren z.B. in C# wird ein Programm erstellt, welches von einem Betriebssystem ausgeführt wird.

Das Programm wird hierbei vom Betriebssystem gestartet und kann selbstständig ablaufen, d.h. auch es hat die volle Kontrolle auf seinen eigenen Ablauf wie Threading und Speicherverwaltung. Um Multitasking zu ermöglichen unterbricht das Betriebssystem ein solches Programm zu einem beliebigen Zeitpunkt und für eine beliebige Zeit. Das Programm bekommt eine solche Unterbrechung nicht mit. Es ist Aufgabe des

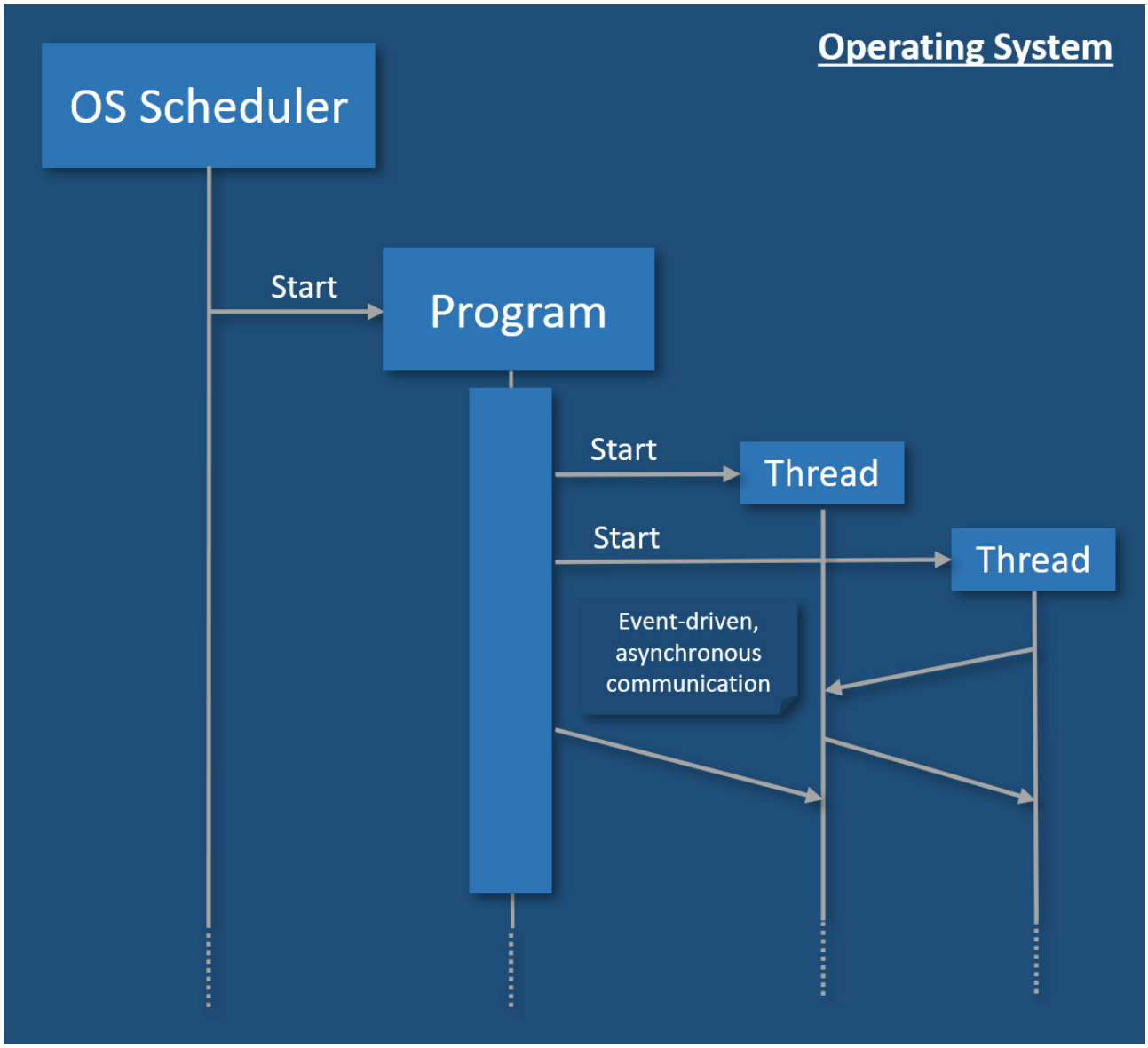
Betriebssystems dafür zu sorgen, dass der Nutzer von diesen Unterbrechungen nichts mitbekommt. Der Datenaustausch des Programms mit der Umwelt erfolgt dabei Event-getrieben, d.h. nicht-deterministisch und häufig blockierend.

Für einen Ablauf unter Echtzeitanforderungen ist das Verhalten nicht hinreichend, denn die Anwendung selbst muss sich auf die bereitstehenden Ressourcen verlassen können um Echtzeiteigenschaften (Antwortgarantien) zusichern zu können.



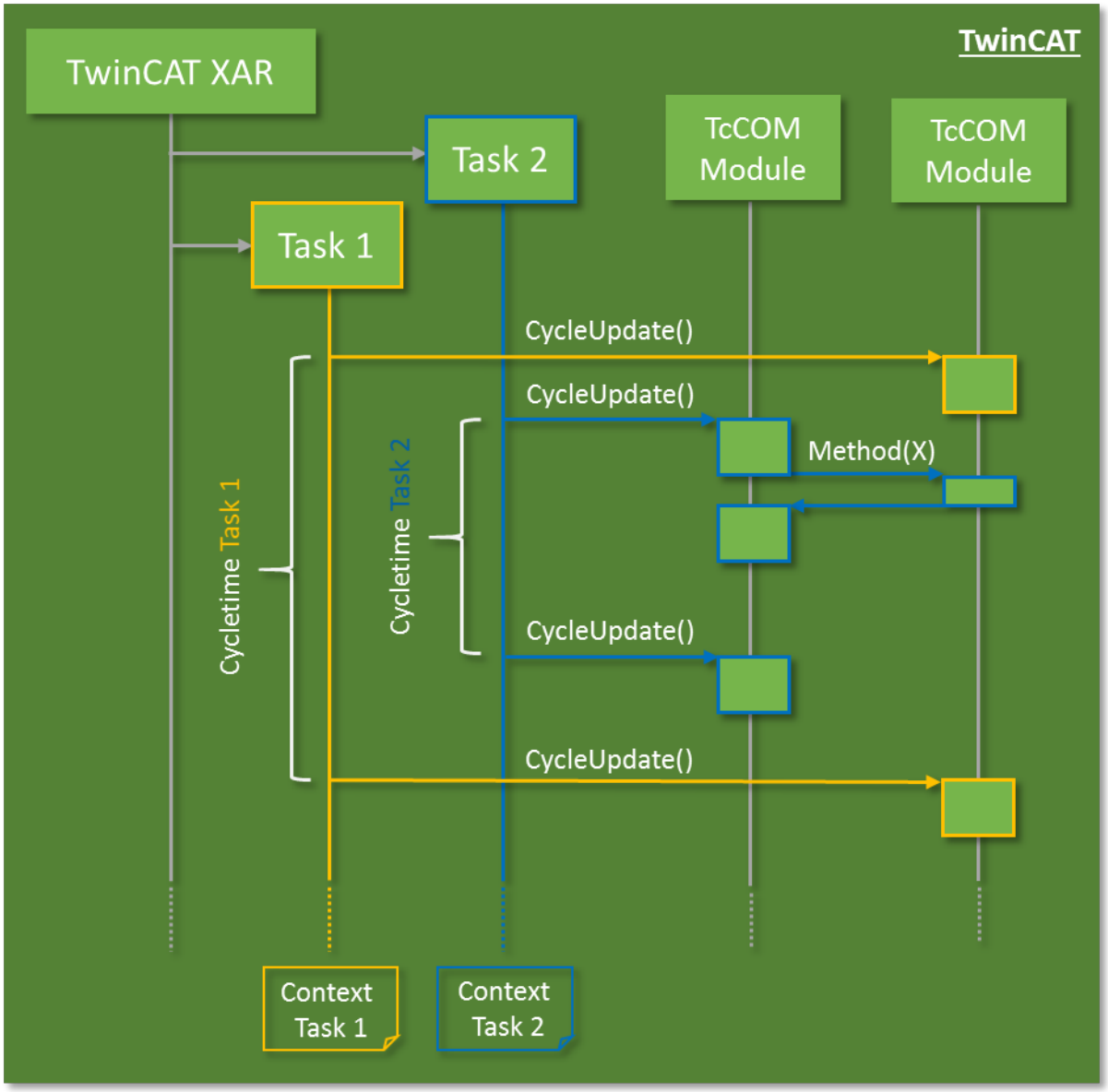
Für TwinCAT C++ wird deswegen der Grundgedanke aus der SPS aufgenommen: Das TwinCAT Echtzeitsystem verwaltet die Echtzeit Tasks, übernimmt das Scheduling und ruft einen Einstiegspunkt in dem Programmcode zyklisch auf. Das Programm muss innerhalb der bereitstehenden Zykluslänge abgearbeitet sein und die Kontrolle zurückliefern. Das TwinCAT System stellt dabei die Daten aus dem I/O Bereich in Prozessabbildern bereit, so dass ein konsistenter Zugriff garantiert werden kann. Hieraus ergibt sich, dass der Programmcode selbst keine Mechanismen wie Threading verwenden kann.

Nebenläufigkeit



Bei der klassischen Programmierung im Usermode liegt die Kontrolle der Nebenläufigkeit im Bereich des Programms. Hier werden Threads gestartet und diese kommunizieren untereinander. All diese Mechanismen benötigen Ressourcen, die allokiert und freigegeben werden müssen, was die Echtzeitfähigkeit gefährden kann. Die Kommunikation zwischen den Threads erfolgt Event-basiert, sodass ein aufrufender Thread keine Kontrolle über den Abarbeitungszeitpunkt im aufgerufenen Thread besitzt.

In TwinCAT werden Tasks genutzt, um Module aufzurufen, was somit eine Nebenläufigkeit darstellt. Tasks sind jeweils einem Core zugeordnet und haben Zykluszeiten sowie Prioritäten, was dazu führt, dass eine höher priore Task eine niedriger priore Task unterbrechen kann. Bei Verwendung von mehreren Cores werden Tasks tatsächlich nebenläufig ausgeführt.



Module können untereinander kommunizieren, sodass bei Nebenläufigkeit die Datenkonsistenz sichergestellt werden muss.
Einen Datenaustausch über die Task-Grenzen hinweg wird z.B. durch das Mapping ermöglicht. Bei direktem Zugriff mittels Methoden ist der Datenzugriff z.B. durch CriticalSections zu schützen.

Startup-/Shutdown-Verhalten

Der TwinCAT C++ Code wird im sogenannten „Kernel Kontext“ und „TwinCAT Echtzeit-Kontext“ und nicht als UserMode-Anwendung ausgeführt.

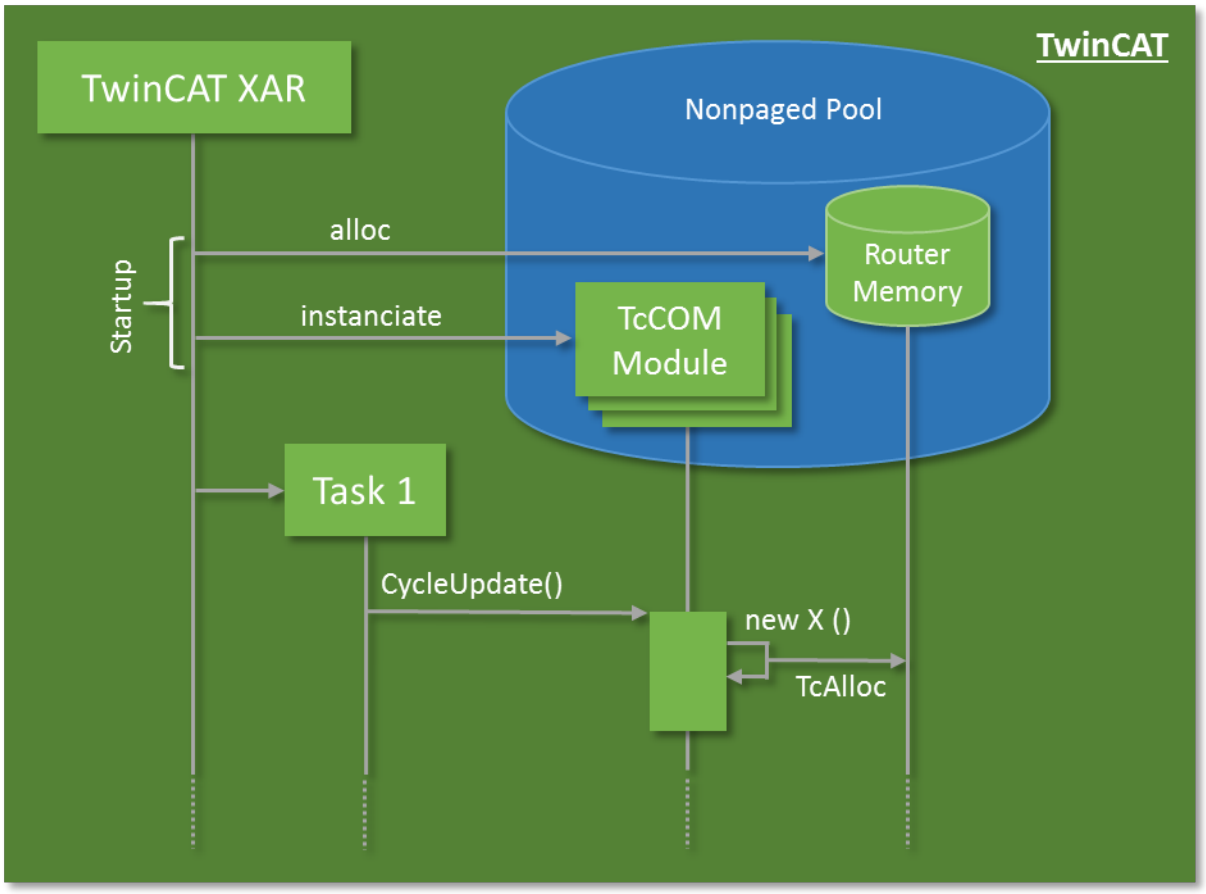
Beim Hoch-/Runterfahren der Module wird Code zur (De)Initialisierung zunächst im Kernel Kontext ausgeführt; erst die letzte Phase sowie die zyklischen Aufrufe werden im TwinCAT Echtzeit-Kontext ausgeführt.

Details sind im Kapitel Modul-Statemachine [▶ 48] beschrieben.

Speichermanagement

TwinCAT bringt ein eigenes Speichermanagement mit, welches auch im Echtzeit Kontext verwendet werden kann. Dieser Speicher wird aus dem so genannten „Nonpaged Pool“, der durch das Betriebssystem bereitgestellt wird, bezogen. In diesem Speicher werden die TcCOM Module mit ihrem Speicherbedarf instanziiert.

Zusätzlich wird der sogenannte „Routerspeicher“ in diesem Speicherbereich durch TwinCAT bereitgestellt, aus dem im Echtzeit-Kontext dynamisch von den TcCOM Modulen Speicher allokiert werden kann (z.B. mit Operator new).



Generell sollte Speicher möglichst vorab allokiert werden und nicht im zyklischen Code. Bei jeder Allokation muss geprüft werden, ob der Speicher auch wirklich zur Verfügung steht. Bei Allokationen im zyklischen Code hängt der Ablauf dann also von der Verfügbarkeit des Speichers ab.

4 Installation

4.1 Systemvoraussetzungen

Engineering (XAE)

Technische Daten	Voraussetzungen
Betriebssystem	Windows 10/11
Zielplattform	x64
TwinCAT-Version	TwinCAT 3.1 Build 4024 | Build 4026
Erforderliche TwinCAT-Lizenz	Keine (Visual Studio-Lizenz von Microsoft)

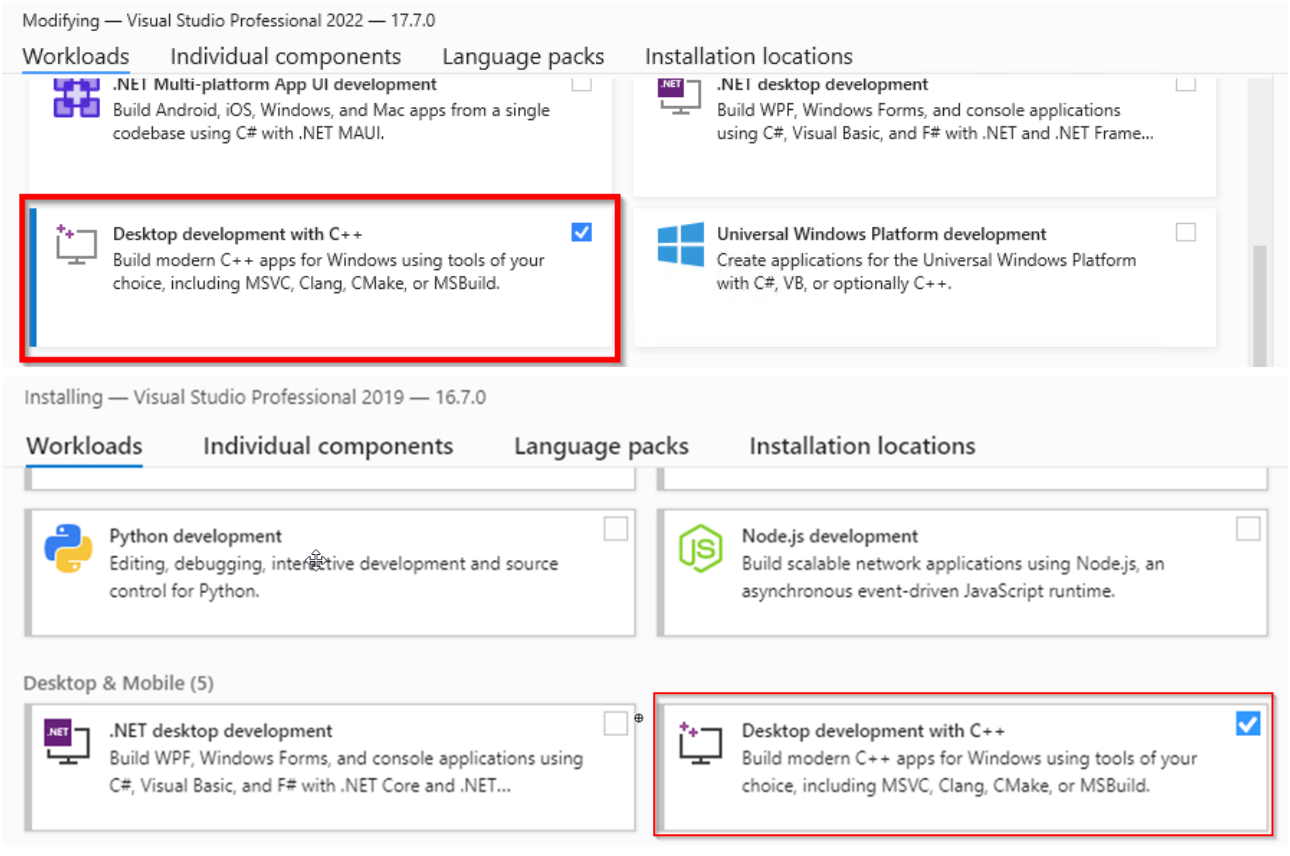
Runtime (XAR)

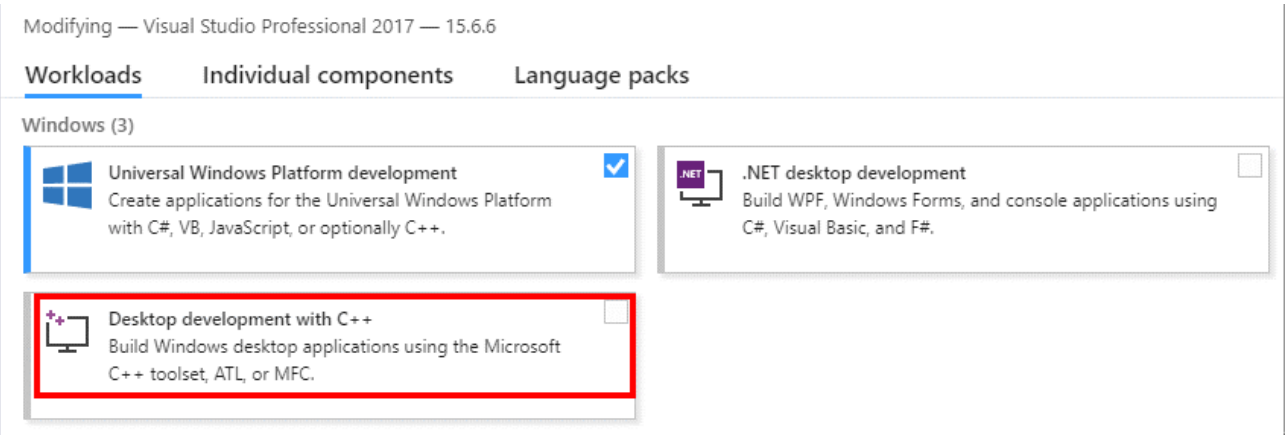
Technische Daten	Voraussetzungen
Betriebssystem	Windows 10, TwinCAT/BSD
Zielplattform	x64
TwinCAT-Version	TwinCAT 3.1 Build 4024 | Build 4026
Erforderliche TwinCAT-Lizenz	TC1300

4.2 Installation

Voraussetzungen Engineering

Auf dem Engineering PC muss Microsoft Visual Studio® 2017, 2019 oder 2022 Professional/ Enterprise installiert sein.
Im Visual Studio® Installer muss zusätzlich die Option zu **Desktop development with C++** ausgewählt werden, da diese Option bei der automatischen Installation nicht selektiert ist:





Für die Integration und Nutzung bestehender, binärer C++-Module in eine TwinCAT 3 SPS-Umgebung ist die XAE-Shell ausreichend (kein Visual Studio® erforderlich).

Voraussetzungen Runtime

Microsoft Visual Studio® muss **nicht** installiert werden.

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

TwinCAT Package Manager: Installation (TwinCAT 3.1 Build 4026)

Eine ausführliche Anleitung zur Installation von Produkten finden Sie im Kapitel Workloads installieren in der Installationsanleitung TwinCAT 3.1 Build 4026.

Installieren Sie den folgenden Workload, um das Produkt nutzen zu können:

TC1300 | TwinCAT 3 C/C++

TwinCAT Setup: Installation (TwinCAT 3.1 Build 4024 und früher)

Ist im Setup TwinCAT 3.1 Full enthalten.

TwinCAT/BSD:

keine Installation notwendig

5 Vorbereitung - nur einmal

Ein PC für das Engineering von TwinCAT C++-Modulen muss vorbereitet werden. Diese Schritte müssen Sie nur einmal durchführen:

- Konfigurieren Sie die TwinCAT Basis [▶ 23] sowie die Konfigurations- und Plattform-Symbolleiste.
- Signieren Sie Module, damit sie ausgeführt werden können, siehe Dokumentation zum Setup einer Testsignierung [▶ 23].

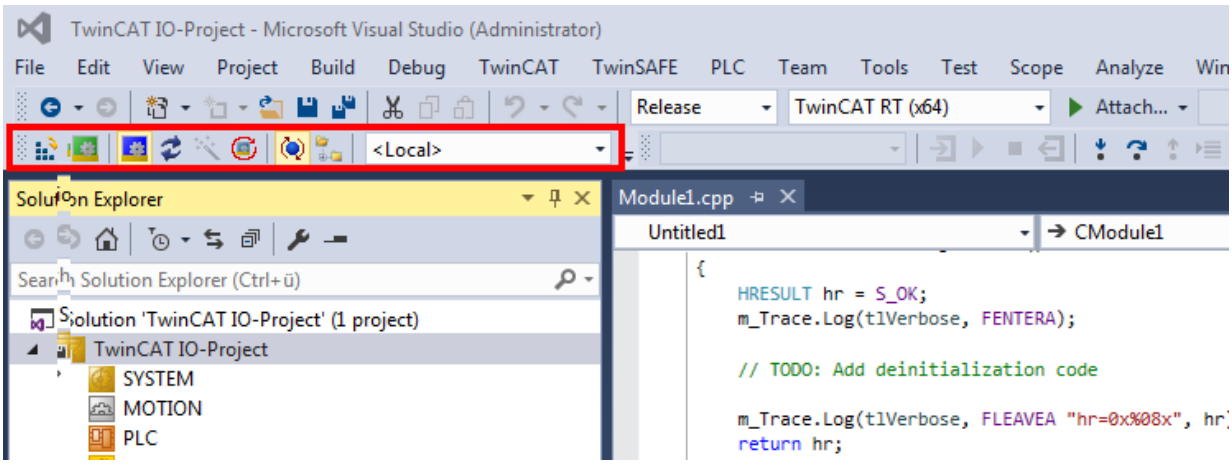
5.1 Visual Studio - TwinCAT XAE Base-Symbolleiste

Effizientes Engineering durch TwinCAT XAE Base-Symbolleiste

TwinCAT 3 integriert für eine bessere Effizienz seine eigene Symbolleiste in das Visual Studio-Menü, die Sie bei der Erstellung von C++-Projekten unterstützt. Diese Symbolleiste wird vom TwinCAT 3 Setup automatisch zum Visual Studio-Menü hinzugefügt. Wenn Sie sie allerdings manuell hinzufügen möchten, tun Sie folgendes:

1. Öffnen Sie das Menü **View** und wählen **Toolbars\TwinCAT XAE Base**

⇨ Die ausgewählte Symbolleiste erscheint unter dem Menü.



5.2 Modul-Signierung

TwinCAT C++-Module müssen durch ein Zertifikat signiert werden, damit sie ausgeführt werden können.

Durch die Signatur wird sichergestellt, dass auf Produktiv-Systemen nur C++-Software ausgeführt wird, deren Herkunft nachvollziehbar ist.

Hierbei werden die C++-Module durch das TwinCAT-Laufzeitsystem geladen und müssen dafür mit einem TwinCAT-Nutzerzertifikat signiert sein.

Zu Testzwecken können Zertifikate zum Signieren verwendet werden, die nicht überprüft werden können. Dies ist allerdings nur möglich, wenn das Betriebssystem im Testmodus ist, damit diese Zertifikate nicht auf Produktiv-Systemen genutzt werden.

Engineering erfordert keine Signierung

Lediglich die Ausführung erfordert Zertifikate - das Engineering nicht.

Organisatorische Trennung von Entwicklungs- und Produktiv-Software

Beckhoff empfiehlt organisatorisch mit (mindestens) zwei Zertifikaten zu arbeiten.

1. Ein Zertifikat, welches nicht gegengezeichnet ist, somit den Testmodus benötigt, für den Entwicklungsprozess. Dieses Zertifikat kann sich ggf. auch individuell jeder Entwickler selbst ausstellen. Die Testsysteme werden dann in den Testmodus gesetzt.
2. Nur die Software, welche durch entsprechende finale Tests gelaufen ist, wird durch ein gegengezeichnetes Zertifikat signiert. Diese Software kann damit auch auf Maschinen aufgespielt und ausgeliefert werden.

Eine solche Trennung der Entwicklung und des Betriebes stellt sicher, dass nur getestete Software auf produktiven Anlagen läuft.

Sehen Sie dazu auch

- Online-Change [▶ 155]
- Versionierte C++ Projekte [▶ 53]

5.2.1 TwinCAT

Versionierte C++ Projekte werden als Binary in einer TMX-Datei (TwinCAT Module Executeable) abgelegt.

Für die Implementierung von TwinCAT 3 C++ Modulen muss diese kompilierte, ausführbare TMX Datei mit einem TwinCAT-Nutzerzertifikat signiert sein, wenn sie durch die TwinCAT Runtime geladen werden soll.

Signierung

TwinCAT TMX Dateien können nur nach einer erfolgreichen Signierung geladen werden.

HINWEIS

Signierung auf 32bit und 64bit Systemen

Im Gegensatz zu der Betriebssystem-Signierung ist die TwinCAT Signierung sowohl auf 32bit wie auch auf 64bit Systemen vorgesehen. Damit wird für eine Testsignierung auch auf 32bit Systemen der Testmodus vorausgesetzt.

Für die Signierung einer TMX Datei wird ein TwinCAT Nutzerzertifikat benötigt [▶ 24], welches entsprechend im Projekt zur Signierung [▶ 151] konfiguriert wird.

Testsignierung

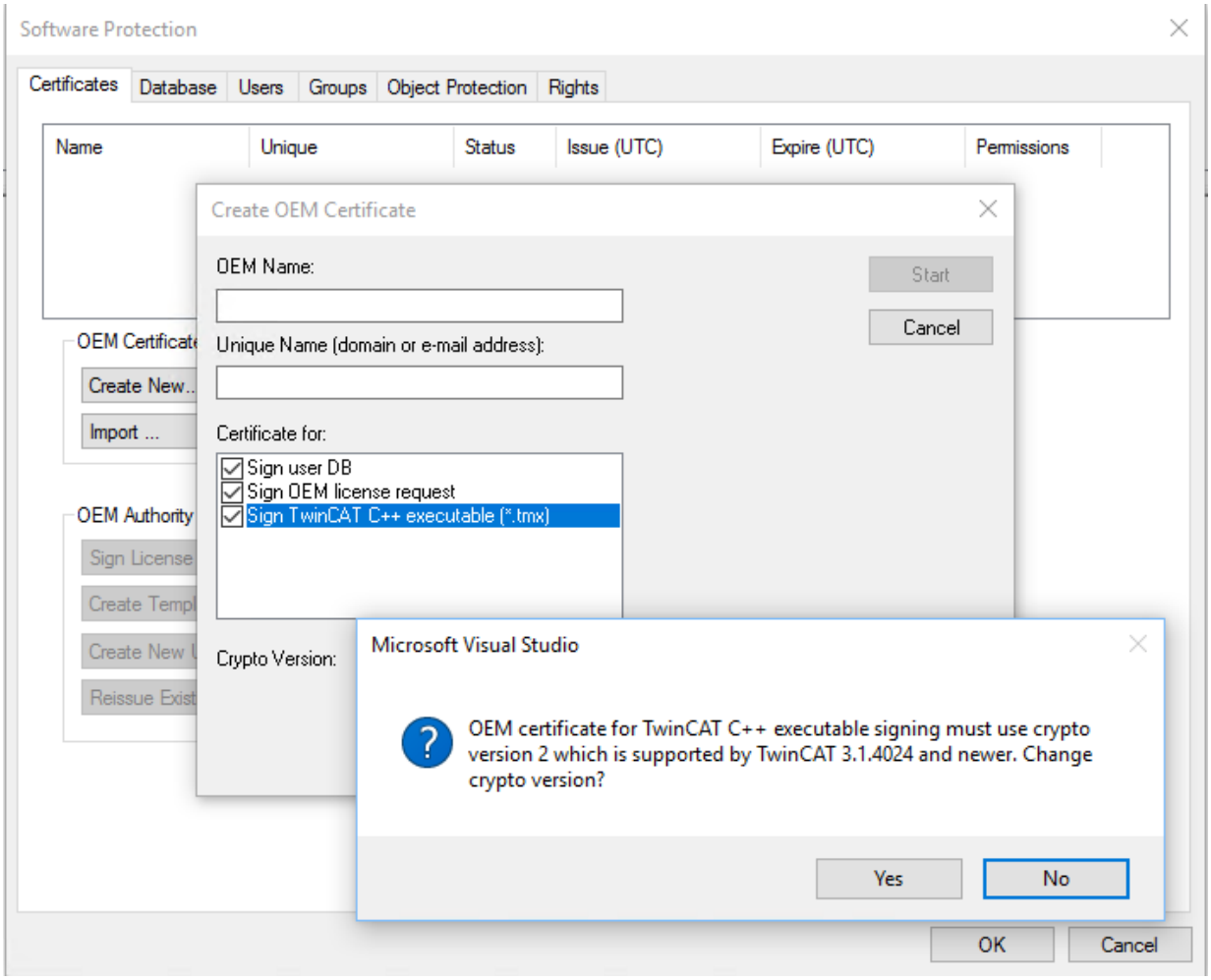
Das Nutzerzertifikat kann lokal in TwinCAT erstellt werden. Solange es nicht von Beckhoff gegengezeichnet ist, ist es nötig, den Testmode [▶ 24] zu aktivieren.

Sobald das TwinCAT-Nutzerzertifikat von Beckhoff gegengezeichnet [▶ 26] wurde, kann auf den Testmodus entsprechend verzichtet werden. Er kann analog zum aktivieren auch wieder deaktiviert werden.

5.2.1.1 Testsignierung

Die Testsignierung kann für TwinCAT mit dem gleichen TwinCAT-Nutzerzertifikat erfolgen, wie bei der eigentlichen Auslieferung (vgl.TwinCAT 3 Nutzerzertifikat beantragen [▶ 27]).

1. Für einen Testbetrieb z. B. während der Software-Entwicklung genügt die Erstellung eines TwinCAT-Nutzerzertifikates [▶ 28]. Beachten Sie dabei, dass Sie den Verwendungszweck „Sign TwinCAT C++ executeable (*.tmx)“ wählen. Hierfür ist dann die Crypto Version 2 erforderlich, eine Meldung dazu erscheint.

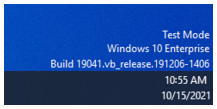


Auf XAR (und XAE, wenn es sich um einen lokalen Test handelt) aktivieren Sie den Testmodus, damit das Betriebssystem die selbstsignierten Zertifikate akzeptieren kann. Dies kann sowohl auf dem Engineering System (XAE) als auch auf dem Laufzeit-Systemen (XAR) vorgenommen werden.

Für Windows

Führen Sie mittels Administrator-Eingabeaufforderung folgendes aus:
`bcdedit /set testsigning yes`
und starten Sie das Zielsystem neu.
Es kann sein, dass Sie hierfür „SecureBoot“ abschalten müssen, was im Bios erfolgen kann.

Wenn der Testsigniermodus aktiviert ist, wird das rechts unten auf dem Desktop angezeigt. Das System akzeptiert nun alle signierten Treiber zur Ausführung.



Für TwinCAT/BSD

In der Datei `/usr/local/etc/TwinCAT/3.1/TcRegistry.xml` den Eintrag `"<Value Name="EnableTestSigning" Type="DW"1</Value>"` unter Key `"System"` eintragen.

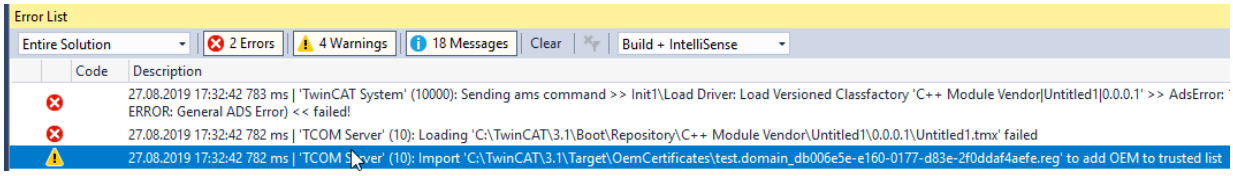
```
<Key Name="System">
  <Value Name="RunAsDevice" Type="DW">1</Value>
  <Value Name="RTimeMode" Type="DW">0</Value>
  <Value Name="AmsNetId" Type="BIN">052445B00101</Value>
  <Value Name="LockedMemSize" Type="DW">33554432</Value>
  <Value Name="EnableTestSigning" Type="DW">1</Value>
</Key>
```

Danach den TwinCAT System Service neu starten:

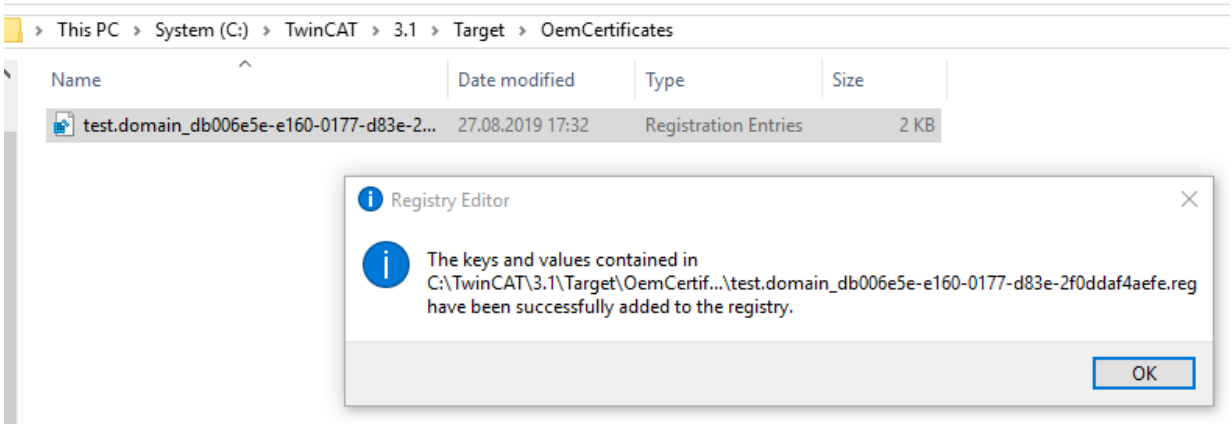
```
doas service TcSystemService restart
```

Nach der jeweiligen Vorgehensweise akzeptiert das System alle signierten Treiber zur Ausführung.

1. Bei dem ersten Aktivieren (Activate Configuration) mit einem TwinCAT-Nutzerzertifikat wird von dem Zielsystem festgestellt, dass dem Zertifikat nicht vertraut wird und der Aktivierungsvorgang wird abgebrochen:



Für Windows:
Ein lokaler Nutzer mit Administrationsrechten kann über die erstellte REG-Datei durch einfaches Ausführen dem Zertifikat vertrauen:



Für TwinCAT/BSD:
Falls das Paket „Tcimportcert“ nicht installiert ist dieses installieren: `pkg install tcimportcert`
Mittels `doas tcimportcert /usr/local/etc/TwinCAT/3.1/Target/OemCertificates/<CreatedFile>.reg` dem Zertifikat vertrauen.
Danach den TwinCAT System Service neu starten oder das System neu starten:
`doas service TcSystemService restart`

   ⇨ Diese Verfahren ermöglichen es nur, C++ Module mit einer Signatur von den vertrauten TwinCAT-Nutzerzertifikaten in die Ausführung zu bringen.

2. Nach diesem Prozess können Sie das TwinCAT-Nutzerzertifikat zur Signierung mit dem Testmodus des Betriebssystems verwenden.
   Dieses wird in den Projekt-Eigenschaften [▶ 151] konfiguriert.
   Um das Passwort des TwinCAT-Nutzerzertifikates nicht im Projekt abzulegen, wo es beispielsweise auch in einer Versionsverwaltung landen würde, verwenden Sie das TcSignTool [▶ 59].

Wenn Sie das TwinCAT-Nutzerzertifikat ohne TestMode zur Auslieferung nutzen wollen, müssen Sie das Zertifikat von Beckhoff gegenzeichnen [▶ 26] lassen.

5.2.1.2 Nutzerzertifikate zur Auslieferung ohne Testmode

Systemvoraussetzungen
- Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 oder TwinCAT/BSD (auf dem Zielsystem)

Mit TwinCAT Build 4024 bietet Beckhoff Bestandskunden die Ausstellung eines „TwinCAT 3 OEM-Nutzerzertifikates“ an, das für die Signierung von mit TwinCAT 3 in C++ erstellte TMX-Dateien verwendet werden kann.

- Dieses Zertifikat erfordert durch die Nutzung im Windows-Umfeld eine sichere Validierung der Antragstellerdaten. TwinCAT 3 Nutzerzertifikate müssen daher zur Validierung der Adress- und Kontaktdaten offiziell bestellt werden, und werden nur an Beckhoff Bestandskunden vergeben.
- Bestellnummer: **TC0008**
- Die Ausstellung dieses TwinCAT 3 Nutzerzertifikates ist kostenlos.
- Verzeichnis zum Speichern des Zertifikates: **C:\TwinCAT\3.1\CustomConfig\Certificates**

Das TwinCAT 3 Nutzerzertifikat ist nicht für die Nutzung der TwinCAT 3 TMX-Dateien erforderlich

Das TwinCAT 3 Nutzerzertifikat wird ausschließlich für die einmalige Signierung der TMX-Dateien verwendet., und ist nicht für die Nutzung der damit signierten TMX-Dateien erforderlich.

Auf welchen Rechnern ist das TwinCAT 3 Nutzerzertifikat TC0008 erforderlich?

Das TwinCAT 3 Nutzerzertifikat sollte sich ausschließlich auf dem Engineering-Rechner befinden, auf dem die Signierung der TMX-Dateien erfolgt – also **NICHT** auf jedem Zielsystem.

Gültigkeit des TwinCAT 3 Nutzerzertifikats

Die Gültigkeit des TwinCAT 3 Nutzerzertifikats ist aus Sicherheitsgründen auf zwei Jahre begrenzt.

Was passiert, wenn die Gültigkeit des Zertifikats abgelaufen ist?

Sie können neue TMX-Dateien nicht mehr signieren.
Die Nutzung der bereits signierter TMX-Dateien ist aber ohne Einschränkung weiter möglich.

Sie können vor Ablauf der zwei Jahre (und auch noch danach) eine Verlängerung Ihres Zertifikats beantragen.

Um ein TwinCAT 3 Nutzerzertifikat zu verlängern, gilt der gleiche Prozess wie bei der Beantragung eines neuen Zertifikats. Auch in diesem Fall muss das Zertifikat bestellt werden (die Bestellnummern für eine Zertifikatsverlängerung sind dieselben wie zur Zertifikatsneubeantragung).

Sie erzeugen in diesem Falle aber kein neues „OEM Certificate Request File“, sondern schicken Ihr bestehendes Zertifikat zur Verlängerung an die Beckhoff Zertifikatsstelle. Teilen Sie in der Email mit, das es sich um eine Zertifikatsverlängerung, und keine Neuausstellung handelt. Für den restlichen Inhalt der Email gelten ansonsten dieselben Kriterien wie bei der Beantragung eines neuen Zertifikates.

Das bestehende Zertifikat erhält ein neues Ablaufdatum, wird dann neu signiert und ist weitere 2 Jahre gültig.

Das neu signierte Zertifikat ist damit vollständig kompatibel zur Ursprungsversion.

5.2.1.2.1 TwinCAT 3 Nutzerzertifikat beantragen

Übersicht über den Bestell- und Validierungsprozess

Die Beantragung eines TwinCAT Nutzerzertifikates erfordert eine offizielle Bestellung.

- Bestellnummer: **TC0008** (TwinCAT 3 Certificate Extended Validation)
- Die Ausstellung (und Verlängerung) eines TwinCAT 3 Nutzerzertifikates ist kostenlos.
- Da es sich bei einem TwinCAT 3 Nutzerzertifikat um einen digitalen Ausweis handelt, ist eine Verifizierung der Kontaktdaten des Anfragers nach den marktüblichen Standards erforderlich.
- Die Ausstellung eines TwinCAT 3 Nutzerzertifikates erfolgt daher nur für Beckhoff Bestandskunden.

Ihre Email-Adresse muss ein Firmen-Email-Account sein (Freemailer wie GMail oder Ähnliches sind nicht zulässig) und mit dem Firmennamen des Bestellers korrespondieren.

1. Kontaktieren Sie Ihren Beckhoff Vertriebskontakt und kündigen Sie die Beantragung eines TwinCAT 3 OEM-Zertifikats an. Bestellen Sie “TC0008”.
2. Wichtig: Geben Sie, als Anfrager, Ihre Kontaktdaten als Lieferadresse (= Kontaktname und Email-Adresse) und den Einsatzbereich des Zertifikats an (Firmenname, Adresse).
3. Die im Auftrag angegebenen Kontaktdaten werden verifiziert und Sie (der in der Lieferadresse genannte Anfrager) werden vom Beckhoff Vertrieb kontaktiert.
4. Bei der Beantragung eines neuen OEM-Zertifikats erstellen Sie im TwinCAT 3 Engineering eine „Certificate Request Datei“ [▶ 28].
5. Ermitteln Sie mit Hilfe des TwinCAT Engineerings den „File Fingerprint“ der OEM-Zertifikatsdatei (siehe File Fingerprint der Zertifikatsdatei ermitteln [▶ 31]). Teilen Sie diesen File Fingerprint im Rahmen Ihrer Kontaktdatenverifizierung dem Beckhoff Vertriebskontakt mit. Die Übermittlung des File Fingerprints muss auf einem anderen Kommunikationsweg erfolgen als für die Zusendung der OEM-Zertifikatsanfragedatei.
6. Schicken Sie die „OEM-Zertifikatsdatei“ nun an den Beckhoff Vertriebskontakt.
7. Nach der Signierung der Zertifikatsdatei in der Beckhoff Zentrale erhalten Sie diese über Ihren Ansprechpartner per Email zugesandt.

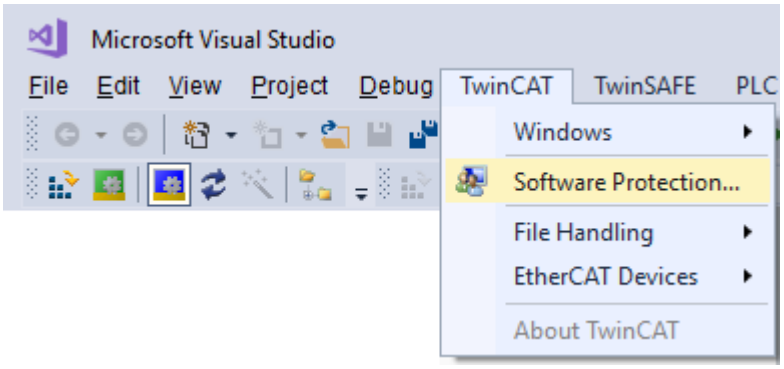
Beachten Sie, dass die Validierung der Kontaktdaten und die Ausstellung des Zertifikats einige Tage Zeit in Anspruch nehmen können.

5.2.1.2.2 Erstellung der Certificate Request Datei für TC0008

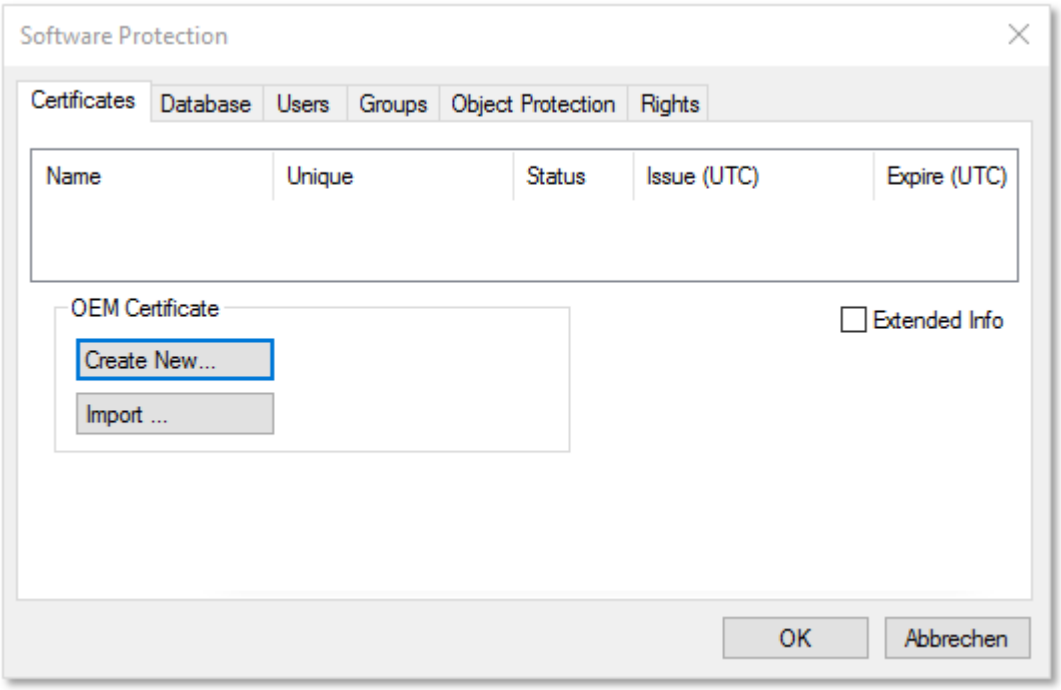
Systemvoraussetzungen

- Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 oder TwinCAT/BSD (auf dem Zielsystem)

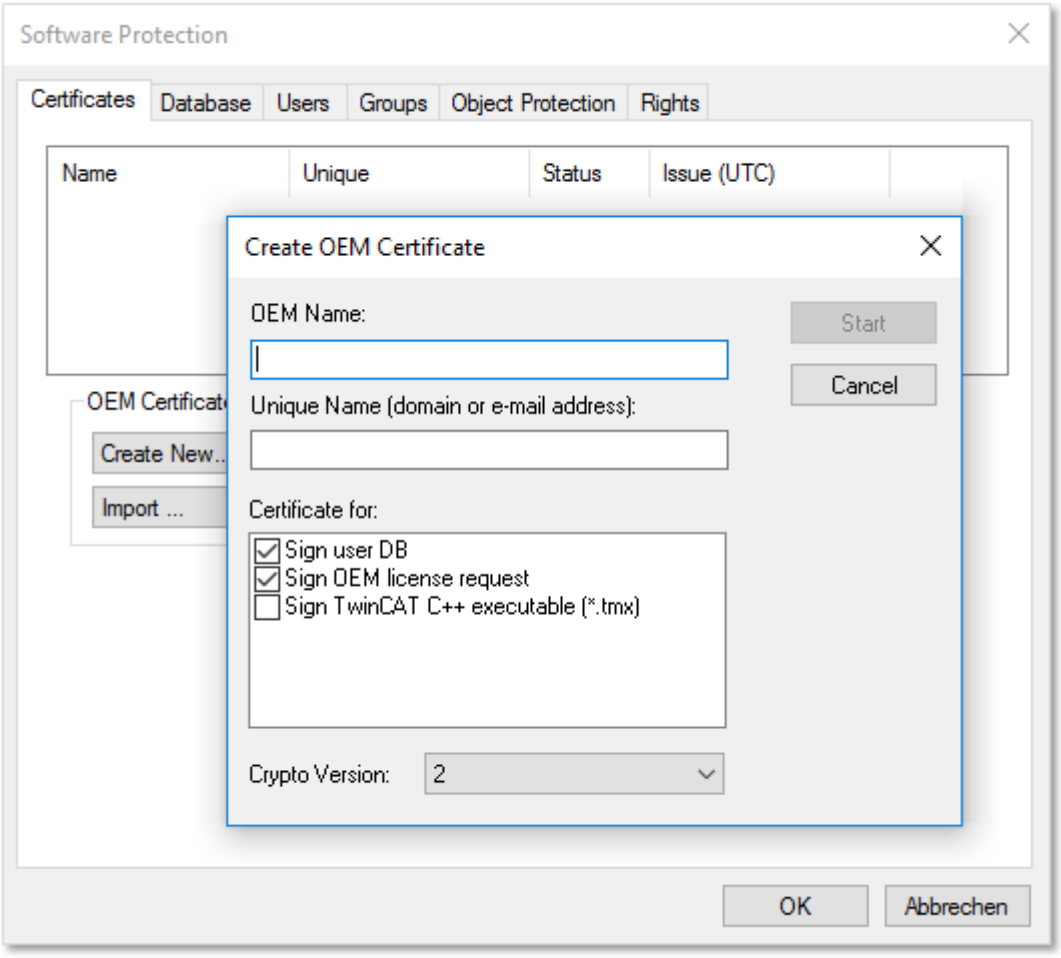
1. Rufen Sie den Konfigurator der Software Protection auf. Wählen Sie dazu im Hauptmenü unterhalb des Punktes **TwinCAT** den Menüpunkt **Software Protection** aus:



2. In dem sich öffnenden Fenster wählen Sie die Registerkarte **Certificates** aus.
   Klicken Sie auf **Create New…**:

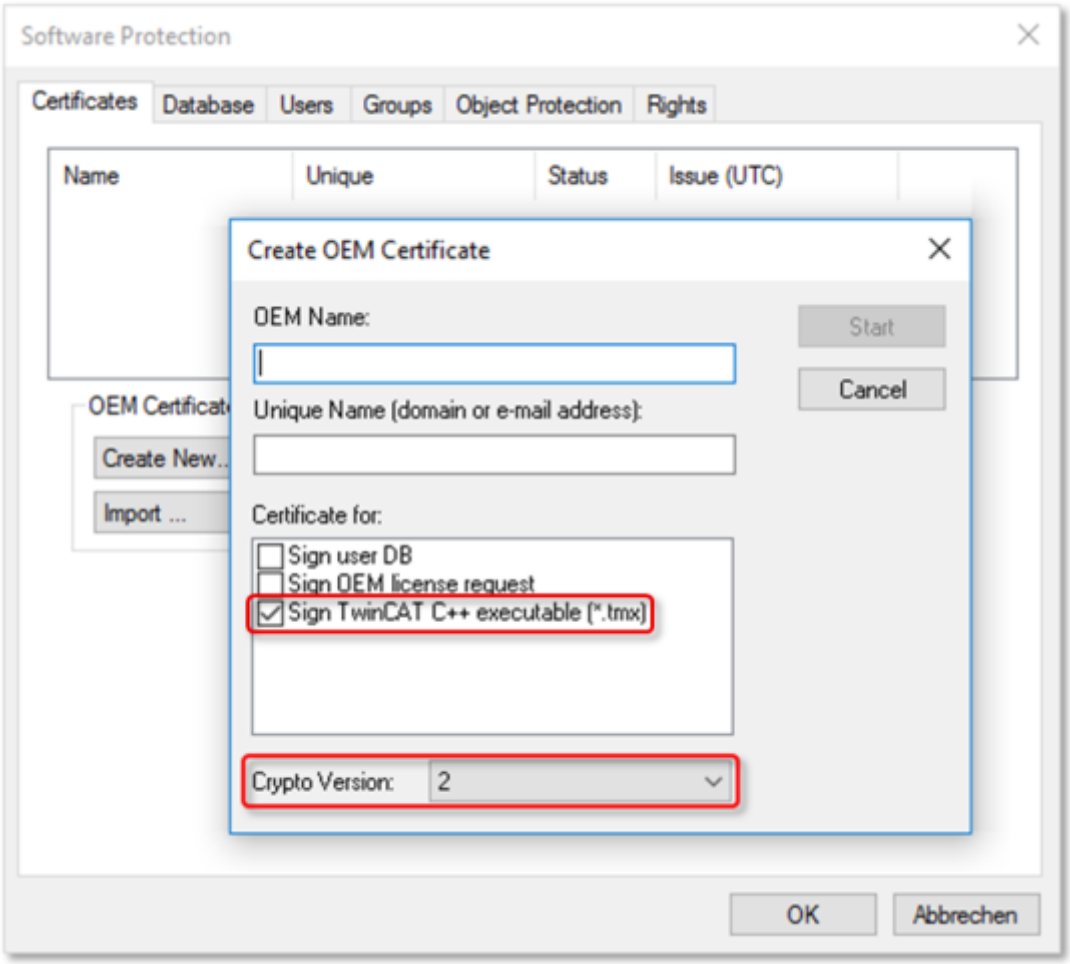


3. Das Eingabefenster **Create OEM Certificate** öffnet sich:



4. Geben Sie die erforderlichen Daten ein:
   - Geben Sie im Textfeld **OEM Name** Ihren Firmennamen ein. Der Name muss einen klaren Bezug zu Ihrer Firma oder Ihrem Unternehmensteil haben.

- Geben Sie einen **Unique Name** ein. Der „OEM Unique Name“ muss ein einmaliger Name sein, anhand dessen der Eigentümer des Zertifikats weltweit eindeutig identifiziert werden kann, vorzugsweise die URL der Webseite Ihrer Firma oder Ihre E-Mail-Adresse. Die E-Mail-Adresse muss eine Firmen-E-Mail-Adresse sein, also eindeutig Ihrer Firma zugeordnet werden können.
- Markieren Sie die Checkbox **Sign TwinCAT C++ executables**:



  Wenn Sie mit diesem Zertifikat nur TwinCAT Treiber Software signieren wollen, schalten Sie die anderen beiden Checkboxen aus. (Diese werden nur im SPS-Bereich genutzt.)

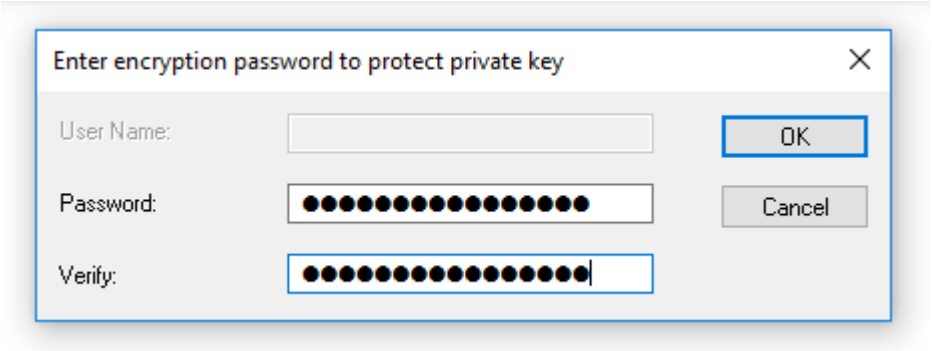
- Achten Sie darauf, dass die Crypto-Version 2 (für den verschlüsselten Inhalt des Zertifikatsinhaltes) eingestellt ist. (Standardeinstellung)

5. Wenn Sie die Daten eingegeben haben, klicken Sie auf **Start** und Sie können ein Verzeichnis auswählen, um die Datei zu speichern.
   Übernehmen Sie einfach das vorgeschlagene Verzeichnis „c:\twincat\3.1\customconfig\certificates“. Sie benötigen die neu erzeugte Datei in diesem Verzeichnis, um in einem späteren Schritt den „File Fingerprint“ für diese Datei auslesen [▶ 31] **zu können.**

   ⇨ Ein Dialog zur Auswahl eines Passworts für den OEM Private Key öffnet sich.

6. Vergeben Sie ein Passwort für den OEM Private Key.
   Passwort-Sicherheit - Verwenden Sie unbedingt ein starkes Passwort für Ihr Zertifikat! Schützen Sie Ihr Passwort durch geeignete Maßnahmen, damit es nicht in fremde Hände fallen kann! Passwort bei Verlust nicht wiederherstellbar - Beckhoff kann Ihr Passwort nicht wiederherstellen oder zurücksetzen. Wenn Sie das Passwort für Ihr Zertifikat vergessen oder verlieren, können Sie das Zertifikat nicht mehr verwenden und müssen ein neues Zertifikat ausstellen lassen.

7. Bestätigen Sie das Passwort durch eine wiederholte Eingabe und schließen Sie den Dialog mit **OK**.



⇨ Die Datei wird gespeichert.

Die so erzeugte „Certificate Request Datei“ muss nun noch von der Beckhoff Zertifikatsstelle signiert werden, um gültig zu sein. Das Verfahren dazu wird im Kapitel Zertifikat beantragen [▶ 27] beschrieben.

5.2.1.2.3 File Fingerprint der OEM-Zertifikatsdatei ermitteln

Diese Funktionalität benötigen Sie für die Beantragung eines **TwinCAT OEM Certificate Extended Validation** (TC0008).

Systemvoraussetzung

Diese Funktionalität erfordert mindestens die Version TwinCAT 3.1 Build 4024.

Die OEM Certificate Request Datei wird durch die Signierung von Beckhoff zum TwinCAT OEM-Zertifikat. Bis auf diese Signatur unterscheiden sich die Dateien nicht. Daher wird im Folgenden für beide Dateiversionen der Begriff „TwinCAT OEM-Zertifikatsdatei“ verwendet.

Den „File Fingerprint“ einer OEM-Zertifikatsdatei über das TwinCAT 3 Engineering auslesen

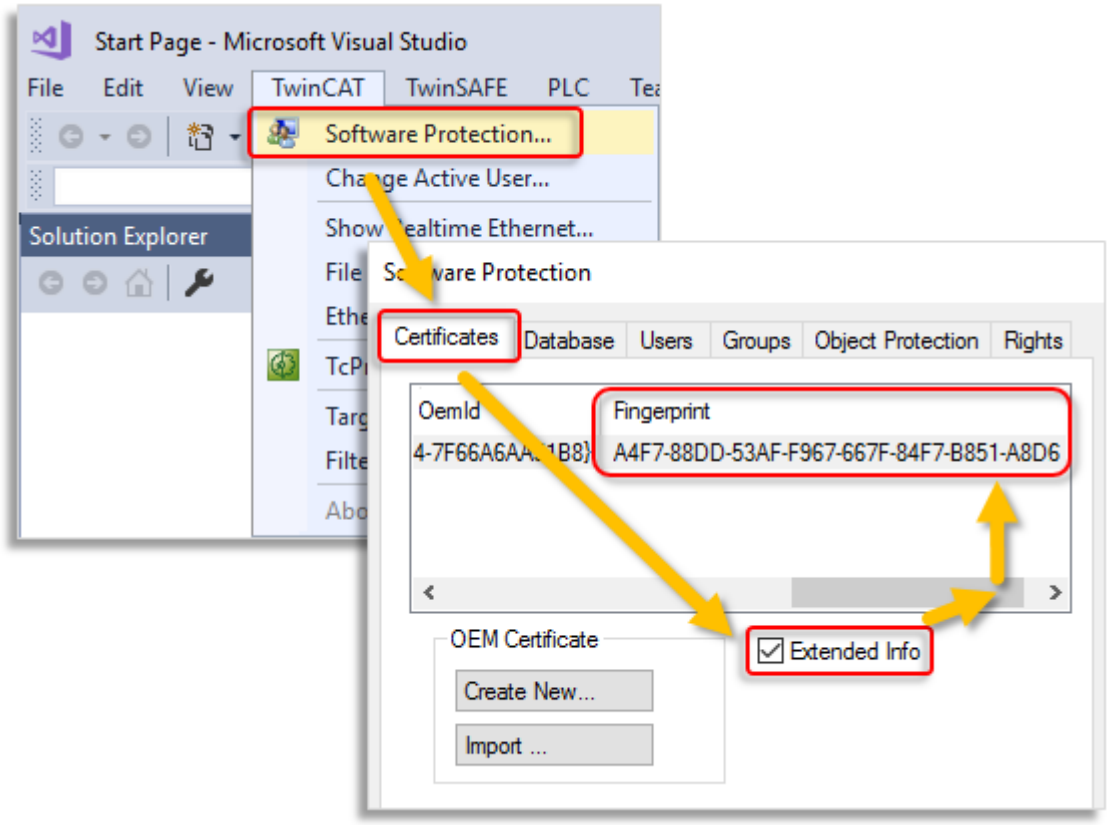
Für diese Funktion ist es erforderlich, dass die OEM-Zertifikatsdatei in diesem Verzeichnis liegt: „c:\twincat\3.1\customconfig\certificates“.

In diesem Verzeichnis liegt ihr OEM-Zertifikat, sofern Sie bereits eines haben und dieses verlängern wollen.

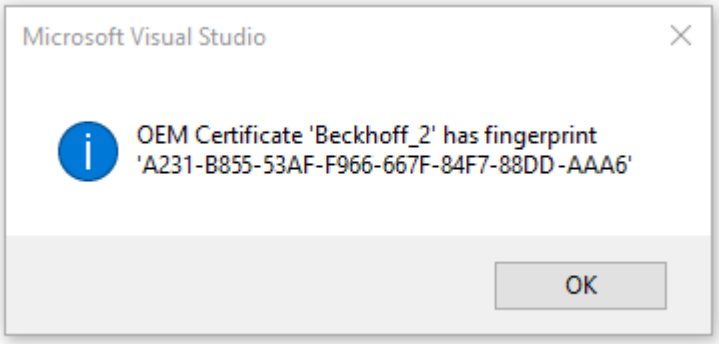
Sofern Sie beim Erstellen der „OEM Certificate Request Datei“ das vorgeschlagene Verzeichnis nicht geändert haben, liegt die Datei bereits in diesem Verzeichnis.

Vorgehensweise:

1. Rufen Sie den TwinCAT 3 Software Protection Konfigurator auf.



2. Wählen Sie den Tab **Certificates** aus.
3. Markieren Sie die Checkbox **Extended Info.**
4. Scrollen Sie im Fenster so weit nach rechts, bis Sie die Spalte **Fingerprint** sehen. (Alternativ können Sie auch einfach einen Doppelklick auf die Zertifikatszeile machen. In einem Popup-Fenster wird dann der File Fingerprint angezeigt):



Mit [Ctrl] + [C] können Sie die Fingerprint-Daten aus dem Meldungsfenster ins Windows Clipboard kopieren.

5.2.1.2.4 Speichern des fertig signierten TwinCAT Nutzerzertifikates

Empfohlenes Verzeichnis zum Speichern des Zertifikates: **C:\TwinCAT\3.1\CustomConfig\Certificates**

Systemvoraussetzungen

- Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 oder TwinCAT/BSD (auf dem Zielsystem)

Das TwinCAT 3 Nutzerzertifikat ist nicht für die Nutzung der TwinCAT 3 TMX-Dateien erforderlich

Das TwinCAT 3 Nutzerzertifikat wird ausschließlich für die einmalige Signierung der TMX-Dateien verwendet., und ist nicht für die Nutzung der damit signierten TMX-Dateien erforderlich.

Auf welchen Rechnern ist das TwinCAT 3 Nutzerzertifikat TC0008 erforderlich?

Das TwinCAT 3 Nutzerzertifikat sollte sich ausschließlich auf dem Engineering-Rechner befinden, auf dem die Signierung der TMX-Dateien erfolgt – also **NICHT** auf jedem Zielsystem.

5.2.2 Betriebssystem

HINWEIS

Migration zu TMX mit TwinCAT Loader empfohlen

Seit TwinCAT 3.1 4024.0 stehen versionierte C++ Projekte bereit, deren Binaries direkt von TwinCAT geladen werden können. Eine Migration wird empfohlen!

Für die Implementierung von TwinCAT 3 C++ Modulen auf x64-Plattformen muss der Treiber (*.sys Datei) mit einem Zertifikat signiert sein, wenn er durch das Betriebssystem geladen werden soll.

Die Signatur, die beim TwinCAT 3 Build-Prozess automatisch vorgenommen wird, wird von 64-Bit-Windows-Betriebssystemen für die Authentifizierung der Treiber verwendet.

Für die Signierung eines Treibers wird ein Zertifikat benötigt. Diese Microsoft Dokumentation beschreibt den Prozess und Hintergrundwissen zum Erhalt eines Test- und Freigabezertifikats, das von 64-Bit-Windows-Betriebssystemen akzeptiert wird.

Um ein solches Zertifikat in TwinCAT 3 zu verwenden, konfigurieren Sie den Schritt nach Kompilieren Ihres x64 Build-Targets wie in „Ein Testzertifikat für den Testmodus erstellen [▶ 33]“ dokumentiert.

Testzertifikate

Zum Testen können selbstsignierte Testzertifikate ohne technische Limitierung erstellt und verwendet werden.

In den folgenden Lernprogrammen ist beschrieben, wie diese Möglichkeit aktiviert wird.
Um Treiber mit echten Zertifikaten für Produktionsmaschinen zu erstellen, muss diese Möglichkeit deaktiviert werden.

- Ein Testzertifikat für den Testmodus erstellen [▶ 33]
- (Test-)Zertifikate löschen [▶ 36]

Weitere Referenzen:

MSDN, MakeCert Testzertifikate (Windows Treiber),
https://docs.microsoft.com/en-us/windows-hardware/drivers/install/makecert-test-certificate

5.2.2.1 Testsignierung

Übersicht

Für die Implementierung von TwinCAT 3 C++ Modulen auf x64-Plattformen muss der Treiber mit einem Zertifikat signiert sein.

Dieser Artikel beschreibt, wie Sie ein Testzertifikat zum Testen eines C++ Treibers erstellen und installieren können.

Vorgehen beim Erstellen von Testzertifikaten beachten

Entwickler verfügen möglicherweise über eine Vielzahl an Werkzeugen, um selbst Zertifikate zu erstellen - bitte folgen Sie dieser Beschreibung genauestens, um den Testzertifikat-Mechanismus anzuschalten.

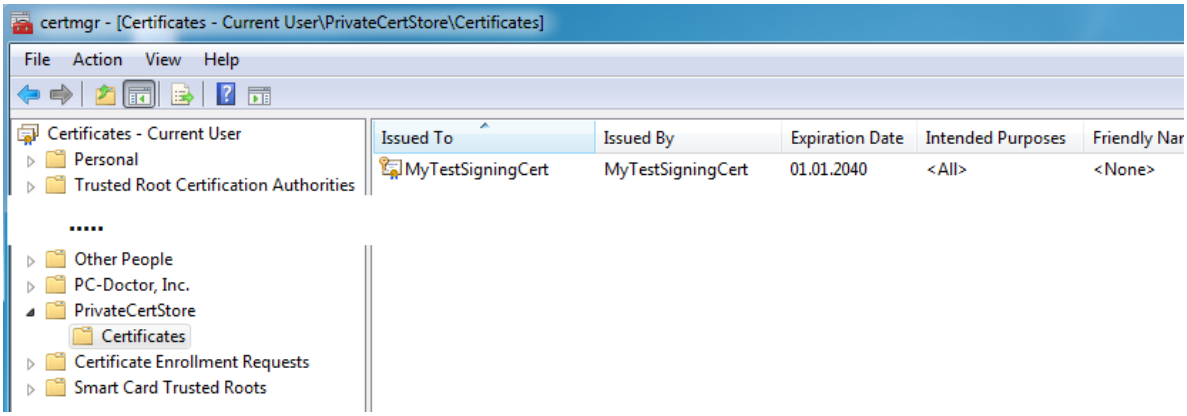
Die folgenden Befehle müssen in einer Kommandozeile ausgeführt werden, die auf eine der beiden Arten geöffnet wurde:

- **Eingabeaufforderung von Microsoft Visual Studio® 2010 / 2012 mit Administratorrechten**. (Über: **All Programs** -> **Microsoft Visual Studio 2010/2012** -> **Visual Studio Tools** -> Visual Studio Command Prompt, dann Rechtsklick auf **Run as administrator**)
- **Developer Command Prompt von Microsoft Visual Studio® 2017 / 2019 mit Administratorrechten**. (Über: **All Programs** -> **Visual Studio 2017** -> Visual Studio Command Prompt for VS 2017/2019, dann Rechtsklick auf **Run as administrator**)
- Nur, wenn das WINDDK installiert wurde:
  Normale Eingabeaufforderung (**Start** -> Command Prompt) mit Administratorrechten, dann in das Verzeichnis *%WINDDK7%\bin\x86* wechseln, das die entsprechenden Tools enthält.

1. Auf XAE:
   geben Sie im Engineering System folgenden Befehl in die Microsoft Visual Studio® 2010 / 2012 Eingabeaufforderung mit Administratorrechten ein (siehe Hinweis oben):
   ```
   makecert -r -pe -ss PrivateCertStore -n CN=MyTestSigningCert
   MyTestSigningCert.cer
   ```
   (Sollten Sie keine Zugriffsrechte auf den PrivateCertStore haben, können Sie eine andere Ablage nutzen. Diese muss dann im PostBuild-Event auch verwendet werden [▶ 37]**.)**

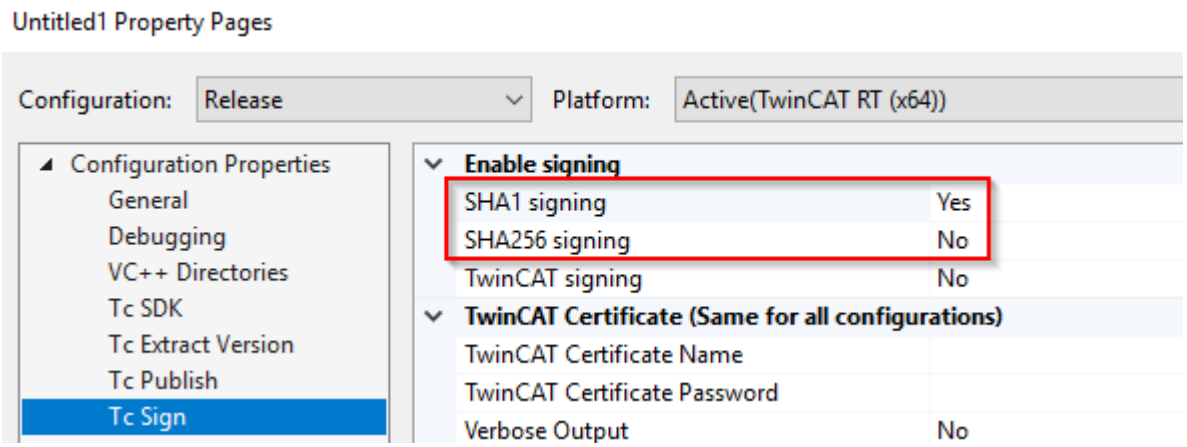
   ⇨ Daraufhin wird ein selbstsigniertes Zertifikat erstellt und in der Datei „MyTestSigningCert.cer“ und im Windows Certificate Store gespeichert.

   ⇨ Überprüfen Sie das Ergebnis mit mmc (**Use File->Add/Remove Snap-in->Certificates**):



2. Auf XAE:
   konfigurieren Sie das Zertifikat, sodass es von TwinCAT XAE auf dem Engineering System erkannt wird. Setzen Sie die Umgebungsvariable TWINCATTESTCERTIFICATE auf „MyTestSigningCert“ im Engineering System oder bearbeiten Sie jeweils den Post-Build Event von Debug|TwinCAT RT (x64) und Release|TwinCAT RT (x64).
   Der Name der Variablen ist NICHT der Name der Zertifikatsdatei, sondern der CN-Name (in diesem Falle MyTestSigningCert).

Ab TwinCAT 3.1 4024.0 wird die Konfiguration des zu verwendenden Zertifikates unter dem Punkt Tc Sign in den Projekt-Eigenschaften vorgenommen. Wenn Sie die Signierung über das Betriebssystem verwenden wollen, wie sie hier beschrieben ist, dann achten Sie bitte auf die Projekt-Einstellungen.

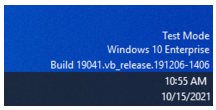


Auf XAR (und XAE, wenn es sich um einen lokalen Test handelt) aktivieren Sie den Testmodus, damit das Betriebssystem die selbstsignierten Zertifikate akzeptieren kann. Dies kann sowohl auf dem Engineering System (XAE) als auch auf dem Laufzeit-Systemen (XAR) vorgenommen werden.

Für Windows

Führen Sie mittels Administrator-Eingabeaufforderung folgendes aus:
`bcdedit /set testsigning yes`
und starten Sie das Zielsystem neu.
Es kann sein, dass Sie hierfür „SecureBoot“ abschalten müssen, was im Bios erfolgen kann.

Wenn der Testsigniermodus aktiviert ist, wird das rechts unten auf dem Desktop angezeigt. Das System akzeptiert nun alle signierten Treiber zur Ausführung.



Nach der jeweiligen Vorgehensweise akzeptiert das System alle signierten Treiber zur Ausführung.

1. Prüfen Sie, ob eine Konfiguration mit einem in einen TwinCAT C++ Treiber implementierten TwinCAT-Modul auf dem Zielsystem aktiviert und gestartet werden kann.

⇨ Die Kompilierung des x64-Treibers generiert eine Ausgabe wie folgt:

```
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 ==========
```

Referenzen:
MSDN, MakeCert Testzertifikate (Windows Treiber),
https://docs.microsoft.com/en-us/windows-hardware/drivers/install/makecert-test-certificate

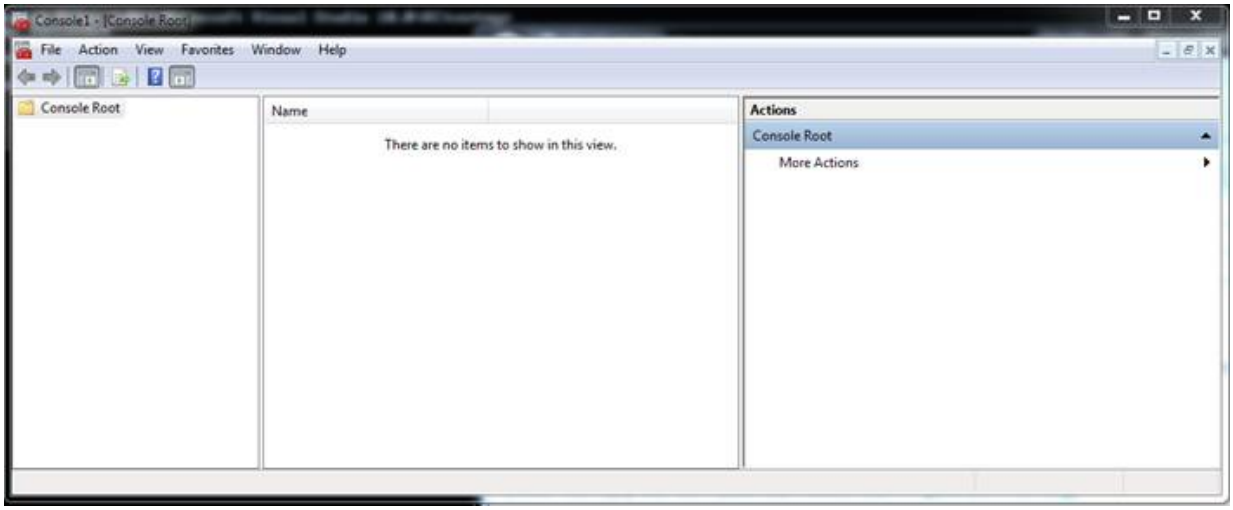
5.2.2.2 Testzertifikat löschen

Dieser Artikel beschreibt, wie ein Testzertifikat gelöscht wird.

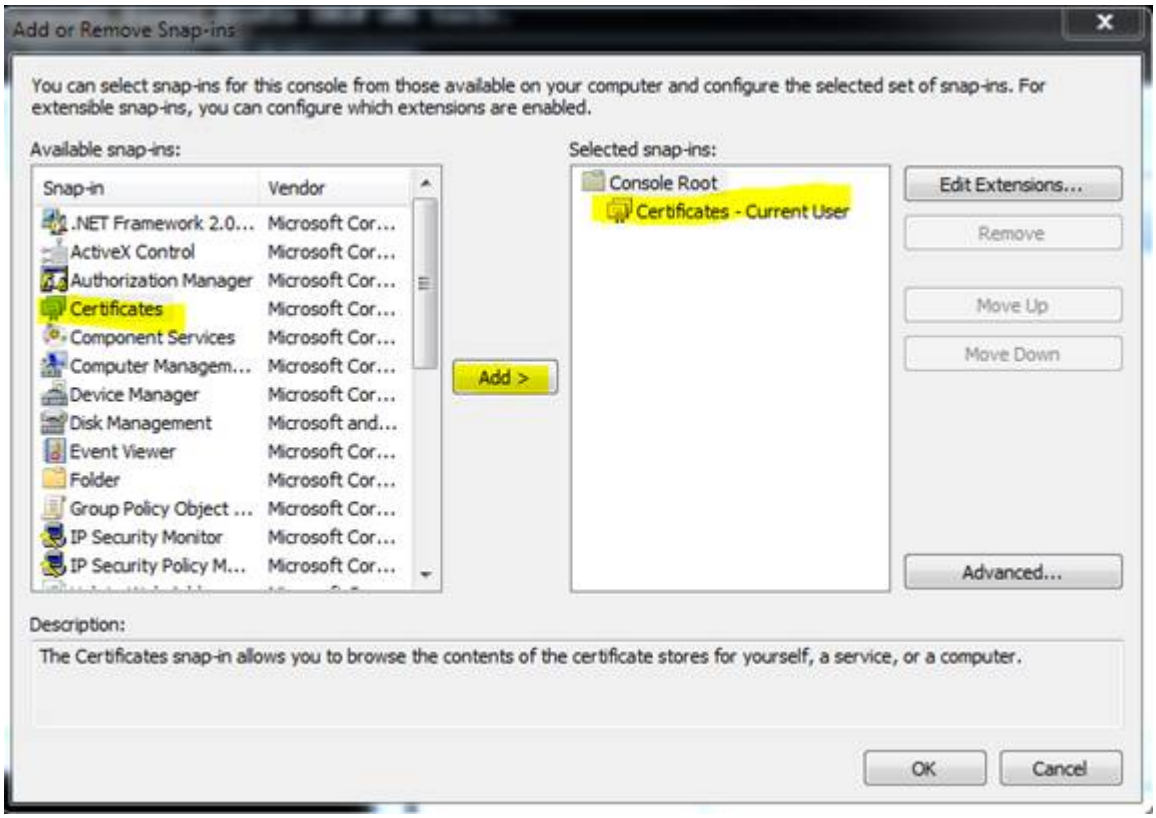
Übersicht

Ein Zertifikat kann mit der Microsoft Management Console gelöscht werden:

1. Starten Sie die Management console MMC.exe über Start-Menü oder die Oberfläche.

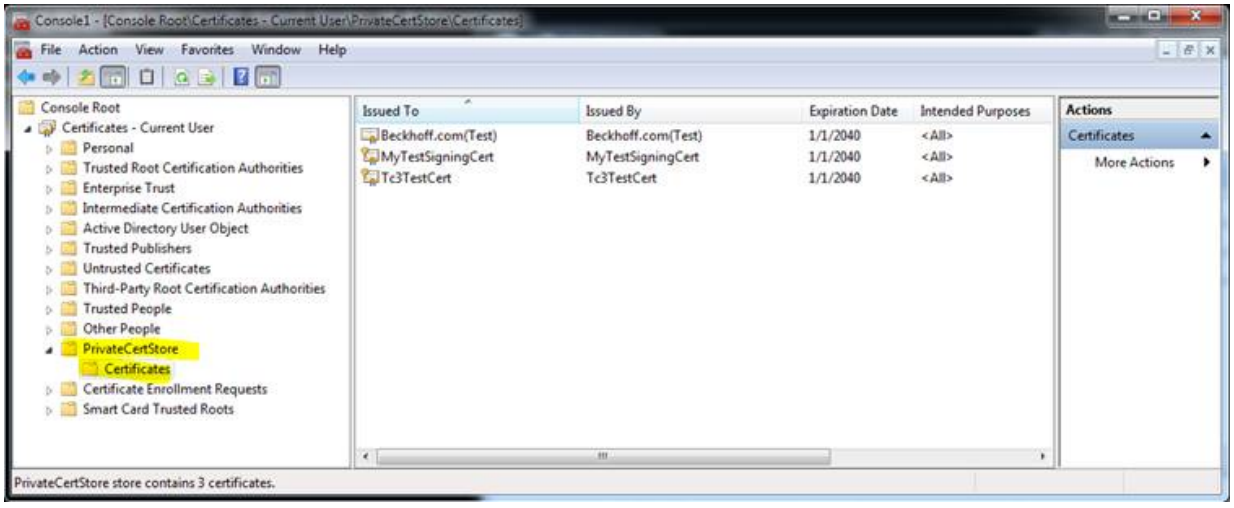


2. Klicken Sie im Menü auf **File** -> **Add/Remove Snap-in..** und wählen Sie Zertifikat-Snap-In für den aktuellen Nutzer, schließen Sie mit **OK** ab.



⇨ Die Zertifikate sind im Knoten unter **PrivateCertStore/Certificates** aufgelistet.

3. Wählen Sie das zu löschende Zertifikat aus.



5.2.2.3 Windows Treiber ohne Testmode

Für Windows Betriebssysteme ist eine Signierung des Treibers über das „Attestation Signing“ nötig. Hierfür wird ein sogenanntes EV-Zertifikat benötigt.
Microsoft stellt hierfür eine Anleitung bereit: https://docs.microsoft.com/en-us/windows-hardware/drivers/dashboard/attestation-signing-a-kernel-driver-for-public-release
Die hierdurch erzeugten Treiber sind auch geeignet für Geräte mit angeschaltetem Secure-Boot.

Das frühere Verfahren mittels CrossSigning-Zertifikaten (Signtool mit Parameter /ac) ist von Microsoft zum Juli 2021 abgekündigt. Es kann (in Abhängigkeit von dem Ablaufzeitpunkt des individuellen CrossSigning-Zertifikats) nicht mehr genutzt werden.

Für TwinCAT C++ existieren seit längerem die versionierten C++ Projekte, welche über den TwinCAT Loader [▶ 54] geladen werden, sodass sie keine Treiber im Sinne des Betriebssystems sind. Beckhoff empfiehlt deswegen auf die versionierten C++ Projekte zu setzen.

Bei der Migration von TwinCAT C++ Treibern zu versionierten C++ Projekten existiert eine Anleitung im How-to Bereich.

6 Module

Das TwinCAT-Modulkonzept ist eines der Kernelemente für die Modularisierung moderner Maschinen. Dieses Kapitel beschreibt das Modulkonzept und den Umgang mit Modulen.

Das Modulkonzept gilt für jedes TwinCAT-Modul, nicht nur für C++ Module. Die meisten Details betreffen aber nur das Engineering von C++ Modulen.

6.1 Das TwinCAT Component Object Model (TcCOM) Konzept

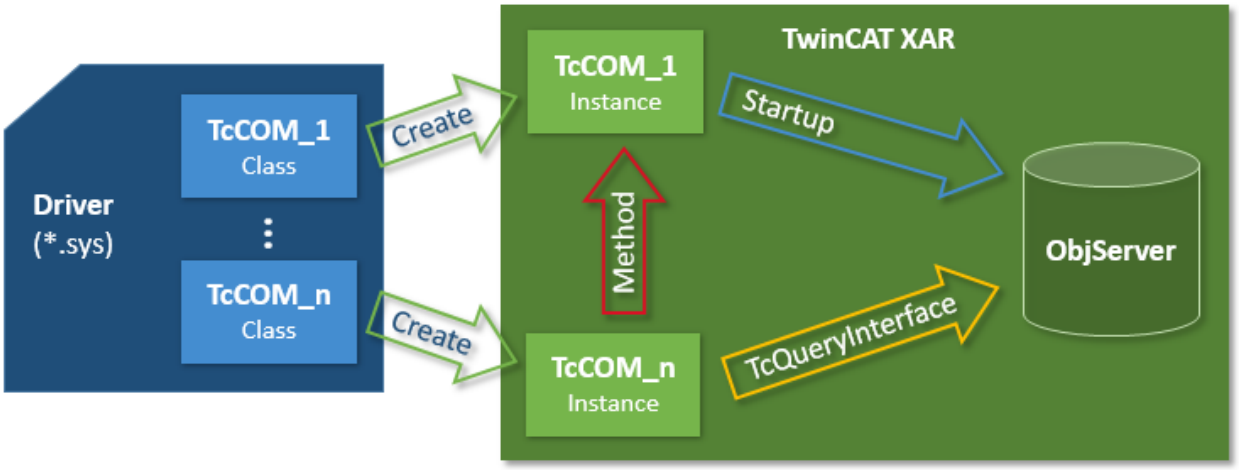
Das TwinCAT Component Object Model definiert die Eigenschaften und das Verhalten der Module. Das vom „Component Object Model“ (COM) von Microsoft Windows abgeleitete Modell beschreibt die Art und Weise, wie verschiedene Software-Komponenten, die unabhängig voneinander entwickelt und kompiliert wurden, zusammen arbeiten können. Damit das möglich ist, müssen ein genau definierter Verhaltensmodus und die Beachtung von Schnittstellen des Moduls definiert werden, so dass sie interagieren können. Eine solche Schnittstelle ist damit beispielsweise auch ideal um Module unterschiedlicher Hersteller in Interaktion zu setzen.

TcCOM macht Anleihen bei COM (Component Object Model aus der Microsoft Windows-Welt), wobei lediglich eine Untermenge von COM verwendet wird. Allerdings enthält TcCOM im Vergleich zu COM zusätzliche Definitionen, die über COM hinausgehen, z.B. die Modul Zustandsmaschine.

Überblick und Verwendung von TcCOM Modulen

Einleitend hier ein Überblick, der die einzelnen Themen leichter verständlich machen soll.

Ein oder mehrere TcCOM Module werden in einem Treiber zusammengefasst. Dieser Treiber wird vom TwinCAT Engineering mittels des MSVC Compilers erstellt. Die Module und Schnittstellen werden in einer TMC (TwinCAT Module Class) Datei beschrieben. Die Treiber mit ihrer TMC Datei können nun zwischen den Engineering Systemen ausgetauscht und kombiniert werden.



Mittels des Engineerings werden nun Instanzen von diesen Modulen erstellt. Zu diesen gibt es eine TMI Datei. Die Instanzen können parametriert und untereinander mit anderen Modulen oder zu dem IO verknüpft werden. Eine entsprechende Konfiguration wird auf das Zielsystem übertragen und dort ausgeführt.

Dabei werden entsprechende Module gestartet, die sich beim TwinCAT-ObjectServer anmelden. Das TwinCAT XAR sorgt auch für die Bereitstellung der Prozessabbilder. Module können den TwinCAT-ObjectServer nach einer Referenz auf ein anderes Objekt in Bezug auf eine bestimmte Schnittstelle fragen. Wenn sie eine solche Referenz erhalten, können sie die Methoden der Schnittstelle auf der Modulinstanz aufrufen.

Die folgenden Abschnitte konkretisieren die einzelnen Themengebiete.

ID Management

Es werden verschiedene Typen von IDs für die Interaktion der Module untereinander und auch innerhalb der Module verwendet. TcCOM verwendet GUIDs (128 Bit) sowie 32 Bit lange Ganzzahlen.

TcCOM verwendet

- GUIDs für: ModulIDs, ClassIDs und InterfaceIDs.
- 32 Bit lange Ganzzahlen werden verwendet für: ParameterIDs, ObjectIDs, ContextIDs, CategoryID.

Schnittstellen

Eine wichtige Komponente von COM, und damit auch von TcCOM, sind Schnittstellen.

Schnittstellen definieren einen Satz Methoden, die zusammengefasst werden, um eine bestimmte Aufgabe zu erfüllen. Eine Schnittstelle wird mit einer eindeutigen ID (InterfaceID) referenziert, die bei gleichbleibender Schnittstelle niemals geändert werden darf. Dank dieser ID können verschiedene Module feststellen, ob sie mit anderen Modulen zusammenarbeiten können. Gleichzeitig kann der Entwicklungsprozess unabhängig erfolgen, wenn die Schnittstellen fest definiert sind. Änderungen an Schnittstellen führen also zu unterschiedlichen IDs. Im TcCOM Konzept ist deswegen vorgesehen, dass InterfaceIDs andere (ältere) InterfaceIDs überlagern können („Hides“ in der TMC Beschreibung / im TMC Editor). Auf diese Weise stehen zum einen beide Varianten des Interfaces bereit, zum anderen wird aber auch immer klar, welches die aktuellste InterfaceID ist. Das gleiche Konzept gibt es auch für die Datentypen.

TcCOM selber definiert bereits eine ganze Reihe an Schnittstellen, die in manchen Fällen vorgeschrieben (z.B. ITComObject), aber in den meisten Fällen optional sind. Viele Schnittstellen machen nur in bestimmten Anwendungsbereichen Sinn. Andere Schnittstellen sind dermaßen allgemein, dass sie häufig wiederverwendet werden können. Kunden-definierte Schnittstellen sind vorgesehen, so dass z.B. zwei Module eines Herstellers in der Lage sind, miteinander in Verbindung zu treten.

- Alle Schnittstellen werden von der grundlegenden Schnittstelle ITcUnknown abgeleitet, die, wie die entsprechende Schnittstelle von COM, die grundlegenden Dienste zur Abfrage von anderen Schnittstellen des Moduls (TcQueryInterface) und zur Steuerung der Lebensdauer des Moduls (TcAddRef und TcRelease) bereitstellt.
- Die ITComObject-Schnittstelle, die von jedem Modul implementiert sein muss, enthält Methoden um auf den Namen, die ObjectID, die ObjectID des Parent, die Parameter und die Zustandsmaschine des Moduls zuzugreifen.

Einige allgemeine Schnittstellen werden von vielen Modulen verwendet:

- ITcCyclic wird von Modulen, die zyklische aufgerufen werden sollen („CycleUpdate“), implementiert. Mit Hilfe des ITcCyclicCaller Interfaces einer TwinCAT-Task kann sich das Modul anmelden um zyklische Aufrufe zu erhalten.
- Durch die ITcADI-Schnittstelle kann auf Datenbereiche eines Moduls zugegriffen werden.
- ITcWatchSource wird standardmäßig implementiert und erlaubt unter anderem ADS-DeviceNotifcations zu erhalten.
- Die ITcTask-Schnittstelle, die von den Tasks des Echtzeitsystems implementiert wird, stellen Informationen bezüglich der Zykluszeit, der Priorität und anderer Informationen zur Task zur Verfügung.
- Die Schnittstelle ITComObjectServer wird vom ObjectServer implementiert und von allen Modulen referenziert.

Es wurden bereits eine ganze Reihe allgemeiner Schnittstellen definiert. Allgemeine Schnittstellen haben den Vorteil, dass deren Verwendung den Austausch und die Wiederverwertung von Modulen unterstützt. Nur dann, wenn keine geeigneten allgemeinen Schnittstellen bestehen, sollten eigene Schnittstellen definiert werden.

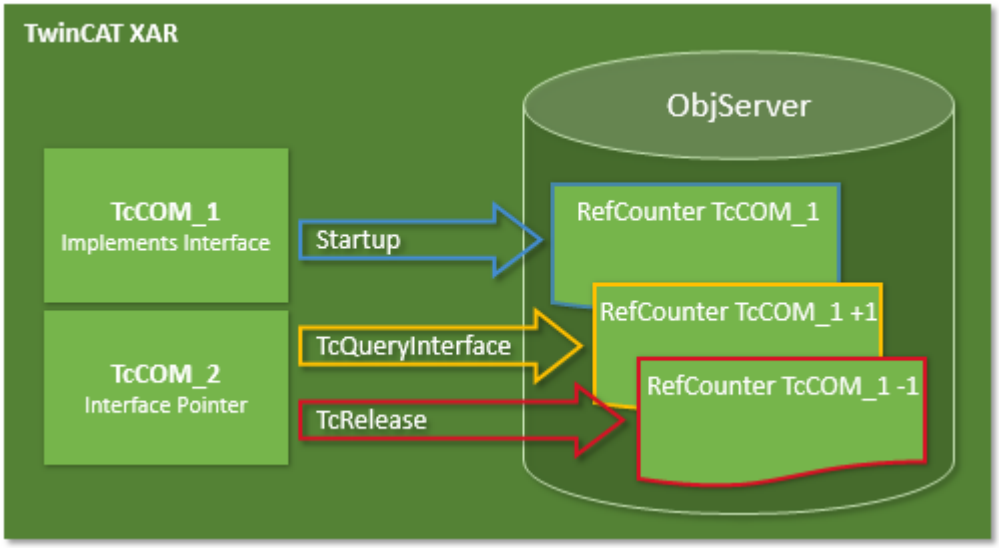
Class Factories

Für die Erzeugung von in C++ Modulen werden sogenannte „Class Factories“ verwendet. Alle Module, die in einem Treiber sind, besitzen eine gemeinsame Class Factory. Die Class Factory meldet sich einmal beim ObjectServer an und bietet ihre Dienste für die Erstellung bestimmter Modulklassen an. Die Modulklassen sind durch die eindeutige ClassID des Moduls gekennzeichnet. Wenn der ObjectServer ein neues Modul

anfordert (auf Grundlage der Initialisierungsdaten des Konfigurators oder durch andere Module während der Laufzeit), wählt das Modul die richtige Class Factory auf Grundlage der ClassID aus und veranlasst die Erzeugung des Moduls über seine ITcClassFactory-Schnittstelle.

Modul-Lebensdauer

Auf ähnliche Weise wie im Falle von COM wird die Lebensdauer eines Moduls über einen Verweiszähler (RefCounter) bestimmt. Jedes Mal, wenn eine Schnittstelle eines Moduls abgefragt wird, wird der Verweiszähler inkrementiert. Er wird wieder dekrementiert, wenn die Schnittstelle freigegeben wird. Eine Schnittstelle wird auch bei der Anmeldung eines Moduls beim ObjectServer abgefragt, (die ITComObject-Schnittstelle), so dass der Verweiszähler zumindest auf eins steht. Bei der Abmeldung wird er erneut dekrementiert. Wenn der Zähler den Wert 0 erreicht, löscht das Modul sich selbst automatisch - normalerweise nach der Abmeldung beim ObjectServer. Wenn aber bereits ein anderes Modul einen Verweis hält (einen Schnittstellenzeiger besitzt), dann besteht das Modul weiter - und der Schnittstellenzeiger bleibt so lange gültig, bis dieser Zeiger auch freigegeben wird.



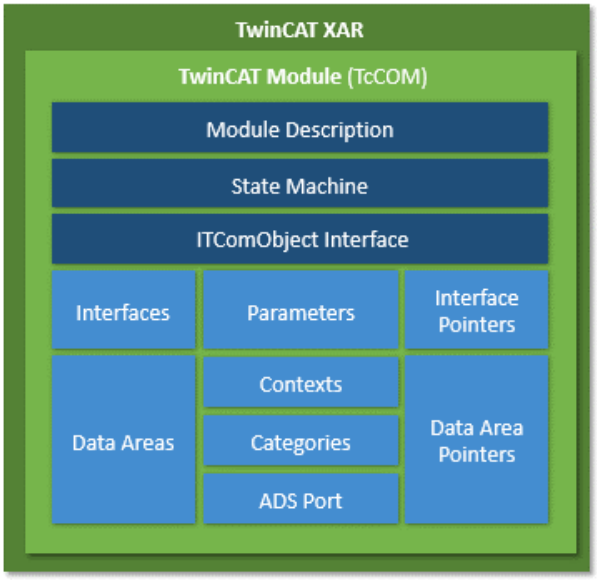
6.1.1 TwinCAT-Modul Eigenschaften

Ein TcCOM-Modul hat eine Reihe formal definierter, vorgeschriebener und optionaler Eigenschaften. Die Eigenschaften sind so weit formalisiert, dass eine Verwendung untereinander möglich ist. Jedes Modul hat eine Modulbeschreibung, die die Eigenschaften des Moduls beschreibt. Diese werden für eine Konfiguration der Module und deren Beziehungen untereinander verwendet.

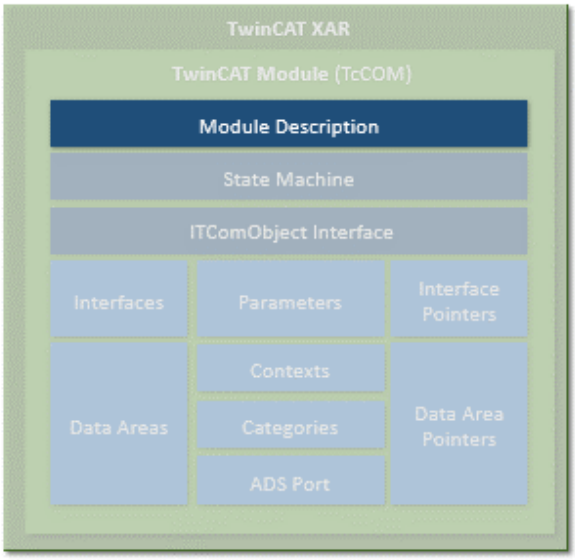
Wenn ein Modul in der TwinCAT Runtime instanziiert wird, dann meldet es sich selber bei einer zentralen Systeminstanz, dem ObjectServer, an. Dadurch wird es für andere Module und auch für allgemeine Tools erreichbar und parametrierbar. Module können unabhängig voneinander kompiliert und demzufolge auch entwickelt, getestet und aktualisiert werden. Module können sehr einfach konzipiert sein, z.B. nur eine kleine Funktion wie einen Tiefpassfilter enthalten. Sie können aber auch intern sehr komplex sein und z.B. das gesamte Steuerungssystem einer Maschinenunterbaugruppe beinhalten.

Es gibt sehr viele Anwendungen für Module; alle Aufgaben eines Automatisierungssystems können in Module spezifiziert werden. Demzufolge wird nicht unterschieden, ob das Modul primär die Grundfunktionen eines Automatisierungssystems darstellt, wie Echtzeit-Tasks, Feldbus-Treiber oder ein SPS-Laufzeitsystem, oder eher benutzer- und anwendungsspezifische Algorithmen für die Steuerung oder Regelung einer Maschineneinheit.

Die Abbildung unten zeigt ein gewöhnliches TwinCAT-Modul mit seinen wichtigsten Eigenschaften. Die dunkelblauen Blöcke definieren vorgeschriebene, die hellblauen Blöcke optionale Eigenschaften.



Modulbeschreibung



Jedes TcCOM Modul hat einige allgemeine Beschreibungsparameter. Dazu gehört eine ClassID, die die Klasse des Moduls eindeutig referenziert. Sie wird durch die passende ClassFactory instanziiert. Jede Instanz eines Moduls hat eine ObjectID, die in der TwinCAT Runtime eindeutig ist. Darüber hinaus gibt es eine Parent-ObjectID, die auf einen möglichen logischen Parent verweist.

Modulbeschreibung, Zustandsmaschine und Parameter des unten beschriebenen Moduls können über die ITComObject-Schnittstelle erreicht werden (Siehe „Schnittstellen“).

Klassenbeschreibungsdateien (*.tmc)

Die Klassen der Module werden in den Klassenbeschreibungsdateien (TwinCAT Module Class; *.tmc) beschrieben.

In dieser Datei beschreibt der Entwickler Eigenschaften und Schnittstellen des Moduls, so dass andere es nutzen und einbetten können. Neben den allgemeinen Informationen (Herstellerangaben, Klassen-ID des Moduls, usw.) werden optionale Eigenschaften des Moduls beschrieben.

- unterstützte Kategorien
- implementierte Schnittstellen
- Datenbereiche mit entsprechenden Symbolen
- Parameter

- Schnittstellenzeiger
- Datenzeiger, die gesetzt werden können

Die Klassenbeschreibungsdateien werden vom Konfigurator des Systems in erster Linie als Grundlage für die Einbindung einer Instanz des Moduls in die Konfiguration, zum Festlegen der Parameter und zwecks Konfiguration der Verbindungen mit anderen Modulen verwendet.

Sie enthalten zudem die Beschreibung aller Datentypen in den Modulen, die dann vom Konfigurator in sein allgemeines Datentypsystem aufgenommen werden. Damit werden also alle Schnittstellen der im System vorhandenen TMC Beschreibungen für alle Module nutzbar.

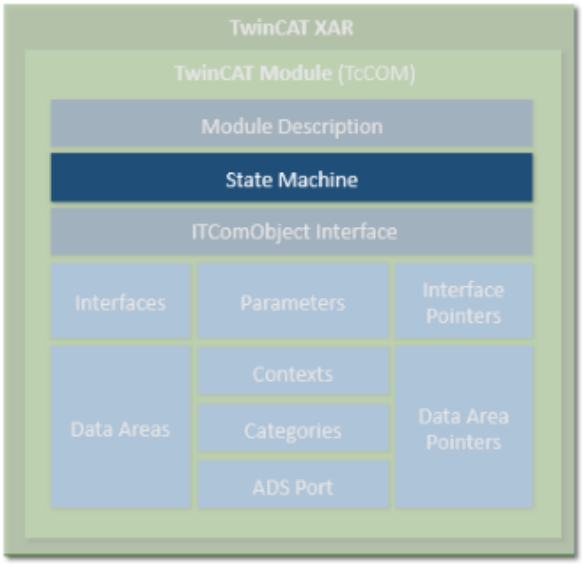
Auch komplexere Konfigurationen mehrerer Module können in den Klassenbeschreibungsdateien beschrieben werden, die für eine spezifische Anwendung vorkonfiguriert und verbunden sind. Demzufolge kann ein Modul für eine komplexe Maschineneinheit, die intern aus einer Reihe von Untermodulen besteht, bereits im Verlauf der Entwicklungsphase als ein Ganzes definiert und vorkonfiguriert werden.

Instanzenbeschreibungsdateien (*.tmi)

Ein Instanz eines bestimmten Moduls wird in der Instanzenbeschreibungsdatei (TwinCAT Module Instance; *.tmi) beschrieben. Die Instanzbeschreibungen sind durch ein ähnliches Format beschrieben, enthalten entgegen den Klassenbeschreibungsdateien aber bereits konkrete Festlegungen der Parameter, Schnittstellenzeiger usw. für die besondere Instanz des Moduls innerhalb eines Projekts.

Die Instanzenbeschreibungsdateien werden vom TwinCAT Engineering (XAE) erstellt, wenn eine Instanz einer Klassenbeschreibung für ein konkretes Projekt erstellt wird. Sie dienen hauptsächlich dem Datenaustausch zwischen allen an der Konfiguration beteiligten Tools. Allerdings können die Instanzenbeschreibungen auch projektübergreifend genutzt werden, wenn z.B. ein besonders parametriertes Modul in einem neuen Projekt erneut verwendet werden soll.

Zustandsmaschine

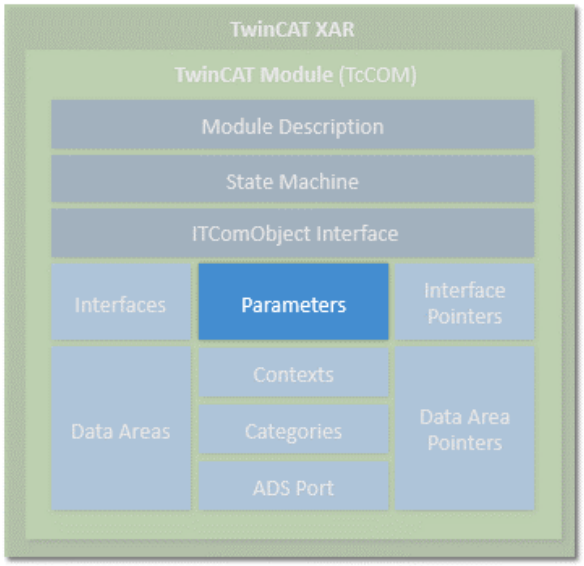


Jedes Modul enthält eine Zustandsmaschine, die den Initialisierungszustand des Moduls und die Mittel, mit denen dieser Zustand von außen verändert werden kann, beschreibt. Diese Zustandsmaschine beschreibt die Zustände, die beim Starten und Beenden des Moduls durchlaufen werden. Dieses betrifft die Modulerzeugung, -parametrierung und Herstellung der Verbindung mit den anderen Modulen.

Anwendungsspezifische Zustände (z.B. von Feldbus oder Treiber) können in ihren eigenen Zustandsmaschinen beschrieben werden. Die Zustandsmaschine der TcCOM-Module definiert die Zustände INIT, PREOP, SAFEOP und OP. Auch wenn es dieselben Zustandsbezeichnungen sind wie unter EtherCAT-Feldbus sind es doch nicht die gleichen Zustände. Wenn das TcCOM-Modul einen Feldbustreiber für EtherCAT implementiert, hat es zwei Zustandsmaschinen (Modul- und Feldbuszustandsmaschine), die nacheinander durchlaufen werden. Die Modulzustandsmaschine muss den Betriebszustand (OP) erreicht haben, bevor die Feldbuszustandsmaschine überhaupt starten kann.

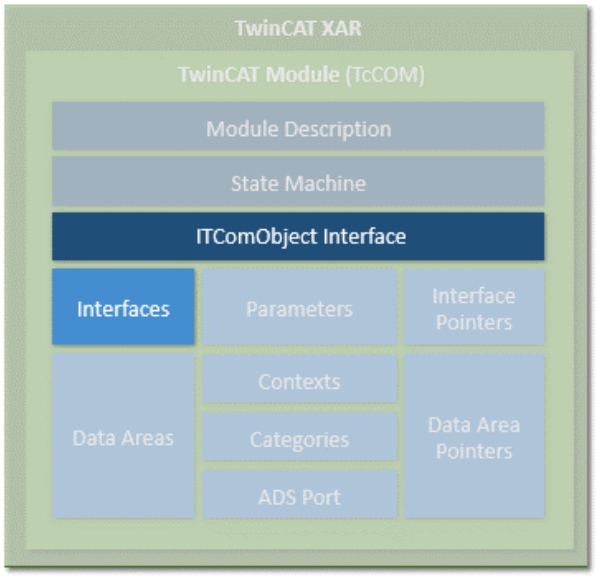
Die Zustandsmaschine ist im Detail separat beschrieben [▶ 48].

Parameter



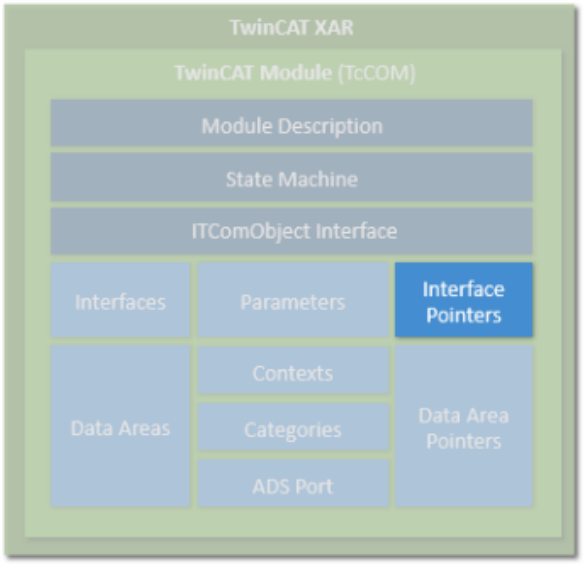
Module können Parameter haben, die während der Initialisierung oder später während der Laufzeit (OP-Zustand) gelesen oder geschrieben werden können. Jeder Parameter ist durch eine Parameter-ID gekennzeichnet. Die Eindeutigkeit der Parameter-ID kann global, eingeschränkt global oder modulspezifisch sein. Mehr hierzu im Abschnitt „ID Management“. Neben der Parameter-ID enthält der Parameter die aktuellen Daten; der Datentyp hängt vom Parameter ab und ist für die jeweilige Parameter-ID eindeutig definiert.

Schnittstellen



Schnittstellen bestehen aus einem definierten Satz an Methoden (Funktionen), die Module anbieten und über die sie z.B. von anderen Modulen kontaktiert werden können. Schnittstellen sind durch eine eindeutige ID charakterisiert, wie oben beschrieben. Ein Modul muss mindestens die ITComObject-Schnittstelle unterstützen, kann aber daneben so viele Schnittstellen wie gewünscht beinhalten. Eine Schnittstellen-Referenz kann durch Aufruf der Methode „TcQueryInterface“ mit Angabe der entsprechenden Schnittstellen-ID abgefragt werden.

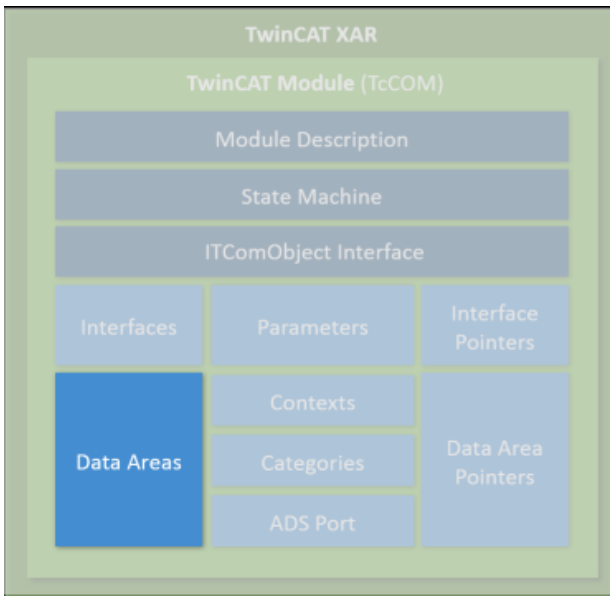
Schnittstellenzeiger



Schnittstellenzeiger verhalten sich wie das Gegenstück von Schnittstellen. Wenn ein Modul eine Schnittstelle eines anderen Moduls nutzen möchte, muss es einen Schnittstellenzeiger des entsprechenden Schnittstellentyps haben und dafür sorgen, dass dieser auf das andere Modul zeigt. Danach können die Methoden des anderen Moduls verwendet werden.

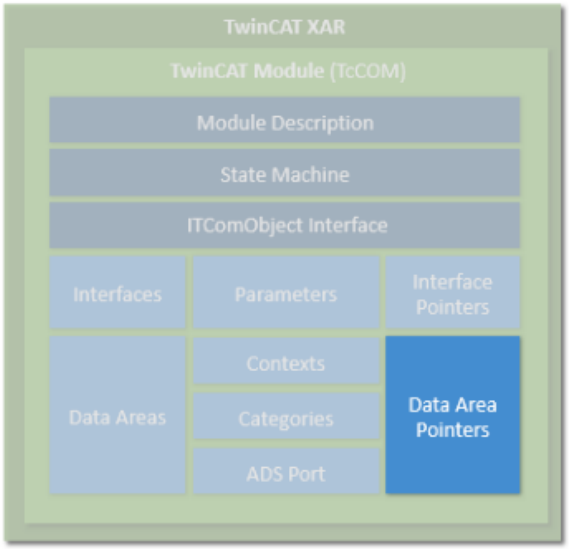
Schnittstellenzeiger werden normalerweise beim Start der Zustandsmaschine gesetzt. Beim Übergang von INIT zu PREOP (IP) empfängt das Modul die Objekt-ID des anderen Moduls mit der entsprechenden Schnittstelle, beim Übergang PREOP zu SAFEOP (PS) oder SAFEOP zu OP (SO) wird die Instanz des anderen Moduls mit dem ObjectServer durchsucht und die entsprechende Schnittstelle mit der Methode Query Interface gesetzt. Beim Zustandsübergang in die entgegengesetzte Richtung, von SAFEOP zu PREOP (SP) oder OP zu SAFEOP (OS) muss die Schnittstelle wieder freigegeben werden.

Datenbereiche



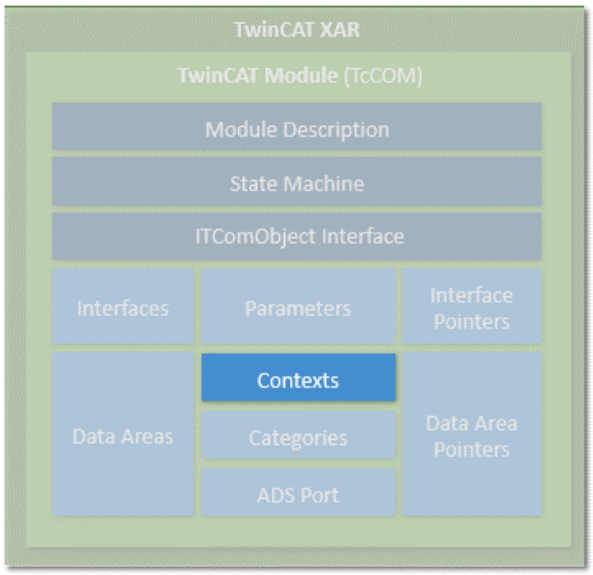
Module können Datenbereiche enthalten, die von der Umgebung verwendet werden können (z.B. von anderen Modulen oder zum IO Bereich von TwinCAT). Diese Datenbereiche können beliebige Daten enthalten. Sie werden oft für Prozessabbilddaten (Ein- und Ausgänge) genutzt. Die Struktur der Datenbereiche wird in der Gerätebeschreibung des Moduls definiert. Wenn ein Modul Datenbereiche hat, die es für andere zugänglich machen möchte, implementiert es die ITcADI-Schnittstelle, die den Zugriff auf die Daten ermöglicht. Datenbereiche können Symbolinformationen enthalten, die die Struktur des jeweiligen Datenbereichs im Einzelnen beschreiben.

Datenbereichszeiger



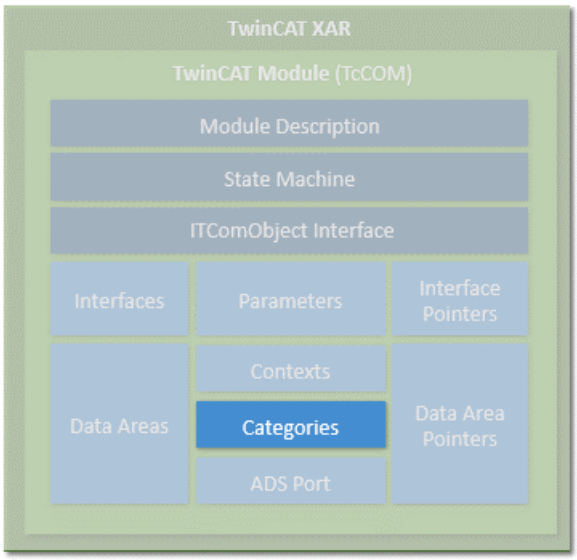
Wenn ein Modul auf den Datenbereich anderer Module zugreifen möchte, kann es Datenbereichszeiger enthalten. Diese werden normalerweise während der Initialisierung der Zustandsmaschine auf Datenbereiche oder Datenbereichsabschnitte anderer Module gesetzt. Es handelt sich dabei um einen direkten Zugriff auf den Speicherbereich, so dass bei Bedarf entsprechende Schutzmechanismen für konkurrierende Zugriffe eingesetzt werden müssen. Häufig bietet sich deshalb besser an, eine entsprechende Schnittstelle zu nutzen.

Kontext



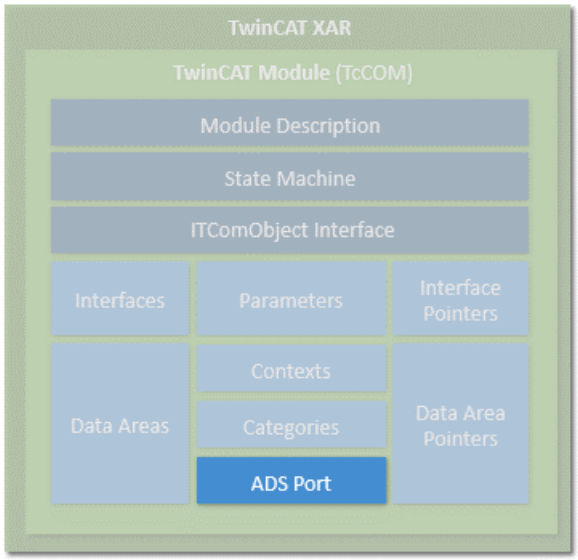
In diesem Zusammenhang ist Kontext als Echtzeit-Task Kontext zu verstehen. Kontexte werden u.a. für die Konfiguration der Module benötigt. Einfache Module arbeiten normalerweise in einem einzigen Zeitkontext, demzufolge muss er nicht näher spezifiziert werden. Andere Module können teilweise in mehreren Kontexten aktiv sein (z.B. ein EtherCAT Master kann mehrere unabhängige Echtzeit Tasks unterstützten, oder eine Regelschleife kann Regelschleifen der darunterliegenden Ebene in anderer Zykluszeit abarbeiten). Wenn ein Modul mehr als einen zeitabhängigen Kontext hat, muss das in der Modulbeschreibung angegeben werden.

Kategorien



Module können Kategorien anbieten, indem sie das Interface ITComObjectCategory implementieren. Kategorien werden vom ObjectServer enumeriert und Objekte, die sich hierrüber Kategorien zuordnen, können durch den ObjectServer (ITComObjectEnumPtr) abgefragt werden.

ADS



Jedes Modul, das beim ObjectServer eingetragen ist, kann per ADS erreicht werden. Der ObjectServer nutzt dabei die ITComObject Schnittstelle der Module, um z.B. Parameter zu lesen oder zu schreiben oder auch um auf die Zustandsmaschine zuzugreifen. Es gibt zusätzlich die Möglichkeit der Implementierung eines eigenes ADS Ports, über den eigene ADS Kommandos empfangen werden können.

Systemmodul

Die TwinCAT Laufzeit stellt darüber hinaus eine Reihe sogenannter Systemmodule zur Verfügung, die die grundlegenden Dienste der Laufzeit für andere Module zur Verfügung stellen. Diese Systemmodule haben eine feste, konstante ObjectID, über welche die anderen Module auf sie zugreifen können. Ein Beispiel eines solchen Systemmoduls ist das Echtzeitsystem, das die grundlegenden Dienste des Echtzeitsystems, d.h. die Generierung von Echtzeit-Tasks, via ITcRTime-Schnittstelle zur Verfügung stellt. Auch der ADS Router ist als Systemmodul implementiert, so dass andere Module an dieser Stelle ihren ADS Port anmelden können.

Erstellung von Modulen

Module können sowohl in C++ als auch in IEC 61131-3 erstellt werden. Die objektorientierten Erweiterungen der TwinCAT PLC werden hierzu verwendet. Module aus den beiden Welten können über Schnittstellen auf die gleiche Weise wie reine C++ Module untereinander interagieren. Mit Hilfe der objektorientierten Erweiterung werden die gleichen Schnittstellen bereitgestellt wie in C++.

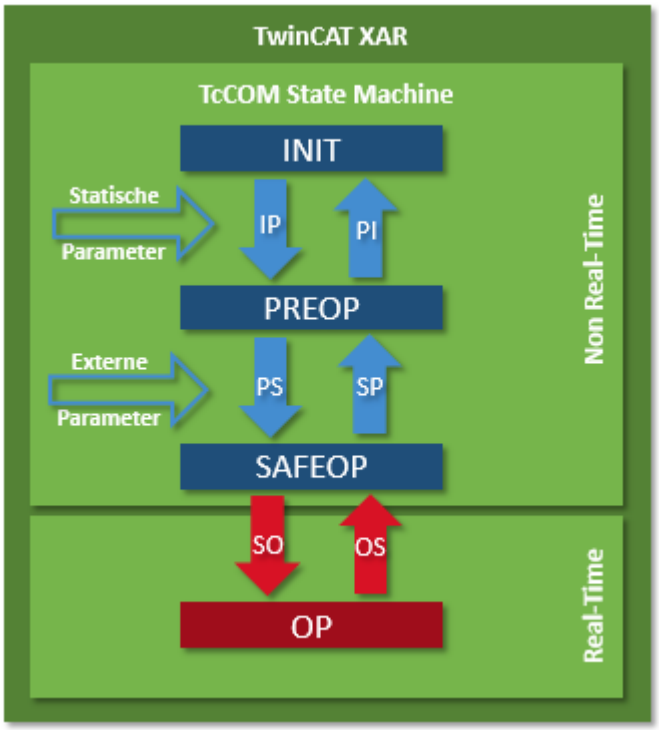
Die SPS-Module melden sich ebenfalls selbst beim ObjectServer an und sind demzufolge über ihn erreichbar. Die Komplexität von SPS-Modulen ist unterschiedlich. Es macht keinen Unterschied, ob nur ein kleines Filtermodul generiert oder ein komplettes SPS-Programm in ein Modul hineingepackt wird. Jedes SPS-Programm ist aufgrund der Automation ein Modul im Sinne der TwinCAT-Module. Jedes klassische SPS-Programm wird automatisch in ein Modul gepackt und meldet sich selbst beim ObjectServer und bei einem oder mehreren Task-Modulen an. Der Zugriff auf die Prozessdaten eines SPS-Moduls (z.B. Mapping in Bezug auf einen Feldbus-Treiber) wird ebenfalls über die definierten Datenbereiche und ITcADI gesteuert.

Dieses Verhalten bleibt transparent und unsichtbar für den SPS-Programmierer, so lange bis er beschließt ausdrücklich Teile des SPS-Programms als TwinCAT-Module zu definieren, damit er diese mit geeigneter Flexibilität nutzen kann.

6.1.2 TwinCAT-Modul Zustandsmaschine

Neben den Zuständen (INIT, PREOP, SAFEOP und OP) gibt es entsprechende Zustandsübergänge, innerhalb derer allgemeine oder modulspezifische Aktionen auszuführen sind oder ausgeführt werden können. Die Zustandsmaschine ist sehr einfach konzipiert; in jedem Falle gibt es nur Übergänge zum nächsten oder vorherigen Schritt.

Hieraus ergeben sich die Zustandsübergänge: INIT zu PREOP (IP), PREOP zu SAFEOP (PS) und SAFEOP zu OP (SO). In umgekehrter Richtung bestehen die folgenden Zustandsübergänge: OP zu SAFEOP (OS), SAFEOP zu PREOP (SP) und PREOP zu INIT (PI). Einschließlich bis zum SAFEOP-Zustand finden alle Zustände und Zustandsübergänge innerhalb des Nicht-Echtzeitkontextes statt. Nur der Übergang SAFEOP zu OP, der Zustand OP und der Übergang OP zu SAFEOP finden im Echtzeitkontext statt. Diese Differenzierung hat einen Einfluss, wenn Ressourcen allokiert oder freigegeben werden, oder wenn Module sich bei anderen Modulen an- oder abmelden.



Zustand: INIT

Der INIT-Zustand ist nur ein virtueller Zustand. Sofort nach Erstellung eines Moduls wechselt das Modul von INIT zu PREOP, d.h. der IP-Zustandsübergang wird ausgeführt. Die Instanziierung und der IP-Zustandsübergang erfolgen immer zusammen, so dass das Modul nie im INIT-Zustand bleibt. Nur beim Entfernen des Moduls, bleibt es kurzzeitig im INIT-Zustand.

Transition: INIT zu PREOP (IP)

Während des IP-Zustandsübergangs meldet sich das Modul mit seiner eindeutigen ObjectID beim ObjectServer an. Die Initialisierungsparameter, die auch während der Objekterstellung allokiert werden, werden an das Modul weitergegeben. Bei diesem Übergang kann das Modul keine Verbindung zu anderen Modulen herstellen, weil nicht sicher ist, ob die anderen Module bereits bestehen und beim ObjectServer angemeldet sind. Wenn das Modul Systemressourcen benötigt (z.B. Speicherplatz), können diese während des Zustandsübergangs allokiert werden. Alle allokierten Ressourcen müssen dann entsprechend beim Übergang PREOP zu INIT (PI) wieder freigegeben werden.

Zustand: PREOP

Im PREOP-Zustand ist das Modul vollständig erstellt und auch normalerweise vollständig parametriert, auch wenn möglicherweise beim Übergang von PREOP zu SAFEOP weitere Parameter hinzukommen. Das Modul ist im ObjectServer angemeldet, es wurden aber noch keine Verbindungen mit anderen Modulen herstellt.

Transition: PREOP zu SAFEOP (PS)

In diesem Zustandsübergang kann das Modul Verbindungen mit anderen Modulen herstellen. Zu diesem Zweck hat es normalerweise, neben anderen Dingen, ObjectIDs von anderen Modulen mit den Initialisierungsdaten erhalten, die nun über den ObjectServer in reale Verbindungen mit diesen Modulen umgewandelt werden.

Der Übergang kann sowohl im Allgemeinen vom System, gemäß dem Konfigurator, als auch von einem anderen Modul (z.B. dem Parent-Modul) veranlasst werden. Im Verlauf dieses Zustandsübergangs können auch weitere Parameter übergeben werden. So kann z. B. das Parent-Modul eigene Parameter an das Child-Modul übergeben.

Zustand: SAFEOP

Das Modul ist noch im Nicht-Echtzeitkontext und wartet darauf, vom System oder von anderen Modulen in den OP-Zustand geschaltet zu werden.

Transition: SAFEOP zu OP (SO)

Sowohl der Zustandsübergang von SAFEOP zu OP als auch der Zustand OP, als auch der Übergang von OP zu SAFEOP finden im Echtzeitkontext statt! Ressourcen des Systems dürfen nicht mehr allokiert werden. Auf der anderen Seite können nun Ressourcen von anderen Modulen angefordert werden und Module können sich bei anderen Modulen anmelden, z. B. im Verlauf von Tasks, um einen zyklischen Aufruf zu erhalten.

Diese Transition sollte nicht für langlaufende Aufgaben verwendet werden. So sollten z. B. Dateioperationen schon im PS ausgeführt werden.

Zustand: OP

Im OP-Zustand nimmt das Modul seine Arbeit auf und ist im Sinne des TwinCAT-Systems voll aktiv.

Transition: OP zu SAFEOP (OS)

Dieser Zustandsübergang findet im Echtzeitkontext statt. Alle Aktionen aus dem SO-Übergang werden umgekehrt und alle beim SO-Übergang angeforderten Ressourcen werden wieder freigegeben.

Transition: SAFEOP zu PREOP (SP)

Alle Aktionen vom PS-Übergang werden umgekehrt und alle beim PS-Übergang angeforderten Ressourcen werden wieder freigegeben.

Transition: PREOP zu INIT (PI)

Alle Aktionen vom IP-Übergang werden umgekehrt und alle beim IP-Übergang angeforderten Ressourcen werden wieder freigegeben. Das Modul meldet sich beim ObjectServer ab und löscht sich normalerweise selbst (siehe „Lebensdauer“).

6.2 Modul zu Modul Kommunikation

TcCOM Module können untereinander kommunizieren. Dieser Artikel soll eine Übersicht über die unterschiedlichen Möglichkeiten geben. Es gibt vier Arten der Modul zu Modul Kommunikation:

- IO Mapping (Verknüpfung von Ein-/Ausgangs-Symbolen)
- IO Data Pointer
- Methodenaufrufe via Interface
- ADS

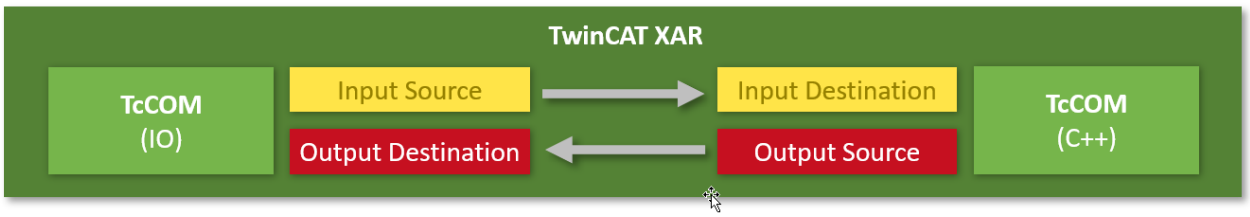
Diese vier Arten werden nun beschrieben.

IO Mapping (Verknüpfung von Ein-/Ausgangs-Symbolen)

Die Ein- und Ausgänge von TcCOM Modulen können durch IO Mapping in der gleichen Weise verknüpft werden, wie auch die Verknüpfungen zu physikalischen Symbolen der Feldbus-Ebene. Dafür werden Data Areas im TMC-Editor [▶ 123] angelegt, die entsprechende Ein-/Ausgänge beschreiben. Diese werden in der TwinCAT Solution dann verknüpft.

Durch ein Mapping werden dabei die Daten von einem Modul zum anderen bereitgestellt bzw. übernommen. Die Datenkonsistenz wird hierbei durch synchrones bzw. asynchrones Mapping sichergestellt, welches nach bzw vor dem eigentlichen zyklischen Programmablauf durchgeführt wird.

Die implementierende Sprache (SPS, C++, Matlab®) spielt dabei keine Rolle.



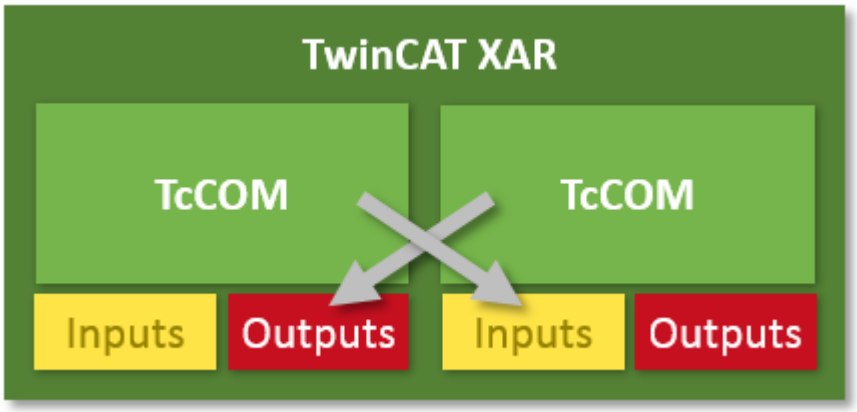
Folgendes Beispiel zeigt die Realisierung:

Beispiel12: Modulkommunikation: Verwendet IO Mapping [▶ 303]

IO Data Pointer

Über die Data Area Pointer, die im TMC-Editor angelegt werden, ist auch ein direkter Speicher-Zugriff innerhalb einer Tasks möglich.

Falls mehrere Aufrufende einer Task oder Aufrufende von anderen Tasks vorkommen, muss der Anwender die Datenkonsistenz durch entsprechende Mechanismen sicherstellen. Datenpointer stehen für C++ und Matlab® bereit.



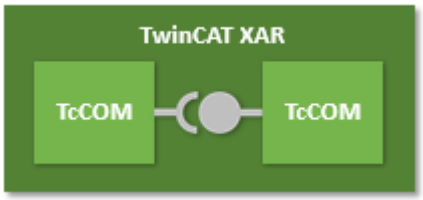
Folgendes Beispiel zeigt die Realisierung:

Beispiel10: Modulkommunikation: Verwendung von Datenzeigern [▶ 274]

Methodenaufrufe via Interfaces

Wie bereits zuvor beschrieben, können TcCOM Module Interfaces anbieten, die ebenfalls im TMC-Editor definiert werden. Wenn ein Modul diese implementiert („Implemented Interfaces“ im TMC Editor [▶ 115]) bietet es entsprechende Methoden an. Ein aufrufendes Modul wird dann einen „Interface Pointer“ zu diesem Modul besitzen, um die Methoden aufzurufen.

Es handelt sich dabei um blockierende Aufrufe, so dass der Aufrufer blockiert bis die aufgerufene Methode zurückkommt und somit die Rückgabewerte der Methoden direkt weiterverwenden kann. Falls mehrere Aufrufende einer Task oder Aufrufende von anderen Tasks vorkommen, muss der Anwender die Datenkonsistenz durch entsprechende Mechanismen sicherstellen.



Folgende Beispiele zeigen die Realisierung:

Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf [▶ 275]

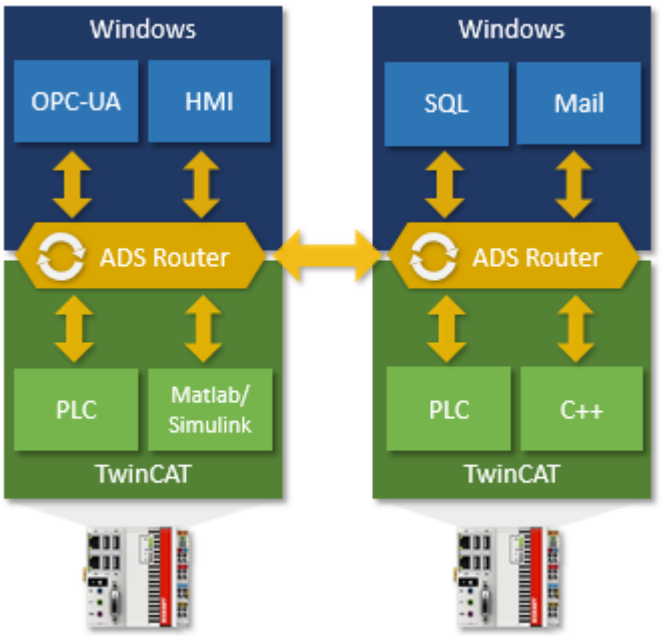
Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an [▶ 302]

Für die Kommunikation mit der PLC existieren weitere Beispiele [▶ 323].

ADS

ADS als interne Kommunikation des TwinCAT Systems allgemein kann auch genutzt werden, um zwischen Modulen zu kommunizieren. Es handelt sich dabei um eine azyklische, event-gesteuerte Kommunikation.

Gleichzeitig kann ADS auch genutzt werden, um Daten aus dem UserMode zu holen bzw. bereitzustellen und mit anderen Steuerungen (also über Netzwerk) zu kommunizieren. Ebenso kann ADS verwendet werden, um eine datenkonsistente Kommunikation z.B. zwischen Tasks / Cores / CPUs sicherzustellen. TcCOM Module können dabei sowohl Client (Anfragender) wie auch Server (Anbietender) sein. Die implementierende Sprache (SPS, C++, Matlab®) spielt dabei keine Rolle.



Folgende Beispiele zeigen die Realisierung:

Beispiel03: C++ als ADS Server [▶ 256]

Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch [▶ 266]

Beispiel07: Empfang von ADS Notifications [▶ 270]

Beispiel08: Anbieten von ADS-RPC [▶ 271]

7 Module - Handhabung

TcCOM-Module werden implementiert und nach einem Build geladen.

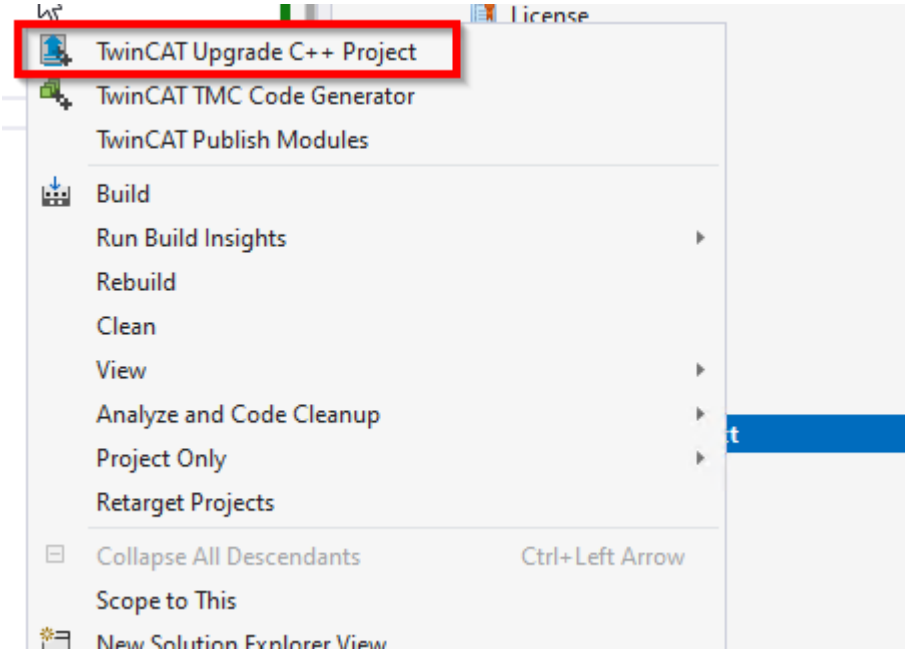
In diesem Abschnitt wird der Umgang mit Modulen beschrieben, wenn sie zwischen Systemen ausgetauscht werden. Diese Module werden in Versionierten C++-Projekte [▶ 53]n, die eine tmx-Datei erzeugen, programmiert um sie mit dem TwinCAT Loader [▶ 54] zu laden (seit TwinCAT 3.1 Build 4024).

Treiberprojekte

Bei älteren Projekten empfiehlt Beckhoff, auf die Versionierten C++-Projekte [▶ 53] konsequent zu wechseln, da diese beispielsweise folgende Vorteile bieten:

- Treibersignatur [▶ 24] über OEM-Zertifikate, die von Beckhoff bezogen werden.
- Versionierte Ablage [▶ 54] der Binaries.
- Online-Change Fähigkeit [▶ 155], wenn gewünscht.

Nach der Installation unter TwinCAT 3.1 Build 4026 steht auf dem C++ Knoten für die Migration von den älteren Projekten („Treiberprojekte“) eine Konvertierung bereit. Diese kann durch das Kontextmenü erreicht werden:



Die Migration erzeugt einen Report als HTML Datei, die beachtet werden soll.

7.1 Versionierte C++ Projekte

Ab TwinCAT 3.1. Build 4024.0

Die hier beschriebene Funktionalität ist ab TwinCAT 3.1. 4024.0 verfügbar.

Versionierte TwinCAT C++ Projekte ergeben beim Bauen eine Architektur-abhängige TMX-Datei und werden über den TwinCAT Loader [▶ 54] geladen. Sie müssen durch ein TwinCAT-Nutzerzertifikat signiert sein.

Ist ein C++ Projekt durch die Vorlage „Versioned C++ Project“ angelegt worden, werden bei einem Publish die Binär-Dateien im TwinCAT Engineering Repository (unter TwinCAT 3.1 Build 4024: *C:\TwinCAT\3.1\Repository* und TwinCAT 3.1 Build 4026 C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository) an einer Vendor- sowie Versions-spezifischen Stelle abgelegt.

Von hier werden Module auf das Zielsystem übertragen, falls sie benötigt werden:

- Windows mit TwinCAT 3.1 Build 4024: C:\TwinCAT\3.1\Boot\Repository
- Windows mit TwinCAT 3.1 Build 4026: C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Repository
- TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot/Repository

Dieses kann entweder zum Zeitpunkt des Aktivierens sein (**Activate Configuration**) oder zum Zeitpunkt des Online-Changes [▶ 155].

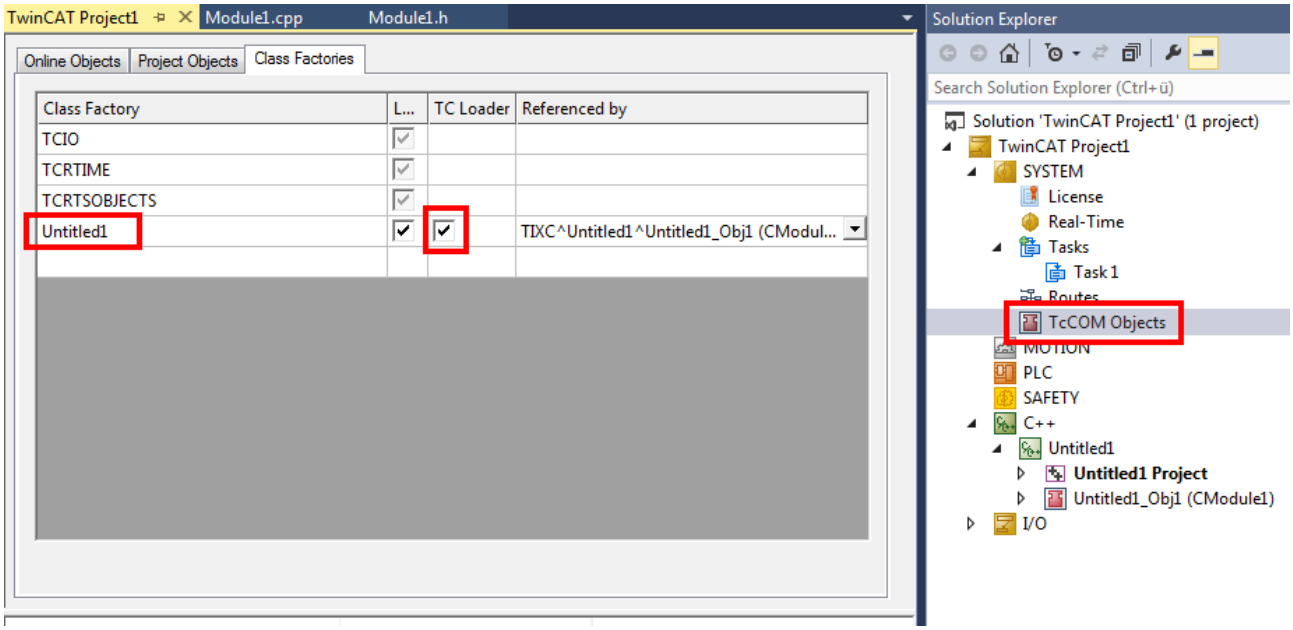
Zusätzlich ist es möglich, zum Transfer zwischen Engineering-Systemen der Binär-Version dieses Projektes ein Archiv erzeugen zu lassen, was durch die Projekt-Eigenschaften [▶ 149] konfiguriert wird.

7.2 Module starten

TwinCAT C++ Module können auf zwei Arten gestartet werden:

- Betriebssystem: Das Betriebssystem startet das TwinCAT-Modul als normalen Treiber.
  Es wird empfohlen auf den TwinCAT Loader mit TMX Dateien zu migrieren.
- TwinCAT Loader: [▶ 54] Der TwinCAT Loader startet das TwinCAT-Modul.
  - Der TwinCAT Loader verlangt eine Signatur [▶ 23] mit TwinCAT-Nutzerzertifikat.
  - Diese Option ist für Verschlüsselte Module [▶ 57] zwingend.
  - Der TwinCAT Loader wird für die Versionierten C++ Projekte [▶ 89] benötigt.

Über **System** -> **TcCOM Modules** -> Registerkarte **Class Factories** ist einsehbar, ob der TwinCAT Loader oder das Betriebssystem verwendet wird:



7.3 TwinCAT Loader

Ab TwinCAT 3.1. Build 4024.0

Die hier beschriebene Funktionalität ist ab TwinCAT 3.1. 4024.0 verfügbar.

TwinCAT 3 hat eine integrierte Funktion zum Laden von Modulen.

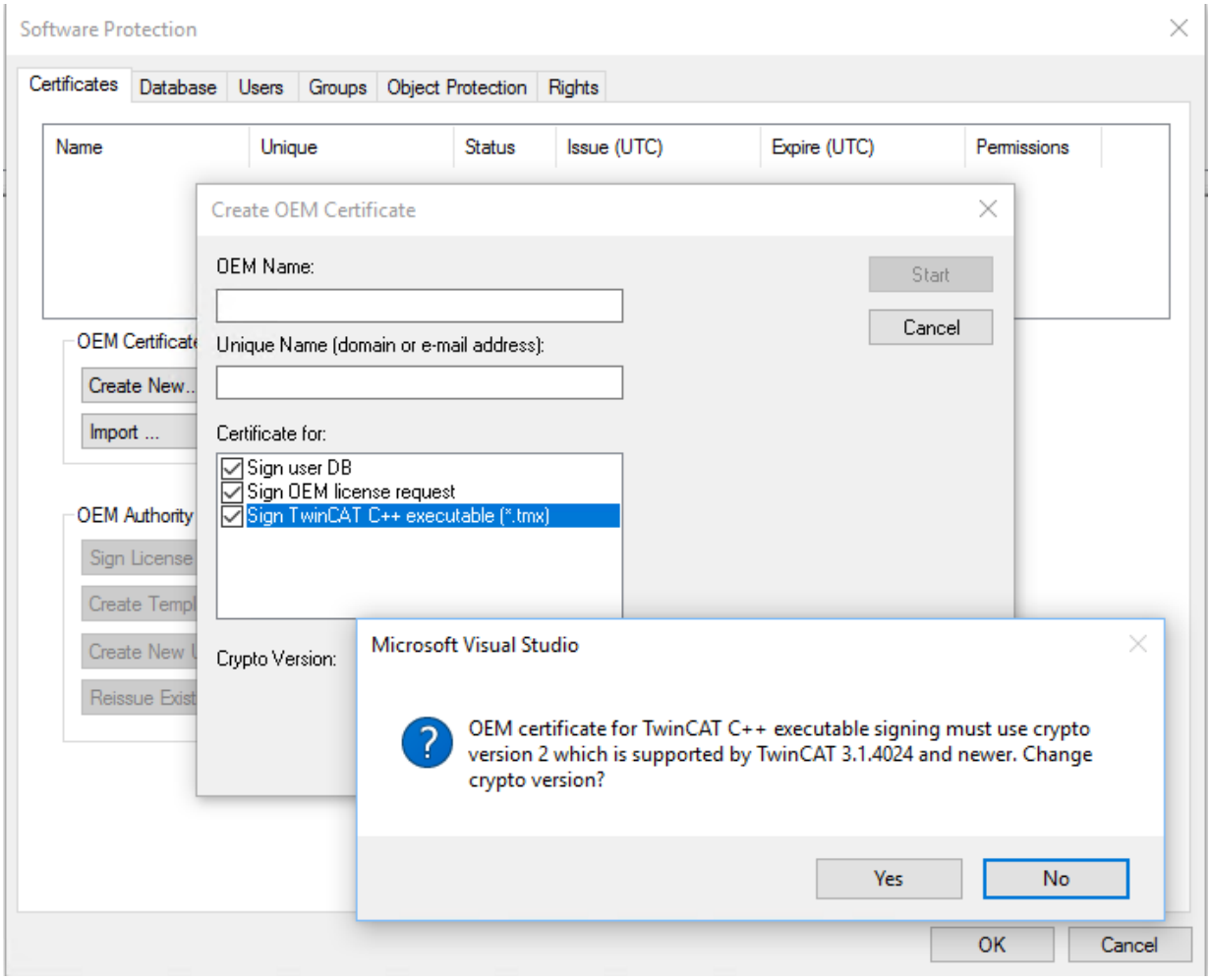
Mit dem TwinCAT Loader geladene Module:

- **Müssen** signiert sein: Testsignierung [▶ 55].
- **Können** verschlüsselt sein: Module verschlüsseln [▶ 57], wofür die TwinCAT Software Protection mit einer User DB konfiguriert sein muss.

7.3.1 Testsignierung

Die Testsignierung kann für TwinCAT mit dem gleichen TwinCAT-Nutzerzertifikat erfolgen, wie bei der eigentlichen Auslieferung (vgl.TwinCAT 3 Nutzerzertifikat beantragen [▶ 27]).

1. Für einen Testbetrieb z. B. während der Software-Entwicklung genügt die Erstellung eines TwinCAT-Nutzerzertifikates [▶ 28]. Beachten Sie dabei, dass Sie den Verwendungszweck „Sign TwinCAT C++ executeable (*.tmx)“ wählen. Hierfür ist dann die Crypto Version 2 erforderlich, eine Meldung dazu erscheint.

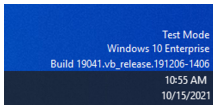


Auf XAR (und XAE, wenn es sich um einen lokalen Test handelt) aktivieren Sie den Testmodus, damit das Betriebssystem die selbstsignierten Zertifikate akzeptieren kann. Dies kann sowohl auf dem Engineering System (XAE) als auch auf dem Laufzeit-Systemen (XAR) vorgenommen werden.

Für Windows

Führen Sie mittels Administrator-Eingabeaufforderung folgendes aus:
`bcdedit /set testsigning yes`
und starten Sie das Zielsystem neu.
Es kann sein, dass Sie hierfür „SecureBoot“ abschalten müssen, was im Bios erfolgen kann.

Wenn der Testsigniermodus aktiviert ist, wird das rechts unten auf dem Desktop angezeigt. Das System akzeptiert nun alle signierten Treiber zur Ausführung.



Für TwinCAT/BSD

In der Datei `/usr/local/etc/TwinCAT/3.1/TcRegistry.xml` den Eintrag "`<Value Name="EnableTestSigning" Type="DW"1</Value>`" unter Key "`System`" eintragen.

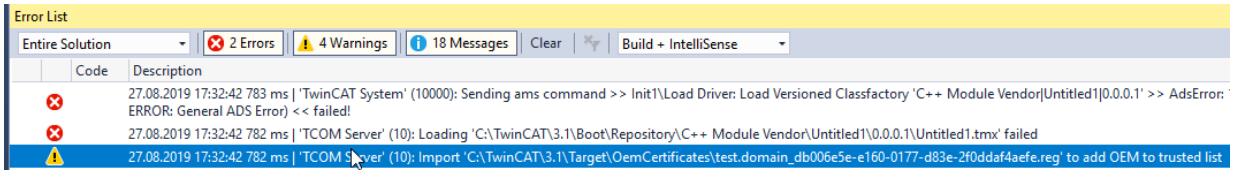
```
<Key Name="System">
  <Value Name="RunAsDevice" Type="DW">1</Value>
  <Value Name="RTimeMode" Type="DW">0</Value>
  <Value Name="AmsNetId" Type="BIN">052445B00101</Value>
  <Value Name="LockedMemSize" Type="DW">33554432</Value>
  <Value Name="EnableTestSigning" Type="DW">1</Value>
</Key>
```

Danach den TwinCAT System Service neu starten:

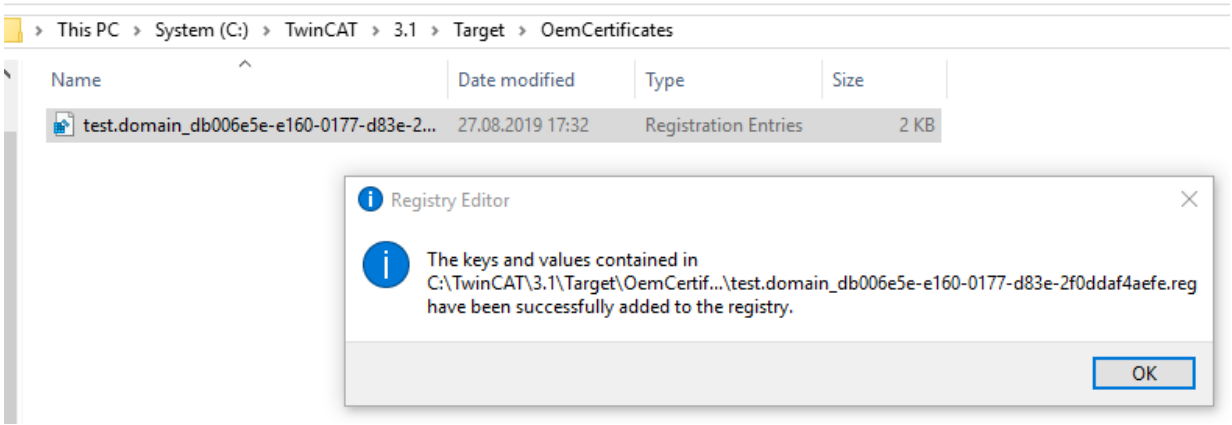
```
doas service TcSystemService restart
```

Nach der jeweiligen Vorgehensweise akzeptiert das System alle signierten Treiber zur Ausführung.

1. Bei dem ersten Aktivieren (Activate Configuration) mit einem TwinCAT-Nutzerzertifikat wird von dem Zielsystem festgestellt, dass dem Zertifikat nicht vertraut wird und der Aktivierungsvorgang wird abgebrochen:

   Für Windows:
   Ein lokaler Nutzer mit Administrationsrechten kann über die erstellte REG-Datei durch einfaches Ausführen dem Zertifikat vertrauen:

   Für TwinCAT/BSD:
   Falls das Paket „Tcimportcert“ nicht installiert ist dieses installieren: `pkg install tcimportcert`
   Mittels `doas tcimportcert /usr/local/etc/TwinCAT/3.1/Target/OemCertificates/<CreatedFile>.reg` dem Zertifikat vertrauen.
   Danach den TwinCAT System Service neu starten oder das System neu starten:
   `doas service TcSystemService restart`

   ⇨ Diese Verfahren ermöglichen es nur, C++ Module mit einer Signatur von den vertrauten TwinCAT-Nutzerzertifikaten in die Ausführung zu bringen.

2. Nach diesem Prozess können Sie das TwinCAT-Nutzerzertifikat zur Signierung mit dem Testmodus des Betriebssystems verwenden.
   Dieses wird in den Projekt-Eigenschaften [▶ 151] konfiguriert.
   Um das Passwort des TwinCAT-Nutzerzertifikates nicht im Projekt abzulegen, wo es beispielsweise auch in einer Versionsverwaltung landen würde, verwenden Sie das TcSignTool [▶ 59].

Wenn Sie das TwinCAT-Nutzerzertifikat ohne TestMode zur Auslieferung nutzen wollen, müssen Sie das Zertifikat von Beckhoff gegenzeichnen [▶ 26] lassen.

7.3.2 Module verschlüsseln

TwinCAT C++ Module, die über den TwinCAT Loader geladen werden (TMX-Dateien) können verschlüsselt werden, d.h. ein Schlüssel schützt den Inhalt des Treibers vor Manipulation und Reverse Engineering auf Datei-Ebene.

Kein Debuggen

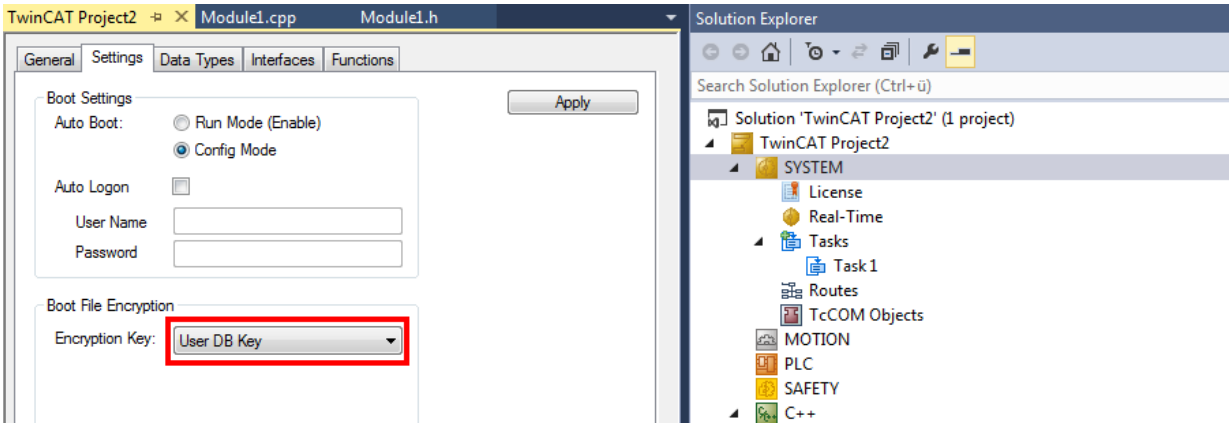
Verschlüsselte Module können nicht nach Fehlern durchsucht werden. Verschlüsselte Module werden nicht im Debugger angezeigt.

Die Modulverschlüsselung wird folgendermaßen aktiviert:

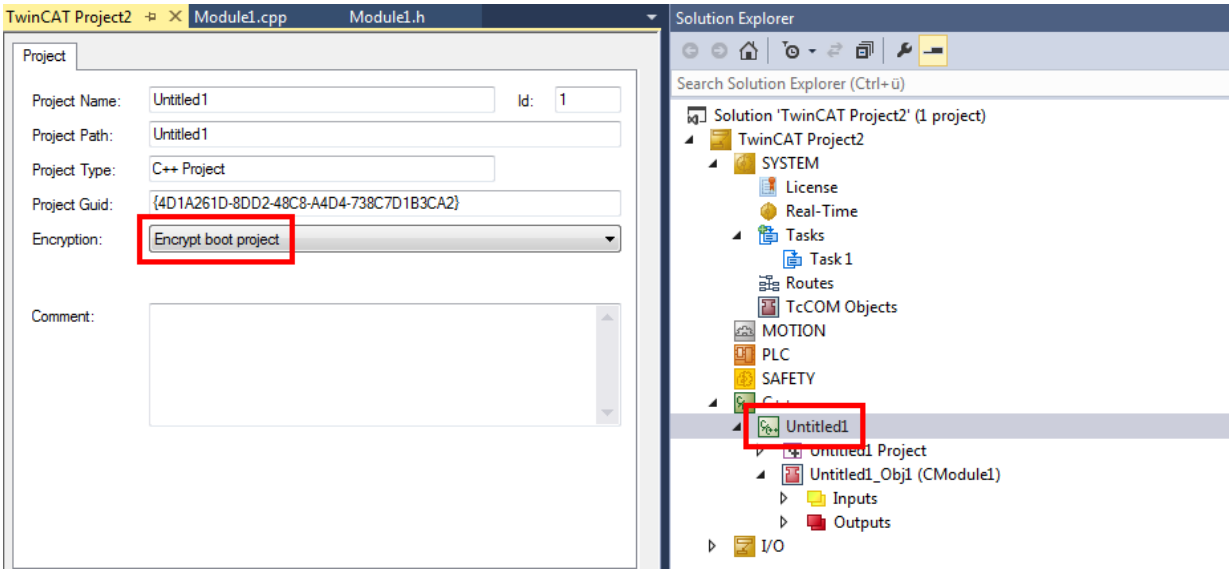
✓ Die TwinCAT Software Protection muss konfiguriert werden.

✓ Es wird ein TwinCAT-Nutzerzertifikat mit Sign UserDB-Rechten benötigt.

1. Wählen Sie im Systembaum der Solution **User DB Key** als Boot File Encryption Key.



2. Wählen Sie das C++ Projekt und aktivieren die Verschlüsselung dort:



3. Zum Starten muss ein verschlüsseltes Modul mit dem TwinCAT Loader (nicht dem Betriebssystem) geladen werden.

⇨ Für nicht-versionierte Treiber: Die Treiber werden bei der Übertragung in das _deployment Verzeichnis des Projekts verschlüsselt.

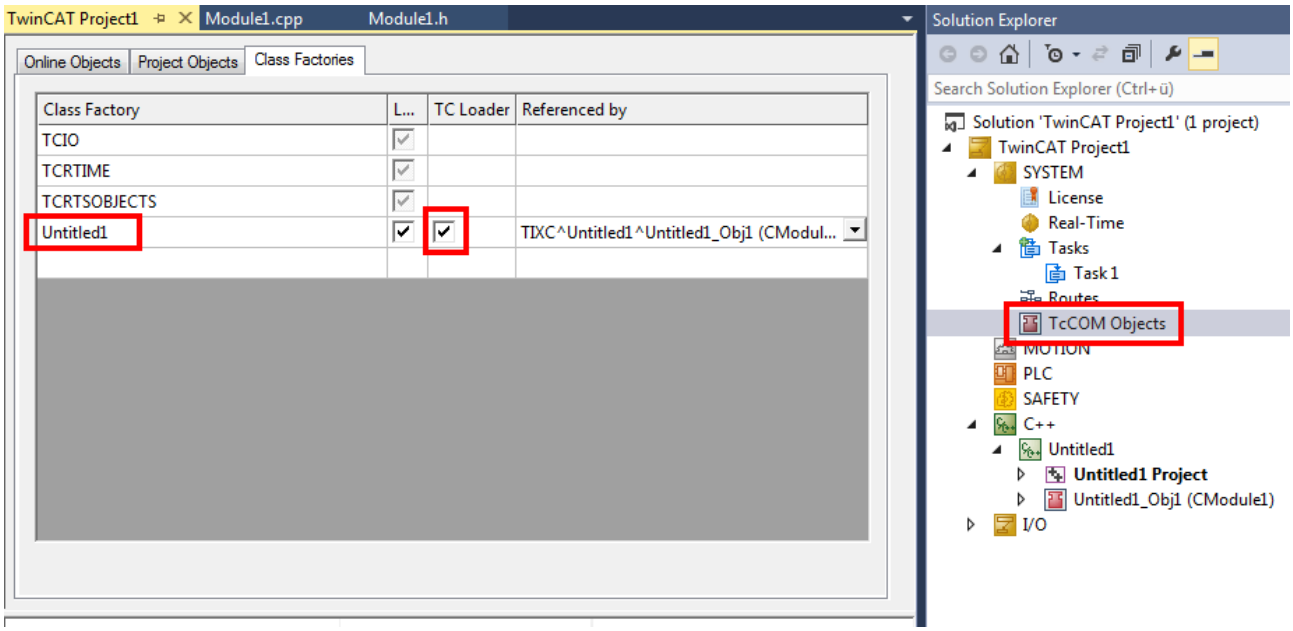
⇨ Für versionierte TMX: Die Treiber werden im XAE unverschlüsselt abgelegt und beim Aktivieren auf das Zielsystem verschlüsselt.

⇨ Sollte die Funktion mit versionierten C++ Projekten eingesetzt werden, werden die TMX Dateien wie üblich im Repository [▶ 53] abgelegt.

TwinCAT C++ Module können auf zwei Arten gestartet werden:

- Betriebssystem: Das Betriebssystem startet das TwinCAT-Modul als normalen Treiber.
  Es wird empfohlen auf den TwinCAT Loader mit TMX Dateien zu migrieren.
- TwinCAT Loader: [▶ 54] Der TwinCAT Loader startet das TwinCAT-Modul.
  - Der TwinCAT Loader verlangt eine Signatur [▶ 23] mit TwinCAT-Nutzerzertifikat.
  - Diese Option ist für Verschlüsselte Module [▶ 57] zwingend.
  - Der TwinCAT Loader wird für die Versionierten C++ Projekte [▶ 89] benötigt.

Über **System** -> **TcCOM Modules** -> Registerkarte **Class Factories** ist einsehbar, ob der TwinCAT Loader oder das Betriebssystem verwendet wird:



7.3.3 Return-Codes

Das Laden eines Moduls mit dem TwinCAT Loader kann aus verschiedenen Gründen fehlschlagen.

Return-Codes unter Windows

Hex	Beschreibung
0xC1	Datei ist beschädigt, PE-Datei Prüfsumme Fehler.
0x241	Signaturfehler: TwinCAT Nutzerzertifikat passt nicht zum Datei-Hash.
0x4FB	Signaturfehler: Datei nicht signiert.
0x1772	Datei ist verschlüsselt, kann aber nicht mit bekannten Schlüsseln dekodiert werden.

Return-Codes unter TwinCAT/BSD

Hex	Beschreibung
0xC0000001 0xC0000004 0xC000007B 0xC0000173 0xC0000221	Datei ist beschädigt, PE-Datei Prüfsumme Fehler.
0xC0000102 0xC0000424	Signaturfehler: TwinCAT Nutzerzertifikat passt nicht zum Datei-Hash.
0x4FB	Signaturfehler: Datei nicht signiert.
0xC0000428	Signaturfehler: Genutztes Zertifikat nicht gegengezeichnet. Testmode nötig.
0xC0000603	Signaturfehler: Zertifikat wird nicht vertraut. Hinzufügen nötig (vgl. „tcinsertcert“)
0xC0000385	Signaturfehler: Datei nicht signiert. Einstellungen im Engineering überprüfen.
0xC0000293	Datei ist verschlüsselt, kann aber nicht mit bekannten Schlüsseln dekodiert werden.
0xC0000225	Datei nicht gefunden.

7.3.4 TcSignTool – Ablage des Zertifikatspassworts außerhalb des Projektes

Das TcSignTool-kann verwendet werden, um ein Passwort zu einem TwinCAT-Nutzerzertifikat in der Registry abzulegen. Somit benötigt man das Passwort nicht in den Projekten, wo die Passwörter ggf. unbeabsichtigt in Versionskontrollsystemen landen würden.

Das TcSignTool ist ein Kommandozeilen-Programm, welches sich im Pfad *C:\TwinCAT\3.1\sdk\Bin* bzw. *C:\Program Files (x86)\Beckhoff\TwinCAT\3.1\SDK\Bin* befindet.

Die Ablage des Passworts wird mit folgenden Parametern durchgeführt:

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

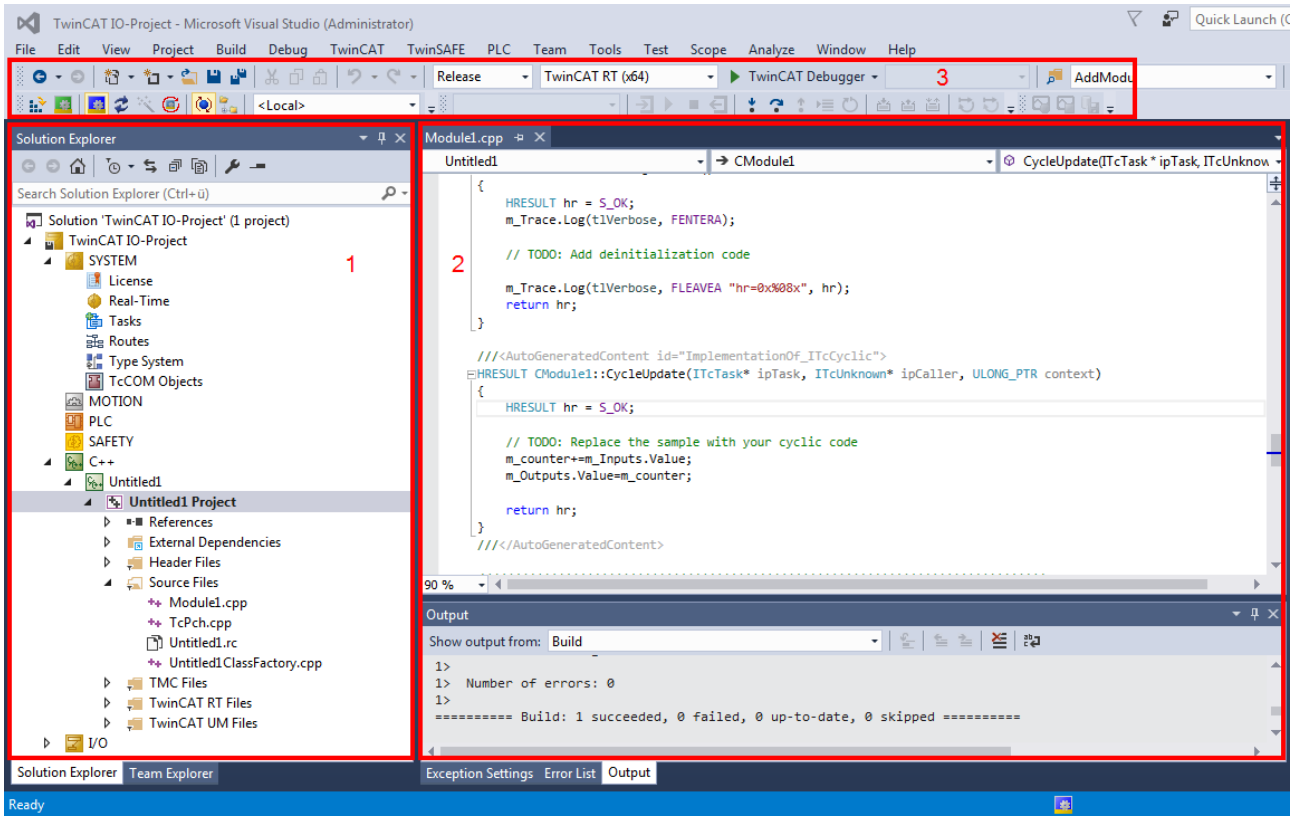
Mit folgenden Parametern wird das Passwort gelöscht:

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

Die Ablage des unverschlüsselten Passworts erfolgt unter
`HKEY_CURRENT_USER\SOFTWARE\Beckhoff\TcSignTool\`

8 TwinCAT C++ Entwicklung

Übersicht der Entwicklungsumgebung



Das Layout des Visual Studios ist flexibel und anpassbar, so dass an dieser Stelle nur eine Übersicht einer üblichen Anordnung gegeben werden kann. Der Nutzer ist aber frei in der Konfiguration der Fenster und Anordnungen.

1. In der TwinCAT Solution kann durch Rechts-Klick auf das C++ Icon ein TwinCAT C++ Projekt angelegt werden. Innerhalb dieses Projektes sind zum einen die Quellen („Untitled Project“) von ggf. mehreren Modulen [▶ 39] enthalten. Zum anderen können Modulinstanzen („Untitled1_Obj1 (CModule1)“) angelegt werden. Die Modulinstanzen haben Ein-/Ausgänge die auf die übliche Weise verbunden („Link“) werden können. Zusätzlich gibt es weitere Möglichkeiten [▶ 50], wie Module interagieren können.
2. Zur Programmierung wird der Visual Studio Editor für Visual C++ verwendet. Hier ist insbesondere auf die Drop-Down Boxen zur schnellen Navigation innerhalb einer Datei hinzuweisen. Im unteren Bereich wird die Ausgabe des Compile-Vorganges ausgegeben. Es kann auch auf die TwinCAT Meldungen (vgl. Modul-Nachrichten zum Engineering (Logging / Tracing) [▶ 223]) umgeschaltet werden.
   In den Editoren können die üblichen Features wie Haltepunkte (vgl. Debuggen [▶ 80]) genutzt werden.
3. In der frei konfigurierbaren Toolbar-Leiste ist üblicherweise die Toolbar für TwinCAT XAE Base platziert. **Activate Configuration**, **RUN**, **CONFIG**, Auswahl des Zielsystems (hier **<Local>**) und einige andere Buttons geben schnellen Zugriff auf häufig genutzte Funktionen. Der **TwinCAT Debugger** ist der Button, um die Verbindung zum Zielsystem in Bezug auf C++ Module aufzubauen (die SPS nutzt einen eigenständigen Debugger). Bei TwinCAT C++ ist wie bei anderen C++ Programmen im Gegensatz zur PLC zu unterscheiden zwischen „Release“ und „Debug“. Bei einem Build-Vorgang für „Release“ wird der Code soweit optimiert, dass ein Debugger unter Umständen die Haltepunkte nicht mehr zuverlässig erreicht, sowie auch falsche Daten angezeigt werden.

Ablauf

In diesem Abschnitt werden die Prozesse für Programmierung, Kompilierung und Starten eines TwinCAT C++ Projekts beschrieben.

Er soll einen allgemeinen Überblick über den Engineering-Prozess für TwinCAT C++ Projekte mit Verweisen auf die entsprechende ausführliche Dokumentation geben. Der Schnellstart [▶ 62] geht die gemeinsamen Schritte im Einzelnen durch.

1. Typen-Deklaration und Modultyp:
   Der TwinCAT Module Class Editor (TMC) [▶ 92] und TMC Code Generator werden für die Definition von Datentypen und Schnittstellen, und auch der Module, die diese verwenden, herangezogen.
   Der TMC Code Generator generiert Quellcode anhand der bearbeiteten TMC-Datei und bereitet Datentypen / Schnittstellen vor, die in anderen Projekten (wie SPS) zu verwenden sind.
   Das Bearbeiten und den Code Generator Starten kann so oft wie gewünscht gemacht werden - die Code-Generierung achtet auf programmierten Benutzercode und bewahrt diesen.
2. Programmierung
   Die bekannte Visual Studio C++ Programmierungsumgebung wird für die Entwicklung und das Debugging [▶ 80] des benutzerdefinierten Codes innerhalb der Code-Vorlage verwendet.
3. Module [▶ 39] instanziieren
   Das Programm beschreibt eine Klasse, die als Objekte instanziiert wird. Der TwinCAT Module Instance Configurator [▶ 135] wird für die Konfiguration der Instanz verwendet. Allgemeine Konfigurationselemente sind: Task zuweisen, Symbolinformation für Laufzeit (TwinCAT Module Instance (TMI) Datei) herunterladen oder Parameter- /Schnittstellenzeiger festlegen.
4. Mapping von Variablen
   Die Ein- und Ausgangsvariablen eines Objekts können mit Variablen von anderen Objekten oder SPS-Projekten unter Verwendung des normalen TwinCAT Systemmanagers verknüpft werden.
5. Erstellung (Building)
   Bei der Erstellung (Kompilieren und Verknüpfung) des TwinCAT C++ Projekts werden alle Komponenten für die ausgewählte Plattform kompiliert. Die Plattform wird automatisch festgelegt, wenn das Zielsystem ausgewählt ist.
6. Veröffentlichung (Publishing)
   Die Veröffentlichung eines Moduls erstellt die Treiber für alle Plattformen und bereitet es für die Verteilung vor. Das erzeugte Verzeichnis im Repository kann verteilt werden, ohne dass der Quellcode übergeben werden muss. Es wird ausschließlich Binärcode mit Schnittstellenbeschreibung übergeben.
7. Signatur (siehe Modul-Signierung [▶ 23])
   Die TwinCAT-C++-Projekte müssen signiert sein. Der Signaturprozess kann benutzerdefiniert [▶ 37] sein.
8. Aktivierung
   Der TwinCAT C++ Treiber kann wie jedes TwinCAT Projekt mittels **Activate Configuration** aktiviert werden. Daraufhin bittet ein Dialog darum, TwinCAT in den RUN-Modus zu setzen.
   Das (von IEC61131-basierten Systemen her bekannte) Debuggen [▶ 80] in Echtzeit, sowie das Setzen von (bedingten) Haltepunkten ist im Falle von TwinCAT C++ Modulen möglich.

⇨ Das Modul läuft unter Echtzeitbedingungen.

9 Schnellstart

Dieser Schnellstart zeigt, wie man in kurzer Zeit mit dem TwinCAT C++-Modul Engineering vertraut werden kann. Jeder Schritt der Erstellung eines im Echtzeitkontext laufenden Moduls ist ausführlich beschrieben.

Dabei wird auf zwei unterschiedliche Szenarien eingegangen:

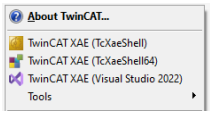
- TwinCAT Versioned C++-Projekte.
  Module auf dieser Projektbasis werden durch TwinCAT geladen und sind in Binärform versioniert abgelegt.
- Aufbauend auf den versionierten C++-Projekten wird gezeigt, wie zwischen den unterschiedlichen Versionen als Online-Change umgeschaltet werden kann.

Bitte beachten Sie vor dem Schnellstart die Vorbereitung - nur einmal! [▶ 23] Insbesondere bereiten Sie die jeweilige Treibersignierung vor.

9.1 TwinCAT 3-Projekt erstellen

TwinCAT Engineering-Umgebung (XAE) starten

Microsoft Visual Studio® kann über das TwinCAT SysTray Icon gestartet werden.

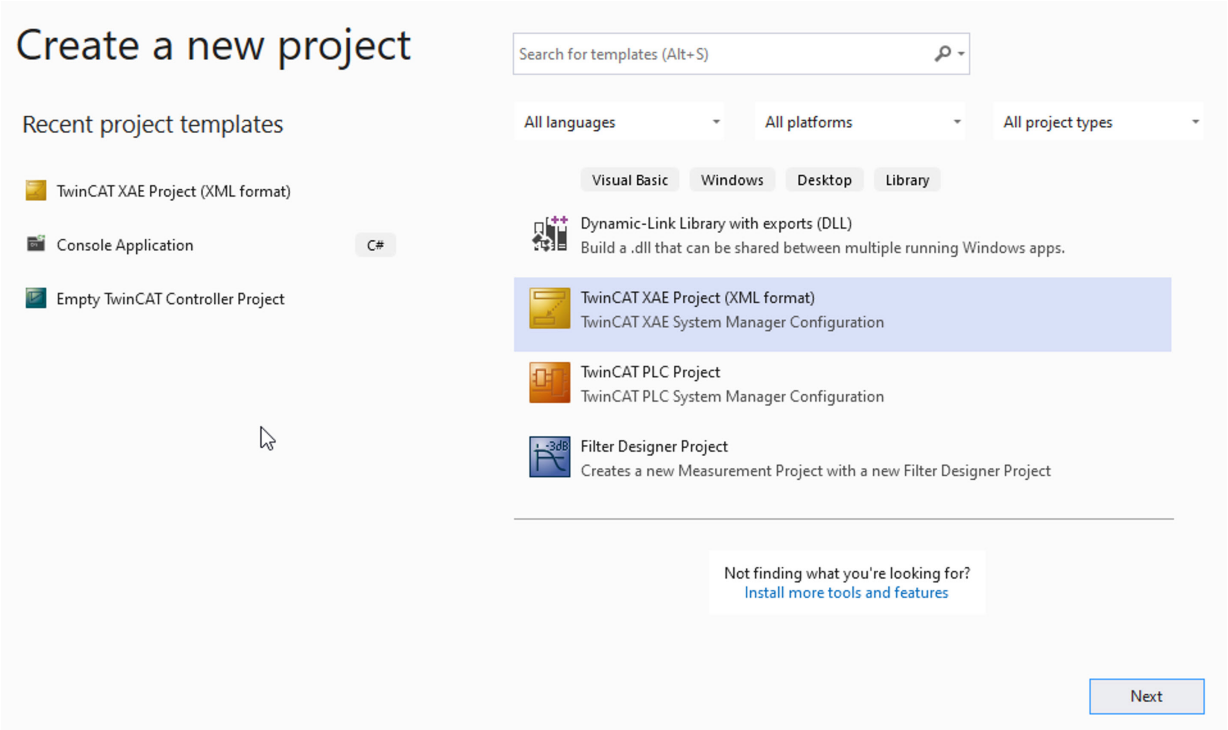


Dabei werden die von Microsoft Visual Studio® Versionen angeboten, in denen TwinCAT aktiviert ist. Alternativ kann das Microsoft Visual Studio® auch über das Start-Menü gestartet werden.

TwinCAT 3 C++-Projekt erstellen

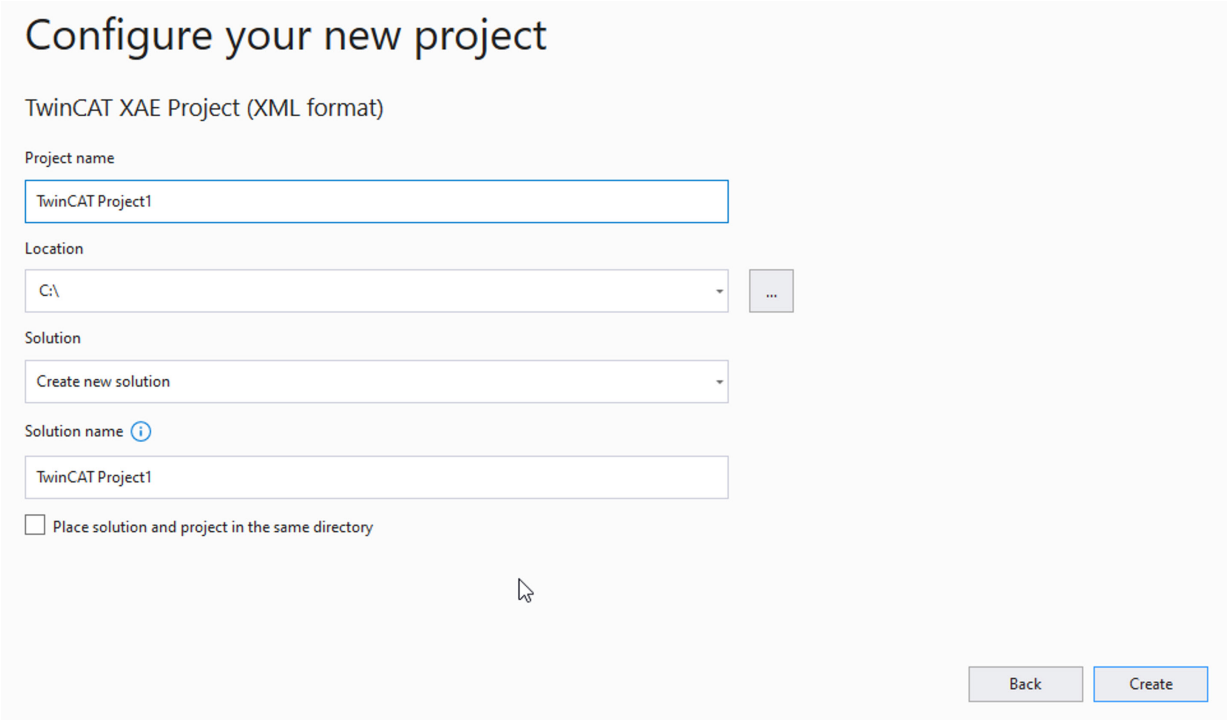
Um ein TwinCAT C++-Projekt zu erstellen, führen Sie folgende Schritte aus:

1. Wählen Sie über die Start Page **New TwinCAT Project ...** aus.

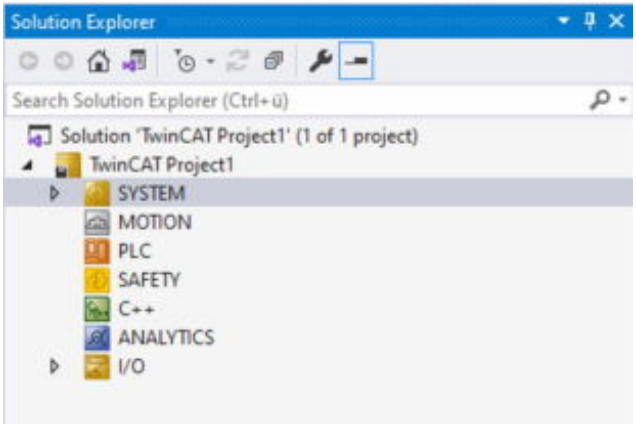


2. Alternativ legen Sie mit Klick auf: **File** -> **New** -> **Project** ein Projekt an.
   ⇨ Es werden alle bestehenden Projektvorlagen angezeigt.
3. Wählen Sie **TwinCAT XAE Project** und geben Sie optional einen passenden Projektnamen ein.

4. Klicken Sie auf **OK**.



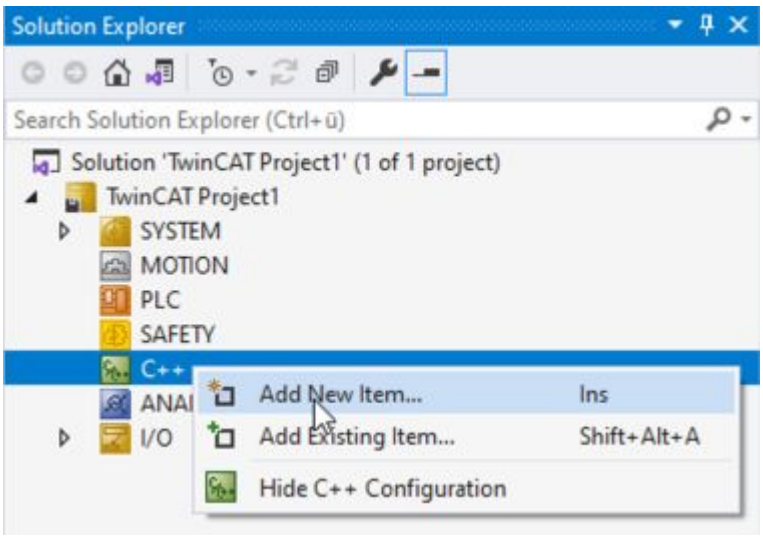
⇨ Daraufhin zeigt der Visual Studio Solution Explorer das TwinCAT 3-Projekt.



9.2 TwinCAT 3 C++ Projekt erstellen

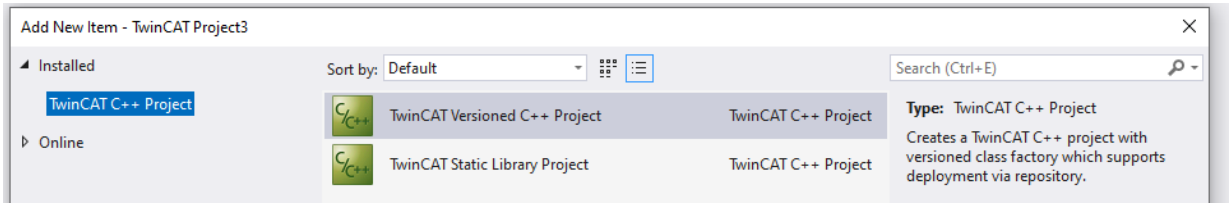
Nach der Erstellung eines TwinCAT 3 Projekts öffnen Sie den C++-Knoten und gehen Sie folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste auf **C++** und wählen Sie **Add New Item...** aus.
   Wenn das grüne C++ Symbol nicht aufgeführt ist, ist entweder ein Zielgerät ausgewählt, welches kein TwinCAT C++ unterstützt oder die TwinCAT Solution wurde aktuell in einem nicht C++ fähigen Visual Studio geöffnet (vgl. Installation [▶ 21]).



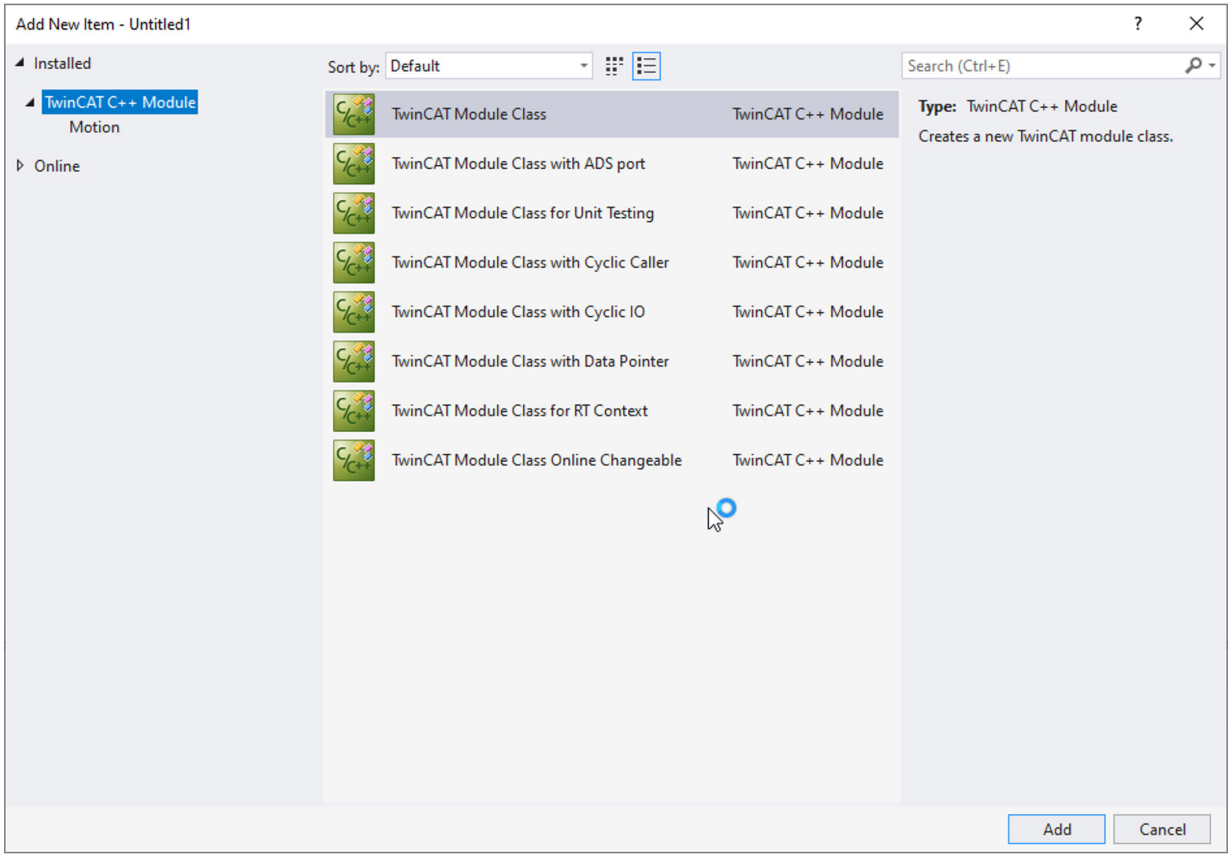
   ⇨ Der TwinCAT C++ Projekt-Assistent [▶ 89] wird eingeblendet und alle bestehenden Projektvorlagen werden aufgeführt.

2. Wählen Sie **TwinCAT Versioned C++ Project** aus, geben optional einen entsprechenden Projektnamen ein und klicken auf **OK**.
   Alternativ verwenden Sie das **TwinCAT Static Library Project**, das eine Umgebung für das Programmieren von statischen TwinCAT-C++ Bibliotheken bereitstellt (siehe Beispiel 25 [▶ 315]).



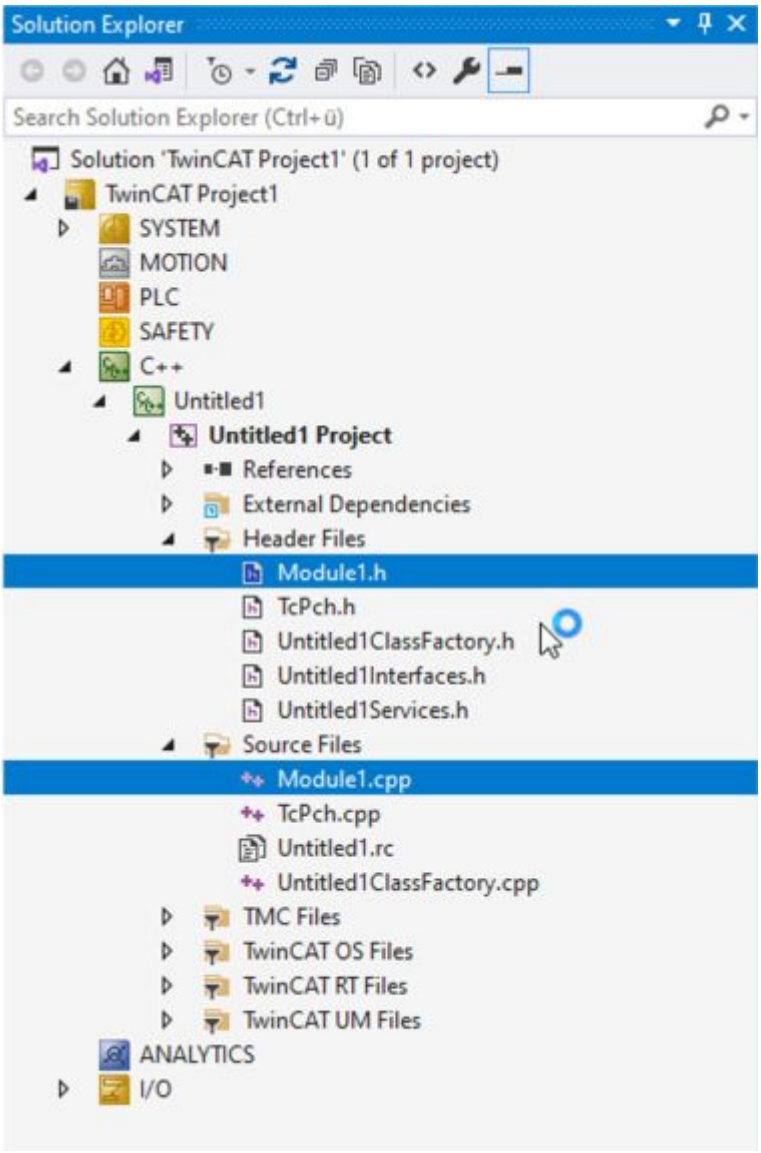
   ⇨ Der TwinCAT Modul-Assistent [▶ 90] wird eingeblendet.

3. Wählen Sie in diesem Fall **TwinCAT Module Class with Cyclic IO** aus und klicken auf **OK**. Falls Sie die Online-Change-Fähigkeit nutzen wollen, wählen Sie das **TwinCAT Module Class Online Changeable** aus.



4. Geben Sie im Dialogfenster **TwinCAT Class Wizard** einen eindeutigen Namen ein oder fahren Sie mit dem Vorschlag „Module1“ fort.

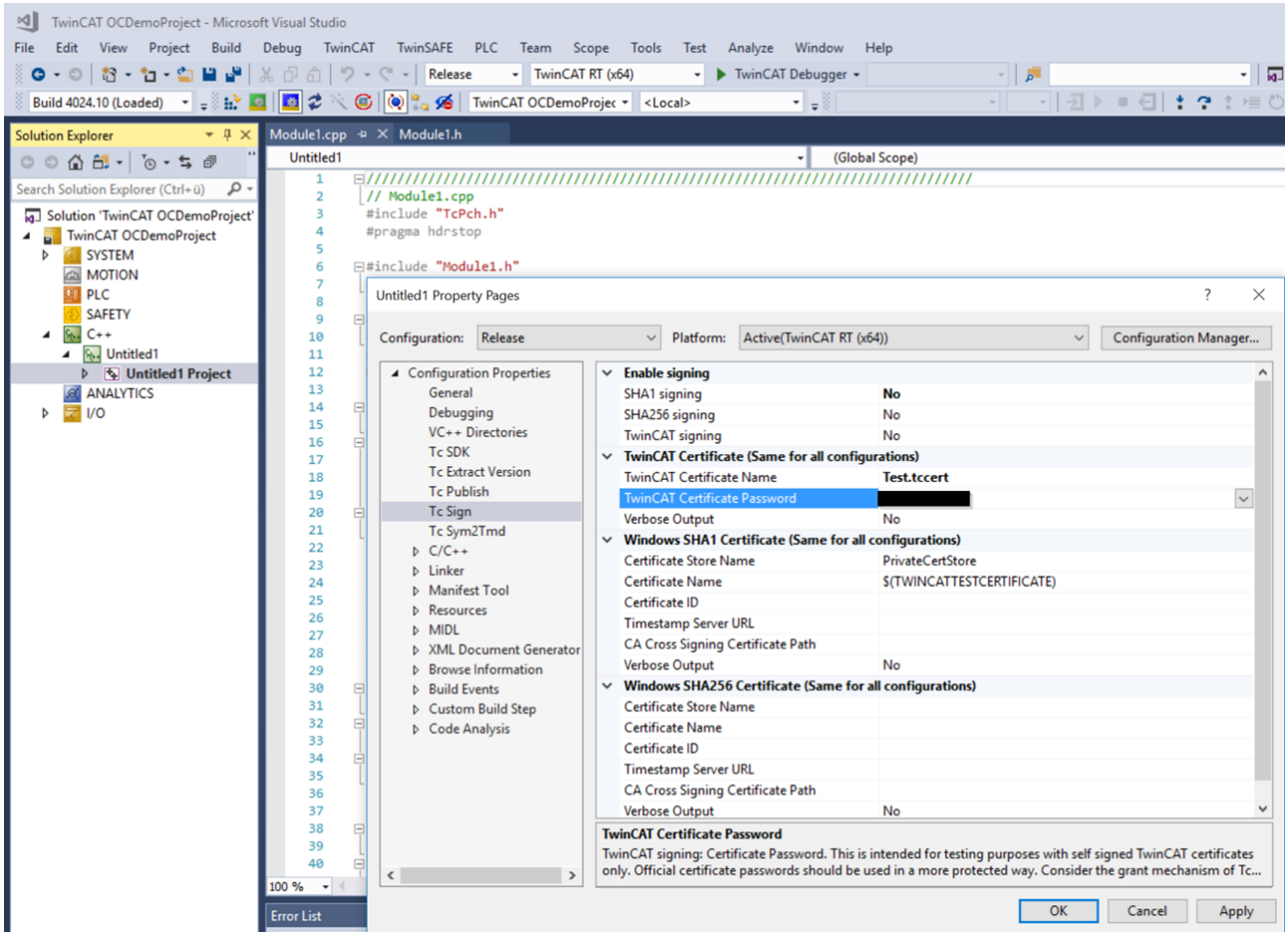
⇨ Daraufhin wird ein TwinCAT 3 C++ Projekt auf Basis des ausgewählten Templates erstellt:



9.3 TwinCAT 3 C++-Projekt konfigurieren

✓ Sie haben ein TwinCAT Versioned C++- Projekt angelegt und auch ein Modul durch den Assistenten erstellen lassen

1. Gehen Sie per Rechtsklick in die Eigenschaften des Projektes.
2. Im Tab **Tc Sign** aktivieren Sie das TwinCAT Signing.
3. Falls Sie noch kein TwinCAT-Nutzerzertifikat erzeugt haben, folgen Sie der Anleitung und beachten dabei das **Sign TwinCAT C++ executeables** anzuwählen.
4. Geben Sie den Dateinamen des TwinCAT-Nutzerzertifikates sowie das Passwort an.
   (Beachten Sie, dass dieses unverschlüsselt in der Solution abgelegt wird und dadurch auch z.B. via Versionskontrolle auf Servern geladen wird. Ggf. das TcSignTool [▶ 59] verwenden.)

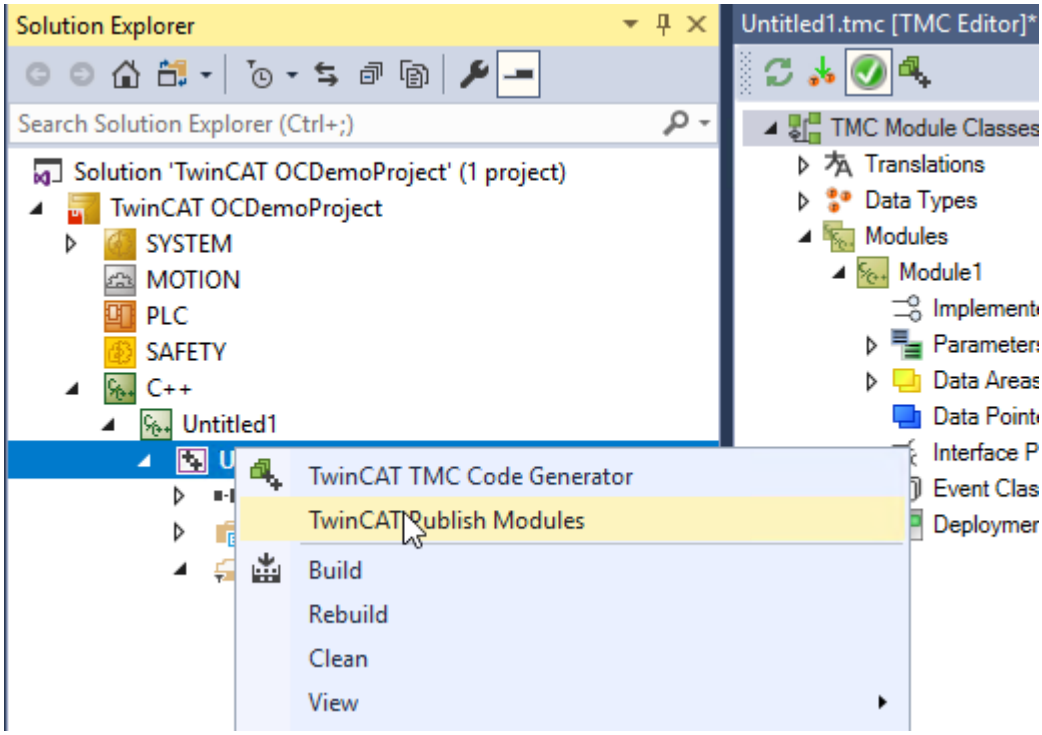


9.4 TwinCAT 3 C++-Projekt implementieren

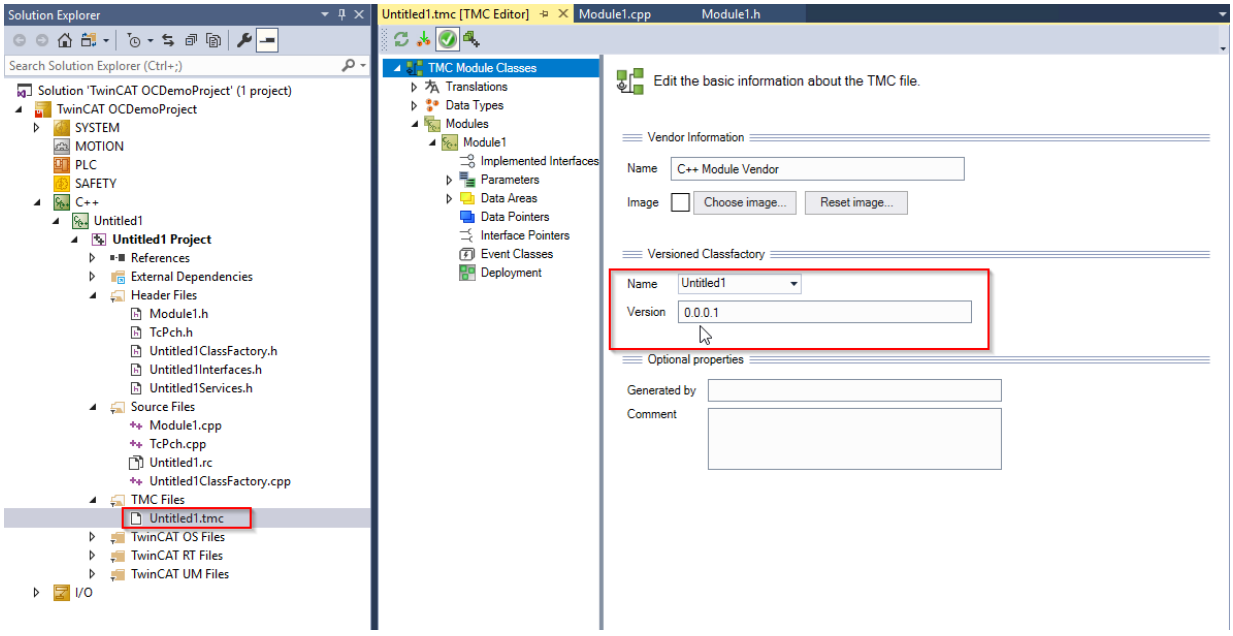
In diesem Artikel wird beschrieben, wie das Beispielprojekt geändert werden kann.

Nach der Erstellung eines TwinCAT C++-Projekts und dem Öffnen der <MyClass>.cpp (in diesem Beispiel Module1.cpp) beginnt die Implementierung.

1. Die Methode <MyClass>::CycleUpdate() wird zyklisch aufgerufen - das ist die Stelle, wo die zyklische Logik zu positionieren ist. Fügen Sie an dieser Stelle den gesamten zyklischen Code hinzu. Zur Navigation verwenden Sie die Dropdown-Menüs am oberen Rand des Editors.



2. In diesem Fall wird ein Zähler um den Wert der Variablen Value im Eingangsabbild (`m_Inputs`) inkrementiert. Ersetzen Sie eine Zeile, um den Zähler zu inkrementieren, ohne Abhängigkeit des Werts vom Eingangsabbild.
   Diese Zeile:
   ```
   m_counter+=m_Inputs.Value;
   ```
   durch diese ersetzen:
   ```
   m_counter++;
   ```
3. Speichern Sie die Änderungen.
4. Falls Sie das Modul für den Online-Change vorbereitet haben, beachten Sie, dass hier die Version 0.0.0.1 implementiert wurde, wie Sie im TMC Editor sehen können.

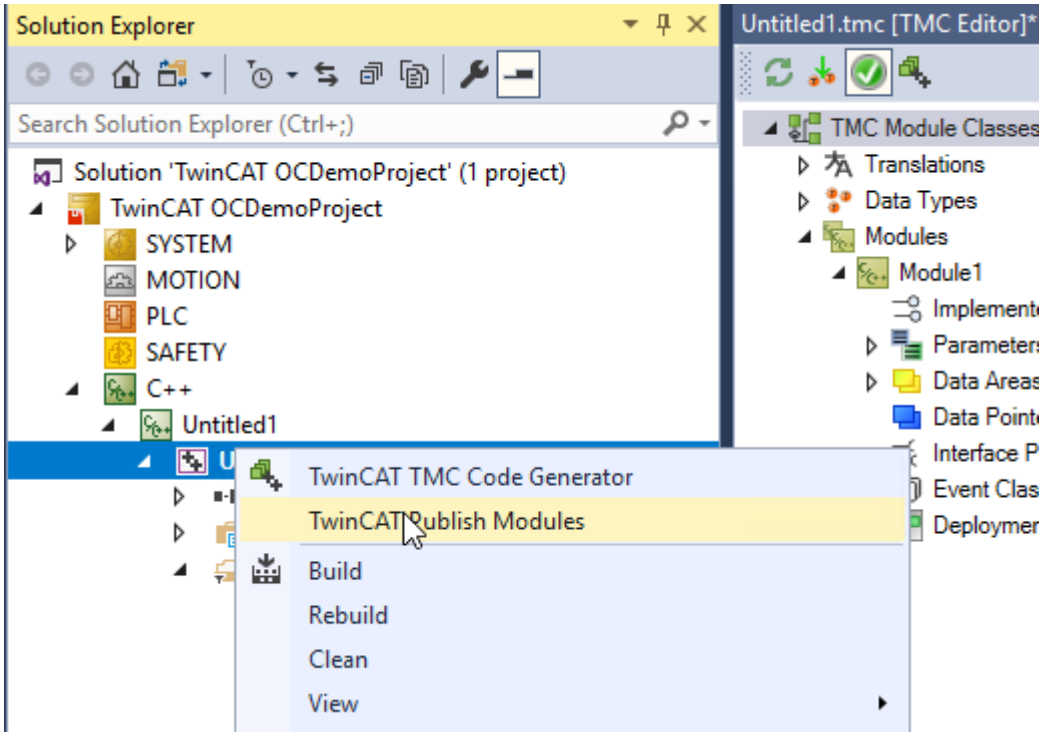


5. Das Projekt kann nun gebaut werden und die eigentliche Implementierung erfolgt, bis lokale Tests erfolgreich verlaufen sind.

9.5 TwinCAT 3 C++-Projekt in der Version 0.0.0.1 publishen

Nachdem ein TwinCAT C++-Projekt erstellt, kompiliert und getestet wurde, kann das Projekt veröffentlicht werden, d. h. es wird zur Weitergabe vorbereitet.

1. Klicken Sie im Kontextmenü auf **TwinCAT Publish Modules** um das Modul zu publizieren und damit im lokalen Repository als Version 0.0.0.1 bereitzustellen



⇨ Das Modul ist in der Version 0.0.0.1 mit dem inkrementierenden Zähler auf dem Engineering-Gerät veröffentlicht.

9.6 TwinCAT 3 C++-Projekt Version 0.0.0.2 implementieren und publishen

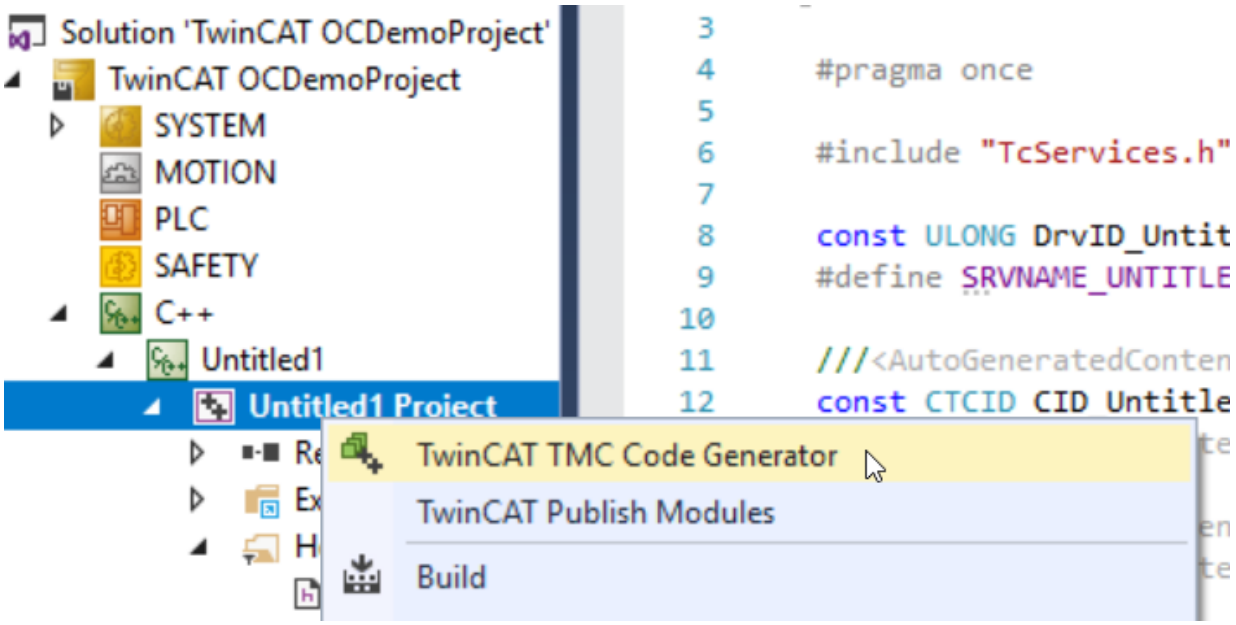
Diese Schritte sind nur notwendig, falls Sie zuvor das Modul auf den Online-Change vorbereitet haben.

In diesem Artikel wird beschrieben, wie das Beispielprojekt geändert werden kann, sodass eine Version 0.0.0.2 entsteht. Diese kann später auf dem Zielsystem per Online-Change ausgetauscht werden.

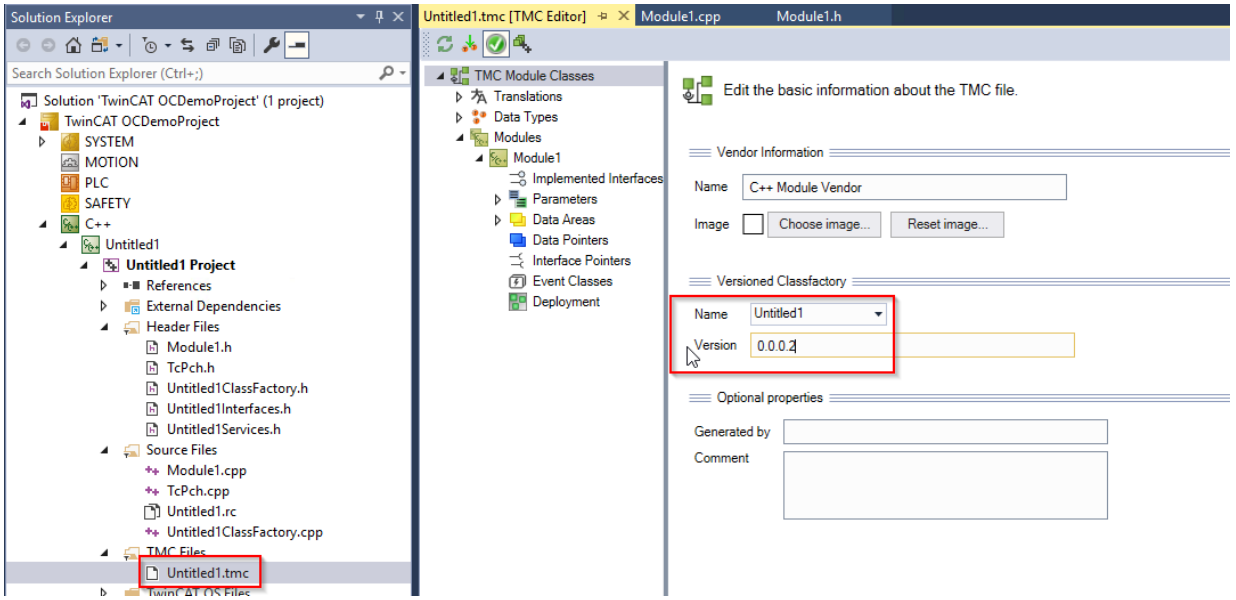
In der <MyClass>.cpp (in diesem Beispiel Module1.cpp) kann die Implementierung geändert werden.

1. Ersetzen Sie eine Zeile, um den Zähler zu dekrementieren statt zu inkrementieren.
   Diese Zeile
   `m_counter++;`
2. durch diese ersetzen
   `m_counter--;`
3. Speichern Sie die Änderungen.

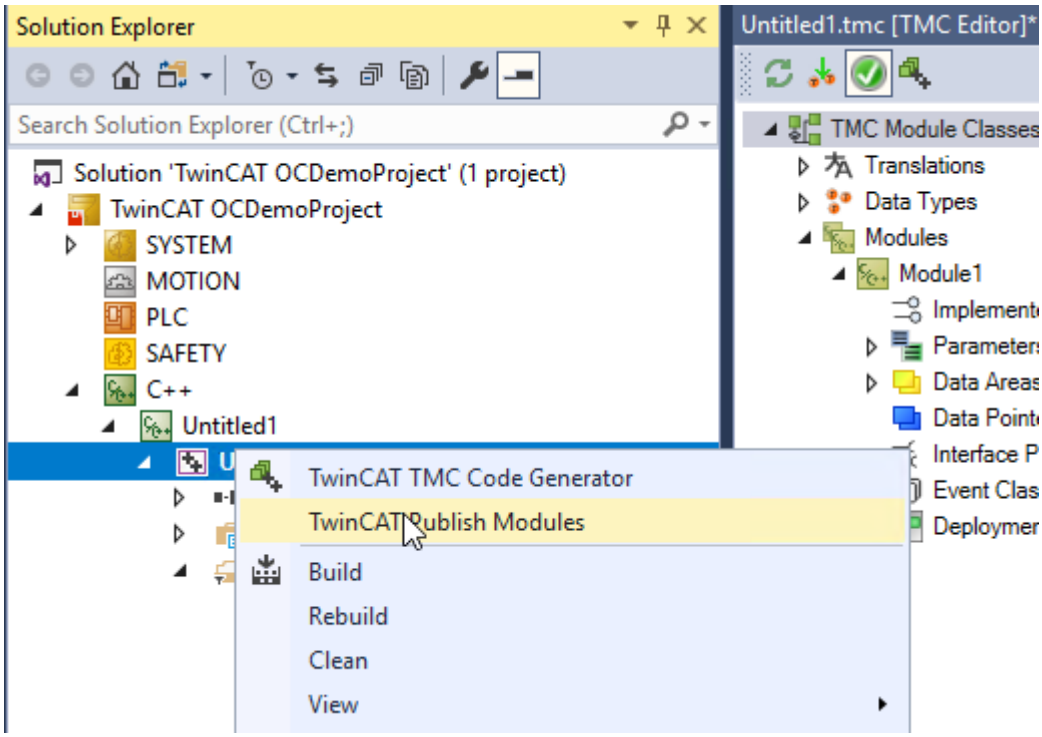
4. Starten Sie den Code Generator, um ggf. erfolgte Änderungen zu übernehmen.



5. Parametrieren Sie die Version 0.0.0.2 im TMC Editor.



6. Veröffentlichen Sie auch diese Version:



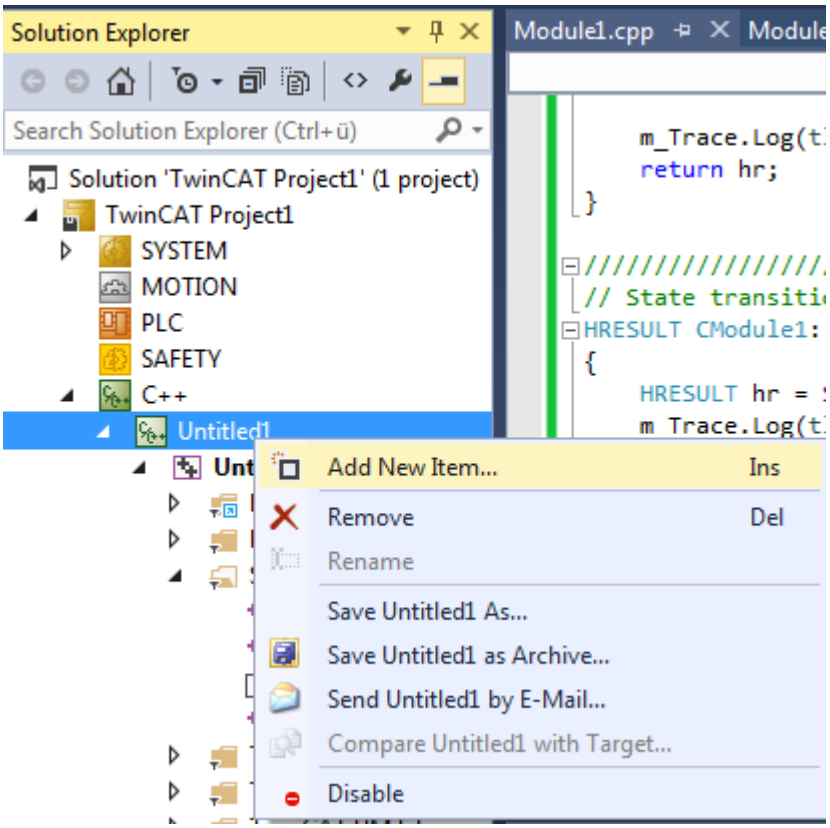
⇨ Es liegen zwei Versionen eines Moduls vor, welche während der Laufzeit ausgetauscht werden können.

9.7 TwinCAT 3 C++ Modulinstanz erstellen

Um es auszuführen, muss eine Instanz des Moduls erstellt werden. Es können mehrere Instanzen eines Moduls existieren.

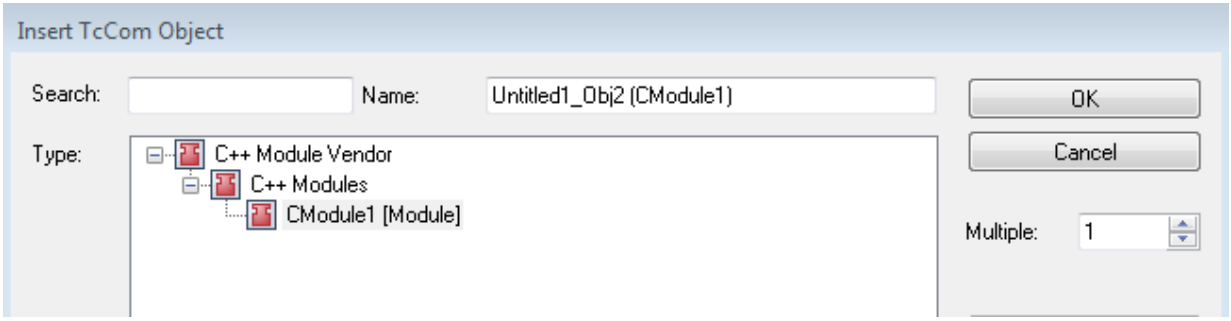
Nach Erstellen eines TwinCAT C++ Moduls öffnen Sie den Knoten **C++** und folgen dann folgenden Schritten, um eine Instanz zu erstellen.

1. Machen Sie einen Rechtsklick auf das C++ Modul (in diesem Fall „Untitled1“) und wählen Sie **Add New Item....**



⇨ Alle bestehenden C++ Module werden aufgelistet.

2. Wählen Sie ein C++ Modul aus. Sie können die Standardbezeichnung verwenden oder optional einen neuen Instanzenname eingeben und mit **OK** bestätigen (in diesem Beispiel wurde die Standardbezeichnung gewählt).



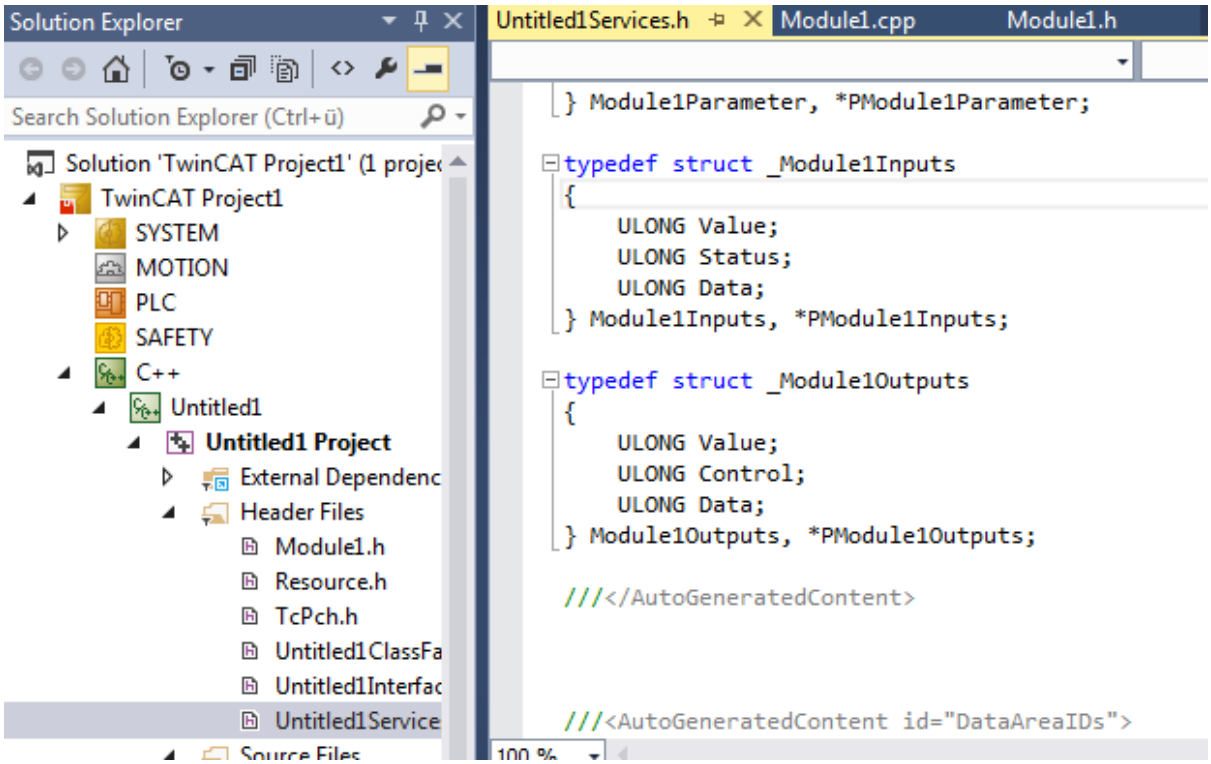
⇨ Die neue Instanz „Untitled1_Obj2 (CModule1)“ wird Teil der TwinCAT 3 Solution: Der neue Knoten findet sich genau unter der TwinCAT 3-C++ Quelle „Untitled1 Project“.

Das Modul stellt bereits eine einfache I/O-Schnittstelle mit je 3 Variablen zur Verfügung:

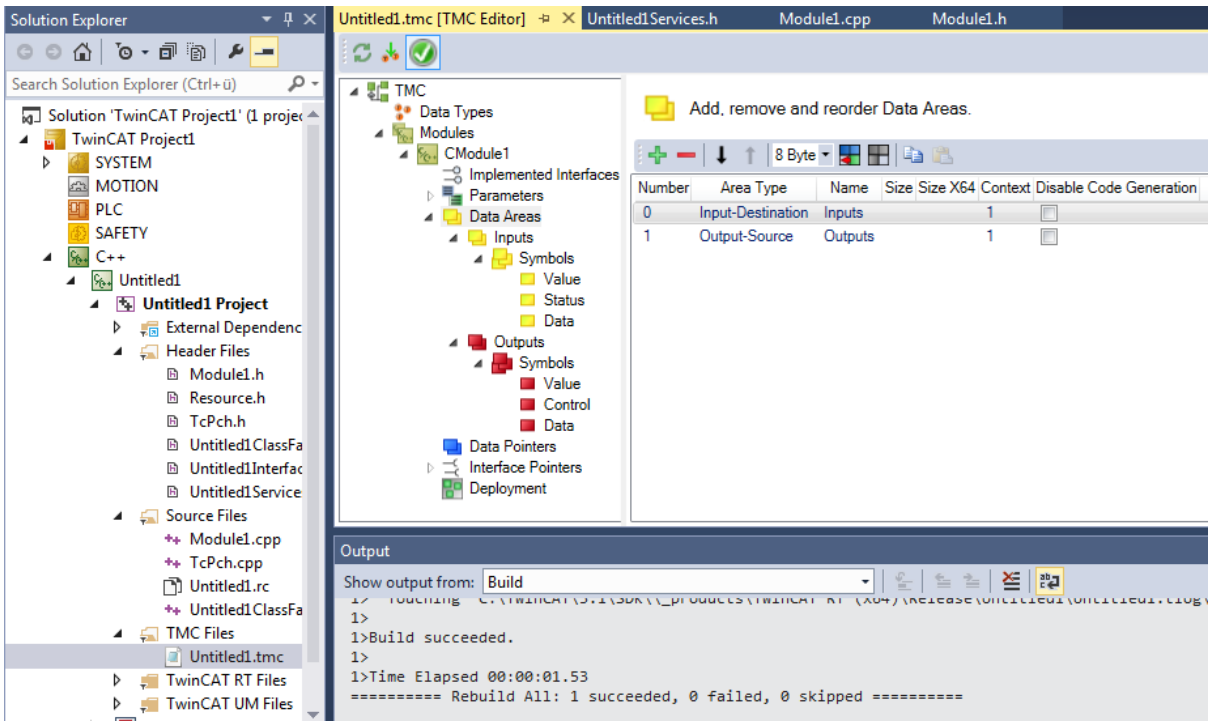
- Input-Area: Value, Status, Data
- Output-Area: Value, Control, Data

Die Beschreibung dieser Schnittstellen entspricht einander an zwei Stellen:

- „<Classname>Services.h“ (in diesem Beispiel „Untitled1Services.h“)



- „TwinCAT Module Configuration“ .tmc-Datei (in diesem Beispiel „Untitled1.tmc“)

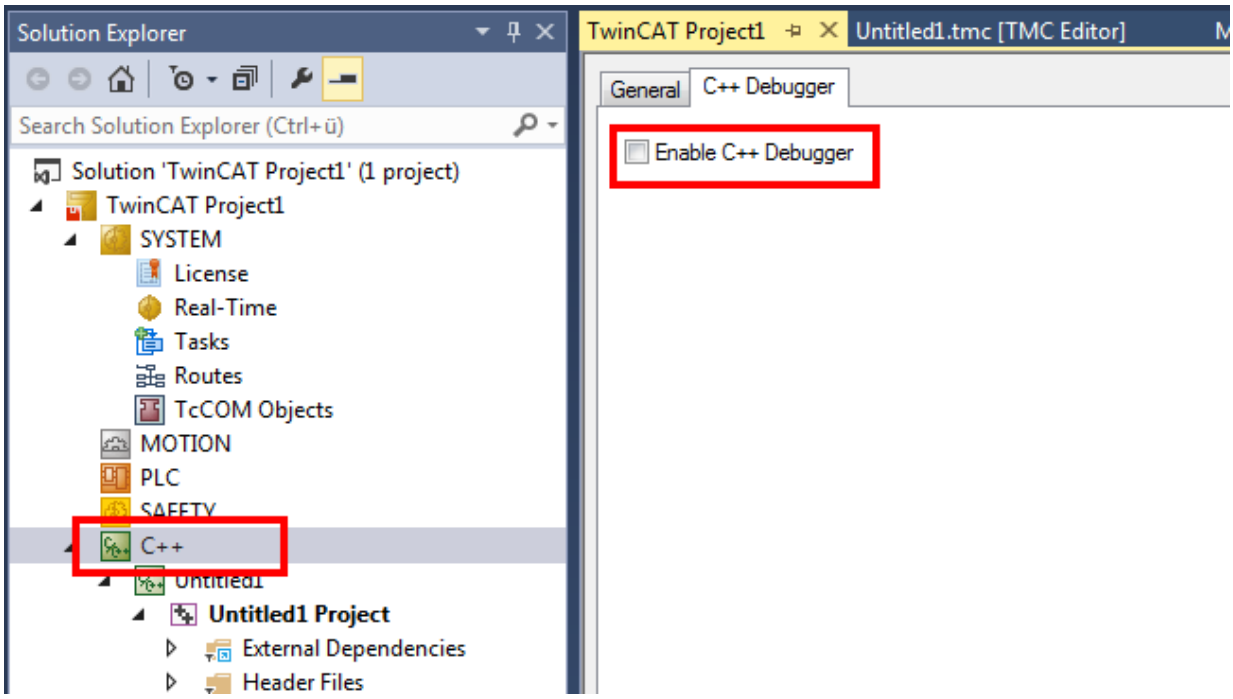


9.8 TwinCAT 3 C++ Debugger aktivieren

Um nicht immer alle Abhängigkeiten zum Debugging [▶ 80] zu laden, ist diese Funktion standardmäßig ausgeschaltet und muss einmal vor der Aktivierung der Konfiguration aktiviert werden.

1. Wählen Sie auf dem C++ Knoten der Solution Registerkarte **C++ Debugger.**
2. Wählen Sie **Enable C++ Debugger** aus.

3. Schalten Sie **Enable C++ Debugger** an.



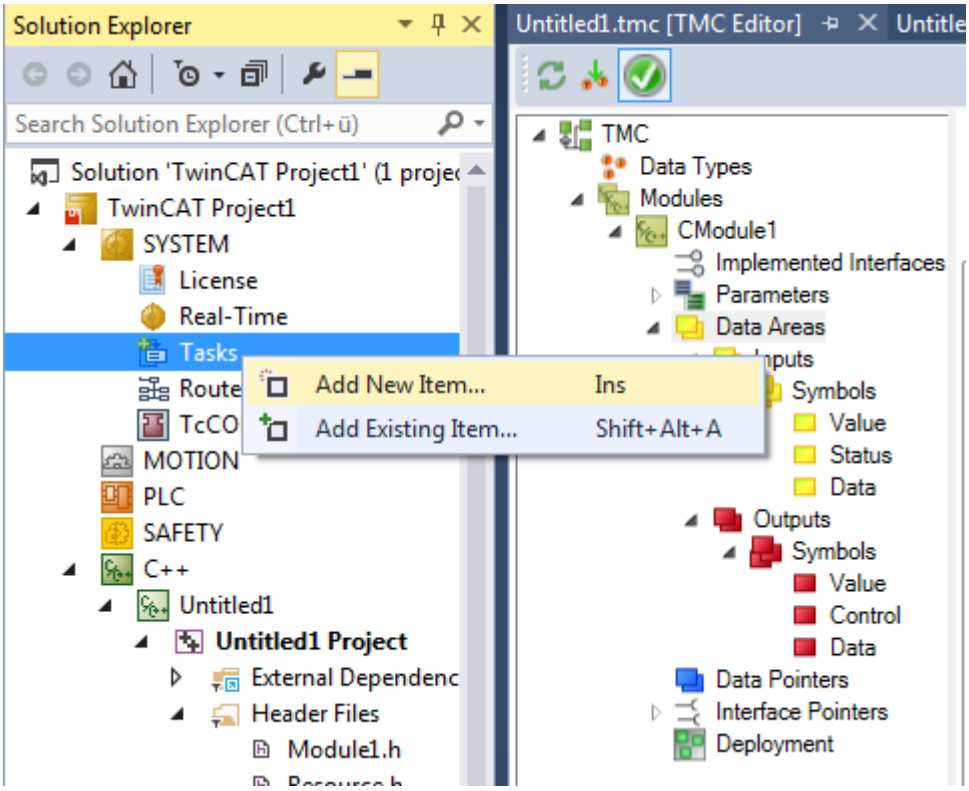
9.9 TwinCAT Task erstellen und auf Modulinstanz anwenden

Diese Seite beschreibt die Verknüpfung von Modulinstanz zu einer Task, sodass das zyklische Interface des Moduls von dem TwinCAT Echtzeitsystem aufgerufen wird.

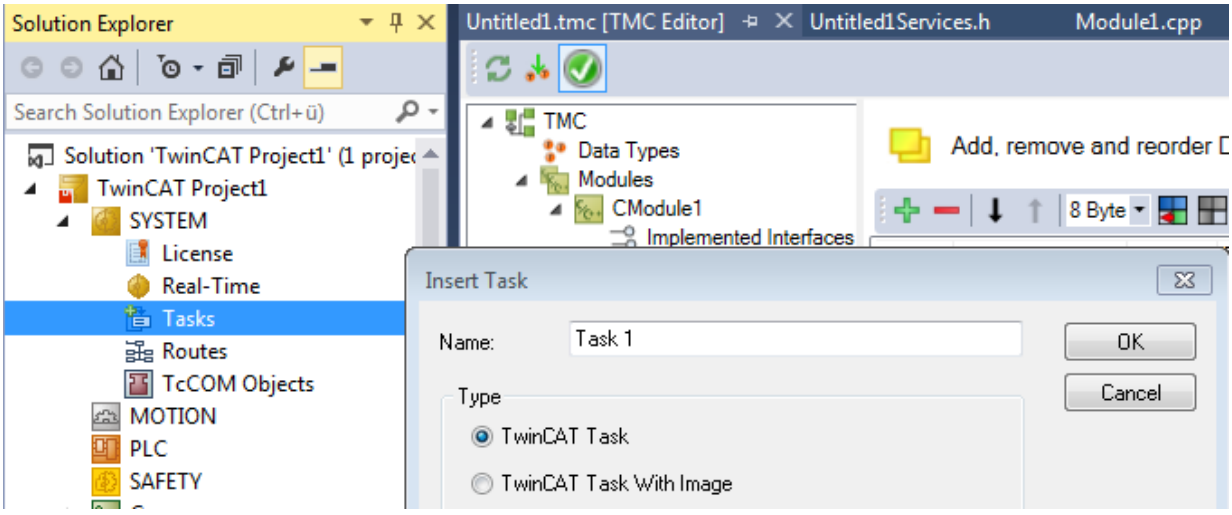
Dieser Konfigurationsschritt muss nur einmal ausgeführt werden. Für spätere Erstellungen/ Neukompilierungen des C++ Moduls im gleichen Projekt muss keine neue Task konfiguriert werden.

TwinCAT 3 Task erstellen

1. Öffnen Sie **System**, machen Sie einen Rechtsklick auf **Tasks** und wählen Sie **Add New Item...**.

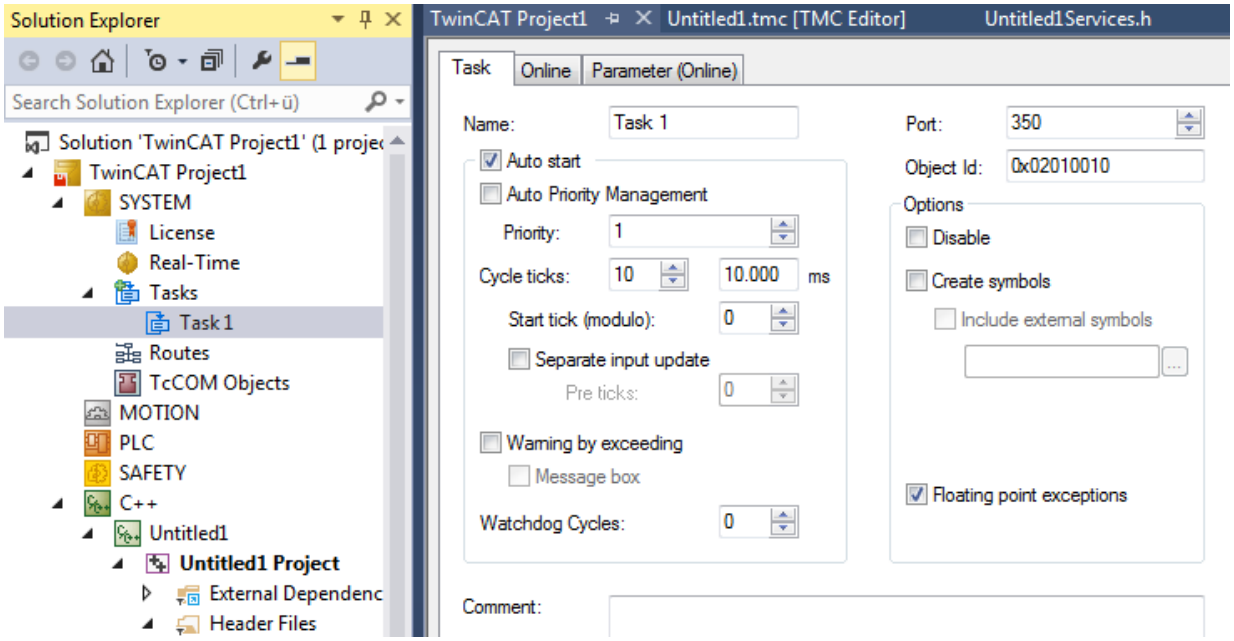


2. Geben Sie einen eindeutigen Namen für die Task ein (oder behalten Sie den Standard bei).
   In diesem Beispiel wird die I/O-Abbildschnittstelle durch eine C++ Modulinstanz bereitgestellt, so dass für das Auslösen der Ausführung der C++ Modulinstanz kein Abbild (Image) an der Task nötig ist.



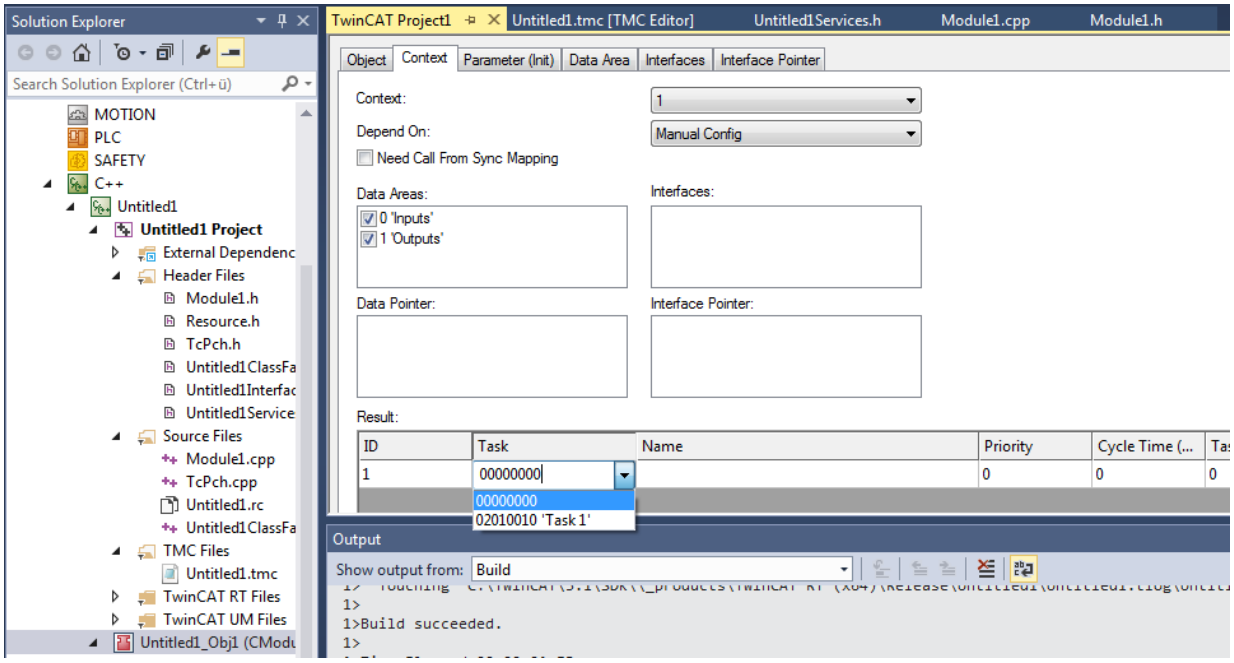
   ⇨ Die neue Task mit Namen „Task 1“ ist angelegt.

3. Nun kann die Task konfiguriert werden, hierfür machen Sie einen Doppelklick auf die Task.
   Die wichtigsten Parameter sind **Auto start** und **Priority**:
   Auto start muss aktiviert werden, um eine zyklisch auszuführende Task automatisch zu starten. Die **Cycle ticks** legen die Taktung des Taktes in Abhängigkeit vom Basistakt (siehe Real-Time Einstellungen) fest.



TwinCAT 3 C++ Modulinstanz konfigurieren, die von der Task aufgerufen wird

1. Wählen Sie die C++ Modulinstanz im Solution-Baum.
2. Wählen Sie im rechten Arbeitsbereich die Registerkarte **Context**.
3. Wählen Sie die Task für den zuvor erstellten Kontext im Dropdown-Taskmenü.
   Wählen Sie im Beispiel die Standard **Task 1**.

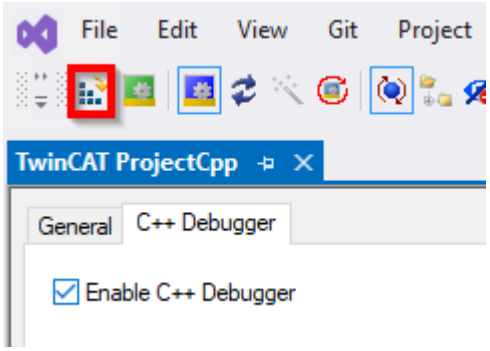


⇨ Mit Abschluss dieses Schritts ist der **Interface Pointer** als **CyclicCaller** konfiguriert. Die Konfiguration ist jetzt abgeschlossen.

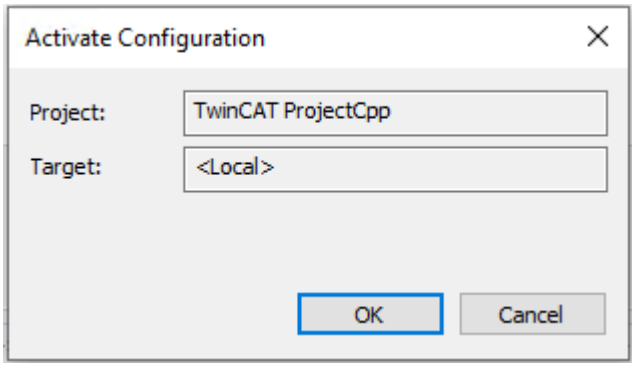
9.10 TwinCAT 3 Projekt aktivieren

Nachdem ein TwinCAT C++ Projekt erstellt, kompiliert und bereitgestellt wurde, muss die Konfiguration aktiviert werden:

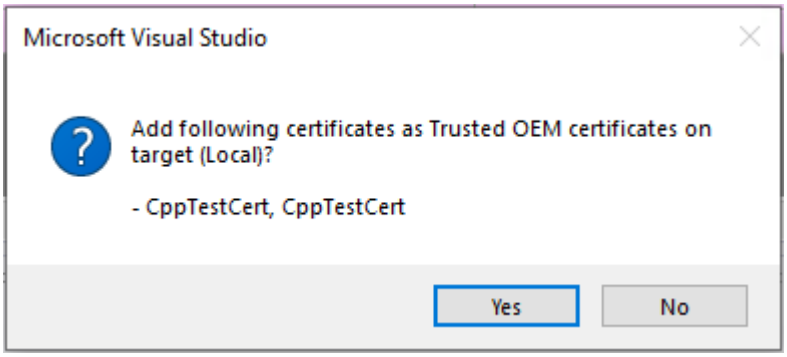
1. Klicken Sie auf das Symbol **Activate Configuration** – alle benötigten Dateien für das TwinCAT Projekt werden auf das Zielsystem übertragen:



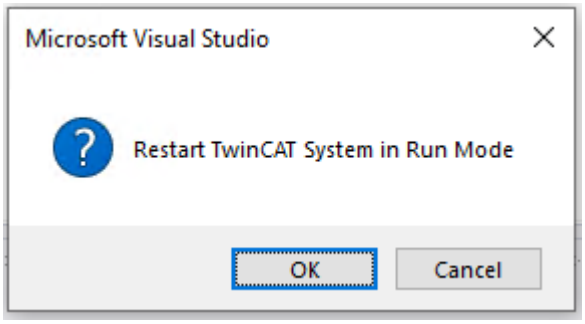
2. Bestätigen Sie im nächsten Schritt die Aktivierung der neuen Konfiguration. Die vorherige alte Konfiguration wird überschrieben.



⇨ Die zur Ausführung benötigten Zertifikate werden angezeigt und können automatisch zugelassen werden:



3. Falls Sie keine Lizenz auf dem Zielsystem haben, wird angeboten, eine Testlizenz für 7 Tage zu erstellen. Dieses kann beliebig häufig wiederholt werden.
4. TwinCAT 3 fragt automatisch, ob in den Run-Modus gewechselt werden soll.



⇨ Bei **OK** wechselt das TwinCAT 3 Projekt in den Run-Modus.
Bei **Cancel** bleibt TwinCAT 3 im **Config-Modus**.

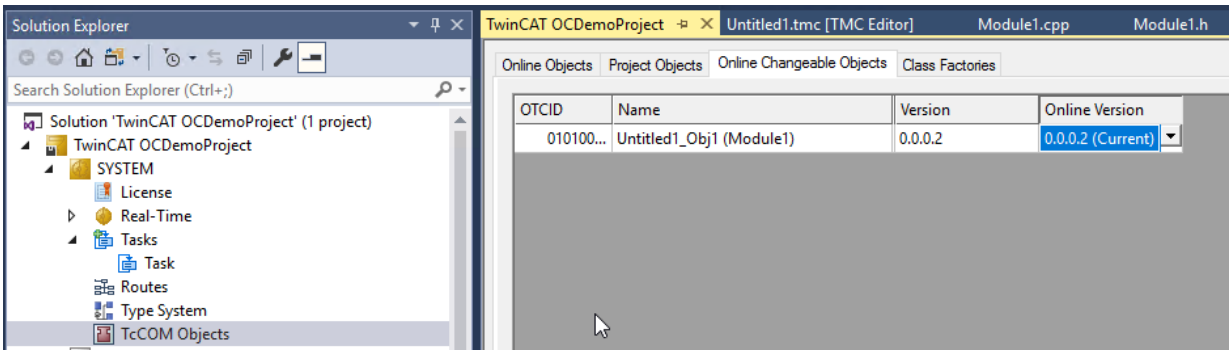
⇨ Nach dem Wechsel in den Run-Modus, leuchtet das TwinCAT System Service Symbol unten im Visual Studio grün.

9.11 TwinCAT 3 C++-Projekt Online-Change durchführen

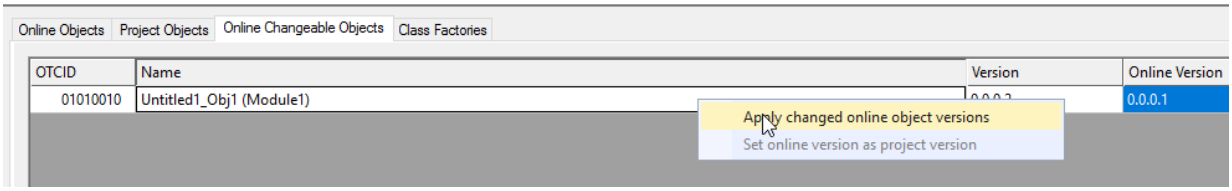
Diese Schritte sind nur notwendig, falls Sie zuvor das Modul auf den Online-Change vorbereitet haben.

✓ Sie haben ein laufendes TwinCAT C++-Projekt, wie zuvor in TwinCAT Projekt aktivieren [▶ 77] beschrieben.

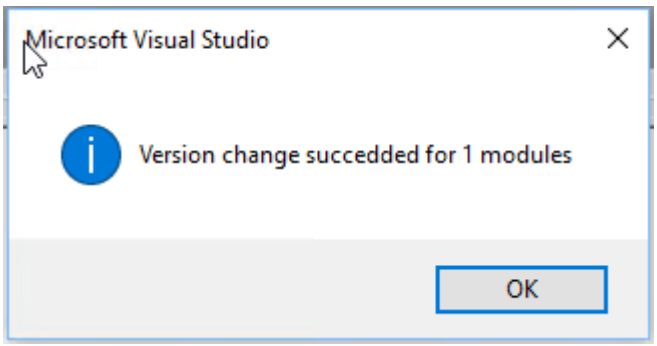
1. Wechseln Sie in die **TcCOM Objects**-Übersicht des **SYSTEM** Bereiches und dort in den Tab **Online Changeable Objects**.



2. In der Spalte **Online Version** ist die aktuell laufende Version vorausgewählt sowie mit der Erweiterung (Current) gekennzeichnet.
3. Stellen Sie eine andere Version ein.
4. Aktivieren Sie diese Änderung durch Rechtsklick und **Apply changed online object versions** auf dem Ziel.



⇨ Der Versions-Wechsel wurde auf dem Ziel vorgenommen



10 Debuggen

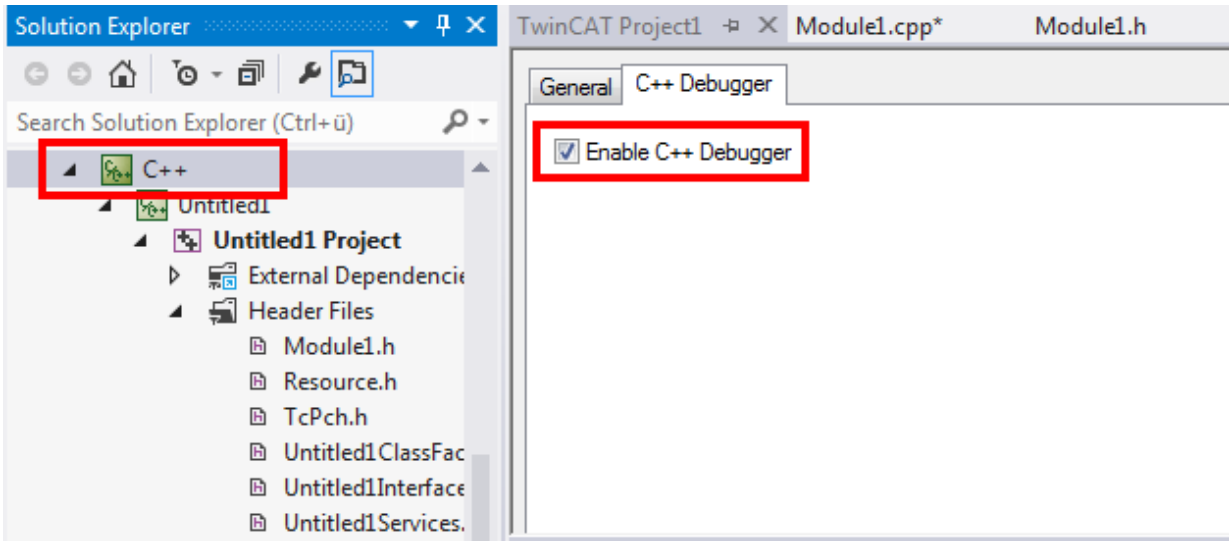
TwinCAT C++ bietet verschiedene Mechanismen für das Debuggen der unter Echtzeitbedingungen laufenden TwinCAT C++ Module.
Die meisten von ihnen entsprechen denjenigen, die von der normalen C++ Entwicklungsumgebung bekannt sind. Die Automatisierungswelt benötigt zusätzliche, leicht abweichende Debugging-Mechanismen, die hier dokumentiert sind.

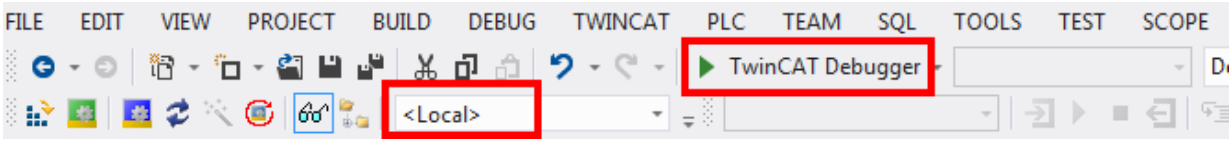
Zusätzlich gibt es eine Übersicht der Visual Studio Werkzeuge, welche in TwinCAT 3 genutzt werden können. Diese wurden erweitert, sodass Daten aus dem Zielsystem angezeigt werden.

Das Debugging muss freigegeben sein.

Dies konfigurieren Sie über den C++ Knoten der Solution:

1. Doppelklicken Sie auf den C++ Knoten und wechseln Sie zum **C++ Debugger**-Reiter, um auf das Prüfkästchen zuzugreifen.



2. Für jegliches Debuggen in TwinCAT C++ verbinden Sie das TwinCAT Engineering mit dem Laufzeitsystem (XAR) über die **TwinCAT Debugger**-Schaltfläche.
3. 

Haltepunkte und schrittweise Ausführung

In den meisten Fällen werden beim Debuggen eines C++ Programms Haltepunkte festgelegt und dann der Code schrittweise abgearbeitet, wobei die Variablen, Zeiger, usw. beobachtet werden.

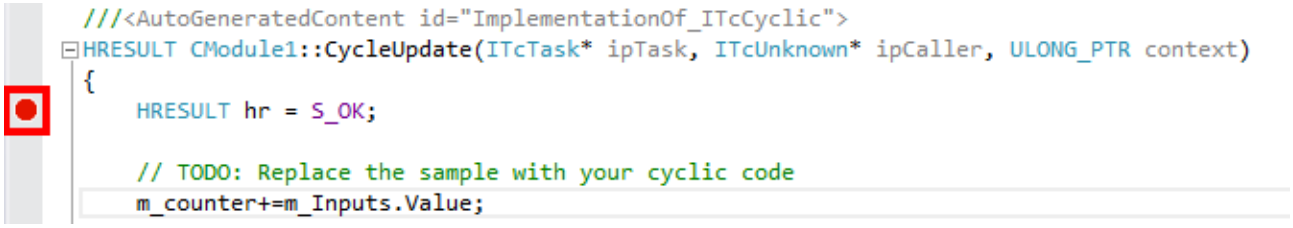
TwinCAT bietet im Rahmen der Visual Studio Debugging Umgebung Möglichkeiten, um einen in Echtzeit ausgeführten Code schrittweise abzuarbeiten. Zum Festlegen eines Haltepunkts kann man durch den Code navigieren und entweder auf die graue Spalte links neben dem Code klicken oder den Hotkey (normalerweise F9) benutzen.

⚠ WARNUNG

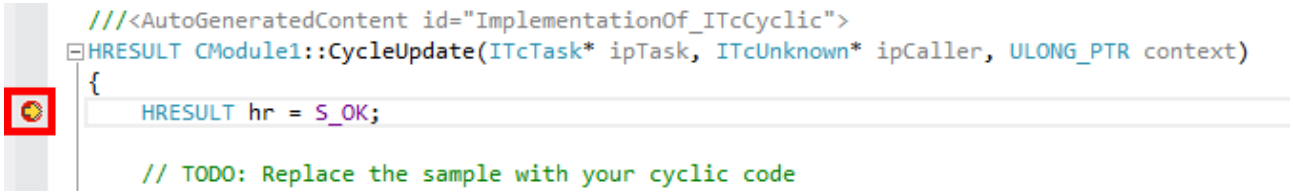
Anlagen- und Personenschäden durch unerwartetes Verhalten der Maschine / Anlage

Haltepunkte verändern das Verhalten der Maschine bzw. Anlage. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

Stellen Sie sicher, dass das veränderte Verhalten des gesteuerten Systems keine Schäden verursacht und beachten Sie unbedingt die Anlagendokumentation.



Beim Erreichen des Haltepunkts (wird mit Pfeil angezeigt) wird die Ausführung des Codes angehalten.



Durch Drücken auf **Step Over** (Debug-Menü, Toolbar oder Hotkey F10) wird der Code schrittweise ausgeführt. Auch stehen die von Visual Studio bekannten **Step in** (F11) und **Step out** (Shift+F11) zur Verfügung.

Bedingte Haltepunkte

Eine fortschrittlichere Technologie erlaubt das Setzen von bedingten Haltepunkten - die Ausführung des Codes auf Höhe eines Haltepunktes wird nur dann angehalten, wenn eine Bedingung erfüllt ist.

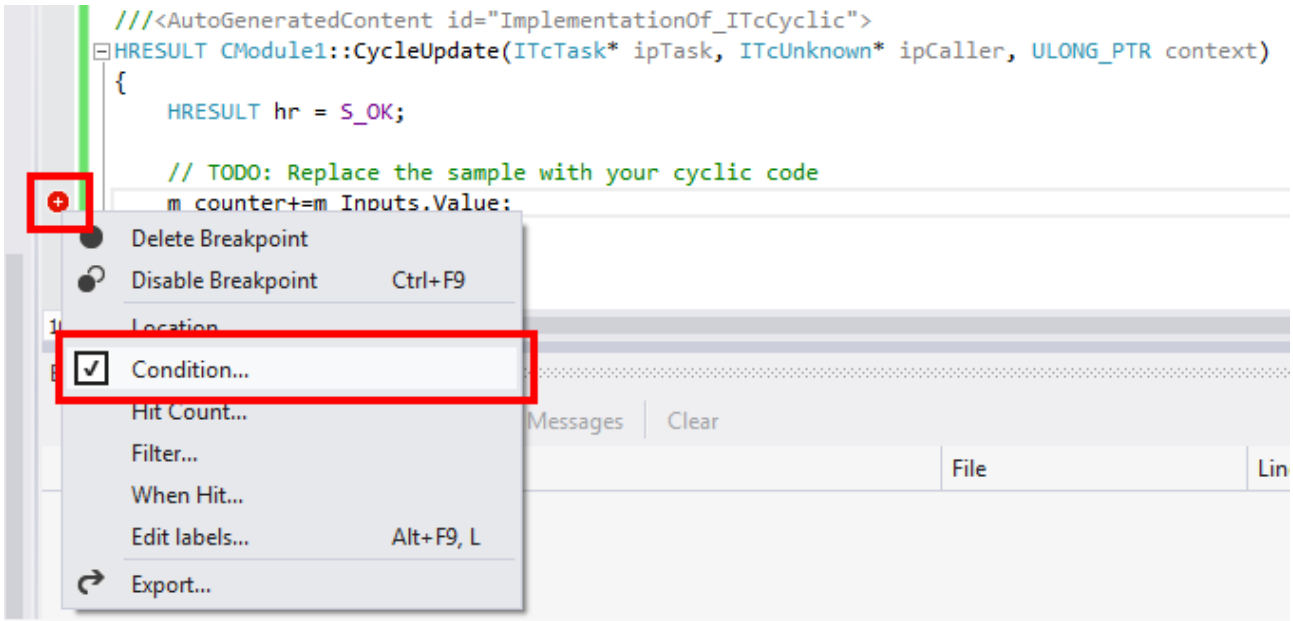
TwinCAT bietet die Implementierung eines bedingten Haltepunktes als Teil der Visual Studio Integration. Zum Festlegen einer Bedingung, setzen Sie zunächst einen normalen Haltepunkt und klicken anschließend mit der rechten Maustaste auf den roten Punkt in der Haltepunkt-Spalte.

⚠ WARNUNG

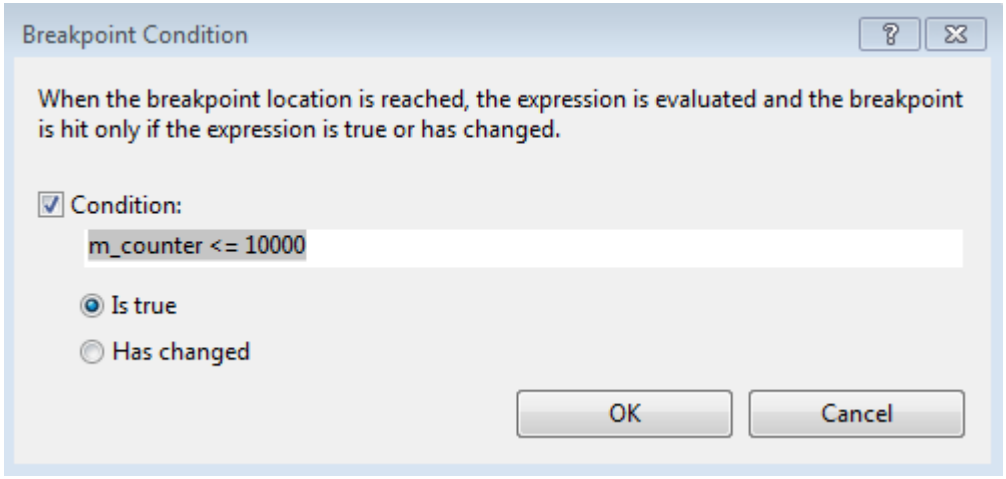
Anlagen- und Personenschäden durch unerwartetes Verhalten der Maschine / Anlage

Haltepunkte verändern das Verhalten der Maschine bzw. Anlage. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

Stellen Sie sicher, dass das veränderte Verhalten des gesteuerten Systems keine Schäden verursacht und beachten Sie unbedingt die Anlagendokumentation.



Wählen Sie **Condition…** um das Bedingungsfenster zu öffnen:



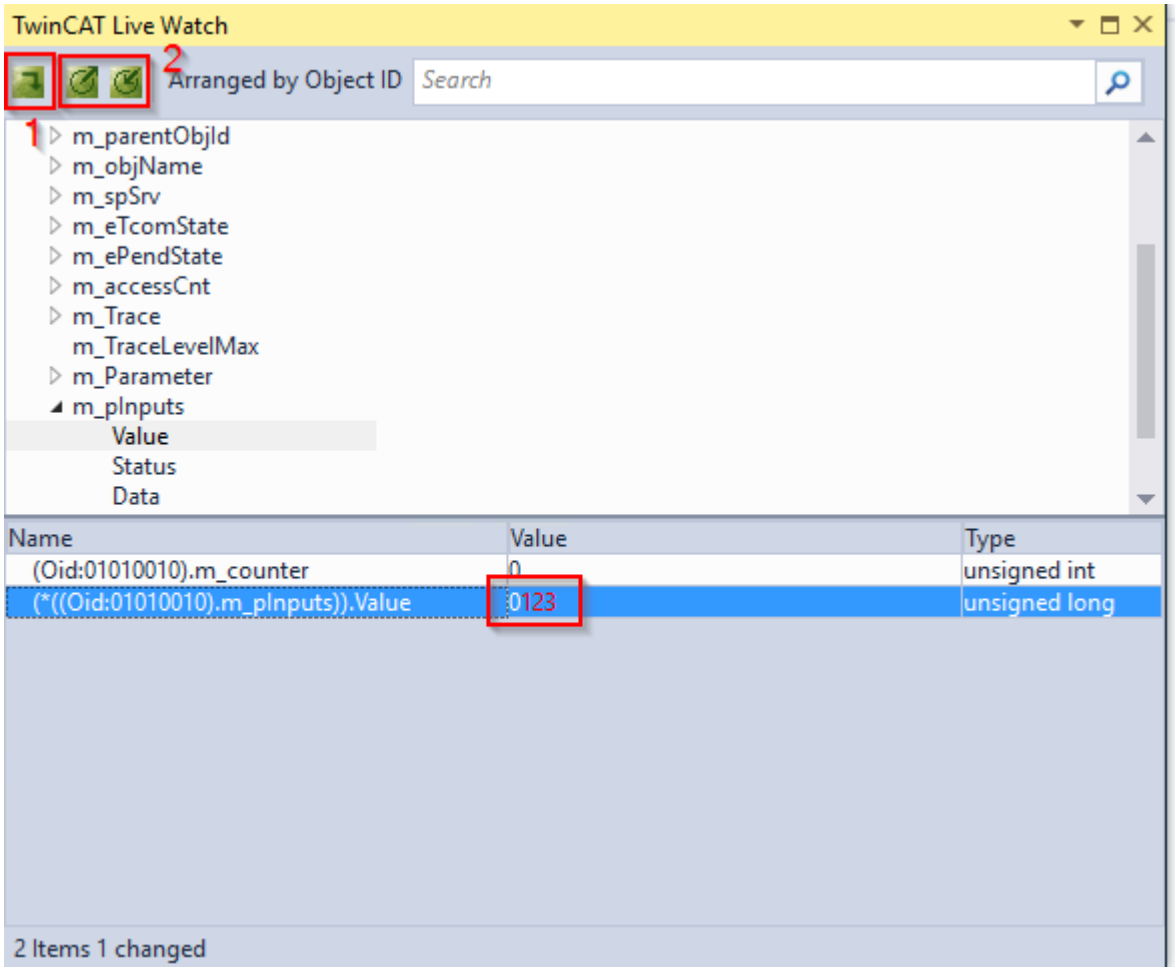
Näheres dazu im Kapitel: Einzelheiten zu den bedingten Haltepunkten [▶ 83].

Live Watch

Bei dem Engineering und der Entwicklung von Maschinen ist es nicht immer ratsam, das System über einen Haltepunkt anzuhalten, weil dies sich auf das Verhalten auswirken wird.
Die TwinCAT-SPS-Projekte bieten eine online-Ansicht und -Handhabung der Variablen im RUN-Zustand, ohne die Echtzeit unterbrechen zu müssen.

TwinCAT C++ Projekte bieten ein ähnliches Verhalten für C++ Code über das **Live Watch**-Fenster.

Das **Live Watch**-Fenster kann über **Debug->Windows->TwinCAT Live Watch** geöffnet werden.
Um das Fenster zu öffnen, stellen Sie zunächst eine Verbindung mit dem Echtzeitsystem her (drücken der **TwinCAT Debugger**-Schaltfläche), woraufhin Visual Studio zur Debug-Ansicht wechselt, ansonsten können keine Daten bereitgestellt werden.



Das TwinCAT Live Watch Fenster ist in zwei Bereiche unterteilt.

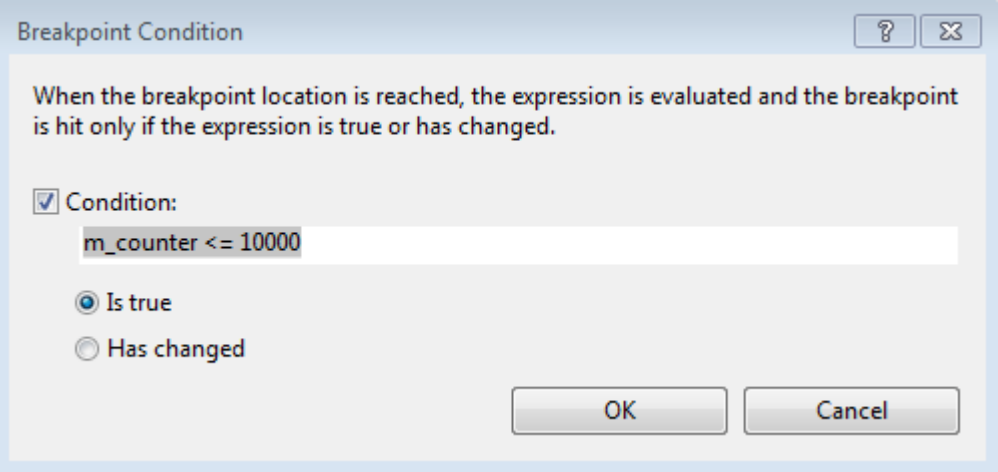
Im oberen Bereich können alle Member-Variablen erkundet werden. Durch Doppelklick auf diesen, werden sie dem unteren Bereich hinzugefügt, wo dann der aktuelle Wert angezeigt wird.

Sie können diese Werte durch Klicken auf den Wert im **Value**-Feld bearbeiten. Der neue Wert wird rot markiert. Um den Wert nun zu schreiben, drücken Sie auf das Symbol oben links (1).

Mithilfe der Import- und Export-Symbole unter (2) können die selektierten Member-Variablen abgespeichert und später wiederhergestellt werden.

10.1 Einzelheiten zu den bedingten Haltepunkten

TwinCAT C++ stellt bedingte Haltepunkte zur Verfügung. Einzelheiten zur Formulierung dieser Bedingungen finden Sie hier.



Im Gegensatz zu den Visual Studio C++ bedingten Haltepunkten werden die TwinCAT-Bedingungen kompiliert und anschließend auf das Zielsystem übertragen, sodass sie während kurzen Zyklenzeiten verwendet werden können.

⚠ WARNUNG

Anlagen- und Personenschäden durch unerwartetes Verhalten der Maschine / Anlage

Haltepunkte verändern das Verhalten der Maschine bzw. Anlage. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

Stellen Sie sicher, dass das veränderte Verhalten des gesteuerten Systems keine Schäden verursacht und beachten Sie unbedingt die Anlagendokumentation.

Die Optionsschaltflächen bieten zwei Optionen, die getrennt voneinander beschrieben werden.

Option: Is true

Bedingungen werden mit Hilfe von logischen Termen, vergleichbar mit den konjunktiven Normalformen definiert.
Sie werden aus einer Kombination von mit „&&“ verbundenen Maxtermen gebildet:

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

wobei jeder Maxterm eine Kombination von || verbundenen Bedingungen darstellt:

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

Mögliche Vergleichsoperatoren: `==, !=, <=, >=, <, >`

Für die Bestimmung der verfügbaren Variablen beachten Sie das Live Watch Fenster. Alle aufgeführten Variablen können für die Formulierung von Bedingungen herangezogen werden. Dazu gehören sowohl TMC-definierte Symbole, als auch lokale Member-Variablen.

Beispiele:

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

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

```
m_counter == 123
```

Überwachung von Modul-Instanzen

Die OID des Objekts ist in `m_objId` gespeichert, somit kann z. B. die Überwachung des OID folgendermaßen aussehen `m_objId == 0x01010010`

Überwachung von Tasks

Es wird eine spezielle Variable `#taskId` bereitgestellt, um auf die OID der aufrufenden Task zugreifen zu können. Z.B. `#taskID == 0x02010010`

Option: Has changed

Die Option „Has changed“ ist einfach zu verstehen: Indem Variablennamen bereitgestellt werden, wird der Wert überwacht und die Ausführung angehalten, wenn der Wert sich gegenüber dem vorangegangenen Zyklus geändert hat.

Beispiele:

```
m_counter
```

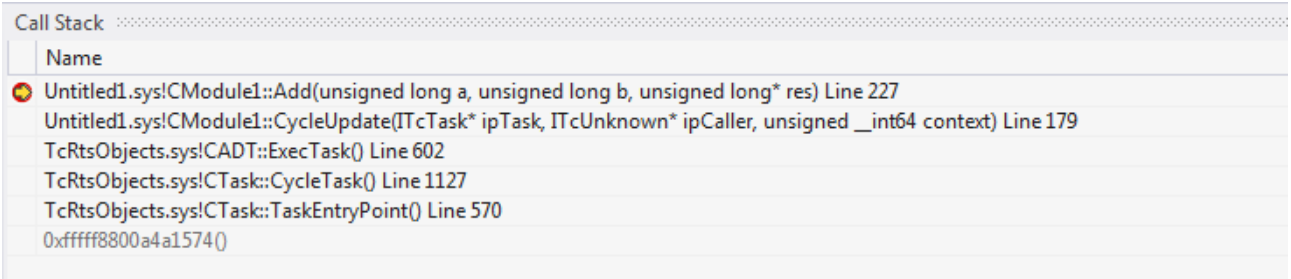
```
m_counter && m_counter2
```

10.2 Visual Studio Werkzeuge

Visual Studio stellt einem C++ Entwickler übliche Werkzeuge zur Entwicklung und Debugging bereit. TwinCAT 3 erweitert diese Visual Studio Werkzeuge, sodass das Debugging von C++ Code, der auf einem Zielsystem ausgeführt wird, im TwinCAT 3 Engineering auch mit den Visual Studio Werkzeugen möglich ist. Die entsprechend erweiterten Werkzeuge werden hier kurz beschrieben. Wenn die entsprechenden Fenster im Visual Studio nicht sichtbar sind, können diese über das Menü **Debug->Windows** hinzugefügt werden. Das Menü ist dabei Kontext-abhängig, d. h. viele der hier beschriebenen Fenster sind erst konfigurierbar, wenn ein Debugger auch mit einem Zielsystem verbunden ist.

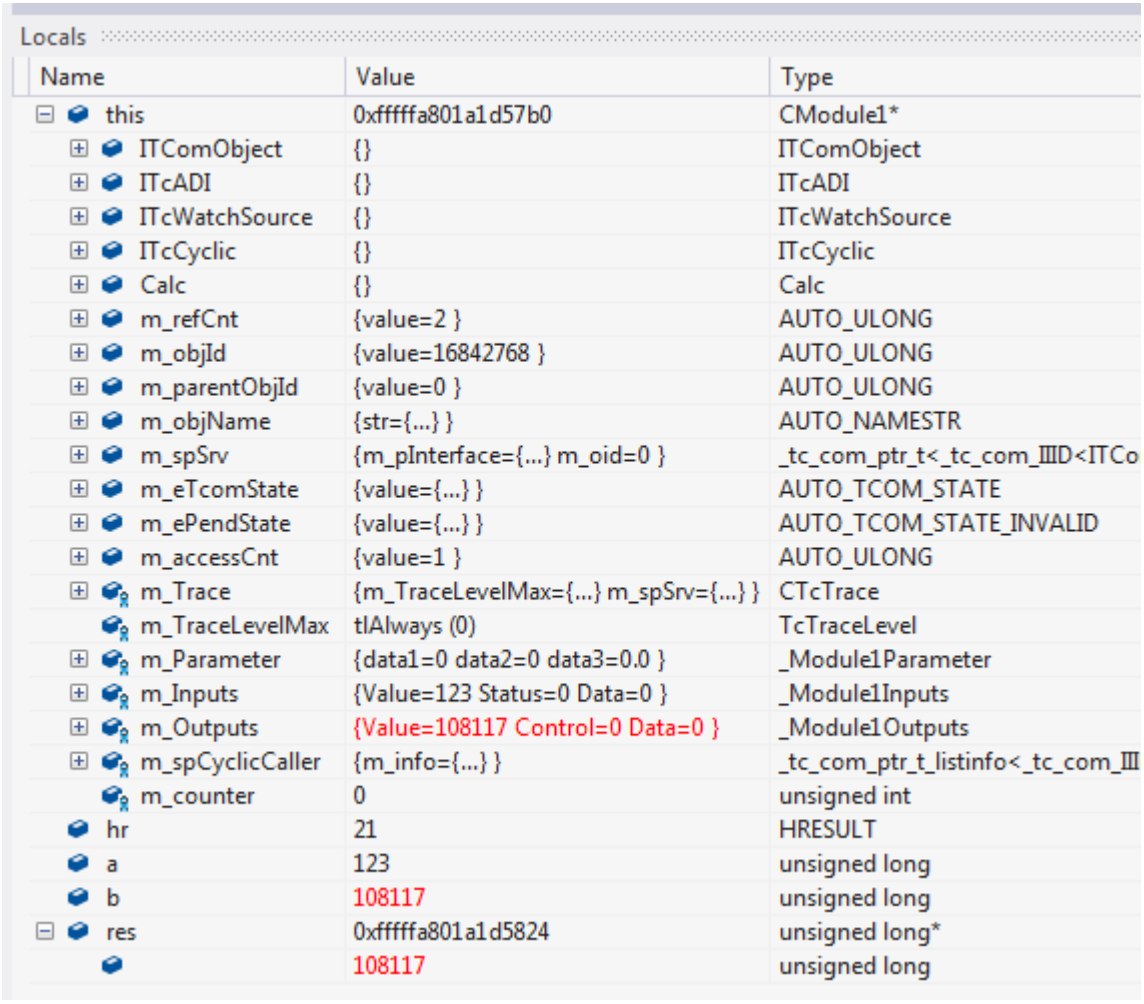
Callstack

Der Callstack wird durch das Toolwindow **Call Stack** dargestellt, wenn ein Haltepunkt erreicht wurde.

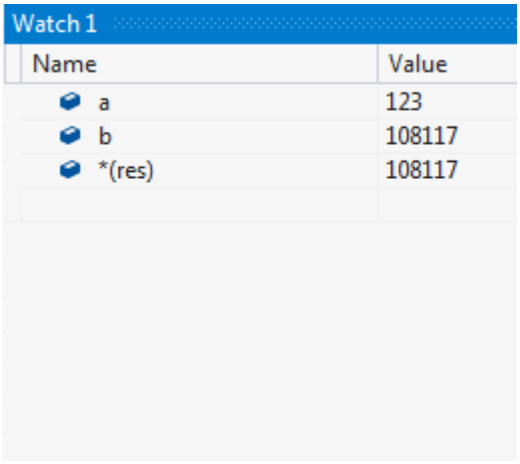


Autos / Locals und Watch

Die entsprechenden Variablen und Werte werden im **Autos** / **Locals** Fenster angezeigt, wenn ein Haltepunkt erreicht ist. Änderungen werden dabei in ROT gekennzeichnet.

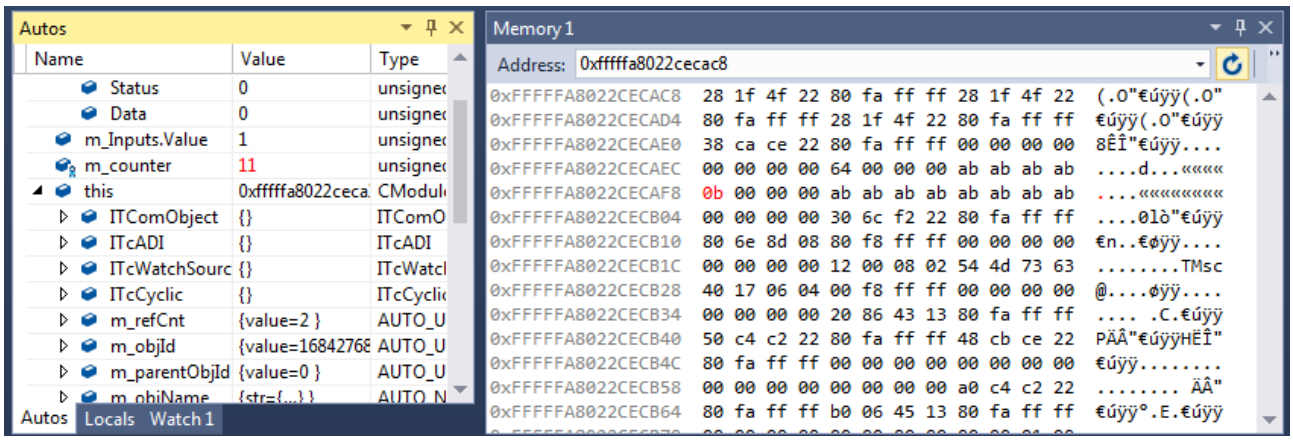


Von hier können per Rechts-Klick die Werte auch in die **Watch**-Fenster übernommen werden:



Memory View

Der Speicher kann auch direkt beobachtet werden. Änderungen werden dabei in ROT gekennzeichnet.



10.3 Arbeiten mit TwinCAT Task Dumps

Ab TwinCAT 3.1 Build 4026

Diese Funktion steht ab TwinCAT 3.1 Build 4026 bereit.

Die TwinCAT Runtime schreibt bei Auftreten einer Exception automatisch „TwinCAT Task Dumps“ in das Boot-Verzeichnis (üblicherweise C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot), wenn diese nicht von der Anwendung behandelt wird.

Diese Task Dumps können vom TwinCAT C++ Entwickler auf ein Entwicklungssystem mit Visual Studio für TwinCAT C++ Entwicklung übertragen und ausgewertet werden.

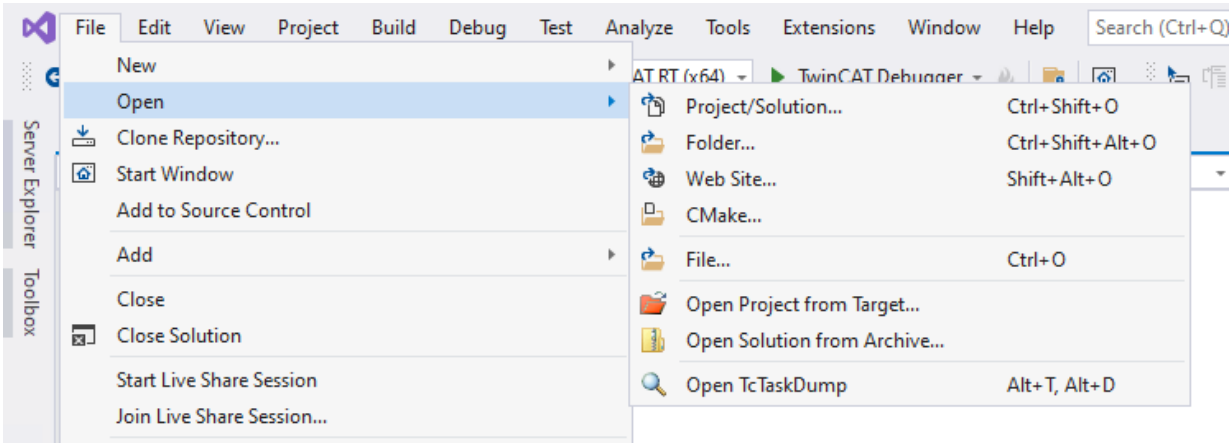
Für die Untersuchung ist es erforderlich, dass neben dem TwinCAT Task Dumps auch die exakt zugehörige TMX und PDB Datei vorliegt.

Wird der TwinCAT Task Dump direkt aus der passenden Solution geladen, werden diese Dateien automatisch gefunden. Wird der TwinCAT Task Dump aus einer leeren Solution geladen, können die zugehörigen TMX und PDB Dateien in dasselbe Verzeichnis des TwinCAT Task Dumps gelegt werden. Alternativ ist es möglich einen Suchpfad in den Visual Studio Optionen einzutragen (Tools->Options->Debugging->Symbols: add new Path to Symbol file (.pdb) locations).

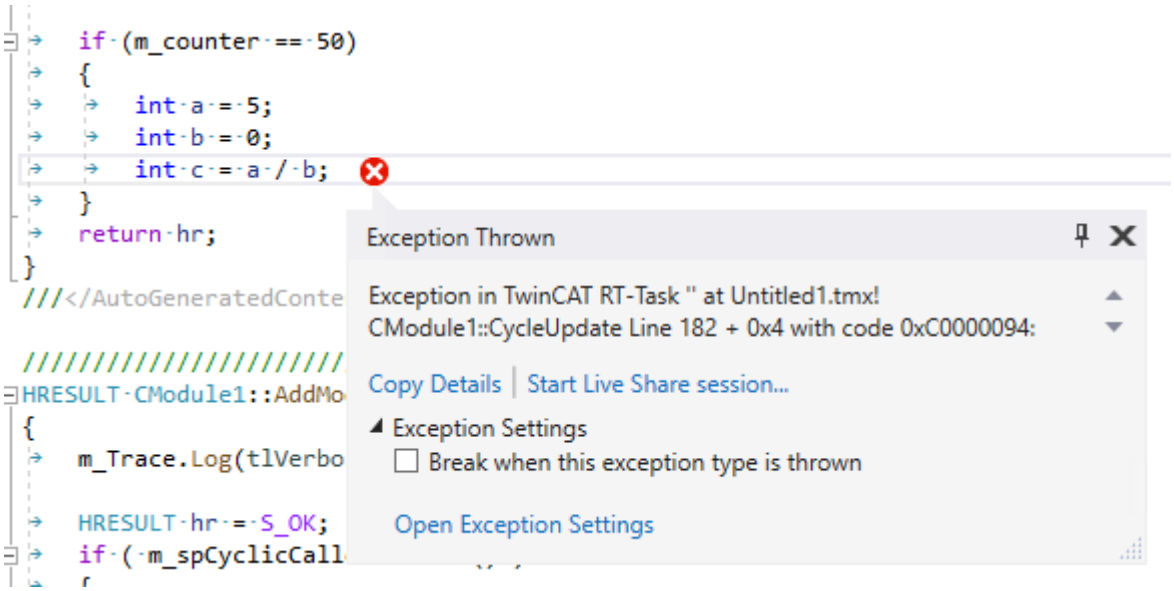
Der Ablauf ist dann wie folgt:

✓ Eine Exception ist auf dem Zielsystem aufgetreten.

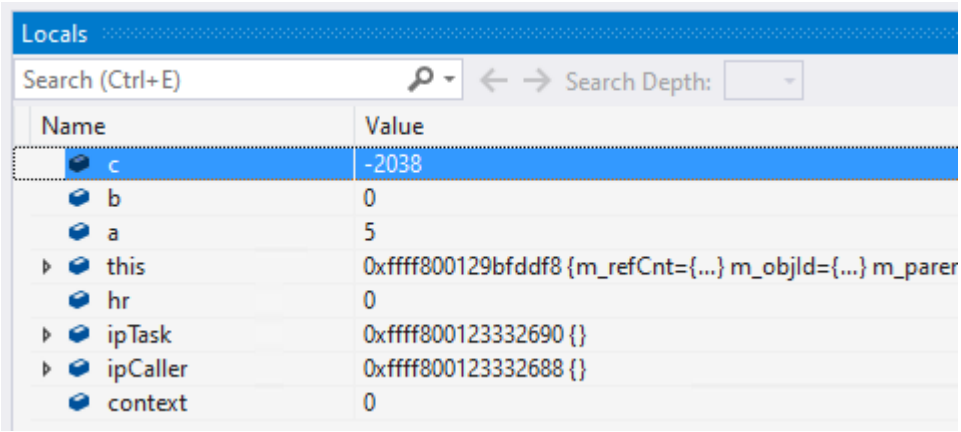
1. Die TwinCAT Task Dump Datei wird von dem Zielsystem auf das Engineering System übertragen. Sie trägt einen Namen wie taskmemory-*.dmp
2. Im Idealfall stehen die passenden Quellen des TwinCAT Projektes bereit und werden im Visual Studio geöffnet.
3. Über File->Open->Open TcTaskDump kann die Datei geöffnet werden:



4. Der Editor springt direkt an die passende Codestelle, die die Exception ausgelöst hat:



Zusätzlich können übliche Fenster im Visual Studio verwendet werden, um beispielsweise Variablen-Werte zu kontrollieren:



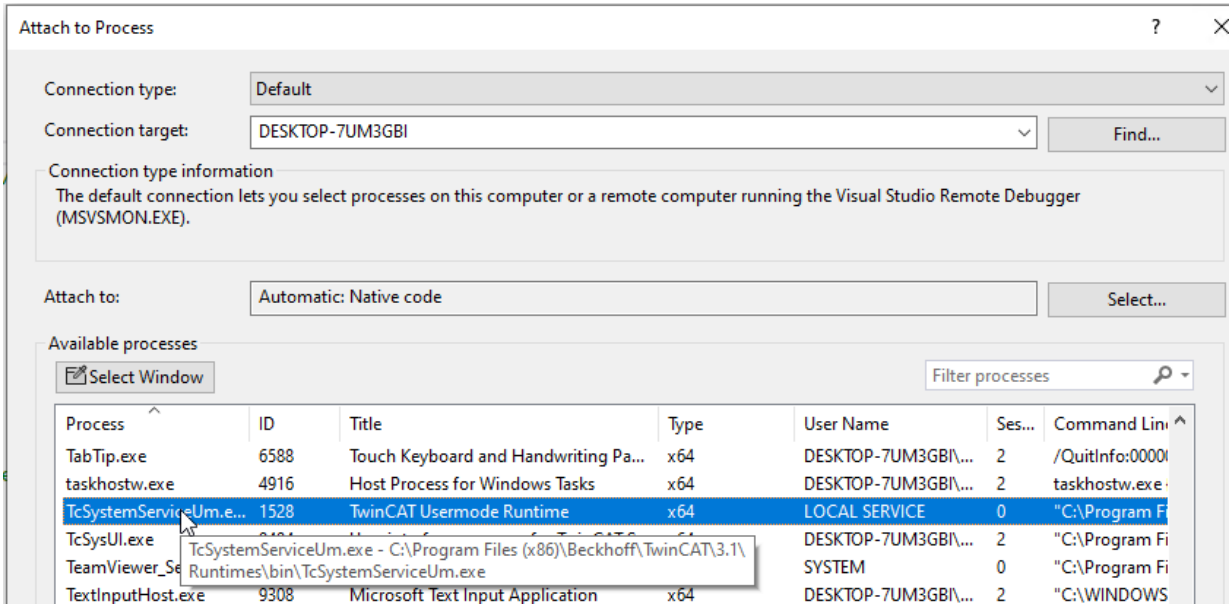
10.4 Debuggen der Statemachine mit der Usermode Runtime

Die Statemachine der TcCOM Module kann in der Echtzeit-Umgebung nicht debugged werden, da zu dem Zeitpunkt der Debugger selber noch gestartet wird.
Die Usermode Runtime kann dafür genutzt werden.

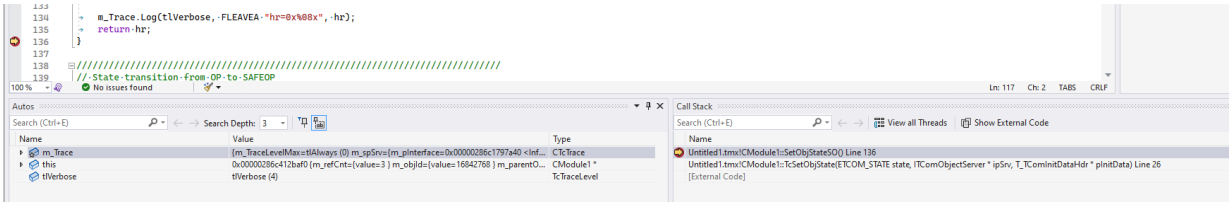
✓ Ein TwinCAT C++ Projekt und ist in Visual Studio geladen

1. Aktivieren Sie das Projekt wird in einer Usermode Runtime
2. Öffnen Sie eine zweite Visual Studio Instanz

3. Verbinden Sie sich per normalem Visual Studio C++ Debugger mit dem Prozess der Usermode Runtime. Debug -> Attach to Process … -> Auswahl des TcSystemServiceUm.exe



4. Fügen Sie in dieser Instanz das TwinCAT C++ Projekt hinzu, welches Sie debuggen möchten. Setzen Sie Breakpoints in den Transitionen.
5. Starten Sie die Usermode Runtime – die Breakpoints werden erreicht.



11 Assistenten

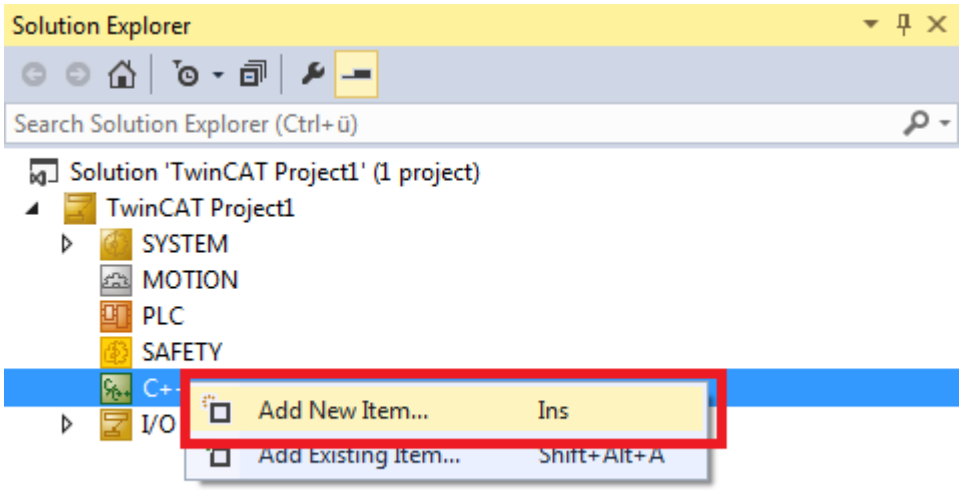
Um den Einstieg in das Engineering des TwinCAT C++-Systems zu vereinfachen, stehen Assistenten zur Verfügung.

- Der TwinCAT Projekt-Assistent [▶ 89] erstellt ein TwinCAT C++-Projekt. Im Falle von Treiber-Projekten wird anschließend der TwinCAT Class Wizard gestartet.
- Der TwinCAT Module Class Wizard [▶ 90] wird automatisch bei der Erstellung eines C++-Moduls gestartet.
  Dieser Assistent stellt verschiedene „gebrauchsfertige“ Projekte als Einstiegspunkt für eigene Entwicklungen zur Verfügung.
- Der TwinCAT Module Class Editor [▶ 92] (TMC) ist ein grafischer Editor für die Definition von Datenstrukturen, Parametern, Datenbereichen, Schnittstellen und Zeigern. Er erzeugt eine TMC-Datei, die vom TMC-Code-Generator verwendet wird.
- Anhand der definierten Klassen werden Instanzen generiert, die über den TwinCAT Module Instance Configurator [▶ 135] konfiguriert werden können.

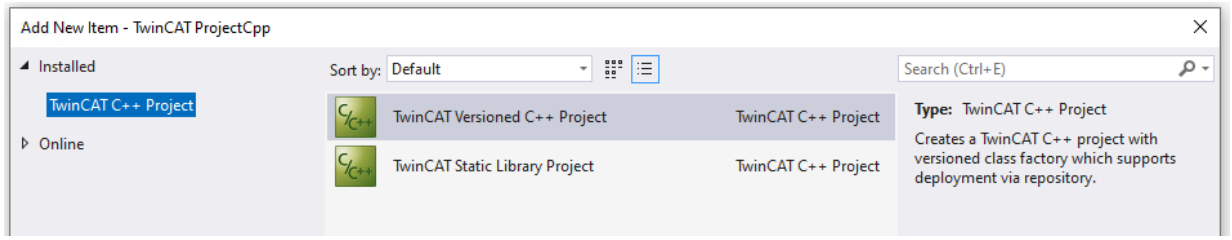
11.1 TwinCAT C++-Projekt-Assistent

Nach der Erstellung eines TwinCAT-Projekts, können Sie mit Hilfe des TwinCAT-C++-Projekt-Assistenten ein C++-Projekt hinzufügen:

1. Machen Sie einen Rechtsklick auf das C++-Symbol und wählen **Add new Item…**, um den C++-Projektassistenten zu starten.



TwinCAT bietet zwei C++-Projekte an:
- Versionierte C++-Projekte: Projekte, die eine Versionierung mit sich bringen und durch diese auch die Möglichkeit bekommen können, zur Laufzeit zwischen den Versionen zu wechseln.
- Statische Bibliothek: Projekte mit C++-Funktionen, die von (verschiedenen) TwinCAT-C++-Treibern verwendet werden.



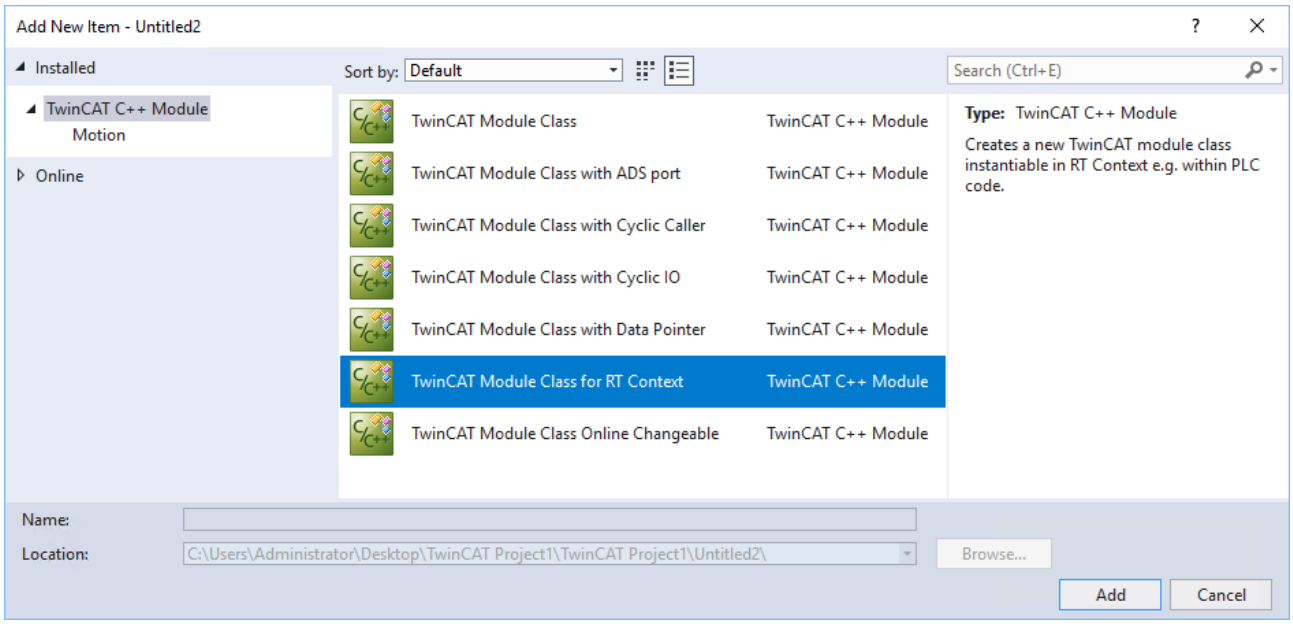
2. Wählen Sie eine der Projektvorlagen, geben Sie einen Namen und einen Speicherort an.
   ⇨ Das TwinCAT C++-Projekt wird erstellt

⇨ Im Falle eines Treibers wird der TwinCAT-C++-Klassenassistent [▶ 90] gestartet.

11.2 TwinCAT Module Klassenassistent

TwinCAT 3 bietet verschiedene Klassenvorlagen, welche verwendet werden können um TcCOM Objektklassen anzulegen:

- TwinCAT Module Class
- TwinCAT Module Class mit ADS Port
- TwinCAT Module Class mit zyklischem Aufrufer
- TwinCAT Module Class mit zyklischem Ein-/Ausgang
- TwinCAT Module Class mit Datenzeiger
- TwinCAT Module Class für Echtzeitkontext
- TwinCAT Module Class mit Online Changeable Fähigkeit



TwinCAT Modules Class

Erstellt eine neue TwinCAT-Modulklasse.

Diese Vorlage erzeugt ein grundlegendes Kernmodul. Es verfügt weder über einen zyklischen Aufrufer, noch über einen Datenbereich, ist aber ein guter Ausgangspunkt für die Implementierung von abrufbaren Funktionen auf Anfrage eines Aufrufers.

Zum Beispiel für die Erstellung einer C++ Methode, die von einem SPS-Modul oder einem anderen C++ Modul aufgerufen wird.

Siehe Beispiel11 [▶ 275]

TwinCAT Modules Class mit ADS Port

Diese Vorlage bietet sowohl das C++ Modul als auch die Funktionsweise eines ADS Servers und ADS Clients.

- ADS Server:
  Kann als einzelne Instanz dieser Vorlage des C++ Moduls laufen und kann mit einer spezifischen ADS Port-Nummer (z.B. 25023) vorkonfiguriert werden.
  Ermöglicht mehrere Instanzen dieser Vorlage, wobei jedem C++ Modul seine eigene eindeutige ADS Port-Nummer von TwinCAT 3 zugewiesen wird (z.B. 25023, 25024, 25025, ...).
  Die ADS-Meldungen können dank der Implementierung des C++ Moduls analysiert und verarbeitet werden.
  Das ADS Handling zwecks Zugriff auf Ein-/Ausgangsdatenbereiche muss nicht über ein eigenes ADS Message Handling implementiert werden.

- ADS Client:
  Diese Vorlage stellt Beispielcodes zur Verfügung, um einen ADS-Aufruf mittels Versand einer ADS-Meldung an einen ADS-Partner zu initiieren.

Da die Module sich wie ADS Client oder ADS Server verhalten, die untereinander über ADS-Nachrichten kommunizieren, können die beiden Module (Aufrufer=Client und der Aufgerufene=Server) im gleichen oder unterschiedlichen Echtzeitkontexten auf dem gleichen oder unterschiedlichen CPU-Kernen laufen.
Weil ADS netzübergreifend arbeiten kann, können die beiden Module auch auf verschiedenen im Netzwerk befindlichen Maschinen laufen.

Siehe Beispiel03 [▶ 256], ADS-Kommunikation [▶ 200]

TwinCAT Modules Class mit zyklischem Aufrufer

Ermöglicht den zyklischen Aufruf eines C++ Programms, das aber über keinen Zugang zur Außenwelt verfügt.

Wird nicht häufig verwendet, eine Modulklasse mit zyklischem Aufrufer und zyklischem I/O wird bevorzugt.

TwinCAT Modules Class mit zyklischem Ein-/Ausgang

Erzeugt eine neue TwinCAT-Modulklasse, die die zyklisch aufrufende Schnittstelle implementiert und einen Ein- und Ausgangsdatenbereich aufweist.

Die Ein- und Ausgangsdatenbereiche können mit anderen Ein-/Ausgangsabbildern oder mit physikalischen E/A-Klemmen verbunden werden.

Wichtig zu verstehen:

Das C++ Modul verfügt über seinen eigenen logischen Ein-/Ausgangsdatenspeicherbereich. Die Datenbereiche des Moduls können mit dem Systemmanager konfiguriert werden.

Wenn das Modul mit einer zyklischen Schnittstelle gemappt wird, bestehen Kopien der Ein- und Ausgangsdatenbereiche in beiden Modulen (dem Aufrufer und dem Aufgerufenen) und auf diese Weise kann das Modul unter einem anderen Echtzeitkontext und selbst auf einem anderen CPU-Kern in Bezug auf ein anderes Modul laufen.
TwinCAT wird die Daten zwischen den Modulen auf ununterbrochene Weise kopieren.

Siehe Schnellstart [▶ 62], Beispiel01 [▶ 254].

TwinCAT Modules Class mit Datenzeiger

Genau wie das TwinCAT Module Class with Cyclic IO erzeugt auch diese Vorlage eine neue TwinCAT-Modulklasse die eine aufrufende Schnittstelle mit einem Ein- und Ausgangsdatenbereich für die Verknüpfung mit anderen Logik-Ein-/Ausgangsabbildern oder mit physikalischen E/A-Klemmen implementiert.

Darüber hinaus bietet diese Vorlage Datenzeiger, mit Hilfe derer auf Datenbereiche von anderen Modulen über Zeiger zugegriffen werden kann.

Wichtig zu verstehen:

Anders als im Falle des zyklischen I/O-Datenbereichs, wo die Daten zwischen Modulen zyklisch kopiert werden, besteht im Falle der Verwendung von C++-Datenzeigern nur ein einziger Datenbereich und dieser gehört dem Zielmodul. Beim Schreiben von einem anderen C++ Modul über den Datenzeigermechanismus auf das Zielmodul wird sich das sofort auf den Datenbereich des Zielmoduls auswirken. (Nicht notwendigerweise gegen Ende eines Zyklus).

Wenn das Modul während der Laufzeit ausgeführt wird, findet der Aufruf sofort statt, wodurch der ursprüngliche Prozess blockiert wird (es ist ein Zeiger...). Aufgrund dessen müssen beide Module (der Aufrufer und der Aufgerufene) sich im selben Echtzeitkontext und auf demselben CPU-Kern befinden.

Der Datenzeiger wird im TwinCAT Module Instance Configurator [▶ 135] konfiguriert.

Siehe Beispiel10 [▶ 274]

TwinCAT Modules Class für Echtzeitkontext

Diese Vorlage erstellt ein Modul, das im Echtzeit-Kontext instanziiert werden kann.

Wie im Kapitel TwinCAT-Modul Zustandsmaschine [▶ 48] beschrieben, verfügen die übrigen Module über Transitionen für das Hochfahren und Runterfahren in Nicht-Echtzeitkontext. Manchmal müssen Module bei bereits laufender Echtzeit gestartet werden, sodass alle Transitionen im Echtzeit-Kontext ausgeführt werden. Dies ist eine entsprechende Vorlage.
Die Module mit dieser (modifizierten) Statemachine können auch verwendet werden, um direkt beim Starten von TwinCAT instanziiert zu werden. In diesem Fall werden die Transitionen wie bei einem normalen Modul ausgeführt.

Das Beispiel TcCOM 03 [▶ 337] zeigt die Verwendung eines solchen Moduls.

TwinCAT Module Class mit Online Change-Fähigkeit

Diese Möglichkeit kann nur verwendet werden, wenn das Modul zu einem Versioned C++ Project hinzugefügt wird.

Diese Vorlage erstellt ein Modul, das Online-Change fähig ist. Durch die Versionierung des Projektes werden diese Module zur Laufzeit austauschbar – es ist also möglich, die Module von unterschiedlichen Versionsständen zur Laufzeit auszutauschen.

Das Verfahren für diesen Online-Change ist im Kapitel Online-Change [▶ 155] beschrieben.

Das Modul selbst entspricht ansonsten einem Modul mit zyklischem Ein-/Ausgang.

11.3 TwinCAT Module Class Editor (TMC)

Der TwinCAT Module Class Editor (kurz TMC Editor) wird für die Definition der Klasseninformation eines Moduls verwendet. Es handelt sich um Datentypdefinitionen incl. deren Verwendung, bereitgestellte und implementierte Schnittstellen, sowie Datenbereiche und Datenzeiger.

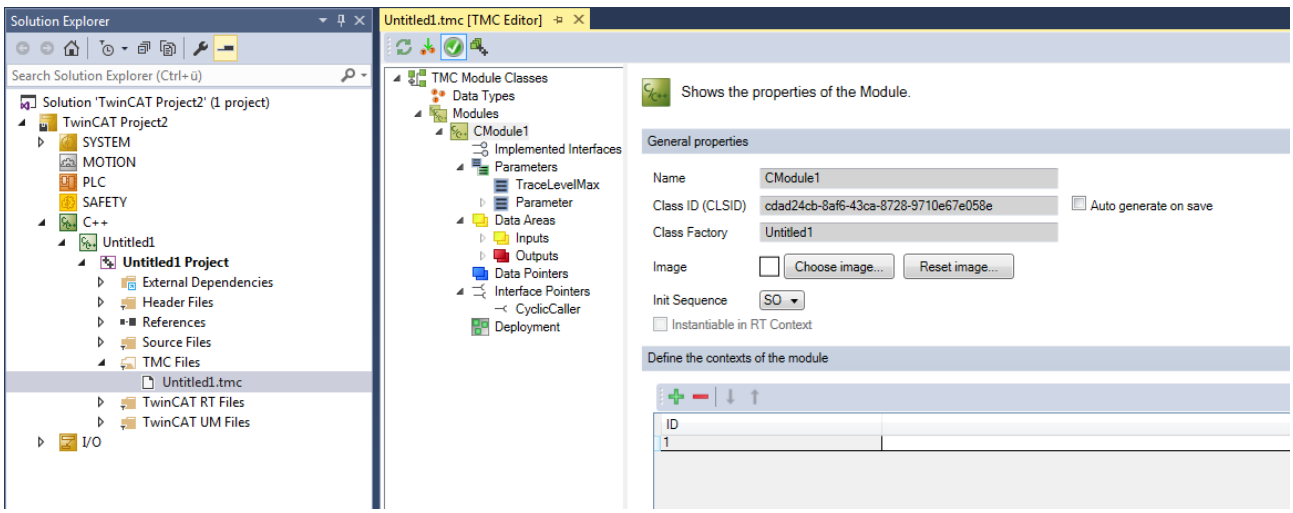
Kurz: alles, was von außen zu sehen ist, muss mit diesem Editor definiert werden.

Die Grundidee ist:

1. Mit dem TMC Editor verändern Sie die Modul-Beschreibungsdatei (TMC-Datei). Diese enthält alle Informationen, die im TwinCAT System selbst zugreifbar sind. Dieses sind beispielsweise Symbole, implementierte Schnittstellen und Parameter.
2. Mit dem TwinCAT Code Generator, der auch aus dem TMC Editor verwendet werden kann, wird der gesamte notwendige C++ Code, d. h. Header- und cpp-Dateien, erzeugt.

Starten Sie den TMC Editor

Öffnen Sie den Editor durch Doppelklick auf die TMC-Datei eines Moduls. Der grafische Editor wird geöffnet:



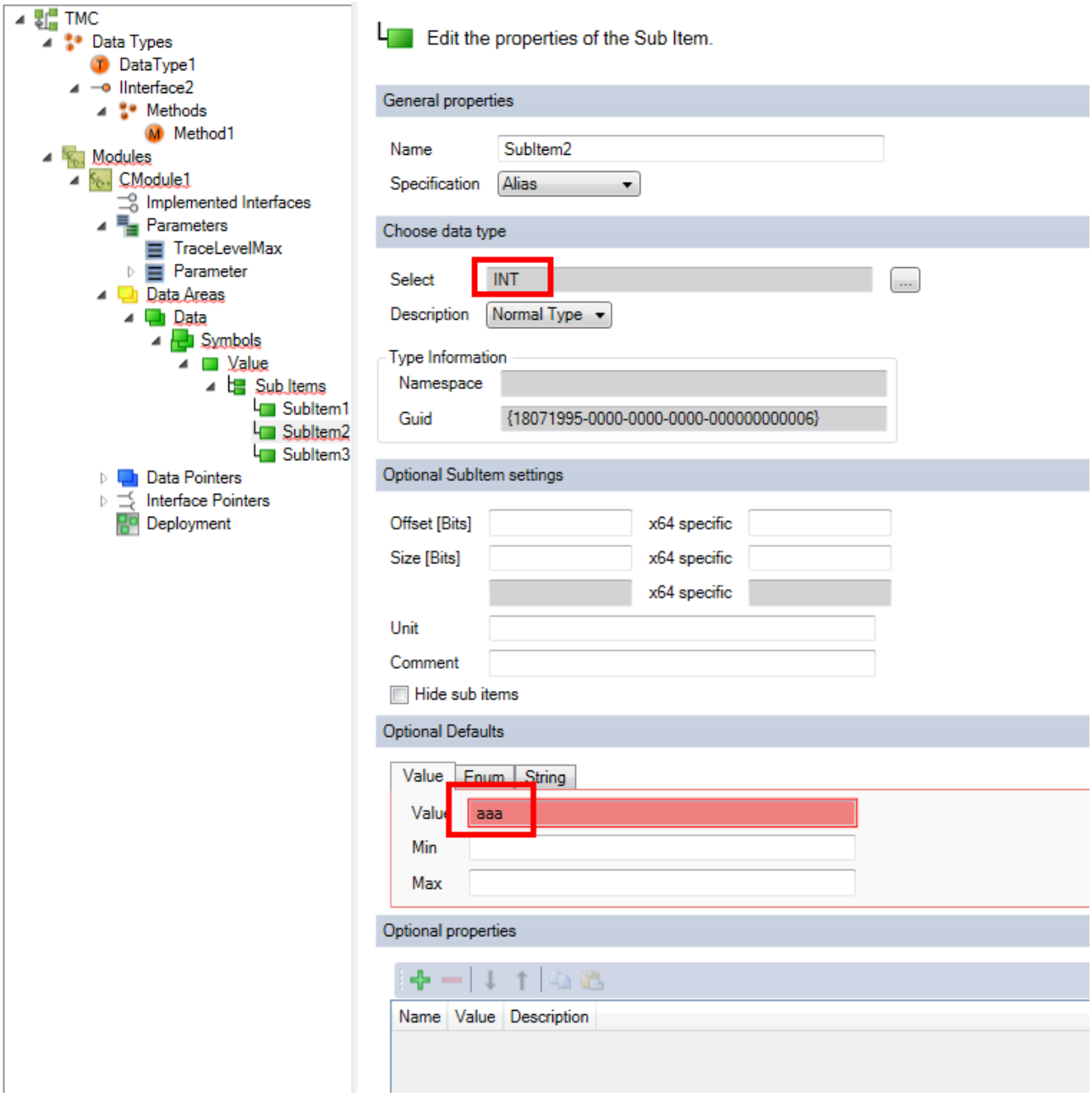
Funktionalitäten des TMC Editors sind:

- Symbole in den Datenbereichen, wie z. B. die logischen Ein- oder Ausgangsprozessabbilder eines Moduls erstellen / löschen / bearbeiten.
- Benutzerdefinierte Datentypdefinitionen erstellen / löschen / bearbeiten.
- Symbole in der Parameterliste eines Moduls erstellen / löschen / bearbeiten.

Nutzer-Hilfen

Der TMC Editor unterstützt den Nutzer bei der Definition seiner Datentypen und C++ Module.

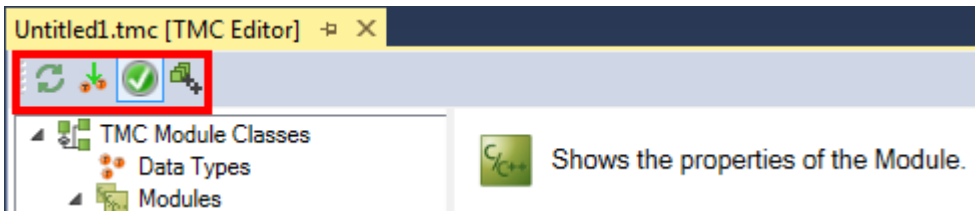
Zum Beispiel bei Problemen (Alignment, ungültige Standarddefinitionen, ...) innerhalb des TMC, wird der Nutzer über die Anzeige von roten Markierungen innerhalb des TMC-Baums zur entsprechenden Stelle geführt:



Trotzdem kann der Nutzer direkt die TMCs bearbeiten, weil es sich um XML handelt, das somit vom Nutzer selbst erzeugt und bearbeitet werden kann.

Werkzeuge

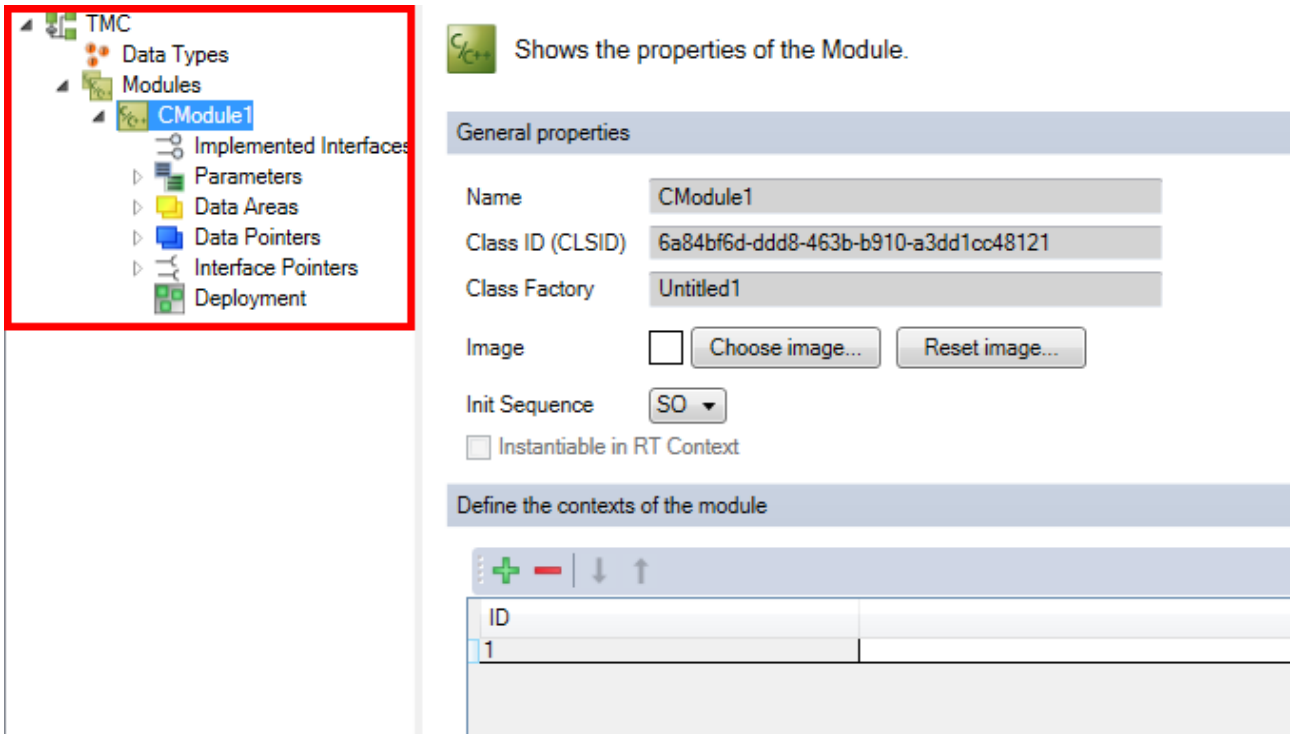
Im oberen Bereich des TMC Editors befinden sich Symbole für die benötigten Arbeitsschritte.



- Erneutes Laden der TMC Datei sowie der Typen aus dem Typsystem.
- Aktualisierung der überlagerten Datentypen.
- An- /Ausschalten der Nutzer-Hilfen [▶ 93].
- Start des TwinCAT TMC Code Generator:

Der Editor wird die eingegebene Information in die TMC-Datei speichern. Diese TMC-Beschreibung wird vom TwinCAT TMC Code Generator in Quellcode umgewandelt, der auch im Kontextmenü des TwinCAT C++ Projekts verfügbar ist.

11.3.1 Übersicht



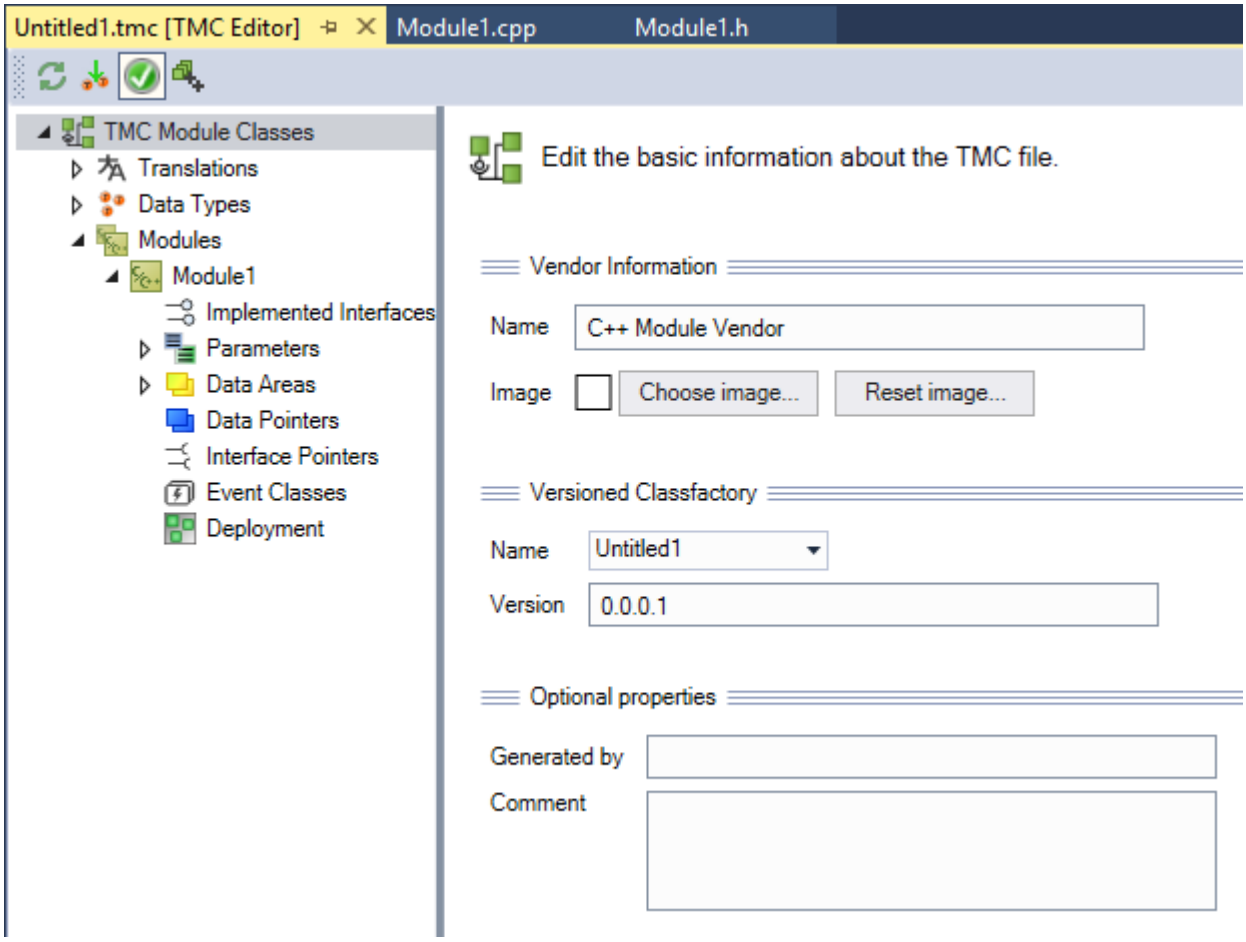
Benutzerinterface

- TMC [▶ 95]: Hier bearbeiten Sie die grundlegenden Angaben des Herstellers des C++ Moduls und fügen ein Bild hinzu.
- Data Types [▶ 96]: Hier fügen Sie Datentypen hinzu, entfernen sie oder ordnen sie neu.
- Modules [▶ 113]: Zeigt die Module des Treibers.
- Implemented Interfaces [▶ 115]: Zeigt die implementierten Schnittstellen des Moduls.
- Parameters [▶ 116]: Hier fügen Sie Ihre Parameter hinzu, entfernen sie oder ordnen sie neu.

  - TraceLevelMax [▶ 122]: Parameter, der die Menge an protokollierten Nachrichten steuert, ist für (fast) jedes Modul vordefiniert.
- Data Areas [▶ 123]: Hier fügen Sie Datenbereiche hinzu, entfernen sie oder ordnen sie neu.
- Data Pointers [▶ 130]: Hier fügen Sie Datenzeiger hinzu, entfernen sie oder ordnen sie neu.
- Interface Pointers [▶ 132]: Hier fügen Sie Interfacezeiger hinzu, entfernen sie oder ordnen sie neu.
- Deployment [▶ 133]: Bestimmt die Dateien, die bereitgestellt werden.

11.3.2 Grundlegende Informationen

Grundlegende Informationen bezüglich der TMC-Datei finden Sie hier:



Informationen zum Anbieter

Name: Dies ist der Name des Anbieters.

Choose Image: Hier fügen Sie ein 16x16 Pixel-Bitmap-Symbol ein.

Reset image: Setzt das Modulbild auf den Standardwert zurück.

Versioned Classfactory

Name: Zeigt alle aus der TMC-Datei referenzierten Classfactories an. Eingestellt werden muss die Classfactory, die das C++ Projekt implementiert. Typischerweise ist dies der Name des Projektes. Wird nur für versionierte C++ Projekte verwendet, ansonsten steht hier „not set“.

Version: Die aktuelle Version, bestehend aus vier Ziffern getrennt durch jeweils einen „.“. Mindestens eine Zahl muss ungleich 0 sein.

Optionale Eigenschaften

Generated by: In diesem Feld wird angegeben, wer die Datei erstellt hat und wer diese pflegen wird. Beachten Sie, dass beim Ausfüllen dieses Feldes Änderungen nicht mehr möglich sind (deaktiviert alle Bearbeitungsvorgänge) im TMC Editor.

Comment: Optional können Sie hier einen Kommentar eingeben.

11.3.3 Datentypen

Im TwinCAT Module Class (TMC) Editor können benutzerdefinierte Datentypen definiert werden.

Bei diesen Datentypen kann es sich um Typendefinitionen, Strukturen, Bereiche, Aufzählungen oder Schnittstellen, z.B. Methoden und deren Signaturen, handeln.

Das TwinCAT Engineering System (XAE) veröffentlicht diese Datentypen gegenüber allen anderen verschachtelten Projekten des TwinCAT Projekts, so dass diese auch z. B. in SPS-Projekten verwendet werden können (wie im Kapitel Beispiel11: Modulkommunikation: Methodenaufruf SPS-Modul nach C++-Modul [▶ 275] beschrieben).

HINWEIS

Namenskonflikt

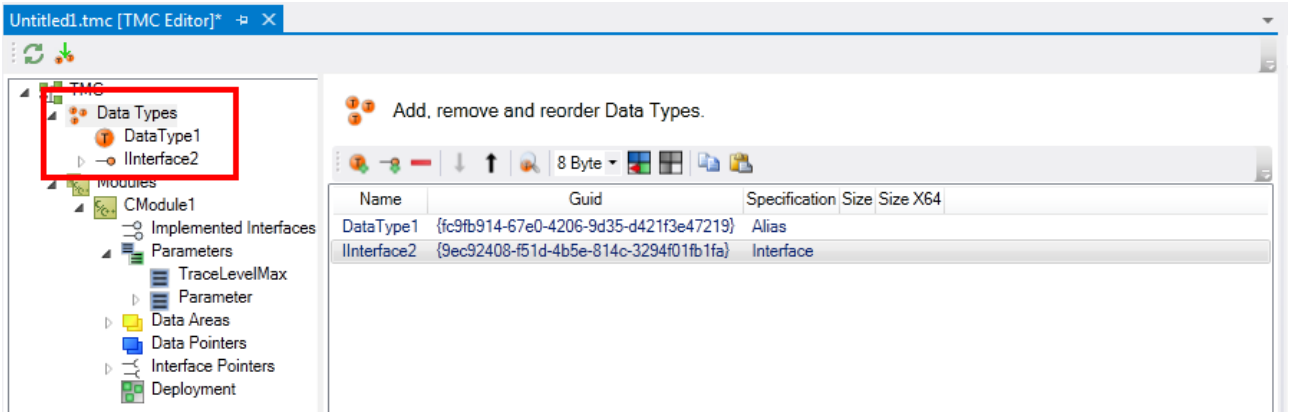
Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.

- Verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen.

In diesem Kapitel wird beschrieben, wie die Fähigkeiten des TMC Editors bei der Definition von Datentypen zu verwenden sind.

11.3.3.1 Übersicht

Benutzerinterface



Symbol	Funktion
	Einen neuen Datentyp hinzufügen.
	Eine neue Schnittstelle hinzufügen.
	Den ausgewählten Typ löschen.
	Das ausgewählte Element um eine Position nach unten verschieben.
	Das ausgewählte Element um eine Position nach oben verschieben.
	Nicht verwendete Typen suchen.
	Byte Alignment auswählen.
	Ausgewählten Datentyp ausrichten (Alignment) . Diese Funktion durchläuft alle genutzten Datentypen (Rekursion). Ist dieses nicht gewünscht, kann Schritt für Schritt vorgegangen werden, indem die Funktion innerhalb der Datentypen verwendet wird.
	Datenformat des ausgewählten Datentyps zurücksetzen.
	Kopieren
	Einfügen

Datentypeigenschaften

Name: Benutzerdefinierter Name des Datentyps.

GUID: Eindeutige ID des Datentyps.

Specification: Festlegung des Datentyps.

Size: Größe des Datentyps, wenn ausdrücklich spezifiziert.

Size X64: Unterschiedliche Größe des Datentyps für x64-Plattform.

11.3.3.2 Datentypen hinzufügen / bearbeiten / löschen

Mit Hilfe des TwinCAT Module Class (TMC) Editors können Datentypen, die von TwinCAT C++ Modulen verwendet werden, hinzugefügt, bearbeitet und gelöscht werden.

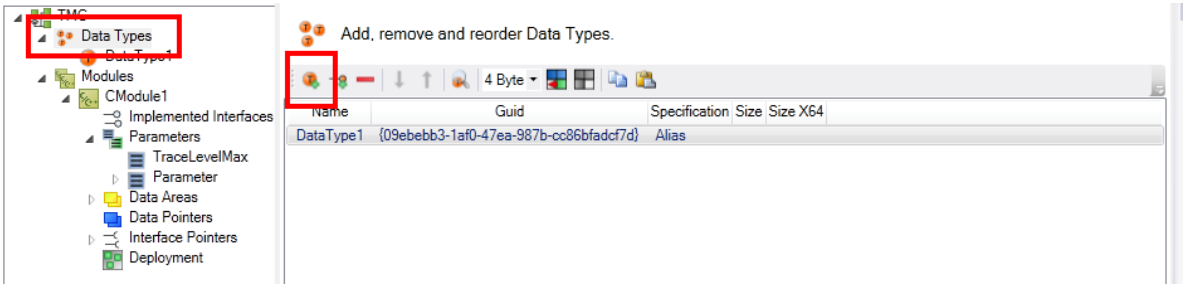
Dieser Artikel beschreibt:

- Schritt 1: Einen neuen Datentyp [▶ 97] in der TMC-Datei erstellen.
- Schritt 2: TwinCAT TMC Code Generator starten [▶ 100], um C++ Code auf der Grundlage einer Modulbeschreibung in der TMC-Datei zu generieren.
- Die Datentypen verwenden [▶ 113].

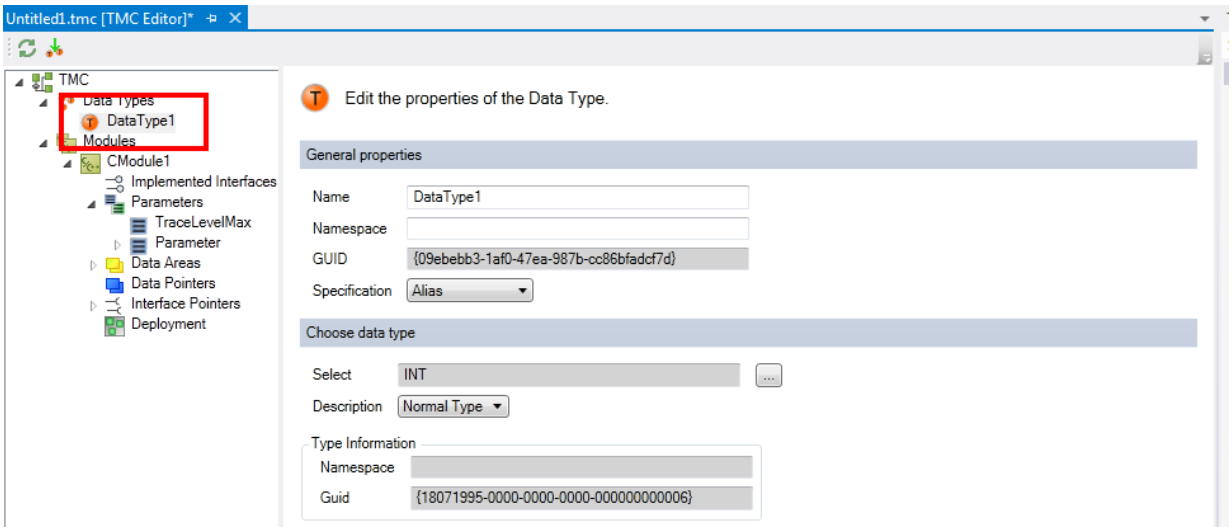
Schritt 1: Einen neuen Datentyp erzeugen

1. Wählen Sie nach dem Starten des TMC Editors den Knoten **Data Types** aus.
2. Erweitern Sie die Liste der Datentypen und Schnittstellen mit einem neuen Datentyp durch Klicken auf die **+** Schaltfläche **Add a new data area**.

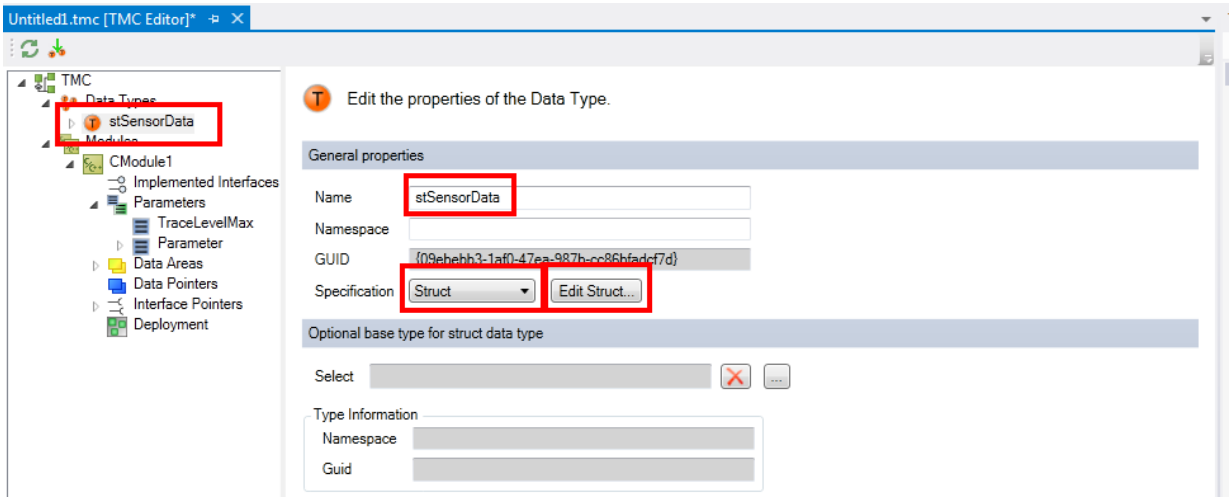
⇨ Daraufhin wird ein neuer **Datentyp** als neuer Eintrag aufgeführt:



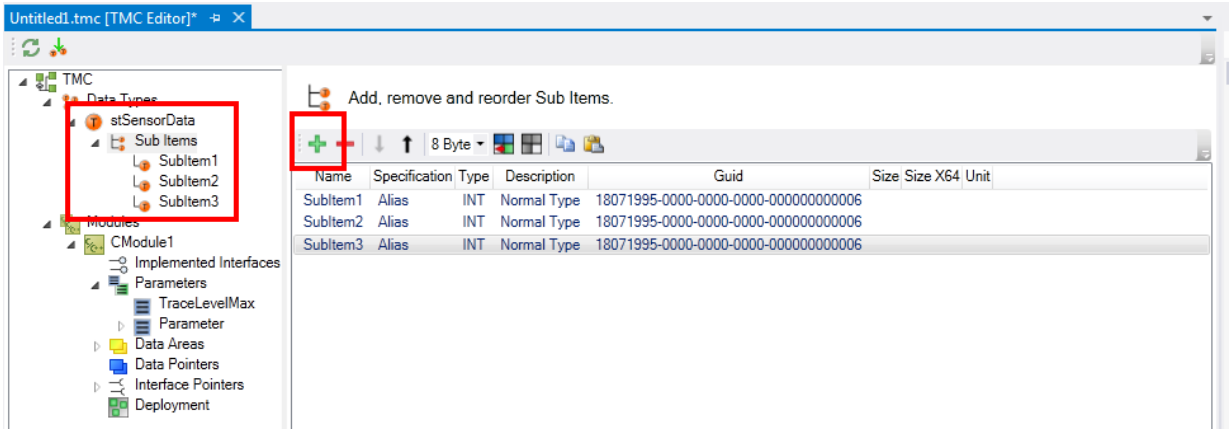
3. Wählen Sie den generierten „Data Type1“ um Einzelheiten zum neuen Datentyp zu erhalten.



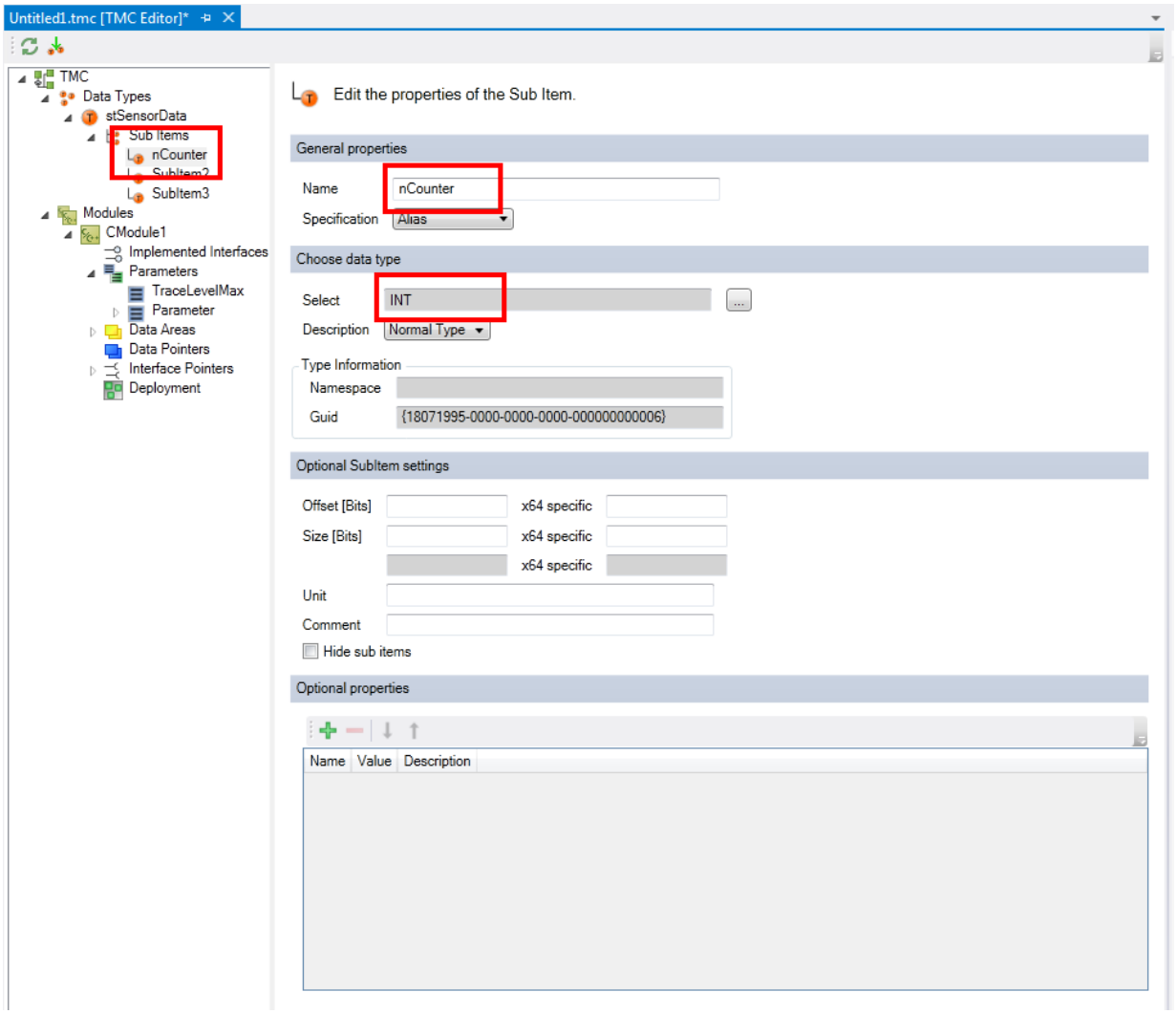
4. Spezifizieren [▶ 107] Sie den Datentyp.
5. Benennen Sie den Datentyp um.
   In diesem Beispiel **stSensorData** wählen Sie die Spezifikation **STRUCT** und klicken auf **Edit Struct**.



6. Fügen Sie neue Unterelemente in die Struktur ein durch Klicken auf die **Add a new sub item**-Schaltfläche.



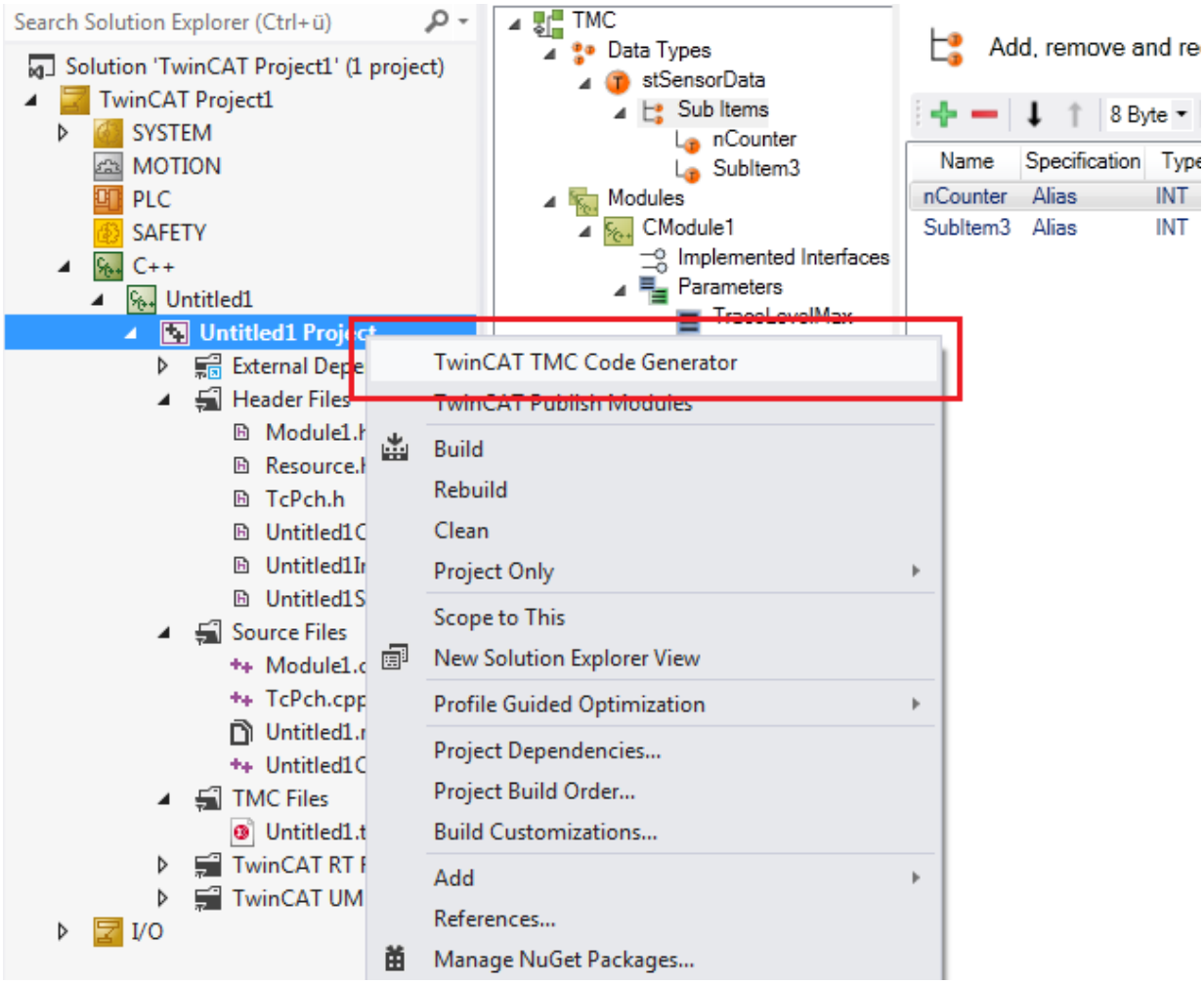
7. Mit Doppelklick auf das Unterelement können Sie die Eigenschaften bearbeiten. Geben Sie dem Unterelement einen neuen Namen und wählen Sie einen geeigneten Datentyp.



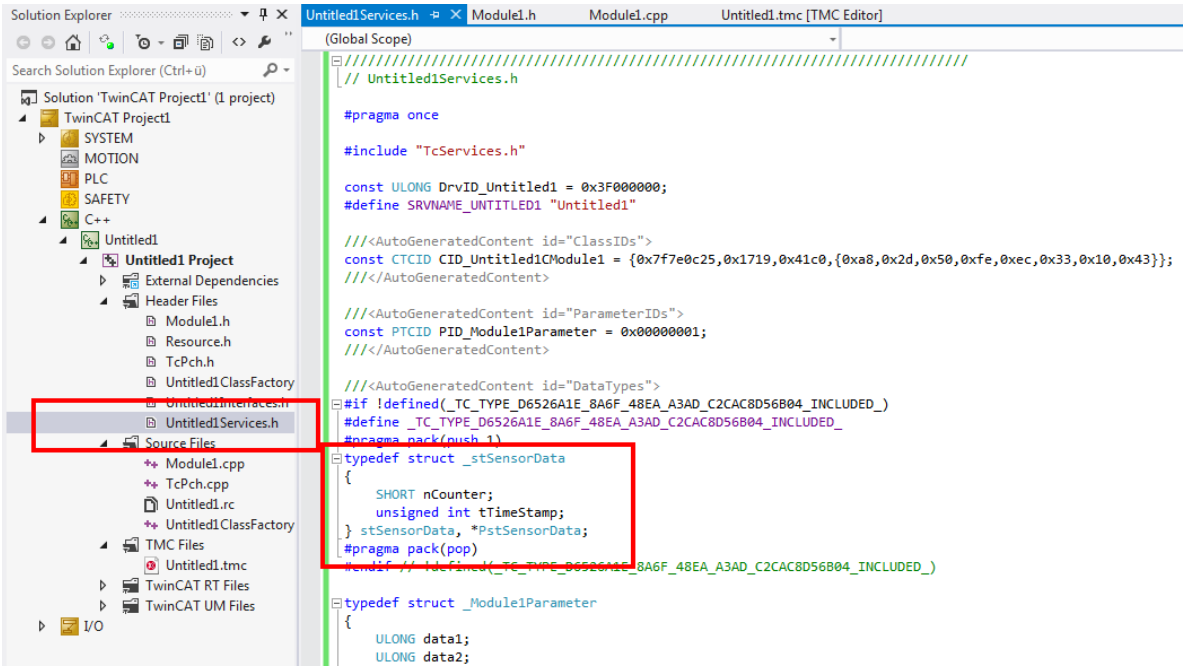
8. Geben Sie den anderen Unterelementen einen neuen Namen und wählen einen geeigneten Datentyp.
9. Speichern Sie Ihre in der TMC-Datei vorgenommenen Änderungen.

Schritt 2: Starten Sie den TwinCAT TMC Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.

1. Klicken Sie mit der rechten Maustaste auf Ihre Projektdatei und wählen Sie **TwinCAT TMC Code Generator**, um den Quellcode Ihres Datentyps zu erzeugen:



⇨ Sie sehen die Datentypdeklaration in der Modul-Headerdatei „Untitled1Services.h“



⇨ Wenn Sie einen weiteren Datentyp oder ein weiteres Unterelement hinzufügen, führen Sie den TwinCAT TMC Code Generator erneut aus.

11.3.3.3 Schnittstellen hinzufügen / bearbeiten / löschen

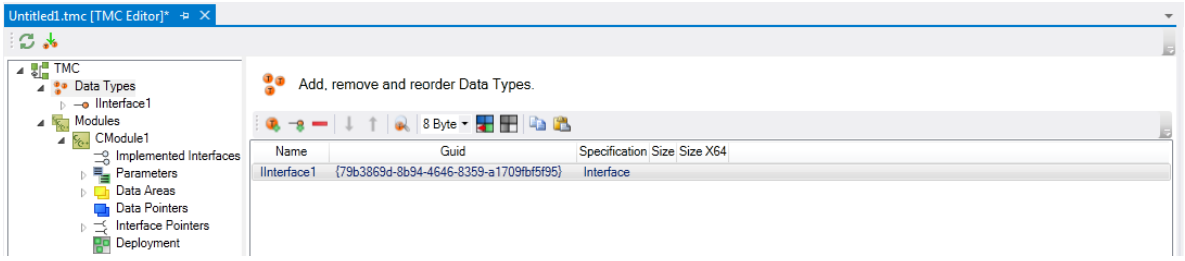
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Schnittstellen eines TwinCAT Moduls hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

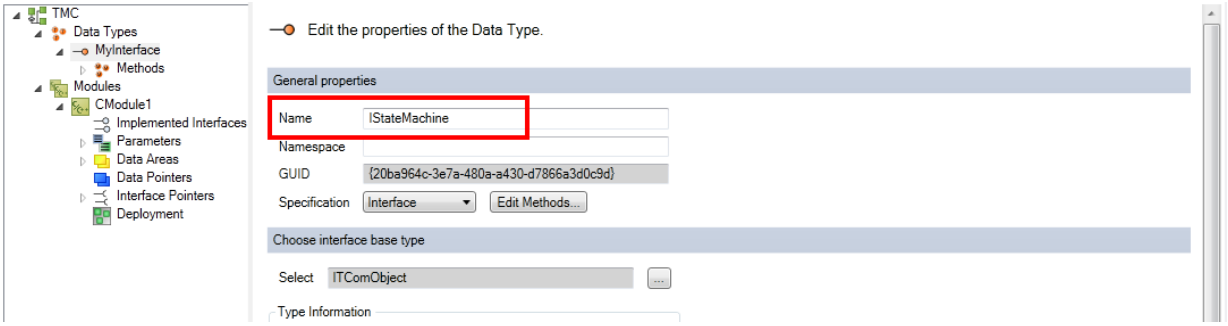
- Schritt 1: Eine neue Schnittstelle [▶ 101] in der TMC-Datei erstellen.
- Schritt 2: [▶ 102]Der Schnittstelle in der TMC-Datei Methoden hinzufügen [▶ 102].
- Schritt 3: Verwenden Sie die Schnittstelle [▶ 103], indem Sie diese zu „Implemented Interfaces“ des Moduls hinzufügen.
- Schritt 4: Starten Sie den TwinCAT TMC [▶ 105] Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.
- Optionale Änderung der Schnittstelle [▶ 105].

Schritt 1: Eine neue Schnittstelle erzeugen

1. Wählen Sie nach dem Starten des TMC Editors den Knoten **Data Types** aus.
2. Klicken Sie auf **Add a new interface**, um die Liste der Schnittstellen um eine neue Schnittstelle zu erweitern.
   ⇨ Daraufhin wird **IInterface1** als neuer Eintrag aufgeführt:



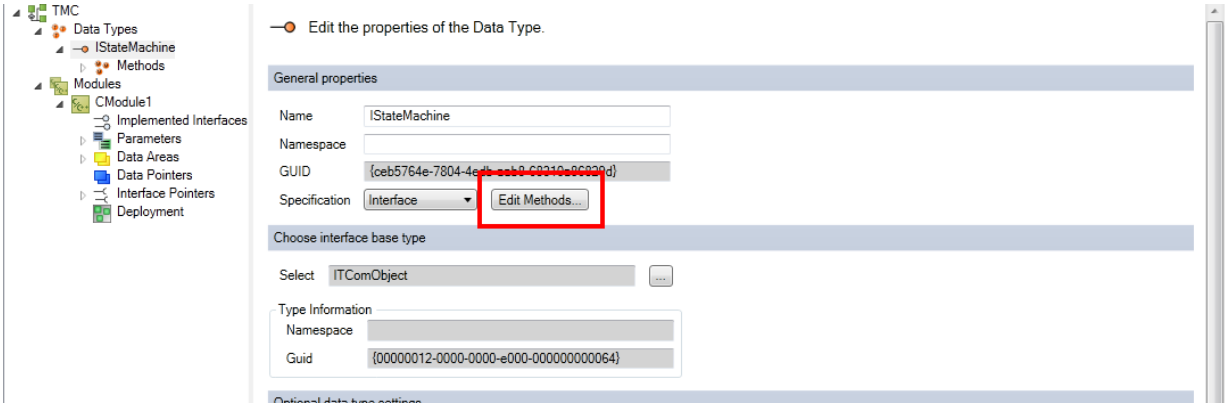
3. Entweder wählen Sie den entsprechenden Knoten im Baum oder machen einen Doppelklick auf die Zeile in der Tabelle, um die Einzelheiten zu öffnen.



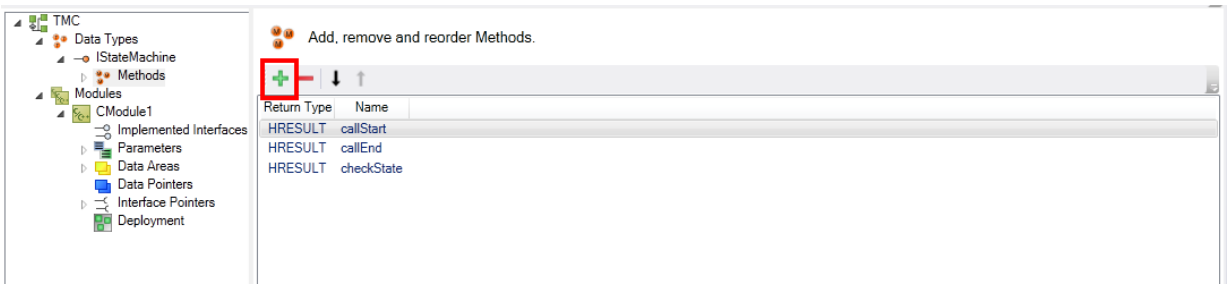
4. Geben Sie einen aussagekräftigeren Namen ein - in diesem Beispiel „IStateMachine“.

Schritt 2: Fügen Sie der Schnittstelle Methoden hinzu

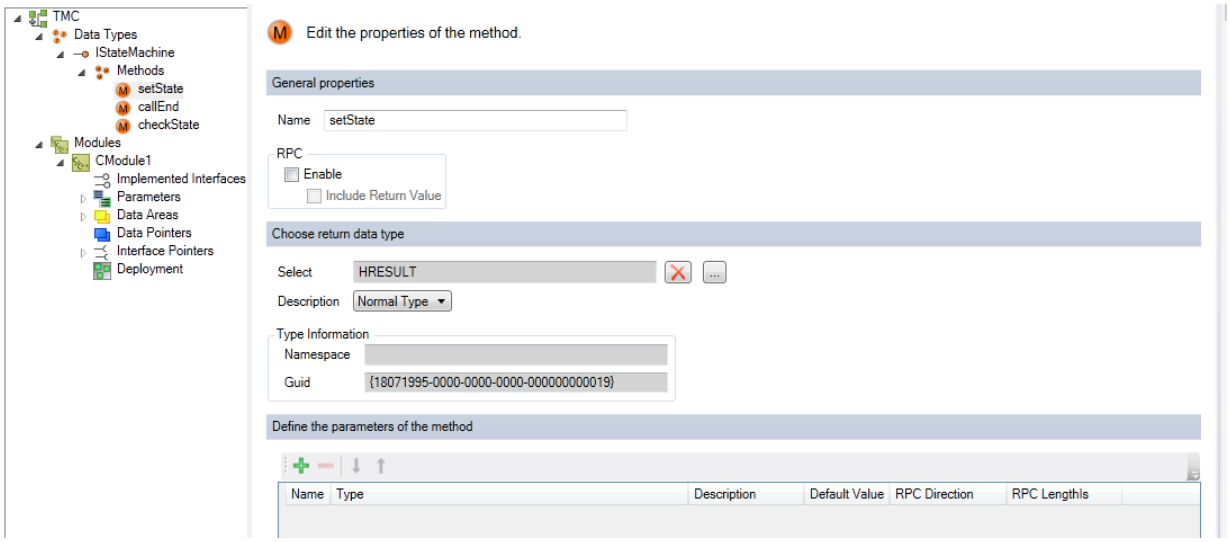
1. Klicken Sie auf **Edit Methods...**, um eine Liste der Methoden dieser Schnittstelle zu erhalten:



2. Klicken Sie auf die **+** Schaltfläche um eine neue standardmäßige Methode „Method1“ zu erzeugen:



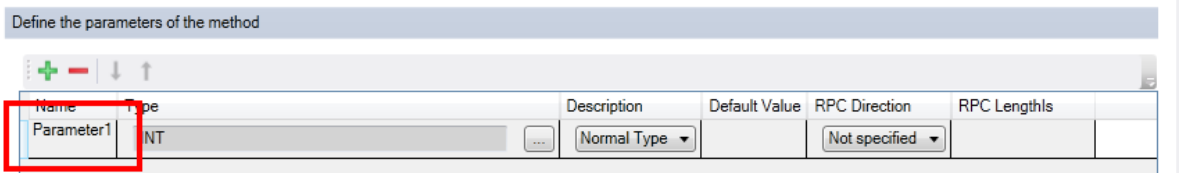
3. Doppelklicken Sie auf die Methode oder wählen den Knoten im Baum aus, um Einzelheiten zu öffnen.
4. Geben Sie der standardmäßigen „Method1“ einen aussagekräftigeren Namen.
5. Anschließend können Sie mit einem Klick auf **Add a new parameter** Parameter hinzufügen bzw. Parameter der Methode „SetState“ bearbeiten.



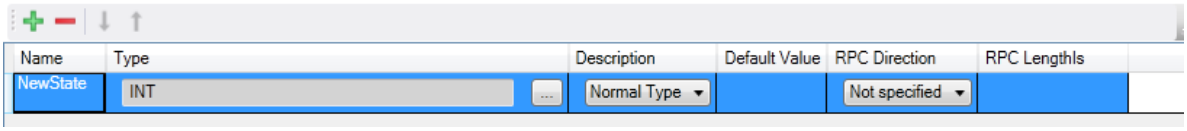
   ⇨ Standardmäßig wird der neue Parameter „Parameter1“ als „Normal Type“ „INTEGER“ erzeugt.

6. Bearbeiten Sie den Parameter durch einen Klick auf den Namen „Parameter1“.

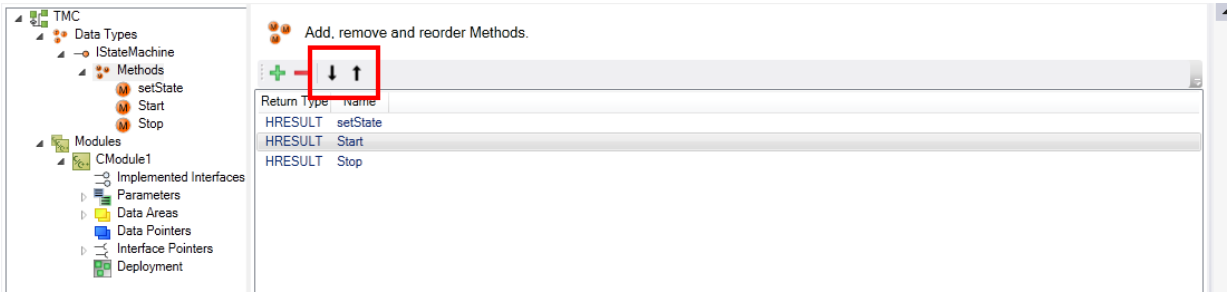
   ⇨ Der „Normal Type“ kann auch in „Pointer“ geändert werden usw. - auch kann der Datentyp selber ausgewählt werden.



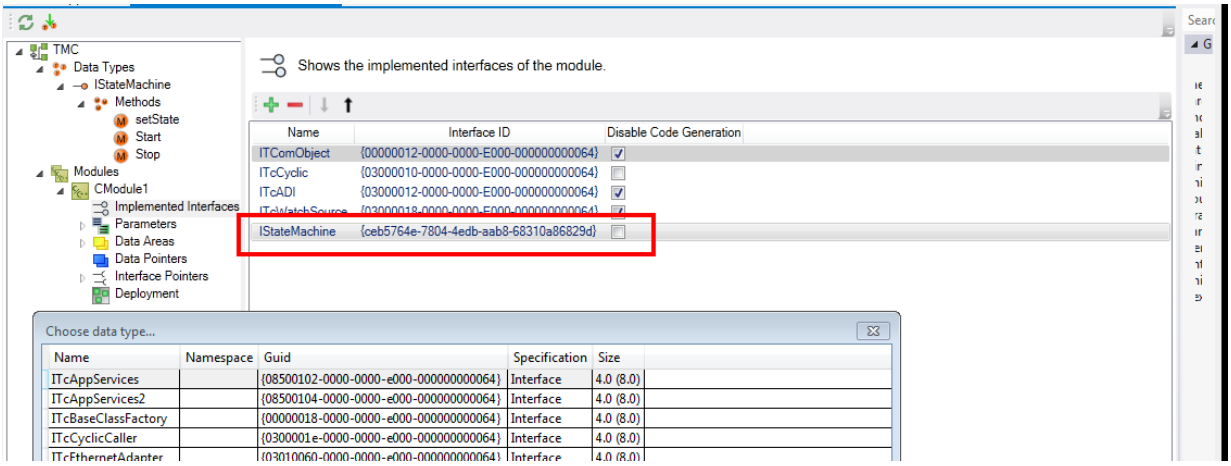
⇨ In diesem Falle ist „NewState“ der neue Name - die übrigen Einstellungen werden nicht geändert.



7. Durch Wiederholen des Schritts 2 „Methoden zur Schnittstelle hinzufügen“ werden alle Methoden aufgelistet - mit Hilfe der **nach oben** / **nach unten** Schaltfläche ordnen Sie die Methoden neu.

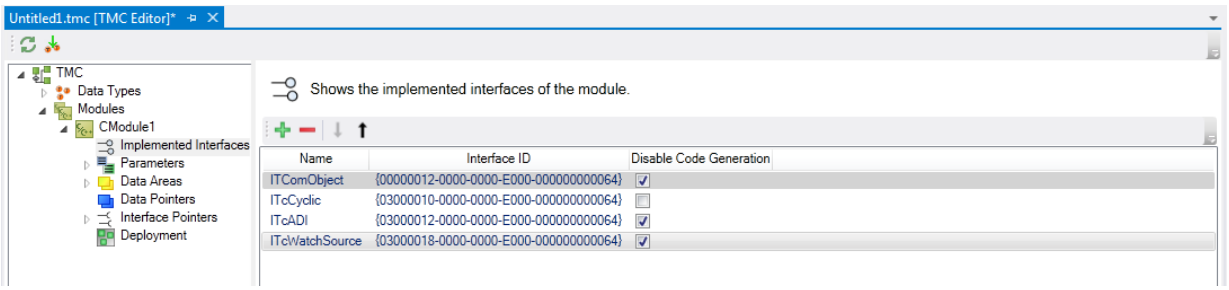


8. Die Schnittstelle ist bereit, durch Ihr Modul implementiert zu werden.

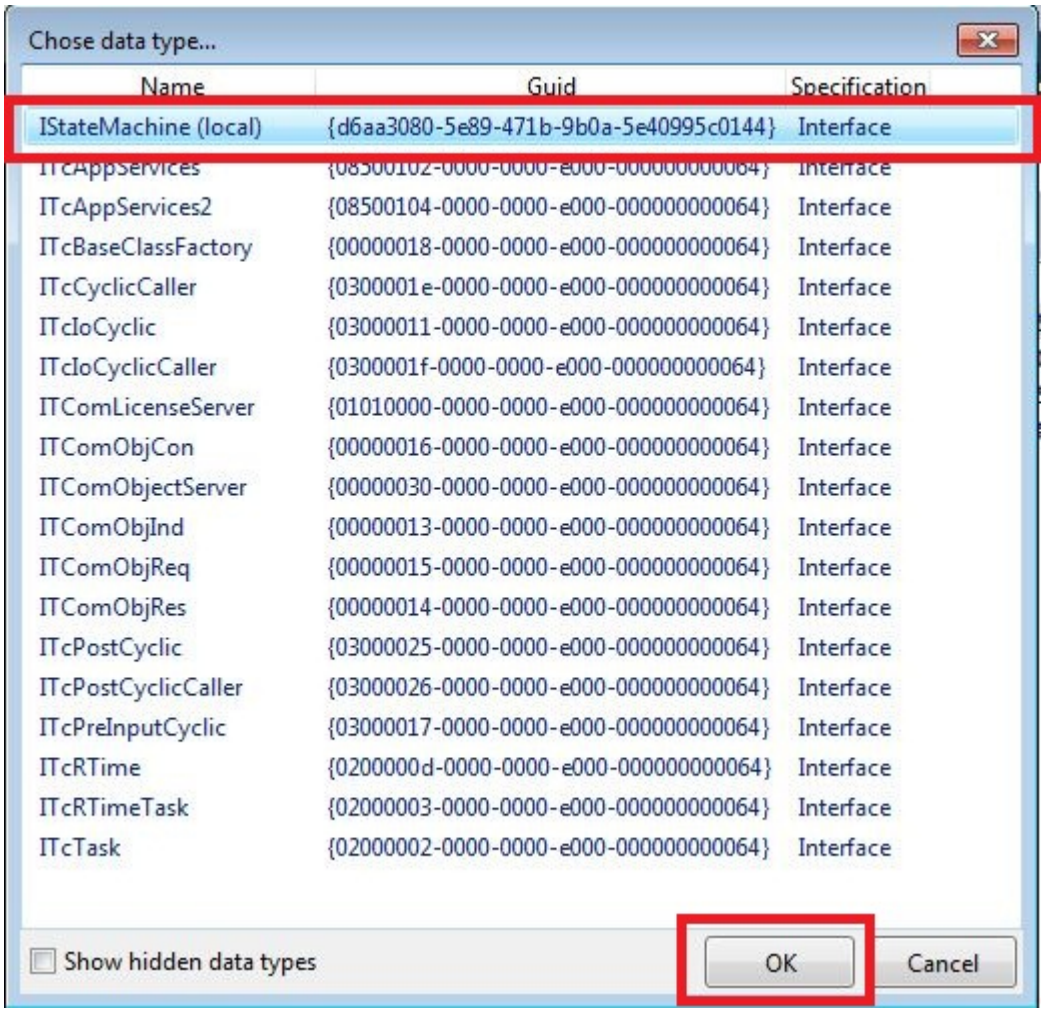


Schritt 3: Die neue Schnittstelle zu Implemented Interfaces hinzufügen

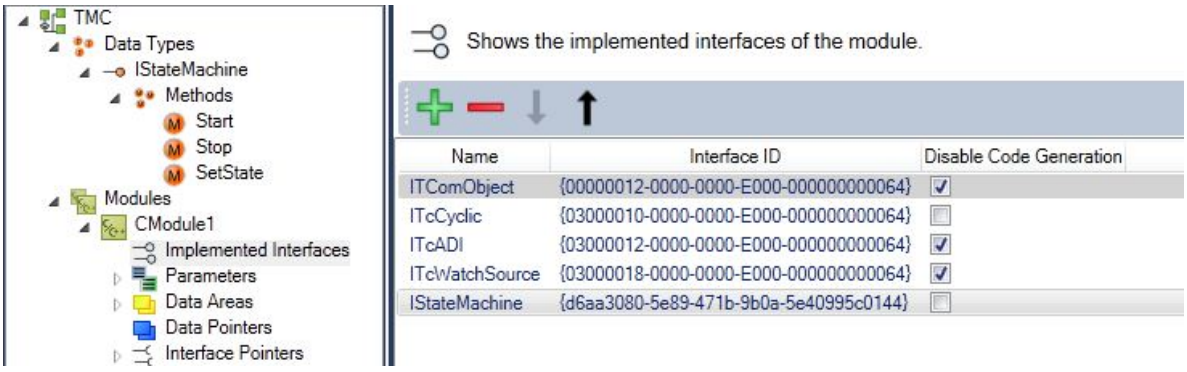
1. Wählen Sie das Modul, das mit der neuen Schnittstelle erweitert werden soll - in diesem Falle wählen Sie das Ziel **Modules->CModule1**.
2. Erweitern Sie die Liste der implementierten Schnittstellen um eine neue Schnittstelle durch Klick auf die **+** Schaltfläche mit **Add a new interface to the module**.



3. Alle verfügbaren Schnittstellen werden aufgeführt - wählen Sie die neue Vorlage „IStateMachine“ und beenden Sie mit **OK**.

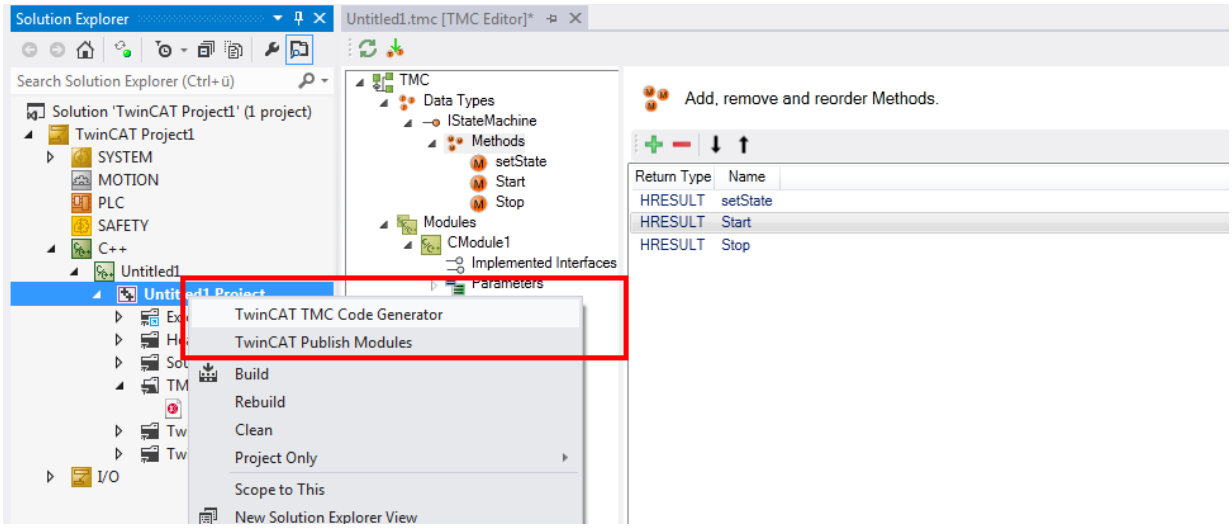


⇨ Die neue Schnittstelle „IStateMachine“ ist Teil der Modulbeschreibung.

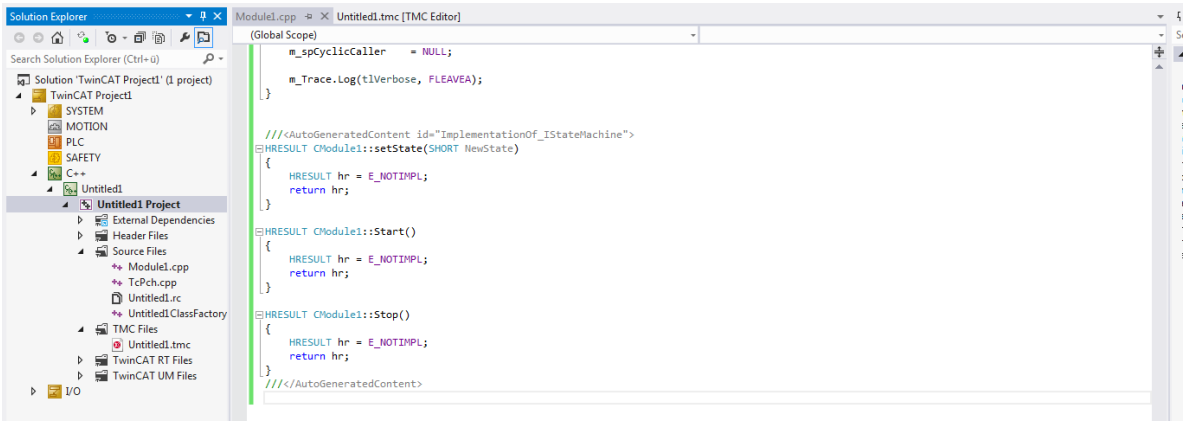


Schritt 4: Starten Sie den TwinCAT TMC Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.

1. Um den C/C++ Code anhand von dieser Modulbeschreibung zu generieren, klicken Sie mit der rechten Maustaste in das C/C++ Projekt und wählen dann den **TwinCAT TMC Code Generator.**



⇨ Daraufhin enthält das Modul „Module1“ die neuen Schnittstellen
CModule1: Start()
CModule1: Stop()
CModule1: SetState(SHORT NewState).



⇨ Fertig - der benutzerdefinierte Code kann nun in diesen Bereich eingefügt werden.

Optionale Änderung der Schnittstelle

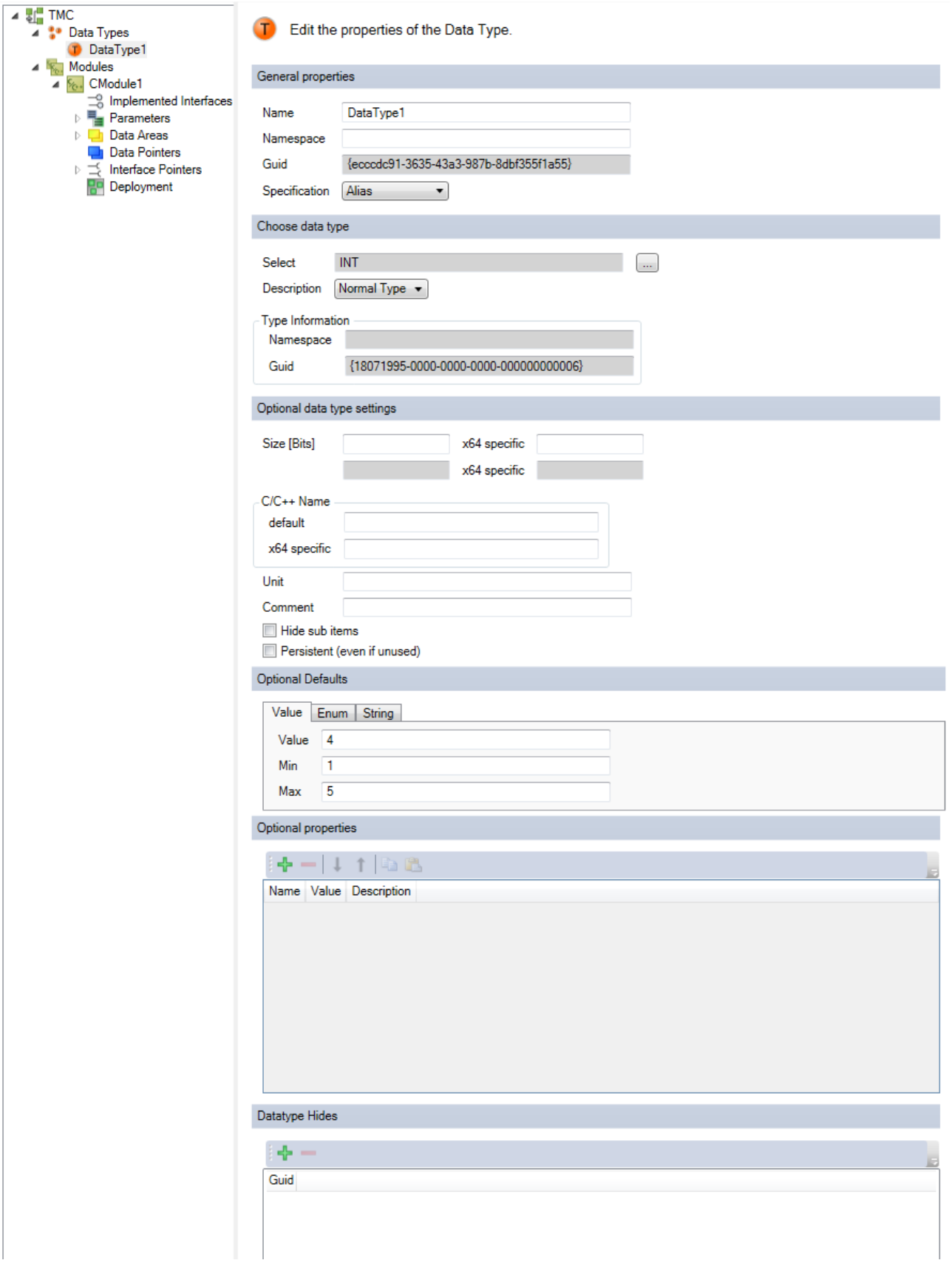
Benutzerdefinierter Code wird nie gelöscht

Im Fall von Änderungen an der Schnittstelle (z. B. die Parameter einer Methode werden später erweitert) wird benutzerdefinierter Code nie gelöscht. Stattdessen wird die bestehende Methode lediglich mit einem Kommentar versehen, wenn der TMC Code Generator die Methoden nicht mappen kann.

```
///<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 Datentypeigenschaften

Die Eigenschaften von Datentypen bearbeiten



Allgemeine Eigenschaften

Name: Benutzerdefinierter Name des Datentyps.

HINWEIS

Namenskonflikt

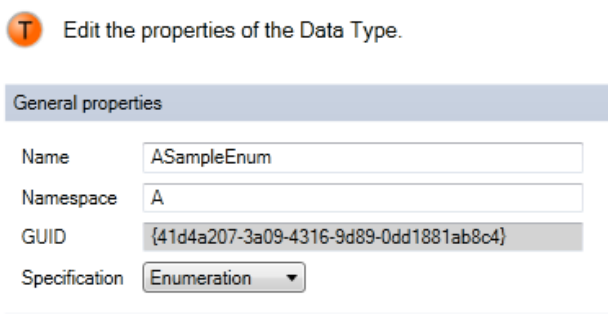
Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.

- Verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen.

Namespace: Benutzerdefinierter Namensraum des Datentyps.

Beachten Sie, dass dieser **nicht** einem C Namensraum zugeordnet wird. Er wird als Präfix Ihres Datentyps verwendet werden.

Beispiel: eine Aufzählung mit einem Namensraum „A“:



Der folgende Code wird generiert:

```
///<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_)
```

Möglicherweise möchten Sie den Namensraumnamen dem Aufzählungselement manuell als Präfix hinzufügen:

```
#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: Eindeutige ID des Datentyps.

Specification: Festlegung des Datentyps.

- **Alias**: Einen Alias eines Standarddatentyps (z.B. INT) erzeugen.
- Array [▶ 110]: Ein benutzerdefiniertes Array erstellen.
- Enumeration [▶ 110]: Eine benutzerdefinierte Aufzählung erstellen.
- Struct [▶ 111]: Eine benutzerdefinierte Struktur erzeugen.
- Interface [▶ 112]: Eine neue Schnittstelle erzeugen.

Datentyp auswählen

Select: Datentyp auswählen - hierbei kann es sich um Basisdatentypen von TwinCAT oder um benutzerdefinierte Datentypen handeln.
Es sind Datentypen äquivalent zu den SPS-Datentypen definiert (wie TIME, LTIME usw.). Siehe Datentypen der SPS für weitere Auskünfte.

Description: Den Typ als Zeiger, Referenz oder Wert mittels entsprechender Auswahl definieren.

- Normaler Typ
- Zeiger
- Zeiger auf Zeiger
- Zeiger auf Zeiger auf Zeiger
- eine Referenz

Typinformation

- **Namespace:** Für ausgewählten Datentyp definiert.
- **GUID**: Eindeutige ID des ausgewählten Datentyps.

Optionale Datentypeinstellungen

Size [Bits]: Größe in Bits (weiße Felder) und in „Byte.Bit“ Notation (graue Felder). Für x64-Plattform kann eine andere Größe festgelegt werden.

C/C++ Name: Im generierten C++ Code verwendeter Name. Der TMC Code Generator wird die Deklaration nicht generieren, sodass benutzerdefinierter Code für diesen Datentyp bereitgestellt werden kann. Darüber hinaus kann für x64 ein anderer Name festgelegt werden.

Unit: Eine Einheit der Variablen.

Comment: Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist.

Hide sub items: Wenn der Datentyp über Unterelemente verfügt, dann wird der Systemmanager keinen Zugriff auf die Unterelemente gewähren. Dies sollte z. B. im Fall großer Arrays verwendet werden.

Persistent (even if unused): Persistenter Typ im globalen Typsystem (vgl. System->Type System->Data Types).

Optionale Standardeinstellungen

Die Standardeinstellungen können in Funktion des Datentyps definiert werden.

Optionale Eigenschaften

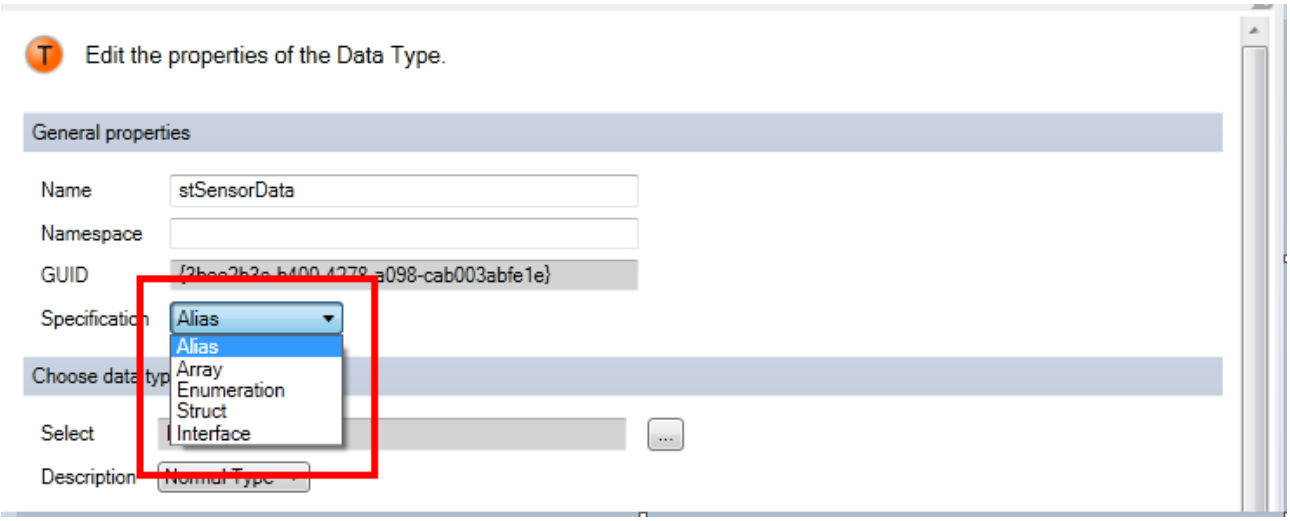
Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Datentyps.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

Datentypausblendungen

Aufgelistete GUIDs verweisen auf Datentypen, die von diesem Datentypen ausgeblendet werden. Normalerweise werden die GUIDs vorheriger Versionen dieses Datentyps hier bei jeder Änderung automatisch eingefügt.

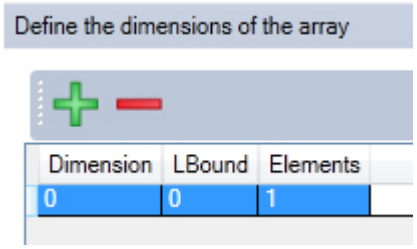
11.3.3.5 Spezifikation

In diesem Abschnitt wird die Spezifikation von Datentypen beschrieben.



11.3.3.5.1 Array

Array: Ein benutzerdefiniertes Array erstellen.



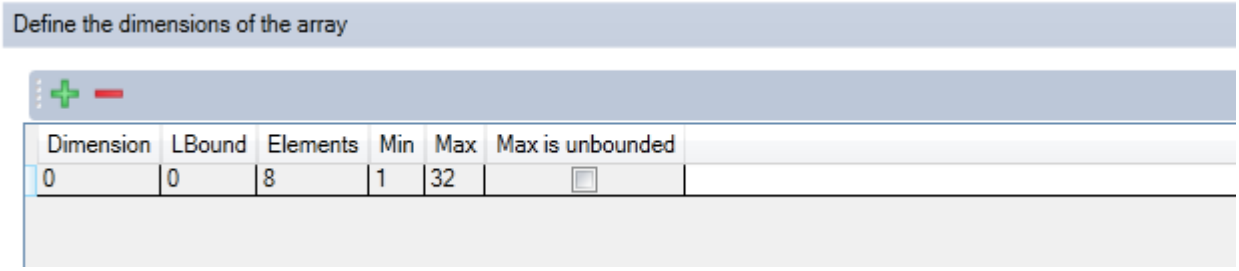
Es wird ein neuer Dialog eingeblendet, um Array-Elemente hinzuzufügen (**+**) oder zu entfernen (**-**).

Dimension: Dimension des Arrays.

LBound: Linke Grenze des Arrays (Standardwert = 0).

Elemente: Menge der Elemente.

Dynamische Arrays für Parameter und Datenzeiger



Im Falle von Parameter [▶ 116] und Datenzeigern [▶ 130] unterstützt TwinCAT 3 Arrays mit dynamischer Größe.

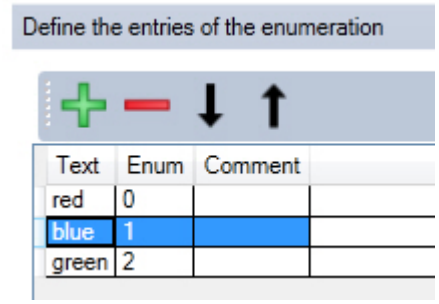
Min: Mindestgröße des Arrays.

Max: Maximale Größe des Arrays.

Max is unbounded: Zeigt an, dass es keine obere Grenze für die Array-Größe gibt.

11.3.3.5.2 Enum

Enumeration: Eine benutzerdefinierte Aufzählung erstellen.



Es wird ein neuer Dialog eingeblendet, um ein Element hinzuzufügen (**+**) oder zu entfernen (**-**). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.

HINWEIS

Eindeutige Namen für Aufzählungselemente erforderlich

Beachten Sie, dass die Aufzählungselemente eindeutig benannt sein müssen, da ansonsten der generierte C++ Code ungültig ist.

Text: Aufzählungselement

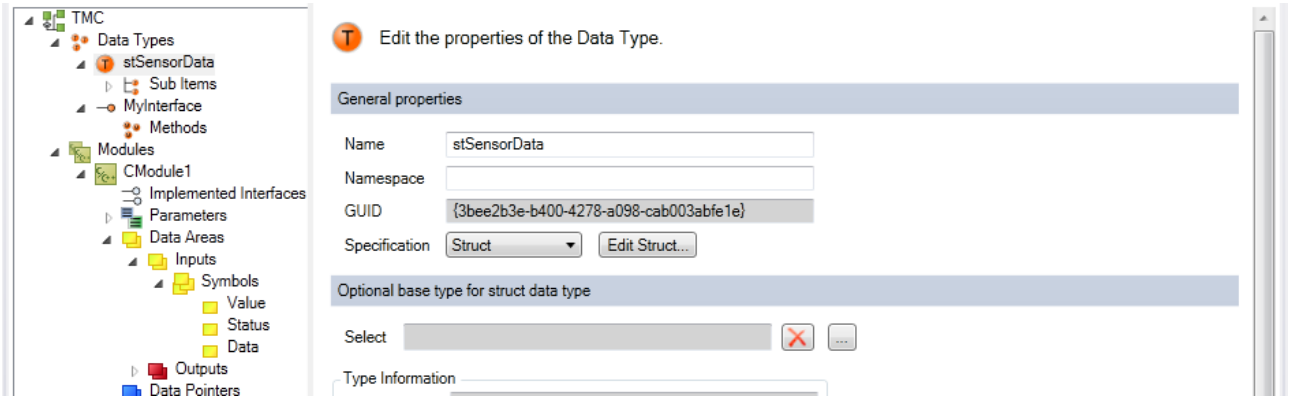
Enum: Geeigneter Integer-Wert.

Comment: Optionaler Kommentar.

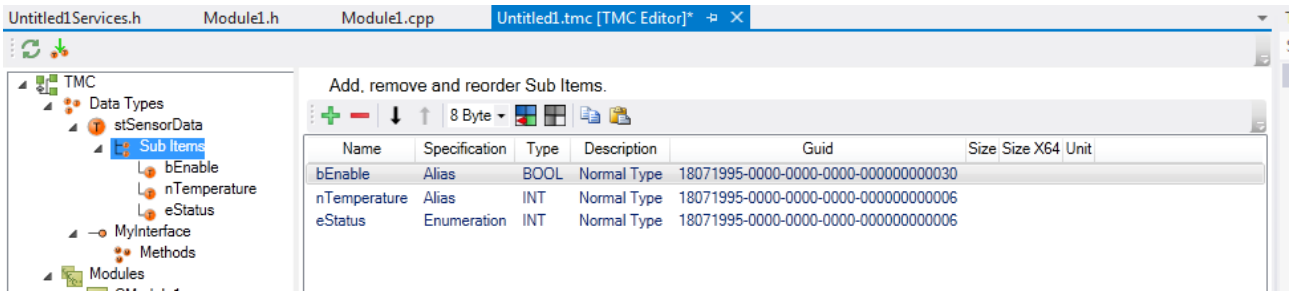
11.3.3.5.3 Struct

Struct: Eine benutzerdefinierte Struktur erstellen.

Wählen Sie den **Sub Items**-Knoten oder klicken Sie auf die **Edit Struct**-Schaltfläche, um zu dieser Tabelle zu wechseln:



Es wird ein neuer Dialog eingeblendet, um ein Element hinzuzufügen (**+**) oder zu entfernen (**-**). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.



Name: Name des Elements.

Specification: Ein Struct kann Alias, Arrays oder Aufzählungen enthalten.

Type: Typ der Variablen.

Size: Größe und Offset des Unterelements.

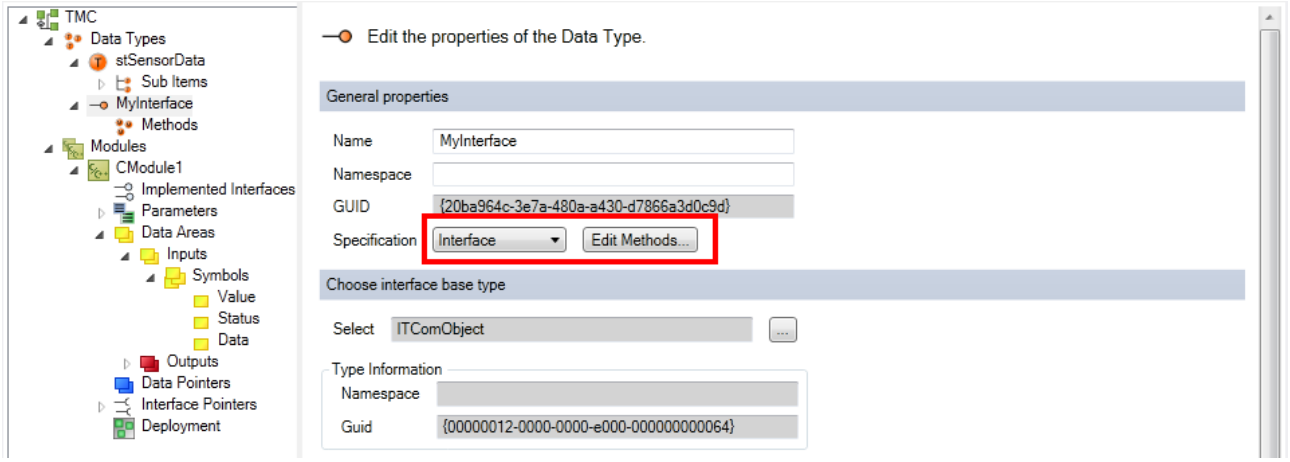
Size X64: Andere Größe für x64 Plattform wird zusätzlich bereitgestellt.

Unit: Optionale Einheit.

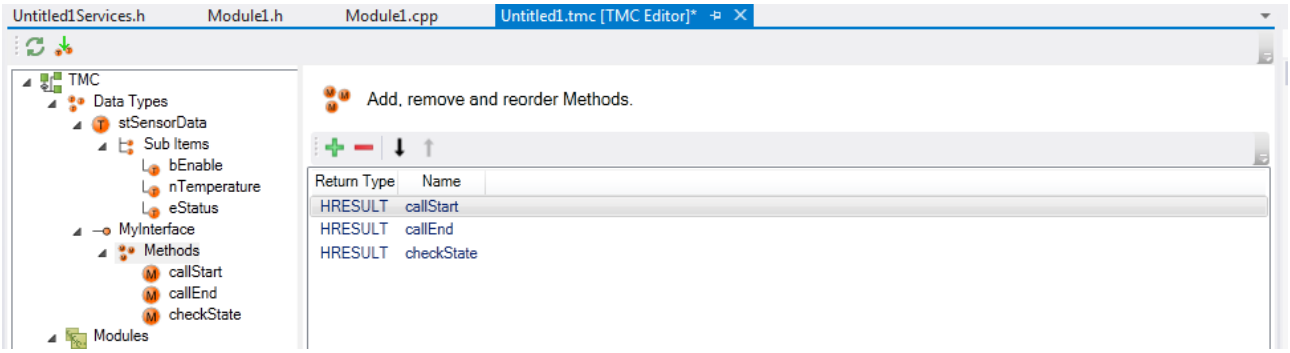
Durch Auswahl des Datentyps oder Doppelklick auf den Tabelleneintrag werden die Details der Konfigurationsseite des Unterelements eingeblendet. Ähnlich wie Datentypeigenschaften [▶ 107].

11.3.3.5.4 Schnittstellen

Interfaces: Eine benutzerdefinierte Schnittstelle erstellen.

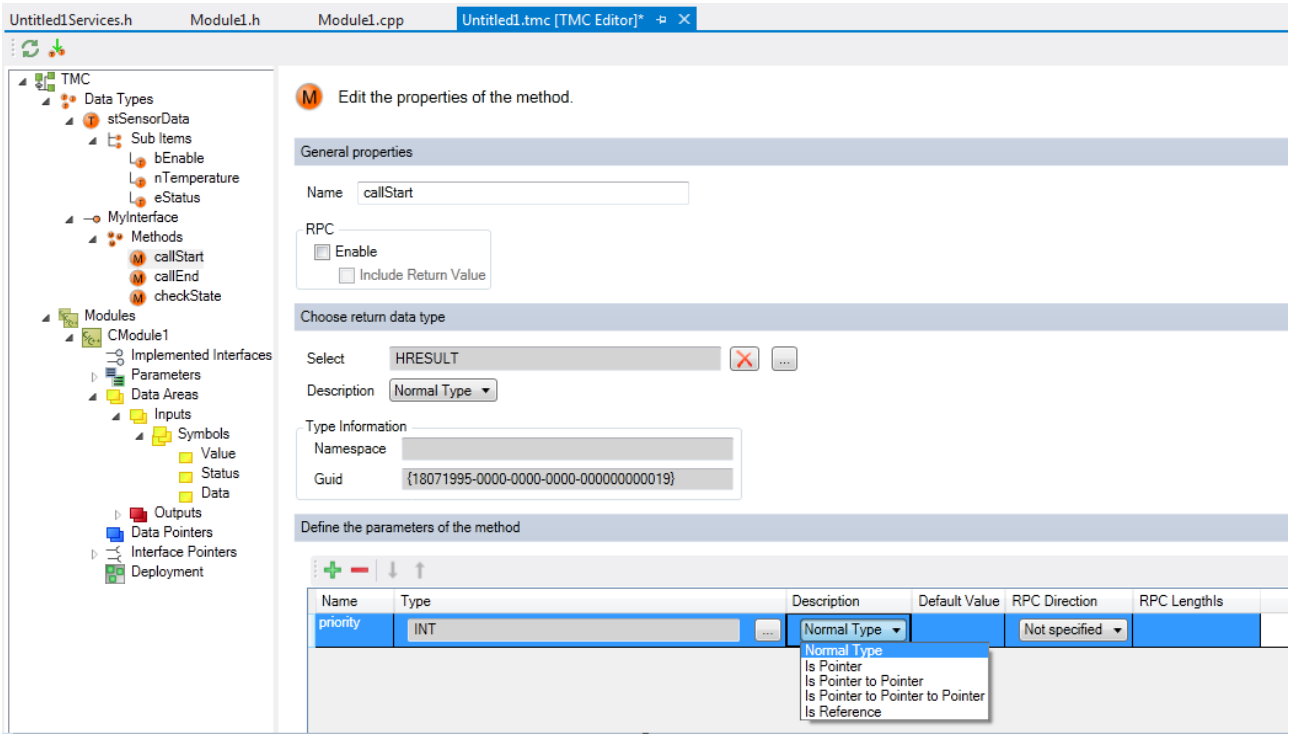


Wählen Sie den **Methods**-Knoten oder klicken Sie auf die **Edit Methods**-Schaltfläche, um zu dieser Tabelle zu wechseln:



Parameter der Methode

Wählen Sie den Knoten der Methode oder doppelklicken Sie auf den Eintrag, um die Details der Methode einzusehen.



Name: Der Name der Methode.

RPC enable: Freigabe von Remoteprozeduraufrufen von außerhalb dieser Methode.

Include Return Value: Freigabe der Weitergabe des Rückgabewerts der Methode.

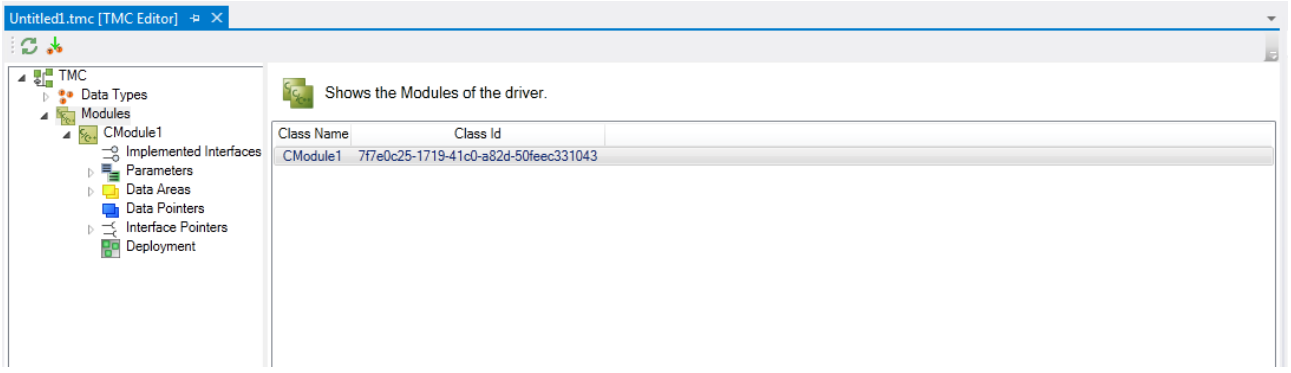
Die Felder entsprechen denjenigen der Datentypeigenschaften [▶ 107].

Parameter der Methode definieren

- **Name:**
- **Type:** Bekannt aus den Datentypeigenschaften [▶ 107].
- **Description:** Bekannt aus den Datentypeigenschaften [▶ 107].
- **Default Value:** Standardwert dieses Parameters; es sind nur Zahlen erlaubt.
- **RPC-Direction:** Wie im Falle von SPS-Funktionsblöcken kann jeder Parameter entweder IN, OUT oder INOUT sein. Darüber hinaus kann er als NONE definiert werden, damit dieser Parameter bei Remoteprozeduraufrufen (RPC) ignoriert wird.

11.3.4 Module

Modules: Zeigt die Module des Treibers.

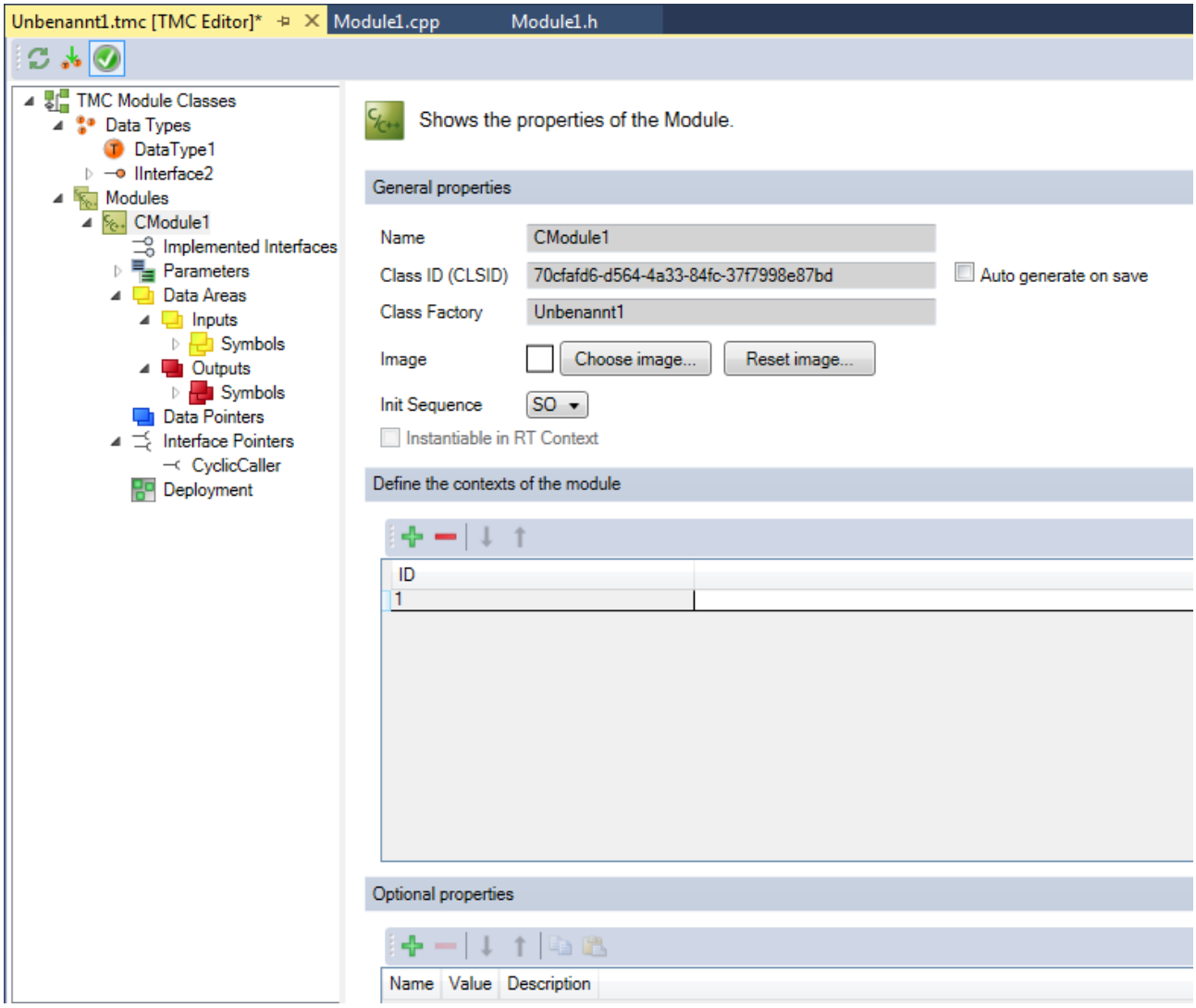


Class Name: Name des Moduls.

Class ID: Eindeutige ID des Moduls.

Module Properties:

Ein Klick auf den Knoten im Baum oder der Zeile in der Tabelle öffnet die Moduleigenschaften.



Allgemeine Eigenschaften

Name: Name des Moduls.

Class ID: Eindeutige ID des Moduls.

Auto generate on save: Ermöglicht, dass TwinCAT die ClassID anhand der Modul-Parameter beim Speichern generiert. Die Änderung der ClassID führt beim Importieren der binären Module dazu, dass die entsprechenden ClassIDs angepasst werden müssen. Somit kann TwinCAT die Interface-Änderung erkennen.

Choose Image: Ein 16x16 Pixel-Bitmap-Symbol einfügen.

Reset image: Modulbild auf Standardwert zurücksetzen.

Init sequence: Start der Zustandsmaschine. Die Auswahlmöglichkeiten mit „late“ im Namen sind intern. (Siehe Objekt [▶ 136] des Instance Configurators für weitere Auskünfte.)

Instantiable in RT Context: Zeigt an, ob dieses Modul unter Echtzeitkontext instanziiert werden kann, siehe TwinCAT Module Klassenassistent [▶ 90]

Die Kontexte des Moduls festlegen

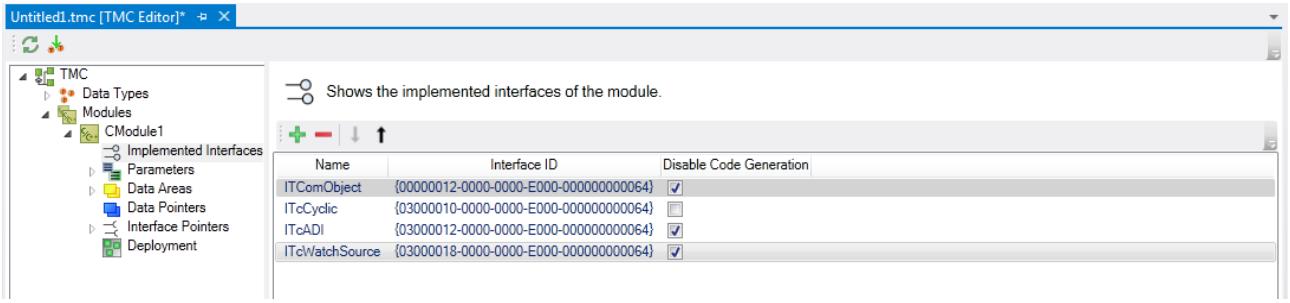
Sie können für das Modul Kontexte hinzufügen (**+**) oder entfernen (**-**). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.
Die Kontext-Id muss ein Integer verschieden von 0 sein.

Optionale Eigenschaften

Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Moduls.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

11.3.4.1 Implementierte Schnittstellen

Implemented Interfaces: Die implementierten Schnittstellen des Moduls ansehen und bearbeiten.

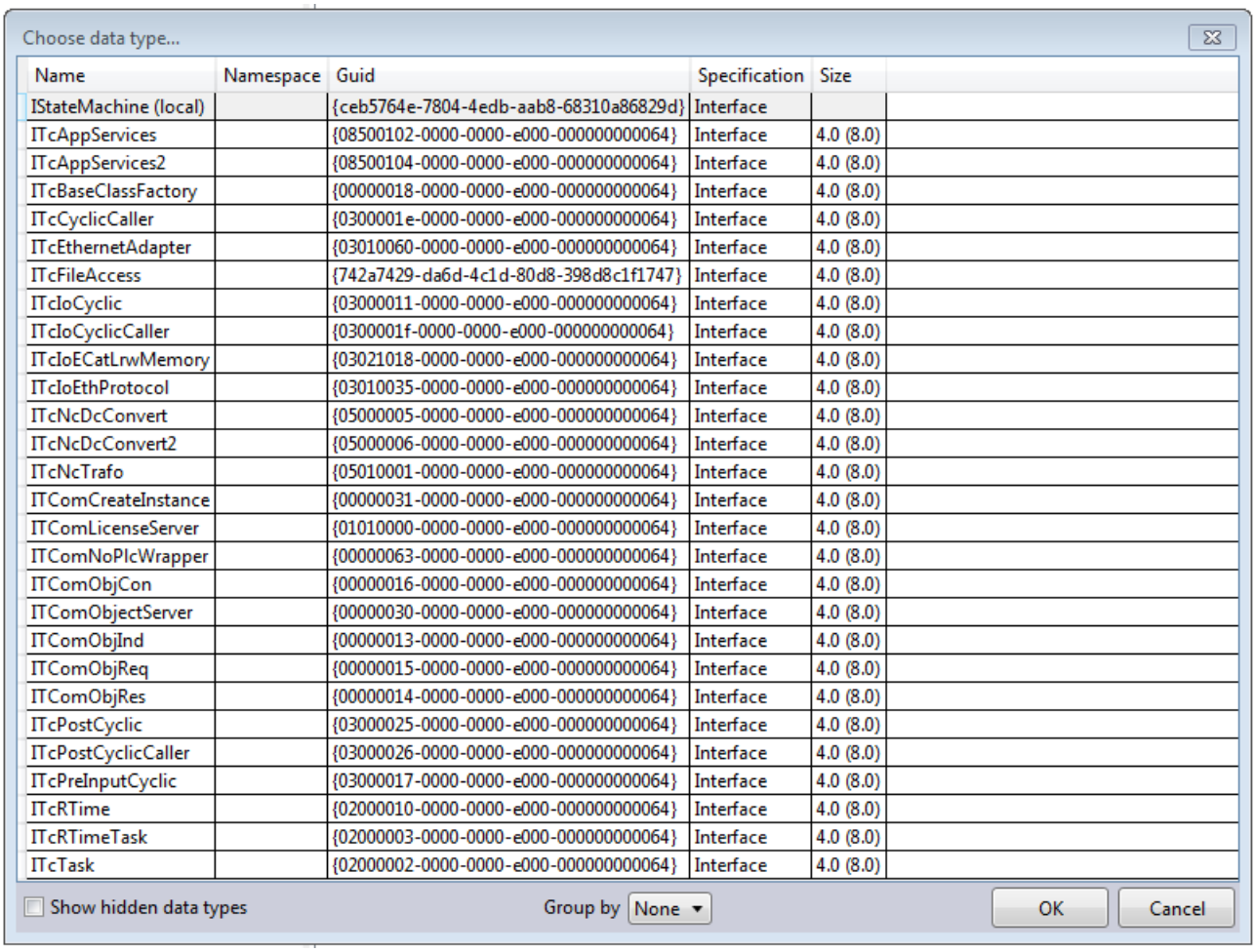


Name: Name der Schnittstelle.

Interface ID: Eindeutige ID der Schnittstelle.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

Sie können für das Modul Kontexte hinzufügen (**+**) oder entfernen (**-**). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.



11.3.4.2 Parameter

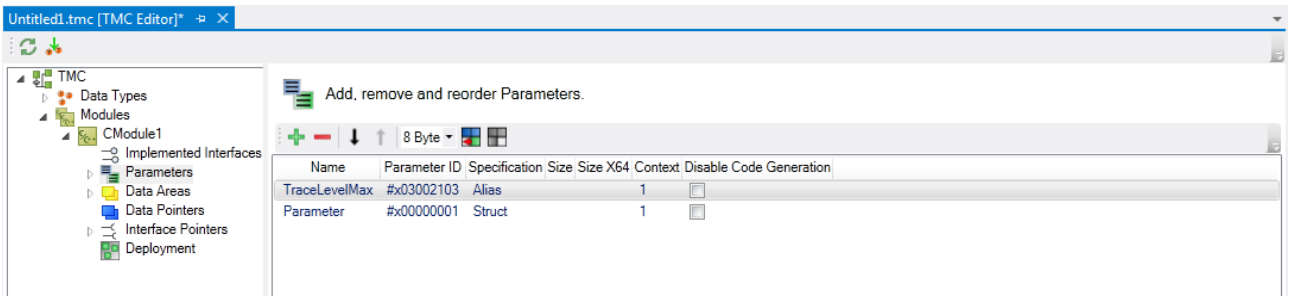
Eine TcCOM-Modulinstanz wird durch verschiedene Parameter konfiguriert.

TwinCAT unterstützt drei Arten von Parameter IDs (PTCID) im Abschnitt Configure the parameter ID [▶ 121].

- „User defined“ (Standardwert neuer Parameter): Es wird eine eindeutige Parameter-ID generiert, die im Nutzercode oder in der Instanzkonfiguration zur Festlegung des Parameters verwendet werden kann.
- „Predefined...“: Spezielle PTCIDs bereitgestellt vom TwinCAT 3 System (z. B. TcTraceLevel).
- „Context based…“: Werte des konfigurierten Kontext [▶ 137]s diesem Parameter automatisch zuweisen. Die ausgewählte Eigenschaft wird für die PTCPID übernommen. Das überschreibt den definierten Standardparameter und den Instanzkonfigurationsparameter („Parameter (Init)“).

Es folgt eine ausführliche Beschreibung der Parameter und ihrer Konfiguration.

Parameters: Zeigt die implementierten Parameter des Moduls.



Symbol	Funktion
	Einen neuen Parameter hinzufügen
	Löscht den ausgewählten Typ
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben
	Byte Alignment auswählen
	Ausgewählten Datentyp ausrichten
	Datenformat des ausgewählten Datentyps zurücksetzen

Name: Name der Schnittstelle.

Parameter ID: Eindeutige ID des Parameters.

Specification: Datentyp des Parameters.

Size: Größe des Parameters. Für x64 sind andere Größen möglich.

Context: Kontext-ID des Parameters.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

11.3.4.2.1 Parameter hinzufügen / bearbeiten / löschen

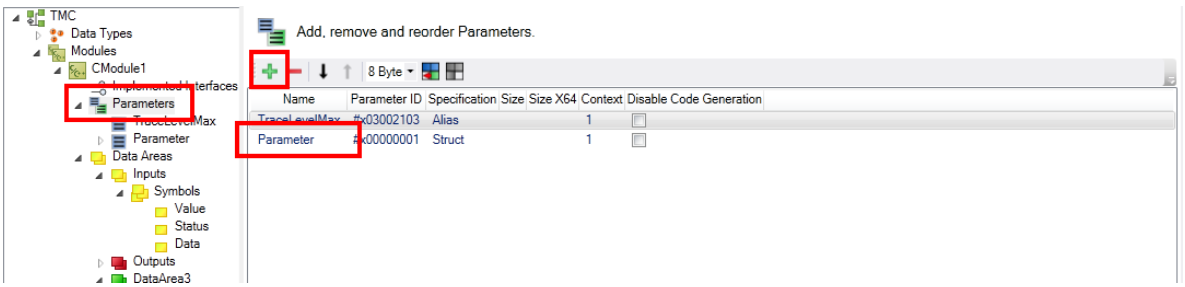
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Eigenschaften und Funktionalitäten einer TwinCAT-Klasse hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

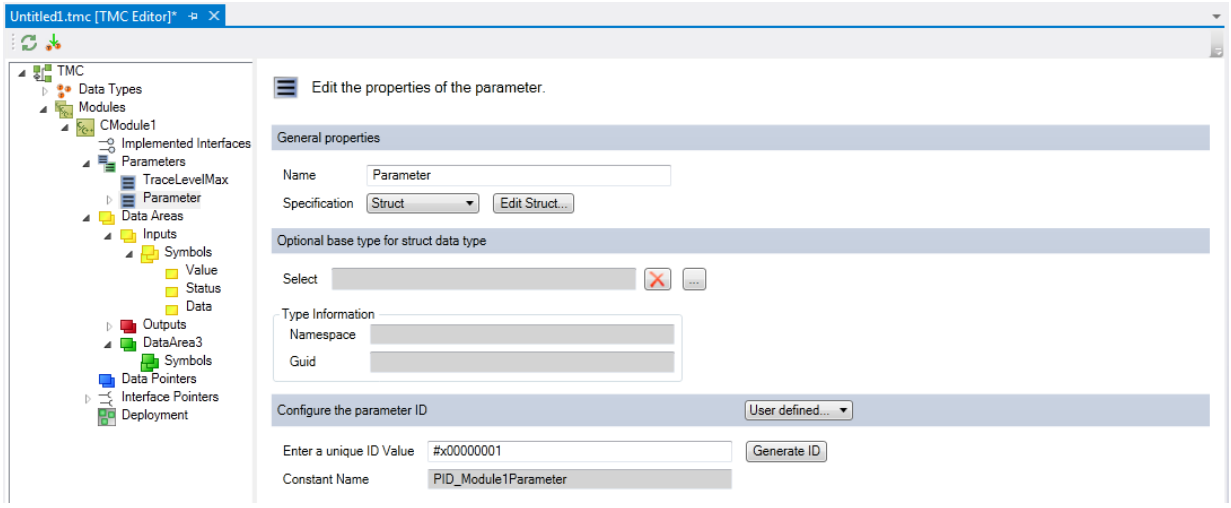
- Schritt 1: Einen neuen Parameter [▶ 117]in der TMC-Datei erstellen.
- Schritt 2: Den TwinCAT TMC Code Generator [▶ 118] starten, um Code für die Modulbeschreibung in der TMC-Datei zu erzeugen.
- Schritt 3: Übergänge der Zustandsmaschine [▶ 119] beachten

Schritt 1: Neuen Parameter erzeugen

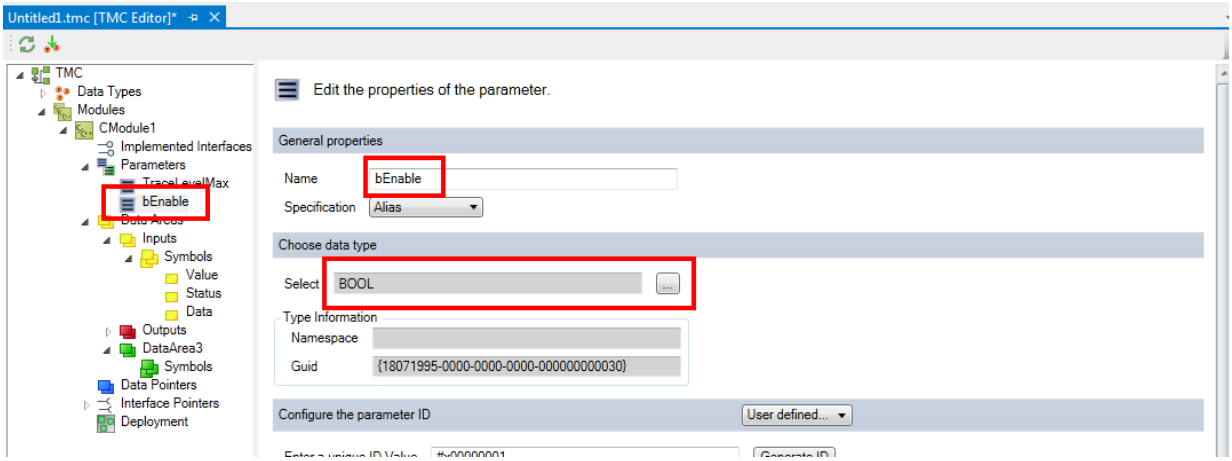
1. Wählen Sie nach dem Starten des TMC Editors das Ziel **Parameters** aus.
2. Erweitern Sie die Liste der Parameter durch Klick auf die **+** Schaltfläche **Add a new parameter** um einen neuen Parameter.
   ⇨ Daraufhin wird ein neuer „Parameter“ als neuer Eintrag aufgeführt:



3. Wählen Sie **Parameter** im linken Baum oder doppelklicken Sie auf den rot markierten „Parameter3“ oder wählen Sie den Knoten im Baum, um Details zum neuen Parameter zu erhalten.



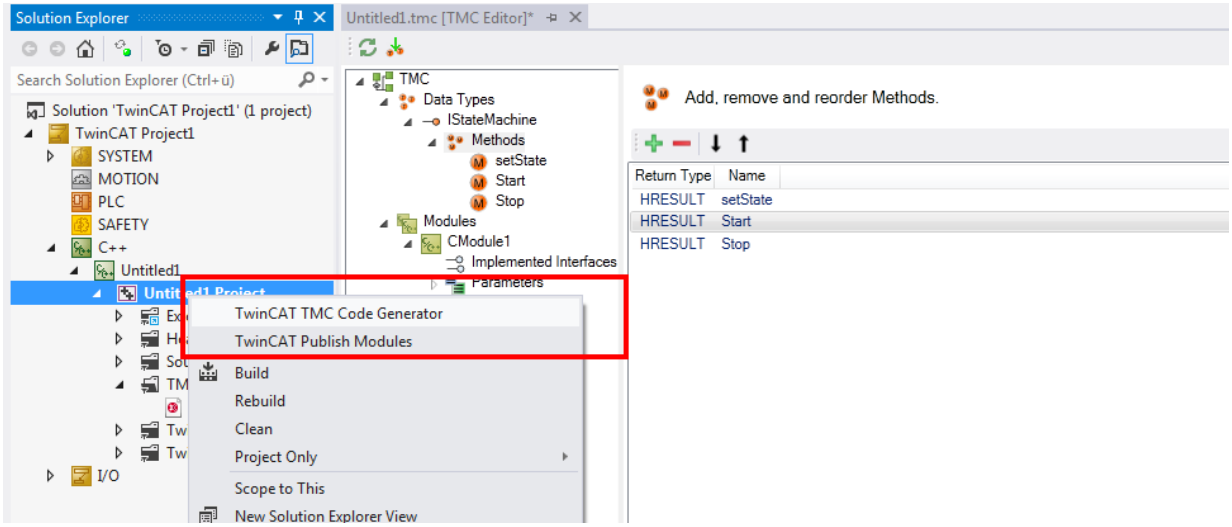
4. Konfigurieren Sie den Parameter sowie die Datentypen [▶ 96].
5. Geben Sie diesem einen aussagekräftigeren Namen - in diesem Beispiel „bEnable“ - und wählen den Datentyp „BOOL“.



6. Speichern Sie Ihre in der TMC-Datei vorgenommenen Änderungen.

Schritt 2: Starten Sie den TwinCAT TMC Code Generator um einen Code für die Modulbeschreibung zu erzeugen.

1. Klicken Sie mit der rechten Maustaste auf Ihre Projektdatei und wählen **TwinCAT TMC Code Generator**, um den Parameter in Ihrem Quellcode zu erhalten:



⇨ Sie sehen die Parameterdeklaration in der Header-Datei „Module1.h“ des Moduls.

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

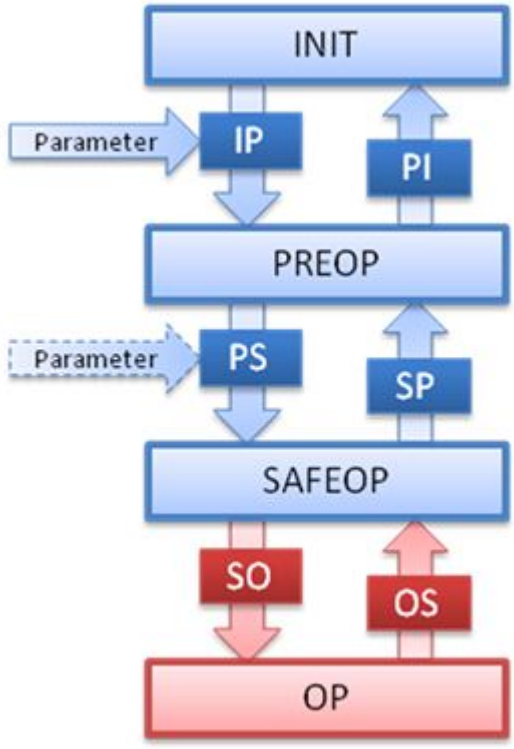
⇨ Die Implementierung des neuen Parameters finden Sie in den get und set Methoden der Modulklasse „module1.cpp“.

```
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()
```

Um einen weiteren Parameter hinzuzufügen, verwenden Sie erneut den TwinCAT TMC Code Generator.

Schritt 3: Übergänge der Zustandsmaschine

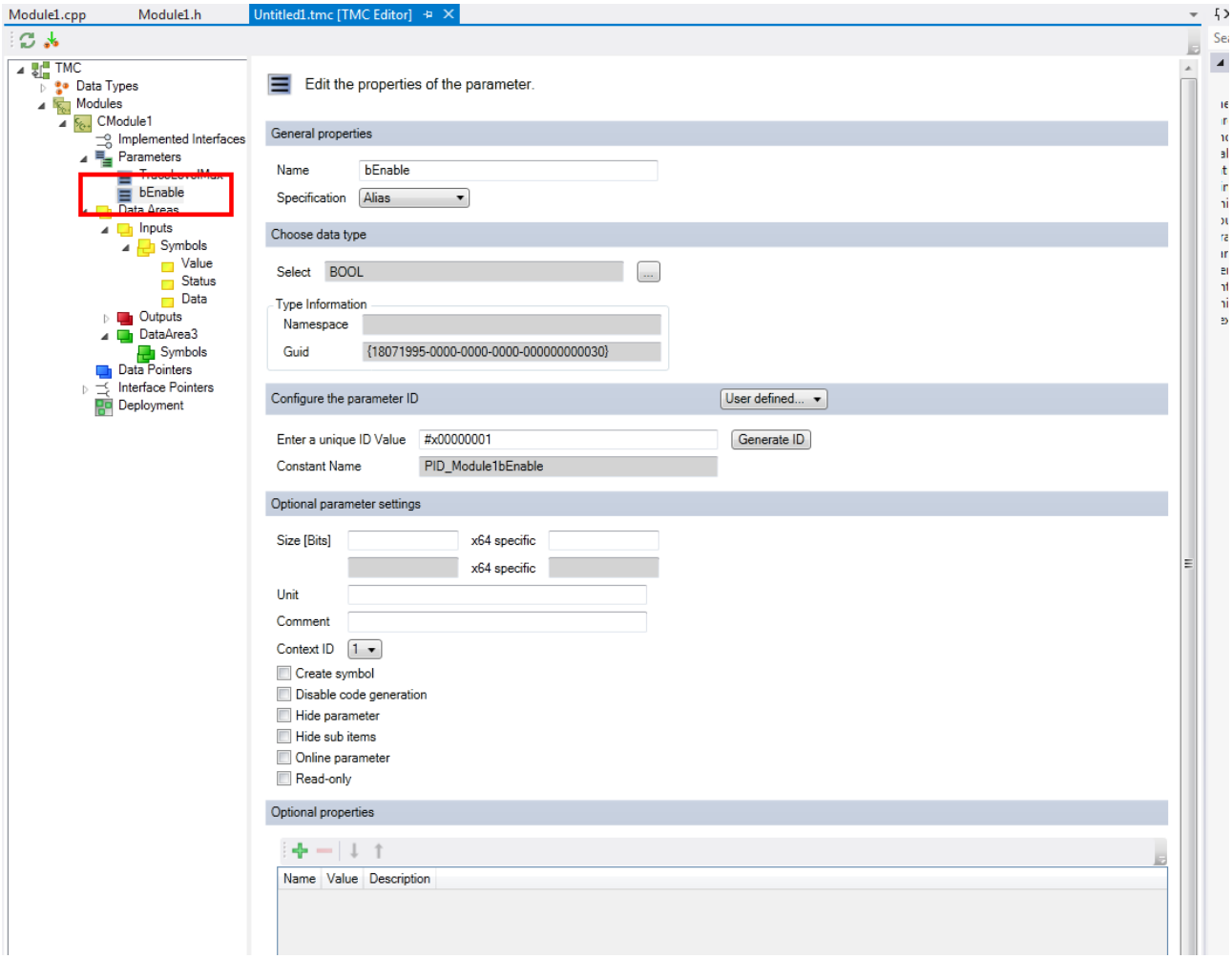
Beachten Sie die verschiedenen Zustandsübergänge Ihrer Zustandsmaschine [▶ 48]:



Die Parameter werden beim Übergang Init->Preop und gegebenenfalls Preop->Safeop festgelegt.

11.3.4.2.2 Parametereigenschaften

Parameter properties: Die Eigenschaften des Parameters bearbeiten.



Allgemeine Eigenschaften

Name: Name der Schnittstelle.

Specification: Datentyp des Parameters, siehe Spezifikation.

Datentyp auswählen

Select: Datentyp auswählen.

Typinformation

- **Namespace:** Benutzerdefinierter Namensraum des Datentyps.
- **GUID**: Eindeutige ID des Datentyps.

Enter a unique ID Value: Einen eindeutigen ID-Wert eingeben, siehe Parameter [▶ 116].

Constant Name: Quellcodename der Parameter-ID.

Optionale Parametereinstellungen

Size [Bits]: Berechnete Größe in Bits (weiße Felder) und in „Byte.Bit“-Notation (graue Felder). Für x64 wird eine besondere Größenkonfiguration bereitgestellt.

Unit: Eine Einheit der Variablen.

Comment: Kommentar, der z. B. im Instanzenkonfigurator sichtbar ist.

Context ID: Kontext, welcher beim Zugriff auf den Parameter durch ADS verwendet wird.

Create Symbol: Standardeinstellung für ADS-Symbolerstellung.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

Hide parameter: Umschalten zwischen Parameter ein-/ausblenden in der Systemmanager-Ansicht.

Hide sub items: Wenn der Datentyp über Unterelemente verfügt, dann wird der Systemmanager keinen Zugriff auf die Unterelemente gewähren. Dies sollte z. B. im Falle großer Arrays verwendet werden.

Online parameter: Als Online-Parameter festlegen.

Read-only: Wechseln zu nur Lesezugriff für Systemmanager.

Optionale Eigenschaften

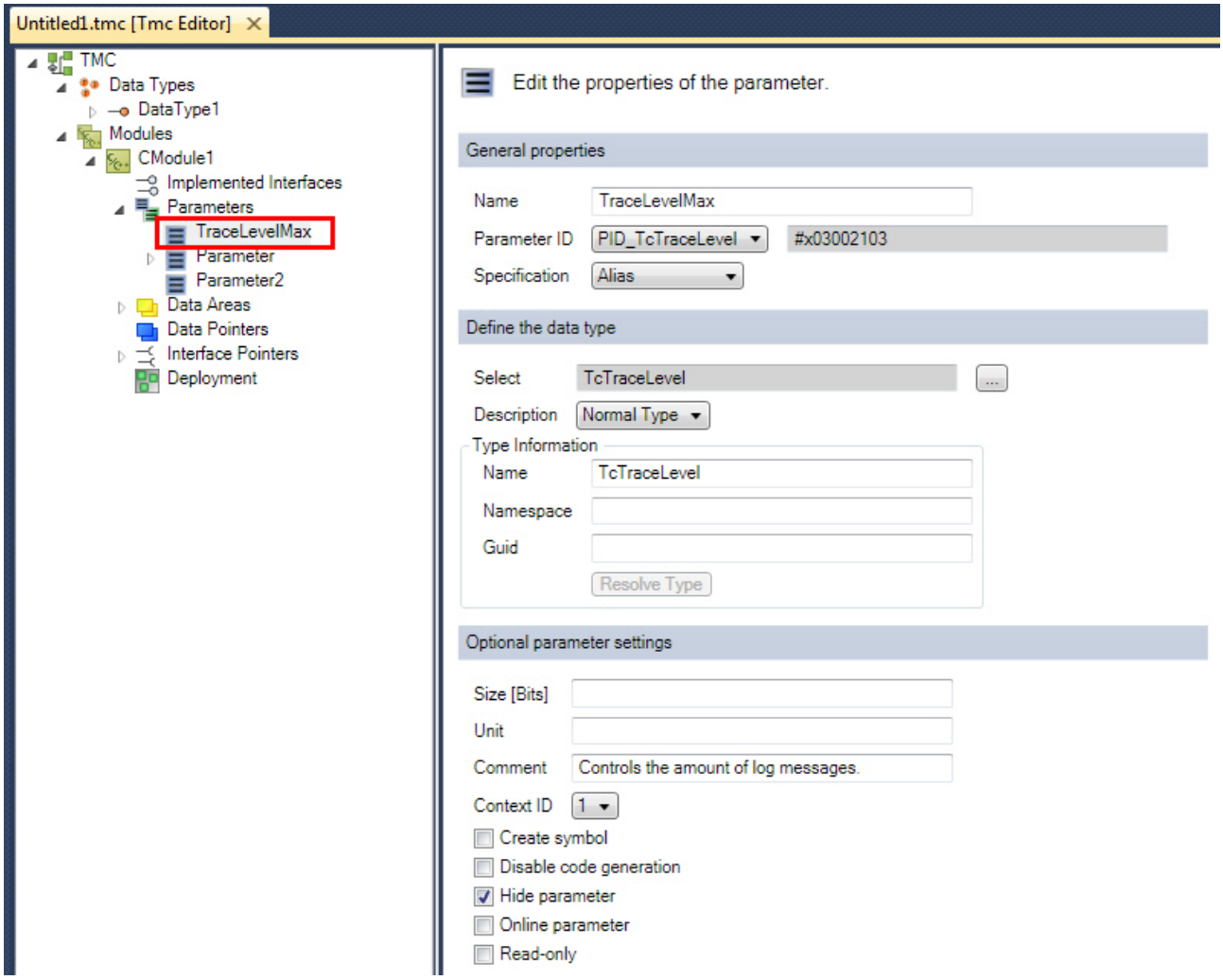
Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Parameters.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

11.3.4.2.3 TraceLevelMax

TraceLevelMax: Parameter der den Tracelevel festlegt.
Dies ist ein vordefinierter Parameter, den die meisten TwinCAT-Modulvorlagen bereitstellen (außer die leere TwinCAT-Modulvorlage).

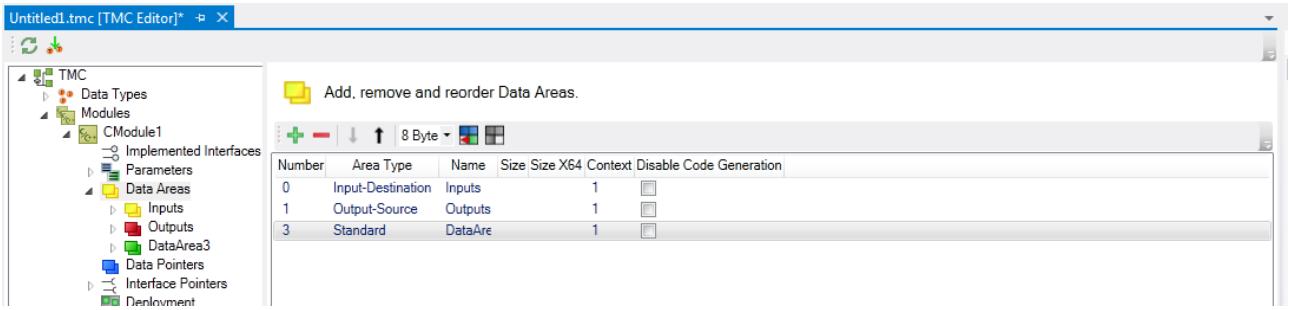
Die Einstellungen dieses Parameters sollten nicht geändert werden.

Siehe Modul-Nachrichten zum Engineering (Logging / Tracing) [▶ 223]



11.3.4.3 Datenbereiche

Data Areas: Dialog zum Bearbeiten der Datenbereiche Ihres Moduls.



Symbol	Funktion
	Einen neuen Datenbereich hinzufügen
	Löscht den ausgewählten Datenbereich
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben
	Byte Alignment auswählen
	Ausgewählten Datentyp ausrichten
	Datenformat des ausgewählten Bereichs zurücksetzen

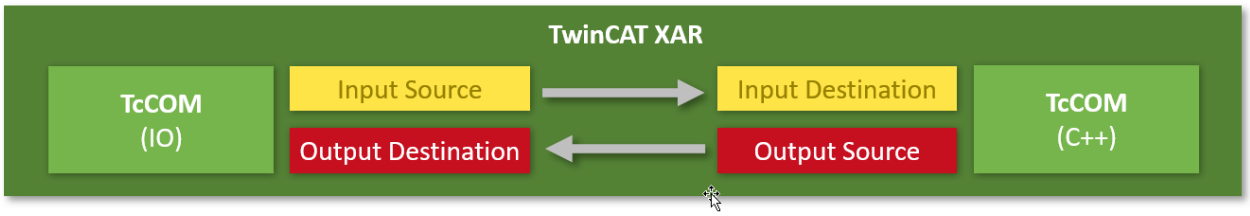
HINWEIS

Rekursion bei Festlegung eines Alignments

Bei der Festlegung des Alignment eines Datenbereichs wird dieses für alle seine Elemente (Symbole und auch deren Unterelemente) zu Grunde gelegt. Benutzerdefiniertes Alignment wird überschrieben.

Number: Nummer des Datenbereichs.

Type: Definiert den Zweck und die Lage des Datenbereichs.
Die Datenbereiche zum Mapping (zu anderen Modulen, wie auch I/O) werden in Input und Output sowie auch in Source und Destination klassifiziert. Die Grafik verdeutlicht dieses:



Name: Name des Datenbereichs.

Size: Größe des Parameters, für x64 sind andere Größen möglich.

Context: Zeigt die Kontext-ID an.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

11.3.4.3.1 Datenbereiche und Variablen hinzufügen / bearbeiten / löschen

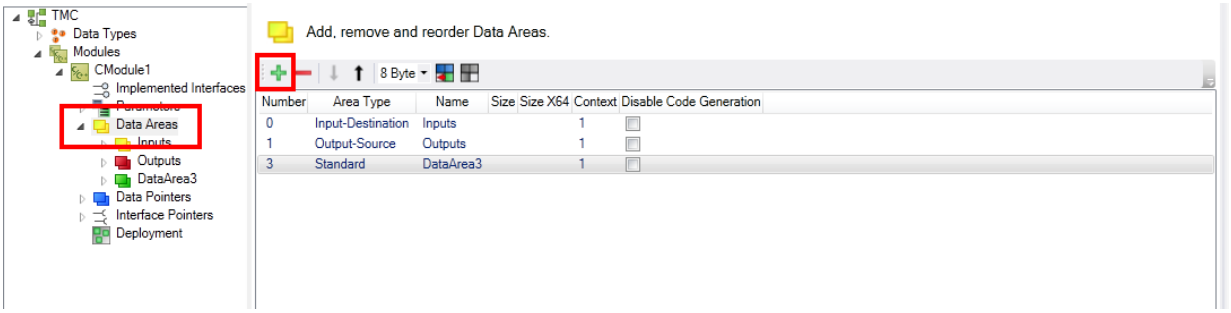
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Eigenschaften und Funktionalitäten einer TwinCAT-Klasse hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

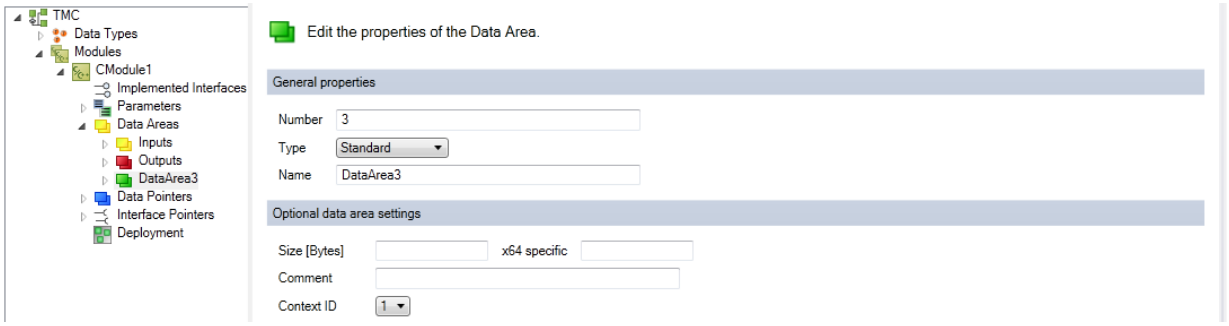
- Einen neuen Datenbereich in der TMC-Datei erstellen.
- Neue Variablen in einen Datenbereich erzeugen [▶ 129].
- Z. B. den Namen oder Datentyp [▶ 129] von in der TMC-Datei bestehenden Variablen bearbeiten.
- In der TMC-Datei bestehende Variablen löschen [▶ 130].

Einen neuen Datenbereich erzeugen

1. Nach dem Starten des TMC Editors wählen Sie den Knoten **Data Areas** des Moduls aus.
2. Klicken Sie auf die Schaltfläche **+** und erzeugen Sie damit einen neuen Datenbereich.



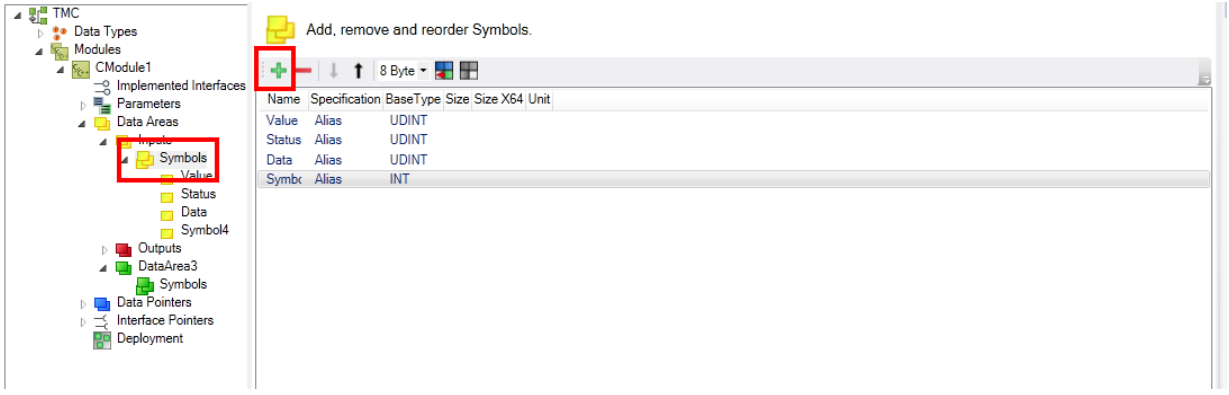
3. Um die Eigenschaften des Datenbereichs zu erhalten, doppelklicken Sie auf die Tabelle oder auf den Knoten.



4. Benennen Sie den Datenbereich um.

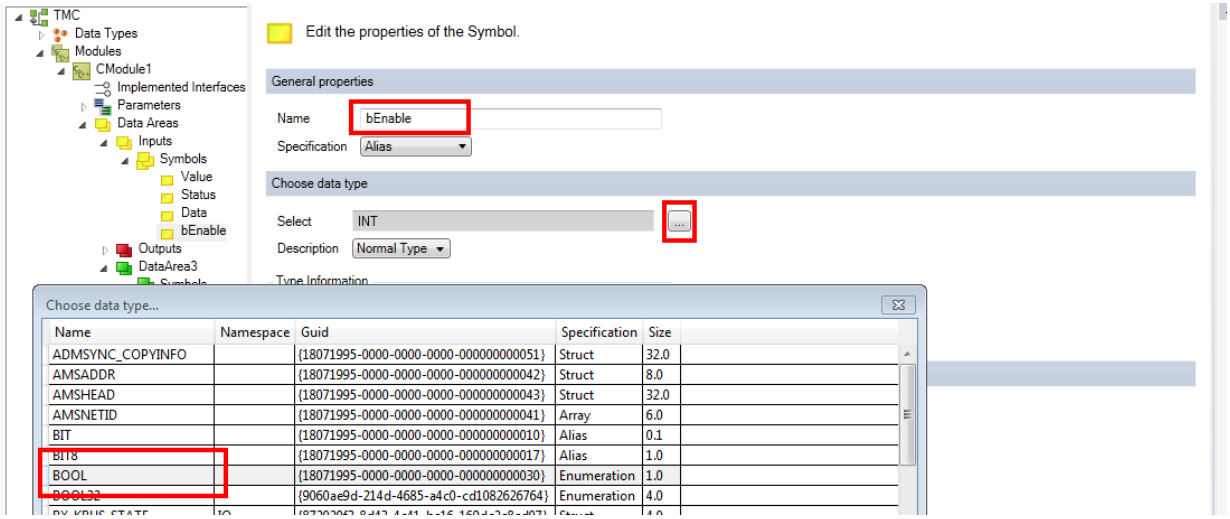
Eine neue Variable erzeugen

1. Wählen Sie den Unterknoten **Symbols** des Datenbereichs aus.
2. Erweitern Sie diesen Datenbereich mit einem Klick auf die Schaltfläche **+** um eine neue Variable. Daraufhin wird „Symbol4“ als neuer Eintrag aufgeführt.



Name oder Datentyp von bestehenden Variablen bearbeiten

1. Wählen Sie den Unterknoten **Symbol4** oder doppelklicken Sie auf die Zeile. Die Variableneigenschaften werden eingeblendet.
2. Geben Sie einen neuen Namen, z.B. „bEnableJob“, ein und ändern den Typ in BOOL.

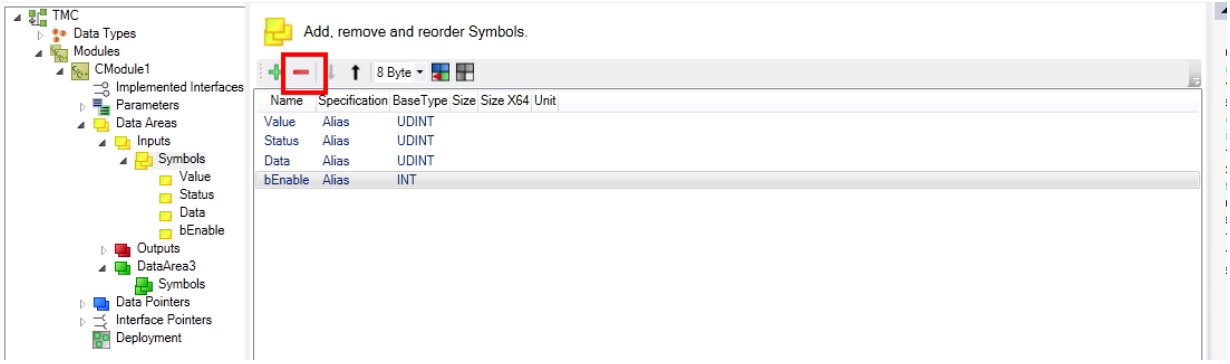


⇨ Die neue Variable „bEnableJob“ im Datenbereich „Input“ wird erzeugt.

Denken Sie daran, den TMC Code Generator erneut auszuführen.

Bestehende Variablen löschen

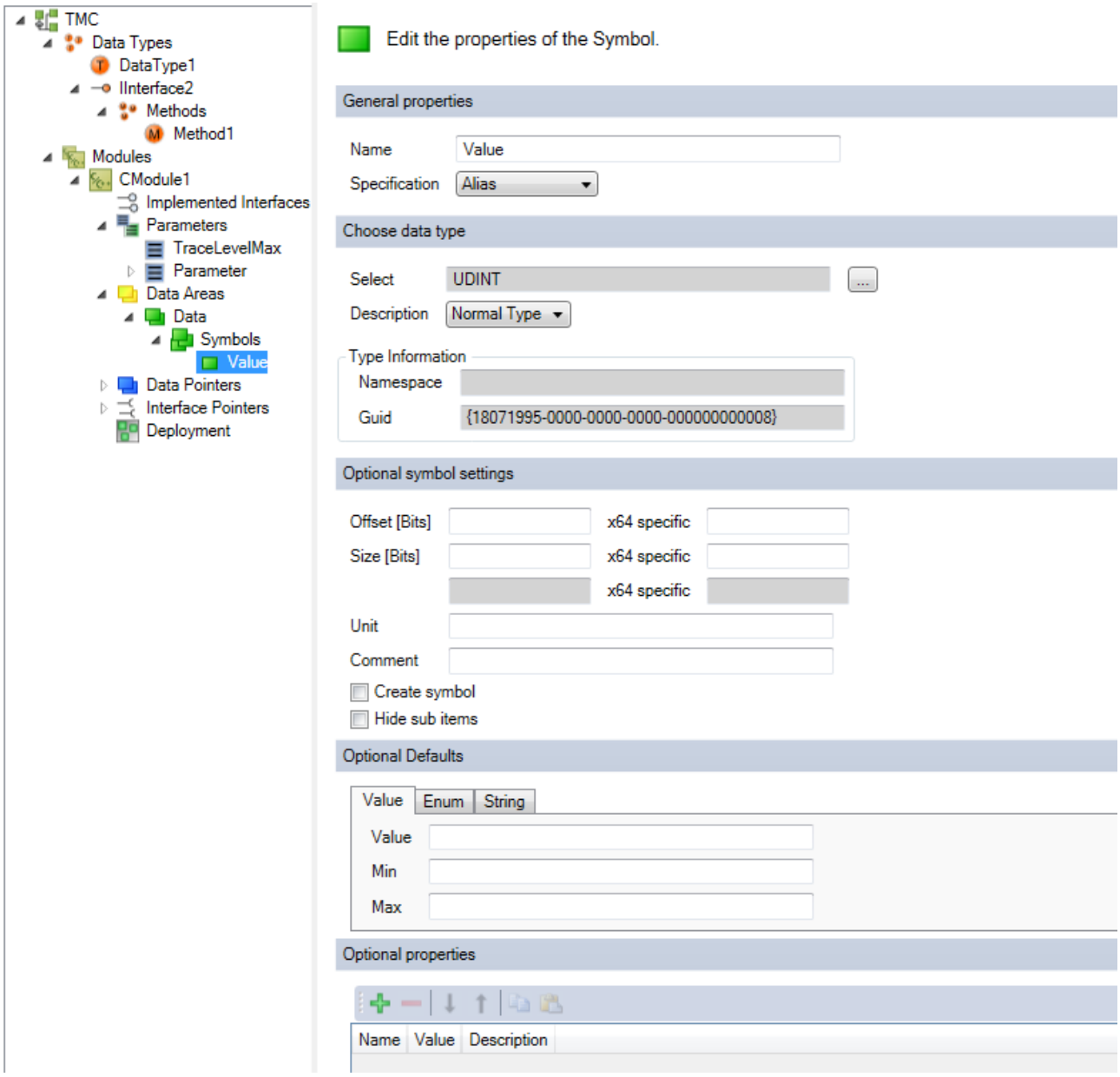
1. Zum Löschen bestehender Variablen im Datenbereich wählen Sie die Variable aus und klicken dann auf das Löschen-Symbol:
   hier im Beispiel wählen Sie „MachineStatus1“ aus und klicken auf **Delete symbol.**



2. Führen Sie den TMC Code Generator erneut aus.

11.3.4.3.2 Datenbereichseigenschaften

Data Areas Properties: Dialog zum Bearbeiten der Datenbereichseigenschaften.



Allgemeine Eigenschaften

Number: Nummer des Datenbereichs.

Type: Definiert den Zweck und die Lage des Datenbereichs. Verfügbar sind:

Verknüpfbare Datenbereiche im Systemmanager:

- Input-Source
- Input-Destination
- Output-Source
- Output-Destination
- Retain-Source (zur Verwendung mit NOV-RAM Speicher, siehe Anhang [▶ 348])
- Retain-Destination (für interne Verwendung)

Weitere Datenbereiche:

- Standard (sichtbar aber nicht verknüpfbar im Systemmanager)
- Internal (für interne Symbolik des Modules; über ADS erreichbare, aber im Systemmanager nicht sichtbare Symbole)
- MArea (für internen Gebrauch)

- Not specified (gleich Standard)

Name: Name des Datenbereichs.

Optionale Parametereinstellungen

Size [Bytes]: Größe in Byte, für x64 wird eine besondere Größenkonfiguration bereitgestellt.

Comment: Optionaler Kommentar, der z. B. im Instanzenkonfigurator sichtbar ist.

Context ID: Kontext-ID aller Symbole dieses Datenbereichs, wird für die Bestimmung des Mappings verwendet.

Data type name: Wenn angegeben, wird ein Datentyp mit dem angegebenen Namen im Typsystem erstellt.

Create Symbol: Standardeinstellung für ADS-Symbolerstellung.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

Optionale Standardeinstellungen

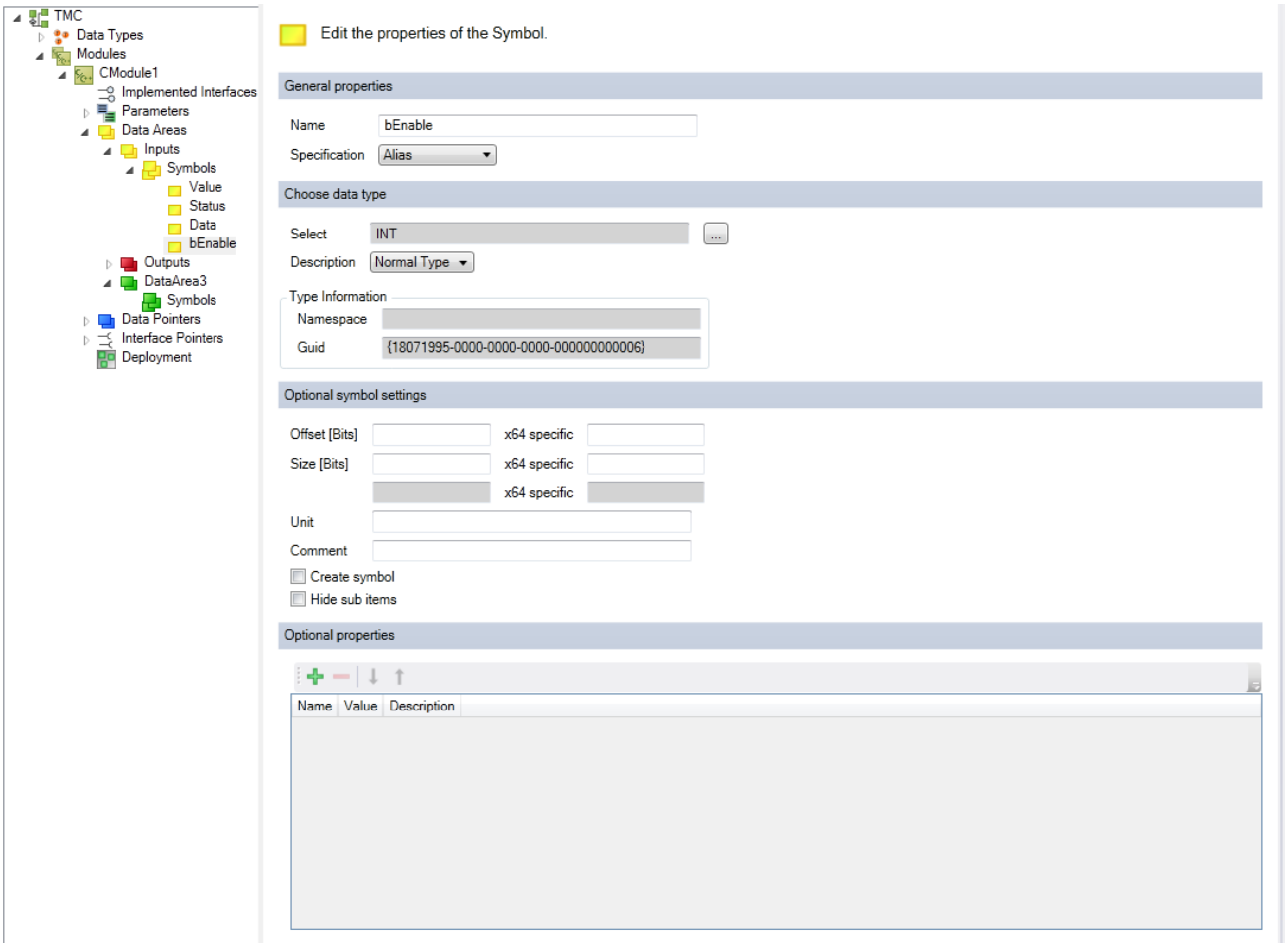
Die Standardeinstellungen können in Funktion des Datentyps definiert werden.

Optionale Eigenschaften

Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Datenbereichs.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

11.3.4.3.3 Symboleigenschaften

Symbols: Dialog für die Bearbeitung der Symbole des Datenbereichs.



Allgemeine Eigenschaften

Name: Name des Symbols.

Specification: Datentyp des Symbols, siehe Datentypeigenschaften [▶ 107].

Datentyp auswählen

Select: Datentyp auswählen - hierbei kann es sich um Basisdatentypen von TwinCAT oder um benutzerdefinierte Datentypen handeln.

Description: Festlegen, ob der Typ folgendes ist:

- Normaler Typ
- Zeiger
- Zeiger auf Zeiger
- Zeiger auf Zeiger auf Zeiger
- eine Referenz

Typinformation

- **Namespace:** Namensraum für ausgewählten Datentyp.
- **GUID**: Eindeutige ID des Datentyps.

Optionale Datentypeinstellungen

Offset [Bits]: Offset des Symbols innerhalb des Datenbereichs, für x64-Plattform kann ein anderes Offset festgelegt werden.

Size [Bits]: Größe in Bits, falls angegeben, für x64-Plattform kann eine andere Größe festgelegt werden.

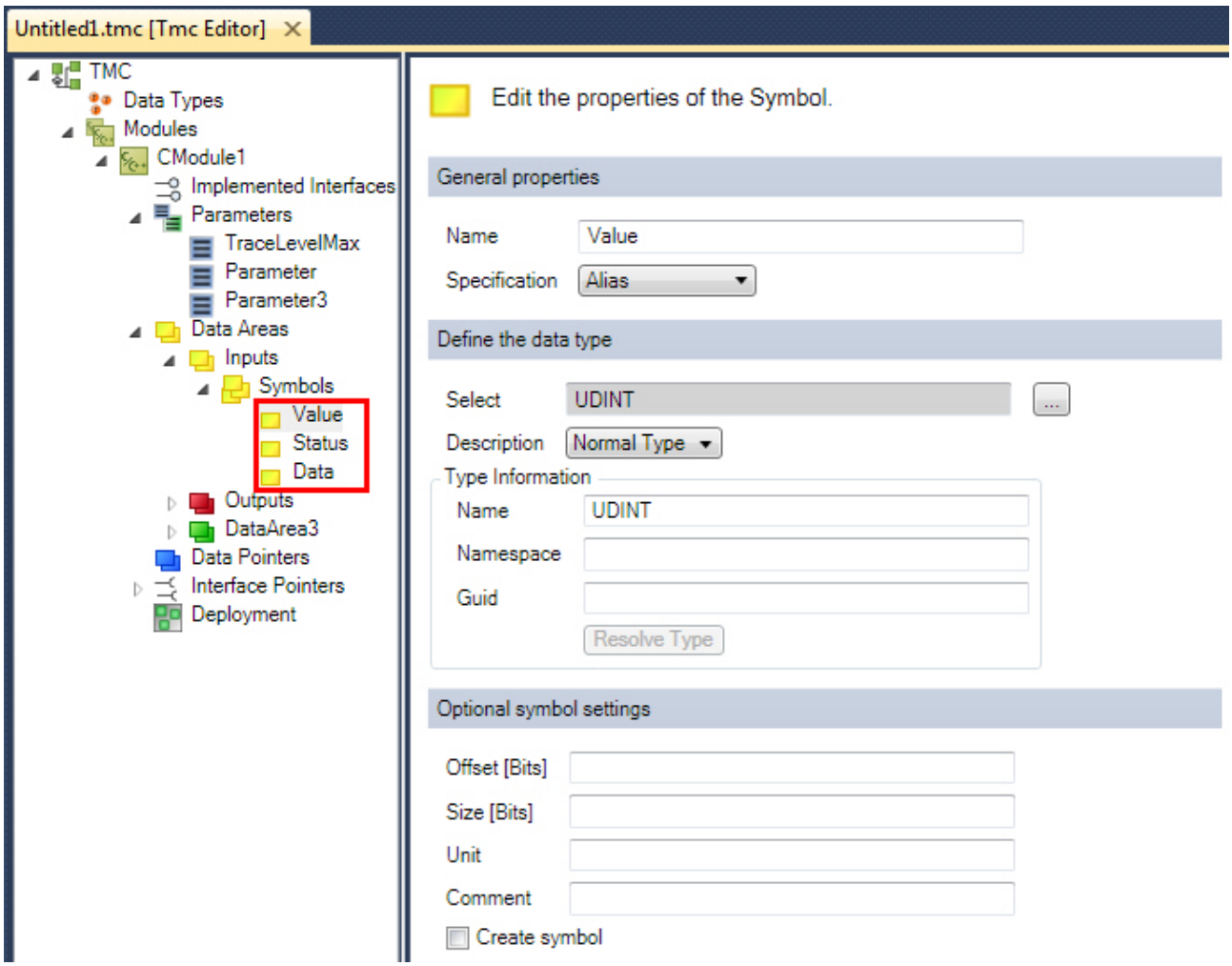
Comment: Optionaler Kommentar, der z. B. im Instanzenkonfigurator sichtbar ist.

Create Symbol: Standardeinstellung für ADS-Symbolerstellung.

Hide sub items: Wenn eine Variable über Unterelemente verfügt, dann wird der Systemmanager keinen Zugriff auf die Unterelemente gewähren. Dies sollte z. B. im Falle großer Arrays verwendet werden.

11.3.4.3.3.1 Symboleigenschaften

Data Areas Symbols Properties: Dialog zum Bearbeiten der Symboleigenschaften.



Allgemeine Eigenschaften

Name: Name der Schnittstelle

Specification: Datentyp des Parameters

Verfügbare Spezifikationen sind:

- **Alias**: Ein Alias eines Standarddatentyps (z.B. INT) erzeugen
- **Array:** Ein benutzerdefiniertes Array erstellen
- **Enumeration:** Eine benutzerdefinierte Aufzählung erstellen
- **Struct:** Eine benutzerdefinierte Struktur erstellen
- **Interface:** Eine neue Schnittstelle erzeugen

Den Datentyp definieren

Select: Datentyp auswählen

Description: Beschreibung festlegen

Typinformation

Name: Name des ausgewählten Standardtyps

Namespace: Benutzerdefinierter Namensraum des Datentyps

GUID: Eindeutige ID des Datentyps

Optionale Datentypeinstellungen

Offset [Bits]: Speicher-Offset

Size [Bits]: Berechnete Größe in Bits

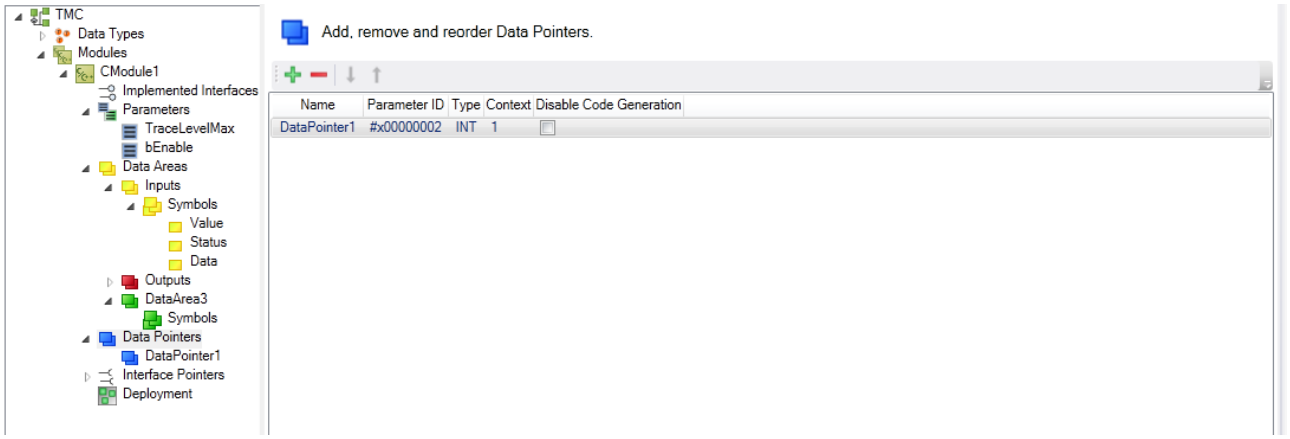
Unit: Optional

Comment: Optional

Create symbol: Standardeinstellung für ADS-Symbolerstellung

11.3.4.4 Datenzeiger

Data Pointer: Dialog zum Bearbeiten der Datenzeiger Ihres Moduls.



Symbol	Funktion
	Einen neuen Datenzeiger hinzufügen
	Löscht den ausgewählten Datenzeiger
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben

Name: Name des Datenzeigers.

Parameter ID: Eindeutige ID des Parameters.

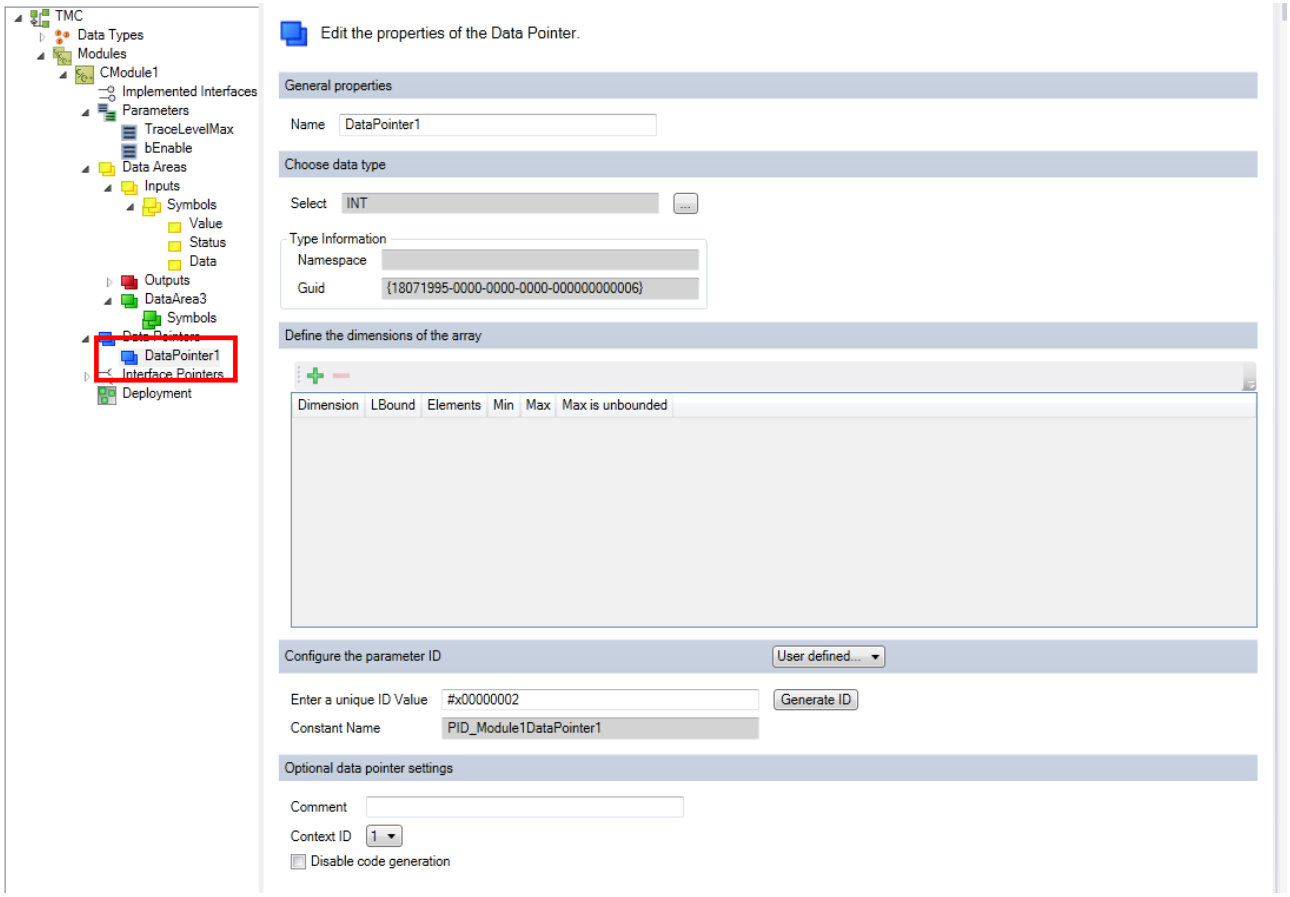
Type: Definiert den Zeigertyp.

Context: Zeigt die Kontext-ID an.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

11.3.4.4.1 Eigenschaften

Data Pointer Properties: Die Eigenschaften des Datenzeigers bearbeiten.



Allgemeine Eigenschaften

Name: Name des Datenzeigers.

Den Datentyp definieren

Select: Datentyp auswählen.

Typinformation

- **Name**: Name des ausgewählten Datentyps.
- **GUID**: Eindeutige ID des Datentyps.

Die Dimension des Arrays definieren

Siehe Kapitel Array [▶ 110].

Die Parameter-ID konfigurieren

Enter a unique ID Value: Einen eindeutigen ID-Wert eingeben, siehe Parameter [▶ 116].

Constant Name: Quellcodename der Parameter-ID.

Optionale Datenzeigereinstellungen

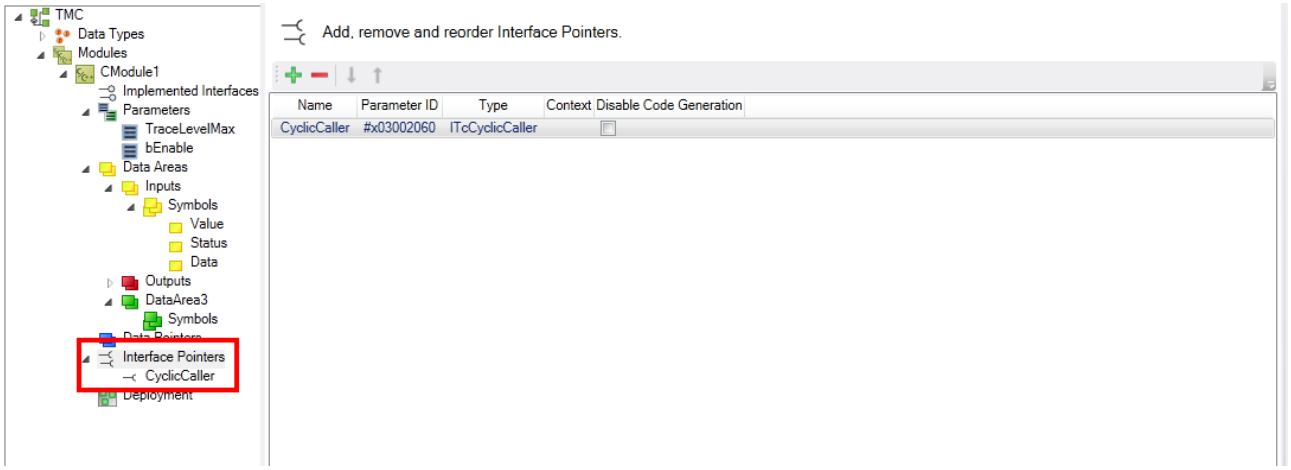
Comment: Kommentar, der z. B. im Instanzenkonfigurator sichtbar ist.

Context ID: Kontext-ID des Datenzeigers.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

11.3.4.5 Schnittstellenzeiger

Interface Pointers: Schnittstellenzeiger hinzufügen, entfernen und neu ordnen.



Symbol	Funktion
	Schnittstellenzeiger hinzufügen
	Löscht den ausgewählten Zeiger
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben

Name: Name der Schnittstelle.

Parameter ID: Eindeutige ID des Schnittstellenzeigers.

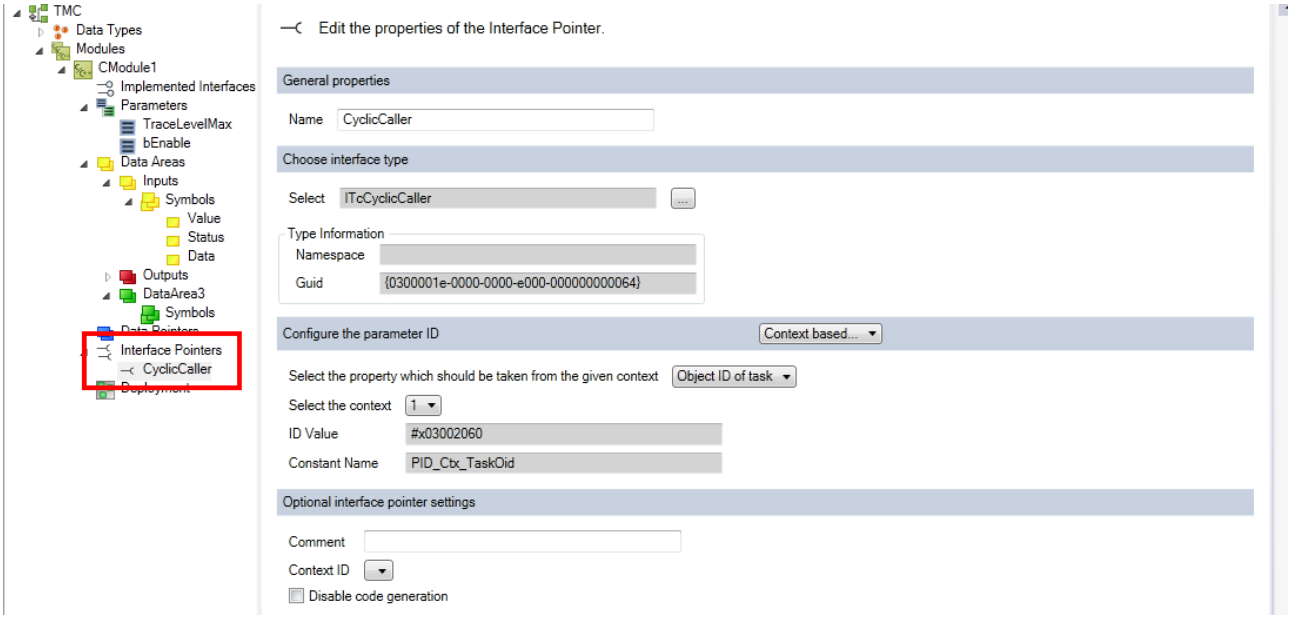
Type: Typ des Schnittstellenzeigers.

Context: Kontext der Schnittstelle.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

11.3.4.5.1 Eigenschaften

Interface Pointer Properties: Die Eigenschaften des Schnittstellenzeigers bearbeiten.



Allgemeine Eigenschaften

Name: Name des Schnittstellenzeigers.

Basisschnittstelle auswählen

Select: Auswahl der Schnittstelle.

Typinformation

- **Namespace:** Namensraum der Schnittstelle.
- **GUID**: Eindeutige ID der Schnittstelle.

Die Parameter-ID konfigurieren

Siehe Parameter [▶ 116].

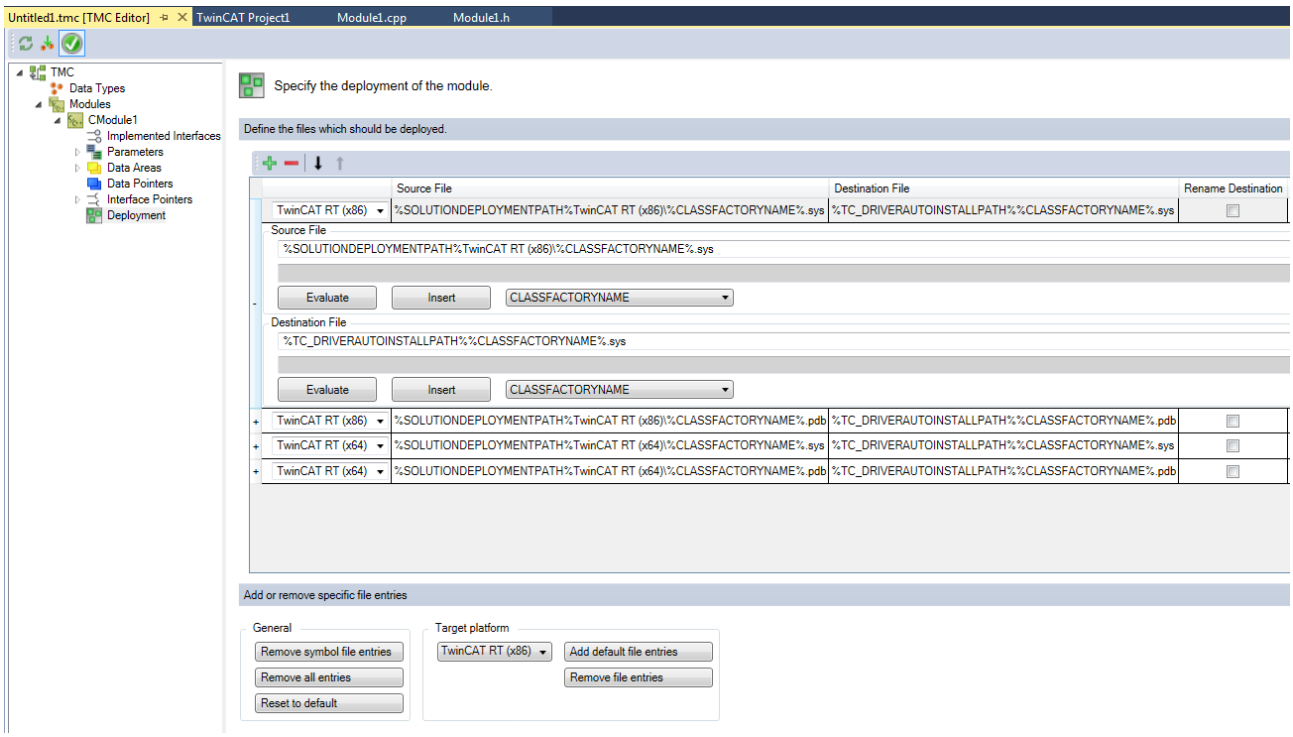
Comment: Optional

Context ID: Kontext-ID des Schnittstellenzeigers.

Disable Code Generation: Die Code-Generierung freigeben/sperren.

11.3.4.6 Bereitstellung

Deployment: Speicherorte für die bereitgestellten Module auf dem Zielsystem festlegen. Die Einträge sind für Versionierte C++ Projekte mit ihren Modulen leer und werden nicht benötigt.



Symbol	Funktion
	Einen neuen Dateieintrag hinzufügen
	Einen Dateieintrag löschen
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben

Dieser Dialog ermöglicht die Konfiguration von Quelle und Ziel-Datei, die für die jeweiligen „Platforms“ auf das Zielsystem übertragen werden.

Define the files, which should be deployed.

Source File: Pfad zu den Quelldateien.

Destination File: Pfad zu den Binärdateien.

Rename Destination: Zieldatei wird umbenannt, bevor die neue Datei übertragen wird. Da dieses für Windows 10 erforderlich ist, wird es implizit gemacht.

Die einzelnen Einträge können aus- und eingeklappt werden durch das **+** bzw. **-** am Anfang.

Evaluate: Setzt zur Überprüfung den berechneten Wert in das Textfeld.

Insert: Fügt die in der DropDown-Liste ausgewählte Variablenbezeichnung ein.

Add or remove specific file entries

Remove symbol file entries: Entfernt die Einträge für die Bereitstellung von Symbol-Dateien (PDB).

Remove all entries: Entfernt alle Einträge.

Reset to default: Setzt die Standard-Einträge .

Add default file entries: Fügt die Einträge für die ausgewählte Plattform hinzu.

Remove file entries: Entfernt die Einträge für die ausgewählte Plattform.

Quell- und Zielpfade für die Bereitstellung können virtuelle Umgebungsvariablen beinhalten, die vom TwinCAT XAE / XAR System aufgelöst werden.

Die nachfolgende Tabelle enthält die Liste dieser unterstützten virtuellen Umgebungsvariablen.

Virtuelle Umgebungsvariable	Registry-Eintrag (REG_SZ) unter Key \HKLM\Software\Beckhoff\TwinCAT3	Defaultwert (Windows)	Defaultwert (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\	nicht verfügbar
%TC_DRIVERAUTOINSTALLPATH%	DriverAutoInstallDir	C:\TwinCAT\3.**1**\Driver\AutoInstall\	/usr/local/etc/TwinCAT/3.**1**/
%CLASSFACTORYNAME%		<Name der Class Factory>	<Name der Class Factory>

11.4 TwinCAT Module Instance Configurator

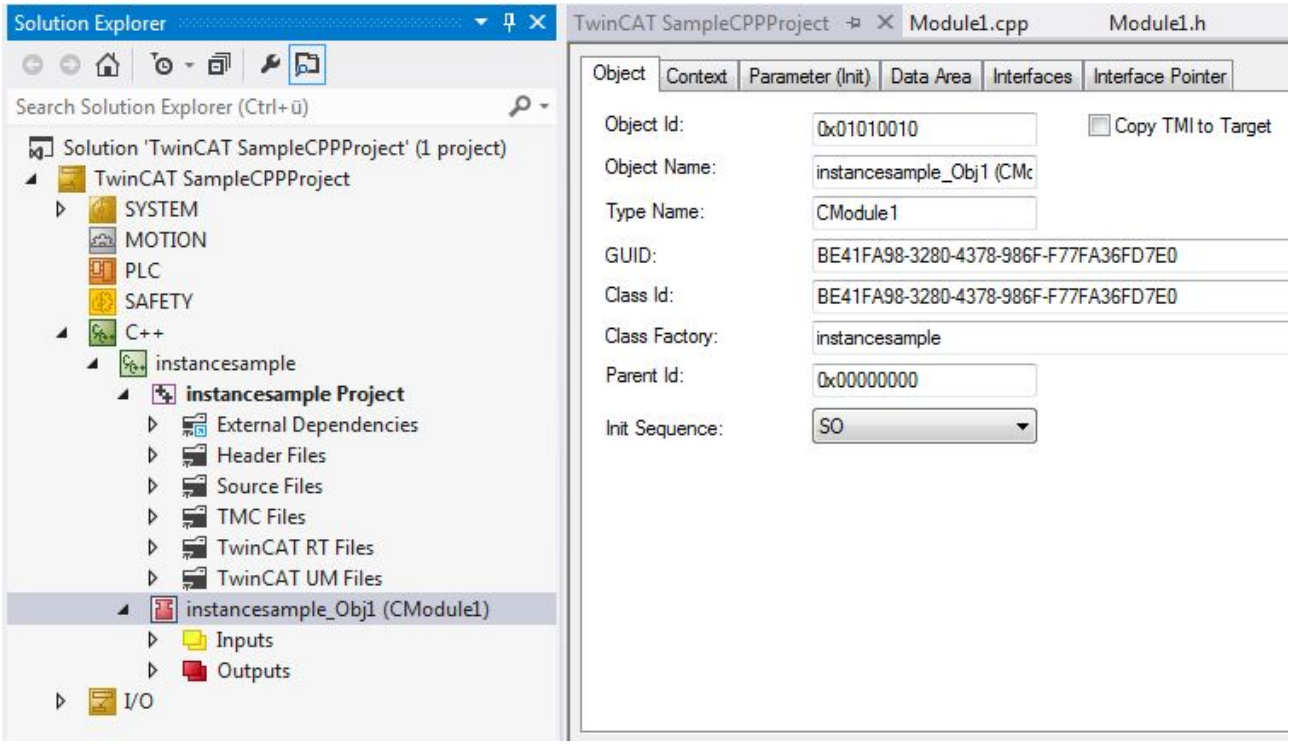
Der oben beschriebene TwinCAT 3 Module Class (TMC) Editor definiert Treiber auf Klassenebene. Diese werden instanziiert und müssen über den TwinCAT 3 Instance Configurator konfiguriert werden.

Die Konfiguration betrifft z. B. den Kontext (einschließlich Task, die das Modul aufruft), Parameter und Zeiger.

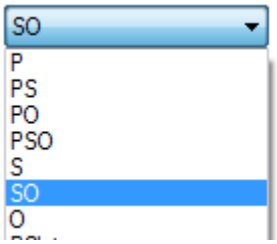
Instanzen von C++ Klassen werden durch Rechtsklick auf den C++ Projektordner erstellt, siehe Schnellstart [▶ 62]. In diesem Kapitel wird die Konfiguration dieser Instanzen ausführlich beschrieben.

Durch Doppelklick auf die generierte Instanz wird der Konfigurationsdialog mit mehreren Fenstern geöffnet.

11.4.1 Objekt



- **Object Id:** Die Objekt-ID, die für die Identifizierung dieser Instanz im TwinCAT System herangezogen wird.
- **Object Name:** Name des Objekts, der für die Darstellung der Instanz im Solution Explorer-Baum verwendet wird.
- **Type Name:** Typinformation (Klassenname) der Instanz.
- **GUID:** Modulklassen-GUID.
- **Class Id:** Klassen-ID der Implementierungsklasse (normalerweise sind GUID und ClassId identisch).
- **Class Factory:** Verweist auf den Treiber, der die Class Factory bereitstellt, welche für die Erstellung einer Instanz des Moduls verwendet wurde.
- **Parent Id:** Beinhaltet die ObjectID des Parent, falls vorhanden.
- **Init Sequence:** Legt die Initialisierungszustände für die Bestimmung des Startup-Verhaltens der interagierenden Module fest. Siehe TwinCAT-Modul Zustandsmaschine [▶ 48] für die genaue Beschreibung der Zustandsmaschine.



Festlegung des Startup-Verhaltens von mehreren TcCOM Instanzen

TcCOM Instanzen können sich gegenseitig auf einander beziehen - z. B. zwecks Wechselwirkung über Daten- oder Schnittstellenzeiger. Zur Bestimmung des Startup-Verhaltens legt die **Init Sequenz** von jeder TcCOM Instanz zu „haltende“ Zustände für alle übrigen Module fest.

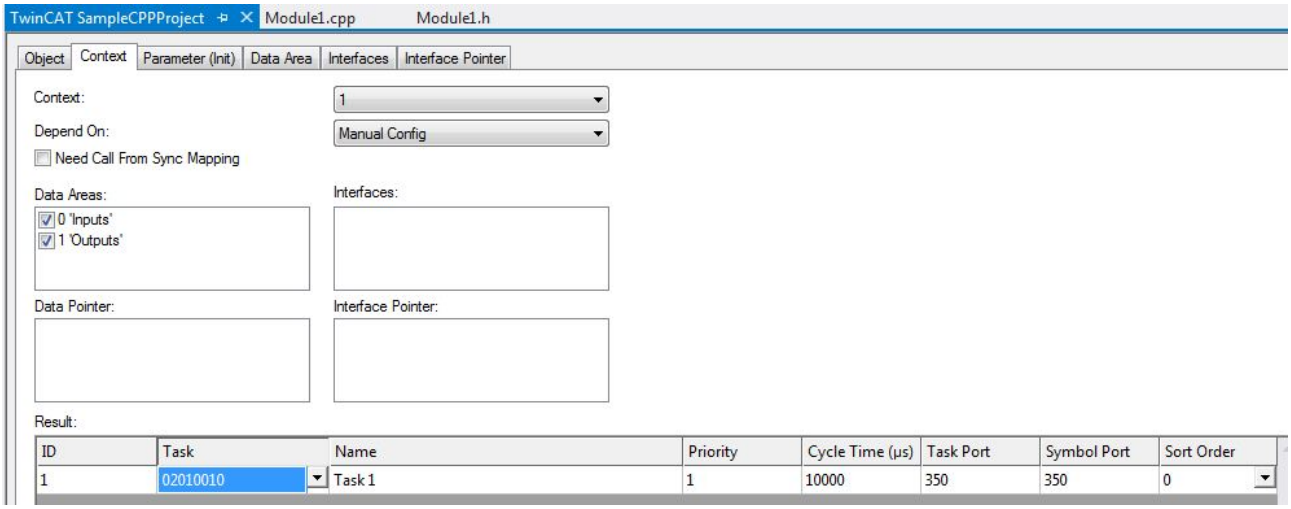
Der Name einer Init Sequenz besteht aus dem Kurznamen der TcCOM Zustandsmaschine. Wenn der Kurzname eines Zustands (I,P,S,O) im Namen der Init Sequenz enthalten ist, dann werden die Module in diesem Zustand warten, bis alle anderen Module zumindest diesen Zustand erreicht haben. Beim nächsten Übergang kann das Modul sich auf alle anderen Modulinstanzen beziehen, um sich mindestens in diesem Zustand zu befinden.

Wenn z. B. ein Modul die Init Sequenz „PS“ aufweist, werden die IP-Übergänge aller anderen Module ausgeführt, so dass alle Module sich im Zustand „Preop“ befinden.

Anschließend wird der PS-Übergang des Moduls ausgeführt und das Modul kann sich darauf verlassen, dass die anderen Module sich im „Preop“-Zustand befinden.

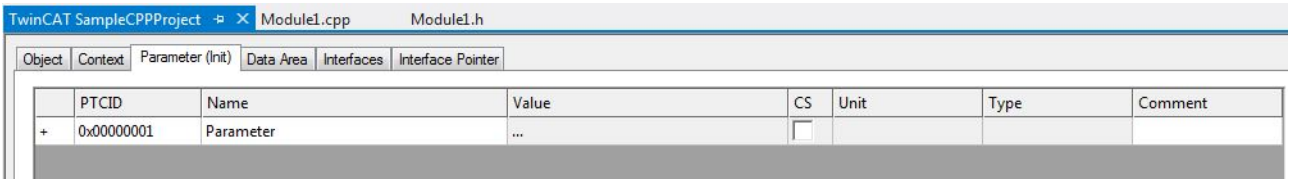
- **Copy TMI to target:** Die TMI (TwinCAT Module Instance) Datei generieren und an das Ziel übergeben.

11.4.2 Kontext



- **Context:** Den zu konfigurierenden Kontext auswählen (siehe TMC Editor für Hinzufügen verschiedener Kontexte).
  Ein Datenbereich ist einem Kontext zugeordnet.
- **Data Areas / Interfaces / Data Pointer und Interface Pointer:** Jede Instanz kann so konfiguriert werden, dass sie in TMC definierte Elemente hat oder nicht.
- **Result Table:** Liste der IDs, die konfiguriert werden müssen. Zumindest muss der Kontext („Task“-Spalte) der Task entsprechend konfiguriert werden.

11.4.3 Parameter (Init)

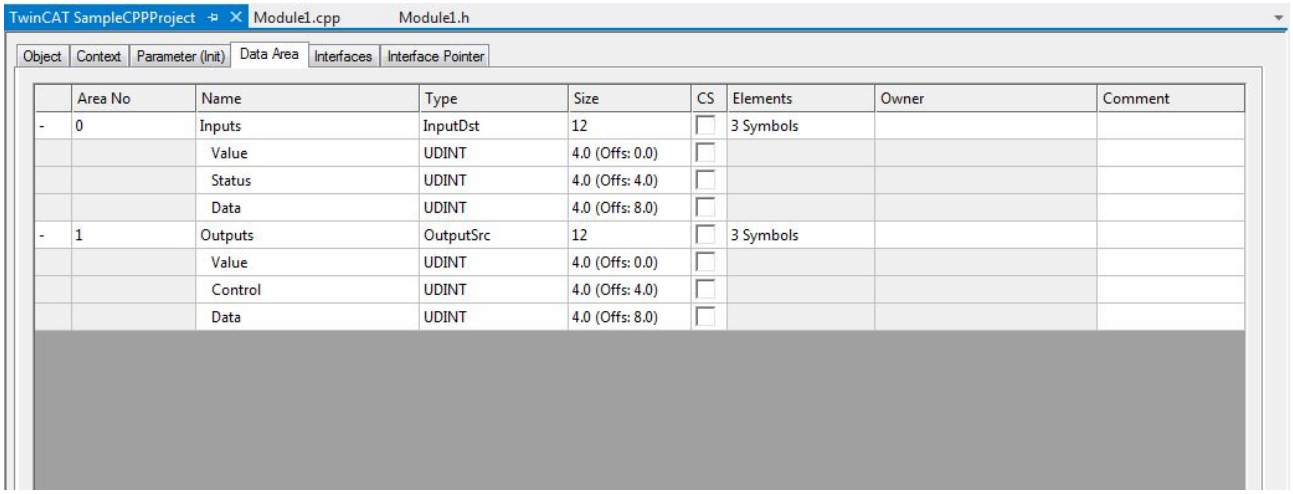


Die Liste aller Parameter (wie in TMC definiert) kann für jede Instanz mit Werten initialisiert werden.

Spezielle ParameterIDs (PTCID) werden verwendet, um die Werte automatisch festzulegen. Diese werden mit Hilfe des Parameterdialogfensters des TMCEditor's wie in Parameter [▶ 116] beschrieben konfiguriert.

Die CS (CreateSymbol) Checkbox erzeugt das ADS-Symbol für jeden Parameter, demzufolge ist er von außen erreichbar.

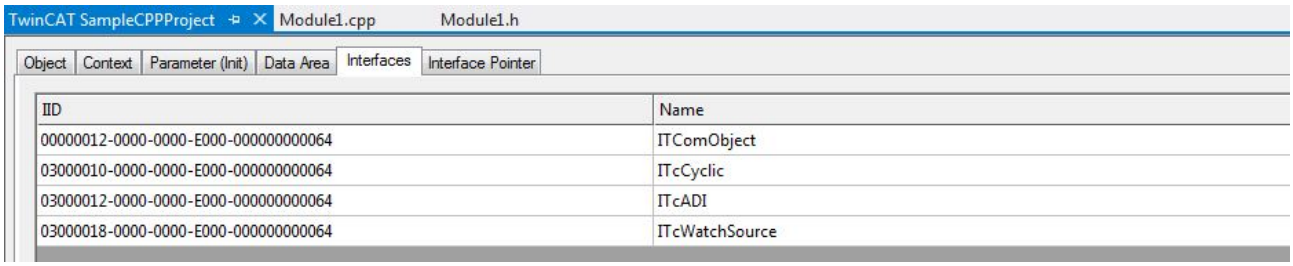
11.4.4 Data Area



Liste aller Datenbereiche und ihrer Variablen (wie in TMC definiert).

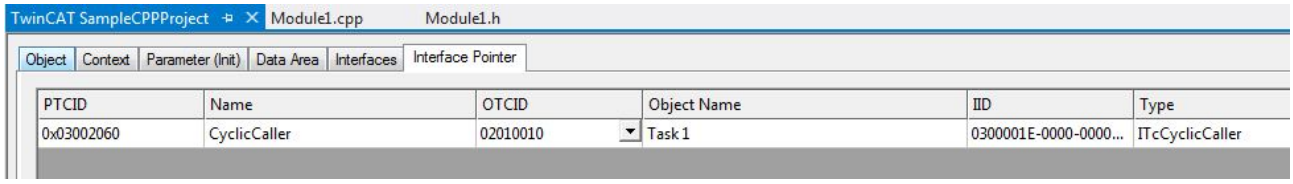
Die CS (CreateSymbol) Checkbox erzeugt das ADS-Symbol für jeden Parameter, so ist die Variable von außen erreichbar.

11.4.5 Schnittstellen



Liste aller implementierter Schnittstellen (wie in TMC definiert).

11.4.6 Schnittstellenzeiger

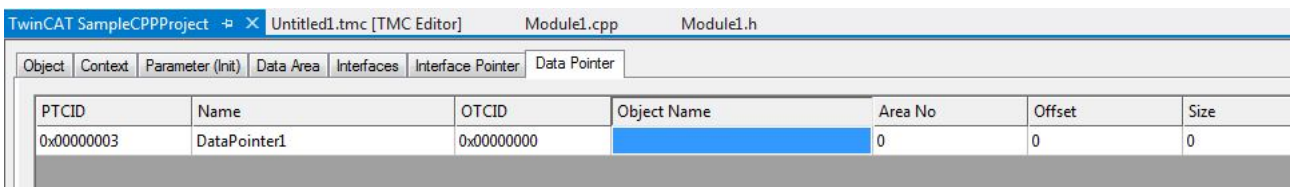


Liste aller Schnittstellenzeiger (wie in TMC definiert).

Spezielle ParameterIDs (PTCID) werden verwendet, um die Werte automatisch festzulegen. Diese werden mit Hilfe des Parameterdialogfensters des TMCEditor's wie in Parameter [▶ 116] beschrieben konfiguriert.

Die OTCID-Spalte definiert die Zeiger auf die Instanz, die zu verwenden ist.

11.4.7 Datenzeiger



Liste aller Datenzeiger (wie in TMC definiert).

Spezielle ParameterIDs (PTCID) werden verwendet, um die Werte automatisch festzulegen. Diese werden mit Hilfe des Parameterdialogfensters des TMCEditor's wie in Parameter [▶ 116] beschrieben konfiguriert.

Die OTCID-Spalte definiert die Zeiger auf die Instanz, die zu verwenden ist.

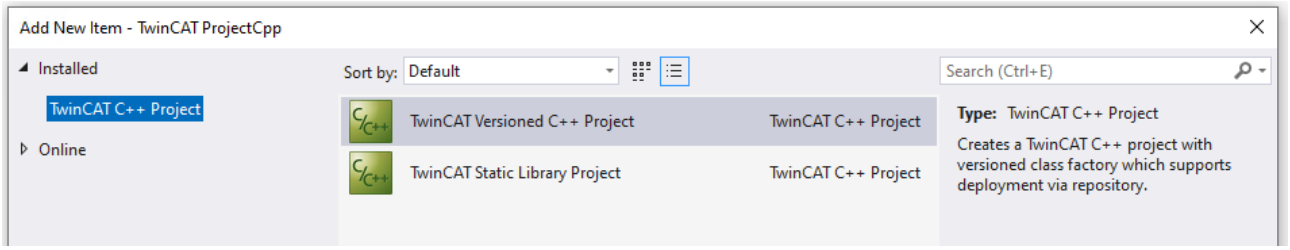
11.5 Kunden-spezifische Projekt-Templates

TwinCAT 3 ist in Visual Studio eingebettet und nutzt damit auch die bereitgestellte Projektverwaltung. TwinCAT 3 C++ Projekte sind „nested projects“ in der TwinCAT Projektmappe (TwinCAT Solution).

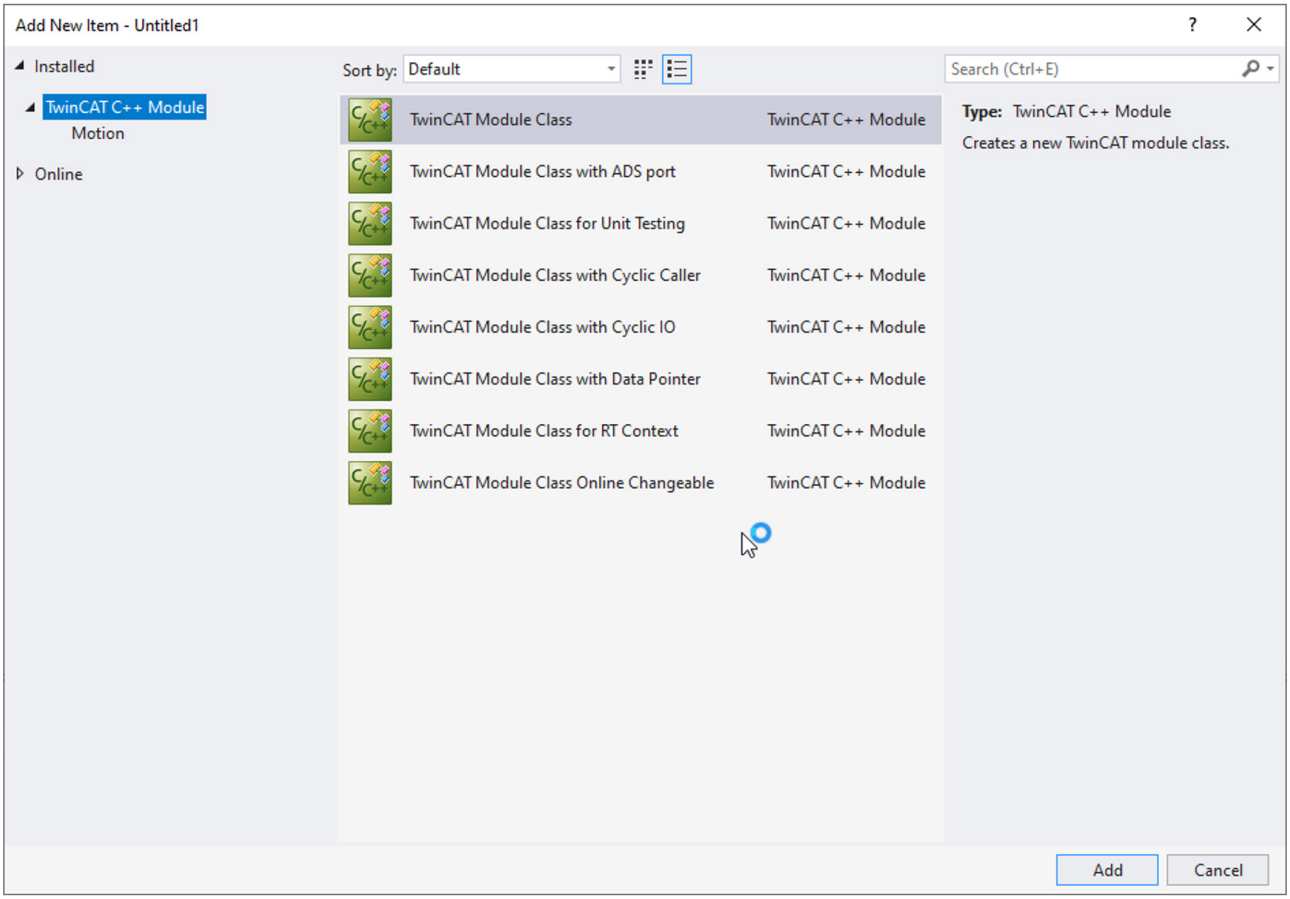
Dieser Abschnitt der Dokumentation beschreibt, wie Kunden eigene Projekt-Templates einbringen können.

11.5.1 Übersicht

Wenn ein TwinCAT C/C++ Projekt angelegt wird, wird zuerst der TwinCAT C++ Project Wizard gestartet. Dieser erzeugt ein Rahmengerüst für ein TwinCAT Versioned C++ Project. Die eigentliche Funktion wird in TwinCAT Modulen implementiert.



Der TwinCAT Class Wizard wird automatisch nach dem Anlegen eines neuen Projektes gestartet, um das erste Modul hinzuzufügen. Die unterschiedlichen Module werden hierbei von dem gleichen TwinCAT Class Wizard erzeugt, jedoch wird die konkrete Ausgestaltung der Module über Templates realisiert.



11.5.2 Beteiligte Dateien

Nahezu alle relevanten Informationen sind im Verzeichnis *C:\TwinCAT\3.1\Components\Base\CppTemplate* enthalten:

Der TwinCAT C++ Project Wizard ruft dabei den TwinCAT Module Class Wizard auf, falls ein Driver Project erstellt werden soll.

Verzeichnis: Driver und Class

In dem Driver- (für TwinCAT C++ Projekt Wizard) und Class-Verzeichnis (für die TwinCAT Module Class Wizard) sind dabei die jeweiligen Projektarten definiert, wobei jede Projektart 3 Dateien umfasst:

TcDriverWizard.ico	10.06.2015 11:14	Icon	267 KB
TcDriverWizard.vsdir	10.06.2015 11:14	VSDIR File	1 KB
TcDriverWizard.vsz	10.06.2015 11:14	Visual Studio Wiza...	1 KB
TcStaticLibraryWizard.ico	10.06.2015 11:14	Icon	267 KB
TcStaticLibraryWizard.vsdir	10.06.2015 11:14	VSDIR File	1 KB
TcStaticLibraryWizard.vsz	10.06.2015 11:14	Visual Studio Wiza...	1 KB

Die *.vsdir* Datei stellt Informationen bereit, die verwendet werden, wenn der jeweilige Assistent Wizard gestartet wird. Im Wesentlichen handelt es sich um einen Namen, Kurzbeschreibung und einen Dateinamen vom Typ *.vsz*, der Details zu dieser Projektart beinhaltet.
Die allgemeine Beschreibung im MSDN ist hier zu finden https://msdn.microsoft.com/de-de/library/Aa291929%28v=VS.71%29.aspx.

Die in der *.vsdir* Datei referenzierte *.vsz* Datei stellt Informationen bereit, die der Assistent (Wizard) benötigt. Die wichtigste Information ist hier der zu startende Wizard und eine Liste von Parametern.

Beide Assistenten haben eine *.xml* Datei als Parameter, die die Transformationen der z. B. Quelldateien vom Template zum konkreten Projekt beschreibt. Diese befinden sich zusammen mit den Vorlagen für Quellcode etc im *Templates*-Verzeichnis.
Der TwinCAT C++ Project Wizard startet für den Fall, dass ein Driver erstellt werden soll, über den Parameter T**riggerAddModule** den TwinCAT Module Class Wizard.

Die allgemeine Beschreibung im MSDN ist hier zu finden https://msdn.microsoft.com/de-de/library/Aa291929%28v=VS.71%29.aspx.

Die *.ico* Datei stellt lediglich ein Icon bereit.

Verzeichnis: Templates

Im *Templates*-Verzeichnis befinden sich in entsprechenden Unterverzeichnissen sowohl die Vorlagen für Quellcode wie auch die in der *.vsz* benannte *.xml* Datei für den TwinCAT Module Class Wizard.

Diese *.xml* Datei beschreibt dabei den Vorgang, um von dem Template zu einem konkreten Projekt zu kommen.

11.5.3 Transformationen

Transformationsbeschreibung (XML Datei)

Die Konfigurationsdatei beschreibt (in XML) die Transformation der Template-Dateien in den Projektordner. Im Normalfall wird es sich hier um *.cpp / .h* sowie ggf. Projektdateien handeln – es können aber alle Arten von Dateien behandelt werden.

Der Wurzelknoten ist ein Element <ProjectFileGeneratorConfig>. Direkt an diesem Knoten kann das Attribute useProjectInterface="true" gesetzt werden. Es setzt den Processing Vorgang in den Modus Visual-Studio Projekte zu erzeugen (im Gegensatz zu TC-C++-Modulen).

Hier folgen mehrere <FileDescription>-Elemente, die jeweils die Transformation einer Datei beschreiben. Nach diesen Elementen besteht die Möglichkeit, in einem Element <Symbols> Symbole, die zur Transformation bereitstehen, zu definieren.

Transformation der Template-Dateien

Ein Element <FileDescription> ist folgendermaßen aufgebaut:

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

- Die Quelldatei aus dem Templates-Verzeichnis wird als <SourceFile> angegeben.
- Die Zieldatei im Projekt-Verzeichnis wird als <TargetFile> angegeben. Dabei wird normalerweise ein Symbol mittels des Kommandos [!output …] genutzt werden.
- Mit dem Attribut „copyOnly" kann angegeben werden, ob die Datei transformiert werden soll, d. h. dass die in der Quelldatei beschriebenen Transformationen ausgeführt werden. Ansonsten wird die Datei lediglich kopiert.
- Mit dem Attribut „openFile" kann angegeben werden, ob die Datei nach dem Erstellen des Projektes in Visual Studio geöffnet werden soll.
- Filter: Es wird im Projekt ein Filter angelegt.
  Hierfür muss am <ProjectFileGeneratorConfig> das Attribute useProjectInterface="true" gesetzt werden.

Transformationsanweisungen

In den Template-Dateien werden Kommandos genutzt, welche die Transformationen selber beschreiben.

Folgende Kommandos stehen bereit:

- `[!output SYMBOLNAME]`
  Dieses Kommando ersetzt das Kommando durch den Wert (Value) des Symbols. Eine Reihe von vordefinierten Symbolen steht bereit.
- `[!if SYMBOLNAME]`, `[!else]` und `[!endif]` beschreiben eine Möglichkeit, entsprechenden Text bei der Transformation nur in bestimmten Situationen einzubinden.

Symbolnamen

Symbolnamen können auf 3 Arten für die Transformationsanweisungen bereitgestellt werden.
Diese werden von den zuvor beschriebenen Kommandos verwendet, um Ersetzungen vorzunehmen.

1. Eine Reihe von vordefinierten Symbolen direkt in der Konfigurationsdatei:
   In der XML-Datei ist eine Liste von <Symbols> vorgesehen. Hier können Symbole definiert werden:
   ```
   <Symbols>
   <Symbol>
   <Name>CustomerSymbol</Name>
   <Value>CustomerString</Value>
   </Symbol>
   </Symbols>
   ```
2. Die generierten Zieldateinamen können durch Hinzufügen des „symbolName"-Attributes bereitgestellt werden:
   ```
   <TargetFile symbolName="CustomerFileName">[!output SYMBOLNAME].txt</
   TargetFile>
   ```
3. Wichtige Symbole werden vom System selbst bereitgestellt.

Symbol Name (Projekte)	Beschreibung
PROJECT_NAME	Der Projektname aus dem Visual Studio Dialog.
PROJECT_NAME_UPPERCASE	Der Projektname in Großbuchstaben.
WIN32_WINNT	0x0400
DRVID	Driver ID im Format: 0x03010000
PLATFORM_TOOLSET	Toolset version, z. B. v100
PLATFORM_TOOLSET_ELEMENT	Toolset version als XML Element, z. B. <PlatformToolset>v100</PlatformToolset>
NEW_GUID_REGISTRY_FORMAT	Erstellt eine neue GUID im Format: {48583F97-206A-4C7C-9EF2-D5C8A31F7BDC}

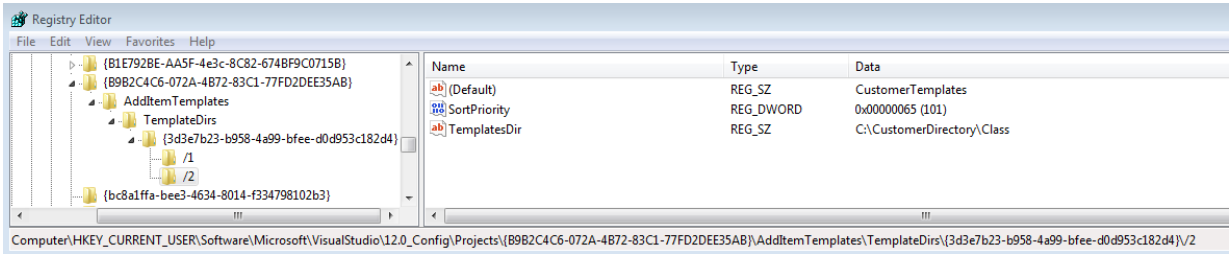
Symbol Name (Klassen)	Beschreibung
PROJECT_NAME	Der Projektname aus dem Visual Studio Dialog.
HEADER_FILE_NAME	Wird durch den Nutzer im Wizard-Dialog eingegeben.
SOURCE_FILE_NAME	Wird durch den Nutzer im Wizard-Dialog eingegeben.
CLASS_NAME	Wird durch den Nutzer im Wizard-Dialog eingegeben.
CLASS_SHORT_NAME	Wird durch den Nutzer im Wizard-Dialog eingegeben.
CLASS_ID	Eine neue durch den Wizard erstellte GUID.
GROUP_NAME	C++
TMC_FILE_NAME	Wird genutzt, um das TMC File zu identifizieren.
NEW_GUID_REGISTRY_FORMAT	Erstellt eine neue GUID im Format: {48583F97-206A-4C7C-9EF2-D5C8A31F7BDC}

11.5.4 Hinweise zum Handling

Template in Kunden-spezifischem Verzeichnis

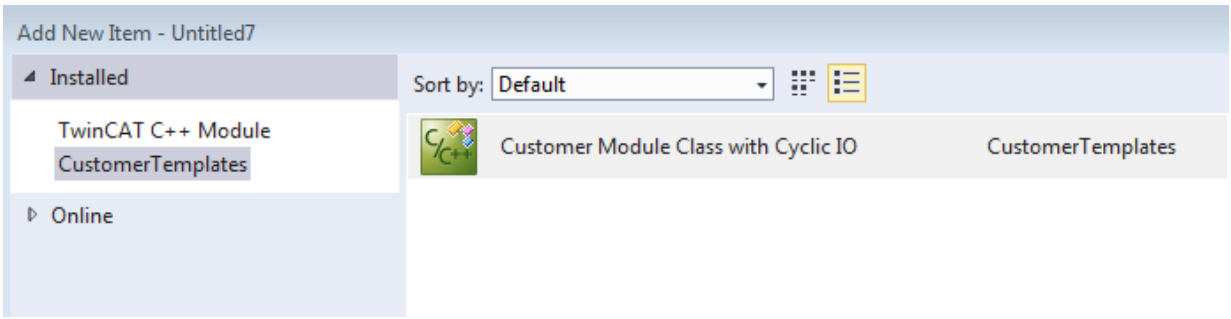
Templates können auch außerhalb vom TwinCAT üblichen Verzeichnis abgelegt werden.

1. Erweitern Sie in der Registry den Suchpfad (hier V12.0, also für VS 2013), in dem der Knoten /2 angelegt wird:
   Registry Key:
   `HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\12.0_Config\Projects\{B9B2C4C6-072A-4B72-83C1-77FD2DEE35AB}\AddItemTemplates\TemplateDirs\{3d3e7b23-b958-4a99-bfee-d0d953c182d4}\`



2. Erhöhen Sie die SortPriority.
3. Empfehlung: legen Sie in dem Verzeichnis ein Unterverzeichnis Class an, welches in der Registry eingetragen wird, und ein Unterverzeichnis Templates, um die *.vsz* / *.vsdir* / *.ico* Dateien von den Templates zu trennen.
4. Passen Sie die Pfade innerhalb der Dateien an.

⇨ Als Ergebnis existiert eine eigene Ordnung für die Templates:



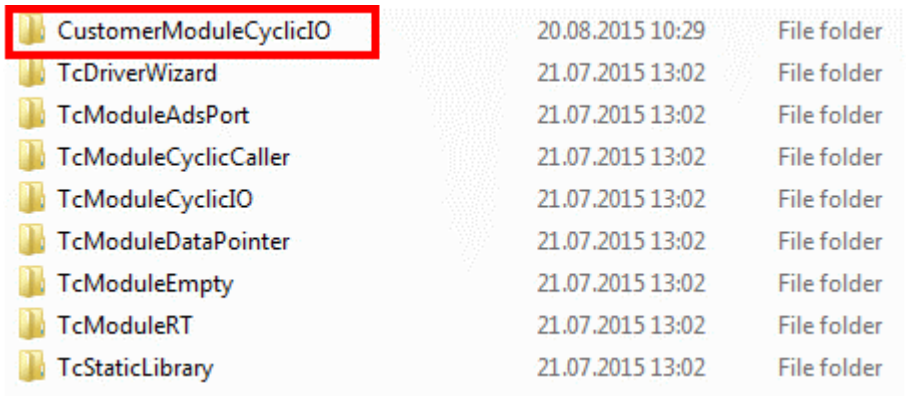
Dieses Verzeichnis bzw. diese Verzeichnisstruktur kann nun z. B. im Versionsverwaltungssystem versioniert werden und ist auch von TwinCAT Installationen / Updates nicht betroffen.

Quickstart

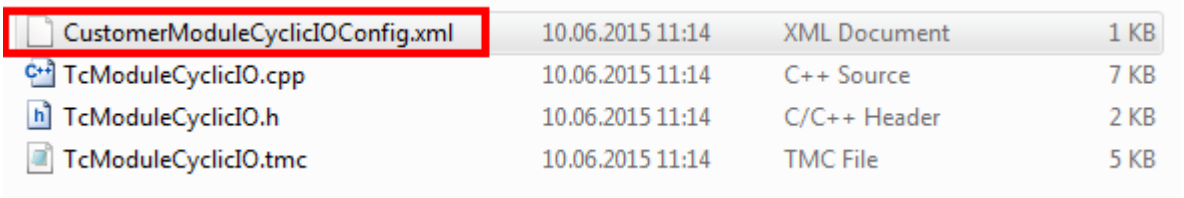
Ein allgemeiner Einstieg in die Assistenten-Umgebung im MSDN ist der Einstiegspunkt: https://msdn.microsoft.com/de-de/library/7k3w6w59%28v=VS.120%29.aspx.

Hier wird beschrieben, wie ein Template zur Erstellung eines Kunden-spezifischen Moduls erfolgt mit dem TwinCAT C++ Modul Wizard.

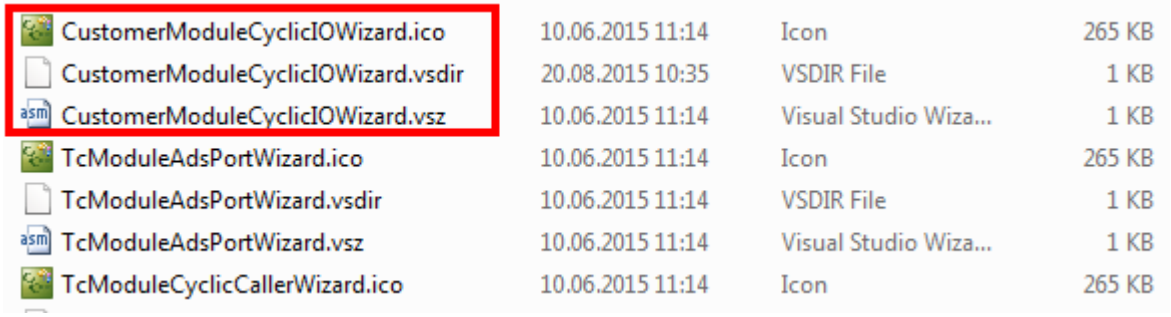
1. Nehmen Sie ein existierendes Modul-Template als Kopiervorlage
   In **C:\TwinCAT\3.1\Components\Base\CppTemplate\Templates**



2. Benennen Sie die *.xml* Datei innerhalb des Ordners um



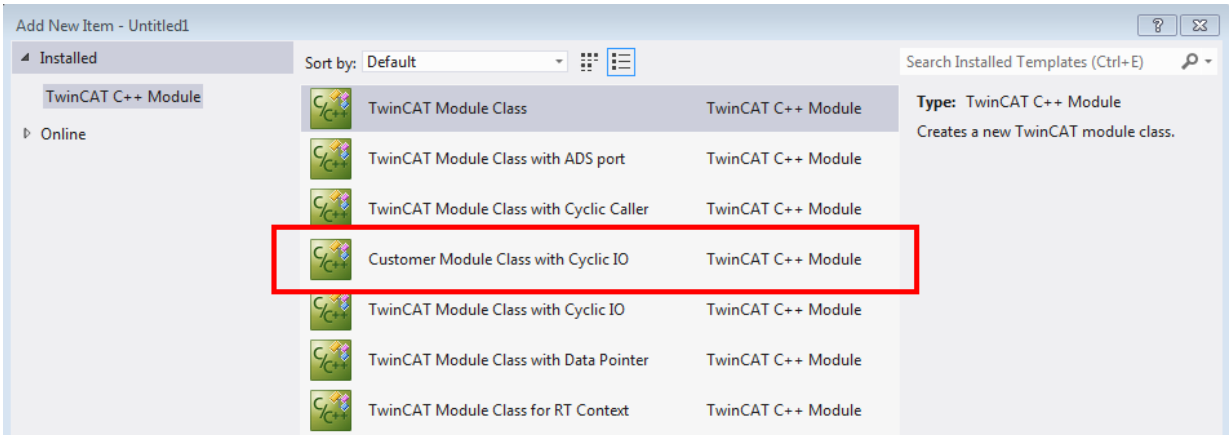
3. Kopieren Sie die entsprechenden Dateien *.ico* / *.vsdir* / *.vsz* auch im Class/



4. Referenzieren Sie in der *.vsdir* Datei nun die kopierte *.vsz* Datei und passen Sie die Beschreibung an.
5. Tragen Sie in der *.vsz* Datei die in Schritt 2 erstellte *.xml* ein.

6. Sie können nun in dem *Template/CustomerModuleCyclicIO/* Verzeichnis Änderungen an den Quelldateien vornehmen. Die *.xml* sorgt für Ersetzungen bei der Erzeugung eines Projektes aus diesem Template.

⇨ Der TwinCAT Module Class Wizard zeigt nun das neue Projekt zur Auswahl an:



Sollten z. B. auch die vsxproj verändert bereitgestellt werden, empfiehlt sich die Anpassung einer Kopie des TwinCAT C++ Project Wizard.

Ggf. ist auch die Verwendung von Einstellungen in *.props* Dateien in Betracht zu ziehen, damit auch bei bestehenden Projekten, die aus einem Template erzeugt wurden, Einstellungen geändert werden können – z. B. dadurch, dass die *.props* Dateien über ein Versionsverwaltungssystem aktualisiert werden.

Alternative Erstellung auf Basis eines existierenden Projektes

Hierfür ist ein gangbarer Weg, dass ein fertiges Projekt erstellt und nachher in ein Template abgewandelt wird.

1. Kopieren Sie das bereinigte Projekt in den *Templates* Ordner.
2. Legen Sie eine Transformationsbeschreibung (XML Datei) an.
3. Bereiten Sie Quelldateien und die Projektdatei mittels der beschriebenen Ersetzungen vor.
4. Stellen Sie die *.ico* / *.vsdir* / *.vsz* Dateien bereit.

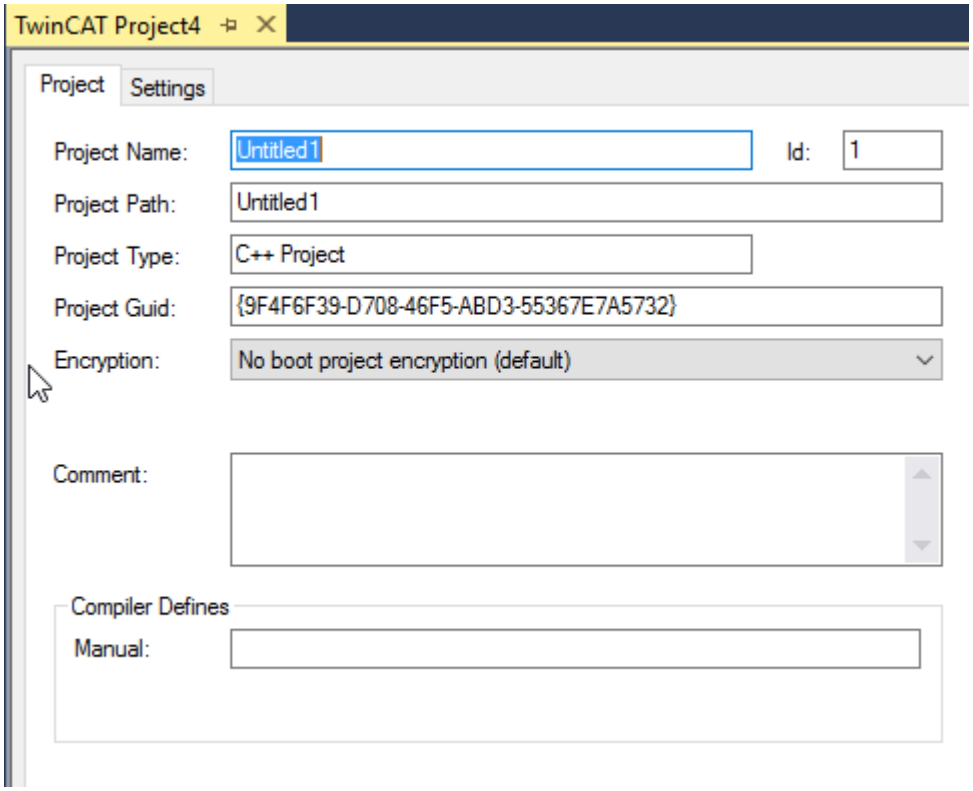
12 Programmierreferenz

TwinCAT bietet eine Vielzahl an Basisfunktionen. Sie alle können für einen TwinCAT C++-Programmierer sehr nützlich sein und werden hier dokumentiert.

Es besteht eine Vielzahl an C++-Beispielen, die wertvolle Informationen über die Handhabung der Module und Schnittstellen beinhalten.

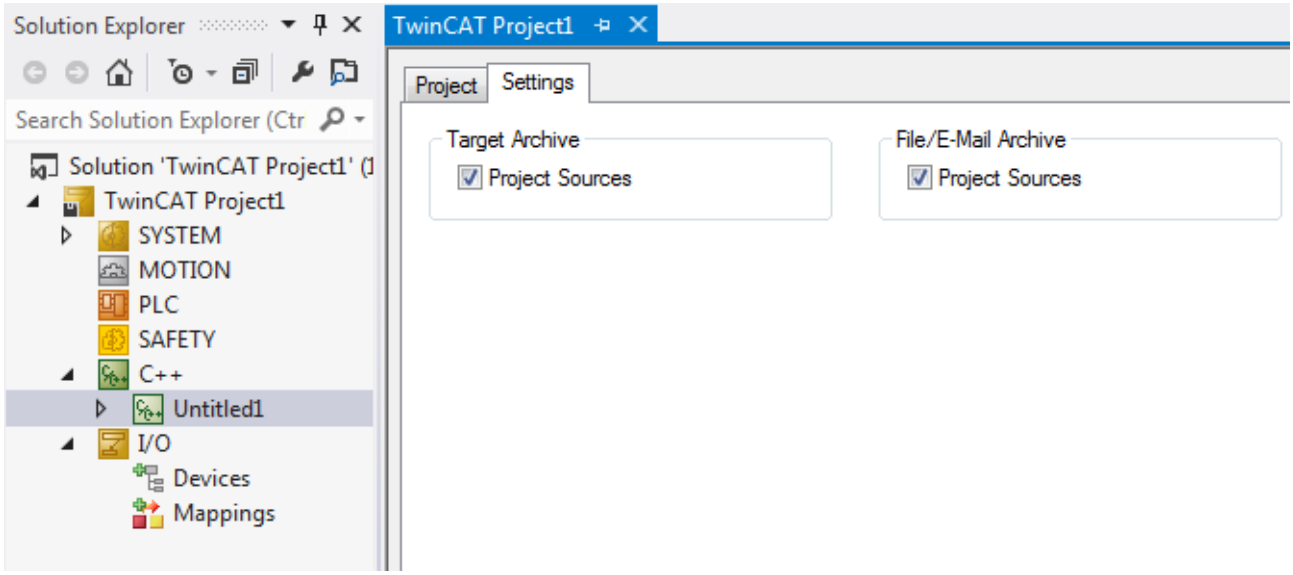
TwinCAT C++-Projekt

Ein TwinCAT C++-Projekt besitzt einige Parameter, die durch einen Doppelklick auf das TwinCAT C++-Projekt (Projektname hier „Untitled1“) geöffnet werden können.



Ein Umbenennen ist hier nicht möglich (vgl. Umbenennen von TwinCAT-C++ Projekten [▶ 230])

Die Verschlüsselung des binären Moduls kann hier eingestellt werden, eine genauere Beschreibung zu den Voraussetzungen findet sich im Kapitel Module verschlüsseln [▶ 57].



Für die beiden Archiv-Typen, die auf das Zielsystem übertragen werden bzw. die per E-Mail versendet werden, kann hier eingestellt werden, ob die Quellen beinhaltet sein sollen.

Bei Deselektion werden entsprechend leere Archive erzeugt.

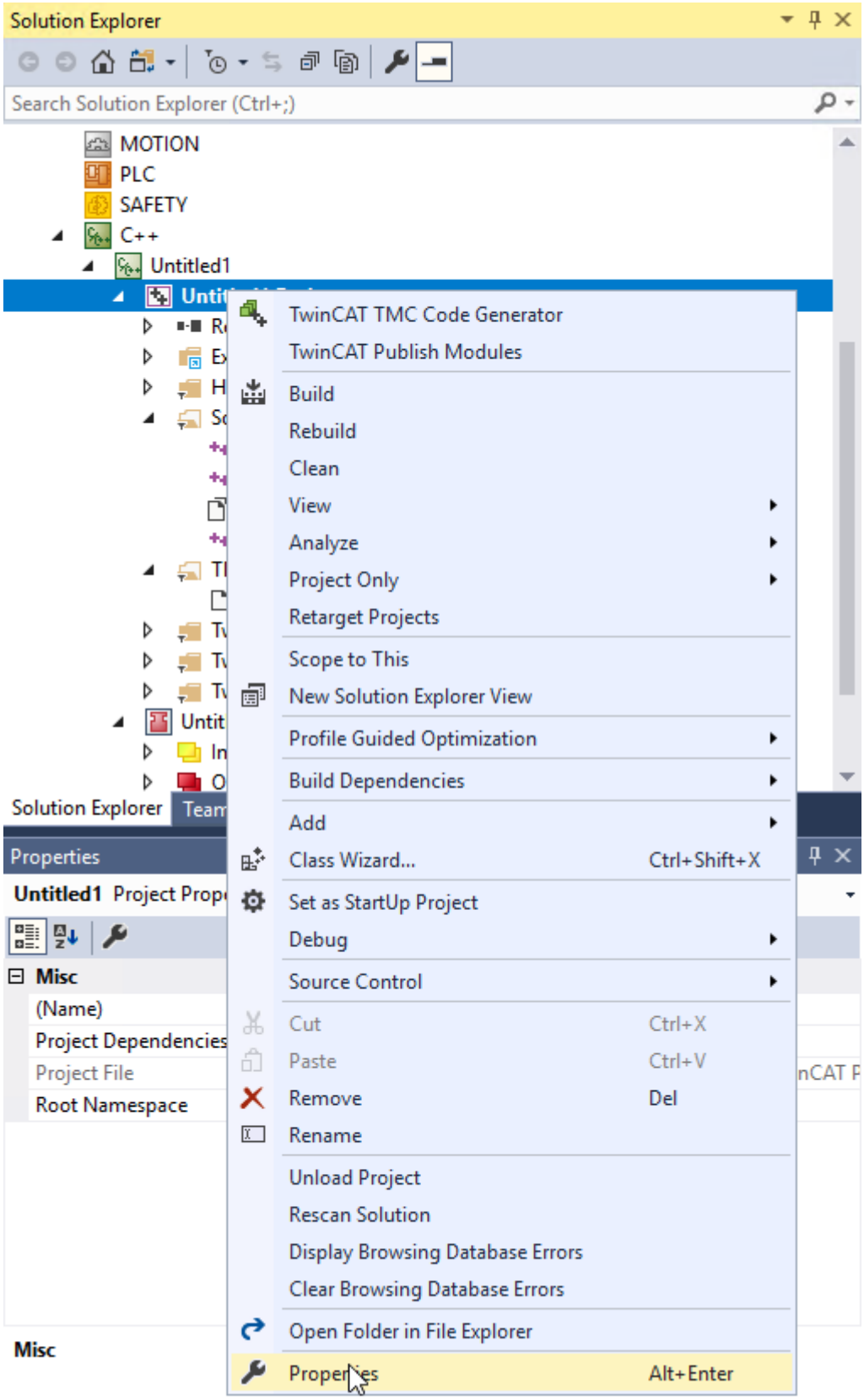
12.1 TwinCAT C++ Projekteigenschaften

Ab TwinCAT 3.1. Build 4024.0

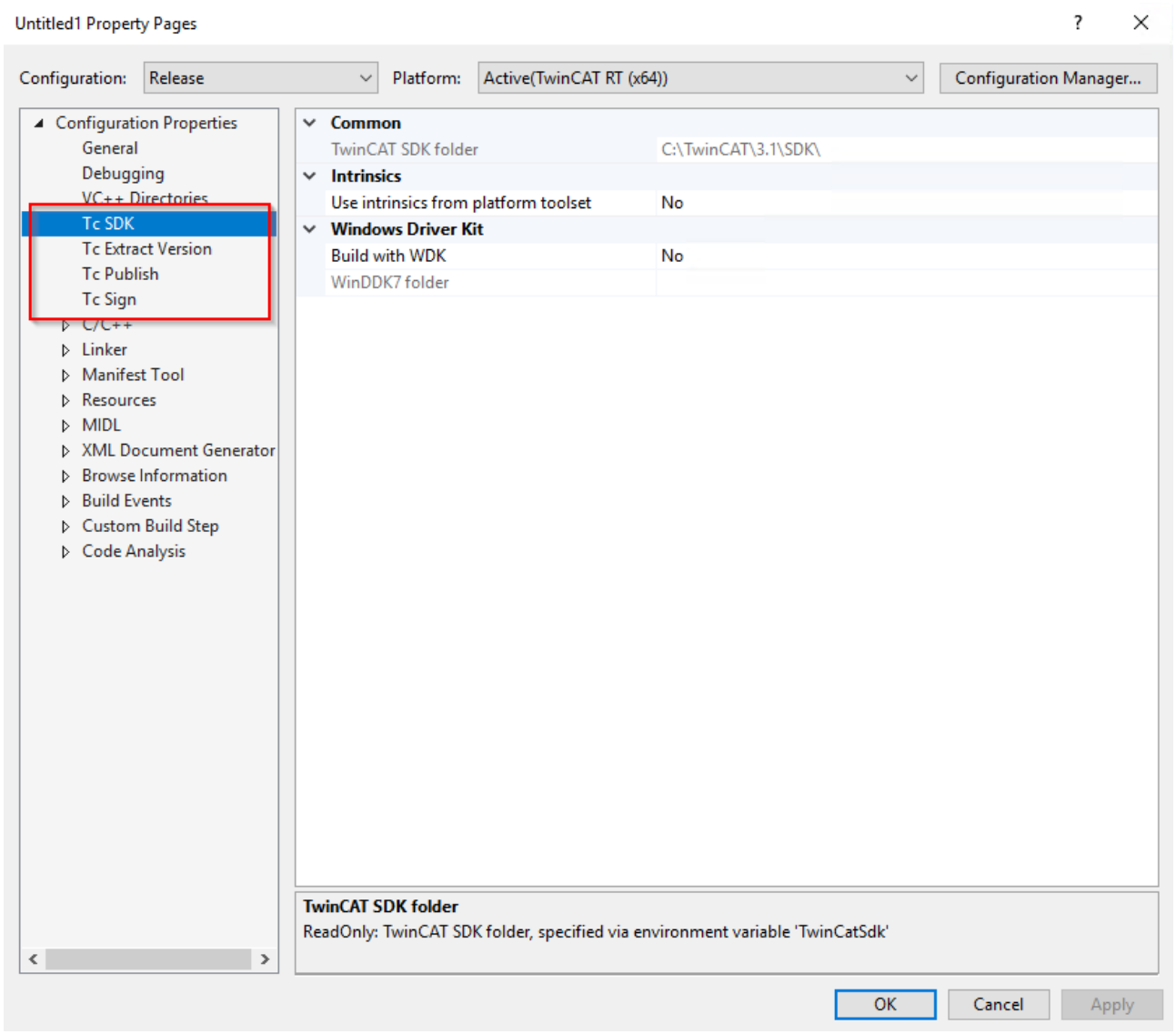
Die hier beschriebene Funktionalität ist ab TwinCAT 3.1. 4024.0 verfügbar.

Für ein TwinCAT C++ Projekt können in den Projekt-Eigenschaften unterschiedliche Einstellungen vorgenommen werden.

Die Projekt-Eigenschaften werden durch Rechts-Klick auf das C++ Projekt -> **Properties** geöffnet.

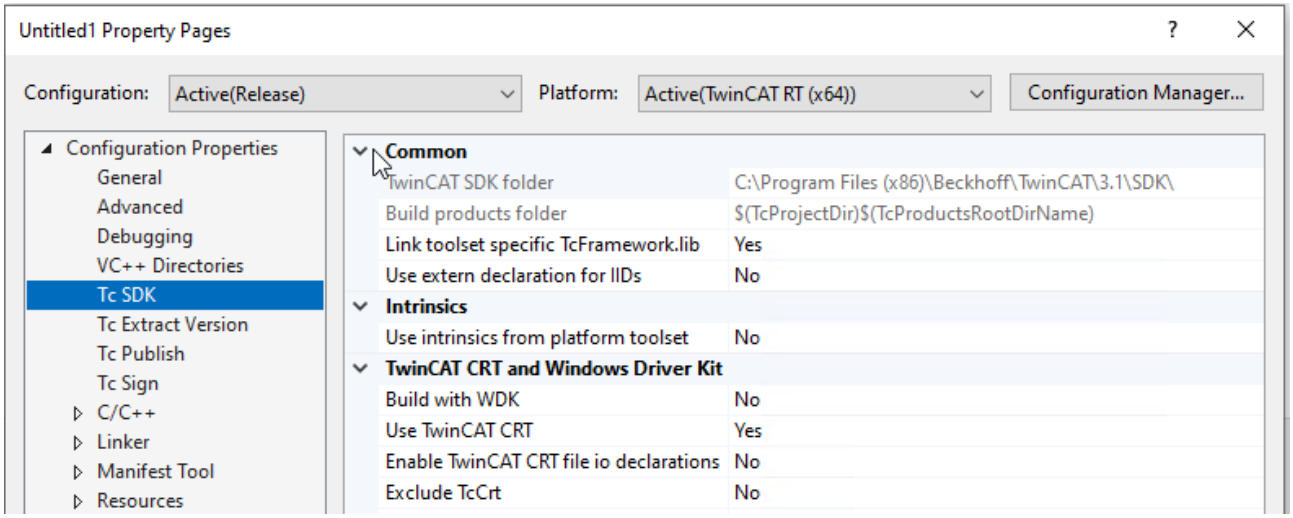


Neben den Visual Studio C++-Dialogen zu den Einstellungen existieren TwinCAT Seiten:



Diese werden auf den Unterseiten beschrieben.

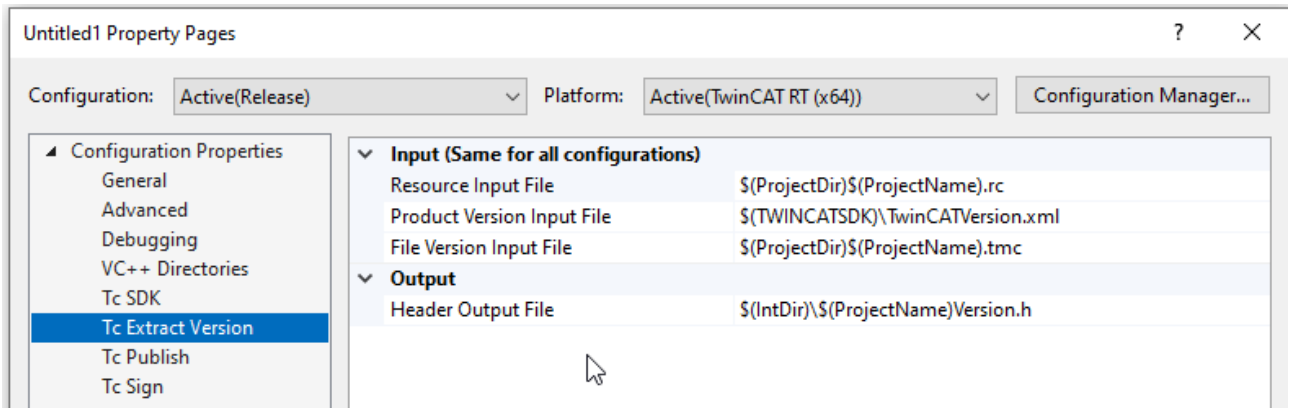
12.1.1 Tc SDK



Einstellungen für das TwinCAT SDK

Kategorie	Feld	Beschreibung
Common	TwinCAT SDK folder	Datei-Ordner, der das TwinCAT SDK bereitstellt, zeigt den Wert der Umgebungsvariable TWINCATSDK.
Intrinsics	Use intrinsics from platform toolset	Die Intrinsics sollen verwendet werden.

12.1.2 Tc Extract Version

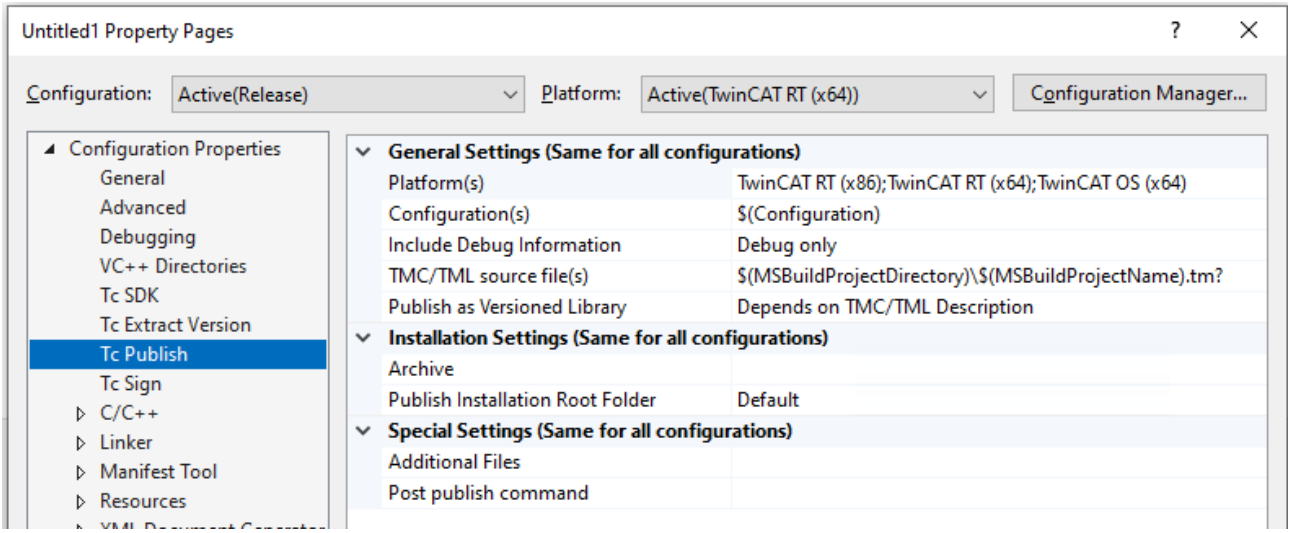


Aus dem Projekt wird eine Versionsinformation in einer Header-Datei bereitgestellt sowie für den Build-Prozess verwendet.

Wenn die *.rc*-Datei Versionsinformationen enthält, wird die Header-Datei hieraus generiert. Bei versionierten C++-Projekten werden die Versionsinformationen aus der TMC-Datei gelesen und in der *.rc*-Datei werden die Makros aus der generierten Header-Datei genutzt.

Kategorie	Feld	Beschreibung
Extract Version	Resource Input File	Die zu berücksichtigende .rc-Datei.
	Product Version Input File	TMC-Datei, welche für versionierte Projekte die Produkt-Version enthält.
	File Version Input File	TMC-Datei, welche für versionierte Projekte die File-Version enthält.
	Header-Output File	Header Datei, in welcher die Information bereitgestellt wird.

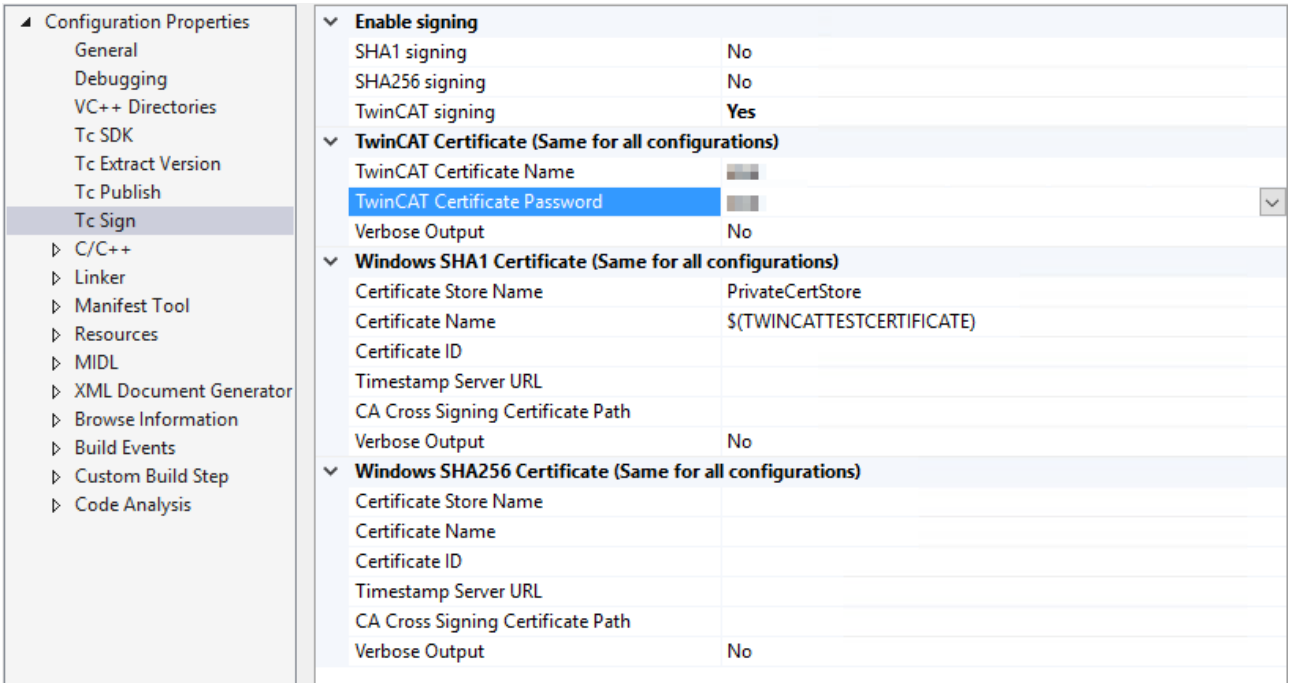
12.1.3 Tc Publish



Information über das Publishen (Veröffentlichen) von Modulen.

Kategorie	Feld	Beschreibung
General Settings	Platform(s)	Welche Plattformen sollen bei einem Publish gebaut werden?
	Configuration(s)	Auswahl, ob Debug / Release gebaut werden sollen.
	Include Debug Information	Für welche Konfigurationen sollen die Debug-Symbole (PDBs) im Repository bereitgestellt werden?
	TMC / TML source file(s)	TMC / TML Dateien aus dem Projekt, die den Ausgangpunkt für den Publish-Vorgang darstellen.
	Publish as Versioned Library	Soll das Publishen in das Repository [▶ 53] erfolgen?
Installation Settings	Archive	Dateipfad für ein Archiv. Endungen .zip (für ein ZIP-Archiv) und .exe (für ein selbst-extrahierendes ZIP Archiv) sind zulässig. Beides enthält den Inhalt für ein Repository (versionierte C++ Projekte) bzw. CustomConfig/Modules (nicht versionierte C++ Treiber) auf einem anderen Engineering System.
	Publish Installation Root Folder	Auf dem lokalen System wird bei „None“ keine Installation vorgenommen. Die Dateien liegen lediglich unter TWINCATSDK/_products/TcPublish bereit. Ein Archiv kann erstellt werden, um diese Dateien manuell auf ein anderes System zu übertragen und dort zu installieren. Bei „Default“ wird auf dem lokalen System eine Installation in das Repository (versionierte C++ Projekte) bzw. CustomConfig/Modules (nicht versionierte C++ Treiber) vorgenommen.
Special Settings	Additional Files	Hinzufügen von zusätzlichen Dateien in den Publish-Prozess, welche bei der Installation im deploy-Unterverzeichnis abgelegt werden.
	Post publish command	Ausführen eines Kommandos nach dem Publish z. B. zum Aufräumen.

12.1.4 Tc Sign



TwinCAT Module müssen signiert [▶ 23] werden, was an dieser Stelle konfiguriert werden kann.

Kategorie	Feld	Beschreibung
Enable Signing	SHA1 signing	Soll eine Betriebssystem-Signatur, die für das Betriebssystem nötig ist, vorgenommen werden?
	SHA256 signing	Soll eine Betriebssystem-Signatur, die für das Betriebssystem nötig ist, vorgenommen werden?
	TwinCAT signing	Soll mit einem TwinCAT Nutzerzertifikat signiert werden? Dies ist für den TwinCAT Loader [▶ 54] nötig.
TwinCAT Certificates Diese Parameter werden für alle Konfigurationen wie z. B. Debug und Release benutzt.	TwinCAT Certificate Name	Name der Zertifikatsdatei (Verzeichnis: C:\ProgramData\Beckhoff\TwinCAT\3.1\CustomConfig\Certificates). Alternativ kann die Umgebungsvariable TcSignTwinCatCertName auf den Namen der Zertifikatsdatei gesetzt werden.
	TwinCAT Certificate Password	Passwort, das das TwinCAT-Nutzerzertifikat schützt (Im Klartext abgelegt, ggf. leer lassen). Um das Passwort des TwinCAT-Nutzerzertifikates nicht im Projekt abzulegen, wo es beispielsweise auch in einer Versionsverwaltung landen würde, kann das TcSignTool [▶ 59] verwendet werden.
	Verbose Output	Sollen erweiterte Informationen während der Signatur ausgegeben werden?
Windows Certificate (SHA1) Nur aus Kompatiblitätsgründen weiterhin enthalten – bitte umgehend auf TwinCAT Certificates umstellen	Certificate Store Name	Name der Zertifikatsablage im Zertifikatsmanager des Betriebssystems.
	Certificate Name	Name des Zertifikates in der Zertifikatsablage.
	Certificate ID	ID des Zertifikates.
	Timestamp Server URL	URL des Timestamp-Servers zur Verwendung während der Signatur. Dieser wird von verschiedenen CAs bereitgestellt.
	CA Cross Signing Certificate Path	Pfad zum Cross Signing Zertifikat.
	Verbose Output	Sollen erweiterte Informationen während der Signatur ausgegeben werden?
Windows Certificate (SHA256) Nur aus Kompatiblitätsgründen weiterhin enthalten – bitte umgehend auf TwinCAT Certificates umstellen	Certificate Store Name	Name der Zertifikatsablage im Zertifikatsmanager des Betriebssystems.
	Certificate Name	Name des Zertifikates in der Zertifikatsablage.
	Certificate ID	ID des Zertifikates.
	Timestamp Server URL	URL des Timestamp-Servers zur Verwendung während der Signatur, wird von der CA bereitgestellt.
	CA Cross Signing Certificate Path	Pfad zum Cross Signing Zertifikat.
	Verbose Output	Sollen erweiterte Informationen während der Signatur ausgegeben werden?

12.2 Dateibeschreibung

Bei der Entwicklung von TwinCAT C++ Modulen ist ein direkter Umgang mit Dateien des Dateisystems möglich. Dies ist von Interesse, entweder um zu verstehen wie das System funktioniert oder für spezielle Anwendungsfälle, wie z. B. manuelle Dateiübertragung, usw.

Hier eine Liste der Dateien, die in Zusammenhang mit C++ Modulen stehen.

Datei	Beschreibung	Weitere Informationen
Engineering / XAE		
*.sln	Visual Studio Solution-Datei, beherbergt TwinCAT- und Nicht-TwinCAT-Projekte	
*.tsproj	TwinCAT Projekt, Sammlung aller verschachtelten TwinCAT-Projekte, wie TwinCAT C++ oder TwinCAT SPS-Projekt	
_Config/	Ordner enthält weitere Konfigurationsdateien *(*.xti)*, die zum TwinCAT-Projekt gehören.	Siehe Menü Tools| Options| TwinCAT| XAE-Environment| File Settings
_Deployment/	Ordner für kompilierte TwinCAT C++ Treiber	
*.tmc	TwinCAT Module Class Datei (XML-basiert)	Siehe TwinCAT Module Class Editor (TMC) [▶ 92]
*.rc	Ressourcendatei	Siehe Setzen von Version/ Herstellerinformationen [▶ 229]
.vcxproj.	Visual Studio C++ Projektdateien	
*ClassFactory.cpp/.h	Class Factory für diesen TwinCAT Treiber	
*Ctrl.cpp/.h	Treiber hochladen und entfernen für TwinCAT UM Plattform	
*Driver.cpp/.h	Treiber hochladen und entfernen für TwinCAT RT Plattform	
*Interfaces.cpp/.h	Deklaration der TwinCAT COM Schnittstellenklassen	
*W32.cpp./.def/.idl		
*.cpp/.h	Eine C++/Header-Datei pro TwinCAT Modul im Treiber. Benutzercode hier einfügen.	
Resource.h	Wird von **.rc* Datei benötigt	
TcPch.cpp/.h	Wird für die Erstellung von vorkompiliertem Header verwendet	
%TC_INSTALLPATH% \Repository\<Vendor>\<PrjName>\<Version>\<Platform>*.tmx	Kompilierter Treiber, der über das TcLoader geladen wird. TwinCAT 3.1.4024.x: *C:\TwinCAT\3.1\Repository\C++ Module Vendor\Untitled1\0.0.0.1\TwinCAT RT *\Unititled1.tmx* Ab TwinCAT 3.1.4026.x: C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository\C++ Module Vendor\Untitled1\0.0.0.1\TwinCAT RT *\Unititled1.tmx	Siehe Versionierte C++ Projekte [▶ 53]
%TC_INSTALLPATH% \CustomConfig\Modules*	Veröffentlichtes TwinCAT C++ Projekt normalerweise TwinCAT 3.1.4024.x: *C:\TwinCAT\3.1\CustomConfig\Modules** Ab TwinCAT 3.1.4026.x: C:\ProgramFiles (x86)\Beckhoff \TwinCAT\3.1\Config\Modules*	Siehe Module exportieren
Laufzeit / XAR		
%TC_BOOTPRJPATH% \CurrentConfig*	Derzeitiges Konfigurationssetup	

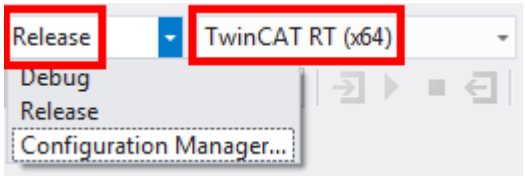
Datei	Beschreibung	Weitere Informationen
	Windows: TwinCAT 3.1.4024.x: *C:\TwinCAT\3.1\Boot* Ab TwinCAT 3.1.4026.x: Windows: C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot *TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot*	
%TC_DRIVERAUTOINSTALLPATH% *.sys/pdb	Kompilierter, plattformspezifischer Treiber, der über das Betriebssystem geladen wird. *Windows:* TwinCAT 3.1.4024.x: *C:\TwinCAT\3.1\Driver\AutoInstall (vom System geladen)* Ab TwinCAT 3.1.4026.x: *<nicht verfügbar>* *Bitte migrieren zu TMX basierten C++ Projekten* *TwinCAT/BSD:* *<nicht verfügbar>*	
%TC_INSTALLPATH%\Boot\Repository\<Vendor>\<Prj Name>\<Version>*.tmx	Kompilierter Plattform-spezifischer Treiber, der über den TcLoader geladen wird. *Windows:* TwinCAT 3.1.4024.x: *C:\TwinCAT\3.1\Boot\Repository\C++ Module Vendor\Untitled1\0.0.0.1\Untitled1.tmx* Ab TwinCAT 3.1.4026.x: 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 Module Instance Datei Beschreibt Variablen des Treibers Dateiname lautet *ObjectID.tmi* Windows: TwinCAT 3.1.4024.x: *C:\TwinCAT\3.1\Boot\TMI\OTCID.tmi* Ab TwinCAT 3.1.4026.x: Windows: C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\TMI\OTCID.tmi *TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot/TMI/OTCID.tmi*	
Temporäre Dateien		
*.sdf	IntelliSense Datenbank	
*.suo / *.v12.suo	Benutzerspezifische und Visual Studio-spezifische Dateien	
*.tsproj.bak	Automatisch generierte Sicherungsdatei von *tsproj*	

Datei	Beschreibung	Weitere Informationen
ipch/	Für vorkompilierten Header erstelltes Zwischen-Verzeichnis	

Ablauf der Kompilierung

Hier wird der Ablauf beschrieben, den ein Build oder Rebuild auf einem TwinCAT C++ Projekt im TwinCAT Engineering XAE auslöst. Dies ist beispielsweise zu berücksichtigen, wenn Firmen-spezifische Umgebungen und Bauprozesse integriert werden sollen.

Welche Konfiguration bei einem Build oder Rebuild gebaut wird, hängt von der aktuellen Auswahl im Visual Studio ab:



Die richtige Ziel-Architektur (TwinCAT RT (x64) hier) wird durch eine Auswahl des Zielsystems passend gesetzt.

Der **Configuration Manager** erlaubt die dezidierte Einstellung der Build-Konfiguration.

Bei Auswahl von **Build** oder **Rebuild** (und damit auch beim **Activate Configuration**) laufen die folgenden Schritte ab:

1. Die Quellen liegen im jeweiligen Projekt-Verzeichnis.
2. Die Kompilate werden architektur-spezifisch erzeugt unterhalb des Projekt-Verzeichnisses _products\TwinCAT RT (x64)\Release\<ProjectName>
3. Nach dem Linkvorgang wird danach die .tmx/.pdb-Datei in dem Projekt-Verzeichnis abgelegt.
4. Ein Druck auf den Activate Configuration-Button führt dazu, dass .tmx/.pdb von dem Engienering Repository auf das Zielsystem übertragen wird (ggf. ist das eine lokale Kopie).

Bei Auswahl von Publish auf dem Projekt baut für die ausgewählten Plattformen und legt das Ergebnis im Engineering-Repository ab: C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository\<vendor>\<ProjectName>\<version>\TwinCAT RT (x64)\Release\, wobei auch die TMC Datei aktualisiert wird.

12.3 Online-Change

Ab TwinCAT 3.1. Build 4024.0

Die hier beschriebene Funktionalität ist ab TwinCAT 3.1. 4024.0 verfügbar.

TwinCAT 3.1 unterstützt das Austauschen von C++ Modulen zur Laufzeit d. h. ohne Unterbrechung des Echtzeitprogramms.

Hierfür werden unterschiedliche Versionen [▶ 53] eines TwinCAT Executable (TMX) auf dem Zielsystem vorgehalten.

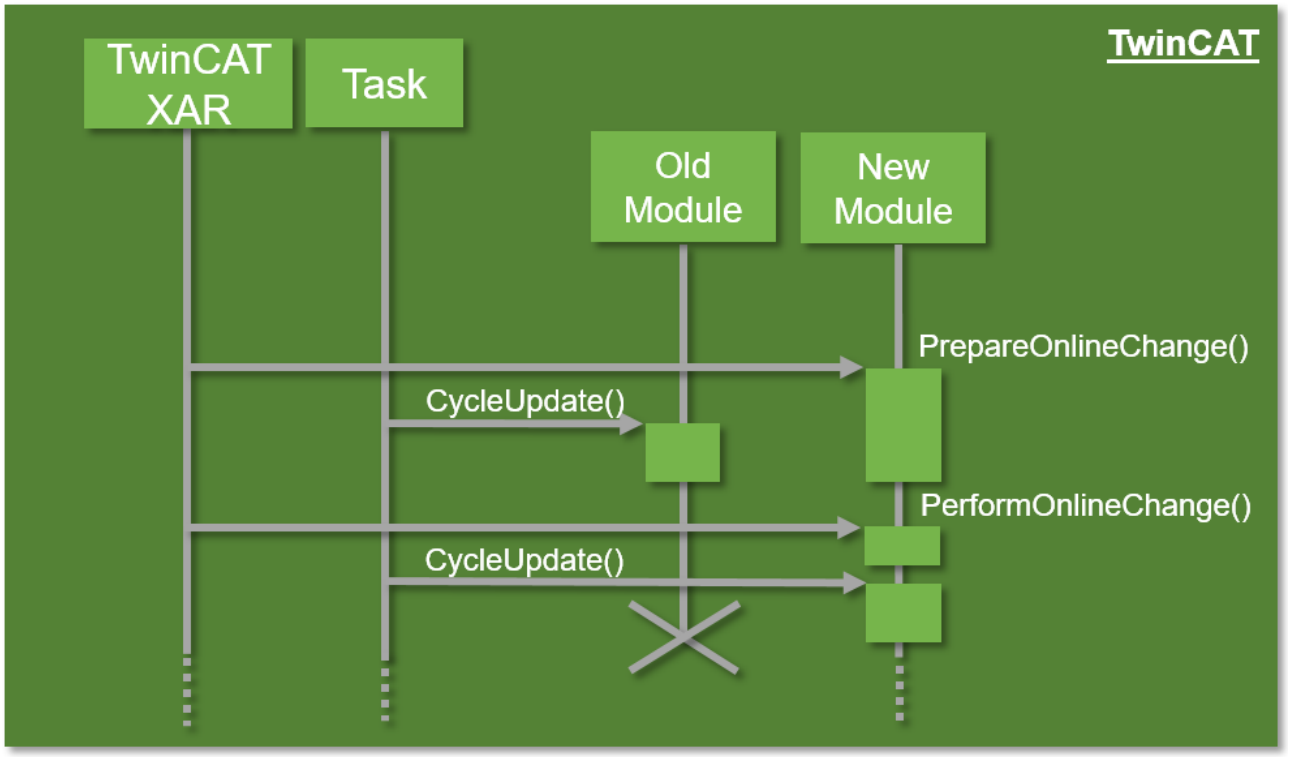
Für alle Modul-Instanzen aus einem TMX kann durch das Engineering eine Umschaltung zwischen den Versionen veranlasst werden.

Der Ablauf ist grob skizziert:

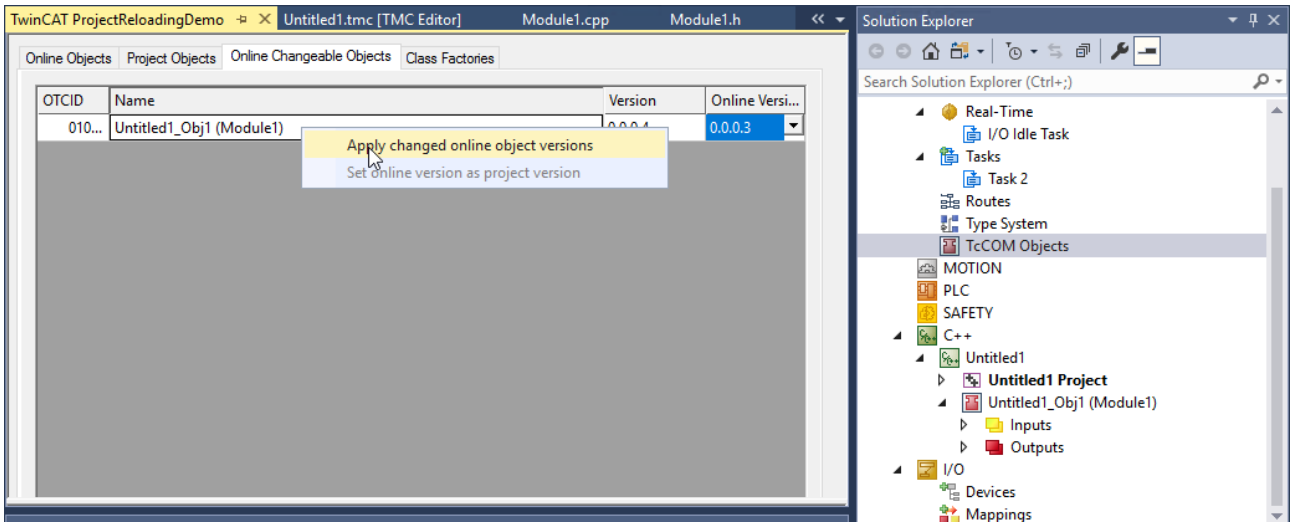
✓ Online-Change fähiges Modul in TMX

1. TwinCAT instanziiert das neue Modul. Das alte Modul wird weiterhin von der Task zyklisch aufgerufen.
2. TwinCAT ruft ITcOnlineChange::PrepareOnlineChange() des neuen Moduls auf.
   Dieser Aufruf kann auf das alte Modul zugreifen und Daten übernehmen, die sich durch die zyklischen Aufrufe des Moduls nicht ändern - beispielsweise Parameterwerte.

3. TwinCAT ruft ITcOnlineChange::PerformOnlineChange() des neuen Moduls auf.
   Dieser Aufruf kann auf das alte Modul zugreifen und wiederum Daten übernehmen, die sich zuvor zyklisch verändert haben. Dieser Aufruf wird ausgeführt, wenn kein zyklischer Aufruf durch eine Task erfolgt. Das alte Modul wird **nicht** nochmal wieder durch die Task aufgerufen, sondern das neue Modul wird aufgerufen. Damit diese Umschaltung von einem Taskzyklus zum nächsten erfolgen kann, muss die Methode PerformOnlineChange() entsprechend wenig Rechenzeit in Anspruch nehmen.
4. Nach Abschluss wird die Task das neue Modul zyklisch aufrufen.



Durch diesen Dialog im Engineering kann der Online-Change vorgenommen werden.



Bei dem Umgang mit dem Online-Change sind deswegen einige Aspekte zu beachten:

- Das Projekt muss eine Versionierung vorsehen.
- Die DataAreas werden für diese Module außerhalb des TcCOM Moduls gehalten und den Modulen bereitgestellt. Diese brauchen also die Daten sowie das Mapping der Symbole in den DataAreas nicht zu beachten.
- Die DataAreas des Modules dürfen sich nicht ändern.
- Referenzen auf interne Datenstrukturen dürfen nicht rausgereicht werden. Zugriffe müssen immer über Interfaces erfolgen, die per TcQueryInterface geholt werden, da diese Referenzen bei einem Online-Change aktualisiert werden.

Nach einem Neustart wird TwinCAT den Treiber in der initialen Version der Module starten.

12.4 Limitierungen

TwinCAT 3 C++ Module [▶ 39] werden im Windows Kernel-Modus ausgeführt. Deswegen müssen die Entwickler einige Limitierungen beachten:

- Win32 API [▶ 157] ist nicht im Kernel-Modus verfügbar
- Windows Kernel Mode API darf nicht direkt genutzt werden.
  TwinCAT stellt in dem TwinCAT SDK Funktionen bereit, die unterstützt werden.
- Usermode Bibliotheken (DLL) können nicht verwendet werden. (siehe Bibliotheken von Drittanbietern [▶ 243])
- Speicherplatz für dynamische Allokation im Echtzeit-Kontext ist durch den Router-Speicher begrenzt (kann im Verlauf des Engineerings konfiguriert werden), siehe Speicherallokation [▶ 158].
- Untermenge der C++ Laufzeit-Bibliotheksfunktionen (CRT) wird unterstützt
- C++ Ausnahmen (C++ Exceptions) werden nicht unterstützt.
- Laufzeit-Typinformation (RTTI [▶ 157], Runtime Type Information) wird nicht unterstützt.
- Untermenge von STL wird unterstützt (siehe STL / Container [▶ 222])
- Unterstützung für Funktionen aus math.h durch TwinCAT Implementierung (siehe Mathematische Funktionen [▶ 218])

TwinCAT-Funktionen als Ersatz für Win32 API Funktionen

Die originale Win32 API ist nicht im Windows Kernel-Modus verfügbar. Aus diesem Grund ist hier eine Liste der normalerweise verwendeten Funktionen aus der Win32 API und was stattdessen in TwinCAT verwendet werden kann:

Win32API	TwinCAT Funktionalität
WinSock	TF6311 TCP/UDP Echtzeit
Message Boxen	Tracing [▶ 223]
Datei-I/O	Siehe Schnittstelle ITcFileAccess [▶ 172], Schnittstelle ITcFileAccessAsync [▶ 180] und Beispiel19: Synchroner Dateizugriff [▶ 308], Beispiel20: FileIO-Write [▶ 308], Beispiel20a: FileIO-Cyclic Read / Write [▶ 309]
Synchronisation	Siehe Beispiel11a: Modulkommunikation: Methodenaufruf C++-Modul nach C++-Modul [▶ 302]
Visual C CRT	Siehe RtlR0.h

RTTI dynamic_cast Funktion in TwinCAT

TwinCAT hat keine Unterstützung für dynamic_cast<>.

Stattdessen kann die TCOM-Vorgehensweise möglicherweise verwendet werden. Definieren Sie eine Schnittstelle ICustom, die von ITcUnknown abgeleitet ist und die die Methoden enthält, die von einer abgeleiteten Klasse aufgerufen werden. Die Basisklasse CMyBase wird von ITcUnknown abgeleitet und implementiert dieses Interface. Die Klasse CMyDerived wird von CMyBase und von ICustom abgeleitet. Sie überschreibt die TcQueryInterface Methode, die dann anstelle von dynamic cast verwendet werden kann.

TcQueryInterface kann auch zur Darstellung der IsType() Funktion mittels Auswertung des Rückgabewerts verwendet werden.

Siehe Schnittstelle ITcUnknown [▶ 197].

12.5 Speicherallokation

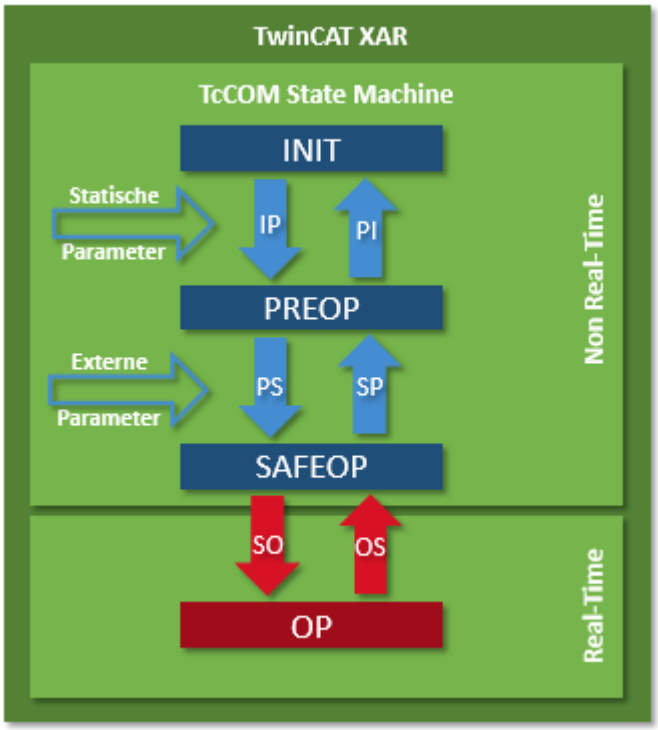
Im Allgemeinen wird empfohlen, Speicher mit Hilfe von Member-Variablen der Modulklasse zu reservieren. Dies wird automatisch für im TMC Editor definierte Datenbereiche gemacht.
Es besteht auch die Möglichkeit, Speicherplatz dynamisch zu allokieren und freizugeben.

- Operator new / delete
- TcMemAllocate / TcMemFree

Diese Speicherallokation kann in den Transitionen [▶ 48] oder dem OP-State der Statemachine genutzt werden.

Wird dabei die Speicherallokation in einem Nicht-Echtzeitkontext vorgenommen, dann wird der Speicher im non-paged Pool des Betriebssystems (blau im Diagramm) allokiert. Im TwinCAT-Echtzeitkontext wird der Speicher im Routerspeicher (rot im Diagramm) allokiert.

Die Freigabe des Speichers kann ebenfalls in den Transitionen oder dem OP-State erfolgen, wobei empfohlen wird, immer in der „symmetrischen“ Transition den Speicher freizugeben, z. B. allokieren in PS und freigeben in SP.



Bei der Nutzung von statischen Variablen sind einige Besonderheiten [▶ 159] zu beachten.

Zusätzlich können einige Hinweise zur Implementierung der Transitionen bereitgestellt werden:

- Transitionen IP, PS and SP, PI werden im Non Real-Time Context (z.B. Windows Kernel Mode) ausgeführt.
- Transitionen SO, OS werden hingegen im TwinCAT RT Task Context ausgeführt (Genauer: TcCOM Server Task).
- Transitionen müssen nach dem Pattern "all-or-nothing" implementiert werden. D.h. alle Schritte die erfolgreich ausgeführt werden müssen rückgängig gemacht werden, wenn ein folgender Schritt fehlschlägt.
- Wenn Speicher im OP Zustand allokiert wird, muss dieser in der OS Transition spätestens freigegeben werden.
- Die Transitionen OS, SP, PI sollten so implementiert werden, dass sie immer erfolgreich sind. Es sollen keine Fehlercodes zurückgegeben werden.
- Transitionen sollten symmetrisch implementiert werden d.h. OS und SO sollten invers sein, dito für PS <-> SP und IP <->PI.

- Transition SO sollte als letzten Schritt das Modul bei der Task anmelden (falls ein ITcCaller vorliegt)
- Transition OS sollte als ersten Schritt das Modul bei der Task abmelden (falls ein ITcCaller vorliegt)
- Semaphoren
  - Können im Non Real-Time Context kreiert und gelöscht werden
  - Nutzung (Pend / Post) ist nur im Real-Time Context erlaubt
- Mehrere Module können gegenseitig in den Zuständen aufeinander warten. Es wird über die InitSeq (default „PSO“) angegeben, in welchen Zuständen die Modul Instanzen jeweils auf das Weiterschalten des Gesamtsystems (und damit aller anderen Module) warten soll.

12.6 Statische Variablen

Speicherallokation zwischen Windows- und Echtzeitkontext im Zusammenhang mit statischen Variablen

Wenn globale Instanzen verwendet werden, ist auf folgendes zu achten:

- Es dürfen maximal 128 globale Instanzen insgesamt existieren.
  Dabei ist zu beachten, dass auch durch TwinCAT-eigene Module globale Instanzen genutzt werden können.
- Bei Überschreitung der Anzahl werden die Destruktoren nicht mehr aufgerufen.
- Ab TwinCAT 3.1 4026 wird bei Überschreiten des Limits eine Meldung im „Output“ des Engineerings ausgegeben:
  Too many static objects detected on shutdown of class factory <ProjectName>.tmx. Max Number of static objects exceeded by N item(s). This may result in a memory leak.
- Der Programmcode kann die aktuell genutzten bzw. verfügbaren globalen Instanzen abfragen:
  ```
  #include "CrtInit.h"
  int overflow = cppGetNumberHandlerOverflow();
  int free = TcCrtGetFreeNumberAtexitHandler();
  ```
- Speicher, der im Echtzeit-Kontext allokiert wird, muss beispielsweise in der OS-Transition freigegeben werden, sodass dies nicht durch den Destruktor passiert.
- Wenn der Konstruktor ausgeführt wird, kann noch nicht auf Ressourcen des TwinCAT Systems zugegriffen werden, weil die Verbindung über die Classfactory zum TcCOM Objektserver noch nicht hergestellt ist. Weiter unten sehen Sie, wie damit umzugehen ist.

In diesem Code sind 3 Beispiele für globale Instanzen:

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

MyClassA A;

class MyClassB
{
    static MyClassA Value;
};

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

Singletons für die Initalisierung globaler Instanzen

Um den Zeitpunkt hinauszuzögern, können globale bzw. sog. Singleton Instanzen auch als lokale statische Objekte angelegt werden. Ganz wichtig bei der Verwendung von globalen Klasseninstanzen ist, dass die Destruktoren spät aufgerufen werden und zu diesem Zeitpunkt keine Verbindung mehr zum TwinCAT System besteht, sodass Ressourcen von TwinCAT vorher freigegeben werden müssen.

Hier ist ein Beispielcode, der ein Singleton in TwinCAT zeigt:

```
#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;
};
```

Hier wird die Verwendung gezeigt:

```
// 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 Multitask-Datenzugriffs-Synchronisation

Wenn von mehreren Tasks auf dieselben Daten zugegriffen wird, kann es je nach Task-/Echtzeitkonfiguration vorkommen, dass die Tasks gleichzeitig auf dieselben Daten zugreifen. Wenn die Daten dabei von mindestens einer der Tasks geschrieben werden, können die Daten während oder nach einer Änderung einen inkonsistenten Zustand haben. Um dies zu verhindern, müssen alle konkurrierenden Zugriffe synchronisiert werden, sodass zu einem Zeitpunkt nur von höchstens einer Task auf die gemeinsam genutzten Daten zugegriffen werden kann.

Wenn also von mehreren Tasks auf dieselben Daten zugegriffen wird und bei mindestens einem dieser Zugriffe die Daten geschrieben werden, müssen alle lesenden und schreibenden Zugriffe synchronisiert werden. Dies gilt unabhängig davon, ob die Tasks auf einem oder mehreren CPU Kernen laufen.

⚠ WARNUNG

Inkonsistenzen und weitere Gefahren durch ungesicherten Datenzugriff

Werden konkurrierende Zugriffe nicht synchronisiert, so besteht die Gefahr eines inkonsistenten oder ungültigen Datensatzes. Je nachdem wie die Daten im weiteren Programmverlauf genutzt werden, kann dies ein Fehlverhalten des Programms, eine ungewünschte Achsbewegung oder auch den plötzlichen Programmstillstand zur Folge haben. Abhängig von der gesteuerten Anlage können Schäden an Anlage und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

Allgemeine Synchronisationsmöglichkeiten zwischen den TcCOM Modulen sind bereits im Kapitel Modul zu Modul Kommunikation [▶ 50] beschrieben. Hierbei wurden die folgenden Mechanismen beschrieben:

- Austausch von Daten über das Prozessabbild (IO Mapping), wobei der Zyklusübergang und damit TwinCAT die Synchronität der Daten zwischen den Quellen und Zielen sicherstellt.
- Methoden-Aufrufe über Schnittstellen, in denen CriticalSections genutzt werden können.
- ADS, welches Daten im Sinne eines Transportmediums transferiert und dadurch die Synchronität der Daten sicherstellt.
- gemeinsamer Datenbereich, der über eine CriticalSection zu schützen ist.

Idealerweise sollte jedoch versucht werden die Notwendigkeit einer Synchronisation zu vermeiden, ansonsten ist der einfachste Weg meistens der Austausch über das Prozessabbild, welche die Daten zwischen den Zyklen von den Ausgangs-Prozessabbildern zu den Eingangs-Prozessabbildern kopieren und dadurch für einen konsistenten Zustand sorgen.
Wenn dieses nicht ausreicht und auf Daten aus unterschiedlichen Kontexten zugegriffen werden muss, kann einer der folgenden Möglichkeiten genutzt werden.

Es ist dabei wichtig, die Task-Kontexte zu unterscheiden, welche auf die Daten zugreifen sollen. In der TwinCAT Runtime wird zwischen den Windows Kernel Mode Thread-Kontexten (kurz: Windows Kontext) sowie den TwinCAT Realtime Task-Kontexten (RT-Kontext) unterschieden. Während der Initialisierung von TwinCAT Modulen werden die Transitionen IP, PS, SP, PI im Windows Kontext ausgeführt. Die Transitionen SO und OS werden im RT-Kontext ausgeführt.

Das SDK bietet für solche Szenarien entsprechende Synchronisierungsmöglichkeiten. Trotzdem sollten die TcCOM Module in einer Weise konzipiert werden, dass sie zumindest im Windows-Kontext unabhängig voneinander sind und damit ohne Synchronisierungs-Bedarf auskommen. Die Transitionen im RT-Kontext können bei Bedarf CriticalSections nutzen.

Für die CriticalSection und Semaphoren gilt, dass im Fall einer Sperre die Task auf die Freigabe wartet, in der Zwischenzeit der Core für andere Tasks freigegeben wird. Bei CriticalSections erbt die jeweils aktive Task dabei ggf. die Priorität der wartenden Task („Priority Inheritence“).

12.7.1 CriticalSections

Instanzen von Critical Sections werden durch die TwinCAT Realtime angelegt. Diese Instanz kann sowohl im Windows-Kontext wie auch im RT-Kontext initialisiert werden. Genutzt werden kann die Critical Section Instanz nur im RT-Kontext.

Die TwinCAT Realtime nutzt dabei “priority inheritance” um zu verhindern, dass eine Task mit geringerer Priorität indirekt einen Task mit hoher Priorität blockiert.

CCriticalSectionInstance

Die Klasse CCriticalSectionInstance bietet die Schnittstelle an, um Critical Sections zu handhaben und besitzt den nötigen Speicher. Um eine Critical Section anzulegen, benötigt die Klasse die Objekt-ID der TwinCAT Realtime Instanz, welche über OID_TCRTIME_CTRL bereitsteht, sowie eine Referenz zu dem TwinCAT Objekt-Server über einen Pointer auf ITComObjectServer.

Methoden:

- CCriticalSectionInstance(OTCID oid=0, ITComObjectServer* ipSrv=NULL);
  Der Default Konstruktor, der die Objekt ID des Critical Section Providers initialisiert. Wenn der Pointer zu dem Objekt-Server gegeben wird, wird die Critical Section auch initialisiert.
- ~CCriticalSectionInstance();
  Der Destruktor löscht die Cricital Section Instanz.
- void SetOidCriticalSection(OTCID oid);
  Setzt den CriticalSection Provider, welcher durch die TwinCAT Realtime gegeben ist und dessen Objekt ID über OID_TCRTIME_CTRL verfügbar ist.
- bool HasOidCriticalSection();
  Gibt TRUE zurück, wenn die Obekt-ID zu einem Wert ungleich 0 gesetzt ist – sonst FALSE. Die Objekt-ID wird dabei nicht überprüft, ob sie zu einem Critical Section Provider gehört.
- bool IsInitializedCriticalSection();
  Gibt TRUE zurück, wenn die Criticals Section erfolgreich initialisiert wurde.
- HRESULT CreateCriticalSection(ITComObjectServer* ipSrv);
  Wenn die Critical Section initialisiert ist, wird diese gelöscht und eine neue Critical Section initialisiert. Gibt im Erfolgsfall S_OK zurück. Fehlerfälle werden durch die Rückgabe der Fehlercodes angezeigt:

ADS_E_INVALIDPARM	Ungültiger Critical Section Provider
ADS_E_NOINTERFACE	Die Objekt ID ist auf eine Referenz gesetzt, die kein Critical Section Provider darstellt, also ITcRTime nicht implementiert.
E_FAIL	Interner Fehler von dem Ciritical Section Provider.

- HRESULT CreateCriticalSection(OTCID oid, ITComObjectServer* ipSrv);
  Initalisiert die Cristical Section. DeleteCriticalSection() muss aufgerufen werden, wenn diese Methode erneut verwendet wird. Rückgabewerte wie in CreateCriticalSection(ITComObjectServer* ipSrv);
- HRESULT DeleteCriticalSection(); Critical Section wird gelöscht. Gibt immer S_OK zurück.
- HRESULT EnterCriticalSection(); Blockiert, bis die Critical Section freigegeben wurde.
- HRESULT LeaveCriticalSection();
  Verlässt die Critical Section und gibt diese damit wieder frei.
  Es wird MAKE_RTOS_HRESULT(OS_CS_ERR) zurückgegeben, wenn der Aufrufende nicht der Besitzer ist.

EnterCriticalSection() und LeaveCriticalSection() müssen im RT-Kontext aufgerufen werden, ansonsten wird folgender Rückgabewert zurückgegeben:

ADS_E_INVALIDCONTEXT	Rückgabewert, wenn Critical Section außerhalb der RT Kontext betreten wird. Die Critical Section wird nicht betreten.

Wenn diese Methoden genutzt werden, ohne dass die Critical Section initialisiert wurde, wird S_OK zurückgegeben, ohne dass etwas gemacht wird. Alle anderen Methoden können sowohl im RT-Kontext wie auch im Windows-Kontext benutzt werden.

Bei der Klasse CCriticalSectionInstance kann das EnterCriticalSection()mehrfach von derselben Task aufgerufen werden. Es muss dann Leave() genauso oft aufgerufen werden.

CriticalSections erlauben verschachtelte („nested“) Aufrufe und verlangen für jeden EnterCriticalSection()-Aufruf einen zugehörigen LeaveCriticalSection()-Aufruf. Die Freigabe der Critical Section ergibt sich dann beim Aufruf des letzten LeaveCritcalSection().

Beispiel:

Das Beispiel [▶ 302] zeigt eine Nutzung einer CriticalSection um über einen Interface-Methodenaufruf eine gleichzeitigen Zugriff auf ein Datum zu verhindern.

Critical Section Concurrent

Die Klasse CCriticalSectionInstanceConcurrent erlaubt einen konkurrierenden Zugriff (Concurrent access). Dieser kann genutzt werden, wenn mehrere Tasks lesend auf die zu schützenden Daten zugreifen, aber bei einem schreibenden Zugriff alle anderen Zugriffe unterbunden werden müssen.

Methoden:

- CCriticalSectionInstanceConcurrent(UINT concurrent, OTCID oid=0, ITComObjectServer* ipSrv=NULL); Der Parameter “concurrent” definiert die Anzahl der Tasks, die gleichzeitig die Critical Section via CsEnterCriticalSectionConcurrent() betreten können. Wenn “concurrent” 1 ist, arbeitet die Critical Section Concurrent wie eine Critical Section.
  Der Wert 0 ist nicht erlaubt und die Critical Section kann dann nicht initialisiert werden.
- ~CCriticalSectionInstanceConcurrent(); Der Destruktor löscht implizit die Critical Section
- void SetOidCriticalSection(OTCID oid); Setzt den CriticalSection Provider, welcher durch die TwinCAT Realtime gegeben ist und dessen Objekt ID über OID_TCRTIME_CTRL verfügbar ist.
- bool HasOidCriticalSection(); Gibt TRUE zurück, wenn die Obect-ID zu einem Wert ungleich 0 gesetzt ist – sonst FALSE. Die Objekt-ID wird dabei nicht überprüft, ob sie zu einem Critical Section Provider gehört.
- bool IsInitializedCriticalSection(); Gibt TRUE zurück, wenn die Criticals Section erfolgreich initialisiert wurde.
- HRESULT CreateCriticalSection(ITComObjectServer* ipSrv); Wenn die Critical Section initialisiert ist, wird diese gelöscht und eine neue Critical Section initialisiert. Gibt im Erfolgsfall S_OK zurück. Fehlerfälle werden durch die Rückgabe der Fehlercodes angezeigt:

ADS_E_INVALIDPARM	Ungültiger Critical Section Provider
ADS_E_NOINTERFACE	Die Objekt ID ist auf eine Referenz gesetzt, die kein Critical Section Provider darstellt, also ITcRTime nicht implementiert.
E_FAIL	Interner Fehler von dem Critical Section Provider.

- HRESULT CreateCriticalSection(OTCID oid, ITComObjectServer* ipSrv);
  Initialisiert die Cristical Section. DeleteCriticalSection() muss aufgerufen werden, wenn diese Methode erneut verwendet wird. Rückgabewerte wie in CreateCriticalSection(ITComObjectServer* ipSrv);
- HRESULT DeleteCriticalSection(); Critical Section wird gelöscht. Gibt immer S_OK zurück.
- HRESULT EnterCriticalSection(); Blockiert, bis die Critical Section freigegeben wurde und exklusiv betreten werden kann.
- HRESULT LeaveCriticalSection(); Verlässt die Critical Section und gibt diese damit wieder frei. Es wird MAKE_RTOS_HRESULT(OS_CS_ERR) zurückgegeben, wenn der Aufrufende nicht der Besitzer ist.
- HRESULT EnterCriticalSectionConcurrent(); Betritt die Critical Section parallel mit anderen Tasks. Die Anzahl der parallelen Zugriffe ist durch den Parameter “concurrent” im Konstruktor definiert. Wenn diese maximale Anzahl erreicht ist, blockiert der Aufruf, bis einer der parallelen Zugriffe beendet wurde.

EnterCriticalSection(), EnterCriticalSectionConcurrent() und LeaveCriticalSection() müssen im RT-Kontext aufgerufen werden, ansonsten wird folgender Rückgabewert zurückgegeben:

ADS_E_INVALIDCONTEXT	Rückgabewert, wenn Critical Section außerhalb der RT Kontext betreten wird. Die Critical Section wird nicht betreten.

Wenn diese Methoden genutzt werden, ohne dass die Critical Section initialisiert wurde, wird S_OK zurückgegeben, ohne dass etwas gemacht wird. Alle anderen Methoden können sowohl im RT-Kontext wie auch im Windows-Kontext benutzt werden.

Bei der Klasse CCriticalSectionInstanceConcurrent, ist es nicht erlaubt, dass die Methode EnterCriticalSection() mehrfach von derselben Task aufgerufen wird, wenn der Parameter concurrent beim Initialisieren größer als 1 ist.

12.7.2 Semaphoren

Semaphoren werden genutzt, um den Zugriff auf limitierte Ressourcen zu synchronisieren. Sie können auch genutzt werden um eine Benachrichtigung (Notification Event) zu realisieren. Semaphoren werden durch die TwinCAT Realtime durch die Klasse CSemaphoreInstance bereitgestellt.

CSemaphoreInstance

Die Klasse CSemaphoreInstance bietet die Schnittstelle an, um Semaphoren zu handhaben. Um eine Semaphore anzulegen, benötigt die Instanz die Objekt-ID der TwinCAT Realtime Instanz, welche über OID_TCRTIME_CTRL bereitsteht, sowie eine Referenz zu dem TwinCAT Objekt-Server über einen Pointer auf ITComObjectServer.

Methoden:

- HRESULT SemCreate(WORD nCntInit, OTCID oid, ITComObjectServer* ipSrv); Erstellt die Semaphore mit nCntInit als initalen Wert der bereitstehenden Resourcen. Es wird S_OK bei Erfolg zurückgegeben und E_FAIL wenn die Semaphore nicht erzeugt werden kann.
- HRESULT SemDelete(); SemDelete() löscht die Semaphore. Liefert S_OK zurück.
- HRESULT SemPost() SemPost() erhöht die Anzahl der verfügbaren Resourcen. Wenn ein Task auf die Semaphore wartet, wird der Task mit der höchsten Priorität freigegeben d.h. sie kann ihre Ausführung fortsetzen. Gibt bei Erfolg S_OK zurück. Mögliche Fehlercodes:

MAKE_RTOS_HRESULT(51)	Semaphoren Overflow. Beispielsweise wenn der interne Speicher nicht ausreicht.

- HRESULT SemPend(OSTICKS nTimeout); Reduziert die Anzahl der verfügbaren Ressourcen; sollten keine Ressourcen zur Verfügung stehen, blockiert die Task an dieser Stelle.
  SemPend() wartet auf eine Semaphore, bis diese zur Verfügung steht. Ein Timeout kann in OSTICKS angegeben werden und ist damit Prozessor-abhängig. Das ITcRTime Interface in kann genutzt werden, um diese zu berechnen. Im Beispiel TcSemaphoreSample ist hierfür eine Methode TimeoutMsToTicks verwendet worden. Als spezielle Werte für nTimeout existieren:

RTIME_NOWAIT (-1)	Methode liefert sofort S_OK, wenn die Semaphore verfügbar ist. Methode liefert ein Timeout, falls keine Ressource verfügbar ist.
RTIME_ENDLESSWAIT (0)	Methode wartet unendlich lange auf die Verfügbarkeit der Semamphore

Die Methode liefert S_OK, wenn den Semaphoren erfolgreich bezogen wurde; ansonsten:

MAKE_RTOS_HRESULT(10)	Zeigt einen Timeout an. Wenn nTimeout als RTIME_NOWAIT gesetzt wurde, ist diese Semaphore nicht verfügbar.

Beispiel:

Beispiel24: Semaphoren [▶ 314]

12.7.3 Fifo Template Klassen

Die Header Datei Fifo.h stellt unterschiedliche Fifo Template Klassen bereit. Alle Klassen erkennen, wenn ihr Speicher gefüllt ist und werden dann keine neuen Elemente aufnehmen, bis ein Element entfernt wurde. Die Element-Typen können primitive Datentypen („Plain Old Datatypes“, PODs) sein. Es ist nicht möglich Element Typen mit virtuellen Tabellen in einem Fifo abzulegen. Pointer sind (unabhängig vom Typ) erlaubt.

CFiFoListBase

Die Klasse CFifoListBase ist die Basisimplementierung eines Fifos. Diese Template-Klasse hat eine StorageType-Instanz, die dafür zuständig ist Elemente von einem Typen zu speichern. Sie nutzt eine SyncType-Instanz um synchronisierten Zugriff auf den Speicher und enthaltenen Daten zu erhalten.

Es gibt dabei unterschiedliche Möglichkeiten des Schutzes:

- CNoSync / SyncObects.h: Kein Schutz des Zugriffs innerhalb der Fifo. Es kann eine außenliegende CriticalSection genutzt werden.

- CSpinLock / TcSpinlock.h: Schutz durch einen Spinlock, wie oben beschrieben.

Die Methoden Add(), Remove() und Clear() haben auch entsprechende _Unprotected Variationen, welche mit den Methoden Lock() und Unlock() genutzt werden können um eine synchronisierte Abfolge von Add() und Delete() Operationen zu implementieren.

Methoden:

- CFiFoListBase(); Der Konstruktor.
- Lock() und Unlock(); Werden an den die Synchronisierungs-Instanz SyncType weitergereiht.
- Clear(), Clear_Unprotected(); Werden and die Speicher-Instanz StorageType weitergereicht.
- Add(), Add_Unprotected(); Ein Element hinzufügen.
- AddBlock(); Mehrere Elemente hinzufügen. Wen nein Block großer ist als der freie Speicher, werden nur die Elemente hinzugefügt, welche zu dem freien Speicher passen. The Anzahl der hinzugefügten Elemente wird zurückgegeben.
- Remove(), Remove_Unprotected(); Entfernt ein Element aus dem Fifo. Liefert FALSE, wenn kein Element vorhanden war, sonst TRUE.
- RemoveBlock(); Entfernt mehrere Elemente aus dem Fifo. Wenn dort weniger Elemente abgelegt sind, werden nur diese entfernt. Die Anzahl der entfernten Elemente wird zurückgegeben.
- GetEntryXBeforeHead(); Holt das Element vor dem x-ten Element vom Kopf, ohne das Element zu entfernen. Wenn also 0 als Parameter genutzt wird, wird das erste Element abgeholt. Sonst muss der Parameter grösser als 0 sein. Liefert FALSE, wenn das Element nicht existiert, sonst TRUE.
- Count(); Liefert die Anzahl der abgelegten Elemente.
- Size(); Liefert die Größe des Speichers, also die maximale Anzahl von Elementen, die in den Speicher passen.
- FreeCount(); Liefert die Anzahl der Elemente, die zum Speicher hinzugefügt werden.
- GetNextEntry(); Iteriert über die Elemente und startet dabei am Kopf.

CFiFoList

Die Klasse CFiFoList basiert auf CFiFoListBase und nutzt als Storage-Type CFiFoListStorageStatic zusammen mit der Synchronisierung-Type CSyncDefault.
Diese Implementierung fügt keine Methoden hinzu und benötigt Element-Typen sowie Größe des Speichers als Template Parameter.

CFiFoListDyn

Die Klasse CFiFoListDyn nutzt den dynamischen StorageType CFifoListStorageDynamic zusammen mit der Synchronisierung-Type CSyncDefault.
Diese Implementierung fügt keine Methoden hinzu und benötigt Element-Typen sowie Größe des Speichers als Template Parameter.

Handover3Buffer

Die Klasse CHandover3Buffer wird genutzt um Elemente von einem Task zu einem anderen weiterzureichen. Der Speicher umfasst 3 Elemente von einem gegebenen Typen. Der schreibende Task kann das Element immer schreiben. Der lesende Task kann das Element immer lesen, wenn ein Element geschrieben wurde. Es liest das Element, welches komplett geschrieben wurde. Elemente werden nicht entfernt nachdem sie gelesen werden. Zugriff via Indices für Lesen und Schreiben auf dem Speicher werden über die Funktion InterlockedExchangeR0 synchronisiert, sodass dieses vom Windows- und RT-Kontext zugreifbar sind.

Methoden:

- CHandover3Buffer(); Der Konstruktor.
- Write(); Schreibt ein Element in den Speicher. Liefert FALSE, wenn der Speicher einen ungültigen Zustand hat. Sonst TRUE.
- Read(); Liest ein Element aus dem Speicher. Liefert FALSE, wenn der Speicher leer ist oder Speicher in einem ungültigen Zustand ist. Sonst TRUE.

CHandoverFifo

Die Klasse CHandoverFifo wird ebenso genutzt um Elemente zwischen Tasks auszutauschen. Die Größe des Speichers ist immer eine zweier Potenz und kann durch den Template Parameter DEPTH_EXP gesetzt werden. Dieser ist per Default 3, sodass sich eine Größe von 8 Elementen ergibt.

Methoden:

- Insert()/Add(); Schreibt ein Element über einen Pointer oder eine Reference in den Speicher. Liefert FALSE, wenn kein Speicher verfügbar ist, sonst TRUE.
- Remove(); Entfernt und liefert ein Element über einen Pointer oder eine Referenz aus dem Speicher. Liefert FALSE, wenn kein Element verfügbar ist, sonst TRUE.

Beispiel:
TcHandoverSample.tszip

12.8 Schnittstellen

Für die Interaktion der vom Benutzer entwickelten Module mit dem TwinCAT 3 System stehen etliche Schnittstellen zur Verfügung. Auf diesen Seiten werden diese (auf API Ebene) ausführlich beschrieben.

Name	Beschreibung
ITcUnknown [▶ 197]	ITcUnknown definiert die Referenzzählung, sowie das Abfragen einer Referenz auf eine spezifischere Schnittstelle.
ITComObject [▶ 186]	Die ITComObject Schnittstelle wird von jedem TwinCAT Modul implementiert.
ITcCyclic [▶ 169]	Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus aufgerufen werden.
ITcCyclicCaller [▶ 170]	Schnittstelle zum Anmelden oder Abmelden der ITcCyclic Schnittstelle eines Moduls bei einer TwinCAT Task.
ITcFileAccess [▶ 172]	Schnittstelle zum Zugriff auf das Dateisystem
ITcFileAccessAsync [▶ 180]	Asynchroner Zugriff auf Dateioperationen.
ITcPostCyclic [▶ 191]	Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus im Anschluss an die Ausgang-Aktualisierung aufgerufen werden.
ITcPostCyclicCaller [▶ 167]	Schnittstelle zum Anmelden oder Abmelden der ITcPostCyclic Schnittstelle eines Moduls bei einer TwinCAT Task.
ITcIoCyclic [▶ 181]	Diese Schnittstelle wird von TwinCAT Modulen implementiert, die bei Eingang-Aktualisierung und bei Ausgang-Aktualisierung innerhalb eines Taskzyklus aufgerufen werden.
ITcIoCyclicCaller [▶ 182]	Schnittstelle zum Anmelden oder Abmelden der ITcIoCyclic Schnittstelle eines Moduls bei einer TwinCAT Task.
ITcRTimeTask [▶ 192]	Abfrage von erweiterten TwinCAT Taskinformationen.
ITcTask [▶ 193]	Abfrage von Zeitstempel und taskspezifischen Informationen einer TwinCAT Task.
ITcTaskNotification [▶ 196]	Führt einen Callback aus, wenn die Zykluszeit beim vorherigen Zyklus überschritten wurde.

Das TwinCAT SDK

Das TwinCAT SDK beinhaltet eine Reihe von Funktionen, die in *C:\TwinCAT\3.1\sdk\Include* gefunden werden können.

- Das TcCOM Framework wird hier bereitgestellt (insb. TcInterfaces.h und TcServices.h).
- Tasks und Datenbereichs-Zugriffe werden über die TcIoInterfaces.h bereitgestellt.
- SDK-Funktionen sind die mathematischen Funktionen [▶ 218].
- Untermenge der STL [▶ 222].
- TwinCAT Runtime RtIR0.h [▶ 199]

- Methoden zur ADS Kommunikation [▶ 200]
- Klassen / Funktionen, die mit „Os“ beginnen, dürfen nicht im Echtzeitkontext genutzt werden.

12.8.1 Schnittstelle ITcPostCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcPostCyclic Schnittstelle eines Moduls bei einer TwinCAT Task.

Syntax

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

Benötigtes include: `TcIoInterfaces.h`

Methoden

Name	Beschreibung
AddPostModule [▶ 167]	Modul anmelden, das die ITcPostCyclic Schnittstelle implementiert.
RemovePostModule [▶ 168]	Zuvor angemeldete ITcPostCyclic Schnittstelle eines Moduls abmelden.

Die ITcPostCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle um seine ITcPostCyclic Schnittstelle bei einer Task anzumelden, normalerweise als letzten Initialisierungsschritt beim SafeOP zum OP Übergang. Nach der Anmeldung wird die Methode PostCycleUpdate() der Modulinstanz aufgerufen. Die Schnittstelle wird ebenfalls zum Abmelden des Moduls verwendet, damit es nicht mehr von der Task aufgerufen wird.

12.8.1.1 Methode ITcPostCyclicCaller:AddPostModule

Meldet die ITcPostCyclic Schnittstelle eines Moduls bei einem zyklischen Aufrufer, z. B. einer TwinCAT Task an.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Wenn der zyklische Aufrufer, d.h. die TwinCAT Task, nicht im OP-Zustand ist, wird der Fehler ADSERR_DEVICE_INVALIDSTATE zurückgegeben.

Parameter

Name	Typ	Beschreibung
pEntry	STcPostCyclicEntry	[in] Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen Aufrufers eingefügt wird, siehe auch Beschreibung.
ipMod	ITcPostCyclic	[in] Schnittstellenzeiger, der vom zyklischen Aufrufer verwendet wird.
context	ULONG_PTR	[optional] Ein Kontextwert, der bei jedem Aufruf an die ITcPostCyclic::PostCyclicUpdate() Methode übergeben wird.
sortOrder	ULONG	[optional] Die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

Beschreibung

Eine TwinCAT-Modulklasse verwendet einen Smart Pointer, um auf den zyklischen Aufrufer vom Typ ITcPostCyclicCallerPtr zu verweisen. Die Objekt-ID der Task ist in diesem Smart Pointer gespeichert und eine Referenz zur Task kann über den TwinCAT Objektserver erhalten werden. Darüber hinaus enthält die Klasse des Smart Pointers bereits einen Listeneintrag. Demzufolge kann der Smart Pointer als erster Parameter für die AddPostModule Methode verwendet werden.

Das folgende Codebeispiel veranschaulicht die Anmeldung der ITcPostCyclicCaller Schnittstelle.

```
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 Methode ITcPostCyclicCaller:RemovePostModule

Eine Modulinstanz vom Aufruf durch einen zyklischen Aufrufer abmelden.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E_FAIL zurück.

Parameter

Name	Typ	Beschreibung
pEntry	STcPostCyclicEntry	Verweist auf den Listeneintrag, der aus der internen Liste des zyklischen Aufrufers zu entfernen ist.

Vergleichbar mit der Methode AddPostModule() wird der Smart Pointer für den zyklischen Aufrufer als Listeneintrag verwendet, wenn die Modulinstanz aus dem zyklischen Aufrufer entfernt werden soll.

Deklaration und Verwendung des Smart Pointers:

ITcPostCyclicCallerInfoPtr m_spPostCyclicCaller;

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

}

m_spPostCyclicCaller = NULL;
```

12.8.2 Rückgabewerte

ITc-Schnittstellen Methoden liefern in der Regel einen HRESULT zurück.

Die folgenden Rückgabewerte können bei ITc-Schnittstellen zurückgegeben werden:

Name	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

Zusätzlich besteht die Möglichkeit ADS Return Codes [▶ 343] als HRESULT zurückzubekommen. Diese stehen im SDK auch als Makros zur Verfügung und heißen dort beispielsweise ADS_E_BUSY für den ADS Error Code ADSERR_DEVICE_BUSY.

12.8.3 Schnittstelle ITcCyclic

Die Schnittstelle ITcCyclic wird von TwinCAT Modulen implementiert, die ein Mal pro Task-Zyklus aufgerufen werden.

Syntax

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

Benötigtes include: `TcIoInterfaces.h`

Methoden

Name	Beschreibung
CycleUpdate [▶ 169]	Wird ein Mal pro Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen Aufrufer angemeldet ist.

Die Schnittstelle ITcCyclic wird von TwinCAT Modulen implementiert. Diese Schnittstelle wird der Methode ITcCyclicCaller::AddModule() übergeben, wenn sich ein Modul bei einer Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen.

12.8.3.1 Methode ITcCyclic: CycleUpdate

Die Methode CycleUpdate wird normalerweise von einer TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Parameter

Name	Typ	Beschreibung
ipTask	ITcTask	Verweist auf den aktuellen Task-Kontext.
ipCaller	ITcUnknown	Verweist auf die aufrufende Instanz.
Context	ULONG_PTR	Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddModule() übergeben wurde.

Beschreibung

In einem Taskzyklus wird die Methode CycleUpdate() aufgerufen, nachdem InputUpdate() für alle angemeldeten Modulinstanzen aufgerufen wurde. Demzufolge muss diese Methode verwendet werden, um eine zyklische Abarbeitung zu implementieren.

12.8.4 Schnittstelle ITcCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcCyclic Schnittstelle eines Moduls bei einer TwinCAT Task.

Syntax

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

Benötigtes include: `TcIoInterfaces.h`

Methoden

Name	Beschreibung
AddModule [▶ 170]	Modul anmelden, das die ITcCyclic Schnittstelle implementiert.
RemoveModule [▶ 171]	Die zuvor angemeldete ITcCyclic Schnittstelle eines Moduls abmelden.

Die ITcCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle, um seine ITcCyclic Schnittstelle bei einer Task anzumelden, normalerweise als letzten Initialisierungsschritt beim Übergang SafeOP zu OP. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen. Die Schnittstelle wird ebenfalls zum Abmelden des Moduls verwendet, damit es nicht mehr von der Task aufgerufen wird.

12.8.4.1 Methode ITcCyclicCaller:AddModule

Meldet die ITcCyclic Schnittstelle eines Moduls bei einem zyklischen Aufrufer, z. B. einer TwinCAT Task an.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Wenn der zyklische Aufrufer, d.h. die TwinCAT Task, nicht im OP-Zustand ist, wird der Fehler ADSERR_DEVICE_INVALIDSTATE zurückgegeben.

Parameter

Name	Typ	Beschreibung
pEntry	STcCyclicEntry	[in] Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen Aufrufers eingefügt wird, siehe auch Beschreibung [▶ 171].
ipMod	ITcCyclic	[in] Schnittstellenzeiger, der vom zyklischen Aufrufer verwendet wird.
context	ULONG_PTR	[optional] ein Kontextwert, der bei jedem Aufruf an die ITcCyclic:: CycleUpdate () Methode übergeben wird.
sortOrder	ULONG	[optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

Beschreibung

Eine TwinCAT-Modulklasse verweist normalerweise mit einem Smart Pointer auf den zyklischen Aufrufer Typ ITcCyclicCallerPtr. Die Objekt-ID der Task ist in diesem Smart Pointer gespeichert und eine Referenz zum Task kann über den TwinCAT Objektserver erhalten werden. Darüber hinaus enthält die Klasse des Smart Pointers bereits einen Listeneintrag. Demzufolge kann der Smart Pointer als erster Parameter für die AddModule Methode verwendet werden.

Der folgende Beispielcode zeigt die Anmeldung der ITcCyclicCaller Schnittstelle.

```
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 Methode ITcCyclicCaller:RemoveModule

Eine Modulinstanz vom Aufruf durch einen zyklischen Aufrufer abmelden.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
pEntry	STcCyclicEntry	verweist auf den Listeneintrag, der aus der internen Liste des zyklischen Aufrufers zu entfernen ist.

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E_FAIL zurück.

Vergleichbar mit der Methode AddModule() wird der Smart Pointer für den zyklischen Aufrufer als Listeneintrag verwendet, wenn die Modulinstanz aus dem zyklischen Aufrufer entfernt werden soll.

Deklaration und Verwendung des Smart Pointers:

ITcCyclicCallerInfoPtr m_spCyclicCaller;

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

12.8.5 Schnittstelle ITcFileAccess

Schnittstelle für Zugriff auf Dateisystem von TwinCAT C++ Modulen aus

Syntax

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

Benötigtes include: `TcFileAccessInterfaces.h`

Methoden

Name	Beschreibung
FileOpen [▶ 173]	Öffnet eine Datei.
FileClose [▶ 173]	Schließt eine Datei.
FileRead [▶ 174]	Liest aus einer Datei.
FileWrite [▶ 174]	Schreibt in eine Datei.
FileSeek [▶ 175]	Legt die Position innerhalb einer Datei für weitere Aktionen fest.
FileTell [▶ 175]	Fragt die derzeit gesetzte Position innerhalb einer Datei ab.
FileRename [▶ 175]	Nennt eine Datei um.
FileDelete [▶ 176]	Löscht eine Datei.
FileGetStatus [▶ 176]	Erhält den Zustand einer Datei.
FileFindFirst [▶ 177]	Sucht nach einer Datei, erste Iteration.
FileFindNext [▶ 178]	Sucht nach einer Datei, nächste Iteration.
FileFindClose [▶ 178]	Schließt eine Dateisuche.
MkDir [▶ 179]	Erstellt ein Verzeichnis.
RmDir [▶ 179]	Löscht ein Verzeichnis.

Die ITcFileAccess Schnittstelle wird für den Zugriff auf Dateien in Dateisystemen verwendet.
Weil die zur Verfügung gestellten Methoden Blockaden verursachen, sollte diese Schnittstelle nicht in CycleUpdate() / Echtzeitkontext verwendet werden. Die abgeleitete Schnittstelle ITcFileAccessAsync [▶ 180] fügt eine Check() Methode hinzu, die stattdessen verwendet werden kann.

Siehe Beispiel20a: FileIO-Cyclic Read / Write [▶ 309].

Die Schnittstelle wird über die Modulklasse CID_TcFileAccess implementiert.

12.8.5.1 Methode ITcFileAccess:FileOpen

Öffnet eine Datei.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT, wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
szFileName	PCCH	[in] der Name der zu öffnenden Datei.
AccessMode	TcFileAccessMode	[in] Art des Zugriffs auf die Datei, siehe TcFileAccessServices.h.
phFile	TcFileHandle	[out] zurückgegebener Datei-Handle.

AccessModes können folgendermaßen verwendet werden:

```
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 Methode ITcFileAccess:FileClose

Schließt eine Datei.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
phFile	TcFileHandle	[out] zurückgegebener Datei-Handle der zu schließenden Datei

12.8.5.3 Methode ITcFileAccess:FileRead

Liest Daten aus einer Datei.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT, wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
hFile	TcFileHandle	[in] verweist auf die zuvor geöffnete Datei.
pData	PVOID	[out] Speicherort der zu lesenden Daten.
cbData	PVOID	[in] maximale Größe der zu lesenden Daten (Größe des Speichers hinter pData).
pcbRead	PUINT	[out] Größe der gelesenen Daten.

12.8.5.4 Methode ITcFileAccess:FileWrite

Daten in eine Datei schreiben.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
hFile	TcFileHandle	[in] verweist auf die zuvor geöffnete Datei.
pData	PVOID	[in] Speicherort der zu schreibenden Daten.
cbData	PVOID	[in] Größe der zu schreibenden Daten (Größe des Speichers hinter pData).
pcbRead	PUINT	[out] Größe der geschriebenen Daten.

12.8.5.5 Methode ITcFileAccess:FileSeek

Legt die Position innerhalb einer Datei für weitere Aktionen fest.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
hFile	TcFileHandle	[in] verweist auf die zuvor geöffnete Datei.
uiPos	UINT	[in] Position auf die zu setzen ist.

12.8.5.6 Methode ITcFileAccess:FileTell

Fragt die derzeit gesetzte Position innerhalb einer Datei ab.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
hFile	TcFileHandle	[in] verweist auf die zuvor geöffnete Datei.
puiPos	PUINT	[out] Speicherort der zurückzugebenden Position.

12.8.5.7 Methode ITcFileAccess:FileRename

Benennt eine Datei um.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

szOldName	PCCH	[in] der umzubenennende Dateiname.
szNewName	PCCH	[in] der neue Dateiname.

12.8.5.8 Methode ITcFileAccess:FileDelete

Löscht eine Datei.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
szFileName	PCCH	[in] Name der zu löschenden Datei.

12.8.5.9 Methode ITcFileAccess:FileGetStatus

Fragt den Zustand einer Datei ab.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
szFileName	PCCH	[in] der Name der fraglichen Datei.
pFileStatus	PTcFileStatus	[out] der Zustand der Datei, vgl. TcFileAccessServices.h .

Beschreibung

Diese Methode fragt Zustandsinformationen bezüglich eines gegebenen Dateinamens ab.

Dazu gehören die folgenden Informationen:

```
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 Methode ITcFileAccess:FileFindFirst

Möglichkeit, die Dateien eines Verzeichnisses zu durchlaufen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
szFileName	PCCH	[in] Verzeichnis oder Pfad, und Name der gesuchten Datei. Der Dateiname kann Platzhalterzeichen wie Sternchen (*) oder Fragezeichen (?) enthalten.
pFileFindData	PTcFileFindData	[out] die Beschreibung der ersten Datei, vgl. TcFileAccessServices.h
phFileFind	PTcFileFindHandle	[out] Handle um weiter mit FileFindNext zu suchen.

Beschreibung

Diese Methode beginnt mit der Suche nach Dateien in einem vorgegebenen Verzeichnis. Die Methode gewährt Zugriff auf PTcFileFindData der ersten gefundenen Datei, mit folgenden Informationen:

```
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 Methode ITcFileAccess:FileFindNext

Die Dateien eines Verzeichnisses weiter durchlaufen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
hFileFind	PTcFileFindHandle	[in] Handle, um weiter mit FileFindNext zu suchen.
pFileFindData	PTcFileFindData	[out] die Beschreibung der nächsten Datei. Vergleiche mit TcFileAccessServices.h

Beschreibung

Diese Methode sucht nach der nächsten Datei in einem Verzeichnis. Die Methode gewährt Zugriff auf PTcFileFindData der gefundenen Datei, mit folgenden Informationen:

```
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 Methode ITcFileAccess:FileFindClose

Die Suche nach Dateien in einem Verzeichnis schließen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
hFileFind	PTcFileFindHandle	[in] Handle um die Suche zu schließen.

12.8.5.13 Methode ITcFileAccess:MkDir

Ein Verzeichnis in einem Dateisystem erstellen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
szDir	PCCH	[in] zu erstellendes Verzeichnis.

12.8.5.14 Methode ITcFileAccess:RmDir

Ein Verzeichnis aus dem Dateisystem entfernen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Ein besonders interessanter Fehlercode ist ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Parameter

Name	Typ	Beschreibung
szDir	PCCH	[in] zu löschendes Verzeichnis.

12.8.6 Schnittstelle ITcFileAccessAsync

Asynchroner Zugriff auf Dateioperationen. Diese Schnittstelle erweitert ITcFileAccess [▶ 172].

Syntax

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

Benötigtes include: `TcFileAccessInterfaces.h`

Eigenschaften

Name	Beschreibung
PID_TcFileAccessAsyncSegmentSize	Größe der an System-Service übergebenen Segmente.
PID_TcFileAccessAsyncTimeoutMs	Setzt den Timeout in [ms].
PID_TcFileAccessAsyncNetId(Str)	NetId des zu kontaktierenden System-Service.

Methoden

Name	Beschreibung
Check [▶ 180]	Zustand der zuvor aufgerufenen Dateioperation abfragen.

Schnittstelle kann von Modulinstanz mit Klassen-ID CID_TcFileAccessAsync erhalten werden.
Bei Verwendung der asynchronen Schnittstelle, geben die von der synchronen Variante geerbten Schnittstellenmethoden ADS_E_PENDING zurück, wenn eine Abfrage erfolgreich unterbreitet, aber noch nicht abgeschlossen wurde. Wenn der Aufruf eingeht, während die vorherige Anfrage immer noch abgearbeitet wurde, wird der Fehlercode ADS_E_BUSY zurückgegeben.

Beschreibung der Modulparameter:

- PID_TcFileAccessAsyncAdsProvider: Objekt-ID einer Task, die die ADS-Schnittstelle bereitstellt.
- PID_TcFileAccessAsyncNetId / PID_TcFileAccessAsyncNetIdStr: AmsNetId des System-Service, der für den Dateizugriff verwendet wird. Die „Str“ Variante nimmt die AmsNetId als Zeichenkette. Bitte eins verwenden.
- PID_TcFileAccessAsyncTimeoutMs: Zeitüberschreitung für einen Dateizugriff.
- PID_TcFileAccessAsyncSegmentSize: Der Lese- und Schreibzugriff auf Datei wird mit dieser Segmentgröße fragmentiert.

Siehe Beispiel20a: FileIO-Cyclic Read / Write [▶ 309].

12.8.6.1 Method ITcFileAccessAsync::Check()

Zustand der zuvor aufgerufenen Dateioperation abfragen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Besonders interessante Fehlercodes sind

- ADSERR_DEVICE_TIMEOUT wenn die Zeitüberschreitung (5 Sekunden) abgelaufen ist.
- ADSERR_DEVICE_PENDING wenn die Dateioperation nicht abgeschlossen ist.

Parameter

keiner

12.8.7 Schnittstelle ITcIoCyclic

Diese Schnittstelle wird von TwinCAT Modulen implementiert, die bei Eingang-Aktualisierung und bei Ausgang-Aktualisierung innerhalb eines Taskzyklus aufgerufen werden.

Syntax

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

Benötigtes include: `TcIoInterfaces.h`

Methoden

Name	Beschreibung
InputUpdate [▶ 181]	Wird zu Beginn eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.
OutputUpdate [▶ 182]	Wird am Ende eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.

ITcIoCyclic kann dazu verwendet werden, ein TwinCAT Modul zu implementieren, das als Feldbustreiber oder I/O-Filtermodul agiert.

Diese Schnittstelle wird der Methode ITcIoCyclicCaller::AddIoDriver übergeben, wenn sich ein Modul bei einer Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung werden die Methoden InputUpdate() und OutputUpdate() der Modulinstanz je einmal pro Taskzyklus aufgerufen.

12.8.7.1 Methode ITcIoCyclic:InputUpdate

Wird zu Beginn eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
ipTask	ITcTask	Verweist auf den aktuellen Task-Kontext.
ipCaller	ITcUnknown	Verweist auf die aufrufende Instanz.
dwStateIn	DWORD	Zukünftigen Erweiterungen vorbehalten, derzeit ist dieser Wert immer 0.
context	ULONG_PTR	Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddIoDriver() übergeben wurde.

Beschreibung

In einem Taskzyklus wird die Methode InputUpdate() für alle angemeldeten Modulinstanzen zuerst aufgerufen. Deswegen muss diese Methode für die Aktualisierung der Datenbereiche vom Typ Input-Source des Moduls verwendet werden.

12.8.7.2 Methode ITcIoCyclic:OutputUpdate

Wird am Ende eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
ipTask	ITcTask	Verweist auf den aktuellen Task-Kontext.
ipCaller	ITcUnknown	Verweist auf die aufrufende Instanz.
pdwStateOut	DWORD	[out] zukünftigen Erweiterungen vorbehalten, derzeit wird der Rückgabewert ignoriert.
context	ULONG_PTR	Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddIoDriver() übergeben wurde.

Beschreibung

In einem Taskzyklus wird für alle angemeldeten Modulinstanzen die Methode OutputUpdate() aufgerufen. Deswegen muss diese Methode für die Aktualisierung der Datenbereiche vom Typ Output-Destination des Moduls verwendet werden.

12.8.8 Schnittstelle ITcIoCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcIoCyclic Schnittstelle eines Moduls bei einer TwinCAT Task.

Syntax

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

Benötigtes include: `TcIoInterfaces.h`

Methoden

Name	Beschreibung
AddIoDriver [▶ 183]	Modul anmelden, das die ITcIoCyclic Schnittstelle implementiert.
RemoveIoDriver [▶ 184]	Zuvor angemeldete ITcIoCyclic Schnittstelle eines Moduls abmelden.

Die ITcIoCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle, um seine ITcIoCyclic Schnittstelle bei einer Task anzumelden, normalerweise als letzten Initialisierungsschritt beim SafeOP zum OP Übergang. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen. Die Schnittstelle wird ebenfalls zum Abmelden des Moduls verwendet, damit es nicht mehr von der Task aufgerufen wird.

12.8.8.1 Methode ITcIoCyclicCaller:AddIoDriver

Meldet die ITcIoCyclic Schnittstelle eines Moduls bei einem zyklischen I/O-Aufrufer, z. B. einer TwinCAT Task an.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
pEntry	STcIoCyclicEntry	Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen I/O-Aufrufers eingefügt wird, siehe auch Beschreibung [▶ 171].
ipDrv	ITcIoCyclic	[in] Schnittstellenzeiger, der vom zyklischen I/O-Aufrufer verwendet wird.
context	ULONG_PTR	[optional] ein Kontextwert, der bei jedem Aufruf an die ITcIoCyclic::InputUpdate() und ITcIoCyclic::OutputUpdate Methode übergeben wird.
sortOrder	ULONG	[optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

Beschreibung

Eine TwinCAT-Modulklasse verwendet normalerweise einen Smart Pointer, um auf den zyklischen I/O-Aufrufer vom Typ ITcIoCyclicCallerPtr zu verweisen. Die Objekt-ID des zyklischen I/O-Aufrufers ist in diesem Smart Pointer gespeichert und eine Referenz kann über den TwinCAT Objektserver erhalten werden. Darüber hinaus enthält die Klasse des Smart Pointers bereits einen Listeneintrag. Demzufolge kann der Smart Pointer als erster Parameter für die AddIoDriver Methode verwendet werden.

Das folgende Codebeispiel veranschaulicht die Anmeldung der ITcIoCyclicCaller Schnittstelle.

```
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 Methode ITcIoCyclicCaller:RemoveIoDriver

Eine Modulinstanz vom Aufruf durch einen zyklischen I/O-Aufrufer abmelden.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E_FAIL zurück.

Parameter

Name	Typ	Beschreibung
pEntry	STcIoCyclicEntry	Verweist auf den Listeneintrag, der aus der internen Liste des zyklischen I/O-Aufrufers zu entfernen ist.

Vergleichbar mit der Methode AddIoDriver() wird der Smart Pointer für den zyklischen I/O-Aufrufer als Listeneintrag verwendet, wenn die Modulinstanz aus dem zyklischen I/O-Aufrufer entfernt werden soll.

Deklaration und Verwendung des Smart Pointers:

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

12.8.9 Schnittstelle ITComOnlineChange

Die ITComOnlineChange Schnittstelle wird genutzt, um Online-Changes von Modulen durchzuführen.

Syntax

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

Methoden

Name	Beschreibung
PrepareOnlineChange [▶ 184]	Diese Methode wird von TwinCAT zur Vorbereitung des Online-Change aufgerufen.
PerformOnlineChange [▶ 185]	Diese Methode wird von TwinCAT zum Durchführen des Online-Change aufgerufen.

Die Implementierung dieses Interfaces ist notwendig, damit ein Modul Online-Change fähig ist. Weiterhin muss ein solches Modul in einem versionierten C++ Projekt angelegt werden.

- Hier [▶ 155] ist eine allgemeine Beschreibung des Vorgehens.
- Bei existierenden Modulen kann diesem Vorgehen gefolgt werden: Online-Change [▶ 155].

12.8.9.1 Methode ITComOnlineChange:PrepareOnlineChange

Diese Methode wird von TwinCAT zur Vorbereitung des Online-Change aufgerufen.

Sie läuft asynchron im Hintergrund, welches bei Zugriffen auf das existierende Objekt beachtet werden muss.
Die Vorbereitung sollte alle Operationen beinhalten, die schon getätigt werden können.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169].
Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
ipOldObj	ITComObject*	Referenz auf das existierende Objekt, welches ausgetauscht wird.
pOldInfo	TmcInstData*	Referenz auf Informationen des existierenden Objektes.

Über ipOldObj werden die Daten des bisher existierenden Objektes zur Übergabe bereitgestellt, sodass sie übernommen werden können.

Beispielsweise:

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

12.8.9.2 Methode ITComOnlineChange:PerformOnlineChange

Diese Methode wird von TwinCAT zum Durchführen des Online-Change aufgerufen.

Sie wird blockierend aufgerufen. Sie sollte deswegen nur kurze Zeit benötigen.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169].
Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
ipOldObj	ITComObject*	Referenz auf das existierende Objekt, welches ausgetauscht wird.
pOldInfo	TmcInstData*	Referenz auf Informationen des existierenden Objektes.

Über ipOldObj werden die Daten des bisher existierenden Objektes zur Übergabe bereitgestellt, sodass sie übernommen werden können.

Beispielsweise:

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

12.8.10 Schnittstelle ITComObject

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt grundlegende Funktionalitäten zur Verfügung.

Syntax

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

Methoden

Name	Beschreibung
TcGetObjectId(OTCID& objId) [▶ 186]	Speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.
TcSetObjectId [▶ 186]	Setzt die Objekt-ID des Objekts auf die gegebene OTCID.
TcGetObjectName [▶ 187]	Speichert den Objektnamen im Puffer mit der gegebenen Länge.
TcSetObjectName [▶ 187]	Setzt den Objektnamen des Objekts auf gegebenen CHAR*.
TcSetObjState [▶ 187]	Initialisiert einen Übergang zu einem vorgegebenen Zustand.
TcGetObjState [▶ 188]	Fragt den aktuellen Zustand des Objekts ab.
TcGetObjPara [▶ 188]	Fragt einen mit seiner PTCID identifizierten Objektparameter ab.
TcSetObjPara [▶ 189]	Setzt einen mit seiner PTCID identifizierten Objektparameter.
TcGetParentObjId [▶ 189]	Speichert die Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.
TcSetParentObjId [▶ 189]	Setzt die Parent-Objekt-ID auf die gegebene OTCID.

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt Funktionalitäten zur Verfügung bezüglich der Zustandsmaschine und Informationen vom/an das TwinCAT-System.

12.8.10.1 Methode ITcComObject:TcGetObjectId

Die Methode speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
objId	OTCID&	Referenz auf OTCID-Wert.

12.8.10.2 Methode ITcComObject:TcSetObjectId

Die Methode TcSetObjectId setzt die Objekt-ID des Objekts auf die gegebene OTCID.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Parameter

Name	Typ	Beschreibung
objId	OTCID	Die zu setzende OTCID - zeigt den Erfolg der ID-Änderung an.

12.8.10.3 Methode ITcComObject:TcGetObjectName

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
objName	CHAR*	Der zu setzende Name.
nameLen	ULONG	Die maximale, zu schreibende Länge.

12.8.10.4 Methode ITcComObject:TcSetObjectName

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR*.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
objName	CHAR*	Der zu setzende Name des Objekts.

12.8.10.5 Methode ITcComObject:TcSetObjState

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
state	TCOM_STATE	Stellt den neuen Zustand dar.
ipSrv	ITComObjectServer*	ObjServer, der das Objekt handhabt.
pInitData	PTComInitDataHdr	Zeigt auf eine Liste von Parametern (optional), siehe Makro IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA als Beispiel, wie die Liste iteriert werden kann.

12.8.10.6 Methode ITcComObject:TcGetObjState

Die Methode TcGetObjState fragt den aktuellen Zustand des Objekts ab.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
pState	TCOM_STATE*	Zeiger auf den Zustand.

12.8.10.7 Methode ITcComObject:TcGetObjPara

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
pid	PTCID	Parameter-ID des Objektparameters.
nData	ULONG&	Max. Länge der Daten.
pData	PVOID&	Zeiger auf die Daten.
pgp	PTCGP	Für zukünftige Erweiterung vorbehalten, NULL weitergeben.

12.8.10.8 Methode ITcComObject:TcSetObjPara

Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
pid	PTCID	Parameter-ID des Objektparameters.
nData	ULONG	Max. Länge der Daten.
pData	PVOID	Zeiger auf die Daten.
pgp	PTCGP	Für zukünftige Erweiterung vorbehalten, NULL weitergeben.

12.8.10.9 Methode ITcComObject:TcGetParentObjId

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
objId	OTCID&	Referenz auf OTCID-Wert.

12.8.10.10 Methode ITcComObject:TcSetParentObjId

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Parameter

Name	Typ	Beschreibung
objId	OTCID	Referenz auf OTCID-Wert.

12.8.11 Schnittstelle ITComObject (C++ Convenience)

Die ITComObject Schnittstelle wird von jedem TwinCAT Modul implementiert. Sie stellt grundlegende Funktionalitäten zur Verfügung.

In TwinCAT C++ stehen zusätzliche Funktionen bereit, die nicht direkt durch das Interface definiert sind.

Syntax

Benötigtes include: `TcInterfaces.h`

Methoden

Name	Beschreibung
OTCID TcGetObjectId [▶ 190]	Fragt die Objekt ID vom Objekt ab.
TcTryToReleaseOpState [▶ 190]	Gibt Ressourcen frei, muss implementiert sein.

Es existieren weitere Methoden, die hier nicht einzeln aufgeführt werden.

Diese Funktionalität wird durch die Modul-Assistenten bei den Modulen standardmäßig bereitgestellt.

12.8.11.1 Methode TcGetObjectId

Die Methode fragt die Objekt-ID des Objektes ab.

Syntax

```
OTCID TcGetObjectId(void)
```

Rückgabewert

Name	Beschreibung
OTCID	Gibt die OTCID des Objekts zurück.

12.8.11.2 Methode TcTryToReleaseOpState

Die Methode TcTryToReleaseOpState gibt Ressourcen frei, z. B. Datenzeiger, um das Verlassen des OP-Zustands vorzubereiten.

Syntax

```
BOOL TcTryToReleaseOpState(void)
```

Rückgabewert

Es wird TRUE oder FALSE zurückgegeben.

Beschreibung

Die Methode TcTryToReleaseOpState muss implementiert werden, um mögliche wechselseitige Abhängigkeiten von Modulinstanzen aufzuheben.

Sie wird dabei nur aufgerufen, wenn ein anderes Modul eine Referenz auf dieses Modul hält.

12.8.12 Schnittstelle ITcPostCyclic

Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus im Anschluss an die Ausgang-Aktualisierung aufgerufen werden (vergleichbar mit Attribut TcCallAfterOutputUpdate der SPS).

Syntax

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

Benötigtes include: `TcIoInterfaces.h`

Methoden

Name	Beschreibung
PostCycleUpdate [▶ 191]	Wird ein Mal pro Taskzyklus nach der Ausgangs-Aktualisierung aufgerufen, wenn die Schnittstelle bei einem zyklischen Aufrufer angemeldet wurde.

Die ITcPostCyclic Schnittstelle wird von TwinCAT Modulen implementiert. Diese Schnittstelle wird der Methode ITcCyclicCaller::AddPostModule() übergeben, wenn ein Modul sich selbst bei einer Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung wird die Methode PostCycleUpdate() der Modulinstanz aufgerufen.

12.8.12.1 Methode ITcPostCyclic:PostCyclicUpdate

Die normalerweise von einer TwinCAT Task nach der Ausgang-Aktualisierung aufgerufene Methode PostCyclicUpdate, nachdem die Schnittstelle angemeldet wurde.

Syntax

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

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Parameter

Name	Typ	Beschreibung
ipTask	ITcTask	Verweist auf den aktuellen Task-Kontext.
ipCaller	ITcUnknown	Verweist auf die aufrufende Instanz.
Context	ULONG_PTR	Kontext beinhaltet den Wert, der an die Methode ITcPostCyclicCaller::AddPostModule() übergeben wurde.

Beschreibung

Innerhalb eines Taskzyklusses wird die Methode PostCycleUpdate() aufgerufen, nachdem OutputUpdate() für alle angemeldeten Modulinstanzen aufgerufen wurde. Demzufolge muss diese Methode verwendet werden, um derlei zyklische Abarbeitung zu implementieren.

12.8.13 Schnittstelle ITcRTimeTask

Abfrage von erweiterten TwinCAT Taskinformationen.

Syntax

TCOM_DECL_INTERFACE("02000003-0000-0000-e000-000000000064", ITcRTimeTask)
struct __declspec(novtable) ITcRTimeTask : public ITcTask

Benötigtes include: `TcRtInterfaces.h`

Methoden

Name	Beschreibung
GetCpuAccount [▶ 192]	Abfrage des CPU-Accounts einer TwinCAT Task.

Mit dieser Schnittstelle können TwinCAT Taskinformationen abgefragt und verwendet werden.

Siehe Beispiel30: Zeitmessung [▶ 319]

12.8.13.1 Methode ITcRTimeTask::GetCpuAccount()

Abfrage des CPU-Accounts einer TwinCAT Task.

Syntax

```
virtual HRESULT TCOMAPI GetCpuAccount(PULONG pAccount)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER wenn Parameter pAccount gleich NULL.

Parameter

Name	Typ	Beschreibung
pAccount	PULONG	[out] TwinCAT Task CPU-Account ist in diesem Parameter gespeichert.

Mit Hilfe der GetCpuAccount() Methode kann die aktuelle, für die Task aufgewendete Rechenzeit abgefragt werden.

Code-Ausschnitt, der die Verwendung von GetCpuAccount() zeigt, z. B. innerhalb einer ITcCyclic::CycleUpdate() Methode:

```
// 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 Schnittstelle ITcTask

Abfrage von Zeitstempel und taskspezifischen Informationen einer TwinCAT Task.

Syntax

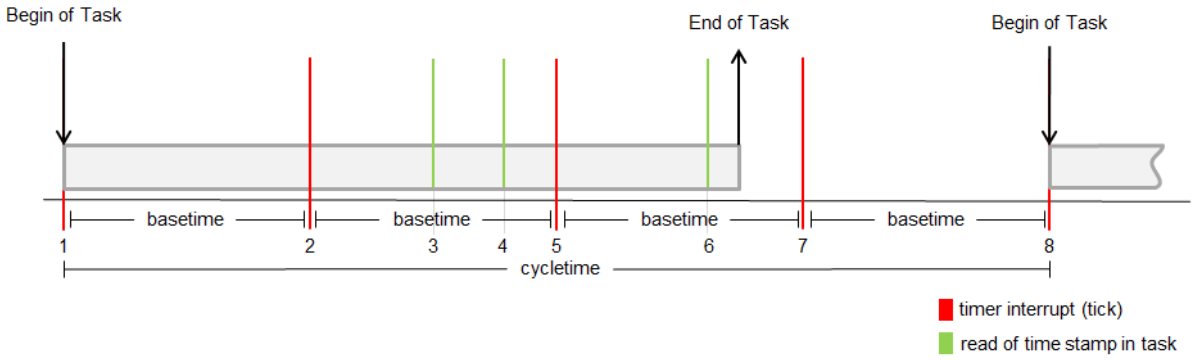
```
TCOM_DECL_INTERFACE("02000002-0000-0000-e000-000000000064", ITcTask)
struct __declspec(novtable) ITcTask : public ITcUnknown
```

Benötigtes include: `TcRtInterfaces.h`

Methoden

Name	Beschreibung
GetCycleCounter [▶ 195]	Anzahl Taskzyklen seit Taskstart abfragen.
GetCycleTime [▶ 196]	Abfrage der Taskzyklenzeit in Nanosekunden, d. h. Zeit zwischen „begin of task“ und nächstem „begin of task“.
GetPriority [▶ 194]	Abfrage der Taskpriorität.
GetCurrentSysTime [▶ 194]	Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).
GetCurrentDcTime [▶ 194]	Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000.
GetCurPentiumTime [▶ 195]	Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Mit der ITcTask Schnittstelle kann die Zeit im Echtzeitkontext gemessen werden.



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 Methode ITcTask:GetPriority

Abfrage der Taskpriorität.

Syntax

```
virtual HRESULT TCOMAPI GetPriority(PULONG
pPriority)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER, wenn Parameter pPriority gleich NULL.

Parameter

Name	Typ	Beschreibung
pPriority	PULONG	[out] Prioritätswert der Task ist in diesem Parameter gespeichert.

Beispiel30: Zeitmessung [▶ 319] veranschaulicht die Verwendung dieser Methode.

12.8.14.2 Methode ITcTask:GetCurrentSysTime

Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Syntax

```
virtual HRESULT TCOMAPI GetCurrentSysTime(PLONGLONG
pSysTime)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER wenn Parameter pSysTime gleich NULL.

Parameter

Name	Typ	Beschreibung
pSysTime	PLONGLONG	[out] In diesem Parameter ist die aktuelle Systemzeit zu Beginn des Taskzyklus gespeichert.

Beispiel30: Zeitmessung [▶ 319] veranschaulicht die Verwendung dieser Methode.

12.8.14.3 Methode ITcTask:GetCurrentDcTime

Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000.

Syntax

```
virtual HRESULT TCOMAPI GetCurrentDcTime(PLONGLONG
pDcTime)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER wenn Parameter pDcTime gleich NULL.

Parameter

Name	Typ	Beschreibung
pDcTime	PLONGLONG	[out] In diesem Parameter ist die Distributed-Clock-Zeit zu Beginn des Taskzyklus gespeichert.

Beispiel30: Zeitmessung [▶ 319] veranschaulicht die Verwendung dieser Methode.

12.8.14.4 Methode ITcTask:GetCurPentiumTime

Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Syntax

```
virtual HRESULT TCOMAPI GetCurPentiumTime(PLONGLONG
pCurTime)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER wenn Parameter pCurTime gleich NULL.

Parameter

Name	Typ	Beschreibung
pCurTime	PLONGLONG	[out] In diesem Parameter wird die aktuelle Zeit (UTC) in 100 Nanosekunden-Intervallen seit dem 1. Januar 1601 gespeichert.

Beispiel30: Zeitmessung [▶ 319] veranschaulicht die Verwendung dieser Methode.

12.8.14.5 Methode ITcTask:GetCycleCounter

Anzahl Taskzyklen seit Taskstart abfragen.

Syntax

```
virtual HRESULT TCOMAPI GetCycleCounter(PULONGLONG
pCnt)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER wenn Parameter pCnt gleich NULL.

Parameter

Name	Typ	Beschreibung
pCnt	PULONGLONG	[out] Die Anzahl von Taskzyklen seit Task gestartet wurde, wird in diesem Parameter gespeichert.

Beispiel30: Zeitmessung [▶ 319] veranschaulicht die Verwendung dieser Methode.

12.8.14.6 Method ITcTask:GetCycleTime

Abfrage der Taskzyklenzeit in Nanosekunden, d. h. Zeit zwischen „begin of task“ und nächstem „begin of task“.

Syntax

```
virtual HRESULT TCOMAPI GetCycleTime(PULONG
pCycleTimeNS)=0;
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

E_POINTER wenn Parameter pCnt gleich NULL.

Parameter

Name	Typ	Beschreibung
pCycleTimeNS	PULONG	[out] In diesem Parameter wird die konfigurierte Taskzyklenzeit in Nanosekunden gespeichert.

Beispiel30: Zeitmessung [▶ 319] veranschaulicht die Verwendung dieser Methode.

12.8.15 Schnittstelle ITcTaskNotification

Führt einen Callback aus, wenn die Zykluszeit beim vorherigen Zyklus überschritten wurde.
Diese Schnittstelle stellt vergleichbare Funktionalitäten wie SPS PlcTaskSystemInfo->CycleTimeExceeded zur Verfügung.

Syntax

```
TCOM_DECL_INTERFACE("9CDE7C78-32A0-4375-827E-924B31021FCD", ITcTaskNotification) struct
__declspec(novtable) ITcTaskNotification: public ITcUnknown
```

Benötigtes include: `TcRtInterfaces.h`

Methoden

Name	Beschreibung
NotifyCycleTimeExceeded [▶ 196]	Wird aufgerufen, wenn die Zykluszeit überschritten wurde.

Beachten Sie, dass der Callback nicht während der Berechnungen, sondern am Ende des Zyklus stattfindet. Also bietet diese Methode keinen Mechanismus, um die Berechnungen sofort zu stoppen.

12.8.15.1 Methode ITcTaskNotification::NotifyCycleTimeExceeded()

Wird aufgerufen, wenn die Zykluszeit zuvor abgelaufen ist, also nicht sofort bei Zeitüberschreitung, sondern danach.

Syntax

```
virtual HRESULT TCOMAPI NotifyCycleTimeExceeded ();
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Parameter

Name	Typ	Beschreibung
ipTask	ITcTask	Verweist auf den aktuellen Task-Kontext.
context	ULONG_PTR	Kontext

12.8.16 Schnittstelle ITcUnknown

ITcUnknown definiert die Referenzzählung, sowie das Abfragen einer Referenz auf eine spezifischere Schnittstelle.

Syntax

```
TCOM_DECL_INTERFACE("00000001-0000-0000-e000-000000000064", ITcUnknown)
```

Deklariert in: TcInterfaces.h

Benötigtes include: -

Methoden

Name	Beschreibung
TcAddRef [▶ 197]	Inkrementiert den Referenzzähler.
TcQueryInterface [▶ 197]	Abfrage der Referenz an eine implementierte Schnittstelle über der IID.
TcRelease [▶ 199]	Dekrementiert den Referenzzähler.

Jede TcCOM Schnittstelle ist direkt oder indirekt von ITcUnknown abgeleitet. Demzufolge implementiert jede TcCOM Modulklasse ITcUnknown, weil sie von ITComObject abgeleitet ist.

Die standardmäßige Implementierung von ITcUnknown sorgt dafür, dass das Objekt nach Freigabe der letzten Referenz gelöscht wird. Aus diesem Grunde darf ein Schnittstellenzeiger nach dem Aufruf von TcRelease() nicht dereferenziert werden.

12.8.16.1 Methode ITcUnknown:TcAddRef

Inkrementiert den Referenzzähler und gibt den neuen Wert zurück.

Syntax

```
ULONG TcAddRef( )
```

Rückgabewert

Daraus resultierender Referenzzählwert.

12.8.16.2 Methode ITcUnknown:TcQueryInterface

Abfrage eines Schnittstellenzeigers in Bezug auf eine Schnittstelle, die per Interface ID (IID) gegeben ist.

Syntax

```
HRESULT TcQueryInterface(RITCID iid, PPVOID pipItf )
```

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte [▶ 169]. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 343].

Wenn die verlangte Schnittstelle nicht verfügbar ist, gibt die Methode ADSERR_DEVICE_NOINTERFACE zurück.

Parameter

Name	Typ	Beschreibung
iid	RITCID	Schnittstelle IID.
pipItf	PPVOID	Zeiger auf Schnittstellenzeiger. Wird gesetzt, wenn der verlangte Schnittstellentyp von der entsprechenden Instanz verfügbar ist.

Beschreibung

Abfrage der Referenz an eine implementierte Schnittstelle über der IID. Es wird empfohlen, Smart Pointer zu verwenden, um Schnittstellenzeiger zu initialisieren und zu halten.

Variante 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;
}
}
```

Die mit dem Smart Pointer verbundene Schnittstellen-ID kann in TcQueryInterface als Parameter verwendet werden. Der Operator „&“ wird den Zeiger auf die interne Schnittstellen-Zeiger-Membervariable des Smart Pointers zurückgeben. Variante 1 geht davon aus, dass der Schnittstellenzeiger initialisiert ist, wenn TcQueryInterface Erfolg anzeigt. Wenn der Bereich bleibt, dann gibt der Destructor des Smart Pointers spObj die Referenz frei.

Variante 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;
}
```

Wenn der Schnittstellenzeiger ip dem Smart Pointer spObj zugewiesen wird, dann wird die TcQueryInterface-Methode implizit aufgerufen mit IID_ITComObject auf der Instanz, auf die ip verweist. Dies führt zu einem kürzeren Code, aber der ursprüngliche Return-Code von TcQueryInterface geht verloren.

12.8.16.3 Methode ITcUnknown:TcRelease

Diese Methode dekrementiert den Referenzzähler.

Syntax

```
ULONG TcRelease( )
```

Rückgabewert

Daraus resultierender Referenzzählwert.

Beschreibung

Dekrementiert den Referenzzähler und gibt den neuen Wert zurück.

Wenn der Referenzzähler 0 wird, löscht das Objekt sich selbst.

12.9 Runtime Library (RtIR0.h)

TwinCAT hat eine eigene Implementierung der Runtime Library. Diese Funktionen sind in RtIR0.h deklariert, einem Teil von TwinCAT SDK.

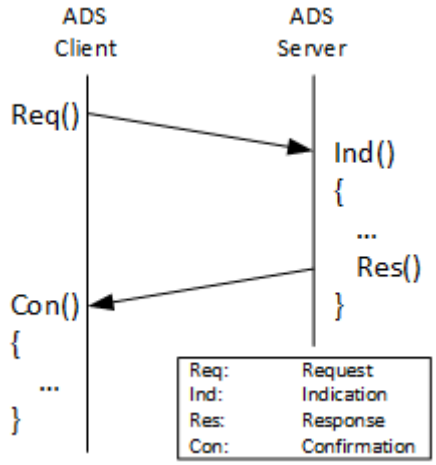
Bereitgestellte Methoden

Name	Beschreibung
abs	Berechnet den absoluten Wert.
atof	Konvertiert einen String (char *buf) in einen Double.
BitScanForward	Sucht vom LSB zum MSB nach einem gesetzten Bit (1).
BitScanReverse	Sucht vom MSB zum LSB nach einem gesetzten Bit (1).
labs	Berechnet den absoluten Wert.
memcmp	Vergleicht zwei Puffer.
memcpy	Kopiert einen Puffer in einen anderen.
memcpy_byte	Kopiert einen Puffer in einen anderen (byte-weise).
memset	Setzt die Bytes eines Puffers auf einen Wert.
qsort	QuickSort zum Sortieren einer Liste.
snprintf	Schreibt formatierte Daten in eine Zeichenfolge.
sprintf	Schreibt formatierte Daten in eine Zeichenfolge.
sscanf	Liest Daten aus einer Zeichenfolge nach Vorgabe einer Formatierung.
strcat	Fügt eine Zeichenkette an eine andere an.
strchr	Sucht ein Zeichen in einer Zeichenkette.
strcmp	Vergleicht zwei Zeichenketten.
strcpy	Kopiert eine Zeichenkette.
strlen	Ermittelt die Länge einer Zeichenkette.
strncat	Fügt eine Zeichenkette an eine andere an.
strncmp	Vergleicht zwei Zeichenketten.
strncpy	Kopiert eine Zeichenkette.
strstr	Sucht eine Zeichenkette in einer Zeichenkette.
strtol	Konvertiert eine Zeichenkette in einen ganzzahligen Wert.
strtoul	Konvertiert eine Zeichenkette in einen ganzzahligen, unsigned Wert.
swscanf	Liest Daten aus einer Zeichenfolge nach Vorgabe einer Formatierung.
tolower	Konvertiert einen Buchstaben in einen Kleinbuchstaben.
toupper	Konvertiert einen Buchstaben in einen Großbuchstaben.
vsnprintf	Schreibt formatierte Daten in eine Zeichenfolge (‚\0‘ Terminierung).
vsprintf	Schreibt formatierte Daten in eine Zeichenfolge.

Alle Funktionen sind an die C++ Runtime Library angelehnt.

12.10 ADS-Kommunikation

Auf Client-Server-Prinzip basierende ADS. Eine ADS-Abfrage ruft die entsprechenden Indikationsmethoden auf der Serverseite auf. Die ADS-Antwort ruft die entsprechende Bestätigungsmethode auf der Client-Seite auf.



ADS Client-Server-Sequenz

In diesem Abschnitt werden sowohl die ausgehende als auch die eingehende ADS-Kommunikation für TwinCAT 3 C++ Module beschrieben.

ADS-Befehlssatz	Beschreibung
AdsReadDeviceInfo [▶ 201]	Mit diesem Befehl können die allgemeinen Geräteinformationen gelesen werden.
AdsRead [▶ 203]	ADS-Lesebefehl, um Daten von einem ADS-Gerät abzufragen.
AdsWrite [▶ 205]	ADS-Schreibbefehl, um Daten an ein ADS-Gerät zu übergeben.
AdsReadState [▶ 209]	ADS-Befehl, um den Zustand von einem ADS-Gerät abzufragen.
AdsWriteControl [▶ 211]	ADS-Steuerungsbefehl, um den Zustand von einem ADS-Gerät zu ändern.
AdsAddDeviceNotification [▶ 213]	Variable beobachten. Der Client wird bei einem Ereignis informiert.
AdsDelDeviceNotification [▶ 215]	Entfernt die Variable, die zuvor verbunden war.
AdsDeviceNotification [▶ 216]	Wird für die Übermittlung des Geräte-Notification-Ereignisses verwendet.
AdsReadWrite [▶ 207]	ADS-Schreib-/Lesebefehl. Mit einem Aufruf werden Daten zu einem ADS-Gerät übermittelt (Write) und dessen Antwortdaten gelesen (Read).

Die ADS Return Codes [▶ 343] gelten für die gesamte ADS-Kommunikation.

Schauen Sie sich als Einstieg das Beispiel07: Empfang von ADS Notifications [▶ 270] an.

12.10.1 AdsReadDeviceInfo

12.10.1.1 AdsReadDeviceInfoReq

Die Methode AdsDeviceInfoReq ermöglicht das Übermitteln eines ADS DeviceInfo Befehls zum Auslesen von Identifizierung und Versionsnummer eines ADS Servers.
AdsReadDeviceInfoCon wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsReadDeviceInfoReq( AmsAddr& rAddr, ULONG invokeId );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

12.10.1.2 AdsReadDeviceInfoInd

Die Methode AdsDeviceInfoInd verweist auf einen ADS DeviceInfo Befehl zum Lesen der Identifizierung und Versionsnummer eines ADS Servers. Anschließend muss AdsReadDeviceInfoRes [▶ 202] aufgerufen werden.

Syntax

```
void AdsReadDeviceInfoInd( AmsAddr& rAddr, ULONG invokeId );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

12.10.1.3 AdsReadDeviceInfoRes

Die Methode AdsReadDeviceInfoRes sendet eine ADS Read Device Info. AdsReadDeviceInfoCon [▶ 203] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsReadDeviceInfoRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, CHAR
name[ADS_FIXEDNAMESIZE], AdsVersion version );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].
name	char[ADS_FIXEDNAMESIZE	[in] enthält den Namen des Geräts.
version	AdsVersion	[in] struct von Build (int), Revision (Byte) und Version (Byte) des Gerätes.

12.10.1.4 AdsReadDeviceInfoCon

Die Methode AdsReadDeviceInfoCon ermöglicht den Empfang von einer ADS-Lesebestätigung einer Geräteinformation. Das empfangende Modul muss diese Methode bereitstellen. Vorher muss das Pendant AdsReadDeviceInfoReq [▶ 201]aufgerufen werden.

Syntax

```
void AdsReadDeviceInfoCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult,
CHAR name[ADS_FIXEDNAMESIZE], AdsVersion version );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].
name	char[ADS_FIXEDNAMESIZE]	[in] enthält den Namen des Geräts.
version	AdsVersion	[in] struct von Build (int), Revision (Byte) und Version (Byte) des Gerätes.

12.10.2 AdsRead

12.10.2.1 AdsReadReq

Die Methode AdsReadReq ermöglicht das Senden eines ADS-Lesebefehls für die Datenübertragung von einem ADS-Gerät.
Beim Eingang der Antwort wird AdsReadCon [▶ 205]aufgerufen.

Syntax

```
int AdsReadReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbLength	ULONG	[in] Enthält die Länge, in Bytes, der zu lesenden Daten (pData).

12.10.2.2 AdsReadInd

Die Methode AdsReadInd ermöglicht den Empfang von einer ADS-Leseanforderung. Die AdsReadRes [▶ 204] muss aufgerufen werden, um das Ergebnis zu senden.

Syntax

```
void AdsReadInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength );
```

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbLength	ULONG	[in] Enthält die Länge, in Bytes, der zu lesenden Daten (pData).

12.10.2.3 AdsReadRes

Die Methode AdsReadRes ermöglicht das Senden einer ADS-Leseantwort. AdsReadCon [▶ 205] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsReadRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Lesebefehls, siehe AdsStatuscodes [▶ 343].
cbLength	ULONG	[in] enthält die Länge, in Bytes, der gelesenen Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die Daten sich befinden.

12.10.2.4 AdsReadCon

Die Methode AdsReadCon ermöglicht den Empfang einer ADS-Lesebestätigung. Das empfangende Modul muss diese Methode bereitstellen.
Das Pendant AdsReadReq [▶ 203] bildet muss vorher aufgerufen worden sein.

Syntax

```
void AdsReadCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Lesebefehls, siehe AdsStatuscodes [▶ 343].
cbLength	ULONG	[in] enthält die Länge, in Bytes, der gelesenen Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

12.10.3 AdsWrite

12.10.3.1 AdsWriteReq

Die Methode AdsWriteReq ermöglicht das Senden eines ADS-Schreibbefehls, für Übertragung von Daten an ein ADS-Gerät.
Beim Eingang der Antwort wird AdsWriteCon [▶ 207] aufgerufen.

Syntax

```
int AdsWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength, PVOID pData );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbLength	ULONG	[in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

12.10.3.2 AdsWriteInd

Die Methode AdsWriteInd gibt einen ADS-Schreibbefehl an, um Daten zu einem ADS-Gerät zu übermitteln. Die AdsWriteRes [▶ 206] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void AdsWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength, PVOID pData );
```

Rückgabewert

void

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbLength	ULONG	[in] Enthält die Länge, in Bytes, der zu schreibenden Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

12.10.3.3 AdsWriteRes

Die Methode AdsWriteRes sendet eine ADS-Schreibantwort. AdsWriteCon [▶ 207] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

Typ: int

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] Enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].

ADS Return Code - siehe AdsStatuscodes [▶ 343].

12.10.3.4 AdsWriteCon

Die Methode AdsWriteCon ermöglicht den Empfang einer ADS-Schreibbestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsWriteReq [▶ 205] bildet das Gegenstück und muss zuvor aufgerufen worden sein.

Syntax

```
void AdsWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].

12.10.4 AdsReadWrite

12.10.4.1 AdsReadWriteReq

Die Methode AdsReadWriteReq ermöglicht das Senden eines ADS-Lese-/Schreibbefehls für die Datenübergabe an ein und von einem ADS-Gerät. Die AdsReadWriteCon [▶ 209] wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsReadWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbReadLength, ULONG cbWriteLength, PVOID pData );
```

Rückgabewert

Typ: int

Fehlercode, siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbReadLength	ULONG	[in] enthält die Länge, in Bytes, der zu lesenden Daten (pData).
cbWriteLength	ULONG	[in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

12.10.4.2 AdsReadWriteInd

Die Methode AdsReadWriteInd gibt einen ADS-Lese-/Schreibbefehl an, um Daten zu einem ADS-Gerät zu übermitteln. Die AdsReadWriteRes [▶ 210] muss aufgerufen werden, um das Ergebnis zu senden.

Syntax

```
void AdsReadWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup,
ULONG indexOffset, ULONG cbReadLength, ULONG cbWriteLength, PVOID pData );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbReadLength	ULONG	[in] enthält die Länge, in Bytes, der zu lesenden Daten (pData).
cbWriteLength	ULONG	[in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

12.10.4.3 AdsReadWriteRes

Die Methode AdsReadWriteRes ermöglicht den Empfang einer ADS-Lese-/Schreibbestätigung. AdsReadWriteCon [▶ 209] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsReadWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].
cbLength	ULONG	[in] enthält die Länge, in Bytes, der gelesenen Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

12.10.4.4 AdsReadWriteCon

Die Methode AdsReadWriteCon ermöglicht den Empfang von einer ADS-Lese-/Schreibbestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsReadWriteReq [▶ 207] bildet das Gegenstück und muss zuvor aufgerufen werden.

Syntax

```
void AdsReadWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].
cbLength	ULONG	[in] enthält die Länge, in Bytes, der gelesenen Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

12.10.5 AdsReadState

12.10.5.1 AdsReadStateReq

Die Methode AdsReadStateReq ermöglicht die Übermittlung eines ADS-Statuslesebefehls für das Lesen des ADS- und des Gerätezustands von einem ADS Server. Beim Eingang der Antwort wird AdsReadStateCon [▶ 211] aufgerufen.

Syntax

```
int AdsReadStateReq(AmsAddr& rAddr, ULONG invokeId);
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

12.10.5.2 AdsReadStateInd

Die Methode AdsReadStateInd zeigt einen ADS-Statuslesebefehl an für das Lesen des ADS-Zustands und des Gerätezustands von einem ADS-Gerät. Die AdsReadStateRes [▶ 210] muss aufgerufen werden, um das Ergebnis zu senden.

Syntax

```
void AdsReadStateInd( AmsAddr& rAddr, ULONG invokeId );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

12.10.5.3 AdsReadStateRes

Die Methode AdsWriteRes ermöglicht das Senden von einer ADS-Statusleseantwort. AdsReadStateCon [▶ 211] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsReadStateRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT
deviceState );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].
adsState	USHORT	[in] enthält den ADS-Zustand des Geräts.
deviceState	USHORT	[in] enthält den Gerätestatus des Geräts.

12.10.5.4 AdsReadStateCon

Die Methode AdsWriteCon ermöglicht den Empfang von einer ADS-Statuslesebestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsReadStateReq [▶ 209] bildet das Gegenstück und muss zuvor aufgerufen werden.

Syntax

```
void AdsReadStateCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT
deviceState );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].
adsState	USHORT	[in] enthält den ADS-Zustand des Geräts.
deviceState	USHORT	[in] enthält den Gerätestatus des Geräts.

12.10.6 AdsWriteControl

12.10.6.1 AdsWriteControlReq

Mit der Methode AdsWriteControlReq kann ein ADS-Schreibsteuerungsbefehl zur Änderung des ADS- und Gerätestatus eines ADS Servers gesendet werden. Beim Eingang der Antwort wird AdsWriteControlCon [▶ 212] aufgerufen.

Syntax

```
int AdsWriteControlReq( AmsAddr& rAddr, ULONG invokeId, USHORT adsState,
USHORT deviceState, ULONG cbLength, PVOID pData );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
adsState	USHORT	[in] neuer ADS-Status (siehe enum nAdsState in Ads.h).
deviceState	USHORT	[in] neuer Geräte-Status.
cbLength	ULONG	[in] enthält die Länge, in Bytes, der Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

12.10.6.2 AdsWriteControlInd

Mit der Methode AdsWriteControlInd kann ein ADS-Schreibsteuerungsbefehl zur Änderung des ADS- und Gerätestatus eines ADS-Geräts gesendet werden. Die AdsWriteControlRes [▶ 212] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void AdsWriteControlInd( AmsAddr& rAddr, ULONG invokeId, USHORT adsState, USHORT deviceState, ULONG
cbLength, PVOID pDeviceData );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
adsState	USHORT	[in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
deviceState	USHORT	[in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
cbLength	ULONG	[in] enthält die Länge, in Bytes, der Daten (pData).
pData	PVOID	[in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

12.10.6.3 AdsWriteControlRes

Die Methode AdsWriteControlRes ermöglicht das Senden von einer ADS-Schreibsteuerungsantwort. AdsWriteControlCon [▶ 212] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsWriteControlRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].

12.10.6.4 AdsWriteControlCon

Die Methode AdsWriteCon ermöglicht den Empfang von einer ADS-Schreibsteuerungsbestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsWriteControlReq [▶ 211] bildet das Gegenstück und muss zuvor aufgerufen werden.

Syntax

```
void AdsWriteControlCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].

12.10.7 AdsAddDeviceNotification

12.10.7.1 AdsAddDeviceNotificationReq

Die Methode AdsAddDeviceNotificationReq ermöglicht das Senden eines ADS-Hinzufügen-Geräte-Notification-Befehls, um eine Geräte-Notification zu einem ADS-Gerät hinzuzufügen. AdsAddDeviceNotificationCon [▶ 214] wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsAddDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG
indexOffset,
AdsNotificationAttrib noteAttrib);
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
noteAttrib	AdsNotificationAttrib	[in] enthält die Spezifikation der Notification-Parameter (cbLength, TransMode, MaxDelay).

12.10.7.2 AdsAddDeviceNotificationInd

Die Methode AdsAddDeviceNotificationInd sollte das Senden von AdsDeviceNotification [▶ 216] ermöglichen. Die AdsAddDeviceNotificationRes [▶ 214] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void AdsAddDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG
indexOffset, AdsNotificationAttrib noteAttrib );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
indexGroup	ULONG	[in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.
indexOffset	ULONG	[in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.
noteAttrib	AdsNotificationAttrib	[in] enthält die Spezifikation der Notification-Parameter (cbLength, TransMode, MaxDelay).

12.10.7.3 AdsAddDeviceNotificationRes

Die Methode AdsAddDeviceNotificationRes ermöglicht das Senden von einer ADS-Gerät-Hinzufügen-Notification-Antwort. AdsAddDeviceNotificationCon [▶ 214] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes.
Handle	ULONG	[in] Handle auf generierte Geräte-Notification.

12.10.7.4 AdsAddDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon bestätigt eine ADS-Gerät-Hinzufügen-Notification-Anforderung. AdsAddDeviceNotificationReq [▶ 213] bildet das Gegenstück und muss zuvor aufgerufen werden.

Syntax

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes.
Handle	ULONG	[in] Handle auf generierte Geräte-Notification.

12.10.8 AdsDelDeviceNotification

12.10.8.1 AdsDelDeviceNotificationReq

Die Methode AdsDelDeviceNotificationReq ermöglicht das Senden eines ADS-Gerät-Löschen-Notification-Befehls, um eine Geräte-Notification von einem ADS-Gerät zu entfernen. Die AdsDelDeviceNotificationCon [▶ 216] wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsDelDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG hNotification );
```

Rückgabewert

Typ: int

Fehlercode - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
hNotification	ULONG	[in] enthält das Handle der zu entfernenden Notification.

12.10.8.2 AdsDelDeviceNotificationInd

Die Methode AdsAddDeviceNotificationCon ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification-Bestätigung. Das empfangende Modul muss diese Methode bereitstellen. Die AdsDelDeviceNotificationRes [▶ 216] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].

12.10.8.3 AdsDelDeviceNotificationRes

Die Methode AdsAddDeviceNotificationRes ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification. AdsDelDeviceNotificationCon [▶ 216] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsDelDeviceNotificationRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

Int

Gibt das Ergebnis des ADS-Befehls zurück, siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Befehls, siehe AdsStatuscodes [▶ 343].

12.10.8.4 AdsDelDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification-Bestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsDelDeviceNotificationReq [▶ 215] bildet das Gegenstück und muss zuvor aufgerufen werden.

Syntax

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des gesendeten Befehls, die InvokeId wird vom Quellgerät spezifiziert und dient der Identifizierung der Befehle.
nResult	ULONG	[in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 343].

12.10.9 AdsDeviceNotification

12.10.9.1 AdsDeviceNotificationReq

Die Methode AdsAddDeviceNotificationReq ermöglicht das Senden einer ADS-Geräte-Notification, um ein ADS-Gerät zu informieren. Die AdsDeviceNotificationInd [▶ 217] wird auf dem Gegenstück aufgerufen.

Syntax

```
int AdsDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength, AdsNotificationStream
notifications[] );
```

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 343].

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis der Geräte-Notification-Anzeige.
notifications[]	AdsNotificationStream	[in] enthält Informationen der Geräte-Notification(s).

12.10.9.2 AdsDeviceNotificationInd

Die Methode AdsDeviceNotificationInd ermöglicht den Empfang von einer ADS-Geräte-Notification-Anzeige. Das empfangende Modul muss diese Methode bereitstellen. Der Empfang wird nicht quittiert.
Die AdsDeviceNotificationCon [▶ 217] muss auf Seite der AdsDeviceNotificationReq [▶ 216] aufgerufen werden um die Übermittlung zu überprüfen.

Syntax

```
void AdsDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength,
AdsNotificationStream* pNotifications );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom antwortenden ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
cbLength	ULONG	[in] enthält die Länge von pNotifications.
pNotifications	AdsNotificationStream*	[in] Zeiger auf die Notifications. Dieses Array besteht aus AdsStampHeader mit Notification Handle und Daten via AdsNotificationSample.

12.10.9.3 AdsDeviceNotificationCon

Mit der Methode AdsAddDeviceNotificationCon kann der Absender die Übermittlung einer ADS-Geräte-Notification überprüfen.
AdsDeviceNotificationReq [▶ 216] muss dafür zuvor aufgerufen werden.

Syntax

```
void AdsDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Rückgabewert

void

Parameter

Name	Typ	Beschreibung
rAddr	AmsAddr&	[in] Struktur mit NetId und Portnummer vom ADS-Server.
invokeId	ULONG	[in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.
nResult	ULONG	[in] enthält das Ergebnis der Geräte-Notification-Anzeige.

12.11 Mathematische Funktionen

TwinCAT hat eigene mathematische Funktionen implementiert, weil die math.h Implementierung von Microsoft nicht echtzeitfähig ist.
Diese Funktionen sind in TcMath.h deklariert, einem Teil von TwinCAT SDK. Die Operationen werden für x64 mittels SSE ausgeführt; auf x86 Sytemen wird die FPU verwendet.

TwinCAT 3.1 4018 und früher

TwinCAT 3.1 4018 stellte eine fpu87.h mit den gleichen Methoden bereit. Diese existiert weiterhin, und leitet auf die TcMath.h um.

Bereitgestellte Methoden

Name	Beschreibung
sqr_	Quadriert.
sqrt_	Berechnet die Quadratwurzel.
sin_	Berechnet den Sinus.
cos_	Berechnet den Kosinus.
tan_	Berechnet den Tangens.
atan_	Berechnet den Winkel, dessen Tangens der angegebene Wert ist.
atan2_	Berechnet den Winkel, dessen Tangens der Quotient zweier angegebener Werte ist.
asin_	Berechnet den Winkel, dessen Sinus der angegebene Wert ist.
acos_	Berechnet den Winkel, dessen Kosinus der angegebene Wert ist.
exp_	Berechnet e hoch angegebene Potenz.
log_	Berechnet den Logarithmus eines angegebenen Werts.
log10_	Berechnet den Logarithmus zur Basis 10 eines angegebenen Werts.
fabs_	Berechnet den Absolutwert.
fmod_	Berechnet den Restbetrag.
ceil_	Berechnet die kleinste Integerzahl die größer oder gleich der angegebenen Zahl ist.
floor_	Berechnet die größte Integerzahl die kleiner oder gleich der angegebenen Zahl ist.
pow_	Berechnet eine angegebene Zahl hoch angegebener Potenz.
sincos_	Berechnet den Sinus und den Kosinus von x.
fmodabs_	Berechnet den Absolutbetrag, der die euklidische Definition der mod-Operation erfüllt.
round_	Berechnet einen Wert und rundet ihn auf den nächsten Integer.
round_digits_	Berechnet einen gerundeten Wert mit einer festgelegten Anzahl Dezimalstellen.
coubic_	Berechnet die Kubikzahl.
ldexp_	Berechnet eine reelle Zahl (double) aus Mantisse und Exponent.
ldexpf_	Berechnet eine reelle Zahl (float) aus Mantisse und Exponent.
sinh_	Berechnet den Hyperbelsinus des angegebenen Winkels.
cosh_	Berechnet den Hyperbelkosinus des angegebenen Winkels.
tanh_	Berechnet den Hyperbeltangens des angegebenen Winkels.
finite_	Ermittelt, ob der angegebene Wert endlich ist.
isnan_	Ermittelt, ob der gegebene Wert keine Zahl ist (NAN „not a number“).
rands_	Berechnet eine Pseudo-Zufallszahl zwischen 0 und 32767. Der Parameter holdrand wird zufällig gesetzt und bei jedem Aufruf geändert.

- Die Funktionen haben die Endung „_“ (Unterstrich), um sie als TwinCAT Implementierung zu kennzeichnen.
- Die meisten sind analog math.h von Microsoft konzipiert, nur für den Datentyp Double.

Siehe auch

MSDN Dokumentation von Funktionen analog math.h.

12.12 Zeitfunktionen

TwinCAT bietet Funktionen für die Zeitumwandlung, sie werden in TcTimeConversion.h, die Teil von TwinCAT SDK ist, deklariert.

Bereitgestellte Methoden

Name	Beschreibung
TcDayOfWeek(WORD day, WORD month, WORD year)	Ermittelt den Wochentag. Eingabe: day (0..30) und month(1..12) Rückgabe: 0 ist Sonntag, 6 ist Samstag
TcIsLeapYear	Ermittelt, ob das gegebene Jahr ein Schaltjahr ist.
TcDaysInYear	Ermittelt die Anzahl Tage im gegebenen Jahr.
TcDaysInMonth	Ermittelt die Anzahl Tage im gegebenen Monat.
TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, FILETIME *lpFileTime);	Konvertiert die gegebene Systemzeit in eine Dateizeit um.
TcFileTimeToSystemTime(const FILETIME *lpFileTime, SYSTEMTIME* lpSystemTime);	Konvertiert die gegebene Dateizeit in eine Systemzeit um.
TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, ULONGLONG& ul64FileTime);	Konvertiert die gegebene Systemzeit in eine Dateizeit um (ULONGLONG Format).
TcFileTimeToSystemTime(const ULONGLONG& ul64FileTime, SYSTEMTIME* lpSystemTime);	Konvertiert die gegebene Dateizeit (ULONGLONG Format) in eine Systemzeit um.
TcIsISO8601TimeFormat(PCCH sDT)	Überprüft, ob ein PCCH dem Zeitformat ISO8601 folgt.
TcDecodeDateTime(PCCH sDT)	Konvertiert ein ULONG als DateTime aus dem PCCH in ISO8601 Format.
TcDecodeDcTime(PCCH sDT)	Konvertiert ein LONGLONG als DcTime aus dem PCCH in ISO8601 Format.
TcDecodeFileTime(PCCH sFT)	Konvertiert ein LONGLONG als FileTime aus dem PCCH in ISO8601 Format.
TcEncodeDateTime(ULONG value, PCHAR p, UINT len)	Konvertiert einen String (p, len) in ISO8601 Format auf Basis des ULONG value in DateTime Format. Minimale Länge für p ist 24 Byte.
TcEncodeDcTime(LONGLONG value, PCHAR p, UINT len)	Konvertiert einen String (p, len) in ISO8601 Format auf Basis des LONGLONG in DcTime Format. Minimale Länge für p ist 32 Byte.
TcEncodeFileTime(LONGLONG value, PCHAR p, UINT len)	Konvertiert einen String (p, len) in ISO8601 Format auf Basis des LONGLONG in FileTime Format. Minimale Länge für p ist 32 Byte.
TcDcTimeToFileTime(LONGLONG dcTime)	Konvertiert ein LONGLONG als FileTime aus dem LONGLONG in DcTime.
TcFileTimeToDcTime(LONGLONG fileTime);	Konvertiert ein LONGLONG als DcTime aus dem LONGLONG in FileTime.
TcDcTimeToDateTime(LONGLONG dcTime)	Konvertiert ein ULONG als DateTime aus dem LONGLONG in DcTime.
TcDateTimeToDcTime(ULONG dateTime)	Konvertiert ein LONGLONG als DcTime aus dem ULONG in DateTime.
TcFileTimeToDateTime(LONGLONG fileTime)	Konvertiert ein ULONG als DateTime aus dem LONGLONG in FileTime.
TcDateTimeToFileTime(ULONG dateTime)	Konvertiert ein LONGLONG als FileTime aus dem ULONG in DateTime.

- Weitere Informationen zu unterschiedlichen Zeitquellen werden hier beschrieben: https://infosys.beckhoff.com/content/1031/ethercatsystem/2469114379.html

12.13 STL / Container

TwinCAT 3 C++ unterstützt STL bezüglich

- List
- Map
- Set
- Stack
- String
- Vector
- WString
- Algorithm (wie binary_search)
  - Siehe *%TWINCAT3DIR%\Sdk\Include\Stl\Stl\algorithm* für eine konkrete Liste der unterstützten Algorithmen.

Einschränkungen

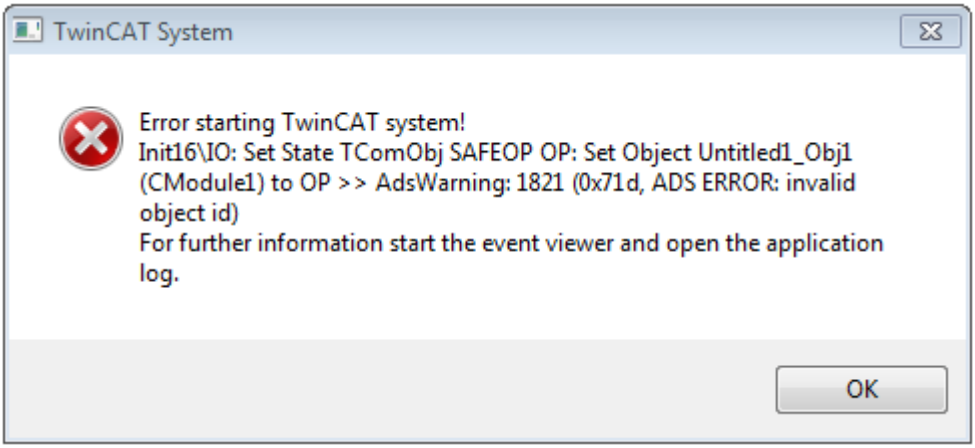
- Classtemplates existieren nicht für alle Datentypen.
- Einige Header-Dateien sollten nicht direkt genutzt werden.

Eine genauere Dokumentation zu der Speicherverwaltung, die bei STL verwendet wird, befindet sich im Kapitel Speicherallokation [▶ 158].

12.14 Fehlermeldungen - Verständnis

Sie enthalten in TwinCAT sehr ausführliche Informationen über aufgetretene Fehler.

So bedeutet z. B. diese Fehlermeldung:



- Der Fehler trat beim Übergang von SAFE OP zu OP auf.
- Das betroffene Objekt ist „Untitled1_Obj1“ (CModule1).
- Der Fehlercode 1821 / 0x71d zeigt an, dass die Objekt-Id ungültig ist.

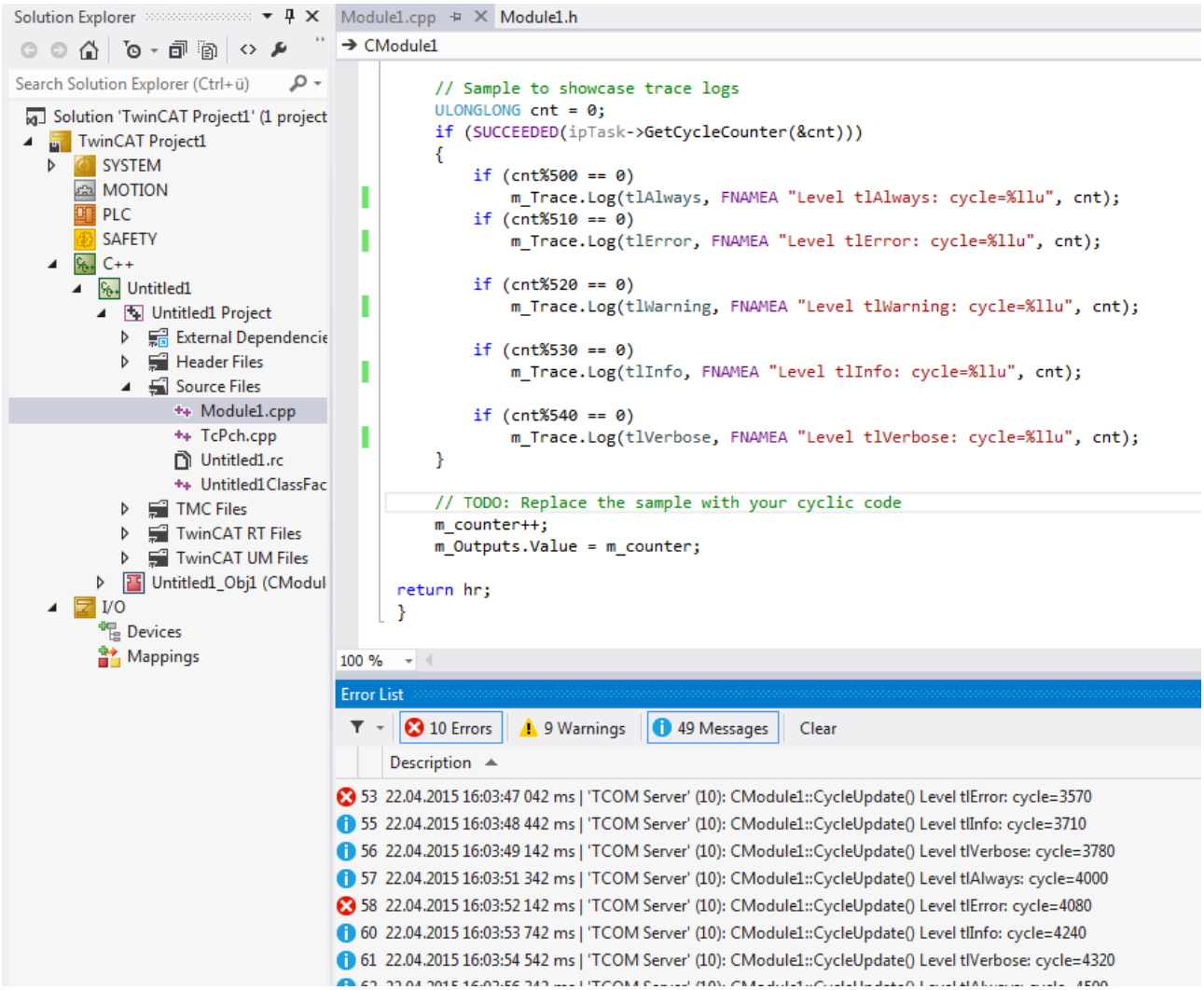
Sie sollten also die Methode „SetObjStateSP()“, die für diesen Übergang verantwortlich ist, genauer untersuchen. Im Falle des generierten Standardcodes erkennen Sie, dass das Hinzufügen des Moduls dort stattfindet.

Der Grund für das Auftreten dieses Fehlers besteht darin, dass diesem Modul keine Task zugewiesen wurde, also kann das Modul keine Task haben, in der es ausgeführt wird.

12.15 Modul-Nachrichten zum Engineering (Logging / Tracing)

Übersicht

TwinCAT 3 C++ bietet die Möglichkeit, Nachrichten aus einem C++ Modul in das TwinCAT 3 Engineering als Tracing oder Logging zu senden.



Syntax

Die Syntax zur Aufzeichnung von Meldungen ist folgende:

```
m_Trace.Log(TLEVEL, FNMACRO"A message", …);
```

Mit diesen Eigenschaften:

- `TLEVEL` kategorisiert eine Meldung in eine von fünf Ebenen.
  Das Aufzeichnen der höheren Ebene beinhaltet immer auch das Aufzeichnen der unteren Ebenen: d.h. eine auf Ebene „tlWarning“ klassifizierte Meldung wird auftreten mit Ebene „tlAlways“, „tlError“ und „tlWarning“ - sie wird die „tlInfo“ und „tlVerbose“ Meldungen NICHT aufzeichnen.

Level 0	tlAlways
Level 1	tlError
Level 2	tlWarning
Level 3	tlInfo
Level 4	tlVerbose

- `FNMACRO` kann verwendet werden, um den Funktionsnamen vor die zu druckende Meldung zu setzen

- FENTERA: Wird beim Eintritt in eine Funktion verwendet, druckt den Funktionsnamen gefolgt von „>>>“.
- FNAMEA: Wird innerhalb einer Funktion verwendet, druckt den Funktionsnamen.
- FLEAVEA: Wird beim Verlassen einer Funktion verwendet, druckt den Funktionsnamen gefolgt von „<<<“.

- Zur Ausgabe von Zeigern und anderen Variablen mit Plattform-spezifischer Größe wird der Formatspezifizierer `%q` verwendet.

Beispiel

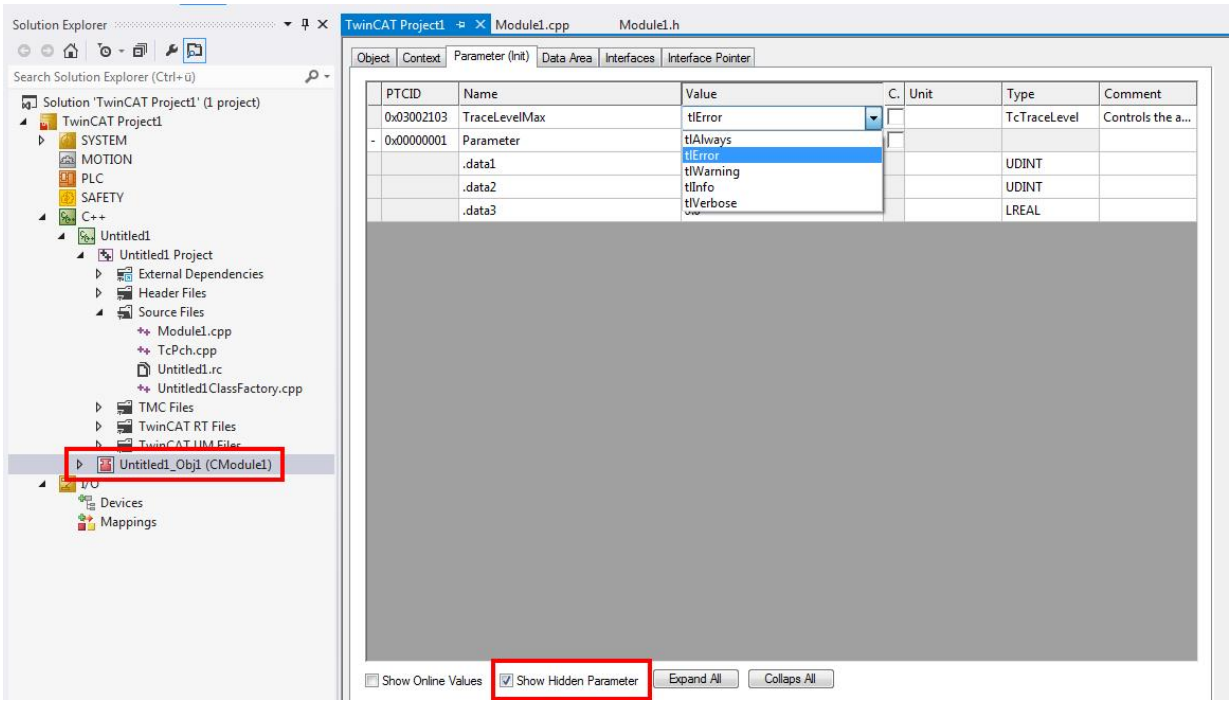
```
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // 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;
}
```

Verfolgungsebene verwenden

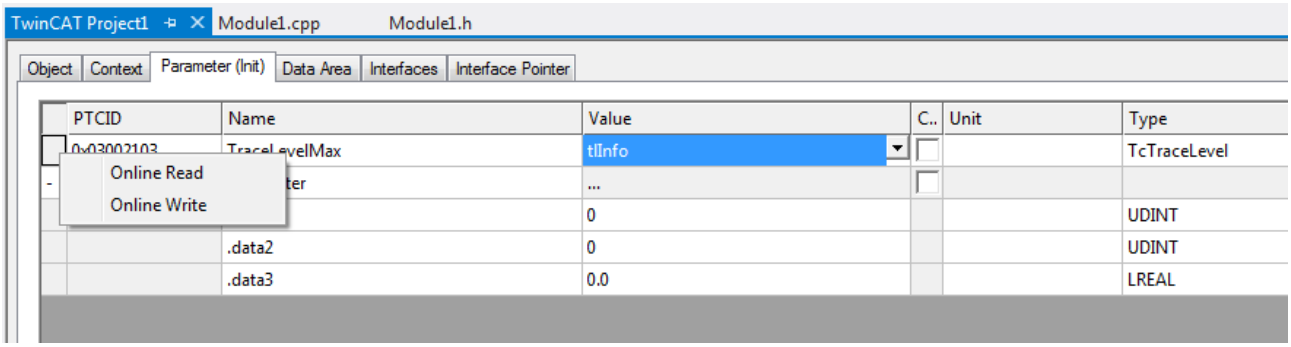
Auf Höhe der Modulinstanz besteht die Möglichkeit, die Verfolgungsebene vorzukonfigurieren.

1. Navigieren Sie zur Instanz des Moduls im Solution-Baum.
2. Wählen Sie den Karteireiter **Parameter (Init)** auf der rechten Seite.
3. Achten Sie darauf, dass sie **Show Hidden Parameters** aktivieren.
4. Wählen Sie die Verfolgungsebene.

5. Um alles zu testen, wählen Sie die höchste Ebene **tlVerbose**.



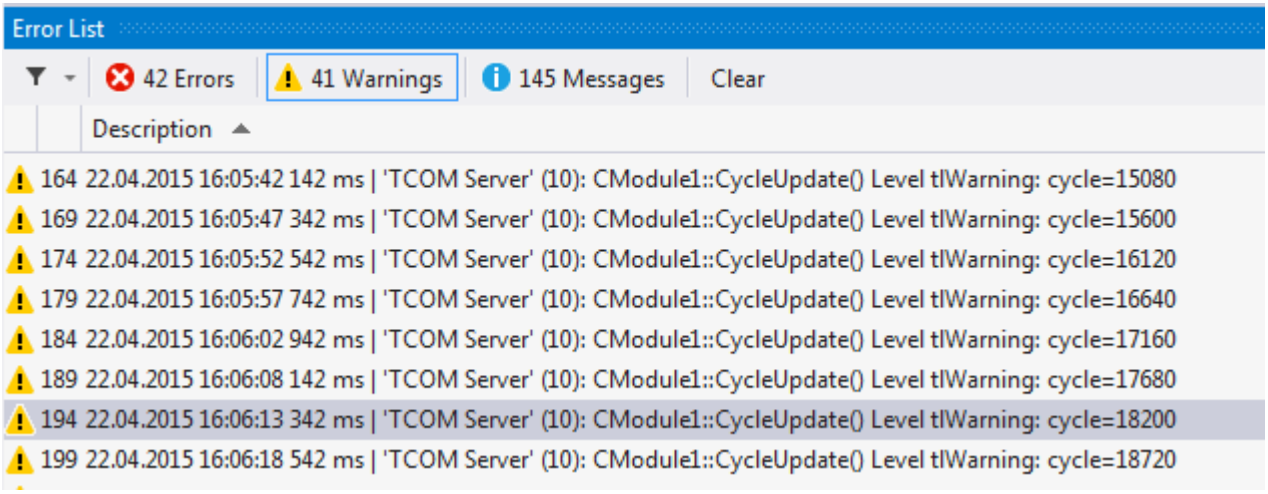
Alternativ dazu können Sie die Verfolgungsebene auch während der Laufzeit ändern, indem Sie zur Instanz gehen, eine Ebene bei **Value** für TraceLevelMax-Parameter auswählen, dann einen Rechtsklick vor der ersten Spalte machen und **Online Write** auswählen.



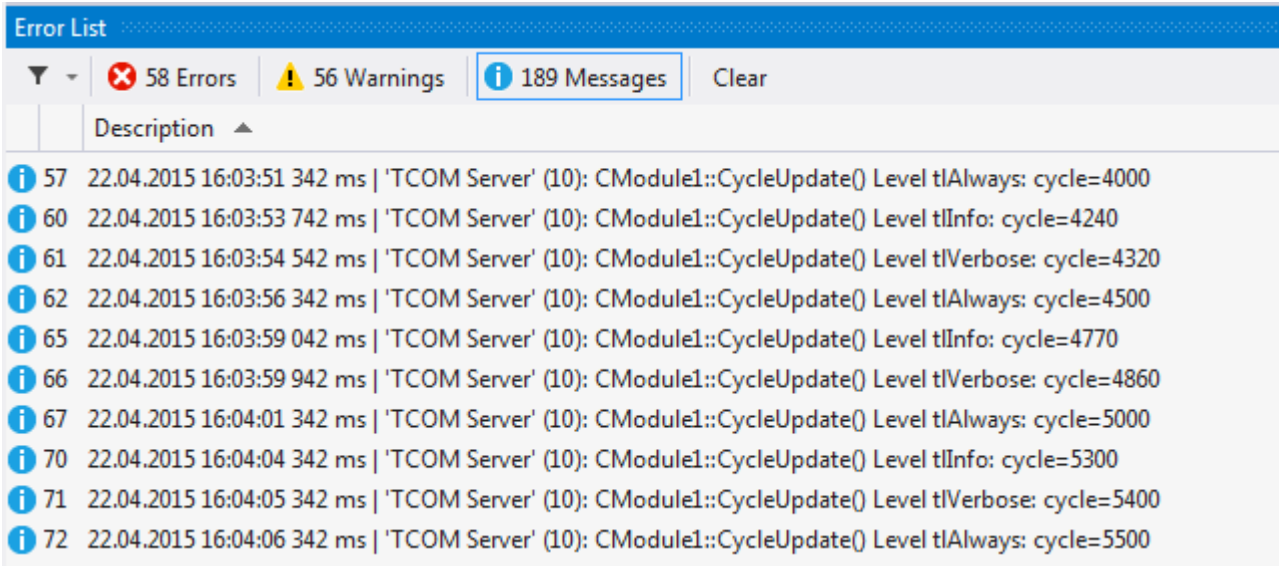
Meldungskategorien filtern

Visual Studio Error List ermöglicht das Filtern der Einträge nach deren Kategorie. Die drei Kategorien **Errors**, **Warnings** und **Messages** können unabhängig voneinander durch einfaches Umschalten der Tasten aktiviert oder deaktiviert werden.

In diesem Screenshot ist nur **Warnings** aktiviert - **Errors** und **Messages** dagegen sind deaktiviert:



In diesem Screenshot ist nur **Messages** aktiviert - **Errors** und **Warnings** dagegen sind für die Anzeige deaktiviert:



13 How to...?

Im Folgenden finden Sie eine Sammlung häufig gestellter Fragen zu allgemeinen Programmierbeispielen und dem Handling von TwinCAT C++-Modulen.

13.1 Verwendung des Automation Interface

Das Automation Interface kann für C++ Projekte verwendet werden.

Hierzu zählen das Anlegen von Projekten [▶ 89] sowie auch die Nutzung der Assistenten zum Anlegen von Modulenklassen [▶ 90].
Zusätzlich können die Projekteigenschaften gesetzt, der TMC Code Generator und das Publishen von Modulen aufgerufen werden. Die zugehörige Dokumentation [▶ 351] ist Teil des Automation Interfaces.

Unabhängig von der Programmiersprache kann der Zugriff auf, die Erstellung von und der Umgang mit TcCOM Modulen [▶ 354] relevant sein.

Von dort aus können gewöhnliche Aufgaben des Systemmanagers wie Verknüpfung von Variablen ausgeführt werden.

13.2 Windows 10 als Zielsystem bis TwinCAT 3.1 Build 4022.2

Bei Windows 10 Zielsystemen können die übertragenen Dateien nicht überschrieben werden, sondern müssen vorher umbenannt werden.

Bis TwinCAT 3.1 Build 4022.2 muss hierfür im Deployment des TMC Editors [▶ 133] die Option **Rename Destination** aktiviert werden. In späteren Versionen wird dieses implizit gemacht, wenn das Zielsystem Windows 10 als Betriebssystem einsetzt.

13.3 Module veröffentlichen auf der Kommandozeile

Durch den folgenden Aufruf kann auch von der Kommandozeile aus der Veröffentlichungsprozess eines Moduls im TwinCAT Engineering (XAE) angestoßen werden:

```
msbuild CppProject.vcxproj /t:TcPublishTMX
```

Der Parameter `CppProject.vcxproj` muss entsprechend der vorhandenen Projektdatei angepasst werden.

Die Ausgabe erfolgt in das Engineering Repository - unter C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository (TwinCAT 3.1 Build 4024: C:\TwinCAT\3.1\Repository).

13.4 Clone

Die Laufzeitdateien können per Dateikopie von einer zur anderen Maschine übertragen werden, wenn sie von derselben Plattform stammen und mit äquivalenter Hardware-Ausrüstung verbunden sind.

Die folgenden Schritte beschreiben ein einfaches Verfahren zum Übertragen einer Binärkonfiguration von einer Maschine „Quelle“ zu einer anderen Maschine „Ziel“.

1. Leeren Sie den Boot-Ordner auf der Quell-Maschine.
   ⇨ < TC3.1.4026.0: *C:\TwinCAT\3.1\Boot*
   ⇨ >=TC3.1.4026.0: *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot*
2. Erstellen (oder aktivieren) Sie das Modul auf der Quellmaschine.
3. Übertragen Sie den Boot-Ordner von der Quelle zum Ziel.
   Dieser Ordner enthält auch das Repository, welches die nötigen TMX Dateien enthält. Der Ordner befindet sich sowohl auf der Quelle als auch auf dem Ziel an dem folgenden Speicherpfad.

⇨ < TC3.1.4026.0: *C:\TwinCAT\3.1\Boot*

⇨ >=TC3.1.4026.0: *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot*

4. Bei TwinCAT-Treiber-Projekten (.sys): Übertragen Sie den Treiber *MYDRIVER.sys* und ggf. auch die PDB-Datei.

   ⇨ < 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. Bei TwinCAT-Treiber-Projekten (.sys) und falls Treiber neu auf einer Maschine sind:
   TwinCAT muss einmalig eine Registrierung ausführen. Schalten Sie TwinCAT dafür per SysTray (Rechts-Klick->**System**->**Start/Restart**) in den RUN-Modus.
   Alternativ kann dieser Aufruf verwendet werden („%1“ als Treibernamen ersetzen):

   ⇨ < 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
   ```

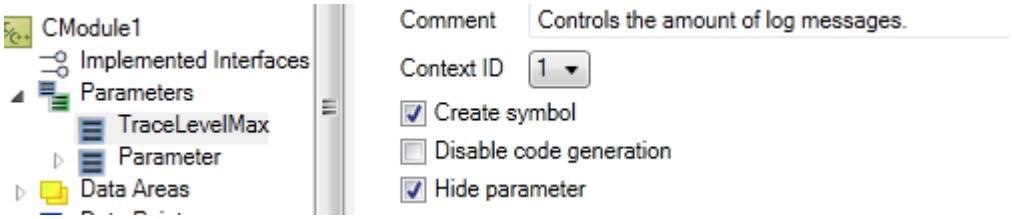
⇨ Sie können nun die Zielmaschine starten.

Umgang mit Lizenzen

Beachten Sie, dass die Lizenzen nicht auf diese Weise übertragen werden können. Verwenden Sie bitte vorinstallierte Lizenzen, Volumenlizenzen oder andere Mechanismen, um Lizenzen bereitzustellen.

13.5 Zugriff auf Variablen über ADS

Variablen von C++ Modulen sind über ADS erreichbar, wenn die Variablen im TMC Editor als „Create Symbol“ gekennzeichnet sind:



Der Name der Variablen für den Zugriff per ADS ist abgeleitet vom Name der Instanz.

Für den TraceLevelMax Parameter könnte er lauten:

```
Untitled1_Obj1 (CModule1).TraceLevelMax
```

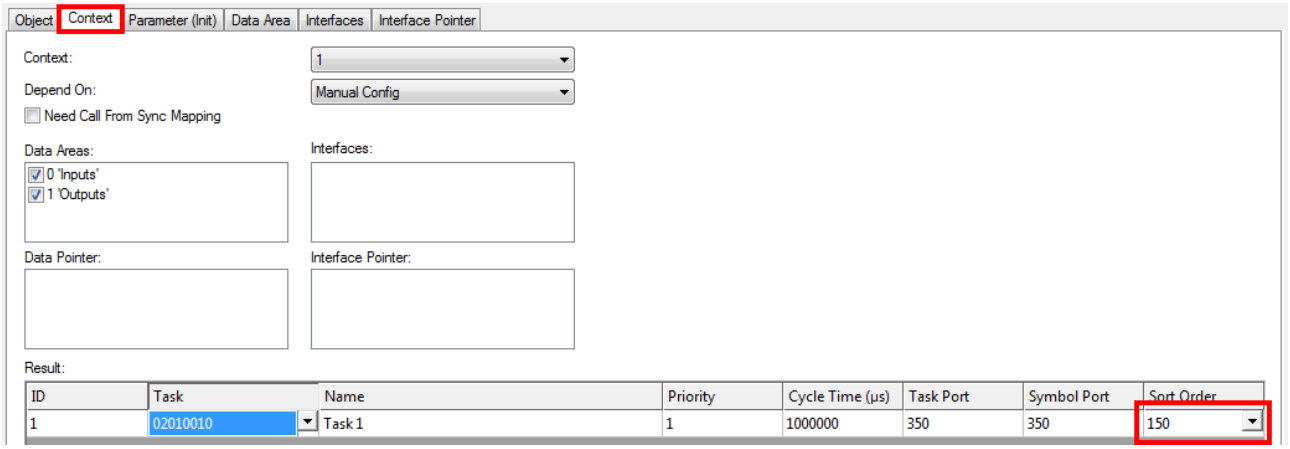
13.6 TcCallAfterOutputUpdate für C++ Module

Vergleichbar mit dem SPS Attribut TcCallAfterOutputUpdate können C++ Module nach der Ausgang-Aktualisierung aufgerufen werden.
Äquivalent zu ITcCyclic [▶ 169] Schnittstelle verwenden Sie die ITcPostCyclic [▶ 191] Schnittstelle.

13.7 Reihenfolgebestimmung der Ausführung in einer Task

Es können verschiedene Modulinstanzen einer Task zugewiesen werden, sodass der Benutzer einen Mechanismus benötigt, um die Reihenfolge der Ausführung in der Task festzulegen.

Sie wird unter **Sort Order** im Kontext [▶ 137] des TwinCAT Module Instance Configurator [▶ 135] konfiguriert.



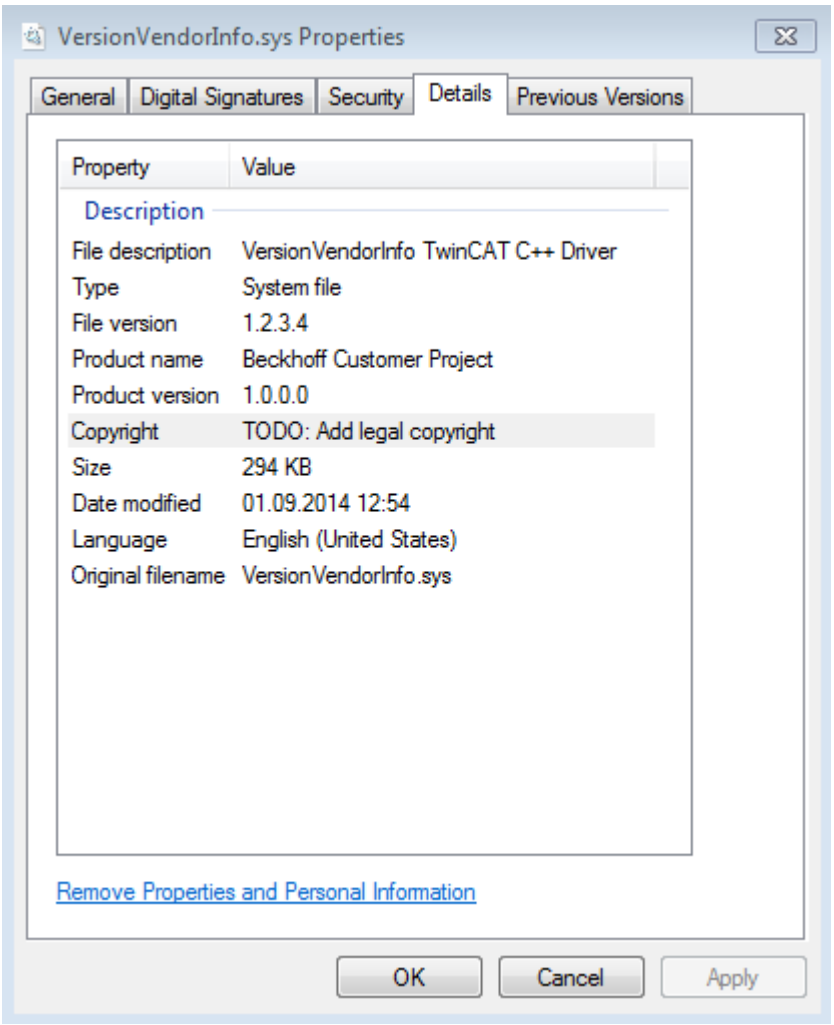
Siehe Beispiel26: Ausführungsreihenfolge in einer Task [▶ 317], wie das zu implementieren ist.

13.8 Setzen von Version/Herstellerinformationen

Windows bietet einen Mechanismus, um Hersteller- und Versionsressourcen abzufragen, die im Verlauf einer .rc Datei für die Kompilationszeit definiert sind.

Windows

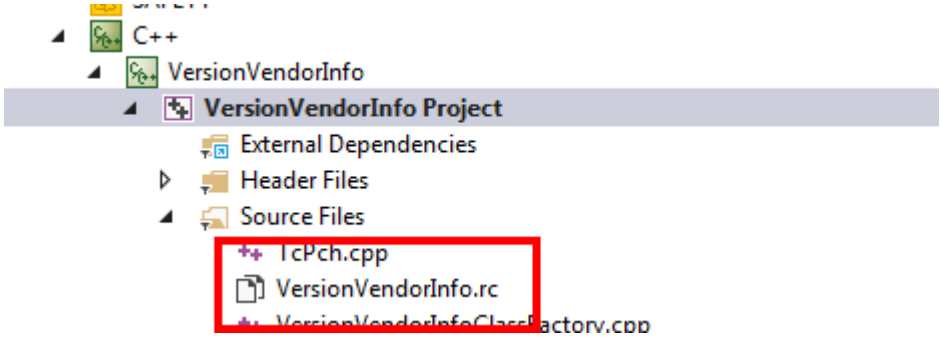
Diese sind z. B. über den Karteireiter **Details** von jeder Eigenschaften-Datei zugänglich.



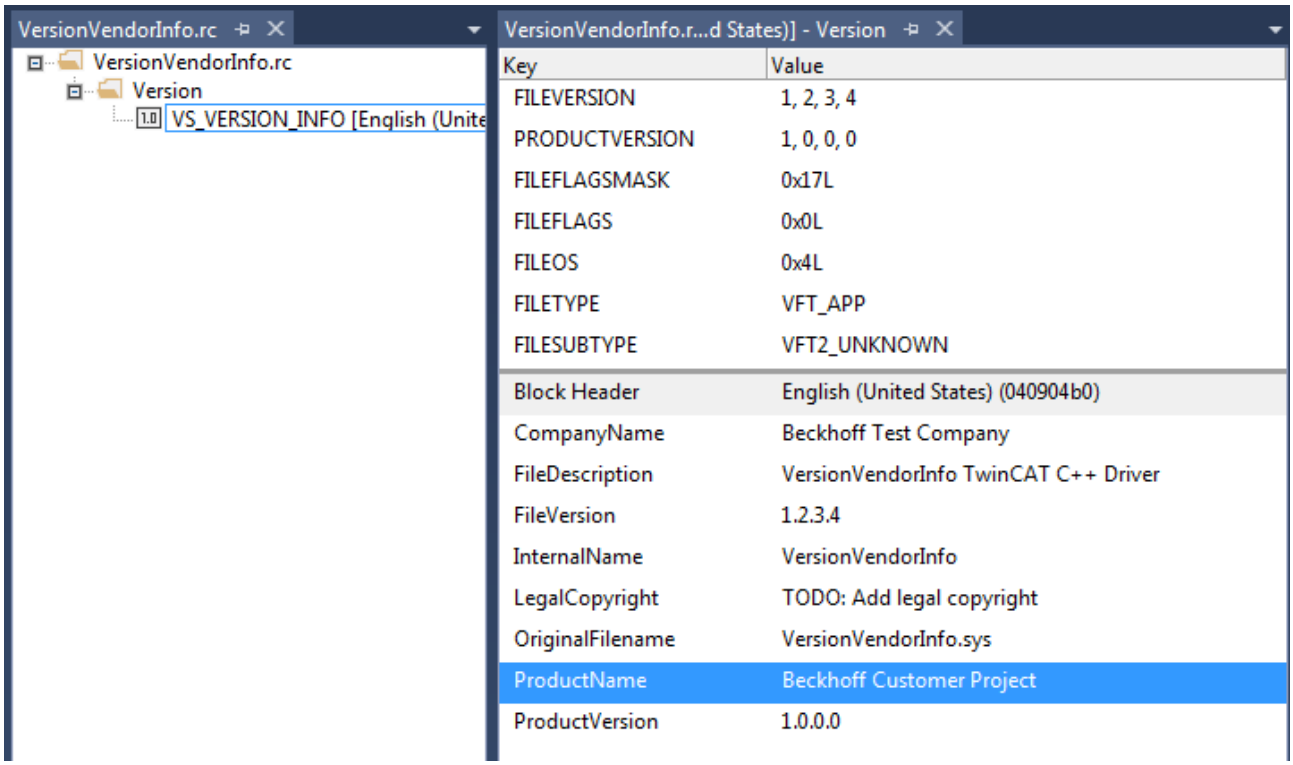
TwinCAT/BSD

Diese Informationen sind nicht in den Dateien vorhanden.

TwinCAT bietet dieses Verhalten über die bekannten Windows-Mechanismen von .rc Dateien, die im Verlauf der TwinCAT C++ Projekterstellung erzeugt werden.



Bearbeiten Sie die .rc-Datei im Source Files-Ordner mit dem Ressource Editor, um diese Eigenschaften festzulegen:



13.9 Umbenennen von TwinCAT-C++ Projekten

Das Umbenennen von TwinCAT-C++ Projekten ist nicht automatisiert möglich.

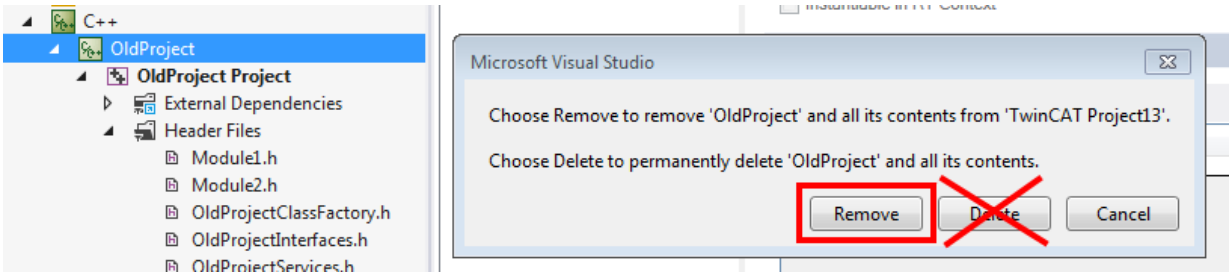
An dieser Stelle wird eine Anleitung gegeben, wie ein Projekt manuell umbenannt werden kann.

Zusammenfassend kann gesagt werden, dass das C++ Projekt zusammen mit den entsprechenden Dateien umbenannt wird.

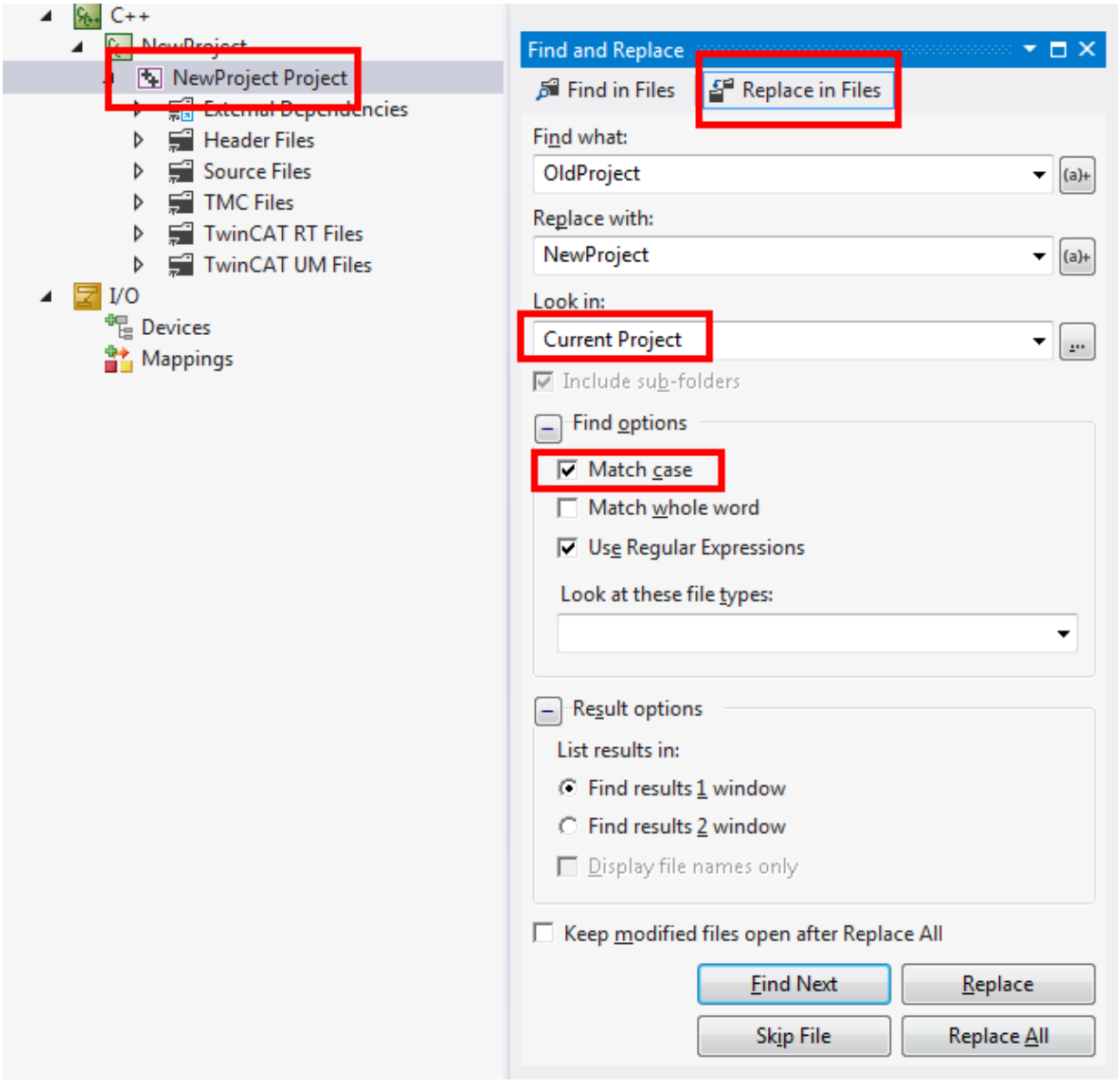
✓ Ein Projekt „OldProject“ existiert und soll in das Projekt „NewProject“ umbenannt werden.

1. Sollten TcCOM Instanzen im Projekt existieren und diese sollen inkl. ihrer Verknüpfung erhalten bleiben, verschieben Sie diese zuerst durch Drag&Drop aus dem Projekt in **System->TcCOM Objects**.

2. Entfernen Sie das alte Projekt aus der TwinCAT Solution per **Remove**.



3. Kompilate des „OldProjekt“ können gelöscht werden. Löschen Sie hierzu im „_Deployment“ die entsprechenden .sys/.pdb Dateien.
   Eine evtl. vorhandene .aps Datei können Sie ebenfalls löschen.
4. Benennen Sie das C++ Projektverzeichnis und die Projektdateien (.vcxproj, .vcxproj.filters) um. Sollte eine Versionsverwaltung eingesetzt werden, müssen Sie dieses Umbenennen über das Versionsverwaltungssystem durchführen.
5. Sollte eine .vcvproj.user Datei existieren, kontrollieren Sie den Inhalt, hier werden Einstellungen des Nutzers abgelegt. Ggf. benennen Sie die Datei ebenfalls um.
6. Öffnen Sie die TwinCAT Solution. Binden Sie das umbenannte Projekt neu ein per **Add existing Item** auf dem C++ Knoten: navigieren Sie in das umbenannte Unterverzeichnis und wählen Sie dort die .vcxproj Datei aus.
7. Benennen Sie die ClassFactory, Services und Interfaces sowie Header-/Quellcode- Dateien um in den neuen Projektnamen. Benennen Sie zusätzlich die TMC-Datei und entsprechende Dateien in den Projekt-Ordnern „TwinCAT RT Files“ und „TwinCAT UM Files“ um.
   Diese Umbenennung soll auch im Versionsverwaltungssystem abgebildet werden – falls das Versionsverwaltungssystem nicht in Visual Studio integriert ist, führen Sie diesen Schritt also wieder im Versionsverwaltungssystem aus. Ersetzen Sie alle Vorkommen im Quellcode (case-sensitive):
   Aus „OLDPROJECT“ wird „NEWPROJECT“ und aus „OldProject“ wird „NewProject".
   Nutzen Sie hierfür den **Find and Replace**-Dialog des Visual Studios, wobei das „NewProject Project“ im Solution Explorer ausgewählt sein muss.



⇨

HINWEIS

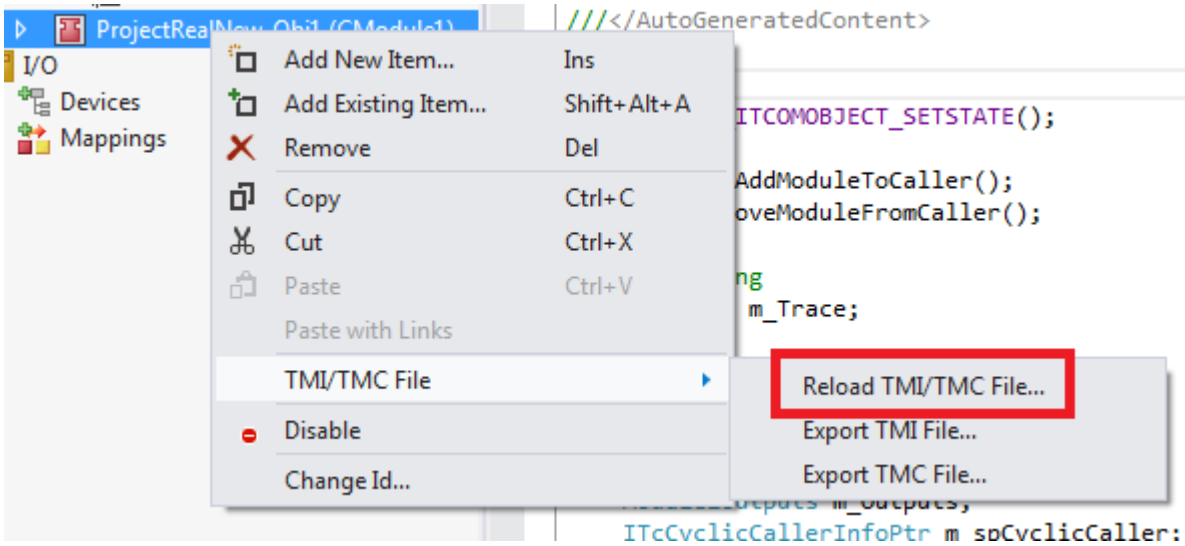
Inkorrekter Quellcode

Durch die einfache Umbenennung aller Vorkommen der Zeichenfolge kann es zu inkorrektem Quellcode kommen. Beispielsweise, wenn der Projektname innerhalb eines Methodennamens verwendet wird.

- Sollten solche Vorkommen möglich sein, führen Sie die Umbenennung einzeln aus (**Replace** statt **Replace All**).

So bauen Sie das Projekt:

1. A) Sollten Instanzen aus dem Projekt existieren, aktualisieren Sie diese. Dazu machen Sie einen Rechts-Klick auf die Instanz, wählen Sie **TMI/TMC File->Reload TMI/TMC File…** und das umbenannte, neue TMC File aus.



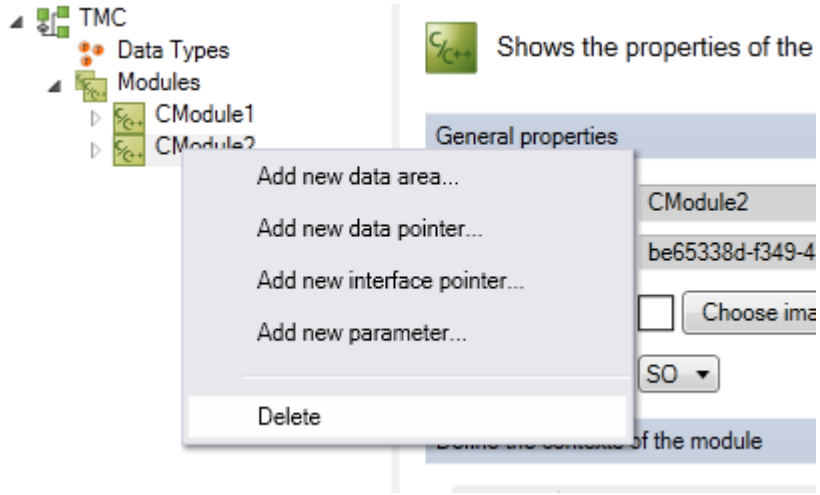
B) Alternativ führen Sie dies über **System->TcCOM Objects** und den Tab **Project Objects** durch Rechts-Klick auf die OTCID aus.

2. Verschieben Sie **System->TcCOM** in das Projekt.
3. Bereinigen Sie das / die Zielsystem(e).
   Für TwinCAT C++ Treiber: Löschen Sie die Dateien „OldProject.sys/.pdb“ im *C:\TwinCAT\3.1\Driver\AutoInstall*.
   Für TwinCAT Versioned C++ Projekte: Das Repository kann aufgeräumt werden unterhalb von *C:\TwinCAT\3.1\Repository*
4. Testen Sie das Projekt.

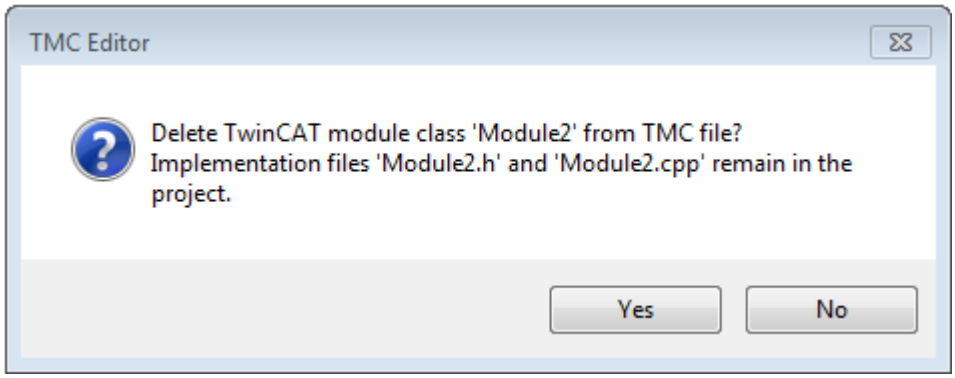
13.10 Modul löschen

Ein TwinCAT C++ Modul kann mit Hilfe des TMC Editors aus einem C++ Projekt gelöscht werden.

1. Rechtsklicken Sie auf das Modul (hier: CModule2)
2. Wählen Sie **Delete**.



3. Bestätigen Sie die Löschung über TMC .



4. Beachten Sie, dass die .cpp und .h Dateien erhalten bleiben – ggf. löschen Sie diese manuell. Löschen Sie andere betreffende Komponenten (z. B. Header- Dateien, Strukturen). Siehe Compiler-Fehlermeldungen für weitere Auskünfte.

13.11 Versionierung und Online-Change nachträglich hinzufügen

Ein existierendes C++-Treiber-Projekt mit C++-TcCOM-Modulen kann nachträglich zu einem versionierten C++-Projekt migriert werden.

Die Umstellung erfolgt in mehreren Schritten:

1. Klicken Sie auf **C++ Projekt TwinCAT Upgrade C+ Project** im Kontextmenü, dies ist die komfortabelste Möglichkeit.
   Alternativ befindet sich hier eine manuelle Anleitung:
   C++-Projekt in ein versioniertes C++ Projekt konvertieren [▶ 234].
2. Anschließend können Sie die Online-Change-Fähigkeit ergänzen, dies ist manuell zu erledigen:
   C++-Modulklasse Online-Change-fähig machen [▶ 237].

Alternativ kann es sinnvoll sein, ein neues versioniertes C++ Projekt selbst anzulegen und die Unterschiede zu übertragen.

13.11.1 C++ Projekt -> Versionierung

Diese Anleitung beschreibt, wie Sie einem TwinCAT C++ Projekt nachträglich eine Versionierung hinzufügen.

In **Fett** ist im Quellcode jeweils der zu ändernde Code hinterlegt.

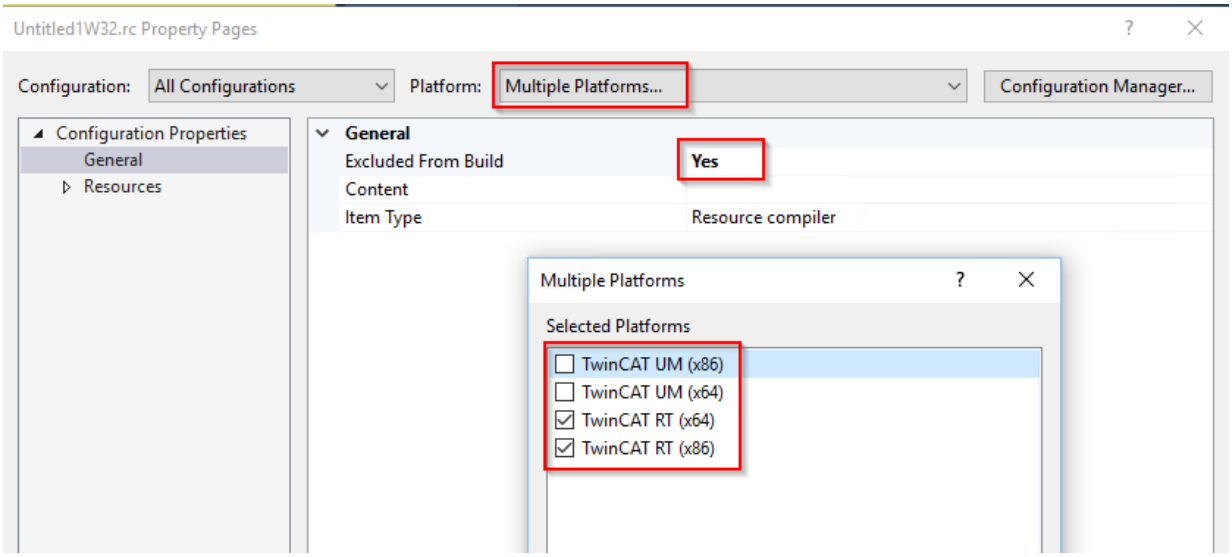
✓ C++ Projekt. Für das Beispiel wird „Untitled1“ als C++ Projektname verwendet.
Zusätzlich soll ein leeres, neues Projekt mit versioniertem C++ Projekt angelegt werden. Dieses dient als Kopiervorlage.

1. Öffnen Sie die Datei *Untitled1.vcxproj* in einem Editor.
2. Nehmen Sie folgende Ergänzung vor:

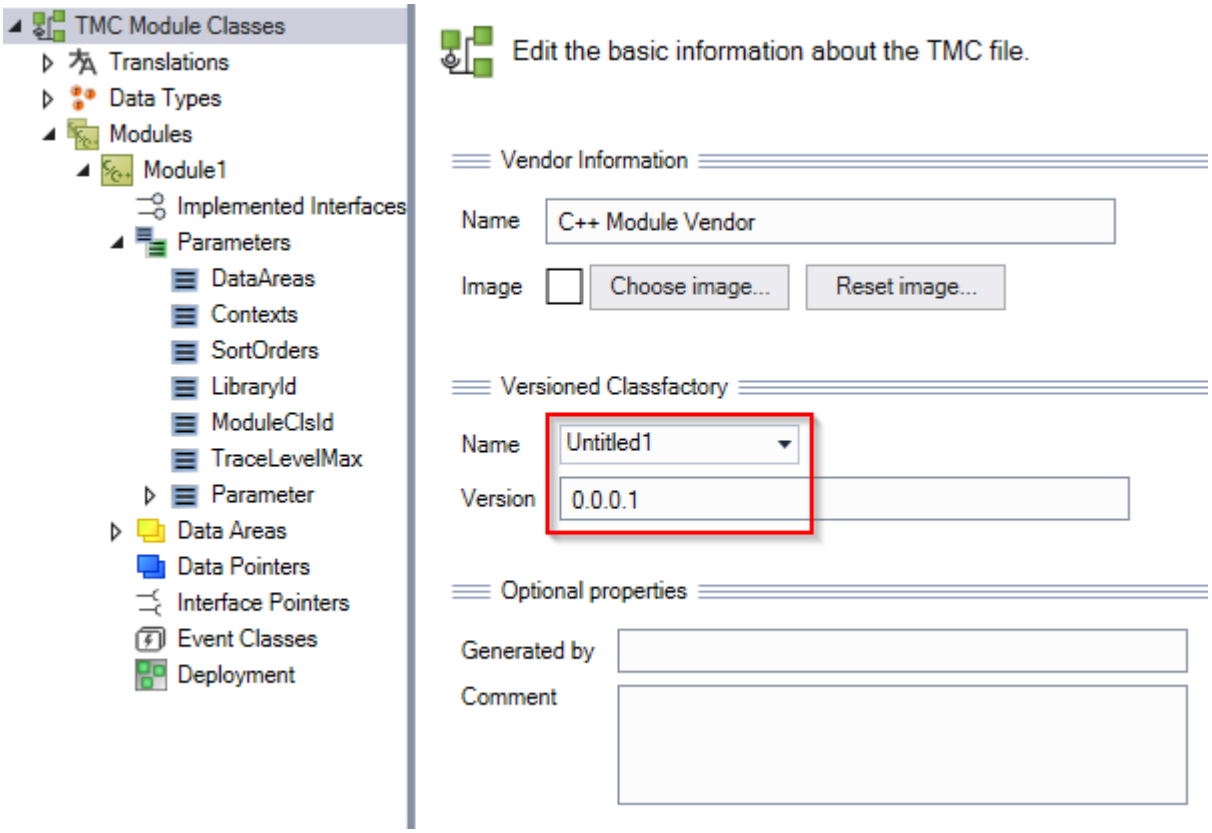
```
<PropertyGroup Label="Globals">
<ProjectGuid>{….}</ProjectGuid>
<RootNamespace>Untitled1</RootNamespace>
<Keyword>Win32Proj</Keyword>
<AutomaticRetargetPlatformVersion>true</AutomaticRetargetPlatformVersion>
</PropertyGroup>
  <PropertyGroup Label="TcGeneral">
<TcGeneralUseTmx>true</TcGeneralUseTmx>
</PropertyGroup>
<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />
```

3. Übertragen Sie aus dem **neuen** Projekt die Dateien *Untitled1.rc* und *Untitled1W32.rc* in das zu migrierende Projekt; überschreiben Sie dabei die Datei *Untitled1.rc*.
4. Öffnen Sie das Projekt.

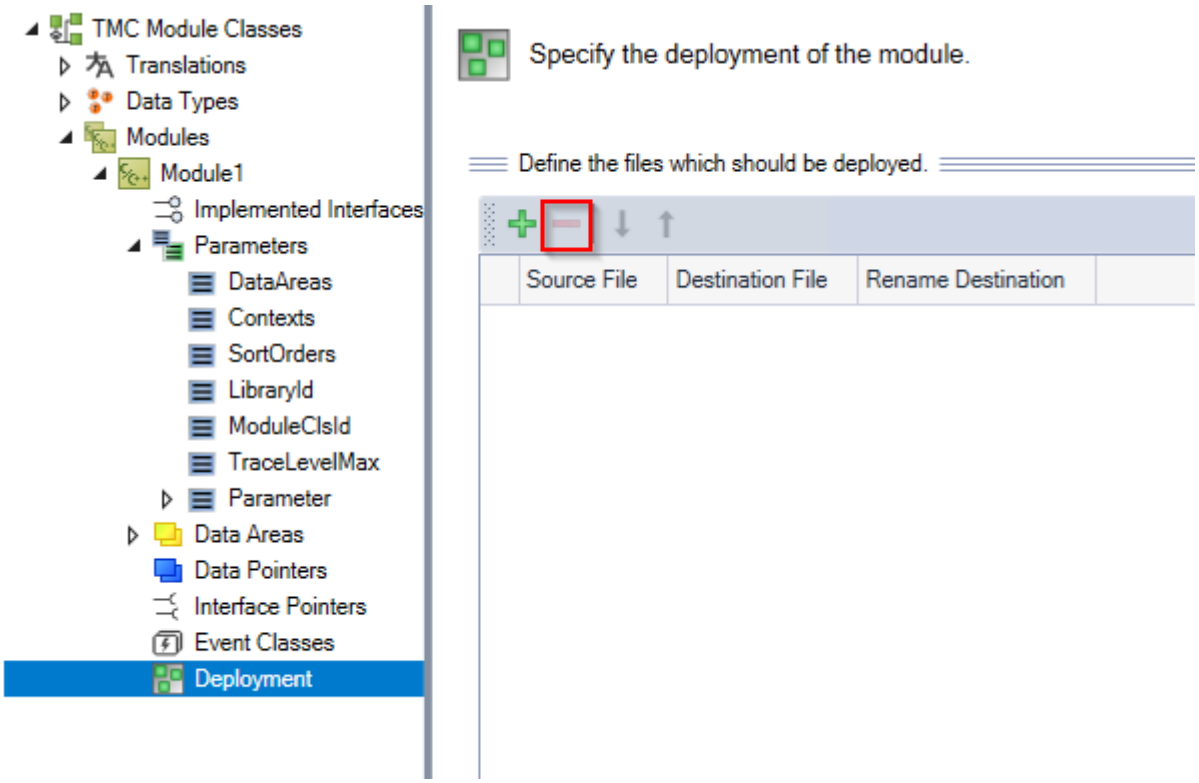
5. Wählen Sie unter **TwinCAT UM Files** -> **Kontextmenü / Add Existing Items** die Datei *Unititled1W32.rc* aus und fügen Sie sie dadurch zu dem Projekt hinzu.
6. Diese Datei muss nur für TwinCAT UM Plattformen gebaut werden, weswegen sie ansonsten ausgeschlossen werden sollte für den Build-Prozess. Dieses geschieht über Rechts-Klick und Eigenschaften:



7. Öffnen Sie den TMC Editor.
8. Setzen Sie den Namen der Classfactory sowie die Version auf „0.0.0.1“.



9. Löschen Sie unter **Deployment** alle Einträge.



10. Verändern Sie im Header *Modul1.h* DECLARE_IPERSIST**_LIB**():

```
public:
DECLARE_IUNKNOWN()
DECLARE_IPERSIST_LIB()
DECLARE_ITCOMOBJECT_LOCKOP()
```

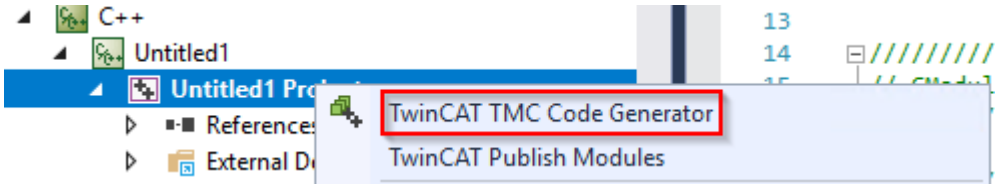
11. Ergänzen Sie im Quellcode *Modul1.cpp*:

```
#pragma hdrstop
#include "Module1.h"
#include "Untitled1Version.h"

#ifdef _DEBUG
```

12. Ergänzen Sie im Quellcode *Modul1.cpp*:

```
END_INTERFACE_MAP()

IMPLEMENT_IPERSIST_LIB(CModule1, VID_Untitled1,
CID_Untitled1CModule1)
IMPLEMENT_ITCOMOBJECT(CModule1)
IMPLEMENT_ITCOMOBJECT_SETSTATE_LOCKOP2(CModule1)
```

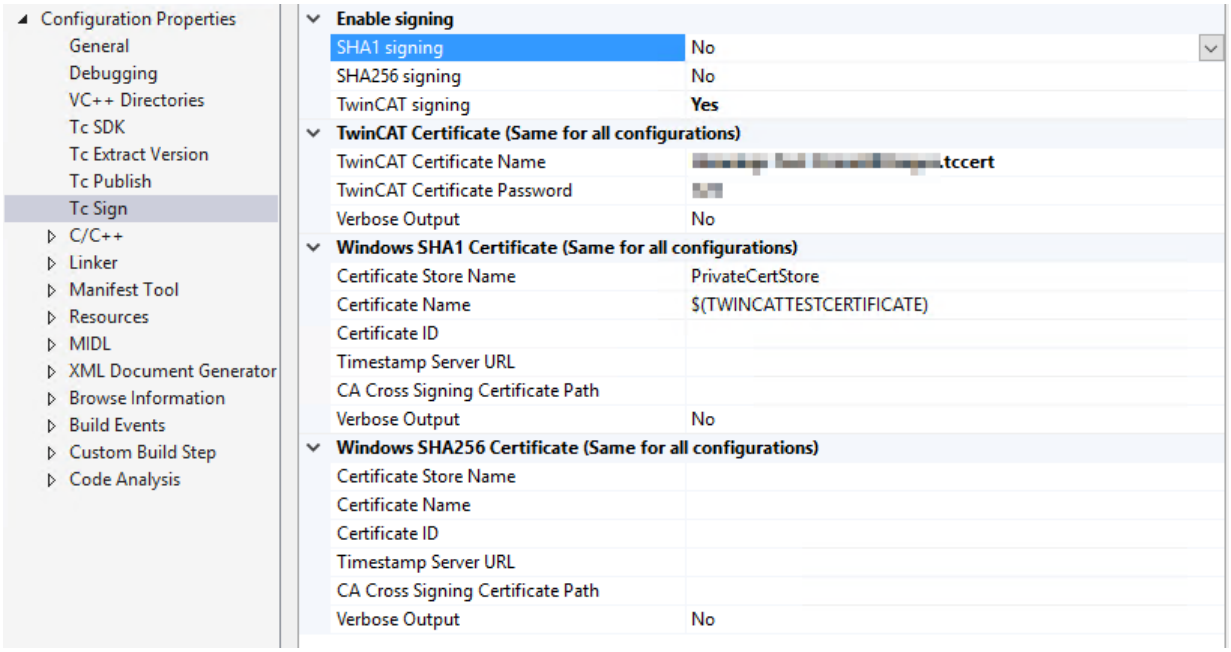
13. Rufen Sie den Code Generator auf.



14. Verändern Sie in der Datei *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. Stellen Sie in den Projekteigenschaften im Tab Tc Sign [▶ 151] die Signierung um. Schalten Sie dafür **SHA1 signing** aus und **TwinCAT signing** an; stellen Sie gleichzeitig TwinCAT-Nutzerzertifikat und Passwort bereit.



16. Triggern Sie den **Rebuild** des Projektes.

⇨ Das Ergebnis ist ein C++ Projekt, welches eine Versionierung unterstützt.

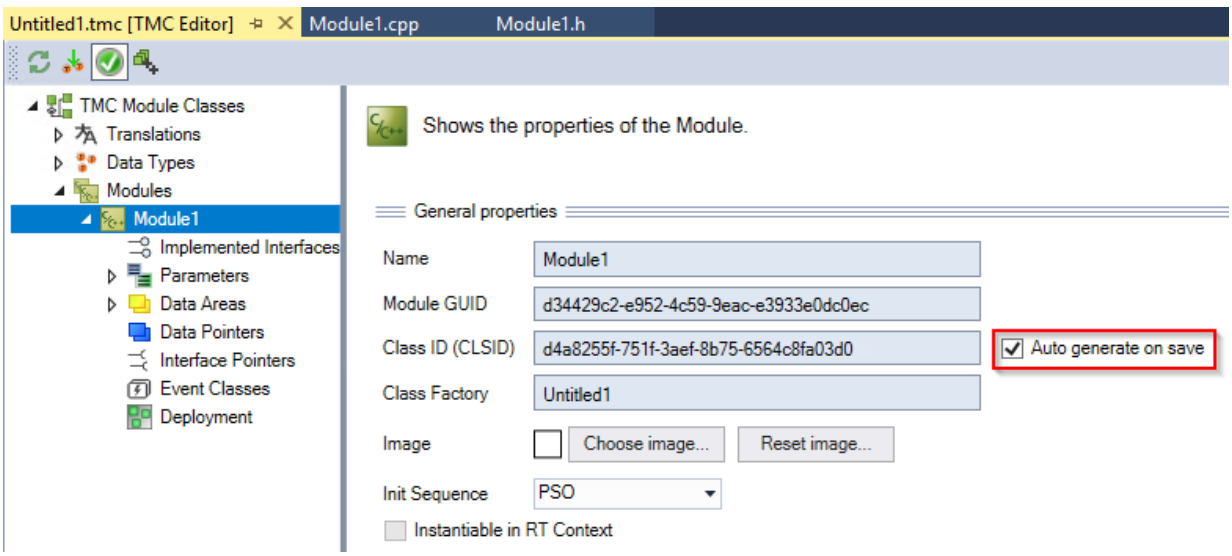
Falls ein Modul Online-Change fähig gemacht werden soll, kann dieses durch die folgende Anleitung [▶ 237] erreicht werden.

13.11.2 C++ Modul -> OnlineChange

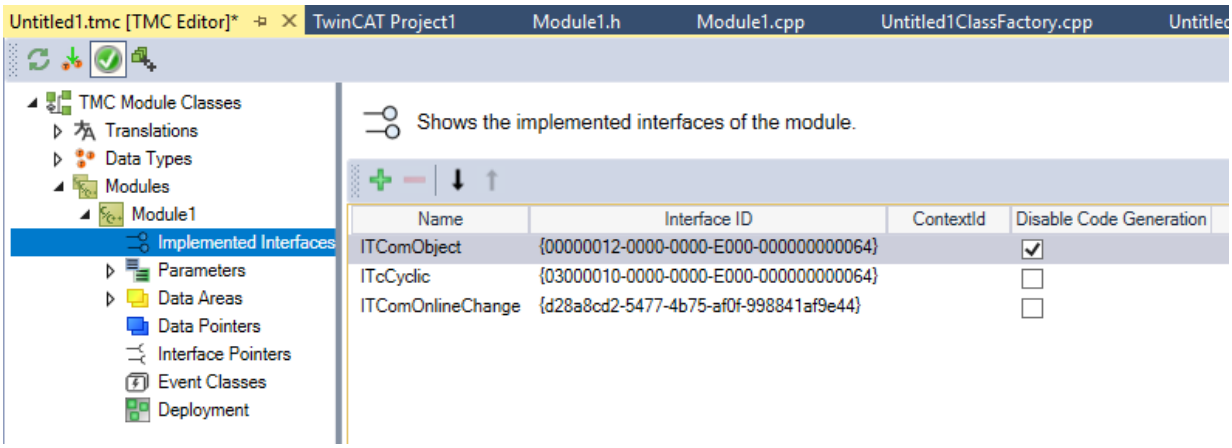
Diese Anleitung beschreibt, wie Sie in einem versionierten TwinCAT C++ Projekt ein Modul nachträglich OnlineChange-fähig machen.

✓ Versioniertes C++ Projekt mit C++ Modul, welches noch nicht Online-Change fähig ist. Für dieses Beispiel wird von einem Modul „Module1“ ausgegangen. Zusätzlich kann ein leeres, neues Projekt mit einem Online-Change fähigen Modul erzeugt werden, aus welchem die Änderungen einfacher übernommen werden können.

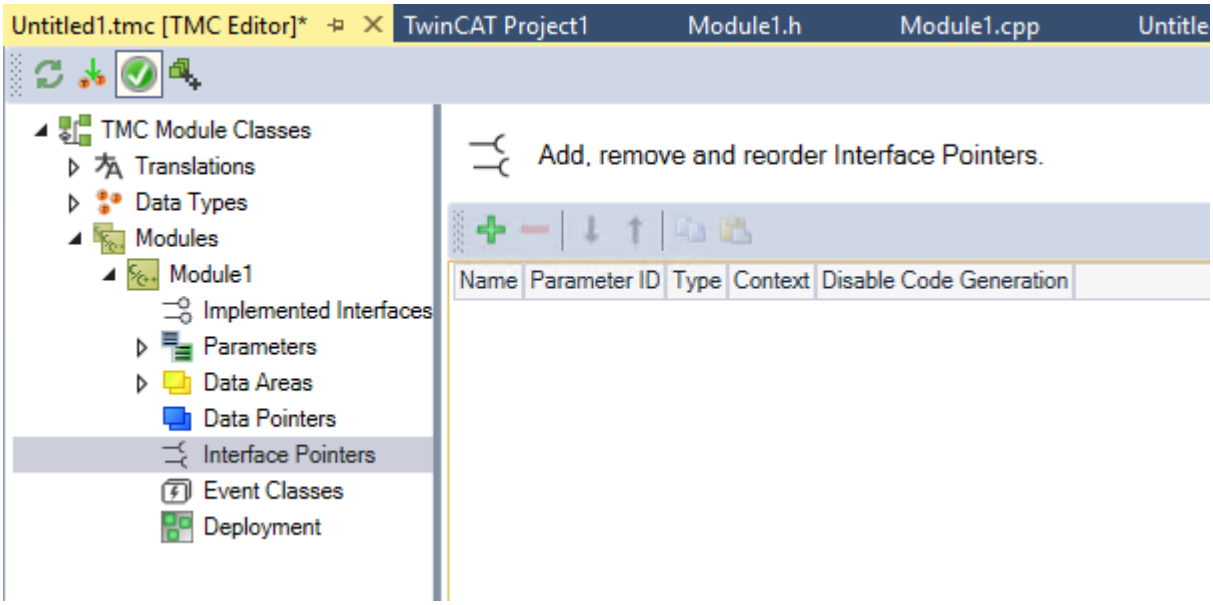
1. Öffnen Sie das Projekt und den TMC Editor.
2. Setzen Sie die **Auto generate on save**-Option für das Modul, sodass die ClassID automatisch geändert wird.



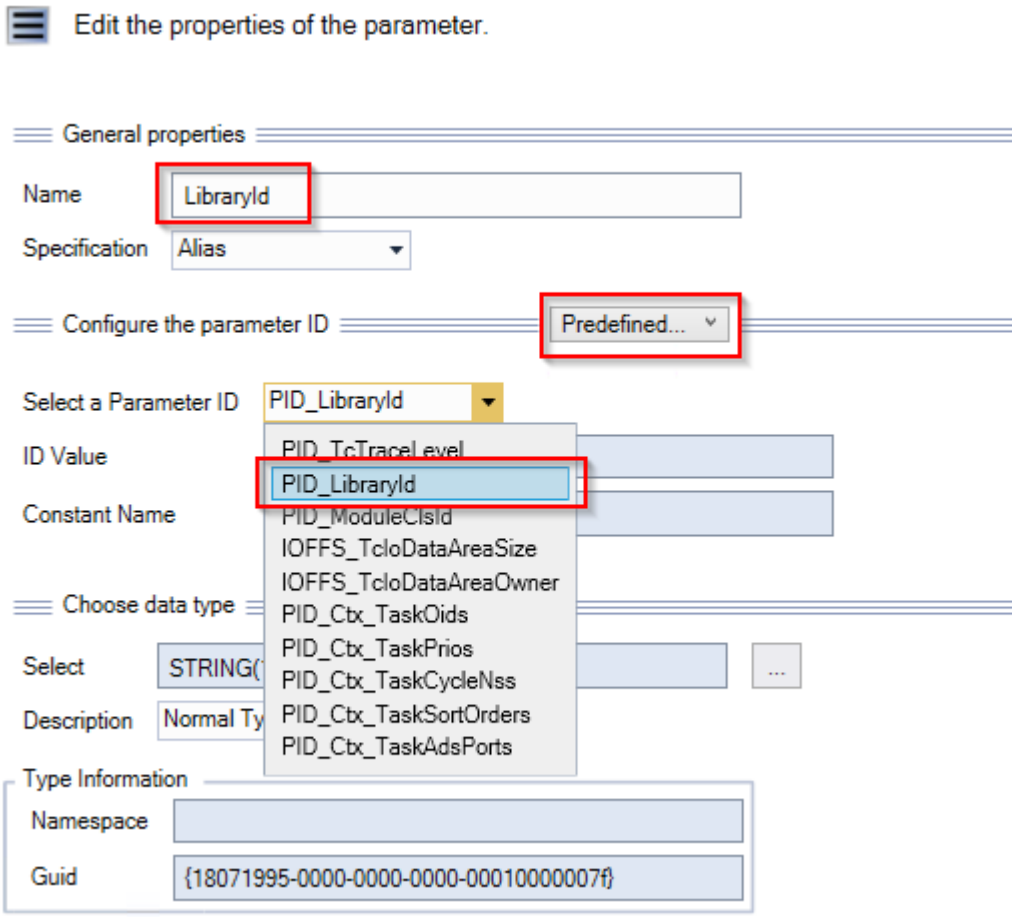
3. Löschen Sie unter **Implemented Interfaces** die Interfaces ITcADI und ITcWatchsource und fügen Sie ITComOnlineChange hinzu.



4. Löschen Sie unter **Interface Pointer** den CyclicCaller.

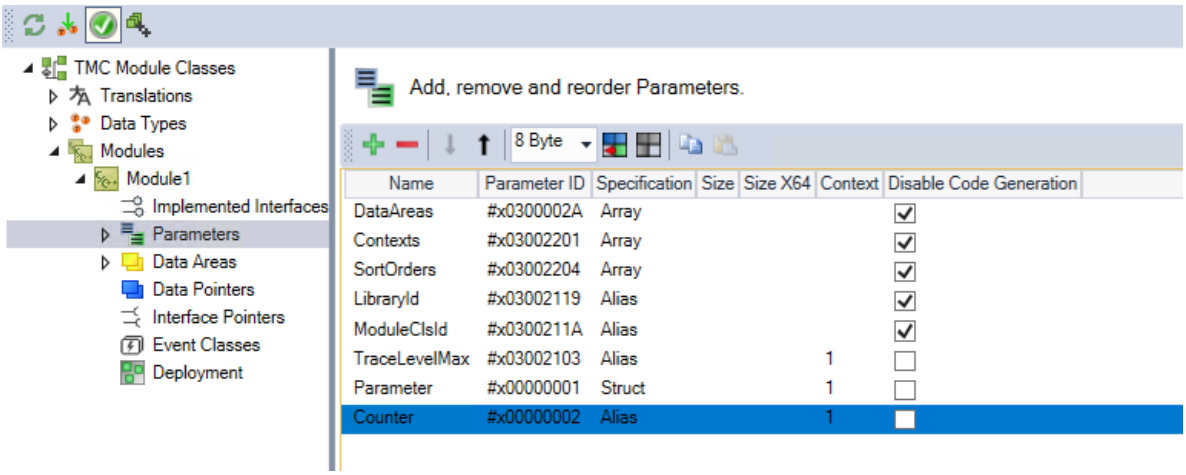


5. Unter **Parameters** müssen einige vordefinierte Parameter hinzufügt werden.
   Drücken Sie hierfür unter **Parameters** auf **+** und wählen Sie jeweils **Predefined** aus, wodurch die vordefinierten ParameterIDs ausgewählt werden können:

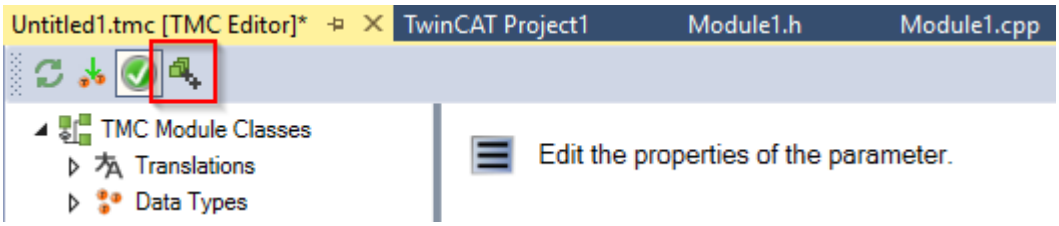


Legen Sie folgende Parameter mit den gegebenen Namen an, jeweils ohne CodeGenerierung:
„PID_LibraryID“ mit Namen „LibraryID“
„PID_ModuleClsId“ mit Namen „ModuleClsId“
„PID_Ctx_TaskSortOrders“ mit Namen „SortOrders“
„PID_Ctx_TaskOids“ mit Namen „Contexts“
„IOFFS_TcIoDataAreaSize“ mit Namen „DataAreas“

⇨ Das Ergebnis in der Übersicht:



6. Starten Sie die Code Generierung.



7. In dem Header des Modules „Module1.h“ müssen einige Änderungen vorgenommen werden. Als erstes löschen Sie die Deklarationen der nicht mehr benötigten Interfaces sowie der zugehörigen Maps an zwei Stellen.

```
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. Legen Sie im Header eine neue Member-Variable an:

```
///</AutoGeneratedContent>
ITcADIPtr m_spADI;
// TODO: Custom variable
```

9. Im Quellcode des Moduls „Module1.cpp“ müssen einige Änderungen vorgenommen werden. Als erstes löschen Sie die Implementierungen der nicht mehr benötigten Interfaces an zwei Stellen.

```
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. Löschen Sie die Implementierung der zu den gelöschten Interfaces gehörigen Maps:

```
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. In der Statemachine muss in der Transition P->S der m_spAPI Pointer bezogen werden:

```
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. Ergänzen Sie in der Statemachine in der Transition S->O folgendes, die Aufrufe zu AddModuleToCaller bzw. RemoveModuleFromCaller fallen weg.

```
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. Ergänzen Sie in der Statemachine in der Transition O->S folgendes, der Aufruf RemoveModuleFromCaller fällt weg:

```
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. Die Methoden AddModuleToCaller sowie RemoveModuleFromCaller werden nicht benötigt und können gelöscht werden.

15. Ändern Sie die Zugriffe auf die DataAreas:

```
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. Implementieren Sie das ITcOnlineChange. Durch die vorherige CodeGenerierung sind die Funktionen angelegt, dürfen aber nicht NOTIMPL zurückgeben.

```
///<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. Starten Sie den TMC Code Generator nochmals, um den Code für die Initialisierung der Data-Area-Pointer zu erzeugen.

13.12 Initialisierung von TMC-Membervariablen

Alle Membervariablen eines TcCOM Moduls müssen initialisiert werden. Der TMC Code Generator unterstützt dies mit:

```
///<AutoGeneratedContent id="MemberInitialization">
```

Wird durch den TMC Code Generator ersetzt durch:

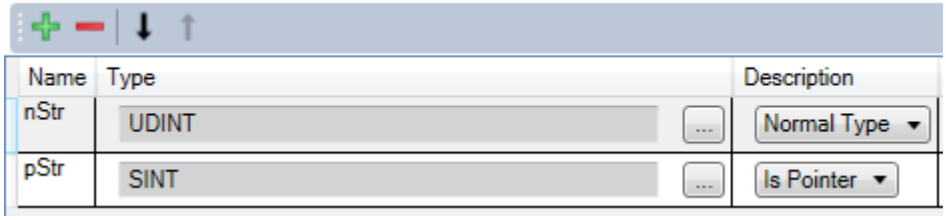
```
 ///<AutoGeneratedContent id="MemberInitialization">
m_TraceLevelMax = tlAlways;
memset(&m_Parameter, 0, sizeof(m_Parameter));
memset(&m_Inputs, 0, sizeof(m_Inputs));
memset(&m_Outputs, 0, sizeof(m_Outputs));
///</AutoGeneratedContent>
```

Die mit dem TwinCAT C++ Assistenten vor TwinCAT 3.1 Build 4018 generierten Projekte verwenden diese Eigenschaft nicht, können aber einfach angepasst werden, indem diese Zeile im entsprechenden Code (z. B. Konstruktor) hinzugefügt wird:

```
///<AutoGeneratedContent id="MemberInitialization">
```

13.13 SPS-Zeichenketten als Methodenparameter verwenden

Um eine Zeichenkette von SPS nach C++ als Methodenparameter zu übergeben, verwenden Sie einen Zeiger mit Längeninformation beim Deklarieren der Methode in TMC:



Eine solche Methode kann mittels Implementierung einer Methode innerhalb des Wrapper-Funktionsbausteins aufgerufen werden:

```
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
```

Der Grund ist der unterschiedliche Umgang mit Methodenparametern in beiden Welten:

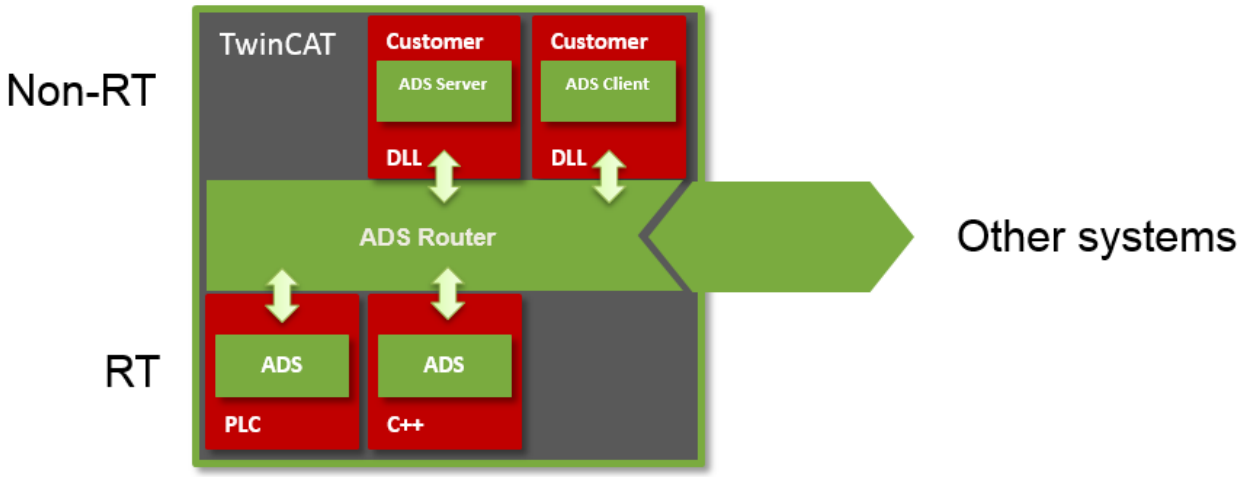
- SPS: Verwendet den Werteaufruf (call by value) für STRING(nn) Datentypen.

- TwinCAT C++ (TMC): Verwendet den Verweisaufruf (call by reference).

13.14 Bibliotheken von Drittanbietern

In Kernelmode vorliegender C/C++ Code kann nicht mit Bibliotheken von Drittanbietern, die für die Ausführung im Usermode entwickelt wurden, verknüpft werden und diese ausführen. Somit besteht keine Möglichkeit, eine beliebige DLL direkt in TwinCAT C++ Modulen zu verwenden.

Stattdessen können Sie die Verbindung von TwinCAT 3 Echtzeitumgebung über ADS-Kommunikation realisieren. Sie können eine User-Modus Anwendung implementieren, die die Bibliothek von Drittanbietern verwendet, die TwinCAT Funktionalitäten über ADS zur Verfügung stellt.



Dieses Vorgehen einer ADS Komponente im Usermode kann sowohl als Client (d.h. die DLL übermittelt bei Bedarf Daten an die TwinCAT Echtzeit) wie auch als Server (d. h. die TwinCAT Echtzeit holt bei Bedarf Daten aus dem Usermode) geschehen.

Eine solche ADS Komponente im Usermode kann auf die gleiche Weise auch aus der SPS genutzt werden. Zusätzlich kann ADS über Gerätegrenzen hinweg kommunizieren.

Die folgenden Beispiele erläutern die Verwendung von ADS in C++ Modulen:

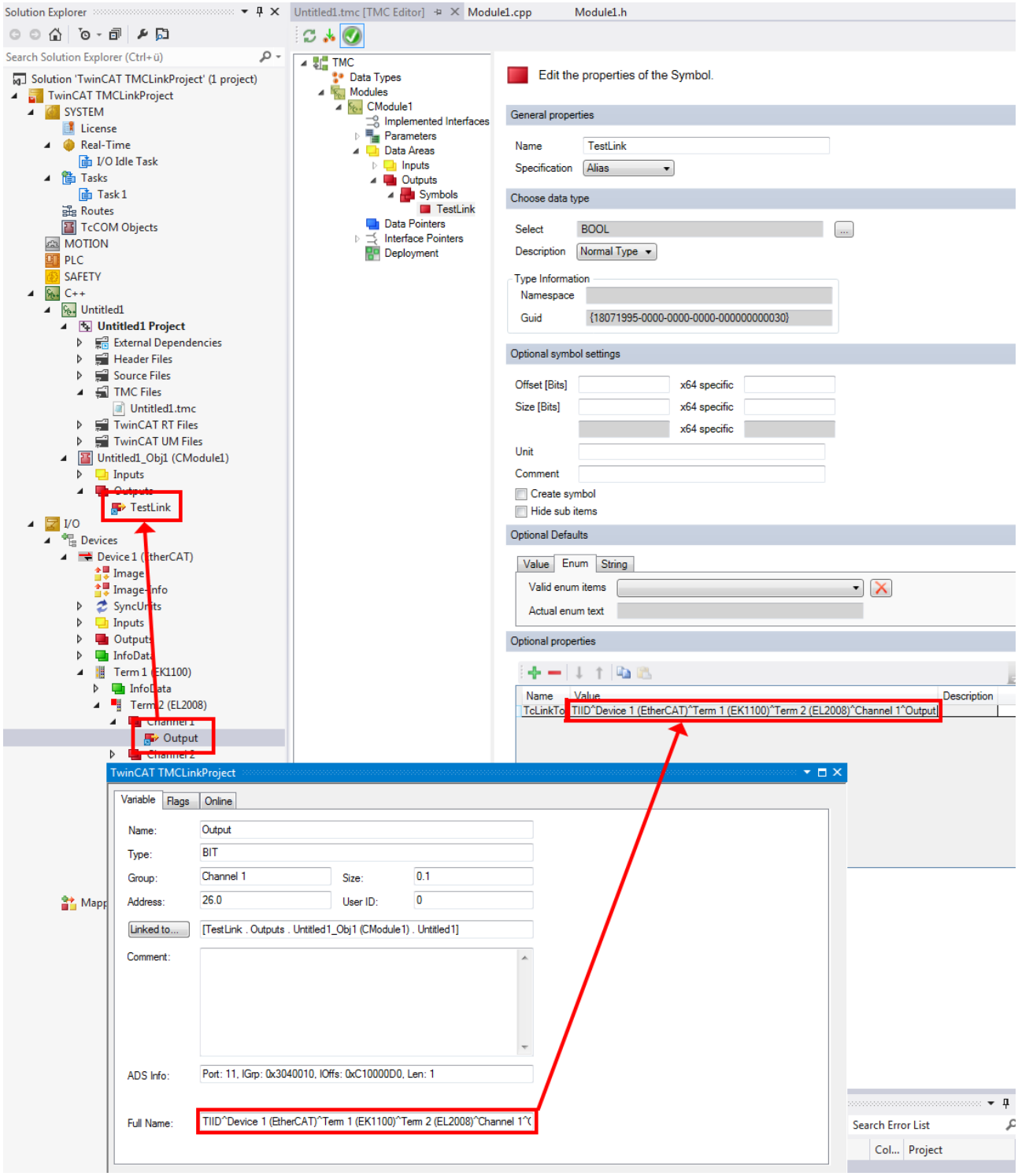
Beispiel03: C++ als ADS Server [▶ 256]

Beispiel07: Empfang von ADS Notifications [▶ 270]

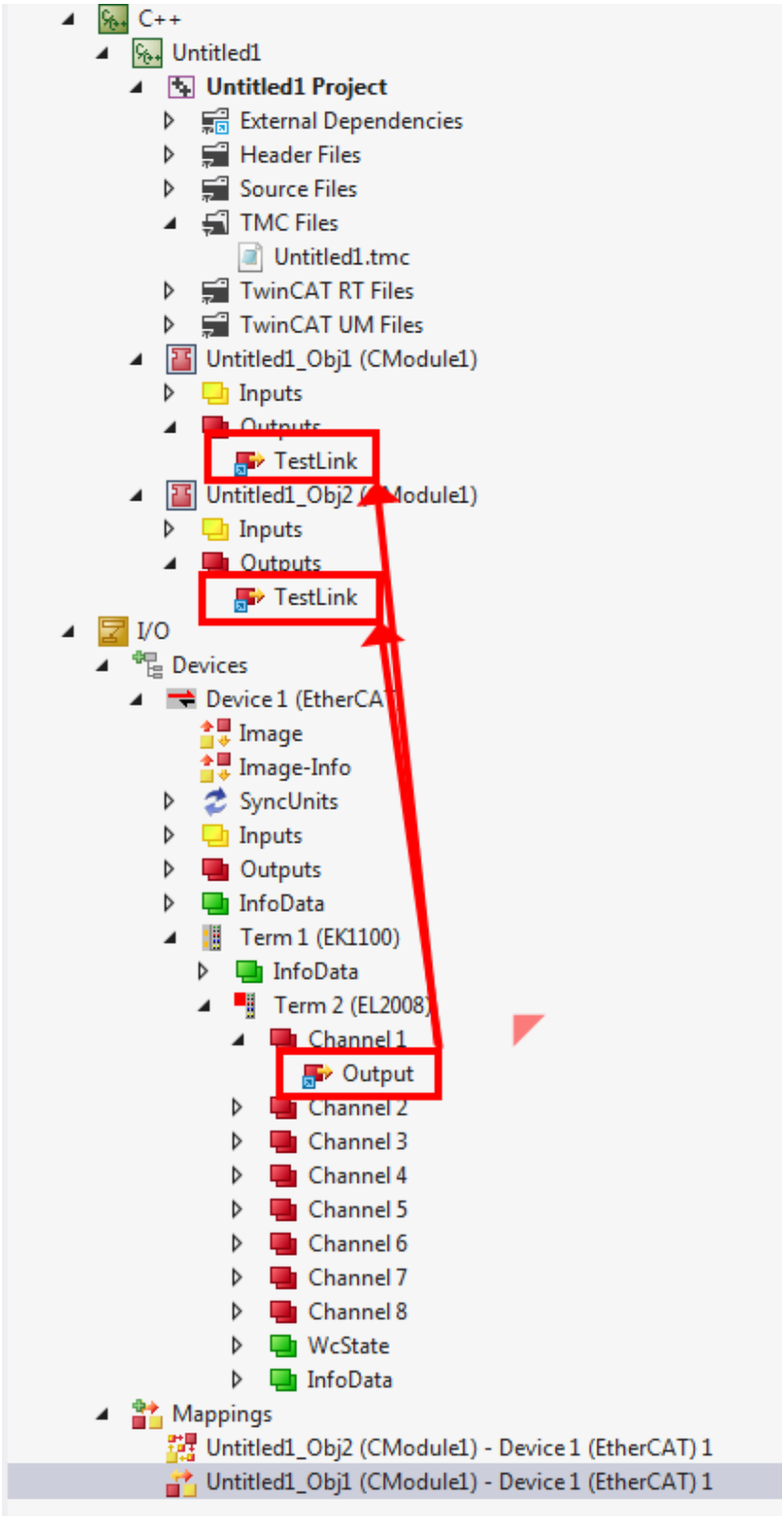
Beispiel08: Anbieten von ADS-RPC [▶ 271]

13.15 Verknüpfungen mittels TMC Editor (TcLinkTo)

Analog zu der SPS kann auch in TwinCAT C++ zum Zeitpunkt des Kodierens eine Verknüpfung z. B. zu der Hardware vorgegeben werden.
Dieses geschieht im TMC Editor an dem Symbol, welches verlinkt werden soll. Es wird ein Property **TcLinkTo** mit dem Value des Zieles angegeben. Im folgenden Screenshot wird dieses verdeutlicht:



Beachten Sie, dass eine solche Anweisung für alle Instanzen des Moduls gilt:



13.16 Online Change per ADS

Der Online Change [▶ 155] wird im Normalfall aus dem XAE Engineering vorgenommen. Es ist jedoch auch möglich, per ADS und damit in eigenen Programmen den Online Change durchzuführen:

- ADS WRITE
- AmsPort: 11
- IdxGrp: ObjectID
- IdxOffset: 16#03002119 (PID_LibraryId)

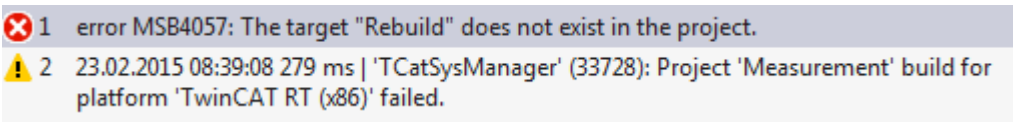
- Daten: Library Id (“<vendor>|<library>|<version>”)

14 Fehlersuche

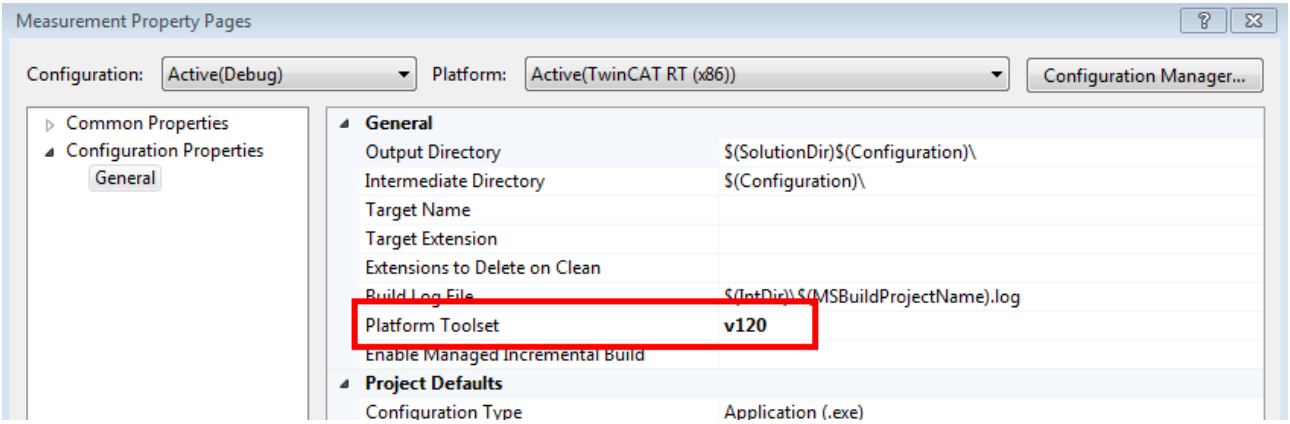
Liste von Fallstricken und Pannen beim Umgang mit TwinCAT C++-Modulen.

14.1 Build - „The target ... does not exist in the project“

Insbesondere bei der Übertragung einer TwinCAT Solution von einer Maschine auf die andere gibt Visual Studio möglicherweise Fehlermeldungen aus, gemäß derer alle Ziele (wie Build, Rebuild, Clean) nicht im Projekt existieren.

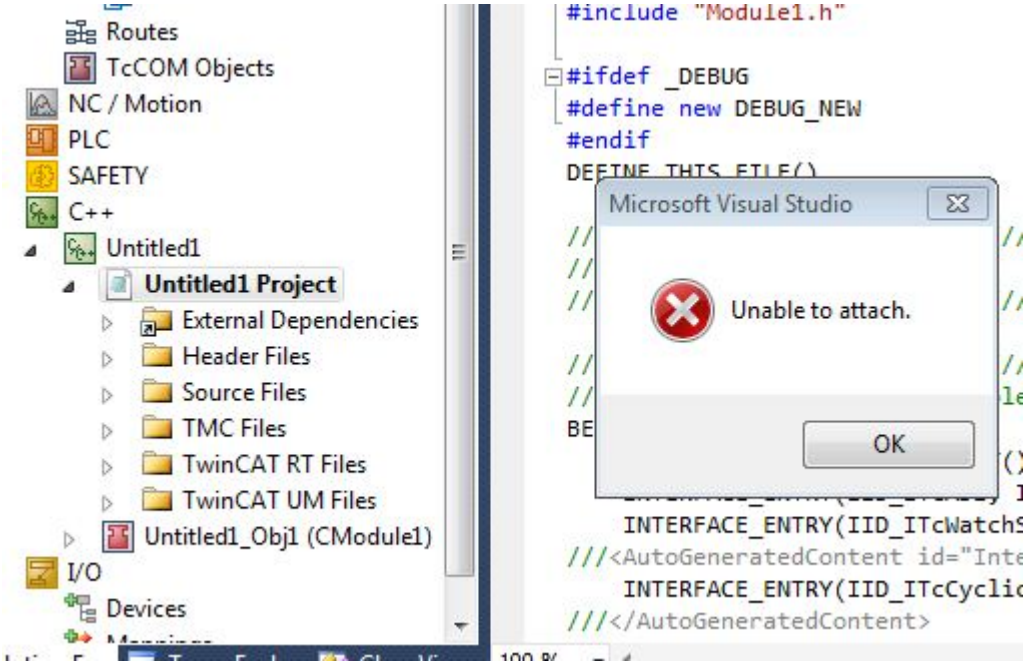


Überprüfen Sie die Konfiguration von „platform toolset“ des C++ Projekts. Sie muss gegebenenfalls neu konfiguriert werden, wenn Solutions von einer zur anderen Visual Studio Version migrieren:

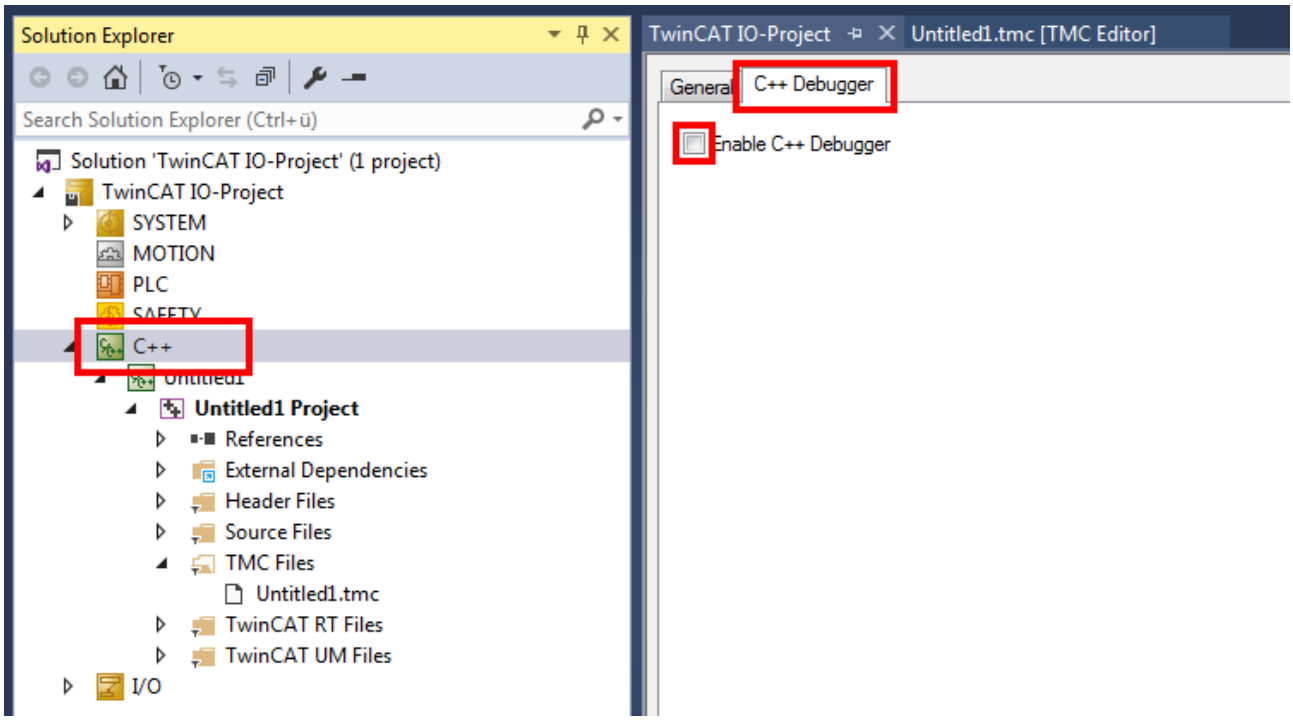


14.2 Debug - „Unable to attach“

Wenn diese Fehlermeldung beim Starten des Debuggers zum Debugging eines TwinCAT C++ Projekts erscheint, fehlt ein Konfigurationsschritt:

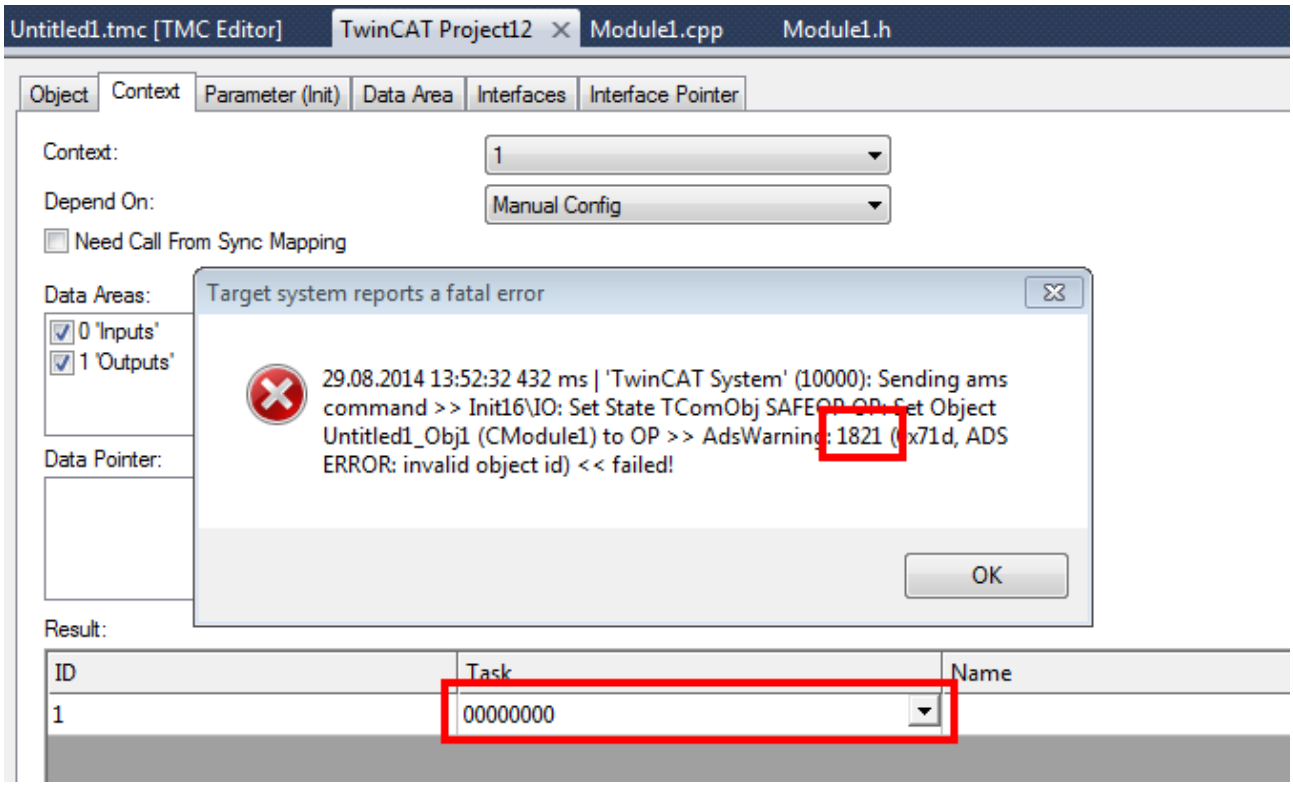


In diesem Fall navigieren Sie zu **System** -> **Real-Time**, wählen Sie die Registerkarte **C++ Debugger** und aktivieren Sie die Option **Enable C++ Debugger**.



14.3 Activation – „invalid object id“ (1821/0x71d)

Wenn im Verlauf des Starts der ADS Return Code 1821 / 0x71d berichtet wird, überprüfen Sie den Kontext der Modulinstanz wie in Schnellstart [▶ 75] beschrieben.



14.4 Verwendung von C++ Klassen in TwinCAT C++ Modulen

Beim Hinzufügen von (nicht TwinCAT) C++ Klassen bei der Verwendung vom Visual Studio-Kontextmenü **Add->Class...** meldet der Compiler/Linker:

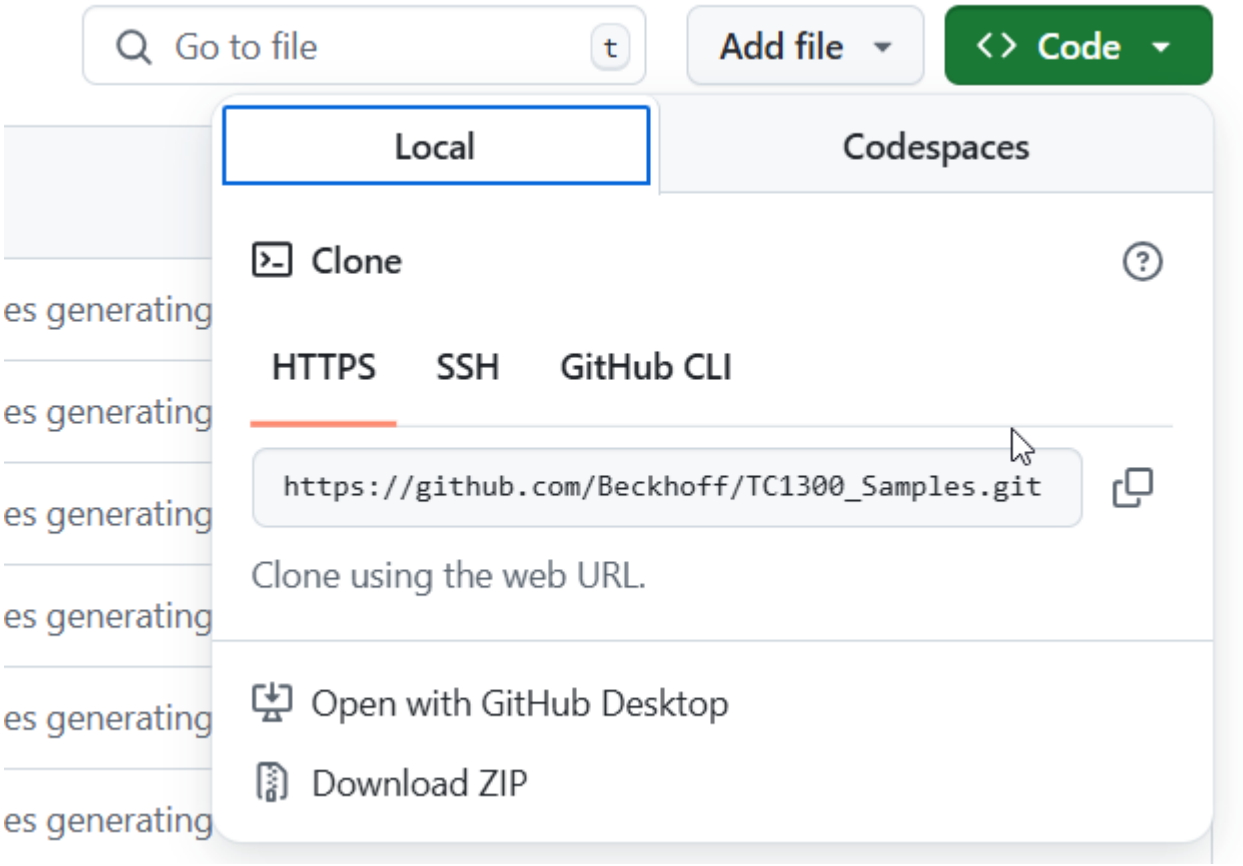
```
Error 4 error C1010: unexpected end of file while looking for precompiled
header. Did you forget to add '#include ""' to your source?
```

Fügen Sie die folgenden Zeilen am Anfang Ihrer erzeugten Klassendatei ein:

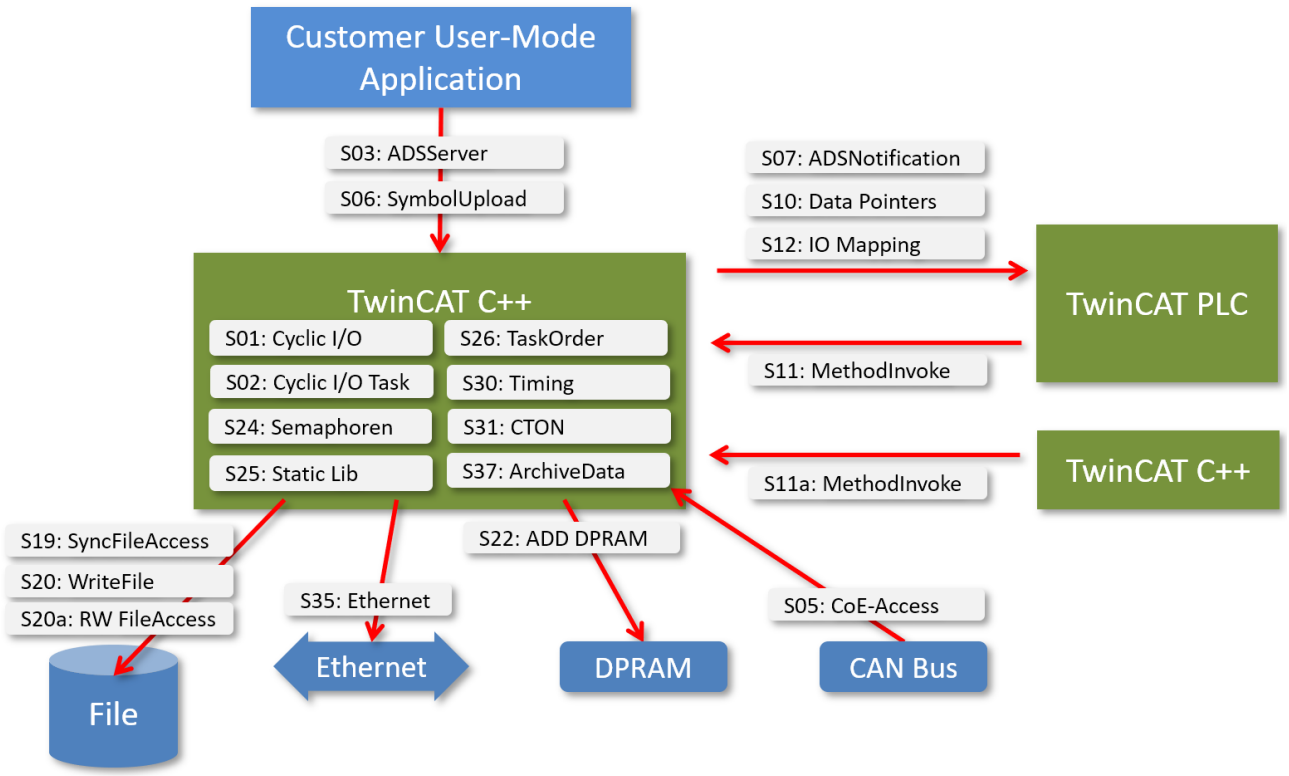
```
#include "TcPch.h"
#pragma hdrstop
```

15 C++-Beispiele

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://github.com/Beckhoff/TC1300_Samples. Sie haben dort die Möglichkeit, das Repository zu clonen oder ein ZIP-File mit den Samples herunterzuladen.



Dieses Bild zeigt einen Überblick in grafischer Form und legt dabei den Schwerpunkt auf die Interaktionsmöglichkeiten eines C++-Moduls.



Darüber hinaus ist hier eine Tabelle mit kurzen Beschreibungen der Beispiele:

Nummer	Titel	Beschreibung
01	Beispiel01: Zyklisch mit IO-Modul [▶ 254]	Dieser Artikel beschreibt die Implementierung eines TwinCAT 3 C++-Moduls, das ein mit physikalischem IO gemapptes IO-Modul verwendet. Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++-Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang „Value“ im Datenbereich zuweist. Der Datenbereich kann dem physikalischen IO oder einem anderen logischen Eingang einer anderen Modulinstanz zugewiesen werden.
02	Beispiel02: Zyklisch mit IO Task [▶ 255]	Beschreibt die Flexibilität von C++-Code bei der Arbeit mit IOs, welche an der Task konfiguriert sind. Dank dieser Herangehensweise kann ein abschließend kompiliertes C++-Modul weit flexibler auf verschiedene, mit der IO Task verbundene IOs einwirken. Eine Anwendung könnte darin bestehen, zyklische analoge Eingangskanäle zu überprüfen, wobei die Anzahl Eingangskanäle von einem Projekt zum anderen unterschiedlich sein kann.
03	Beispiel03: ADS Server Client [▶ 256]	Beschreibt den Entwurf und die Implementierung einer eigenen ADS-Schnittstelle in einem C++-Modul. Das Beispiel enthält zwei Teile: • In TwinCAT 3 C++ implementierter ADS Server mit benutzerspezifischer ADS-Schnittstelle, in C# implementierte ADS Client UI, die benutzerspezifische ADS-Meldungen an den ADS-Server sendet.
05	Beispiel05: CoE Zugriff über ADS [▶ 264]	Zeigt, wie über ADS auf CoE Register von EtherCAT-Geräten zugegriffen werden kann.
06	Beispiel06: ADS C#-Client lädt ADS-Symbole hoch [▶ 266]	Zeigt, wie über die ADS-Schnittstelle auf Symbole in einem ADS Server zugegriffen werden kann. C# ADS Client tritt in Verbindung mit einem in SPS/ C++ / Matlab® implementierten Modul. Hochladen der verfügbaren Symbolinformation und Lese-/Schreiben-Abonnieren für Prozesswerte.
07	Beispiel07: Empfang von ADS Notifications [▶ 270]	Beschreibt die Implementierung eines TwinCAT 3 C++-Moduls, das ADS Notifications bezüglich Datenänderungen auf anderen Modulen empfängt.
08	Beispiel08: Anbieten von ADS-RPC [▶ 271]	Beschreibt die Implementierung von Methoden, welche per ADS über die Task aufrufbar sind.
10	Beispiel10: Modulkommunikation: Verwendung von Datenzeigern [▶ 274]	Beschreibt die Interaktion zwischen zwei C++-Modulen mit einem direkten Zeiger (DataPointer). Die beiden Module müssen auf demselben CPU-Kern im selben Echtzeitkontext ausgeführt werden.
11	Beispiel11: Modulkommunikation: Methodenaufruf SPS-Modul nach C++-Modul [▶ 275]	Dieses Beispiel beinhaltet zwei Teile: • Ein C++-Modul, das als Zustandsmaschine fungiert, die eine Schnittstelle mit Methoden zum Starten/Stoppen, aber auch zum Setzen/Erhalten der Zustandsmaschine zur Verfügung stellt. Zweites SPS-Modul um mit erstem Modul zu interagieren, indem Methoden vom C++-Modul aufgerufen werden.

Nummer	Titel	Beschreibung
11a	Beispiel11a: Modulkommunikation: Methodenaufruf C++-Modul nach C++-Modul [▶ 302]	Dieses Beispiel beinhaltet zwei Klassen in einem Treiber (kann auch zwischen zwei Treibern gemacht werden). • Ein Modul, das eine Berechnungsmethode bereitstellt. Der Zugriff ist durch eine CriticalSection geschützt. Zweites Modul, das als Aufrufer agiert, um die Methode im anderen Modul zu verwenden.
12	Beispiel12: Modulkommunikation: Verwendet IO Mapping [▶ 303]	Beschreibt, wie zwei Module über das Mapping von Symbolen des Datenbereichs verschiedener Module untereinander interagieren können. Die beiden Module können auf demselben oder auf verschiedenen CPU-Kernen ausgeführt werden.
13	Beispiel13: Modulkommunikation: Methodenaufruf C++-Modul nach SPS-Modul [▶ 304]	Beschreibt, wie ein TwinCAT 3 C++-Modul per TcCOM Interface Methoden eines Funktionsbausteins der SPS aufruft.
19	Beispiel19: Synchroner Dateizugriff [▶ 308]	Beschreibt, wie die File-IO-Funktionalität bei einem C++-Modul auf synchrone Art und Weise verwendet werden kann. Das Beispiel schreibt Prozesswerte in eine Datei. Das Beschreiben der Datei wird von einem deterministischen Zyklus veranlasst - die Ausführung von File IO ist entkoppelt (asynchron), d. h.: Der deterministische Zyklus läuft weiter und wird nicht durch das Schreiben in der Datei behindert. Der Status der Routine für entkoppeltes Schreiben in der Datei kann überprüft werden.
20	Beispiel20: FileIO-Write [▶ 308]	Beschreibt, wie die File-IO-Funktionalität bei C++-Modul verwendet werden kann. Das Beispiel schreibt Prozesswerte in eine Datei. Das Beschreiben der Datei wird von einem deterministischen Zyklus veranlasst - die Ausführung von File IO ist entkoppelt (asynchron), d. h.: Der deterministische Zyklus läuft weiter und wird nicht durch das Schreiben in der Datei behindert. Der Status der Routine für entkoppeltes Schreiben in der Datei kann überprüft werden.
20a	Beispiel20a: FileIO-Cyclic Read / Write [▶ 309]	Ist ein umfangreicheres Beispiel als S20 und S19. Es beschreibt zyklischen Lese- und/oder Schreibzugriff auf Dateien von einem C++-Modul aus.
22	Beispiel22: Automation Device Driver (ADD): Zugang DPRAM [▶ 311]	Beschreibt, wie der TwinCAT Automation Device Driver (ADD) für den Zugriff auf die DPRAM zu schreiben ist.
23	Beispiel23: Strukturierte Ausnahmebehandlung (SEH) [▶ 312]	Beschreibt die Verwendung von Structed Exception Handling (SEH) anhand von fünf Varianten.
24	Beispiel24: Semaphoren [▶ 314]	Beschreibt die Verwendung von Semaphoren.

Nummer	Titel	Beschreibung
25	Beispiel25: Statische Bibliothek [▶ 315]	Beschreibt, wie die in einem anderen C++-Modul enthaltene C++ statische Bibliothek verwendet werden kann.
26	Beispiel26: Ausführungsreihenfolge in einer Task [▶ 317]	Beschreibt die Bestimmung der Taskausführungsreihenfolge, wenn einer Task mehr als ein Modul zugeordnet ist.
30	Beispiel30: Zeitmessung [▶ 319]	Beschreibt die Messung der C++-Zyklus- oder Ausführungszeit.
31	Beispiel31: Funktionsbaustein TON in TwinCAT3 C++ [▶ 320]	Beschreibt die Implementierung eines Verhaltens in C++ das vergleichbar mit einem TON Funktionsbaustein von SPS / 61131 ist.
37	Beispiel37: Daten archivieren [▶ 322]	Beschreibt das Laden und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung.
TcCOM	TcCOM Beispiele [▶ 323]	Mehrere Beispiele die die Modulkommunikation zwischen SPS und C++ verdeutlichen.

15.1 Beispiel01: Zyklisches Modul mit IO

Dieser Artikel beschreibt die Implementierung eines TwinCAT 3 C++ Moduls, dessen Modul-IO mit einem physikalischen IO gemappt ist.

Download

Hier erhalten Sie den Quellcode für dieses Sample:
https://github.com/Beckhoff/TC1300_Samples/tree/main/S01-CyclicIO

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**.
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++ Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang Value im Datenbereich zuweist.
Der Datenbereich kann dem physikalischen IO oder einem anderen logischen Eingang einer anderen Modulinstanz zugewiesen werden.

Das Beispiel ist in der Kurzanleitung [▶ 62] Schritt für Schritt beschrieben.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340291083.zip

15.2 Beispiel02: Zyklische C++ Logik, die IO von der IO Task verwendet

Dieser Artikel beschreibt die Implementierung eines TwinCAT 3 C++ Moduls das ein Image einer IO Tasks verwendet.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S02-CyclicIOTask

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Quellcode, der vom Assistenten nicht automatisch generiert wird, ist mit einem Beginn-Flag „//sample code“ und Ende-Flag „//sample code end“ gekennzeichnet.
Somit können Sie nach diesen Zeichenketten in den Dateien suchen, um sich ein Bild von den Einzelheiten zu machen.

Beschreibung

Dieses Beispiel beschreibt die Flexibilität von C++ Code bei der Arbeit mit IOs, welche an der Task konfiguriert sind. Durch diesen Ansatz kann ein kompiliertes C++ Modul flexibler reagieren, wenn mit der IO Task unterschiedlich viele IOs verbunden sind. Eine Einsatzmöglichkeit wäre die zyklische Prüfung analoger Eingangskanäle mit je nach Projekt unterschiedlicher Anzahl Kanäle.

Das Beispiel enthält

- das C++ Modul TcIoTaskImageAccessDrv mit einer Instanz des Moduls TcIoTaskIMageAccessDrv_Obj1
- Eine Task „Task1“ mit einem Image, 10 Eingangsvariablen (Var1..Var10) und 10 Ausgangsvariablen (Var11..Var20).
- Sie sind verbunden: Die Instanz wird von der Task aufgerufen und die Instanz nutzt das Image von Task1.

Der C++ Code greift über ein Datenimage auf die Werte zu, das beim Übergang SAFEOP to OP (SO) initialisiert wird.

In der zyklisch ausgeführten Methode CycleUpdate wird durch Aufruf der Helfermethode CheckValue der Wert jeder Eingangsvariablen überprüft. Ist er kleiner als 0, wird die entsprechende Ausgangsvariable auf 1 gesetzt, ist er größer als 0, wird sie auf 2 gesetzt und ist sie gleich 0, dann wird der Ausgang auf 3 gesetzt.

Nach der Aktivierung der Konfiguration können Sie auf die Variablen über den Solution Explorer zugreifen und diese setzen.
Doppelklicken Sie auf das Task1-Abbild des Systems, um eine Übersicht zu erhalten.
Die Eingangsvariablen können geöffnet und dann mit dem Karteireiter **Online** gesetzt werden.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340292747.zip

15.3 Beispiel03: C++ als ADS Server

Dieser Artikel beschreibt:

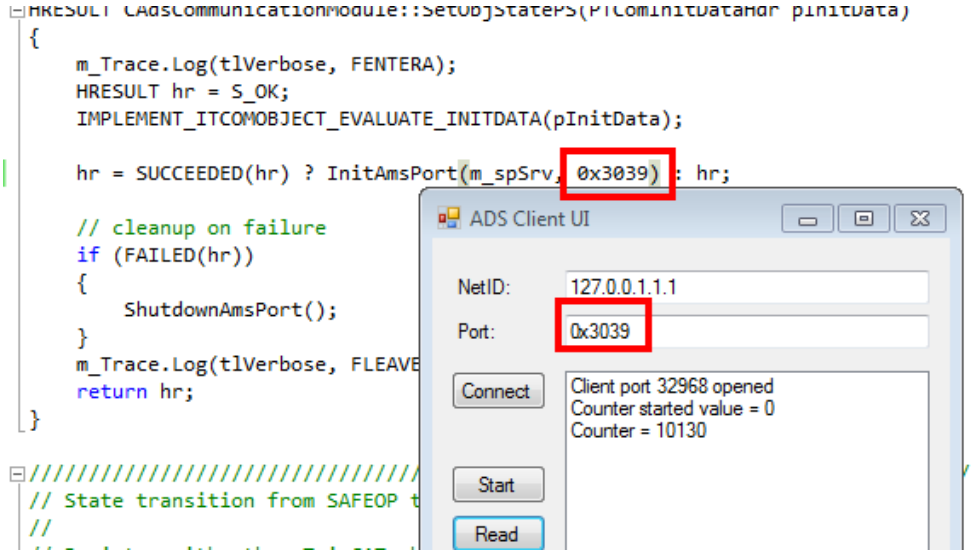
- Die Erstellung eines TwinCAT 3 C++ Moduls, das als ADS Server fungiert.
  Der Server stellt eine ADS-Schnittstelle zum Starten / Stoppen / Zurücksetzen einer Zählervariablen im C++ Modul bereit. Der Zähler steht als Modulausgang zur Verfügung und kann einer Ausgangsklemme (analog oder Anzahl digitaler IOs) zugeordnet werden.
  Wie die in C++ geschriebene TC3 ADS Server-Funktionalität zu implementieren ist [▶ 256].

- Die Erstellung eines C# ADS Clients, um mit dem C++ ADS Server zu interagieren.
  Der Client stellt eine UI zur Verfügung, um lokal oder über Netzwerk mit einem ADS Server mit zu zählender ADS-Schnittstelle verbunden zu werden. Die UI ermöglicht das Starten / Stoppen/ Lesen / Überschreiben und Zurücksetzen des Zählers.
  Beispielcode: ADS Client UI geschrieben in C# [▶ 261].

Das Beispiel verstehen

Im Beispiel werden Möglichkeiten zur automatischen Bestimmung eines ADS Ports verwendet. Dies hat den Nachteil, dass der Client bei jedem Start konfiguriert werden muss, um auf den richtigen ADS Port zuzugreifen.

Alternativ kann der ADS Port im Modul, wie unten gezeigt, hart kodiert werden.
Nachteil hier: Das C++ Modul kann nicht mehr als einmal instanziiert werden, da das Teilen eines ADS Ports nicht möglich ist.



15.3.1 Beispiel03: Der in C++ geschriebene TC3 ADS Server

In diesem Artikel wird die Erstellung eines als ADS Server agierenden TwinCAT 3 C++ Moduls beschrieben. Der Server stellt eine ADS-Schnittstelle zum Starten / Stoppen / Zurücksetzen einer Zählervariablen im C++ Modul bereit.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S03-ADSServer

.

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**

2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Dieses Beispiel beinhaltet ein C++ Modul, das als ADS Server agiert, der einen Zugang zu einem Zähler gewährt, der gestartet, gestoppt und gelesen werden kann.

Die Header-Datei des Moduls definiert die Zählervariable m_bCount, und die entsprechende .cpp-Datei initialisiert den Wert im Konstruktor und implementiert die Logik in der CycleUpdate Methode.

Die Methode AdsReadWriteInd in der .cpp-Datei analysiert die eingehenden Meldungen und sendet die Rückgabewerte zurück. Für einen weiteren hinzugefügten Meldungstyp wird ein define in der Header-Datei hinzugefügt.

Details, wie die Definition der ADS-Meldungstypen, werden im folgenden Kochbuch beschrieben, wo Sie das Beispiel manuell zusammenstellen können.

Kochbuch

Nachfolgend werden die einzelnen Schritte der Erstellung des C++ Moduls beschrieben.

1. Erstellen Sie eine neue TwinCAT 3 Project Solution

Folgen Sie den Schritten für die Erstellung eines neuen TwinCAT 3 Projekts [▶ 62].

2. Erstellen Sie ein C++ Projekt mit ADS Port

Folgen Sie den Schritten für die Erstellung eines neuen TwinCAT 3 C++ Projekts [▶ 63].

Wählen Sie im **Class templates**-Dialog **TwinCAT Module Class with ADS port**.

3. Fügen Sie dem Projekt die Beispiellogik hinzu

1. Öffnen Sie die Header-Datei <MyClass>.h (in diesem Beispiel Module1.h) und fügen den Zähler m_bCount als neue Membervariable in den geschützten Bereich hinzu:

```
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. Öffnen Sie die Klassendatei <MyClass>.cpp (in diesem Beispiel Module1.cpp) und initialisieren die neuen Werte im Konstruktor:

```
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
}
```

⇨ Der Beispielcode ist hinzugefügt.

3.a. Fügen Sie der ADS-Serverschnittstelle die Beispiellogik hinzu.

Normalerweise empfängt der ADS Server eine ADS-Meldung, die zwei Parameter (indexGroup und indexOffset) und gegebenenfalls weitere Daten pData enthält.

ADS-Schnittstelle entwerfen
Unser Zähler sollte gestartet, gestoppt, zurückgesetzt, mit Wert überschrieben werden oder dem ADS Client auf Anfrage einen Wert liefern:

indexGroup	indexOffset	Beschreibung
0x01	0x01	m_bCount = TRUE, Zähler wird inkrementiert.
0x01	0x02	Zählerwert wird ADS Client übergeben.
0x02	0x01	m_bCount = FALSE, Zähler wird nicht mehr inkrementiert.
0x02	0x02	Zähler zurücksetzen.
0x03	0x01	Zähler mit vom ADS Client übergebenen Wert überschreiben.

Diese Parameter sind in modules1Ads.h definiert – ändern Sie den Quellcode, um einen neuen Befehl für IG_RESET hinzuzufügen.

```
#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
};
```

Ändern Sie den Quellcode in Ihrer <MyClass>::AdsReadWriteInd() Methode (in diesem Falle in 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. Fügen Sie die Beispiellogik in den zyklischen Teil ein

Die Methode <MyClass>::CycleUpdate() wird zyklisch aufgerufen - das ist die Stelle, wo die Logik zu verändern ist.

```
// TODO: Replace the sample with your cyclic code
m_Counter+=m_Inputs.Value; // replace this line
m_Outputs.Value=m_Counter;
```

In diesem Falle wird der Zähler **mCounter** inkrementiert, wenn die boolsche Variable **m_bCount** TRUE ist.

Fügen Sie diesen If-Fall in Ihre zyklische Methode ein.

```
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. Server-Beispiel ausführen

1. Führen Sie den TwinCAT TMC Code Generator aus, um die Ein-/Ausgänge für das Modul bereitzustellen.
2. Speichern Sie das Projekt.
3. Kompilieren Sie das Projekt.
4. Erstellen Sie eine Modulinstanz.
5. Erstellen Sie eine zyklische Task und konfigurieren Sie das C++ Modul für die Ausführung in diesem Kontext.
6. Scannen Sie die Hardware IO und ordnen Sie das Symbol Value von Ausgängen bestimmten Ausgangsklemmen zu (dies ist optional).
7. Aktivieren [▶ 77] Sie das TwinCAT Projekt.

⇨ Das Beispiel ist einsatzbereit.

5. Finden Sie den ADS Port der Modulinstanz heraus

Im Allgemeinen kann der ADS Port

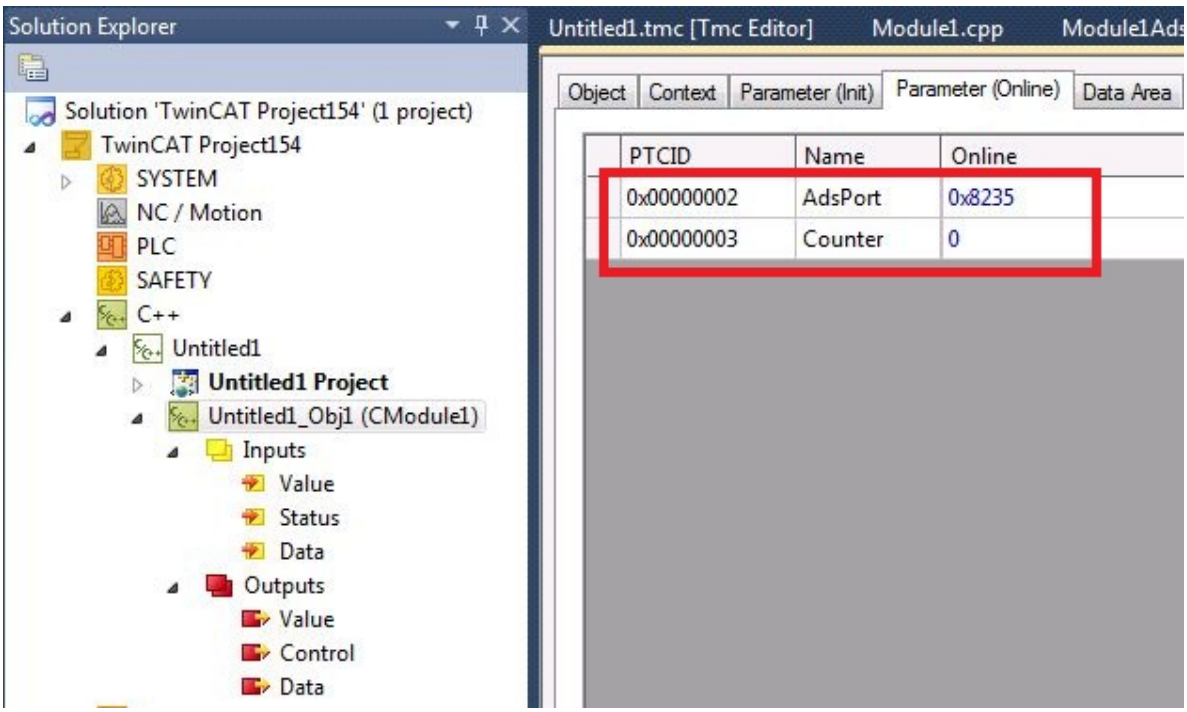
- vornummeriert sein, damit es immer der gleiche Port für diese Modulinstanz ist.
- flexibel gehalten werden, um mehreren Instanzen des Moduls die Möglichkeit zu bieten, dass ihnen beim Start des TwinCAT Systems ihr eigener ADS Port zugewiesen wird.

In diesem Beispiel ist die Standardeinstellung (flexibel halten) gewählt. Zunächst ist der ADS Port zu ermitteln, der dem Modul, das soeben aktiviert wurde, zugewiesen wurde:

1. Navigieren Sie zur Modulinstanz.
2. Wählen Sie den Karteireiter **Parameter Online**.

   ⇨ Dem ADSPort ist 0x8235 bzw. dezimal 33333 zugewiesen (das kann in Ihrem Beispiel anders sein). Werden mehr und mehr Instanzen erstellt, erhält jede Instanz ihren eigenen eindeutigen AdsPort.

   ⇨ Der Zähler steht immer noch auf „0“, da die ADS-Meldung zum Starten der Inkrementierung nicht gesendet worden ist.



⇨ Der Server-Teil ist abgeschlossen - fahren Sie fort mit ADS Client sendet die ADS-Meldungen [▶ 261].

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340296075.zip

15.3.2 Beispiel03: ADS Client UI in C#

Dieser Artikel beschreibt den ADS Client, der ADS-Meldungen an den zuvor beschriebenen ADS Server sendet.

Die Implementierung des ADS Servers hängt weder von der Sprache (C++ / C# / SPS / ...), noch von der TwinCAT Version (TwinCAT 2 oder TwinCAT 3) ab.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S03-ADSClient

✓ Dieser Code erfordert .NET Framework

1. Öffnen Sie die enthaltene sln-Datei mit Visual Studio.
2. Erstellen Sie das Beispiel auf Ihrer lokalen Maschine (rechtsklicken Sie auf das Projekt und klicken Sie **Build**).
3. Starten Sie das Programm mit einem Rechtsklick auf **Projekt**, **Debug**->**Start new instance.**

Beschreibung

Dieser Client führt zwei Aufgaben aus:

- Den zuvor beschriebenen ADS Server testen.
- Beispielcode für die Implementierung eines ADS Client bereitstellen.

Den Client benutzen

Kommunikationspartner auswählen

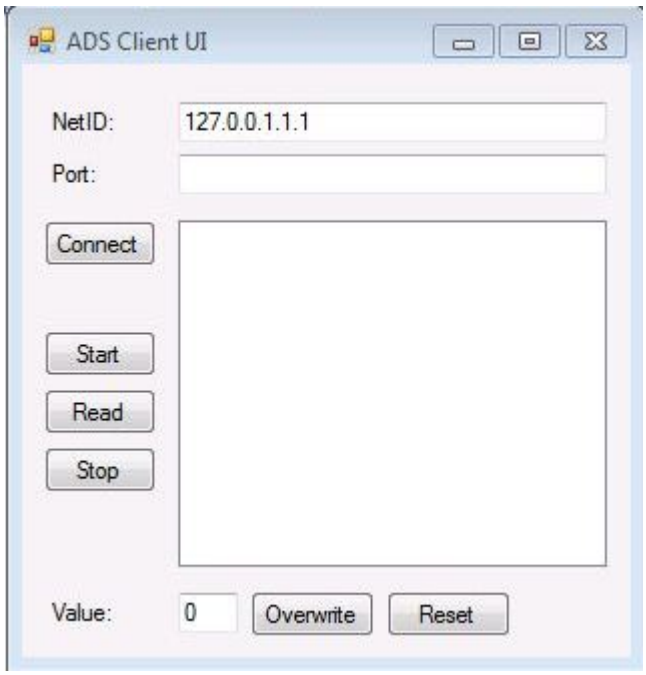
Geben Sie die beiden ADS-Parameter ein, um Ihren ADS-Kommunikationspartner zu bestimmen:

- NetID:
  127.0.0.1.1.1 (für ADS-Partner auch mit lokalem ADS Message Router verbunden)
  Geben Sie eine andere NetID ein, wenn Sie über das Netzwerk mit einem an einen anderen ADS Router angeschlossenen ADS-Partner kommunizieren möchten.
  Zuvor müssen Sie einmal eine ADS Route zwischen Ihrem Gerät und dem fernen Gerät herstellen.

- AdsPort
  Geben Sie den AdsServerPort Ihres Kommunikationspartners ein.
  Verwechseln Sie nicht den ADS Server Port (der ausdrücklich Ihren eigenen Message-Handler implementiert hat) mit dem regulären ADS Port zwecks Zugriff auf Symbole (es gibt nichts zu tun, wird automatisch zur Verfügung gestellt).
  Finden Sie den zugewiesenen AdsPort [▶ 256] in diesem Beispiel heraus, der AdsPort war 0x8235 (dez 33333).

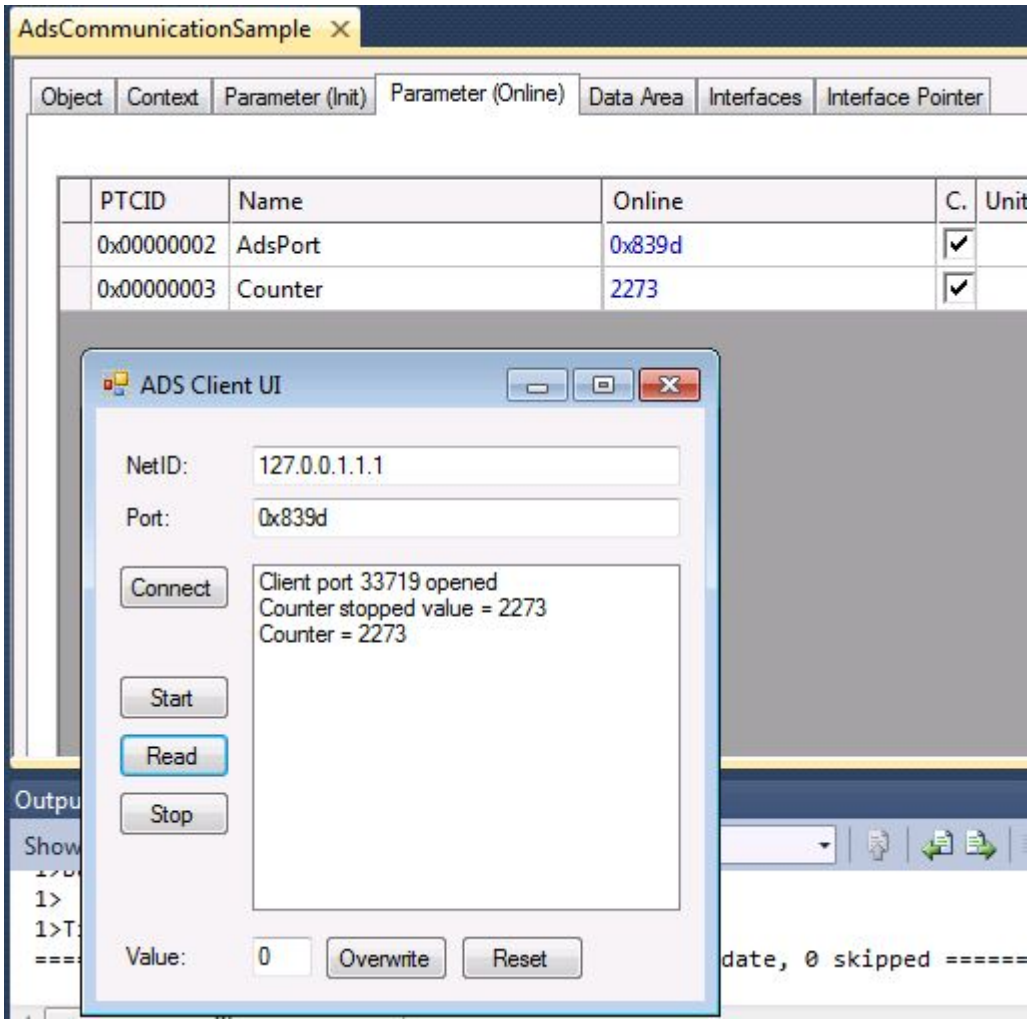
Verbindung mit Kommunikationspartner herstellen

Beim Klicken auf **Connect** wird die Methode TcAdsClient.Connect zwecks Herstellung einer Verbindung mit dem konfigurierten Port aufgerufen.



Mit Hilfe der Schaltflächen **Start** / **Lesen** / **Stopp** / **Überschreiben** / **Zurücksetzen** werden ADS-Meldungen an den ADS Server gesendet.
Die spezifischen indexGroup / indexOffset Befehle wurden bereits in der ADS-Schnittstelle des ADS Servers entworfen [▶ 256].

Das Ergebnis des Klickens auf die Befehlsschaltflächen ist auch in der Modulinstanz auf der Registerkarte **Parameter (Online)** zu sehen.



C# Programm

Hier ist der „Kern“-Code des ADS Client - für die GUI usw. ZIP-Datei oben herunterladen.

```
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);
}
}
}
}
```

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340294411.zip

15.4 Beispiel05: C++ CoE Zugriff über ADS

In diesem Artikel wird die Implementierung eines TwinCAT 3 C++ Moduls beschrieben, das auf das CoE (CANopen over EtherCAT) Register einer EtherCAT Klemme zugreifen kann.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S05-CoEAccess

1. Öffnen Sie die enthaltene zip-Datei in TwinCAT 3 mit einem Klick auf **Open Project …**.
2. Wählen Sie Ihr Zielsystem aus.
3. Bauen Sie das Beispiel auf Ihrer lokalen Maschine (z.B. **Build->Build Solution**).
4. Beachten Sie die unter **Konfiguration** auf dieser Seite aufgeführten Handlungsschritte.
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

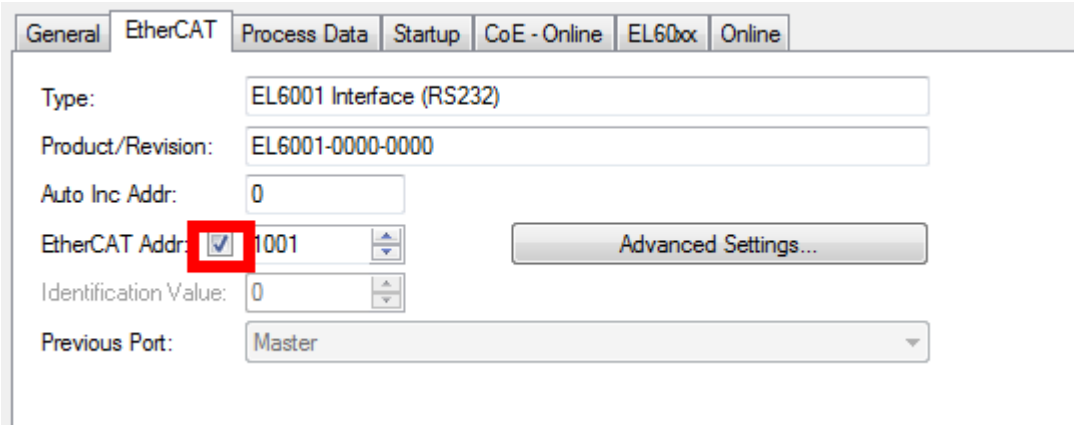
Beschreibung

Dieses Beispiel beschreibt den Zugriff auf eine EtherCAT-Klemme, der die Hersteller-ID liest und die Baudrate für die serielle Kommunikation festlegt.

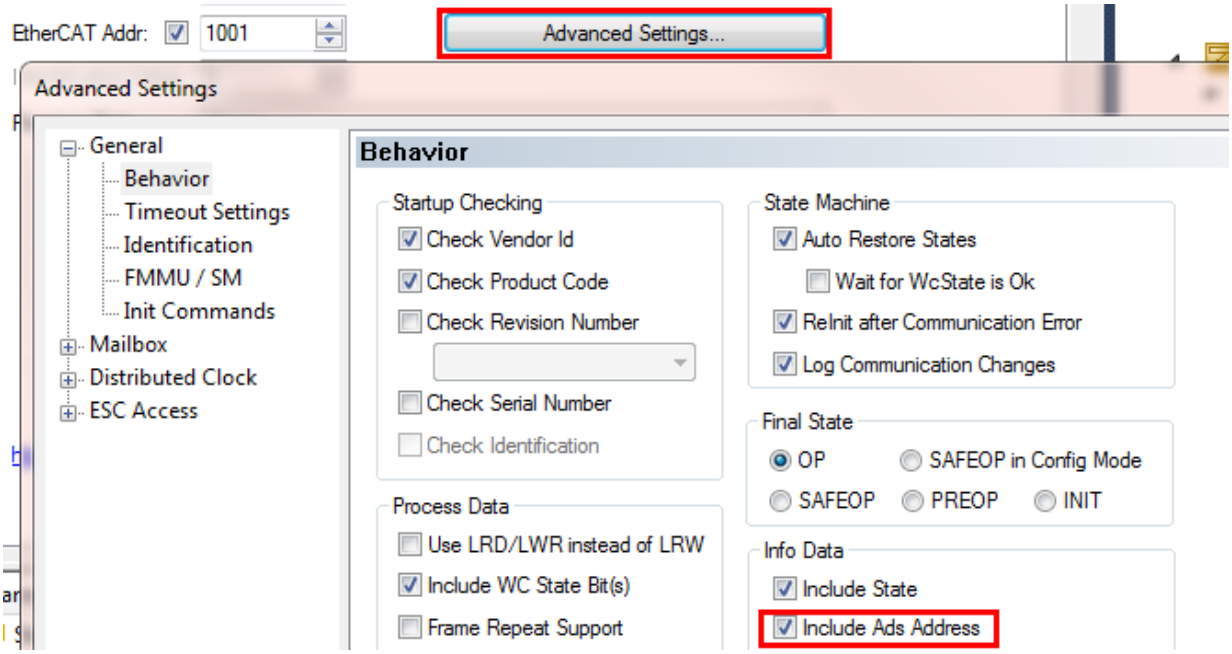
Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++ Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang Value im Datenbereich zuweist.

Konfiguration

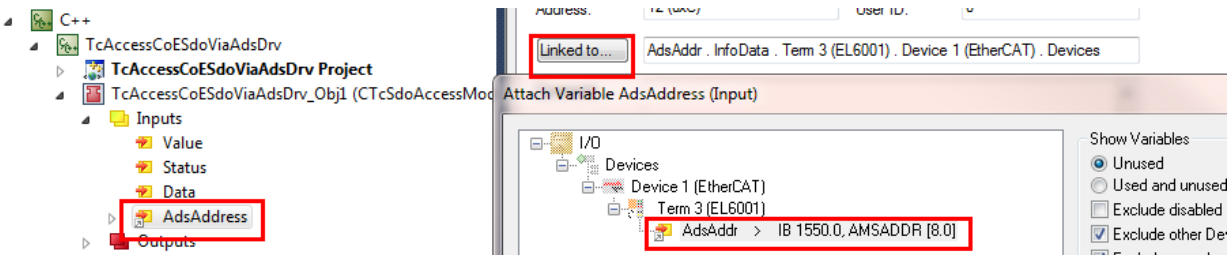
1. Aktivieren Sie die EtherCAT-Adresse der betreffenden Klemme und weisen sie zu.



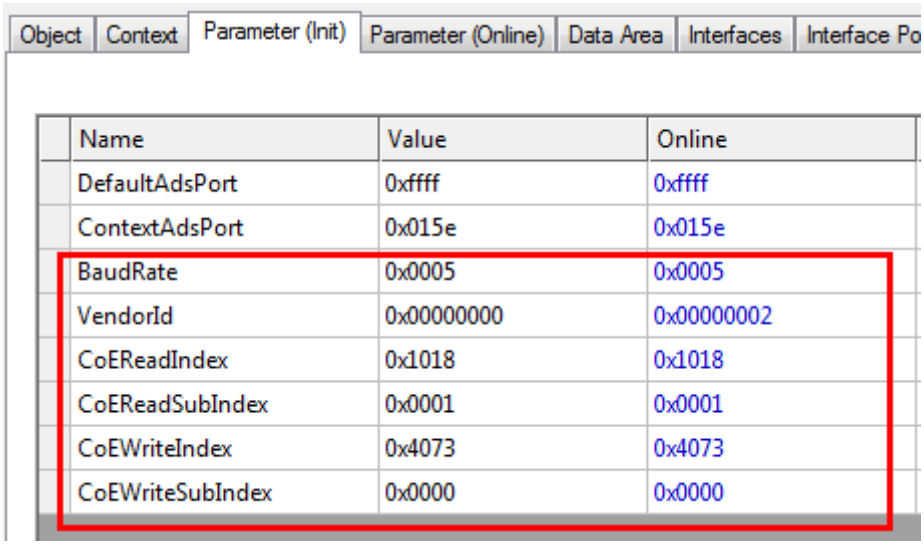
2. Aktivieren Sie den Einschluss der ADS-Adresse in den erweiterten Einstellungen der EtherCAT-Klemme:



3. Weisen Sie die ADS-Adresse (einschließlich netId und port) dem Moduleingang AdsAdress zu:



4. Die Modulparameter werden im Verlauf der Initialisierung durch den Samplecode ausgelesen und angezeigt:



Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340297739.zip

15.5 Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch

Dieser Artikel beschreibt die Implementierung eines ADS Client um

- mit einem ADS Server, der ein Prozessabbild (Datenbereich) bereitstellt, in Verbindung zu treten, die Verbindung kann lokal oder fern über Netzwerk hergestellt werden,
- Symbolinformation hochzuladen,
- Daten synchron zu lesen / schreiben,
- Symbole zu abonnieren, um Werte bei Veränderung als Callback zu erhalten.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S06-SymbolUploadClient

✓ Dieser Code erfordert .NET Framework 3.5 oder höher!

1. Öffnen Sie die enthaltene sln-Datei mit Visual Studio.
2. Erstellen Sie das Beispiel auf Ihrer lokalen Maschine (auf das Projekt rechtsklicken und „Build“ klicken).
3. Starten Sie das Programm mit einem Rechtsklick auf **Projekt**, **Debug->Start new instance.**

Das Client-Beispiel sollte mit Beispiel 03 „C++ als ADS Server“ verwendet werden.

Öffnen Sie Beispiel 03 [▶ 256] bevor Sie mit diesem Client-seitigen Beispiel beginnen!

Beschreibung

Anhand dieses Beispiels werden die Möglichkeiten der ADS beschrieben.

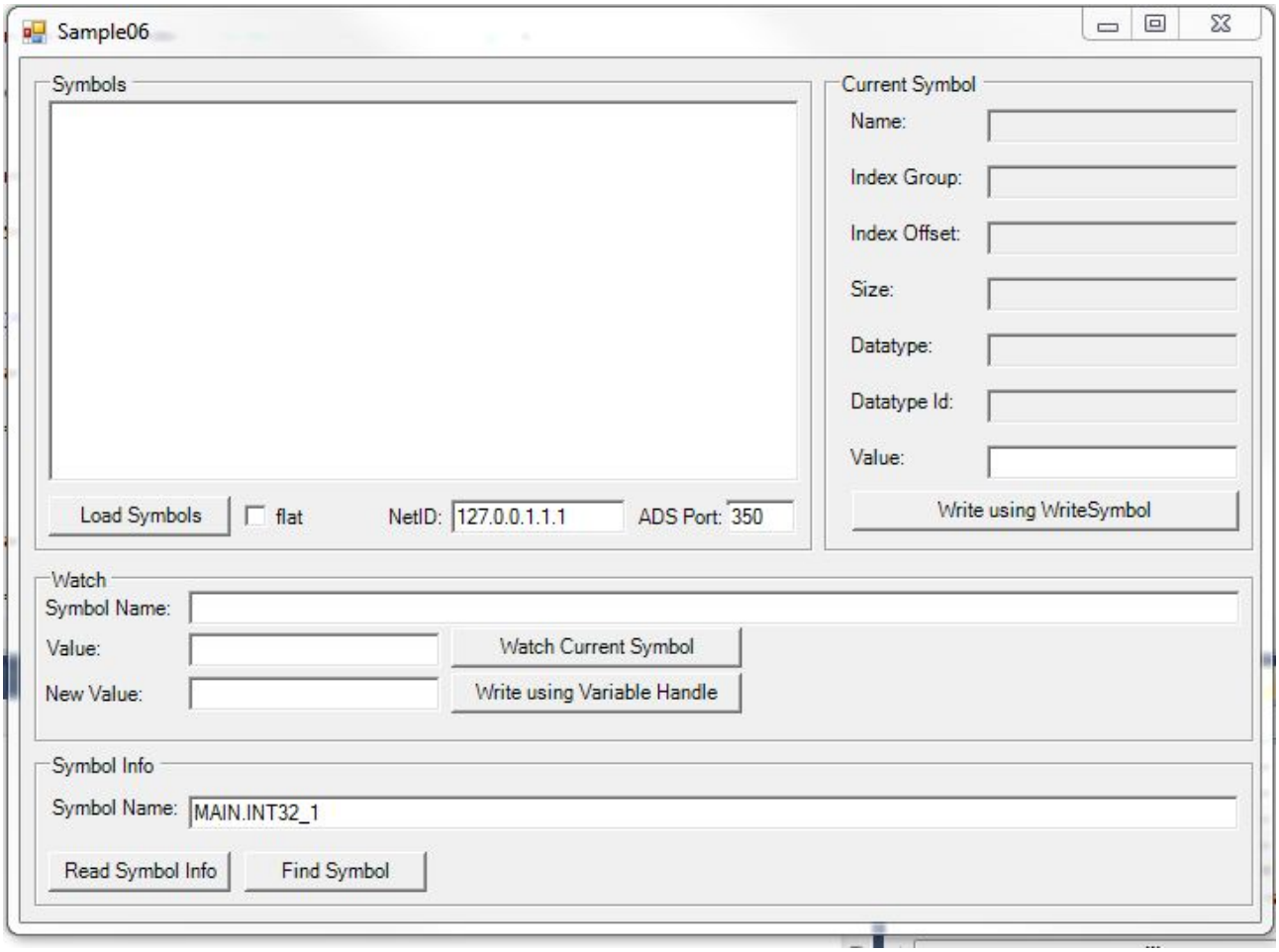
Die Einzelheiten der Implementierung werden in Form1.cs, das im Download enthalten ist, beschrieben. Die Verbindung über ADS mit dem Zielsystem wird in der Methode btnLoad_Click, die beim Klicken auf die **Load Symbols** Schaltfläche aufgerufen wird, hergestellt. Von dort aus können Sie die verschiedenen Funktionen der GUI erkunden.

Hintergrundinformation:

Diesem ADS Client ist es Einerlei, ob der ADS Server auf TwinCAT 2 oder auf TwinCAT 3 basiert. Auch spielt es keine Rolle, ob der Server ein C++ Modul, ein SPS-Modul oder eine IO Task ohne jede Logik ist.

Die ADS Client UI

Nach dem Starten des Beispiels wird die Benutzerschnittstelle (UI) eingeblendet.



Kommunikationspartner auswählen

Geben Sie nach dem Starten des Client die beiden ADS-Parameter ein, um Ihren ADS-Kommunikationspartner zu bestimmen.

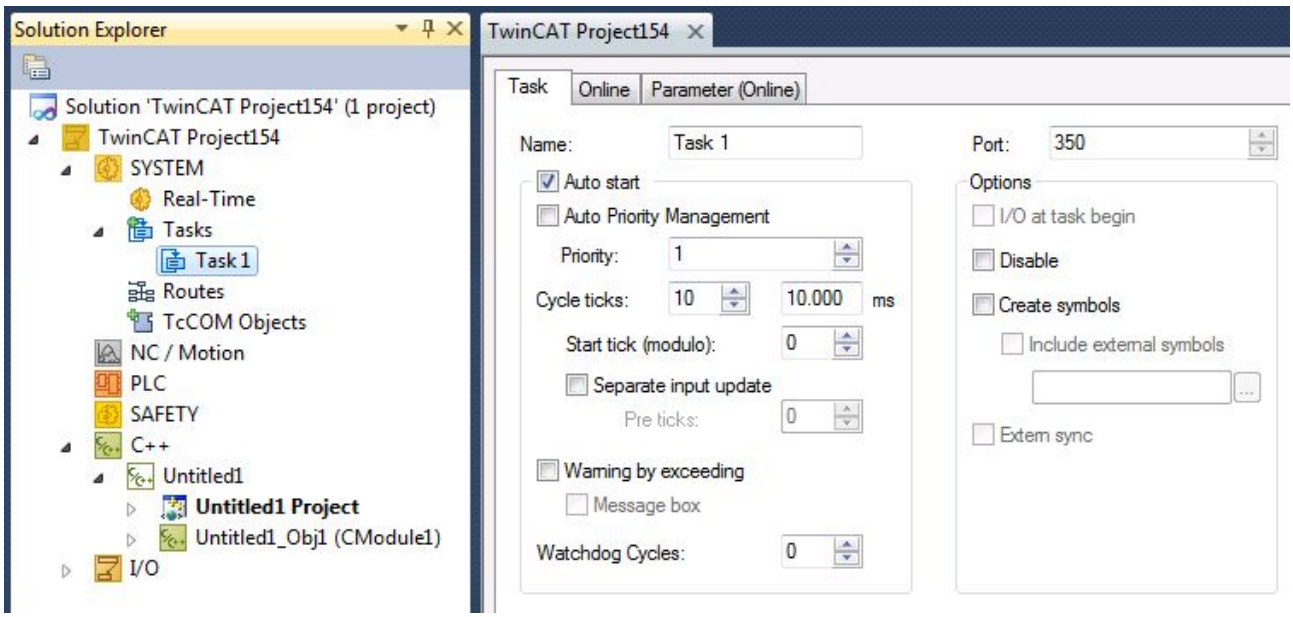
- NetID:
  127.0.0.1.1.1 (für ADS-Partner auch mit lokalem ADS Message Router verbunden).
  Geben Sie eine andere NetID ein, wenn Sie über das Netzwerk mit einem an einen anderen ADS Router angeschlossenen ADS-Partner kommunizieren möchten.
  Zuvor müssen Sie einmal eine ADS Route zwischen Ihrem Gerät und dem fernen Gerät herstellen.
- AdsPort
  Geben Sie den AdsPort Ihres Kommunikationspartners ein: 350 (in diesem Beispiel).

Nicht den ADS Server Port mit dem regulären ADS Port verwechseln.

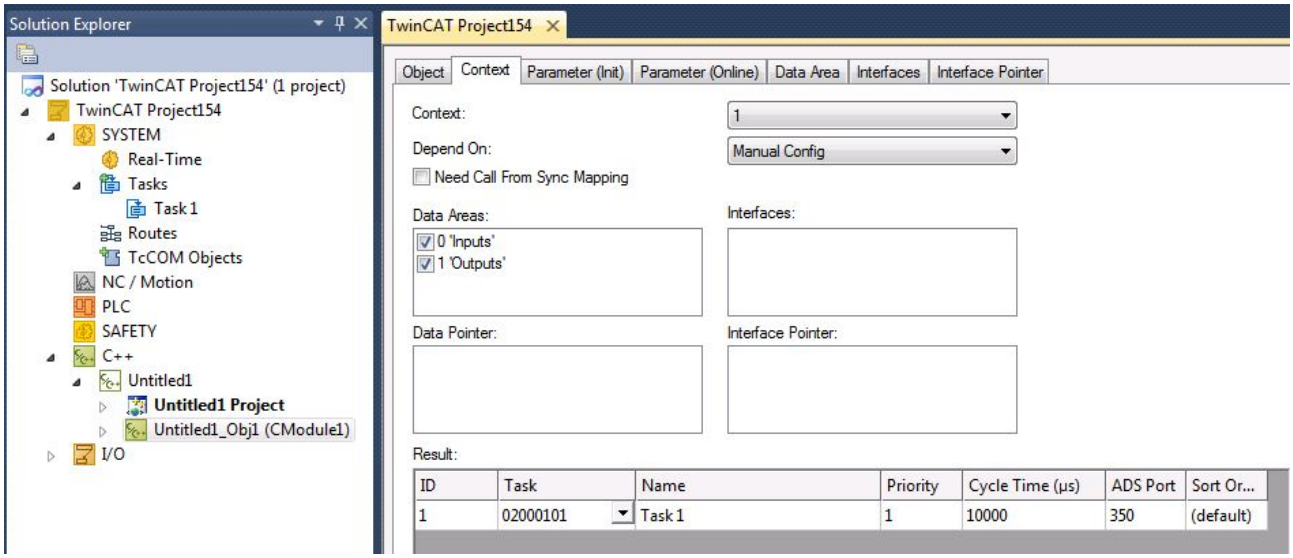
Verwechseln Sie nicht den ADS Server Port (der ausdrücklich in Beispiel 03 implementiert wurde, um Ihren eigenen Message-Handler bereitzustellen) mit dem regulären ADS Port zwecks Zugriff auf Symbole (es gibt nichts zu tun, wird automatisch zur Verfügung gestellt):

Wir benötigen den regulären ADS Port, um auf Symbole zugreifen zu können. Sie können den AdsPort bei der IO Task Ihrer Instanz oder bei der Modulinstanz selber herausfinden (weil das Modul im Kontext der IO Task ausgeführt wird).

Navigieren Sie zur IO Task Task1 und achten auf den Wert von Port: 350.



Weil die C++ Modulinstanz im Kontext von Task1 ausgeführt wird, ist der ADS Port auch gleich 350.

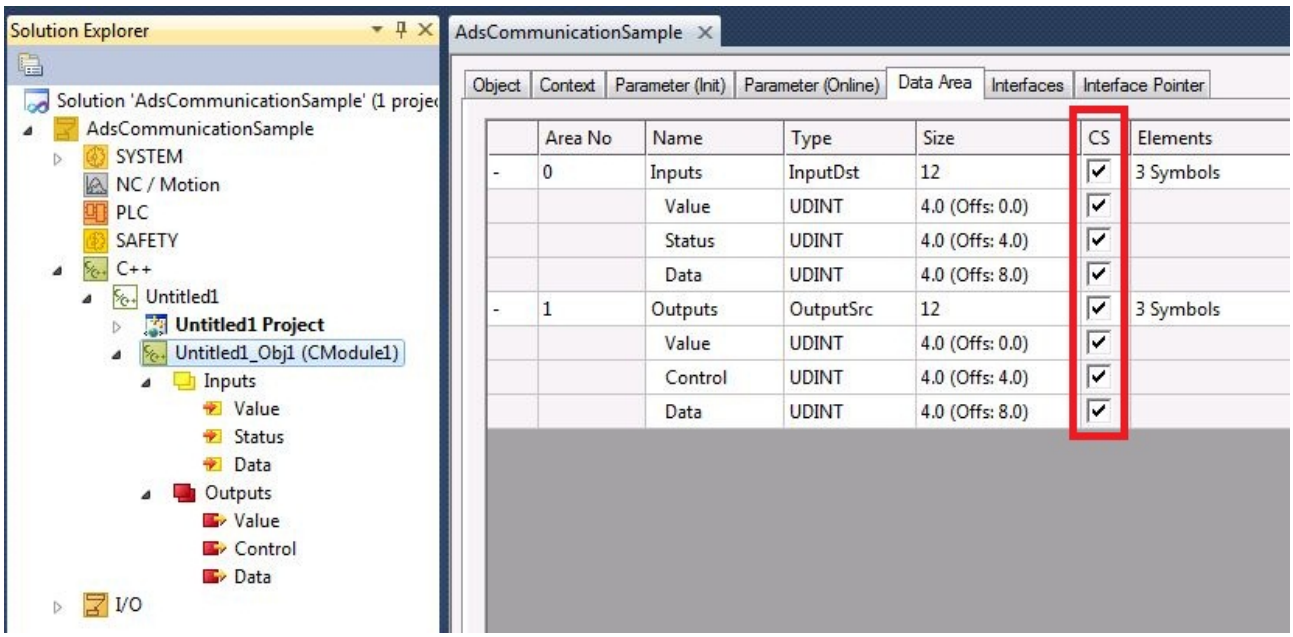


Aktivierte Symbole für Zugriff über ADS verfügbar

Es besteht die Möglichkeit einzelne Symbole oder gar vollständige Datenbereiche für den Zugriff über ADS bereitzustellen oder eben nicht bereitzustellen.
Navigieren Sie zur der Registerkarte **Data Area** Ihrer Instanz und aktivieren/deaktivieren Sie die Spalte **C/S**.

In diesem Beispiel sind alle Symbole gekennzeichnet und demzufolge für den ADS-Zugriff verfügbar.

Nach den Änderungen klicken Sie auf **Konfiguration aktivieren**.



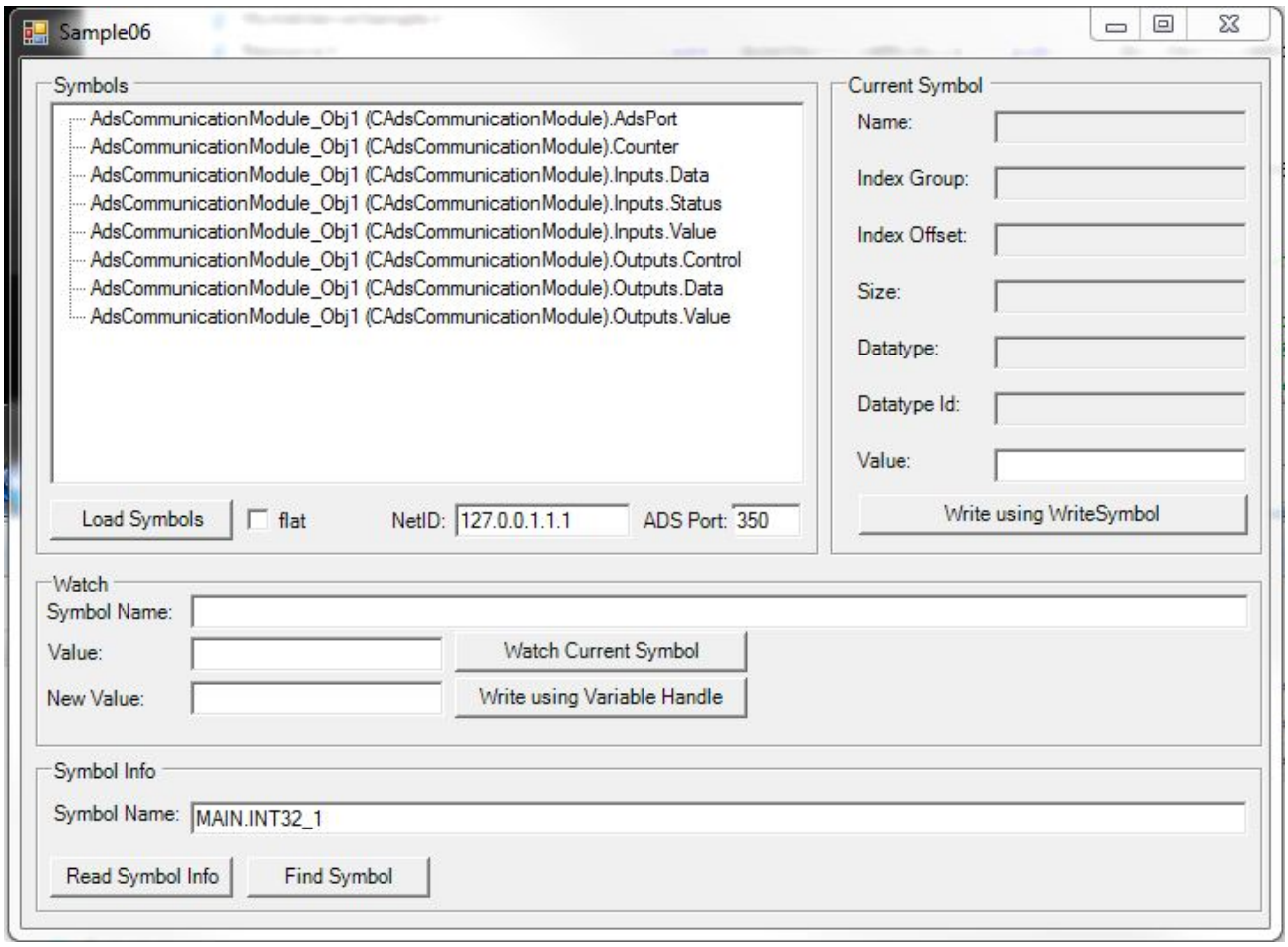
Symbole laden

Nach der Einrichtung der NetID und des ADS Port klicken Sie auf die Schaltfläche **Load Symbols**, um eine Verbindung mit dem Zielsystem herzustellen und die Symbole zu laden.

Daraufhin sind alle verfügbaren Symbole zu sehen. Anschließend können Sie:

- Einen neuen Wert schreiben:
  wählen Sie ein Symbol im linken Baum au, z. B. **Counter.**
  geben Sie einen neuen Wert in das Bearbeitungsfeld **Value** auf der rechten Seite ein und klicken Sie auf **Write using WriteSymbol**.
  Der neue Wert wird in den ADS Server geschrieben.

  Nach dem Schreiben eines neuen Werts mit **Write using WriteSymbol** erhält auch die C# Anwendung einen Callback mit dem neuen Wert.
- Abonnieren, um einen Callback bei Werteveränderungen zu erhalten:
  wählen Sie ein Symbol im linken Baum aus z. B. **Counter.**
  klicken Sie auf **Watch Current Symbol**.



Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340299403.zip

15.6 Beispiel07: Empfang von ADS Notifications

Dieser Artikel beschreibt die Implementierung eines TwinCAT 3 C++ Moduls, das ADS Notifications bezüglich Datenänderungen auf anderen Modulen empfängt.
Weil jede andere ADS-Kommunikation auf ähnliche Weise implementiert werden muss, kann dieses Beispiel als allgemeiner Einstiegspunkt für die Initialisierung der ADS-Kommunikation von TwinCAT 3 C++ Modulen aus betrachtet werden.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S07-AdsNotifications

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Dieses Beispiel beschreibt den Empfang von ADS Notifications in einem TwinCAT C++ Modul.

Hierfür enthält die Solution 2 Module.

- Ein C++ Modul, das sich für die Abfrage von ADS Notifications einer Variablen anmeldet.
- Zum einfachen Verständnis: Ein SPS-Programm, das eine Variable MAIN.PlcVar bereitstellt. Wenn ihr Wert sich verändert, wird eine ADS Notification an das C++ Modul geschickt.
- Das C++ Modul nutzt die Möglichkeiten Meldungen aufzuzeichnen - also zum Verständnis des Codes, starten Sie einfach das Beispiel und achten dann auf den Ausgang / Fehler-Log, wenn Sie den Wert Main.PlcVar des SPS-Moduls ändern.

Die Adresse wird beim Modulübergang PREOP->SAFEOP (SetObjStatePS) vorbereitet. Die Methode CycleUpdate beinhaltet eine einfache Zustandsmaschine, die den erforderlichen ADS-Befehl sendet. Entsprechende Methoden zeigen die Empfangsbestätigungen an.

Die geerbte und überladene Methode AdsDeviceNotificationInd wird beim Eingang einer Notification aufgerufen.

Während der Abschaltung werden ADS-Meldungen beim Übergang zwecks Abmeldung versendet (SetObjStateOS) und das Modul wartet bis zum Auftreten einer Zeitüberschreitung auf den Eingang von Bestätigungen.

Beginn der Modulentwicklung

Erstellen eines TwinCAT C++ Moduls mit Hilfe des ADS Port-Assistenten. Dadurch wird alles, was Sie zum Aufbau einer ADS-Kommunikation benötigen, eingerichtet. Einfach die notwendigen ADS-Methoden von ADS.h, wie im Beispiel gezeigt, verwenden und überschreiben.

Siehe auch

ADS-Kommunikation [▶ 200]

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340301067.zip

15.7 Beispiel08: Anbieten von ADS-RPC

Dieser Artikel beschreibt die Implementierung von Methoden, welche per ADS über die Task aufrufbar sind.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S08-ADSRPC

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Der Download enthält 2 Projekte:

- Das TwinCAT Projekt, welches ein C++ Modul beinhaltet. Dieses bietet einige Methoden an, die per ADS aufgerufen werden können
- Gleichzeitig ist ein Visual C++ Projekt enthalten, welches als Client die Methoden aus dem Usermode aufruft.

Es werden vier Methoden mit unterschiedlichen Signaturen bereitgestellt und aufgerufen. Diese sind in 2 Interfaces organisiert, so dass verdeutlicht wird, wie die ADS Symbolnamen der Methoden zusammengesetzt sind.

Das Beispiel verstehen

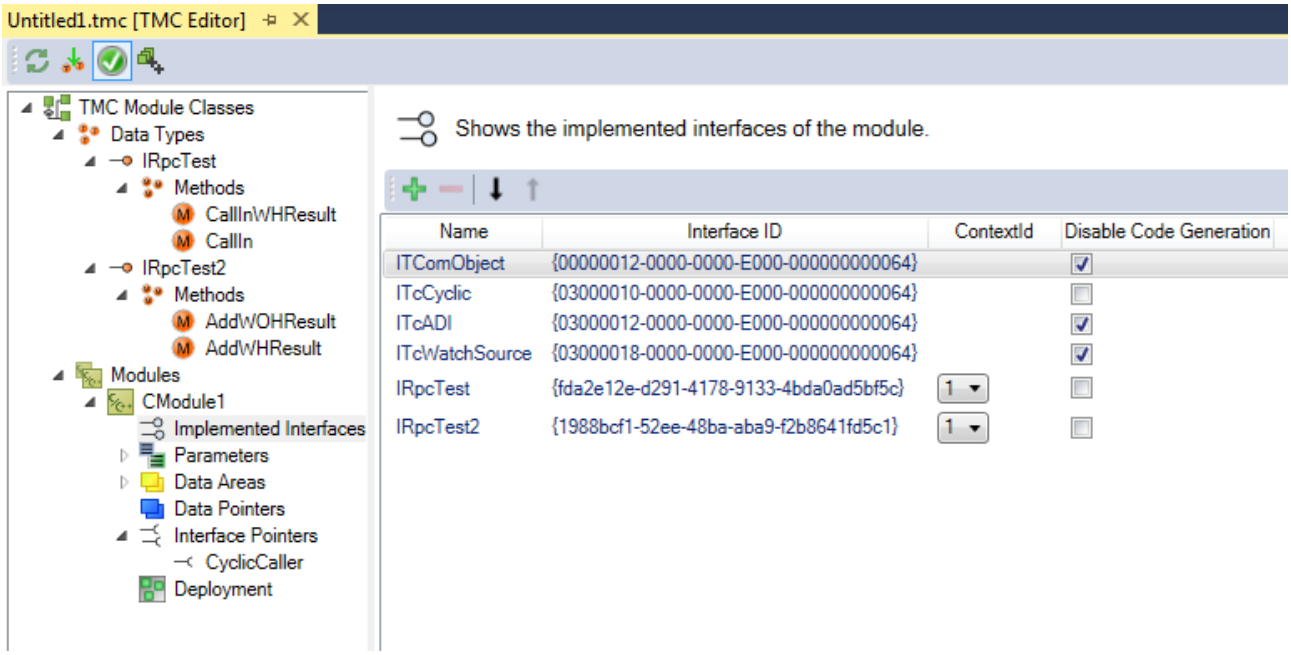
Das Beispiel besteht aus dem TwinCAT C++ Modul, welches die RPC Methoden anbietet und einem C++ Beispielprogramm, welches diese aufruft.

TwinCAT C++ Modul

Das TwinCAT C++ Project beinhaltet ein Modul und eine Instanz des Moduls mit Namen „foobar“.

RPC Methoden sind normale Methoden, welche durch Interfaces im TMC Editor beschrieben sind und zusätzlich durch eine Checkbox **RPC enable** freigegeben werden. In der Beschreibung des TMC Editors [▶ 92] sind die Optionen genauer beschrieben.

Im hier vorliegenden Modul werden zwei Interfaces beschrieben und implementiert, wie im TMC Editor gesehen werden kann:



Die insgesamt vier Methoden haben unterschiedliche Signaturen von Aufruf und Rückgabewerten.

Ihr ADS-Symbolname wird gebildet nach dem Schema: `ModulInstance.Interface#Methodenname`
Wichtig im implementierenden Modul ist insbesondere die ContextId, welche den Kontext für die Ausführung festlegt.

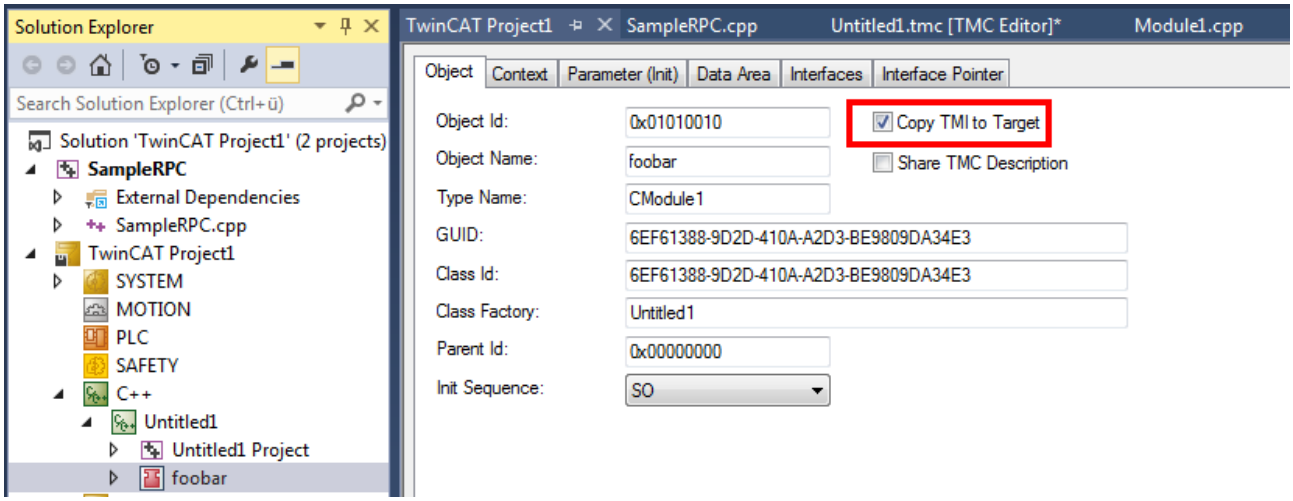
Wie im C++ Code selber zu sehen, werden die Methoden durch den Code Generator generiert und implementiert wie normale Methoden eines TcCOM Moduls.

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>
```

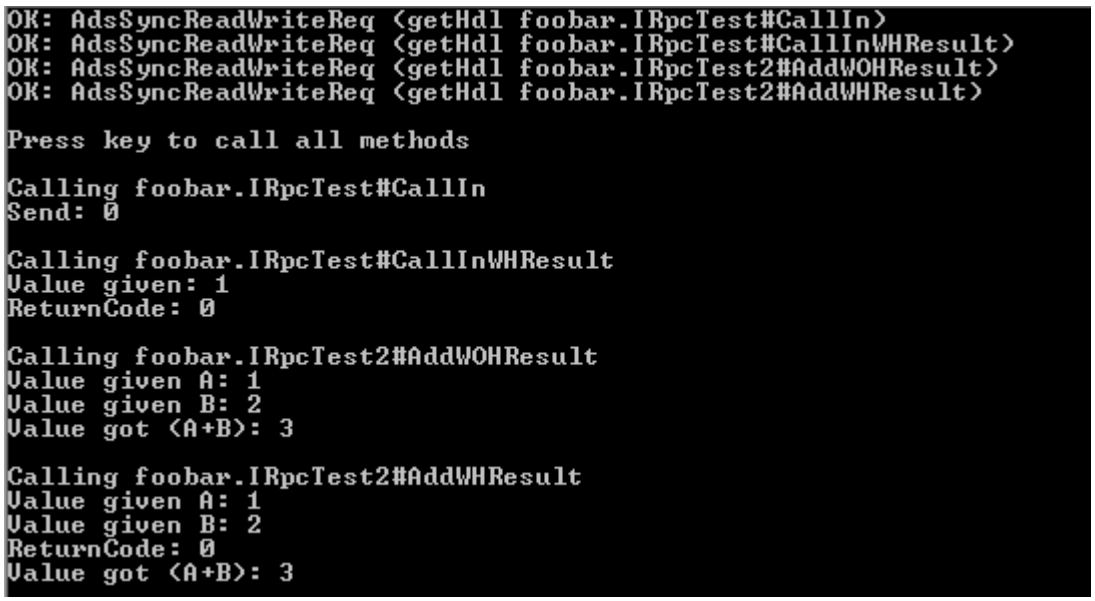
Falls die Typinformationen der Methoden auf dem Zielsystem verfügbar sein sollen, kann die TMI Datei des Moduls auf das Zielsystem übertragen werden.



Der TwinCAT OPC-UA Server bietet die Möglichkeit, diese Methoden auch per OPC-UA aufzurufen – hierfür werden die TMI Dateien auf dem Zielsystem benötigt.

C++ Beispiel Client

Der C++ Client wird direkt nach Starten die Handles holen und dann beliebig häufig die Methoden aufrufen, wobei zwischen den Durchgängen ein [RETURN] erwartet wird. Jede andere Taste führt zur Freigabe der Handles und Beenden des Programms.
Die Ausgaben verdeutlichen die Aufrufe:



Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340302731.zip

15.8 Beispiel10: Modulkommunikation: Verwendung von Datenzeigern

Dieser Artikel beschriebt die Implementierung von zwei TwinCAT 3 C++ Modulen, die über einen Datenzeiger kommunizieren.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S10-Mod2ModDataPointer

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Diese Kommunikation baut auf einem „geteilten“ Datenbereich auf: Von einem Modul bereitgestellt und über Zeiger von einem anderen Modul aus erreichbar.

Es ist nicht möglich, dass zwei verschiedene Datenzeiger mit demselben Eintrag in einen Ausgangs- oder Eingangsdatenbereich verknüpft sind, weil es ohne diese Limitierung zu Synchronisationsproblemen käme. Aus diesem Grunde sammelt ein ModuleDataProvider-Modul Ein- und Ausgang in einen Standarddatenbereich, der nicht dieser Einschränkung unterliegt.

Alles in allem beinhaltet dieses Beispiel die folgenden Module:

- ModuleDataProvider stellt einen Datenbereich zur Verfügung, auf den andere Module zugreifen können.
  Der Datenbereich enthält 4 Bits (2 werden für Eingang, 2 für Ausgang verwendet) und 2 Integer (einen für Eingang, einen für Ausgang).
- ModuleDataInOut stellt „normale“ Eingangsvariablen, die in den Datenbereich des ModuleDataProvider geschrieben werden, und auch Ausgangsvariablen, die
  aus dem Datenbereich gelesen werden, zur Verfügung.
  Diese Instanz der Klasse CModuleDataInOut fungiert als eine Simulation für realen IO.
- ModuleDataAccessA greift auf den Datenbereich vom ModuleDataProvider zu und bearbeitet Bit1 / BitOut1 und die Integer auf zyklische Weise.
- ModuleDataAccessB greift auf den Datenbereich vom ModuleDataProvider zu und bearbeitet Bit2 / BitOut2 und die Integer auf zyklische Weise.

Der Nutzer des Beispiels triggert ModuleDataInOut mittels Setzen der Variablen ValueIn / Bit1 / Bit2:

- Beim Setzen des Eingangs Bit1 wird der Ausgang Switch1 entsprechend gesetzt.
- Beim Setzen des Eingangs Bit2 wird der Ausgang Switch2 entsprechend gesetzt.
- Beim Setzen des Eingangs ValueIn wird der Ausgang ValueOut bei jedem Zyklus zweimal inkrementiert.

Alle Module sind so konfiguriert, dass sie den gleichen Taskkontext haben, was nötig ist, weil der Zugriff über Zeiger keinerlei Synchronisierungsmechanismus bietet. Die Reihenfolge der Ausführung entspricht derjenigen, die auf der Registerkarte Kontextkonfiguration festgelegt wurde. Dieser Wert wird als Parameter SortOrder weitergegeben und im Smart Pointer des zyklischen Aufrufers (m_spCyclicCaller), in dem auch die Objekt-ID des zyklischen Aufrufers enthalten ist, gespeichert.

Das Beispiel verstehen

Das Modul ModuleDataInOut hat Ein- und Ausgangsvariablen. Diese sind mit den entsprechenden Variablen des Datenanbieters verknüpft.

Das Modul ModuleDataProvider stellt einen Eingangs- und einen Ausgangsdatenbereich zur Verfügung und implementiert die Schnittstelle ITcIoCylic. Die Methode InputUpdate kopiert Daten von den Eingangsvariablen auf das DataIn Symbol des standardmäßigen Datenbereichs Data und die Methode OutputUpdate kopiert Daten vom DataOut Symbol auf die Ausgangsvariablen.

Die Module ModuleDataAccessA und ModuleDataAccessB besitzen Zeiger auf Datenbereiche des Datenanbieters über Verknüpfungen. Diese Zeiger werden beim Übergang SAFEOP zu OP initialisiert. ModuleDataAccessA setzt das BitOut1 entsprechend Bit1 auf zyklische Weise. ModuleDataAccessB entsprechend mit BitOut2 / Bit2. Beide inkrementieren ValueOut mittels Multiplikation des internen Zählers mit dem Wert ValueIn.

Wichtig hierbei ist, dass alle Module im gleichen Context ausgeführt werden, da es über Datenzeiger (Datapointer) keinen Synchronisationsmechanismus gibt. Die Reihenfolge der Ausführung wird durch die **Sort Order** im **Context**-Tab des jeweiligen Moduls definiert. Dieses wird als Parameter **SortOrder** im SmartPointer (m_SpCyclicCaller) bereitgestellt, was ebenso die ObjectID bereithält.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340317195.zip

15.9 Beispiel11: Modulkommunikation: Methodenaufruf SPS-Modul nach C++-Modul

Dieser Artikel beschreibt die Implementierung:

- Eines C++ Moduls [▶ 276], das Methoden für die Steuerung einer Zustandsmaschine zur Verfügung stellt.
  Folgen Sie dieser schrittweisen Einführung bezüglich der Implementierung eines C++ Moduls, das eine Schnittstelle zur Zustandsmaschine zur Verfügung stellt.

- Eines SPS-Moduls [▶ 290], um die Funktionalität des C++ Moduls aufzurufen.
  Dass keine hart kodierte Verknüpfung zwischen der SPS und dem C++ Modul existiert, ist ein großer Vorteil. Stattdessen kann die aufgerufene C++ Instanz im Systemmanager konfiguriert werden.
  Folgen Sie dieser schrittweisen Einführung bezüglich der Implementierung eines SPS Projekts, das Methoden von einem C++ Modul aufruft.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S11-Mod2ModMethod

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340320523.zip

15.9.1 Methoden zur Verfügung stellendes TwinCAT 3 C++ Modul

Dieser Artikel beschreibt die Erstellung eines TwinCAT 3 C++ Moduls, das eine Schnittstelle mit einigen Methoden zur Verfügung stellt, die von einer SPS, aber auch von anderen C++ Modulen aufgerufen werden kann.

Die Idee besteht darin, eine einfache Zustandsmaschine in C++ zu erstellen, die von anderen Modulen von außen gestartet und gestoppt werden kann, aber auch das Setzen oder Lesen des bestimmten Zustands der C++ Zustandsmaschine ermöglicht.

Zwei weitere Artikel nutzen das Ergebnis dieser C++ Zustandsmaschine.

- Aufruf der Funktionalität von der SPS Logik her [▶ 275] - also von der SPS auf C++ Code einwirken.
- Aufruf der Funktionalität von der C++ Logik [▶ 302] - also Wechselwirkung zwischen zwei C++ Modulen.

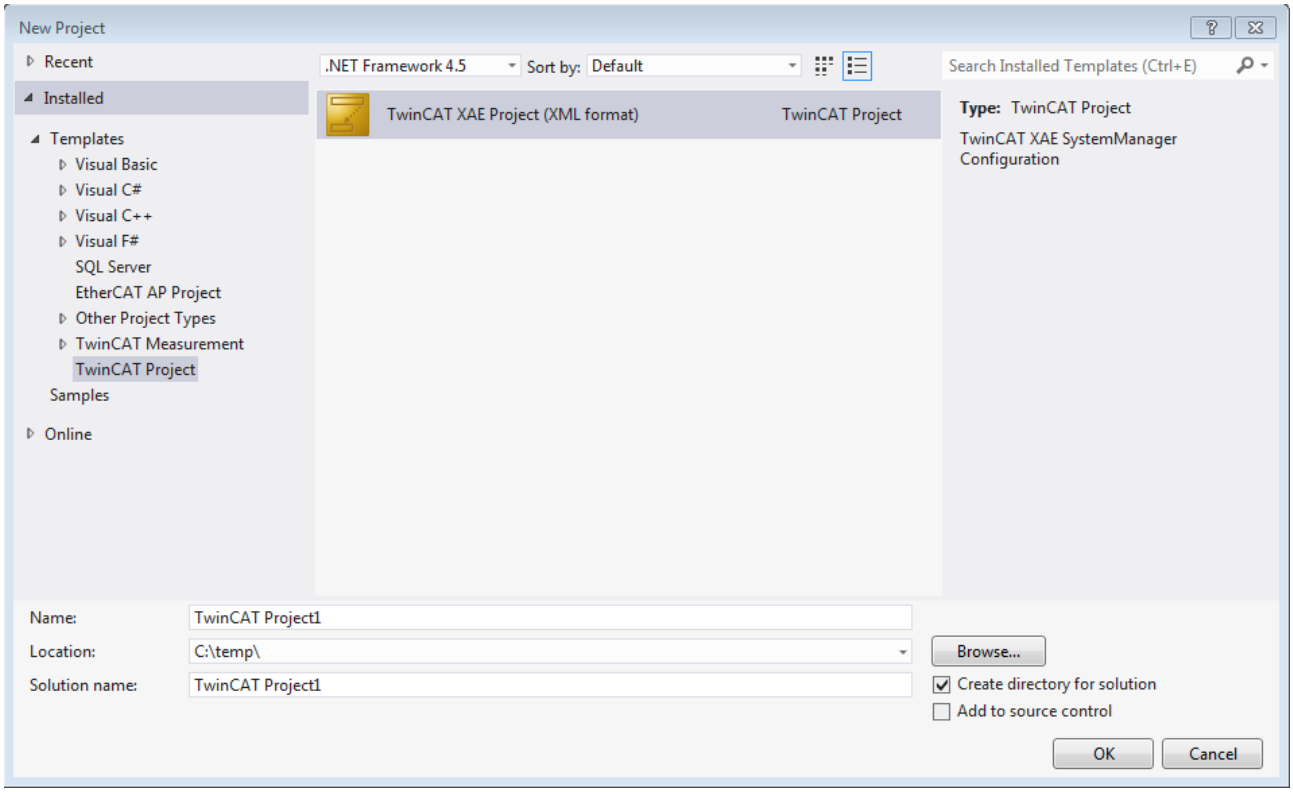
Dieser Artikel beschreibt:

- Schritt 1: Erstellen Sie ein neues TwinCAT 3 Projekt [▶ 277].
- Schritt 2: Erstellen Sie einen neuen TwinCAT 3 C++ Treiber [▶ 277].
- Schritt 3: Erzeugen Sie eine neue TwinCAT 3 Schnittstelle.
- Schritt 4: Fügen Sie der Schnittstelle Methoden hinzu [▶ 280].
- Schritt 5: Fügen Sie eine neue Schnittstelle zum Modul hinzu.
- Schritt 6: Starten Sie den TwinCAT TMC Code Generator, um einen Code für die Modulklassenbeschreibung zu erzeugen [▶ 285].
- Schritt 7: Implementieren Sie die Membervariablen und den Konstruktors.
- Schritt 8: Implementieren Sie Methoden.
- Schritt 9: Implementieren Sie zyklische Aktualisierung.
- Schritt 10: Kompilieren Sie Code kompilieren [▶ 288]
- Schritt 11: Erstellen Sie eine Instanz des C++ Moduls.

- Schritt 12: Fertig, überprüfen Sie die Ergebnisse.

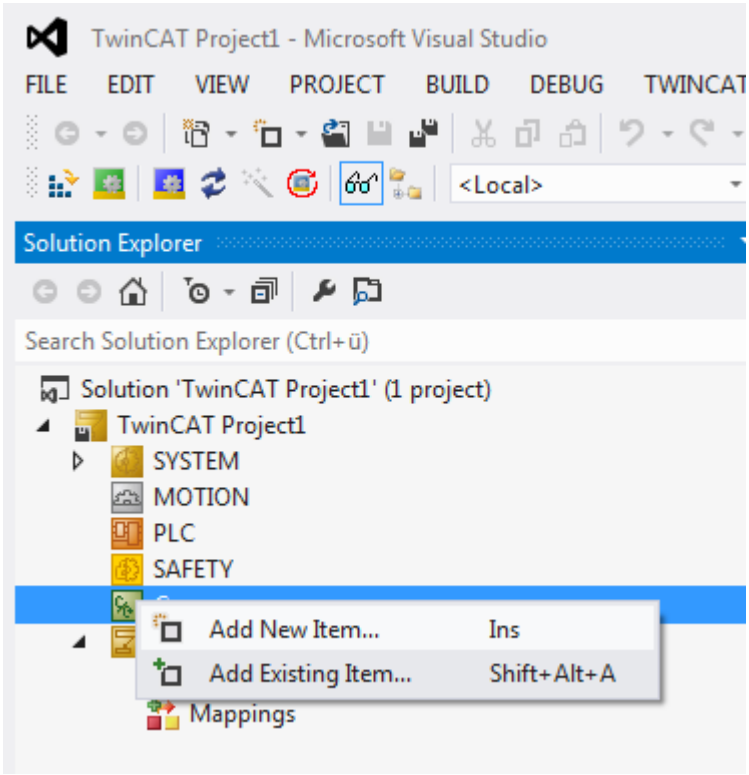
Schritt 1: TwinCAT 3 Projekt erstellen

Als erstes legen Sie wie gewohnt ein TwinCAT Projekt an.

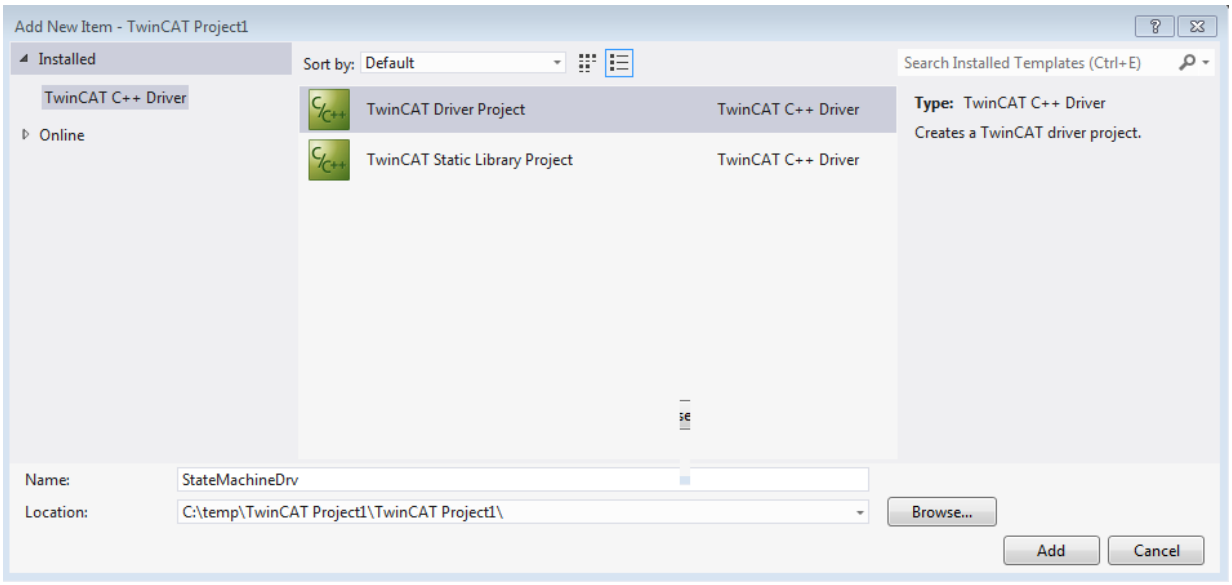


Schritt 2: Erstellen Sie einen neuen TwinCAT 3 C++ Treiber

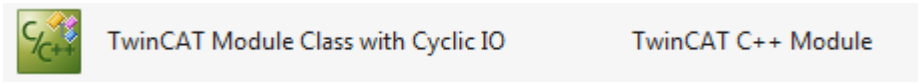
1. Klicken Sie mit der rechten Maustaste auf **C++** und **Add New Item...**



2. Wählen Sie die Vorlage TwinCAT Driver Project und geben einen Treibernamen ein, „StateMachineDrv“ in diesem Beispiel. Klicken Sie auf **Add** um fortzufahren.

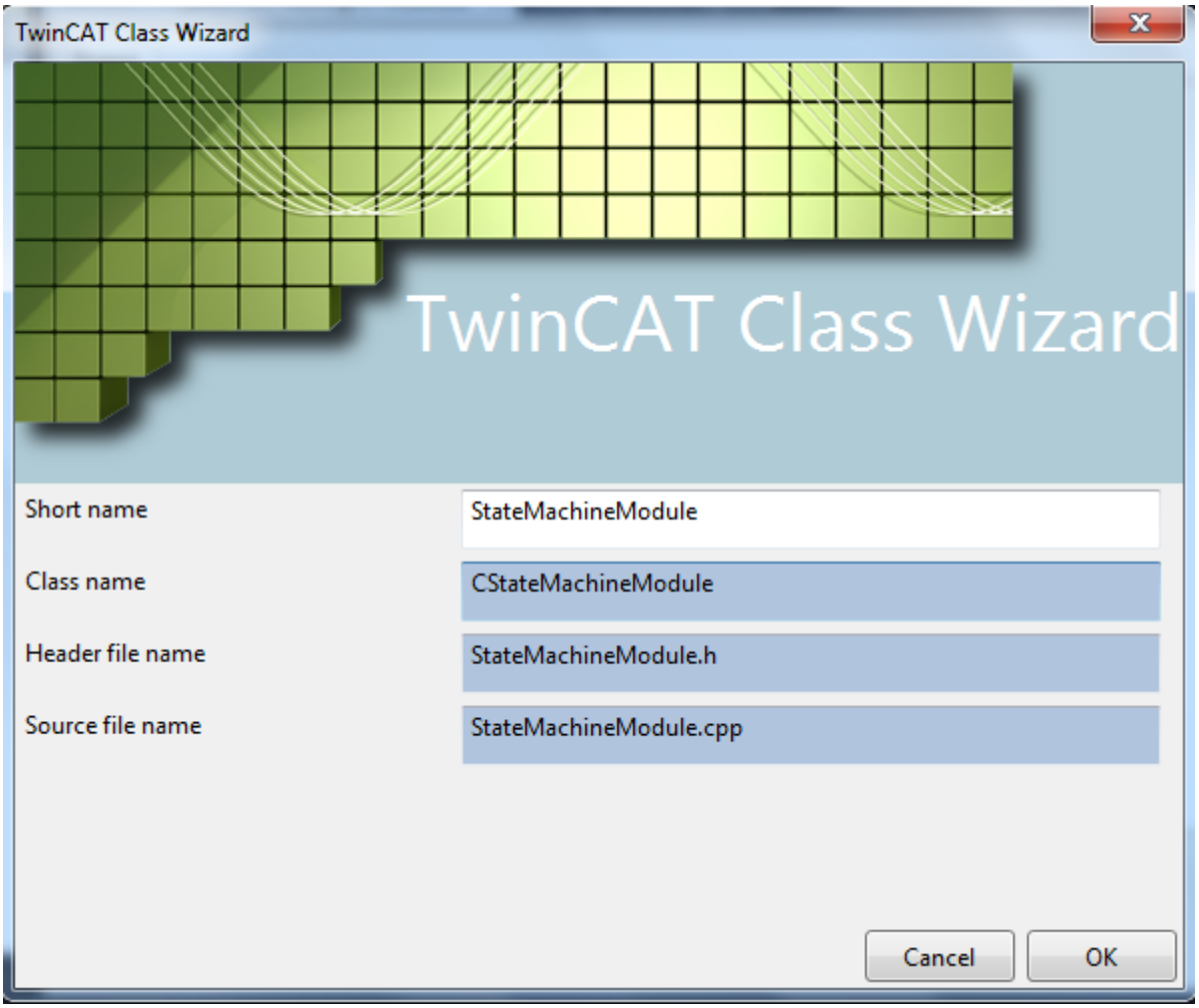


3. Wählen Sie eine für diesen neuen Treiber zu verwendende Vorlage. In diesem Beispiel ist „TwinCAT Module Class with Cyclic IO“ gewählt, da der interne Zähler der Zustandsmaschine verfügbar ist, um den IO zugeordnet zu werden.
4. Klicken Sie auf **Add** um fortzufahren.



5. Geben Sie einen Namen für die neue Klasse im C++ Treiber „StateMachineDrv“ an.
   Aus dem angegebenen „Short Name“ ergeben sich die Namen der Modul-Klasse sowie der Header- und Source-Dateien.

6. Klicken Sie auf **OK** um fortzufahren.



⇨ Daraufhin erstellt der Assistent ein C++ Projekt, das fehlerfrei kompiliert werden kann.

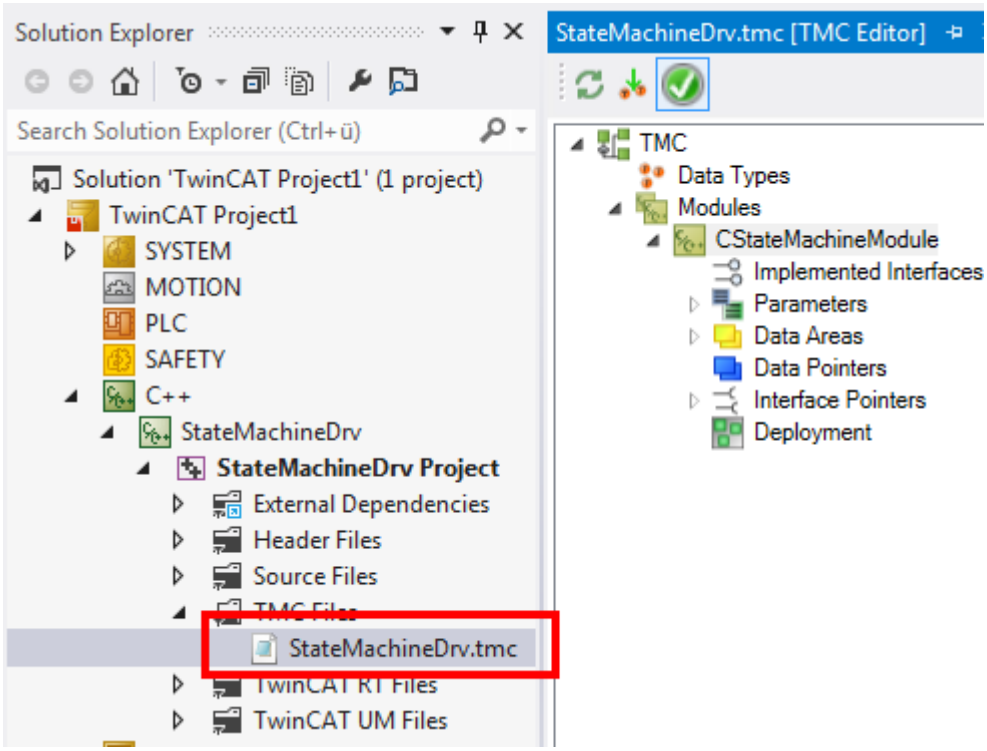
Schritt 3: Ein neues TwinCAT 3 Interface anlegen

HINWEIS

Namenskonflikt

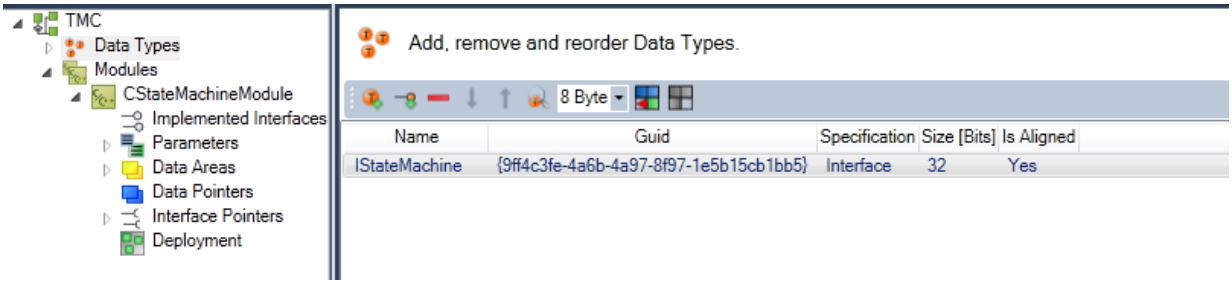
Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.
Verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen.

1. Starten Sie den TMC Editor mittels Doppelklick auf **StateMachineDrv.tmc**.



2. Wählen Sie innerhalb des TMC Editors **Data Types.**

3. Fügen Sie eine neue Schnittstelle hinzu durch Klick auf **Adds a new interface**  .
   ⇨ Daraufhin wird ein neuer Eintrag **IInterface1** aufgeführt.
4. Öffnen Sie **IInterface1** mit Doppel-Klick, um die Eigenschaften des Interfaces zu verändern.
5. Geben Sie einen aussagekräftigeren Namen ein - in diesem Beispiel „IStateMachine“.

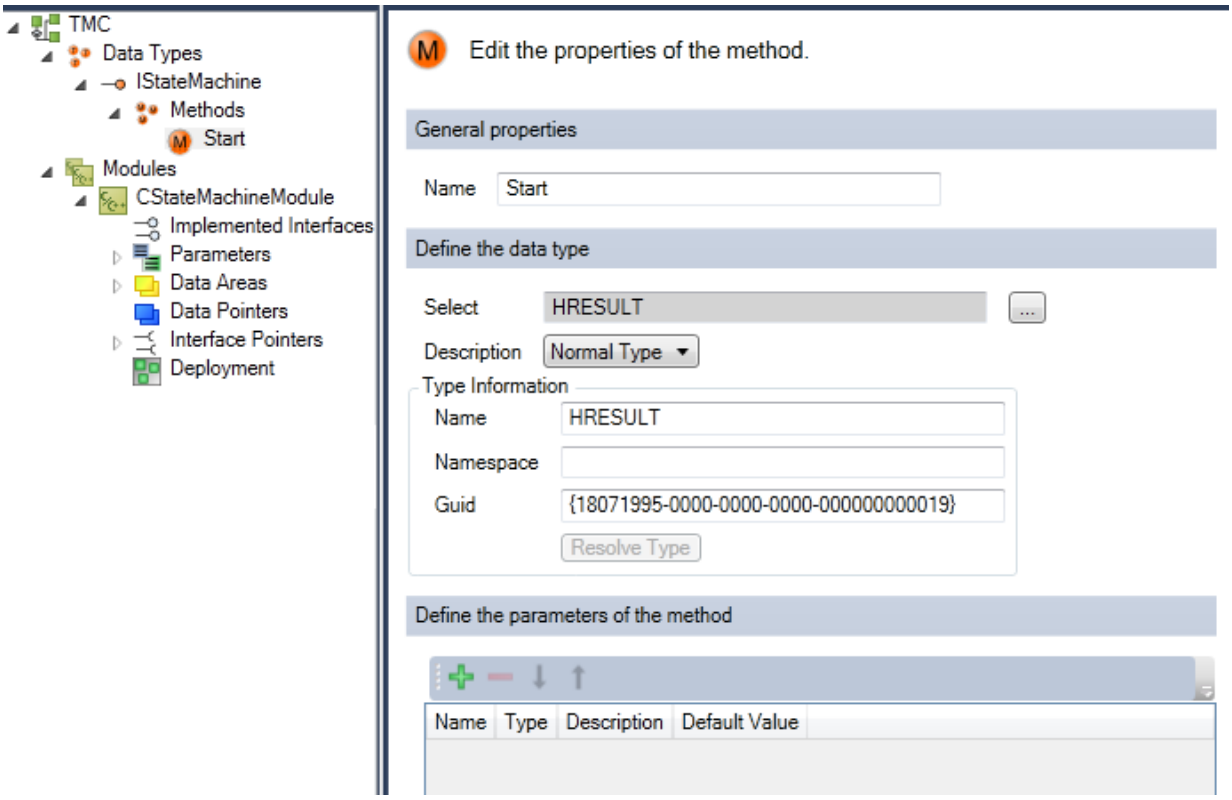


⇨ Das Interface ist angelegt.

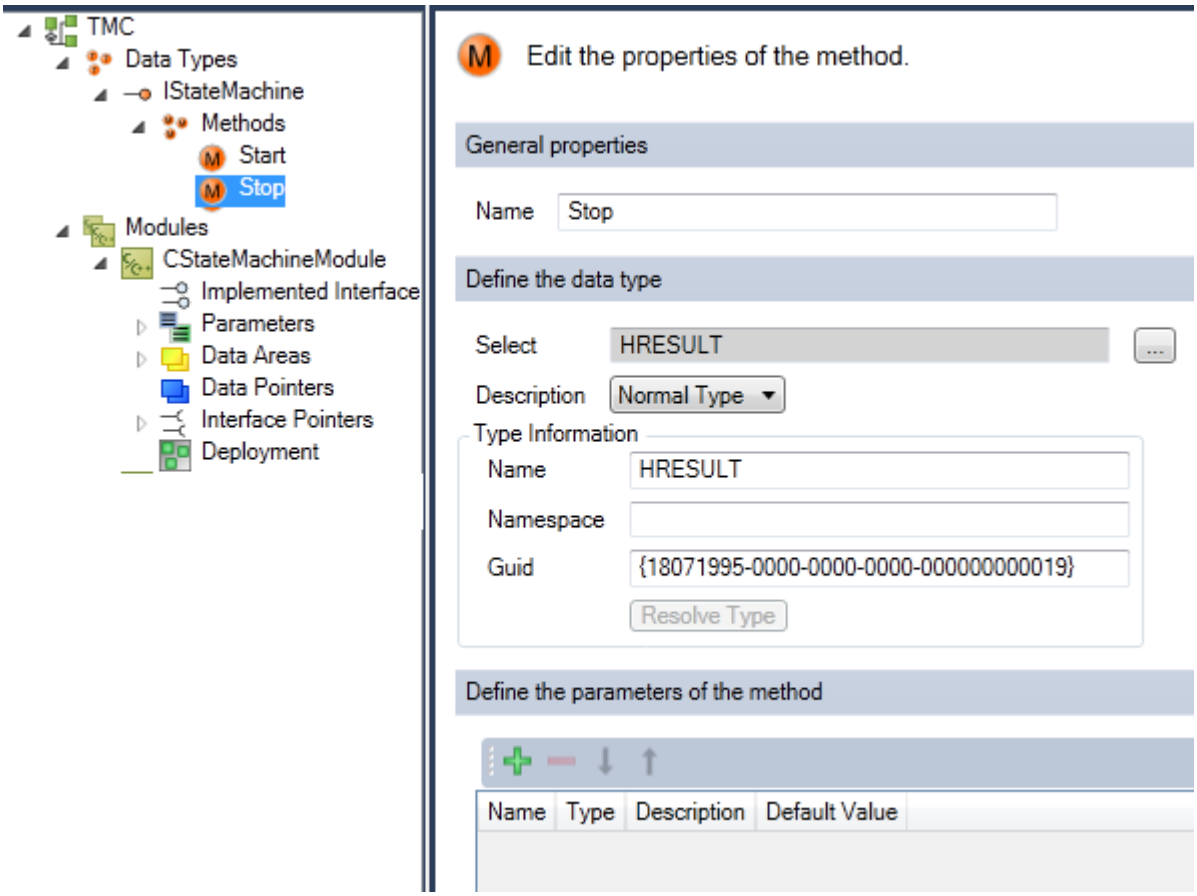
Schritt 4: Fügen Sie der Schnittstelle Methoden hinzu

1. Klicken Sie auf **Edit Methods...**, um eine Liste der Methoden dieser Schnittstelle zu erhalten:
   Klicken Sie auf die **+** Schaltfläche, um eine neue standardmäßige Methode Method1 zu erzeugen.

2. Ersetzen Sie den Standardnamen Method1 durch aussagekräftigen Namen, in diesem Beispiel „Start“.

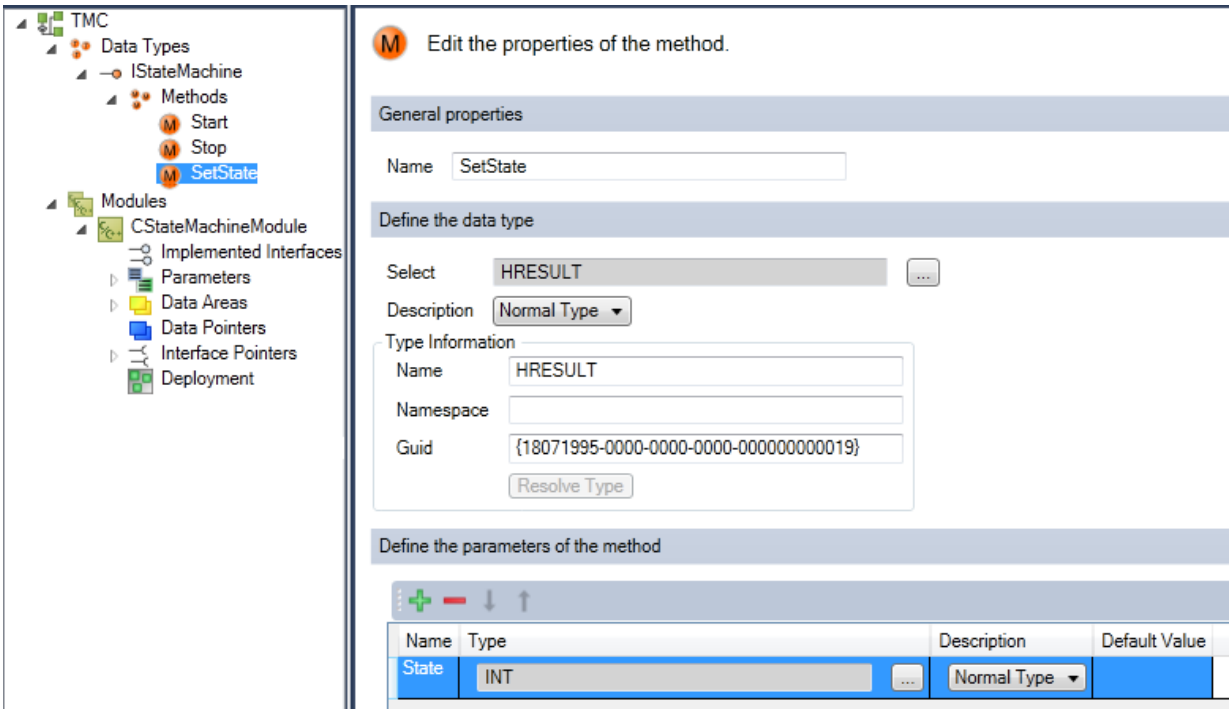


3. Fügen Sie eine zweite Methode hinzu und nennen sie „Stop“.



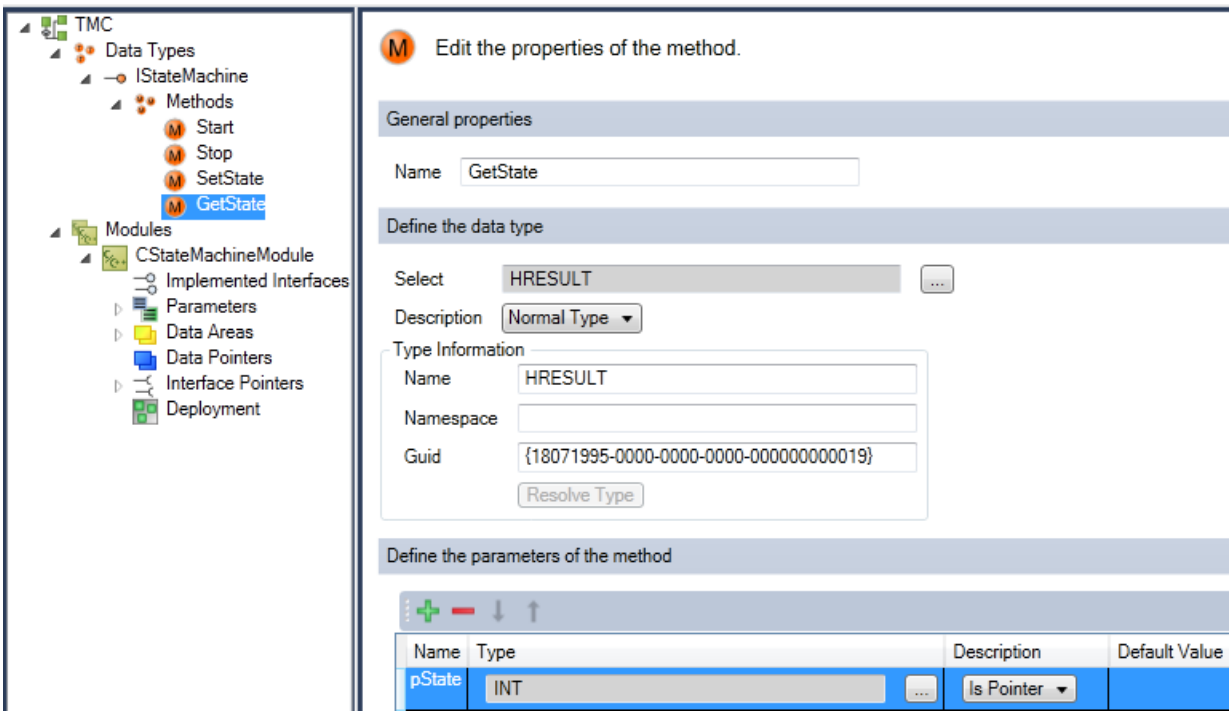
4. Fügen Sie eine dritte Methode hinzu und nennen sie „SetState“.

5. Anschließend können Sie mit einem Klick auf **Add a new parameter** Parameter hinzufügen bzw. Parameter der Methode SetState bearbeiten.

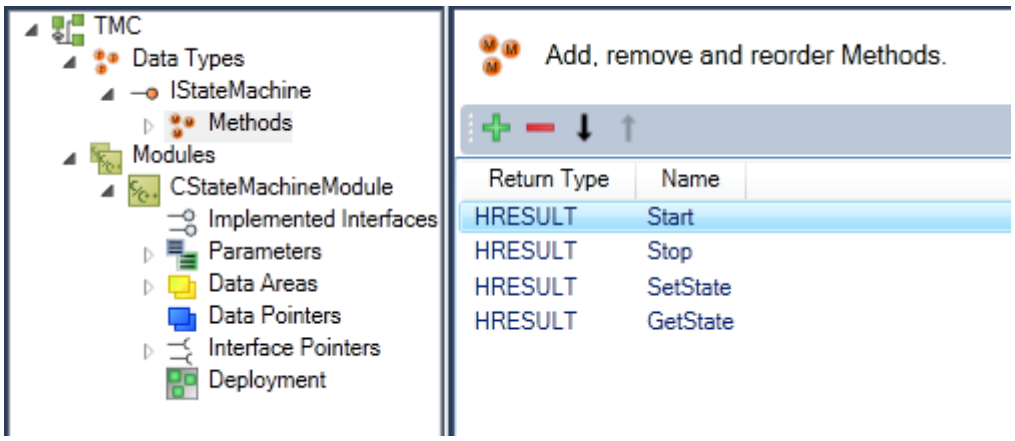


⇨ Standardmäßig wird der neue Parameter Parameter1 als **Normal Type INT** erzeugt.

6. Klicken Sie auf den Namen „Parameter1“, gehen Sie in das Bearbeitungsfeld und ändern Sie den Namen in „State“.
7. Nachdem Start, Stop und SetState definiert sind, definieren Sie eine weitere Methode.
8. Benennen Sie sie in „GetState“ um.
9. Fügen Sie einen Parameter hinzu und nennen ihn „pState“ (ist so konzipiert, um später ein Zeiger zu werden).
10. Ändern Sie Normal Type zu Is Pointer.

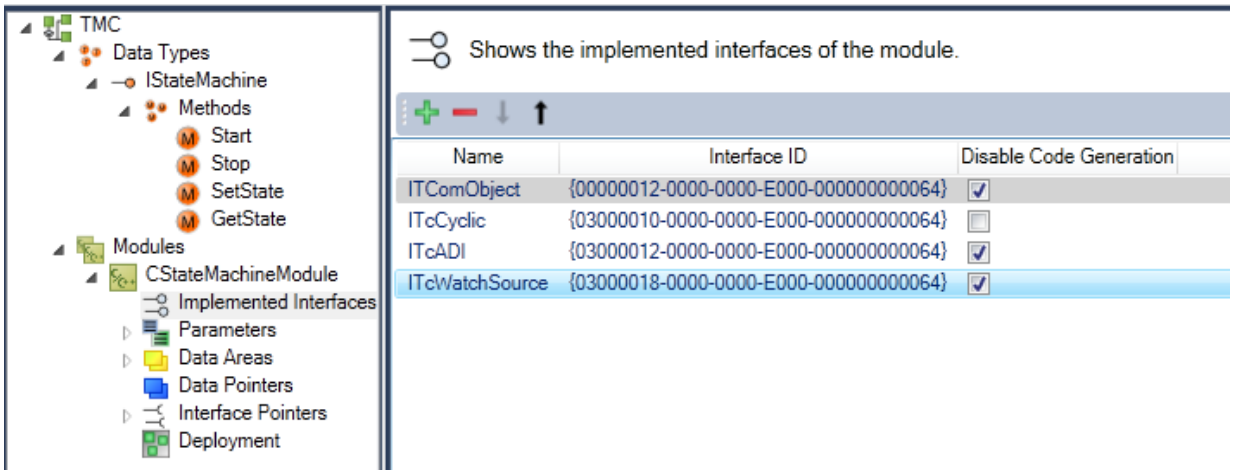


⇨ Daraufhin erhalten Sie eine Liste aller Methoden. Mit den Schaltflächen  können Sie die Reihenfolge der Methoden verändern.

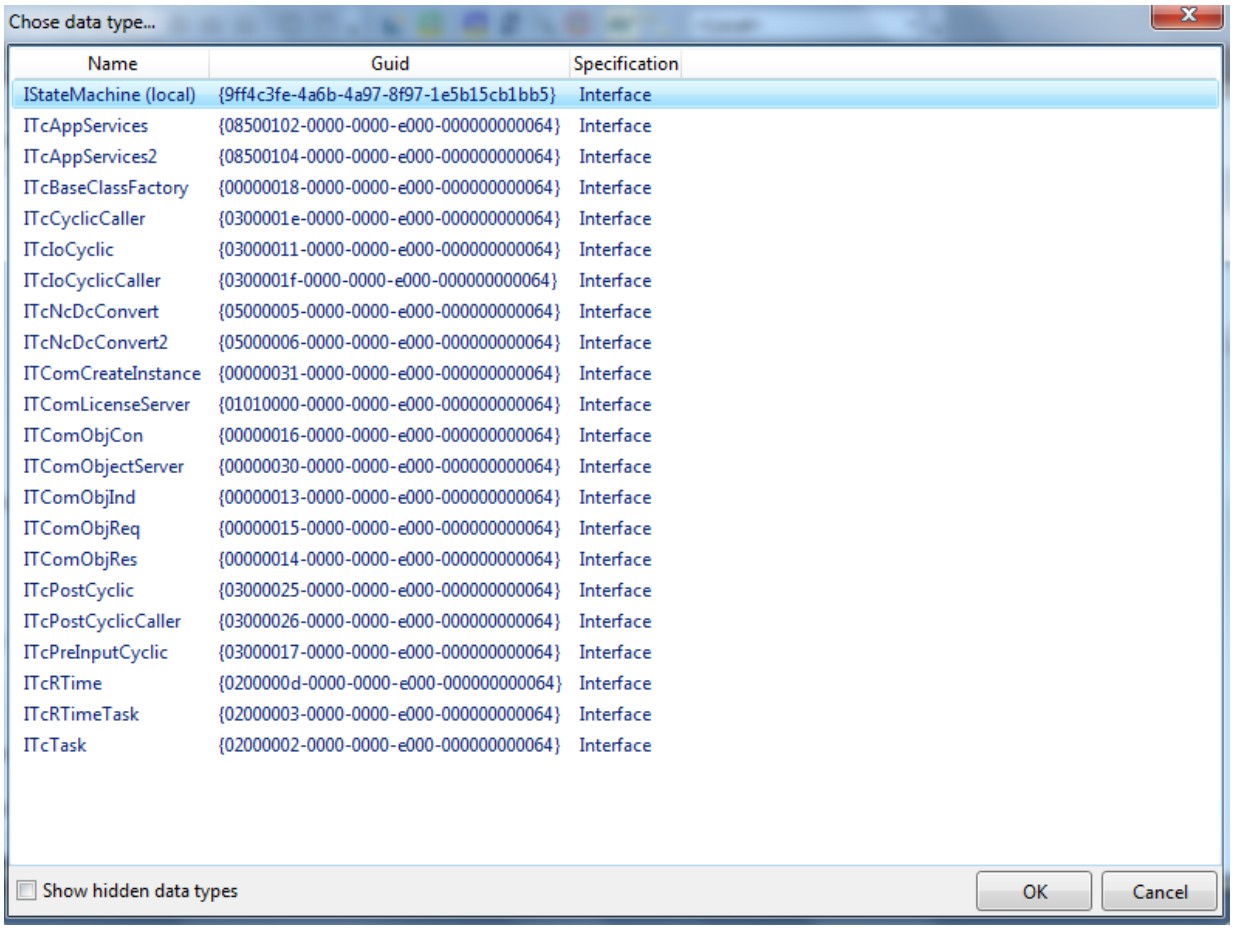


Schritt 5: Neue Schnittstelle zum Modul hinzufügen

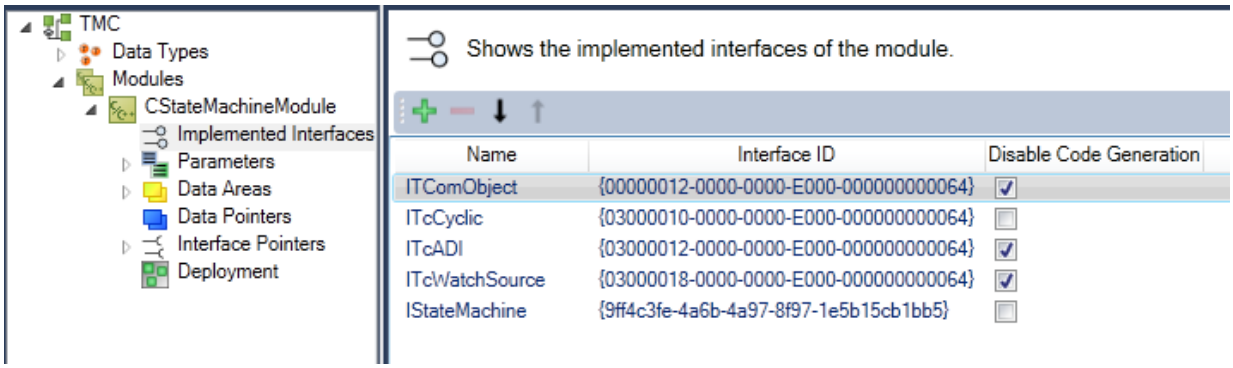
1. Wählen Sie das Modul, das mit der neuen Schnittstelle erweitert werden soll - in diesem Falle wählen Sie das Ziel **Modules->CStateMachineModule**.
2. Erweitern Sie die Liste der implementierten Schnittstellen mit Klick auf die Schaltfläche **+** mit **Add a new interface to the module** um eine neue Schnittstelle.



3. Alle verfügbaren Schnittstellen werden aufgeführt - wählen Sie die neue Schnittstelle IStateMachine und beenden Sie mit **OK.**

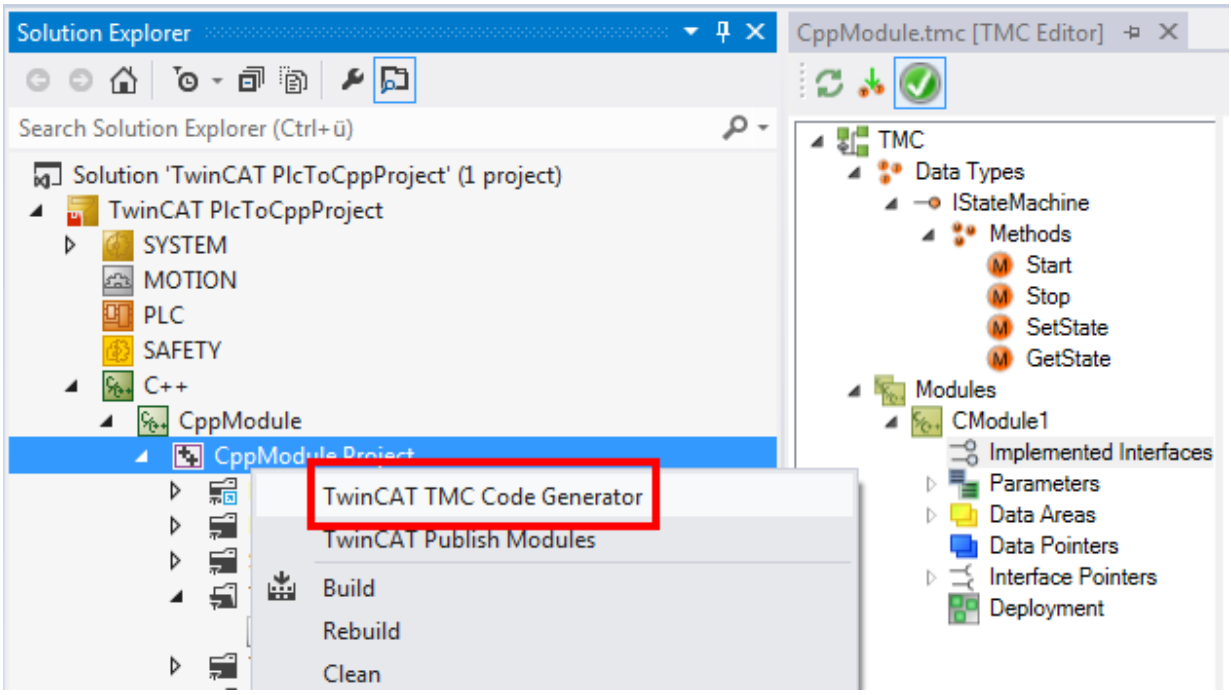


⇨ Die neue Schnittstelle IStateMachine ist Teil der Modulbeschreibung.



Schritt 6: Den TwinCAT TMC Code Generator starten

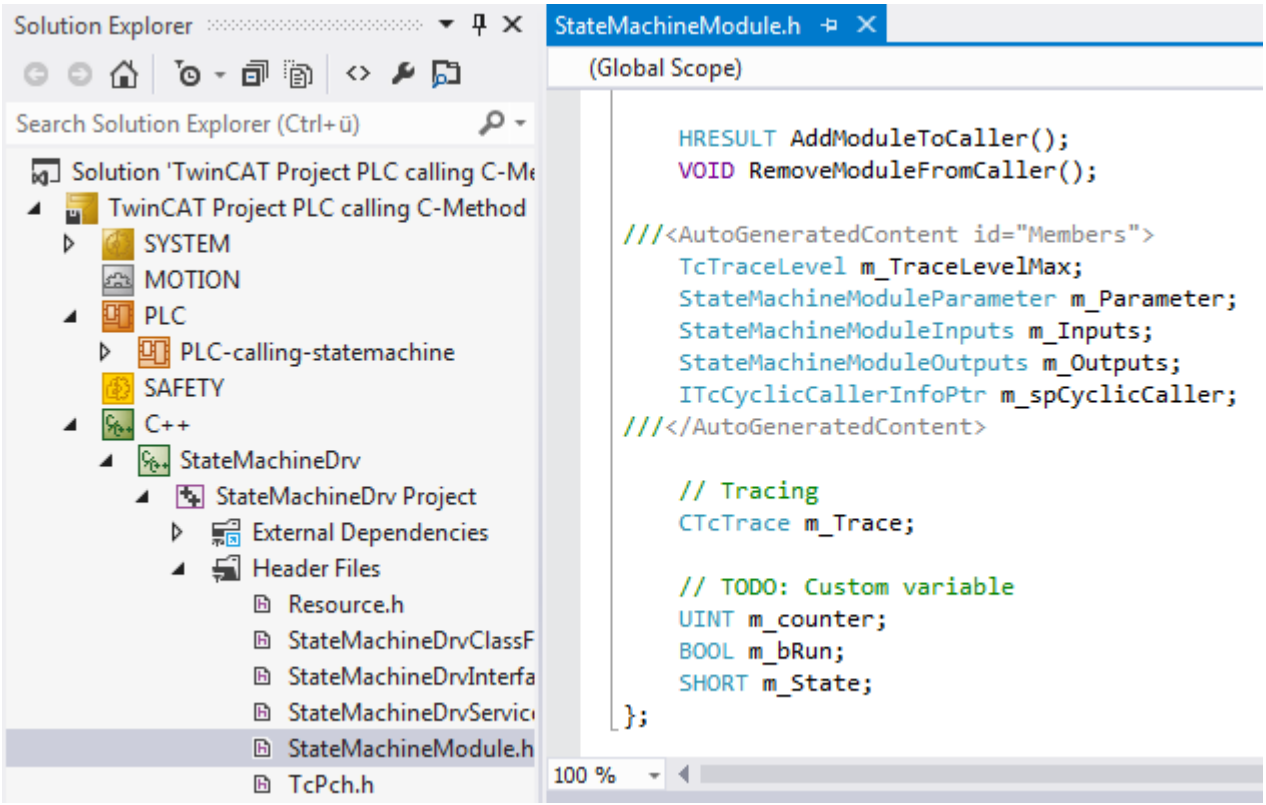
1. Um den C/C++ Code anhand dieses Moduls zu generieren, klicken Sie mit der rechten Maustaste in das C/C++ Projekt und wählen dann den **TwinCAT TMC Code Generator.**



⇨ Nun enthält das Modul StateMachineModule.cpp die neuen Schnittstellen
CModule1: Start()
CModule1: Stop()
CModule1: SetState(SHORT State)
CModule1: GetState(SHORT* pState).

Schritt 7: Implementierung der Membervariablen und des Konstruktors

Fügen Sie die Membervariablen in die Header-Datei StateMachineModule.h hinzu.



Schritt 8: Methoden implementieren

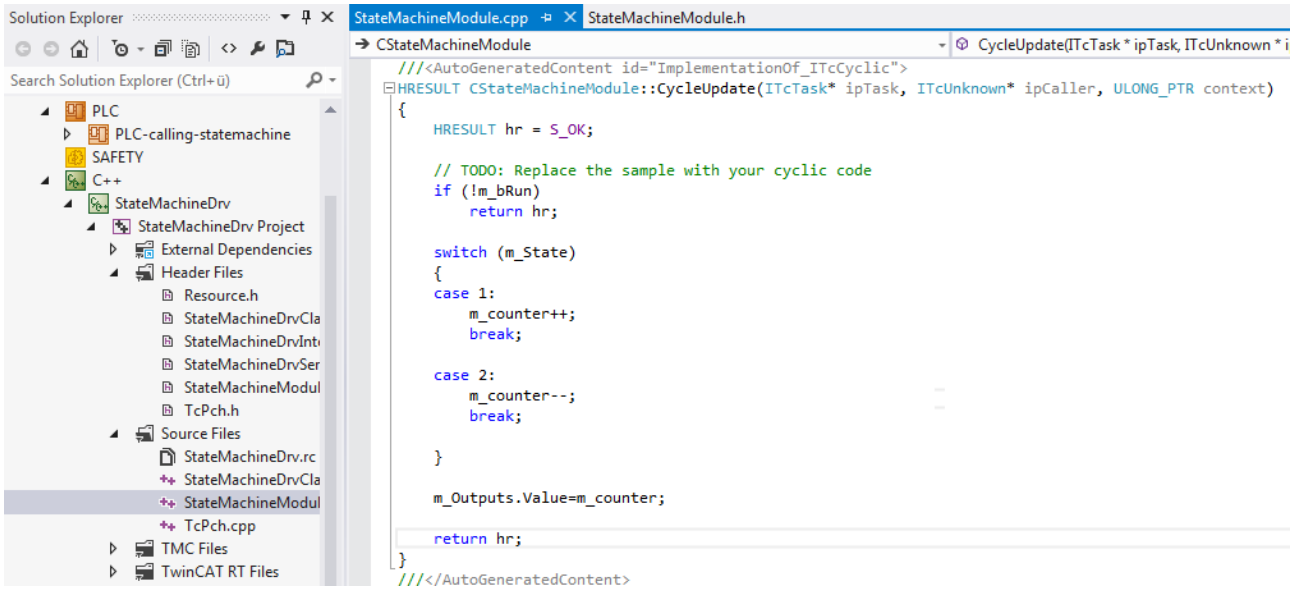
Implementieren Sie den Code für die vier Methoden in der StateMachineModule.cpp:

```
///<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>
```

Schritt 9: Zyklische Aktualisierung implementieren

Die C++ Modulinstanz wird zyklisch aufgerufen - selbst dann, wenn die interne Zustandsmaschine sich im Stop-Modus befindet.

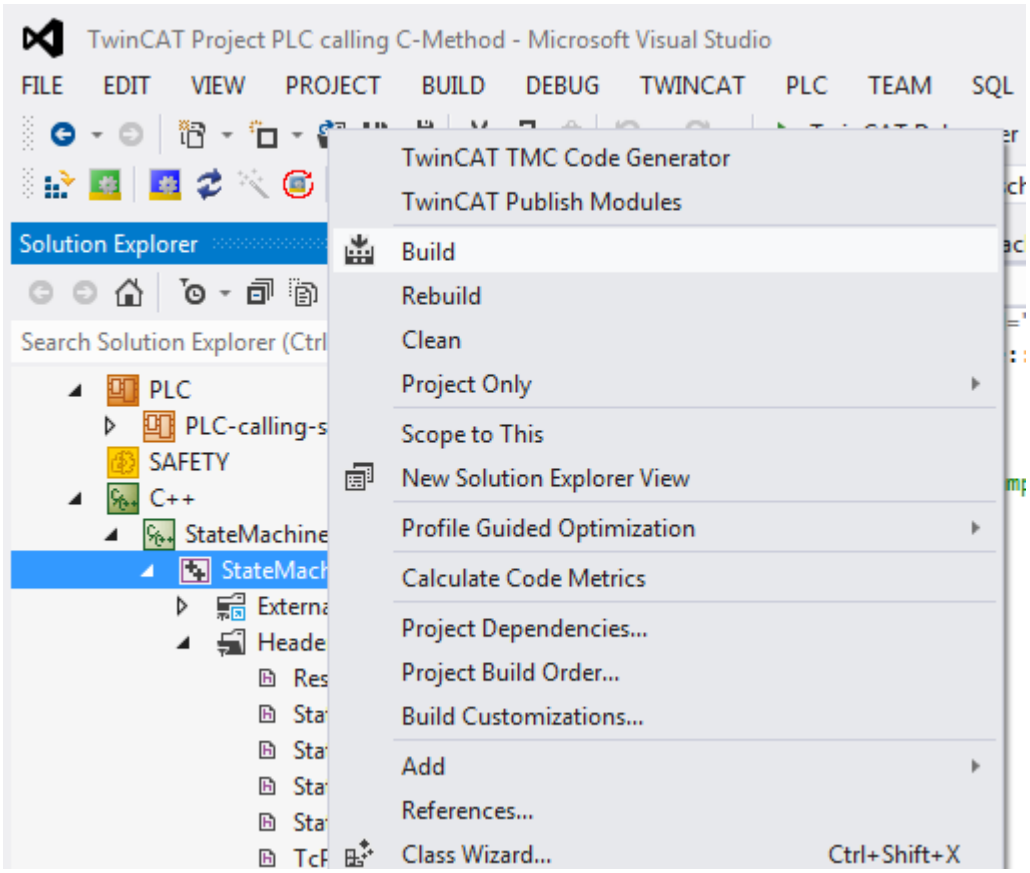
- Wenn die Zustandsmaschine nicht ausgeführt werden soll, dann signalisiert das m_bRun Flag, dass die Codeausführung der internen Zustandsmaschine zu verlassen ist.
- Bei Zustand „1“ muss der Zähler inkrementiert werden.
- Bei Zustand „2“ muss der Zähler dekrementiert werden.
- Der sich daraus ergebende Zählwert wird Value zugewiesen, die Membervariable des logischen Ausgangs des Datenbereichs ist. Dieser kann später der physikalischen IO-Ebene oder anderen Datenbereichen von anderen Modulen zugeordnet werden.



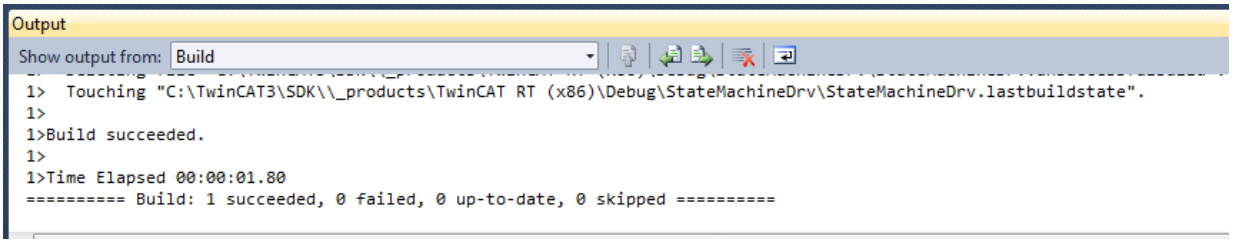
.

Schritt 10: Code kompilieren

1. Nach der Implementierung aller Schnittstellen kompilieren Sie den Code, indem Sie einen Rechtsklick auf die Zustandsmaschine machen und **Build** wählen.

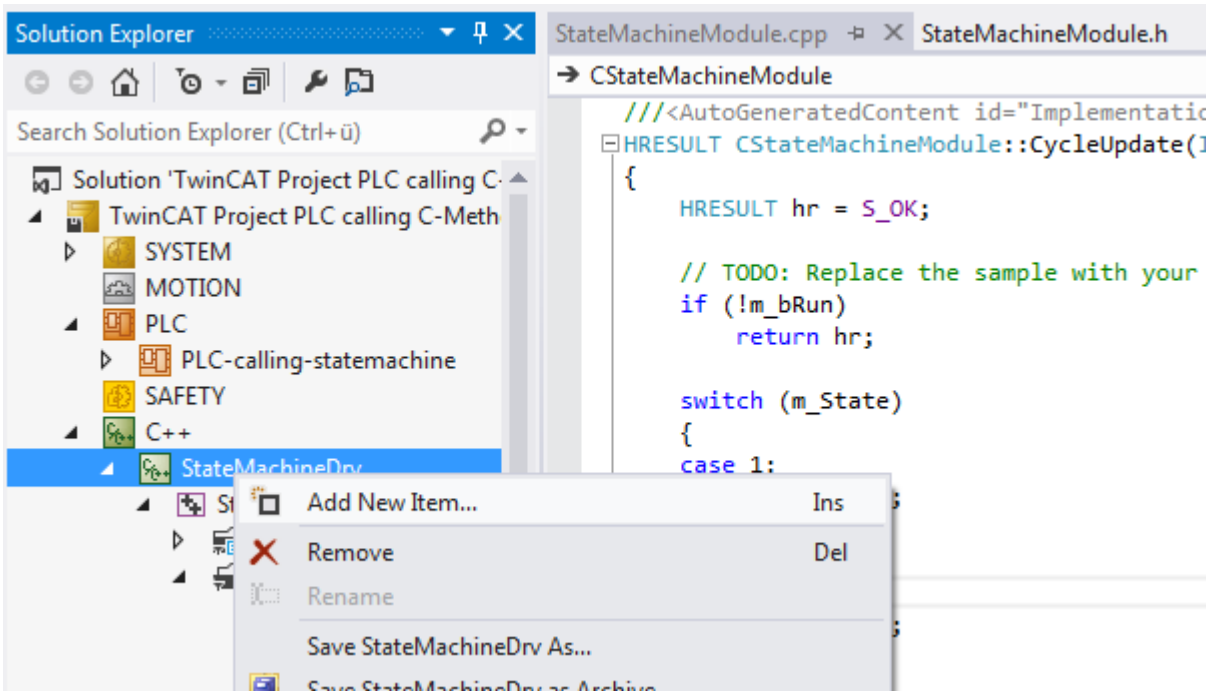


2. Wiederholen Sie die Kompilierung und optimieren Sie Ihren Code so lange, bis das Ergebnis folgendermaßen aussieht:

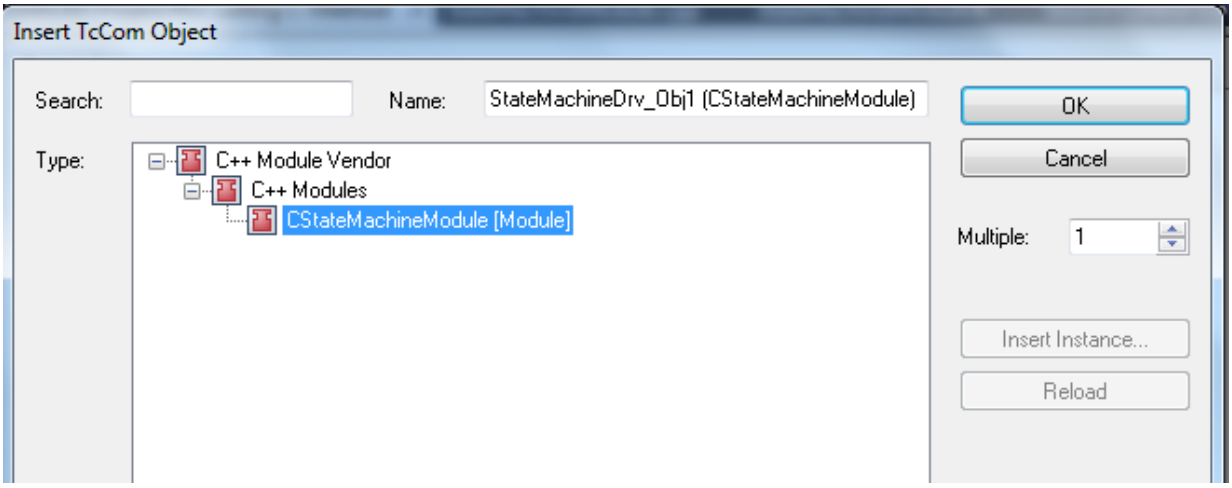


Schritt 11: Eine Instanz des C++ Moduls erstellen

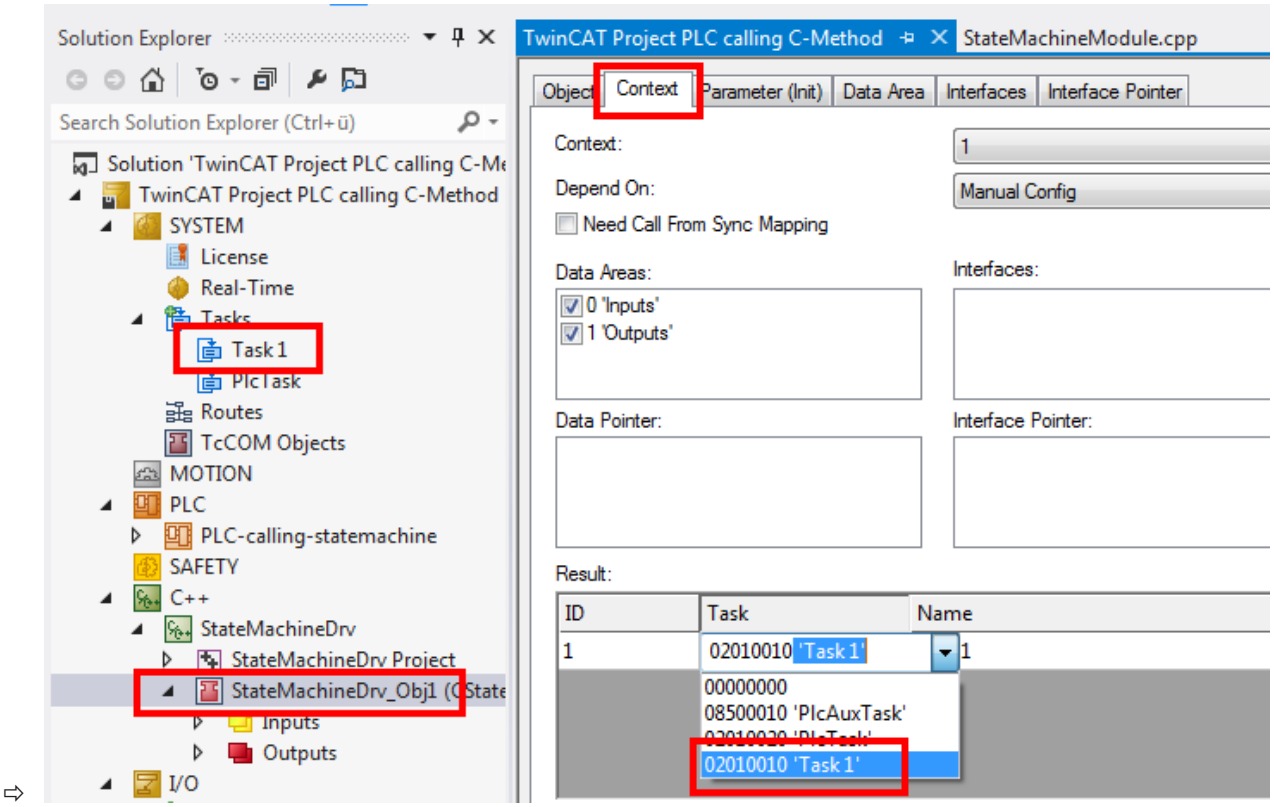
1. Machen Sie einen Rechtsklick auf das C++ Projekt und wählen Sie **Add New Item...,** um eine neue Modulinstanz zu erstellen.



2. Wählen Sie das Modul aus, das als neue Instanz hinzugefügt werden soll - in diesem Fall CStateMachineModule.



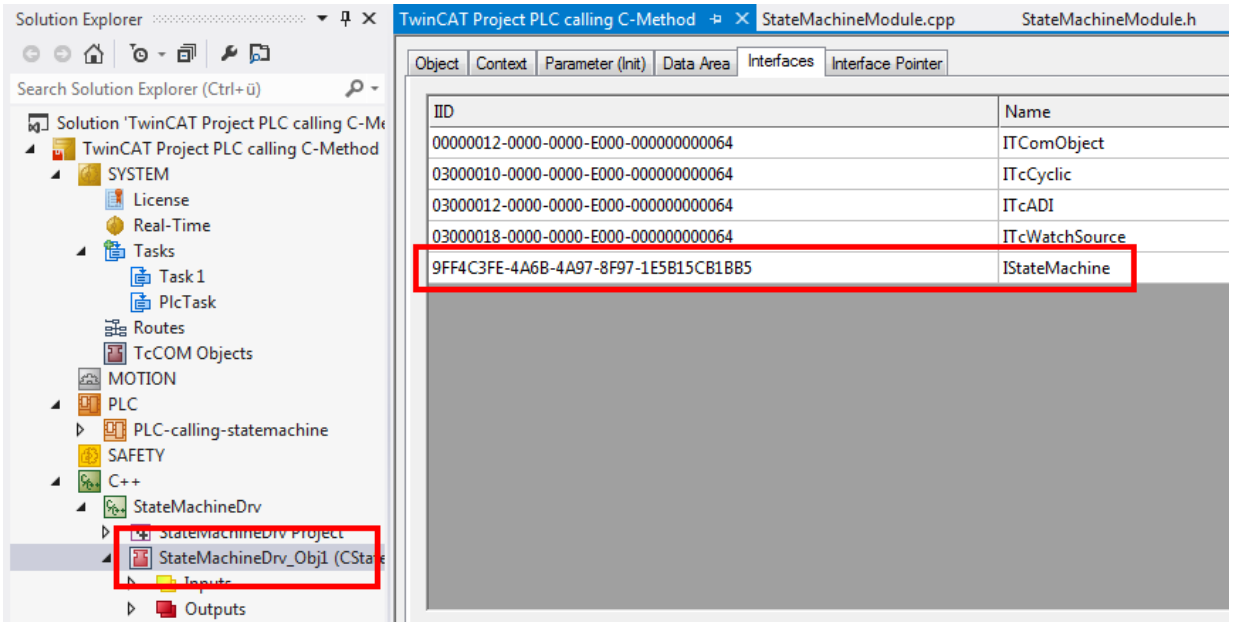
3. Ordnen Sie die Instanz einer Task zu:



Schritt 12: Fertig - Ergebnis überprüfen

1. Navigieren Sie zum neuen, im Solution-Baum aufgeführten, Modul und wählen die Registerkarte **Interfaces** auf der rechten Seite.

⇨ Die neue Schnittstelle IStateMachine ist aufgelistet.



15.9.2 SPS um Methoden aufzurufen, die von einem anderen Modul angeboten werden

Dieser Artikel beschreibt, wie eine SPS eine Methode, die von einem anderen Modul bereitgestellt wird, aufrufen kann - hier: das zuvor definierte C++ Modul.

- Schritt 1: Überprüfen Sie die verfügbare Schnittstellen.

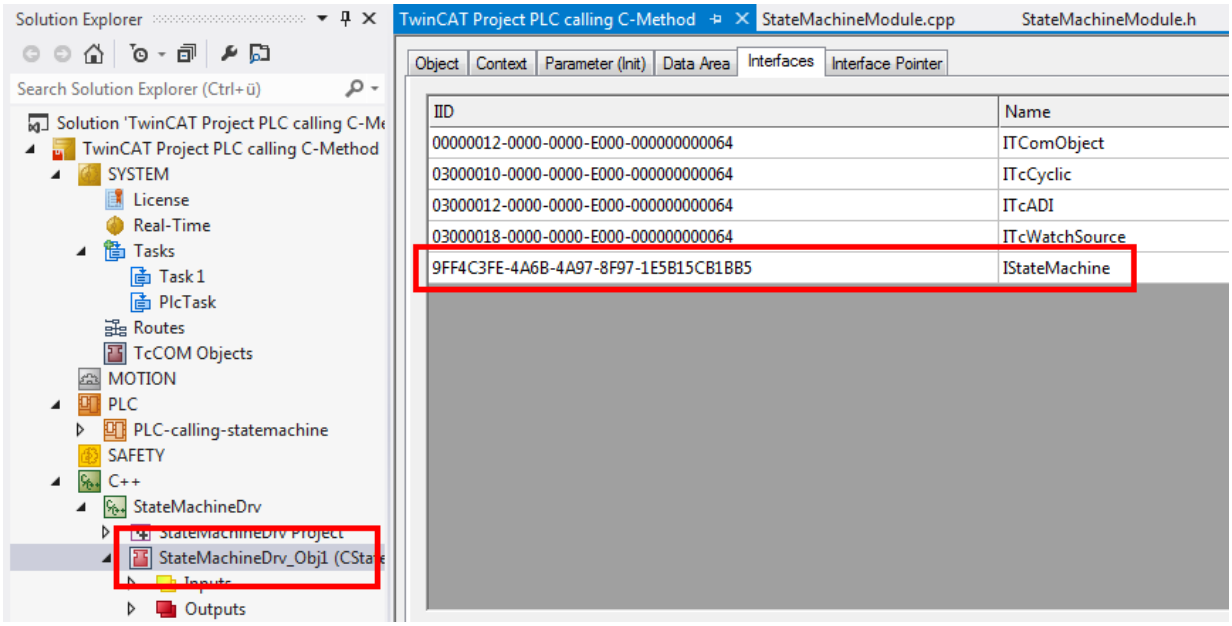
- Schritt 2: Erstellen Sie ein neues SPS-Projekt.
- Schritt 3: Fügen Sie eine neue FB-StateMachine hinzu [▶ 292] (die als der C++ Modulmethoden aufrufende Proxy fungiert).
- Schritt 4: Säubern Sie die Funktionsbaustein-Schnittstelle.
- Schritt 5: Fügen Sie die FB-Methoden „FB_init" und „exit" hinzu [▶ 296].
- Schritt 6: Implementieren Sie die FB-Methoden.
- Schritt 7: Rufen Sie die FB-StateMachine in der SPS auf [▶ 300].
- Schritt 8: Kompilieren Sie den SPS-Code.
- Schritt 9: Verknüpfen Sie die SPS FB mit der C++ Instanz.
- Schritt 10: Beobachten Sie die Ausführung von beiden Modulen, SPS und C++.

Schritt 1: Verfügbare Schnittstellen überprüfen

Option 1:

1. Navigieren Sie zur C++ Modulinstanz.
2. Wählen Sie den Karteireiter **Interfaces** aus.

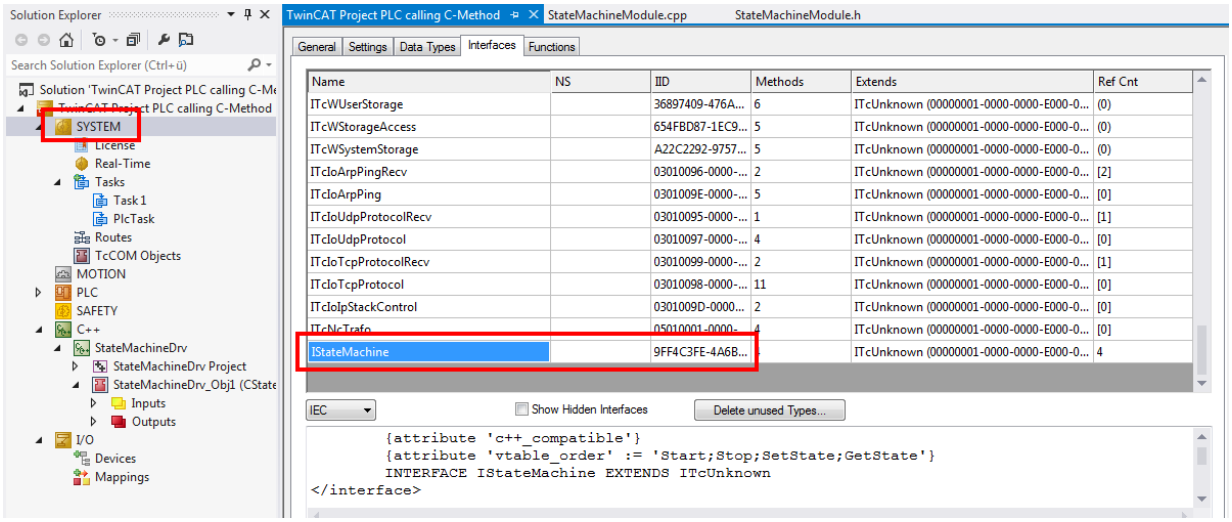
⇨ Die Schnittstelle **IStateMachine** befindet sich in der Liste, mit ihrer spezifischen IID (Interface ID).



Option 2:

1. Navigieren Sie zu **System.**
2. Wählen Sie den Karteireiter **Interfaces** aus.

⇨ Die Schnittstelle **IStateMachine** befindet sich in der Liste, mit ihrer spezifischen IID (Interface ID).

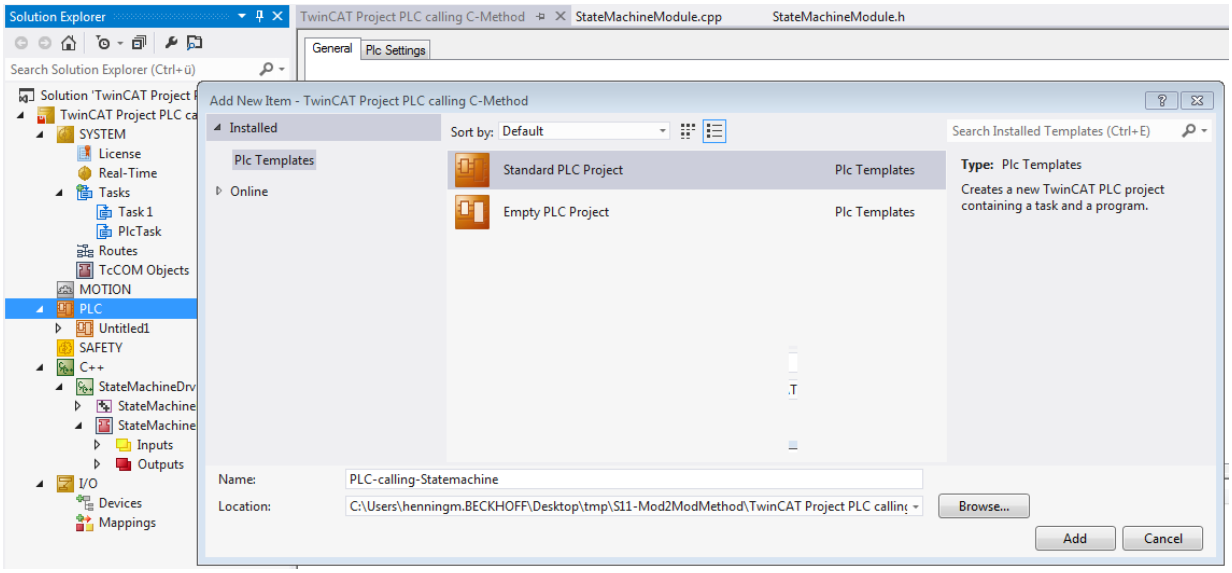


Im unteren Bereich wird der hinterlegte Code in unterschiedlichen Programmiersprachen dargestellt.

Schritt 2: Neues SPS Projekt anlegen

Es wird ein Standard SPS-Project, genannt „PLC-calling-Statemachine“, angelegt.

1. Machen Sie einen Rechts-Klick auf den SPS Knoten.
2. Wählen Sie **Standard PLC Project** aus.
3. Passen Sie den Namen an.

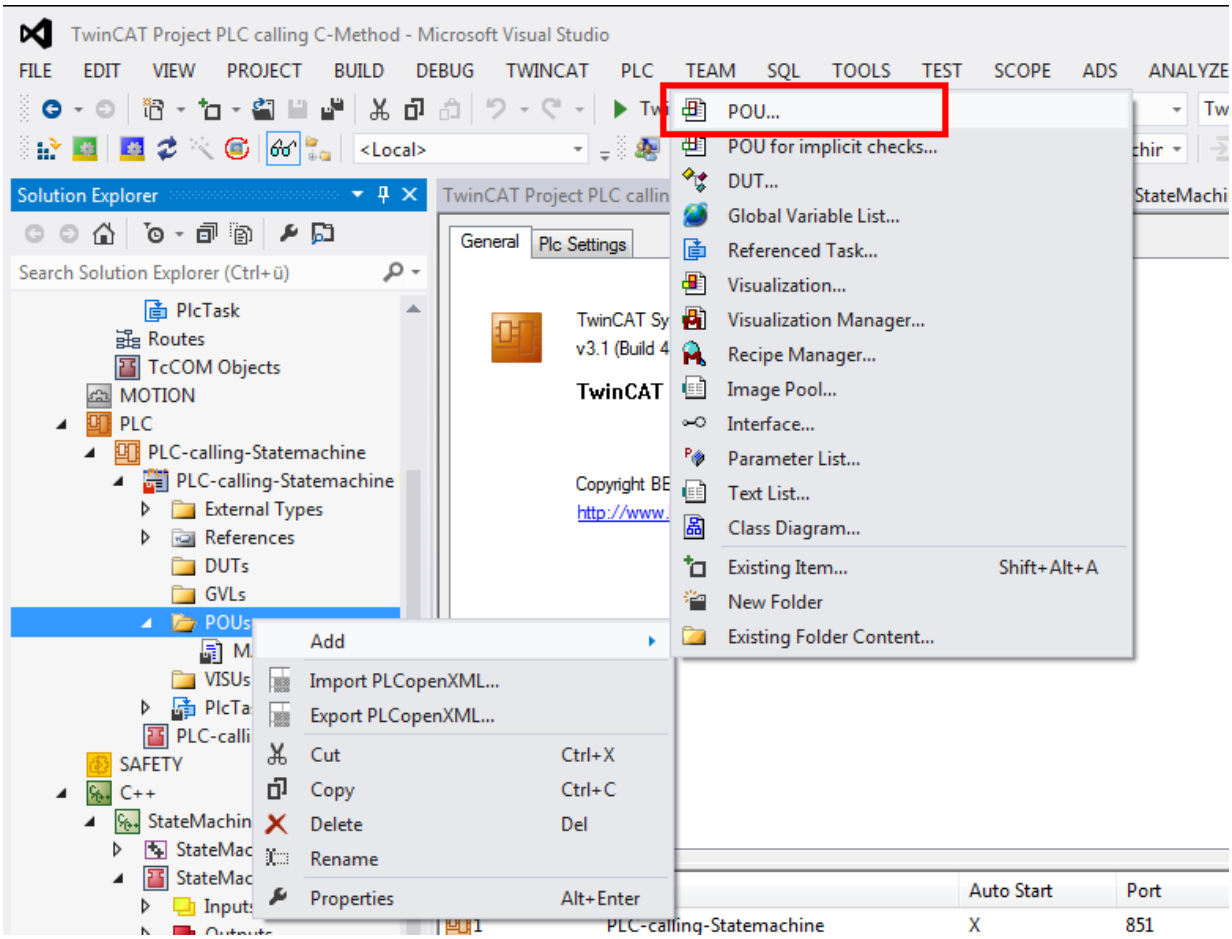


⇨ Das Projekt ist erfolgreich angelegt.

Schritt 3: Einen Funktionsbaustein (FB) hinzufügen (der als der C++ Modulmethoden aufrufende Proxy fungiert)

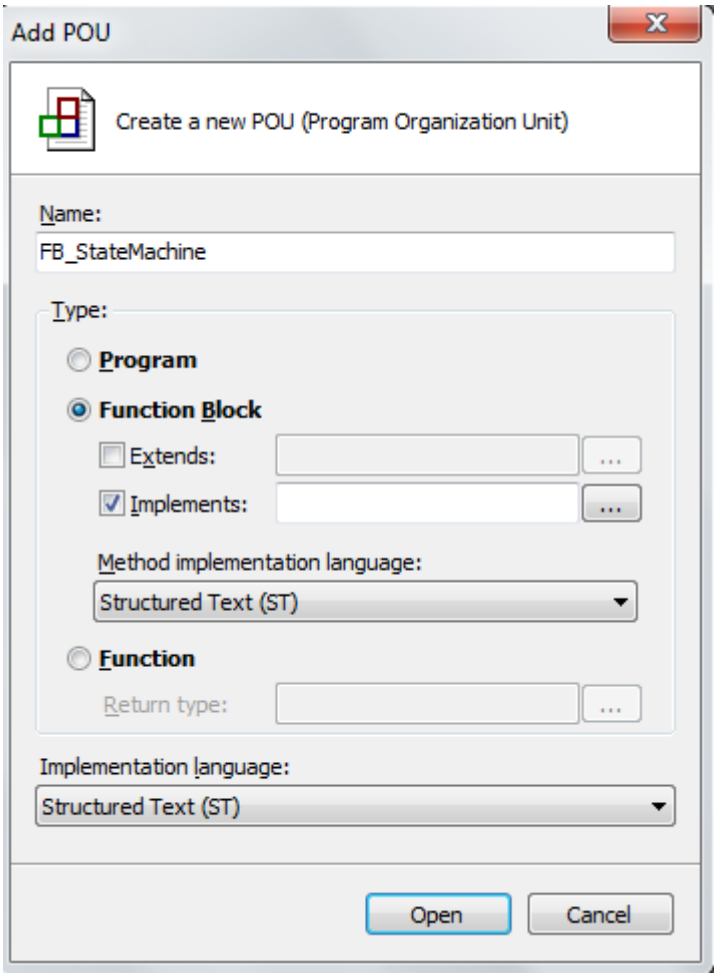
1. Machen Sie einen Rechts-Klick auf **POUs**.

2. Wählen Sie **Add->POU....**

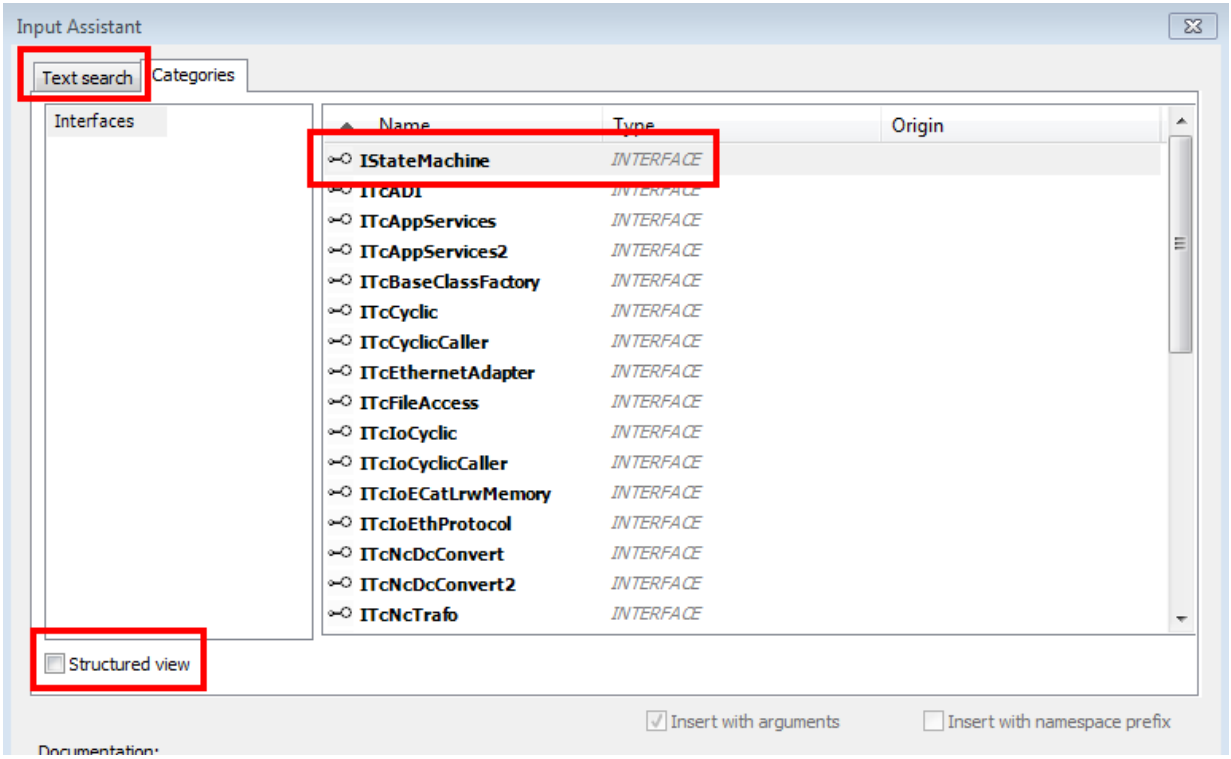


3. Definieren Sie einen neu zu erstellenden FB, der später als Proxy zum Aufrufen von C++ Klasse fungiert: Geben Sie den Namen des neuen FB ein: „FB_StateMachine“.

4. Wählen Sie **Function Block**, dann **Implements** und klicken dann die Schaltfläche **...** an.



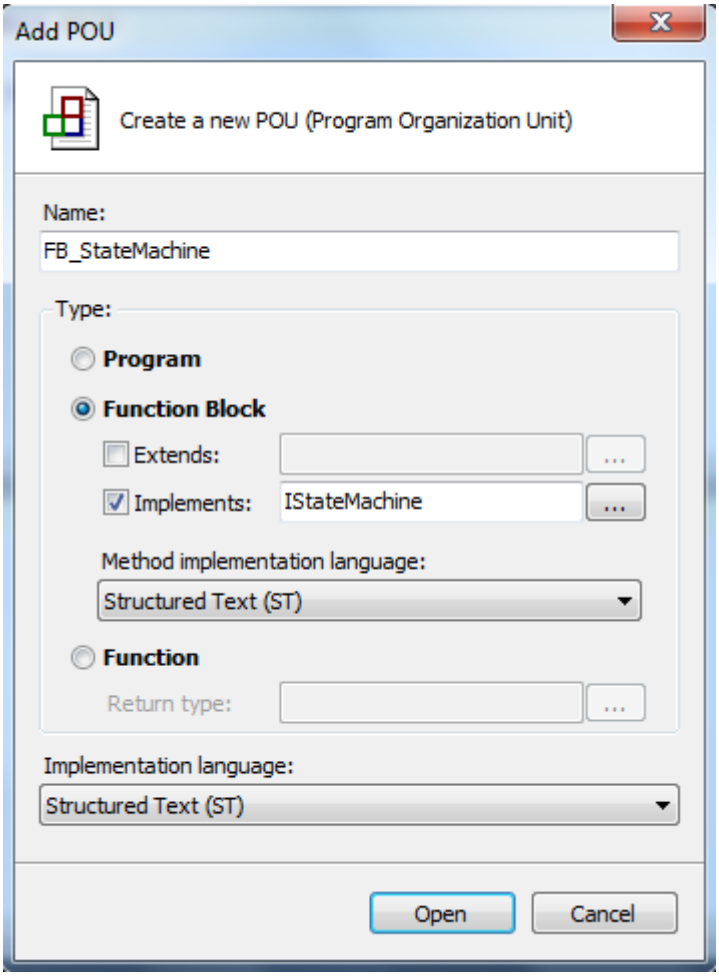
5. Wählen Sie die Schnittstelle entweder über den Karteireiter **Text Search** oder über den Karteireiter **Categories**, indem Sie die **Structured View** abwählen.



6. Wählen Sie **IStateMachine** und klicken Sie auf **OK**.

⇨ Daraufhin wird die Schnittstelle IStateMachine als zu implementierende Schnittstelle aufgelistet.

7. Wählen Sie als **Method implementation language Structured Text (ST)** aus.
8. Wählen Sie als **Implementation language Structured Text (ST)** aus.
9. Beenden Sie diesen Dialog mit **Open**.



⇨ Sie haben den FB erfolgreich hinzugefügt.

Schritt 4: Funktionsbaustein-Schnittstelle anpassen

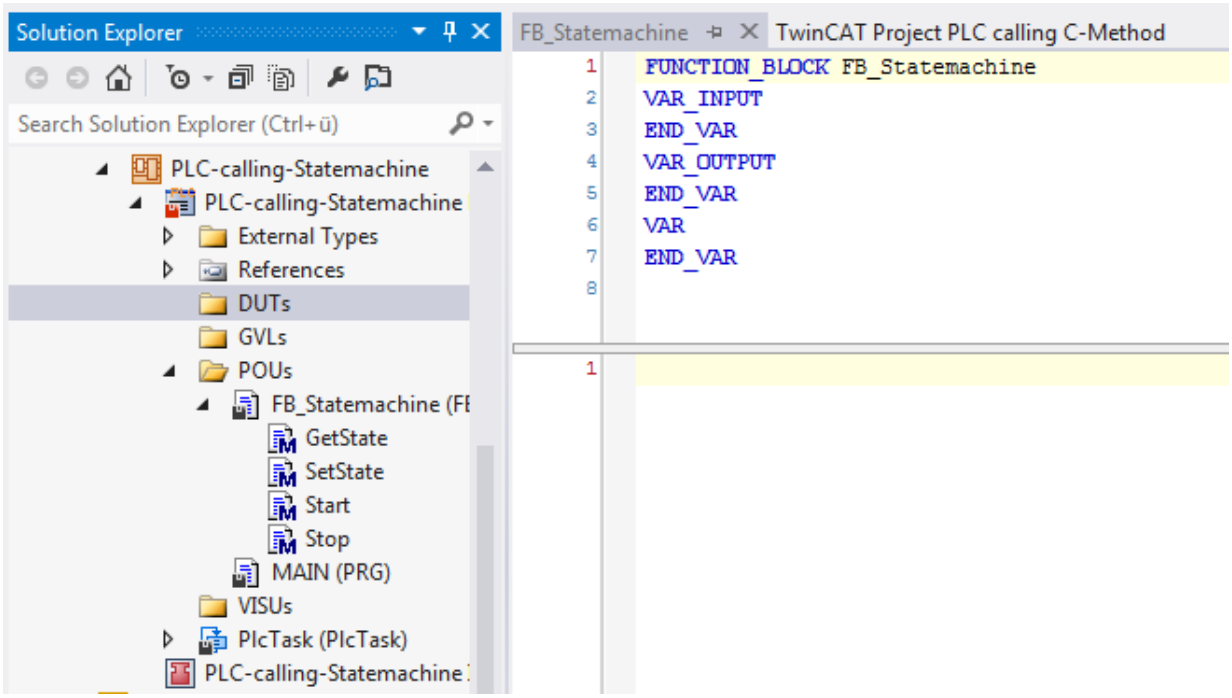
Als Ergebnis der Erstellung eines FB, der die Schnittstelle IStateMachine implementiert, wird der Assistent einen FB mit entsprechenden Methoden erstellen.

Der FB_StateMachine stellt 4 Methoden zur Verfügung:

- GetState
- SetState
- Start
- Stop

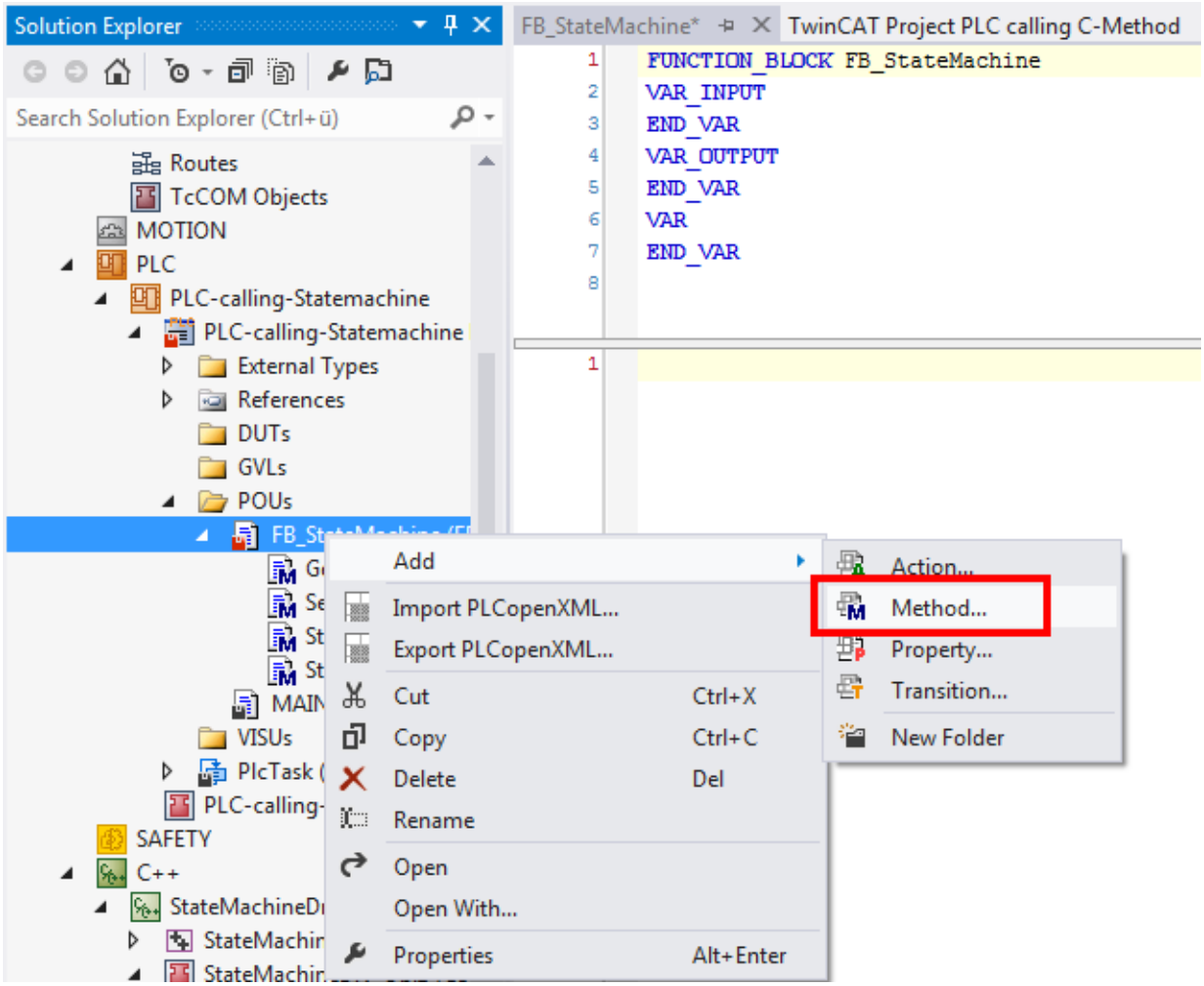
1. Löschen Sie Implements IStateMachine. Da der Funktionsbaustein als Proxy agieren soll, implementiert er selbst nicht das Interface. Also kann er gelöscht werden.
2. Löschen Sie die Methoden TcAddRef, TcQueryInterface und TcRelease. Sie werden für einen Proxy Funktionsbaustein nicht benötigt.

⇨ Es ergibt sich:

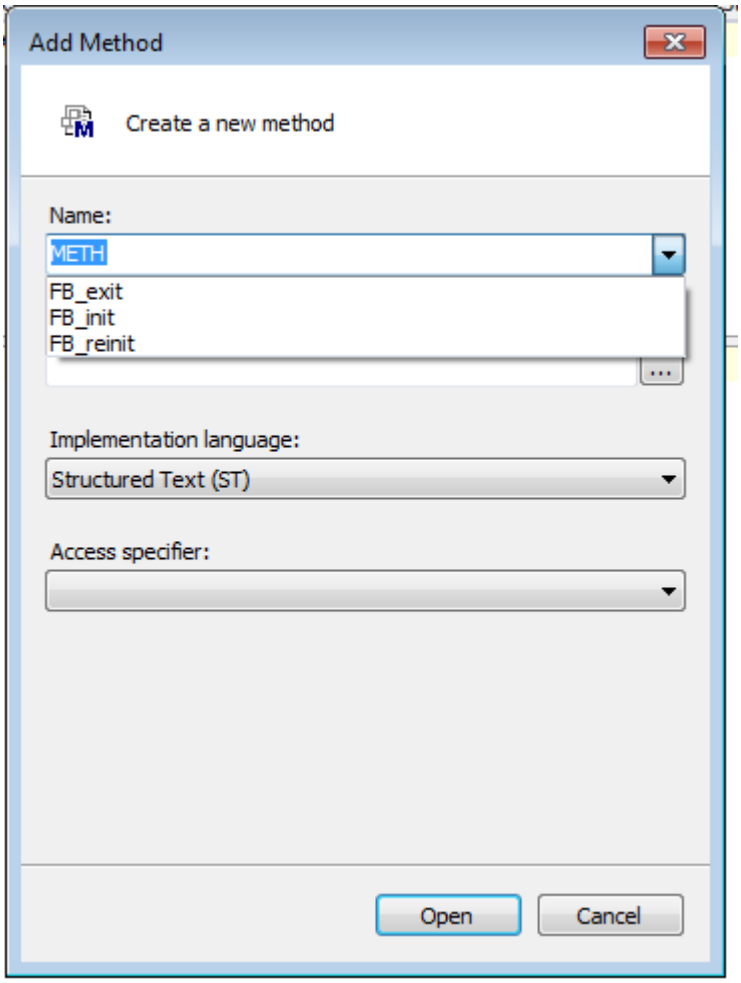


Schritt 5: FB-Methoden FB_init (Konstruktor) und FB_exit (Destruktor) hinzufügen

1. Machen Sie einen Rechtsklick auf **FB_StateMachine** im Baum und wählen Sie **Add / Method…**

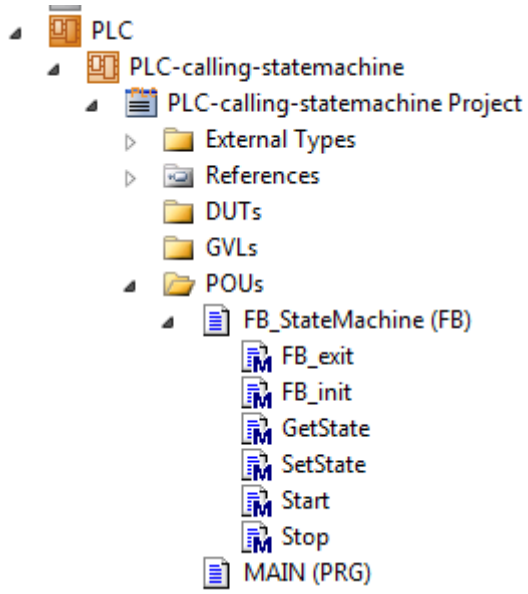


2. Fügen Sie die Methoden FB_exit und FB_init hinzu - beide mit Structured Text (ST) als Implementation language. Sie stehen als vordefinierter Name zur Verfügung.



3. Verlassen Sie den Dialog jeweils mit Klick auf **Open**.

⇨ Zum Schluss stehen alle notwendigen Methoden zur Verfügung:



Schritt 6: FB-Methoden implementieren

Nun müssen alle Methoden mit Code gefüllt werden.

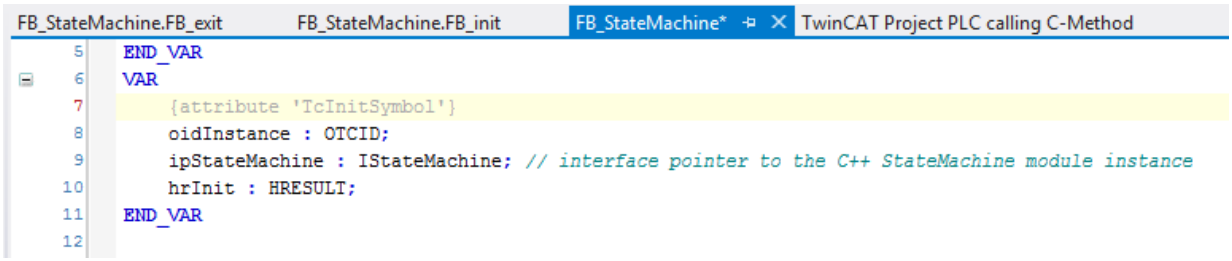
HINWEIS

Fehlende Attribute führen zu unerwartetem Verhalten

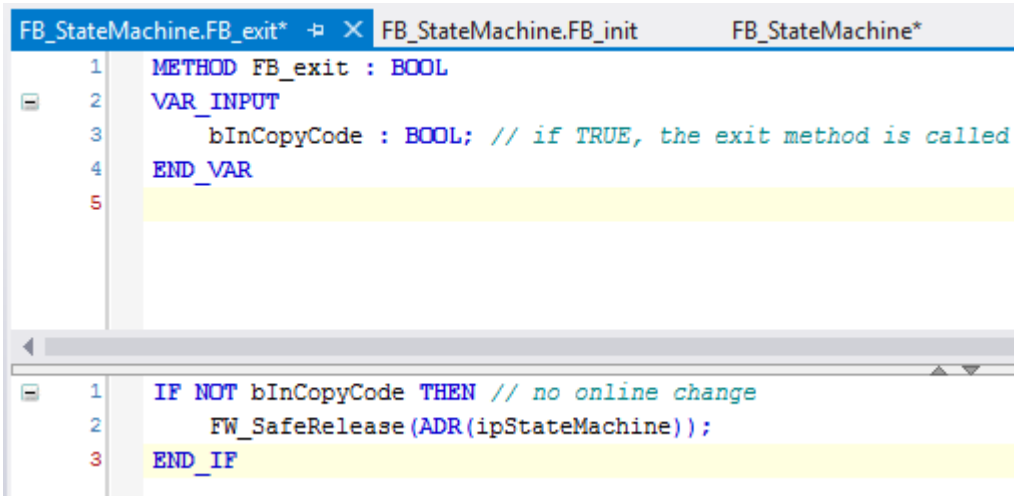
Attribut-Anweisungen mit geschwungenen Klammern stellen Code dar, der einzufügen ist.

Genaueres zu den Attributen ist in der SPS Dokumentation dokumentiert.

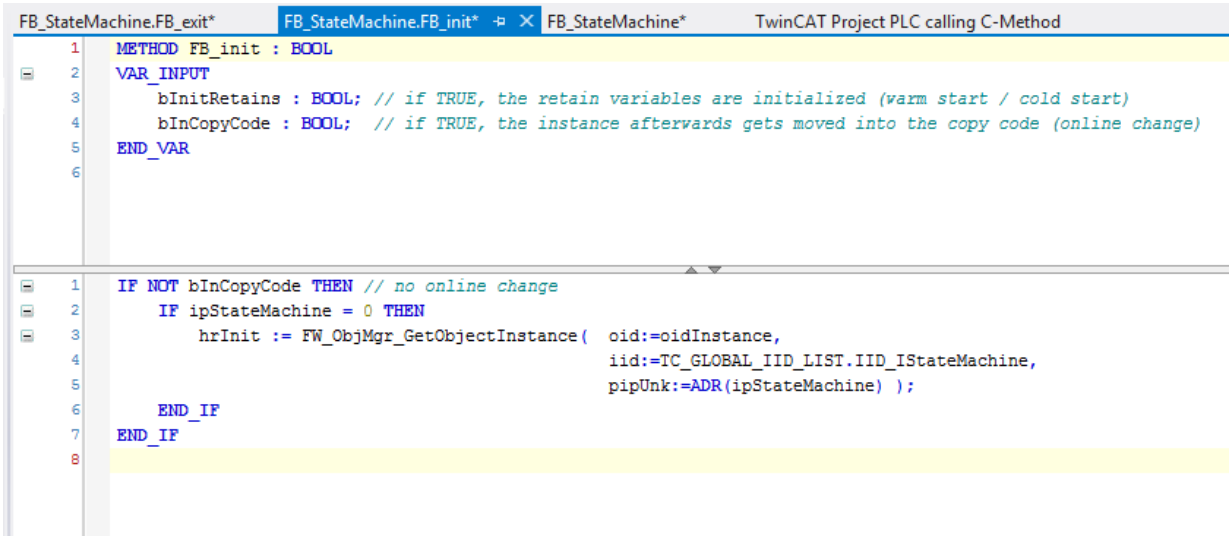
1. Implementieren Sie die Variablendeklaration der FB_Statemachine. Der FB selber braucht keinen zyklisch auszuführenden Code.



2. Implementieren Sie die Variablendeklaration und den Code-Bereich der Methode FB_exit.



3. Implementieren Sie die Variablendeklaration und den Code-Bereich der Methode FB_init.



4. Implementieren Sie die Variablendeklaration und den Code-Bereich der Methode GetState (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden).

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. Implementieren Sie die Variablendeklaration und den Code-Bereich der Methode SetState (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden).

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. Implementieren Sie die Variablendeklaration und den Code-Bereich der Methode Start (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden).

FB_StateMachine.Start | FB_StateMachine.SetState* | FB_StateMachine.GetState*

```
METHOD Start : HRESULT
```

```
IF (ipStateMachine <> 0) THEN
    Start:= ipStateMachine.Start();
END_IF
```

7. Implementieren Sie die Variablendeklaration und Code-Bereich der Methode Stop (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden).

FB_StateMachine.Stop | FB_StateMachine.Start | FB_StateMachine.SetState*

```
METHOD Stop : HRESULT
```

```
IF (ipStateMachine <> 0) THEN
    Stop:= ipStateMachine.Stop();
END_IF
```

⇨ Die Implementierung des FB_StateMachine der als Proxy zum Aufruf der C++ Modulinstanz agiert, ist abgeschlossen.

Schritt 7: FB in SPS aufrufen

Jetzt wird der FB_StateMachine in der POU MAIN aufgerufen.

Dieses einfache Beispiel agiert folgendermaßen:

- Zyklische Inkrementierung eines SPS-Zählers nCounter
- Wenn nCounter = 500, dann wird die C++-StateMachine mit Zustand „1“ gestartet, um seinen internen C++ Zähler zu inkrementieren. Anschließend per GetState() den State von C++ auslesen.
- Wenn nCounter = 1000, dann wird die C++-StateMachine auf Zustand „2“ gesetzt, um seinen internen C++ Zähler zu dekrementieren. Anschließend per GetState() den State von C++ auslesen.
- Wenn nCounter = 1500 dann wird die C++-StateMachine gestoppt. Der SPS nCounter wird ebenfalls auf „0“ gesetzt, sodass alles von vorne beginnt.

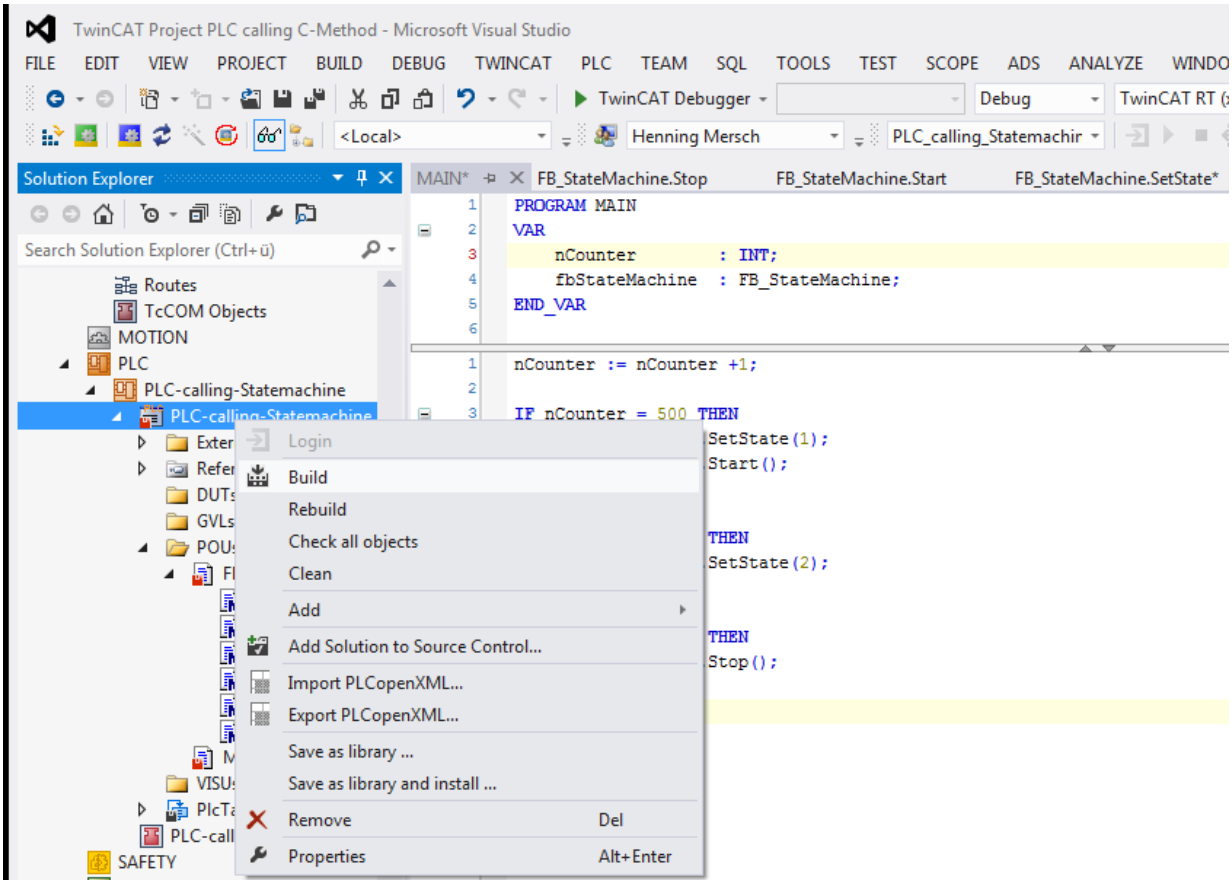
MAIN | TwinCAT Project PLC calling C-Method | FB_StateMachine.GetState

```
1  PROGRAM MAIN
2  VAR
3      nCounter        : INT;
4      nCurrentState   : INT;
5      fbStateMachine  : FB_StateMachine;
6  END_VAR
7
```

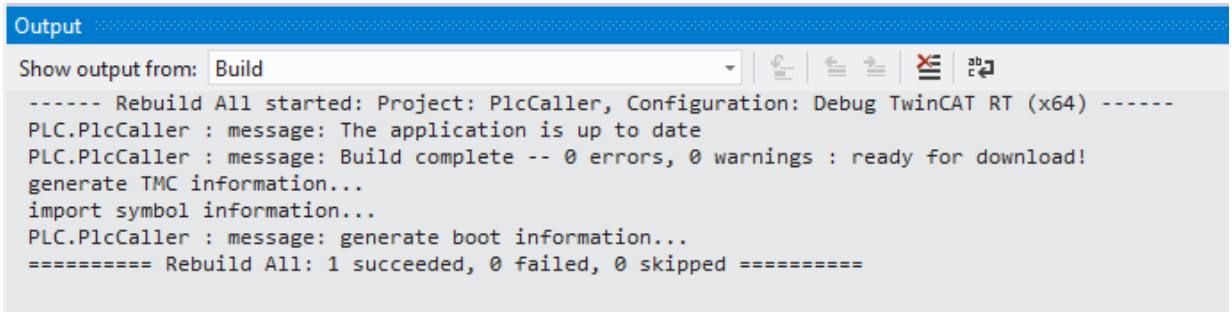
```
1  nCounter := nCounter +1;
2
3  IF nCounter = 500 THEN
4      fbStateMachine.SetState(1);
5      fbStateMachine.GetState(ADR(nCurrentState));
6      fbStateMachine.Start();
7  END_IF
8
9  IF nCounter = 1000 THEN
10     fbStateMachine.SetState(2);
11     fbStateMachine.GetState(ADR(nCurrentState));
12 END_IF
13
14 IF nCounter = 1500 THEN
15     fbStateMachine.Stop();
16     nCounter := 0;
17 END_IF
```

Schritt 8: SPS-Code kompilieren

1. Machen Sie einen Rechtsklick auf das SPS Projekt und klicken Sie auf **Build**.



⇨ Das Ergebnis der Kompilation zeigt „1 succeeded - 0 failed“.



Schritt 9: SPS FB mit C++ Instanz verknüpfen

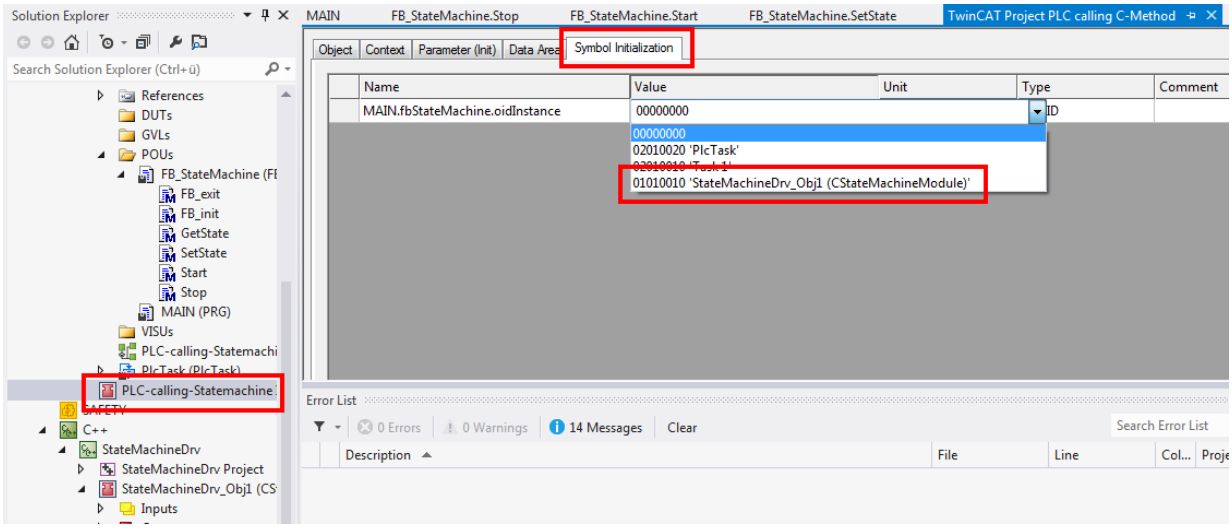
Jetzt werden die Vorteile aller vorherigen Schritte offensichtlich:

Der SPS-FB FB_StateMachine kann im Hinblick auf die Verknüpfung mit jeder Instanz des C++ Moduls StateMachine konfiguriert werden. Das ist eine sehr flexible und leistungsstarke Methode, um SPS und C++ Module auf der Maschine miteinander zu verbinden.

1. Navigieren Sie zur Instanz des SPS Moduls im linken Baum und wählen die Registerkarte **Symbol initialization** auf der rechten Seite.

   ⇨ Es werden alle Instanzen von FB_StateMachine aufgelistet, in diesem Beispiel haben wir lediglich eine FB Instanz in POU MAIN definiert.

2. Wählen Sie das Dropdownfeld **Value** und und dann die C++ Modulinstanz, die mit der FB Instanz verknüpft werden soll.

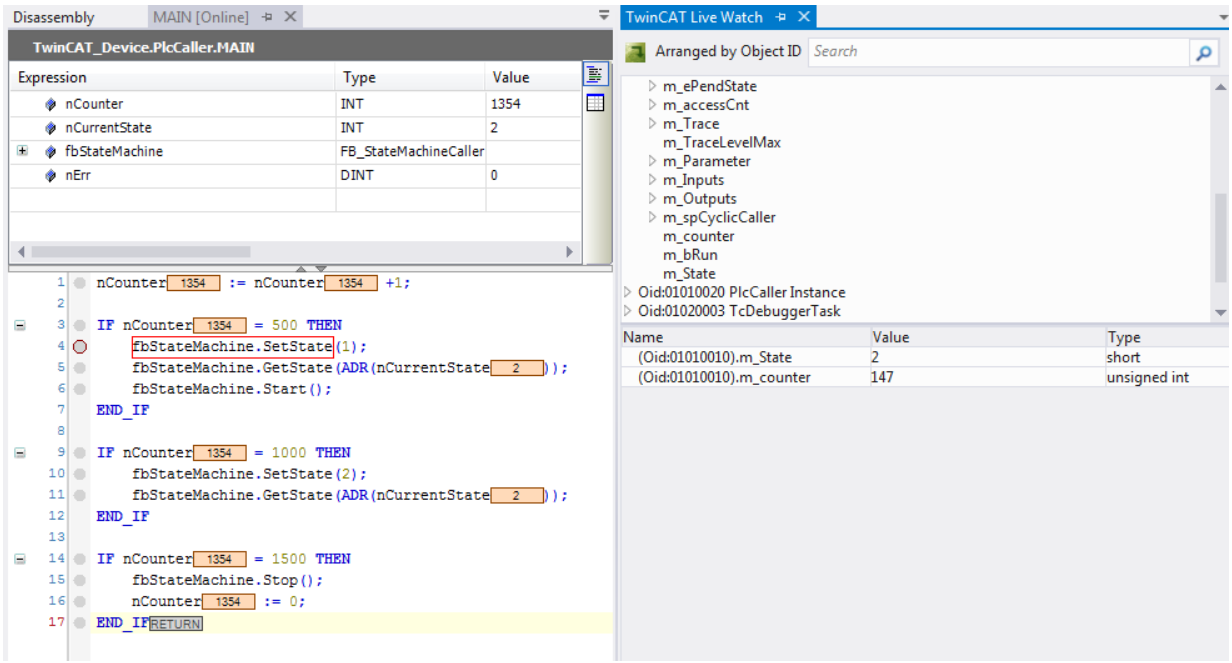


⇨ SPS und C++ Modul sind miteinander verbunden.

Schritt 10: Die Ausführung von beiden Modulen, SPS und C++, beobachten

Nach der Aktivierung der TwinCAT Konfiguration, dem Herunterladen und Starten des SPS-Codes, kann die Ausführung der beiden Codes, SPS und C++, einfach beobachtet werden:

1. Nach dem Login und Start des SPS-Projekts, befindet sich der Editor bereits im online-Modus (linke Seite der folgenden Abbildung).
2. Um auf online-Variablen des C++ Moduls zugreifen zu können, aktivieren Sie das C++ Debugging [▶ 74] und führen die Schritte des Schnellstarts [▶ 62] aus, um das Debugging zu starten (rechte Seite der folgenden Abbildung).



15.10 Beispiel11a: Modulkommunikation: Methodenaufruf C++-Modul nach C++-Modul

Dieser Artikel beschreibt, wie TwinCAT 3 C++ Module über Methodenaufrufe kommunizieren können. Die Methode schützt die Daten durch eine Critical Section, folglich kann der Zugriff von verschiedenen Kontexten / Tasks initiiert werden.

Download

Hier erhalten Sie den Quellcode für dieses Sample**:**

https://github.com/Beckhoff/TC1300_Samples/tree/main/S11a-Mod2ModCS

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Projekt enthält drei Module:

- Die Instanz der CModuleDataProvider Klasse hostet die Daten und schützt vor dem Zugriff mit Hilfe der Methoden Retrieve und Store durch eine Critical Section.
- Die Instanz der Modulklasse CModuleDataRead liest die Daten aus dem DataProvider über den Aufruf der Methode Retrieve.
- Die Instanz der Modulklasse CModuleDataWrite schreibt die Daten aus dem DataProvider über den Aufruf der Store Methode.

Die Read/Write Instanzen werden für den Zugriff auf die DataProvider Instanz konfiguriert, die im Menü **Interface Pointer** auf der Instanzkonfiguration zu sehen ist.
Dort kann man auch den Kontext (Task) konfigurieren, in dem die Instanzen auszuführen sind. Bei diesem Beispiel werden zwei Tasks verwendet - TaskRead und TaskWrite.
Die DataWriteCounterModul- Parameter von CModuleDataWrite und auch DataReadCounterModulo (CModuleDataRead) ermöglichen die Bestimmung des Moments, an dem die Modulinstanzen den Zugriff initiieren.

CriticalSections sind im SDK in der TcRtInterfaces.h beschrieben und damit für den Echtzeit-Kontext vorgesehen.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340318859.zip

15.11 Beispiel12: Modulkommunikation: Verwendet IO Mapping

Dieser Artikel beschreibt, wie zwei TwinCAT 3 C++ Module über das gewöhnliche IO Mapping von TwinCAT 3 kommunizieren können: Zwei Instanzen sind über das IO Mapping verknüpft und greifen periodisch auf den Variablenwert zu.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S12-Mod2ModIOMapping

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**

2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Beide Instanzen werden anhand einer Modulklasse ModuleInToOut realisiert: Die Klasse kopiert zyklisch ihren Eingangsdatenbereich Value auf ihren Ausgangsdatenbereich Value.
Die Instanz Front agiert als Frontend für den Nutzer. Ein Eingangs-Value wird - durch die Methode cycleupdate() - an den Ausgangs-Value übergeben. Dieser Ausgangs-Value von Front wird dem Eingangs-„Value“ der Instanz Back zugeordnet (verknüpft).
Die Back-Instanz wiederum kopiert den Eingangs-Value auf ihren Ausgangs-Value, was vom Nutzer beobachtet werden kann (siehe folgende Schritte des Schnellstarts um das Debugging zu starten: TwinCAT 3 C++ Projekt debuggen)

Letztendlich kann der Nutzer den Eingangs-Value der Front-Instanz festlegen und den Ausgangs-Value von Back beobachten.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340322187.zip

15.12 Beispiel13: Modulkommunikation: Methodenaufruf C++-Modul nach SPS-Modul

Dieser Artikel beschreibt, wie ein TwinCAT 3 C++ Modul per TcCOM Interface Methoden eines Funktionsbausteins der SPS aufruft.

Download

Hier erhalten Sie den Quellcode für dieses Sample: https://github.com/Beckhoff/TC1300_Samples/tree/main/S13-CppToPLC

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

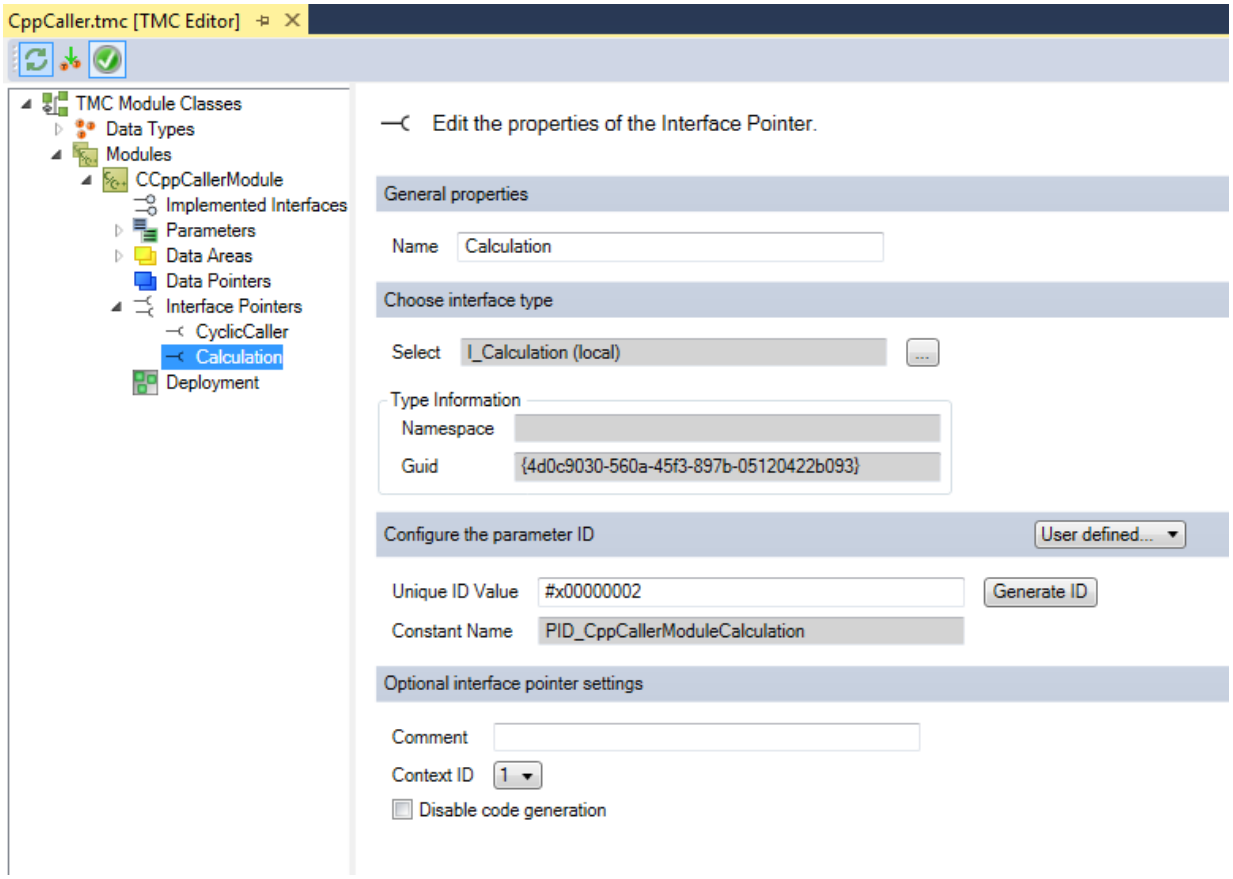
Dieses Beispiel stellt die Kommunikation von einem C++ Modul zu einem Funktionsbaustein einer SPS mittels Methodenaufruf dar. Hierfür wird ein TcCOM-Interface definiert, welches von der SPS angeboten wird und von dem C++ Modul genutzt wird.

Die SPS-Seite als Provider entspricht dabei dem entsprechenden Projekt des Beispiels TcCOM Sample 01 [▶ 323], wo eine SPS nach SPS Kommunikation betrachtet wird. Hier wird nun ein Caller in C++ bereitgestellt, der das gleiche Interface nutzt.

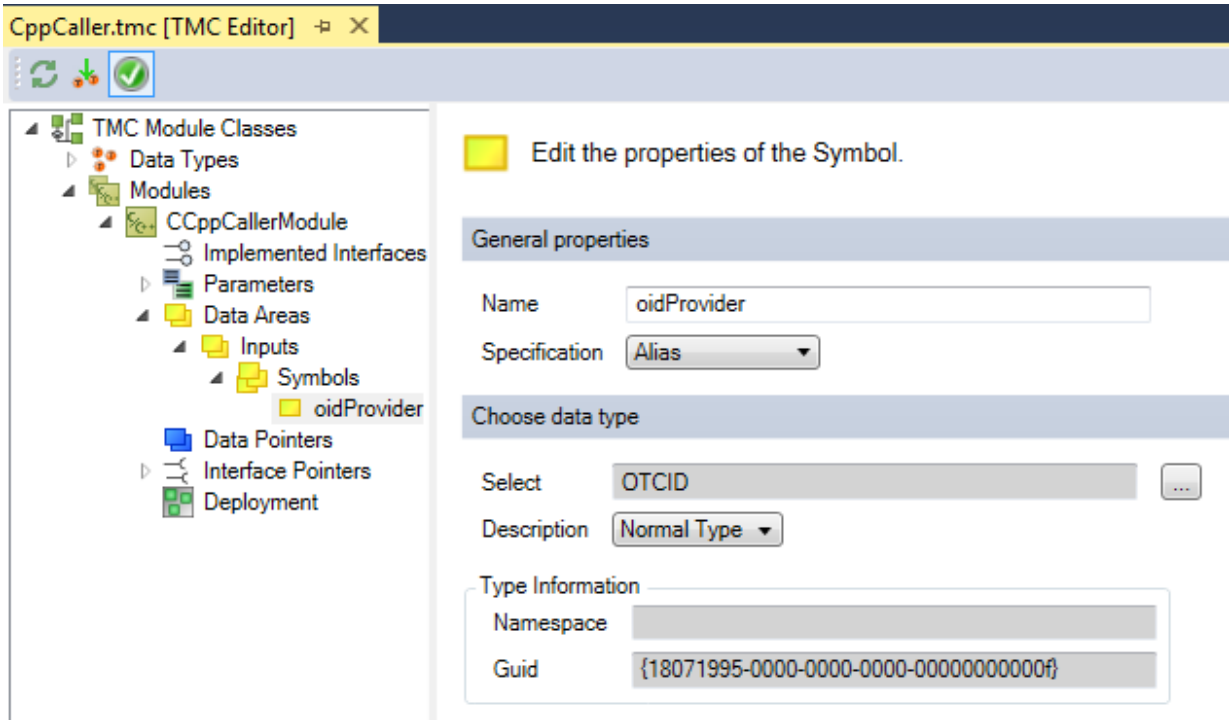
Die SPS-Seite wird von TcCOM Sample 01 [▶ 323] übernommen. Der dort als TcCOM-Modul registrierte Funktionsbaustein bietet die ihm zugeteilte Objekt-ID als Ausgangsvariable an.
Aufgabe des C++ Moduls ist es, das angebotene Interface dieses Funktionsbausteins zugreifbar zu machen.

✓ Es wird von einem C++ Projekt mit einem Cycle IO-Modul ausgegangen.

1. Legen Sie im TMC Editor einen Interface-Pointer mit dem Namen Calculation vom Typ I_Calculation an. Über diesen erfolgt später der Zugriff.

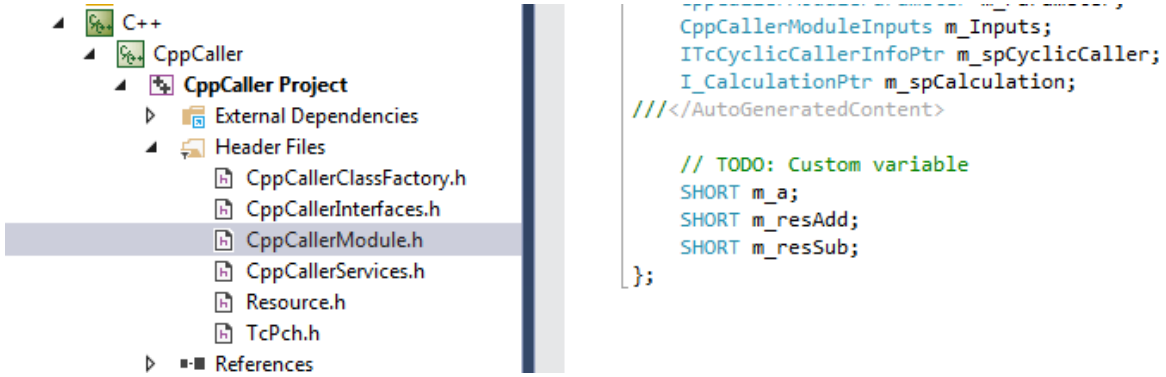


2. Die Data Area Inputs wurde mit dem Typ Input-Destination vom Modul-Wizard bereits erstellt. Hier legen Sie im TMC Editor einen Eingang mit dem Namen oidProvider vom Typ OTCID an. Über diesen wird später die Objekt-ID aus der SPS verknüpft.

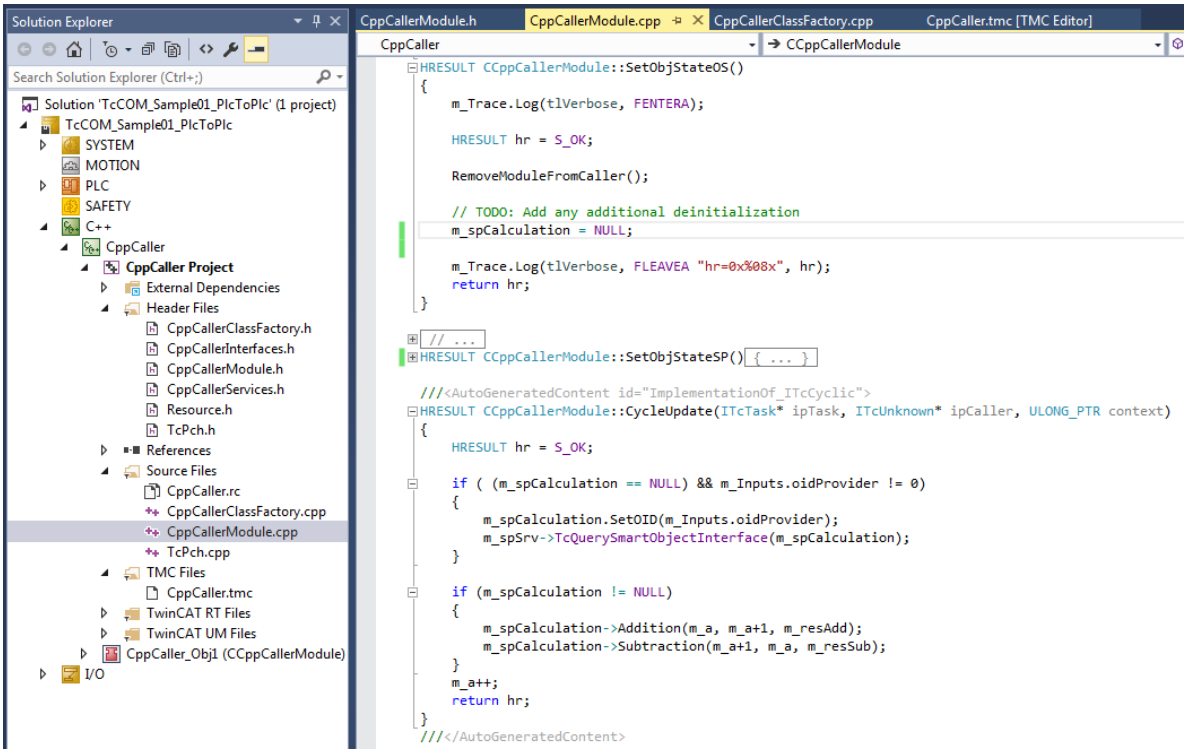


3. Alle weiteren Symbole sind für das Beispiel nicht relevant und können gelöscht werden.

   ⇨ Der TMC-Code-Generator bereitet den Code entsprechend vor:
   In dem Header des Moduls werden einige Variablen angelegt, um die Methoden-Aufrufe später durchzuführen.

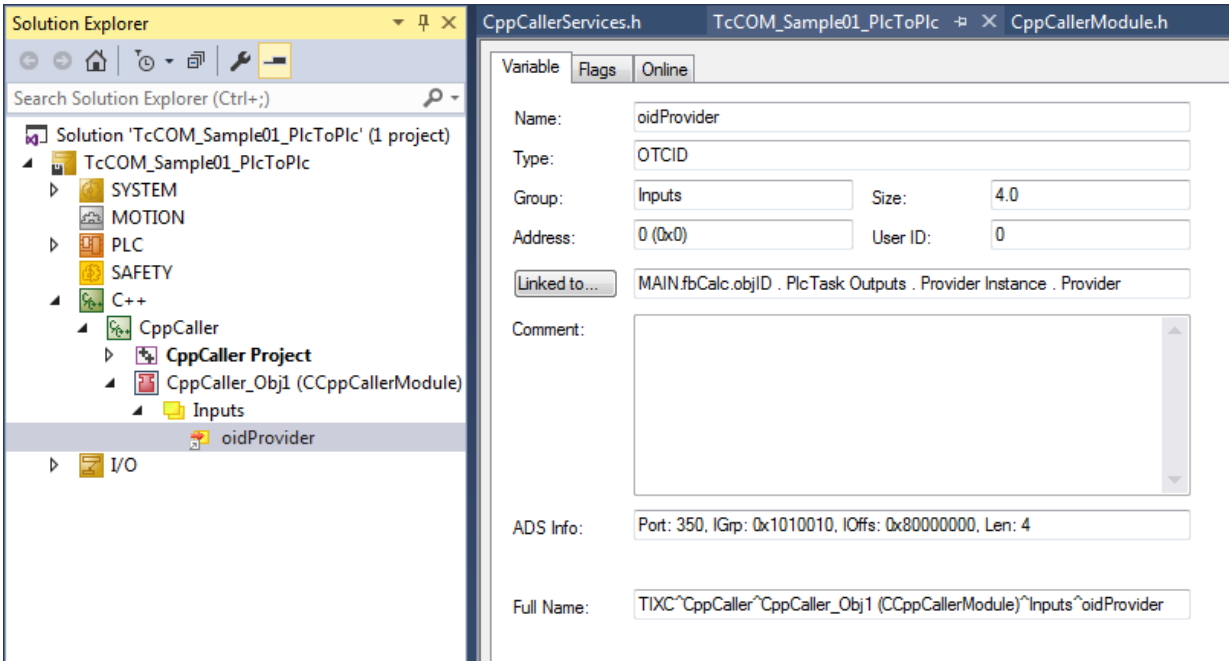


   Im eigentlichen Code des Moduls wird im CycleUpdate() der Interface-Pointer anhand der von der SPS übermittelten Objekt-ID gesetzt. Es ist wichtig, dass dieses im CycleUpdate() und damit im Echtzeitkontext geschieht, da die SPS erst den Baustein bereitstellen muss.
   Ist dies einmalig erfolgt, können die Methoden aufgerufen werden.



Zusätzlich muss, wie oben zu sehen ist, der Interface-Pointer beim Herunterfahren aufgeräumt werden. Dies geschieht in der SetObjStateOS Methode.

4. Bauen Sie das C++ Projekt nun einmal.
5. Legen Sie eine Instanz des Moduls an.
6. Verbinden Sie den Eingang des C++ Moduls mit dem Ausgang der SPS.



⇨ Das Projekt kann gestartet werden. Wenn die SPS läuft, wird die OID durch das Mapping an die C++ Instanz bekannt gemacht. Sobald dies erfolgt ist, können die Methoden aufgerufen werden.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340323851.zip

15.13 Beispiel19: Synchroner Dateizugriff

Dieser Artikel beschreibt die Implementierung eines TwinCAT 3 C++ Moduls, das auf Dateien auf der Festplatte während des Starts eines Moduls, also in Echtzeitumgebung, zugreifen kann.

Download

Hier erhalten Sie den Quellcode für dieses Sample: https://github.com/Beckhoff/TC1300_Samples/tree/main/S19-FileIOSync

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Der gesamte Quellcode, der vom Assistenten nicht automatisch generiert wird, wird mit dem Kommentarbeginn-Flag „//sample code“ und dem Kommentarende-Flag „//sample code end“ gekennzeichnet.
Somit können Sie nach diesen Zeichenketten in den Dateien suchen, um sich ein Bild von den Einzelheiten zu machen.

Beschreibung

Dieses Beispiel beschreibt den Zugriff auf Dateien über die TwinCAT Schnittstelle ITCFileAccess. Der Zugriff ist synchron und kann z. B. für das Lesen einer Konfiguration beim Starten eines Moduls verwendet werden.

Das Beispiel beinhaltet ein C++ Modul TcFileTestDrv mit einer Instanz dieses Moduls TcFileTestDrv_Obj1. Bei diesem Beispiel findet der Dateizugriff beim Übergang PREOP to SAFEOP statt, also in der Methode SetObjStatePS().

Hilfsmethoden kapseln den Umgang mit Dateien ein.
Zuerst wird als Beispiel eine allgemeine Dateiinformation und eine Verzeichnisliste in die Protokollvorrichtung von TwinCAT 3 gedruckt. Anschließend wird eine Datei *%TC_TARGETPATH%DefaultConfig.xml* (normalerweise *C:\TwinCAT\3.1\Target\DefaultConfig.xml*) auf „%TC_TARGETPATH%DefaultConfig.xml.bak“ kopiert.

Zum Zugang zu den Protokolleinträgen, siehe Registerkarte **Error List** vom TwinCAT 3 Ausgabefenster. Sie können die Informationsmenge mittels Änderung der Variablen TraceLevelMax in der Instanz TcFileTestDrv_obj1 in Registerkarte **Parameter (Init)** einstellen.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340325515.zip

15.14 Beispiel20: FileIO-Write

Dieser Artikel beschreibt die Implementierung von TwinCAT 3 C++ Modulen, die (Prozess-)Werte in eine Datei schreiben.

Das Beschreiben der Datei wird von einem deterministischen Zyklus veranlasst - die Ausführung von File IO ist entkoppelt (asynchron), d. h.: der deterministische Zyklus läuft weiter und wird nicht durch das Schreiben in der Datei behindert.

Download

Hier erhalten Sie den Quellcode für dieses Sample: https://github.com/Beckhoff/TC1300_Samples/tree/main/S20-FileIOWrite

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**.
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel beinhaltet eine Instanz von TcAsyncWritingModule, die Daten in die Datei AsyncTest.txt im Verzeichnis BOOTPRJPATH (Windows: *C:\TwinCAT\3.1\Boot; TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot*) schreibt.

TcAsyncBufferWritingModule hat zwei Puffer (m_Buffer1, m_Buffer2), die abwechselnd mit aktuellen Daten gefüllt werden. Die Membervariable m_pBufferFill zeigt auf den derzeit zu füllenden Puffer. Wenn ein Puffer vollständig gefüllt ist, dann wird die Membervariable m_pBufferWrite so gesetzt, dass sie auf den vollen Puffer zeigt.

Diese Daten werden mit Hilfe von TcFsmFileWriter in eine Datei geschrieben.

Beachten Sie, dass die Datei keinen von Menschen lesbaren Inhalt, wie ASCII Zeichen enthält, stattdessen werden in diesem Beispiel binäre Daten in die Datei geschrieben.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340328843.zip

15.15 Beispiel20a: FileIO-Cyclic Read / Write

Dieser Artikel stellt ein umfassenderes Beispiel als S20 und S19 dar. Es beschreibt zyklischen Lese- und/oder Schreibzugriff auf Dateien von einem TwinCAT 3 C++ Modul aus.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S20a-FileIO-CyclicReadWrite

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**.
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel beschreibt den Zugriff auf Dateien für Lesen und/oder Schreiben über die Methode CycleUpdate, also auf zyklische Weise.

Dieses Beispiel beinhaltet die folgenden Projekte und Modulinstanzen.

- Eine statische Bibliothek (TcAsyncFileIo) bietet den Dateizugriff.
  Der Code für den Dateizugriff kann geteilt werden, also befindet sich dieser Code in einer statischen Bibliothek, die von den Treiberprojekten genutzt wird.
- Ein Treiber (TcAsyncBufferReadingDrv) stellt zwei Instanzen zur Verfügung:
  - ReadingModule: verwendet die statische Bibliothek, um die Datei AsyncTest.txt zu lesen.
  - WriteDetectModule: Schreiboperationen erkennen und Leseoperationen veranlassen.
- Ein Treiber (TcAsyncBufferWritingDrv) stellt eine Instanz zur Verfügung:
  - WriteModule: Verwendet die statische Bibliothek, um die Datei AsyncTest.txt zu schreiben.

Beim Starten des Beispiels beginnt das Schreibmodul mit dem Schreiben von Daten in die Datei, die sich im Bootprojekt-Pfad (Windows: *C:\TwinCAT\3.1\Boot\AsyncTest.txt; TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot/AsyncTest.txt*) befindet. Die Eingangsvariable bDisableWriting kann für die Verhinderung des Schreibens verwendet werden.

Die Objekte sind miteinander verbunden: wenn das Schreiben erledigt ist, triggert das WritingModule das DetectModule von TcAsyncBufferReadingDrv. Dadurch wird ein Lesevorgang durch das ReadingModule veranlasst.

Beobachten Sie die nBytesWritten- / nBytesRead-Ausgangsvariablen des WritingModules / ReadingModules. Darüber hinaus werden Protokollnachrichten auf Ebene verbose generiert. Diese können - wie gehabt - mit Hilfe des TraceLevelMax Parameters der Module konfiguriert werden.

- Ein Treiber (TcAsyncFileFindDrv) stellt eine Instanz zur Verfügung
  - FileFindModule: listen Sie die Dateien eines Verzeichnisses mit Hilfe der statischen Bibliothek auf.

Lösen Sie die Aktion mit Hilfe der Eingangsvariablen bExecute aus. Der Parameter FilePath beinhaltet das Verzeichnis, dessen Dateien aufzulisten sind (Windows: *c:\TwinCAT\3.1\Boot*; TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot/**).

Beobachten Sie die Ablaufverfolgung (Protokollebene Verbose) bezüglich der Liste der gefundenen Dateien.

Das Beispiel verstehen

Das Projekt TcAsyncFileIO beinhaltet verschiedene, in einer statischen Bibliothek befindliche Klassen. Diese Bibliothek wird von Treiberprojekten für das Lesen und Schreiben verwendet.

Jede Klasse ist für eine Dateizugriffsoperation wie Öffnen / Lesen / Schreiben / Auflisten / Schließen / ... bestimmt. Da die Ausführung innerhalb eines zyklischen Echtzeitkontextes stattfindet, hat jede Operation einen Status und die Klasse kapselt diese Zustandsmaschine ein.

Als Einstiegspunkt zum Verständnis des Dateizugriffs, beginnen Sie mit den Klassen TcFsmFileReader und TcFsmFileWriter.

Sollten zu viele Verlaufsverfolgungsmeldungen auftreten, die das Verständnis des Beispiels erschweren, deaktivieren Sie Module!

Siehe auch

Beispiel S19 [▶ 308]

Beispiel S20 [▶ 308]

Beispiel S25 [▶ 315]

Schnittstelle ITcFileAccess [▶ 172] / Schnittstelle ITcFileAccessAsync [▶ 180]

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340327179.zip

15.16 Beispiel22: Automation Device Driver (ADD): Zugang DPRAM

Dieser Artikel beschreibt die Implementierung eines TwinCAT 3 C++ Treibers, der als TwinCAT Automation Device Driver (ADD) mit Zugriff auf DPRAM fungiert.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S22-ADD

HINWEIS

Konfigurationsdetails

Lesen Sie vor der Aktivierung die Konfigurationsdetails unten.

1. Öffnen Sie die enthaltene zip-Datei in TwinCAT 3 mit einem Klick auf **Open Project …**.
2. Wählen Sie Ihr Zielsystem aus.
3. Bauen Sie das Beispiel auf Ihrer lokalen Maschine (z.B. **Build->Build Solution**).
4. Beachten Sie die unter **Konfiguration** auf dieser Seite aufgeführten Handlungsschritte.
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

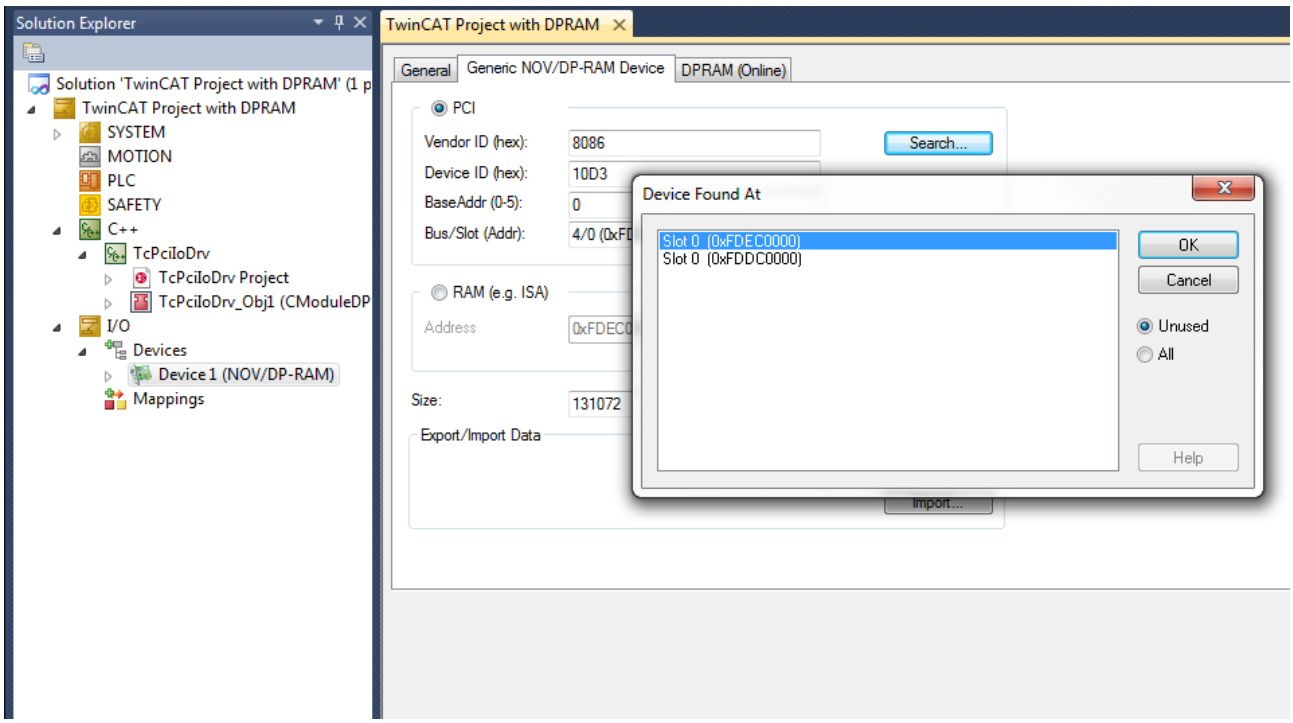
Beschreibung

Dieses Beispiel soll den Link Detect Bit des Netzwerkadapters (d. h. von einem CX5010) zyklisch ein- und ausschalten.

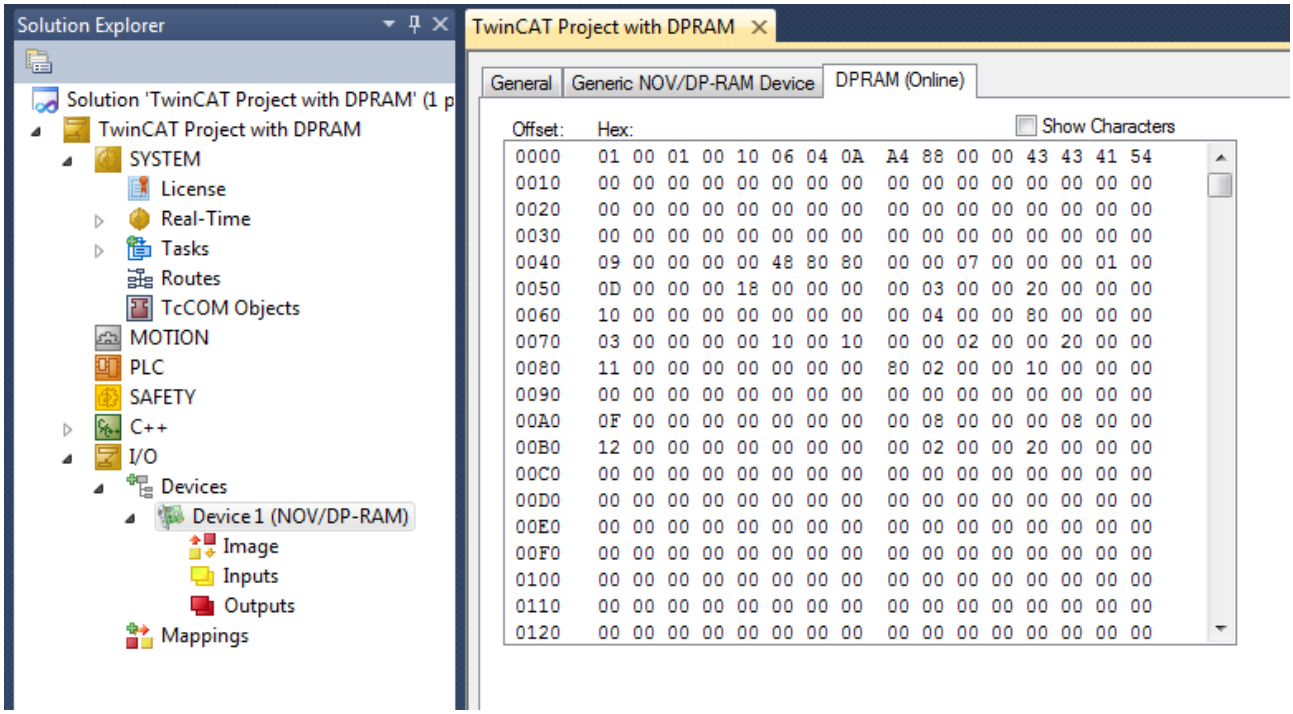
Das C++ Modul ist über den Schnittstellenzeiger PciDeviceAdi des C++ Moduls mit dem NOV/DP-RAM Gerät verbunden.

Konfiguration

Damit das Beispiel funktioniert, müssen Sie die Hardware-Adressen passend zu Ihrer persönlichen Hardware konfigurieren.
Überprüfen Sie die PCI Konfiguration:



Um zu überprüfen, ob die Kommunikation mit NOV/DPRAM korrekt eingerichtet ist, verwenden Sie die DPRAM (Online) Ansicht:



Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340343307.zip

15.17 Beispiel23: Strukturierte Ausnahmebehandlung (SEH)

Dieser Artikel beschreibt die Verwendung von Structed Exception Handling (SEH) anhand von fünf Varianten.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S23-SEH

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**.
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

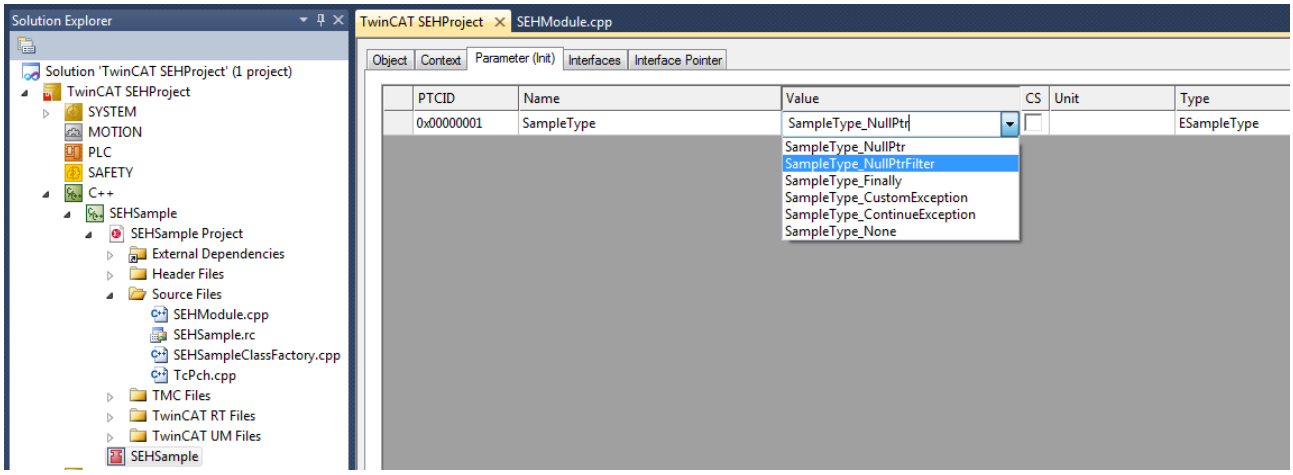
⇨ Das Beispiel ist einsatzbereit.

Beschreibung

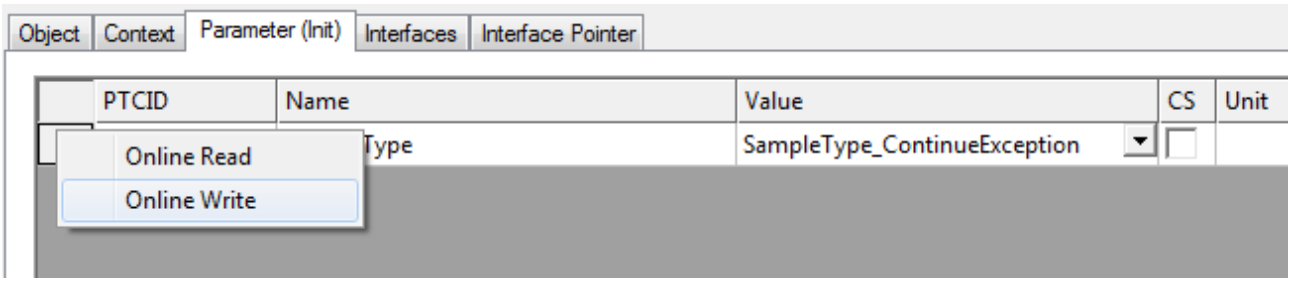
Das Beispiel beinhaltet fünf Varianten, die die Verwendung von SEH im TwinCAT C++ demonstrieren:

1. Exception bei einem NULL-Pointer Zugriff
2. Exception bei einem NULL-Pointer Zugriff mit einem Filter
3. Exception mit Finally
4. Eine kundenspezifische strukturierte Ausnahme (Structured Exception)
5. Exception mit Continue Block

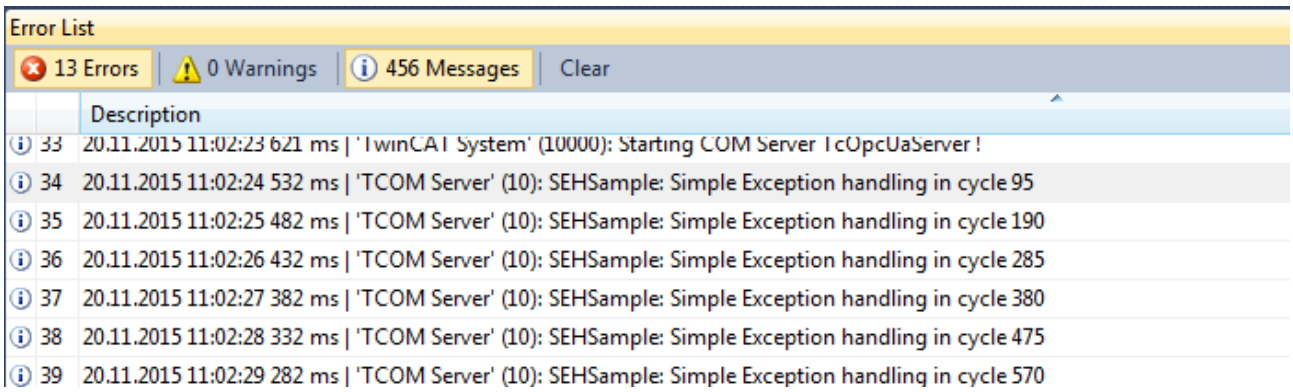
Alle diese Varianten sind durch eine Dropdown Box an der Instanz des C++ auswählbar:



Nach Auswahl einer Variante können Sie mit einem Rechts-Klick auf die vorderste Spalte auch zur Laufzeit den Wert schreiben:



Alle Varianten schreiben Trace-Nachrichten zur Verdeutlichung ihres Verhaltens, so dass Meldungen im TwinCAT Engineering erscheinen:



Das Beispiel verstehen

Die Anwahl in der Dropdown Box ist eine Enumeration, die im CycleUpdate() des Moduls zur Auswahl eines Falles (Switch-Case) verwendet wird. Dadurch können hier die Varianten unabhängig voneinander betrachtet werden:

1. Exception bei einem NULL-Pointer Zugriff
   Hier wird ein PBYTE als NULL angelegt und nachher verwendet, was zu einer Exception führt. Durch den TcTry{} Block wird diese abgefangen und eine Ausgabe erzeugt.
2. Exception bei einem NULL-Pointer Zugriff mit einem Filter
   Diese Variante greift auf einen NULL-Pointer zu, verwendet aber im TcExcept{} eine Methode FilterException(), die im Modul ebenfalls definiert ist. Innerhalb der Methode wird auf unterschiedliche Exceptions reagiert; in diesem Fall wird lediglich eine Meldung ausgegeben.
3. Exception mit Finally
   Hier erfolgt wieder ein NULL-Pointer Zugriff, wobei auf dieser Stelle jedoch in jedem Fall ein TcFinally{}-Block ausgeführt wird.
4. Eine kundenspezifische strukturierte Ausnahme (Structured Exception)
   Mittels TcRaiseException() wird eine Exception erzeugt, die durch die FilterException() Methode abgefangen und bearbeitet wird. Da es sich hier um eine im Modul definierte Exception handelt, gibt die FilterException() Methode zusätzlich eine weitere (spezifische) Meldung aus.
5. Exception mit Continue Block
   Auch hier erfolgt wieder ein NULL-Pointer Zugriff mit TcExcept{}, diesmal wird die Exception jedoch nach der Behandlung in der FilterException() Methode weiter gereicht, sodass auch der weitere TcExcept{} die Exception behandelt.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340344971.zip

15.18 Beispiel24: Semaphoren

Dieser Artikel beschreibt die Verwendung von Semaphoren.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

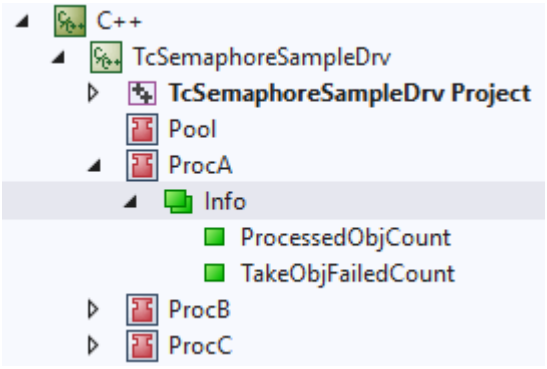
https://github.com/Beckhoff/TC1300_Samples/tree/main/S24-Semaphores

1. Öffnen Sie die enthaltene zip-Datei in TwinCAT 3 mit einem Klick auf **Open Project ...**.
2. Wählen Sie Ihr Zielsystem aus.
3. Bauen Sie das Beispiel auf Ihrer lokalen Maschine (z.B. **Build->Build Solution**).
4. Beachten Sie die unter **Konfiguration** auf dieser Seite aufgeführten Handlungsschritte.
5. Aktivieren Sie die Konfiguration mit einem Klick auf .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel enthält einen „LargeObjPool“, der zwei „LargeObj“ zur Bearbeitung durch die „LargeObjProcessors“ enthält.



Die „LargeObjProcessors“ werden dabei durch unterschiedliche Tasks mit unterschiedlichen Prioritäten und auf unterschiedlichen Cores der CPU angesteuert. Sie zählen in der Info-Dataarea mit, wie häufig eine Bearbeitung durchgeführt werden konnte („Info->ProcessedObjCount“) und wie häufig kein Objekt zur Bearbeitung innerhalb der Wartezeit bereitstand („Info->TakeObjFailedCount“).

Das Beispiel verstehen

Der LargeObjPool wird in der PS-Transition durch die `InitPool()`-Methode mit zwei LargeObj (definiert über den Parameter „ObjCount“) initialisiert. Der Pool bietet im Wesentlichen zwei Methoden an, die durch die „LargeObjProcessors“ genutzt werden:

- `TakeObj()` liefert ein LargeObj zurück, falls eins bereit steht. Dabei wird durch eine Semaphore bis zu einem Timeout gewartet, ob ein Objekt bereitgestellt wird. Die Objekte selbst sind dabei in einer Map gehalten, wobei die Zugriffe auf die Map durch eine CriticalSection geschützt sind.
- `ReturnObj()` stellt ein Objekt nach der Bearbeitung wieder in dem Pool zur erneuten Bearbeitung bereit.

LargeObjProcessor verwendet in seiner `CycleUpdate()` `TakeObj()` des Pools um ein LargeObj zur Bearbeitung zu bekommen. Falls dieses erfolgreich ist, wird eine Verarbeitung durch die Hilfsfunktion `ConsumeTime()` simuliert bevor das LargeObj mittels `ReturnObj()` an den Pool zurückgegeben wird.

Damit das Runterfahren des Systems sichergestellt werden kann, dürfen keine LargeObj mehr in Bearbeitung sein. Dieses wird erreicht, indem der LargeObjPool mittels `TcTryLockOpState()` und `TcReleaseOpState()` dem TwinCAT System mitteilt, dass ein Objekt in Bearbeitung ist. Solange der hinterlegte Zähler nicht 0 ist, wird das TcCOM-Objekt „Pool“ zurückgestellt und erst später durch die Op->SafeOp Transition runtergefahren.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10343761419.zip

15.19 Beispiel25: Statische Bibliothek

Dieser Artikel beschreibt die Implementierung und die Verwendung eines Moduls einer statischen TwinCAT 3 C++ Bibliothek.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S25-StaticLibrary

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**

2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel beinhaltet zwei Projekte - das Projekt DriverUsingStaticLib verwendet den statischen Inhalt des Projekts StaticLib.

StaticLib:
Auf der einen Seite bietet StaticLib die Funktion ComputeSomething in der StaticFunction.h/.cpp.
Auf der anderen Seite wird die Schnittstelle ISampleInterface definiert (siehe TMCEditor) und in die MultiplicationClass implementiert.

DriverUsingStaticLib:
In der CycleUpdate Methode der ModuleUsingStaticLib wird sowohl die Klasse als auch die Funktion von StaticLib verwendet.

Das Beispiel verstehen

Durchlaufen Sie folgende Schritte, um eine statische Bibliothek zu erstellen und zu verwenden.

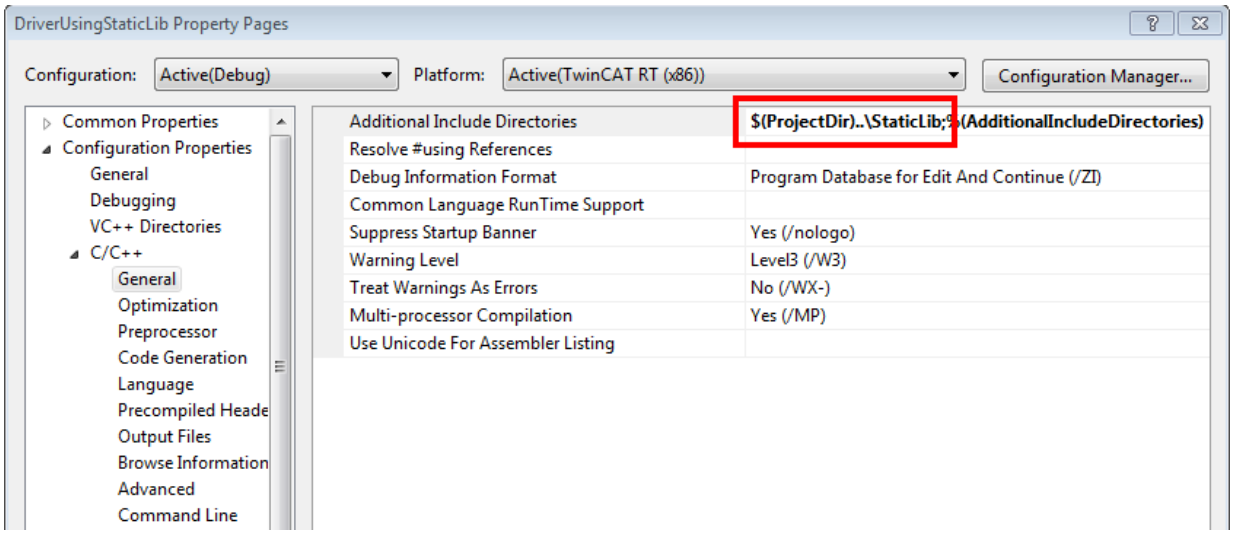
Manuelle Neukompilierung

Beachten Sie, dass Visual Studio die statische Bibliothek bei der Erstellung des Treibers nicht automatisch erneut kompiliert. Führen Sie das manuell aus.

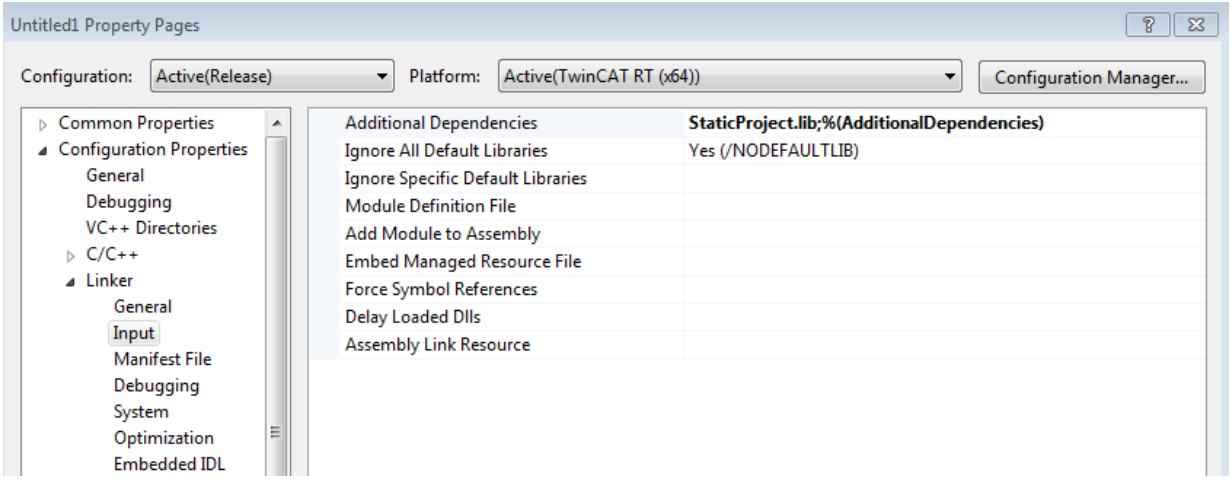
✓ Nutzen Sie bei der Erstellung eines C++ Projekts die Vorlage TwinCAT Static Library Project für die Erstellung einer statischen Bibliothek.

✓ Nutzen Sie für die folgenden Schritte den **Edit**-Dialog von VisualStudio, so dass nachher %(AddtitionalIncludeDirectories) bzw. %(AdditionalDependencies) verwendet wird.

1. Fügen Sie im Treiber dem Compiler das Verzeichnis der statischen Bibliothek unter **Additional Include Directories** hinzu.



2. Fügen Sie im Treiber, der die statische Bibliothek nutzt, diese als eine zusätzliche Abhängigkeit für den Linker hinzu. Öffnen Sie die Projekteigenschaften des Treibers und fügen die statische Bibliothek hinzu:



Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340346635.zip

15.20 Beispiel26: Ausführungsreihenfolge in einer Task

Dieser Artikel beschreibt die Bestimmung der Taskausführungsreihenfolge, wenn einer Task mehr als ein Modul zugeordnet ist.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S26-SortOrder

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel enthält das Modul SortOrderModule das zwei Mal instanziiert wird. Die Sortierreihenfolge bestimmt die Ausführungsreihenfolge, die über den TwinCAT Module Instance Configurator [▶ 135] konfiguriert werden kann.

Zum Beispiel verfolgt die Methode CycleUpdate den Objekt-Namen und die Objekt-ID zusammen mit der Sortierreihenfolge dieses Moduls. Auf dem Konsolenfenster kann man die Ausführungsreihenfolge sehen:

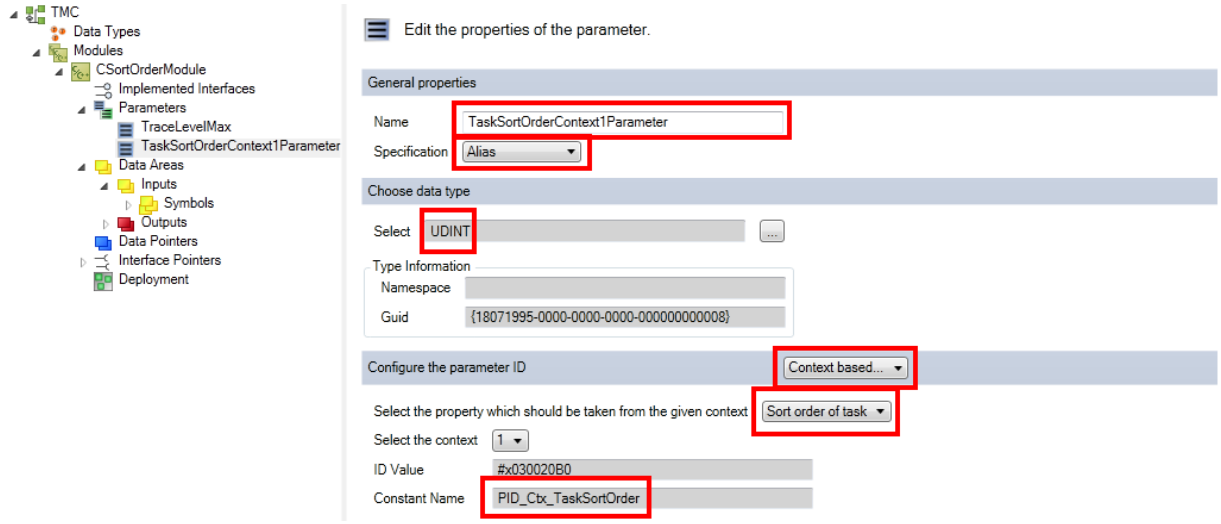
```
32 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
33 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
34 05.09.2014 13:01:15 343 ms | 'TCOM 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
```

Im Beispiel ist eine Instanz mit Sort Order 150 und eine mit 170 konfiguriert, während beide Instanzen einer Task zugeordnet sind.

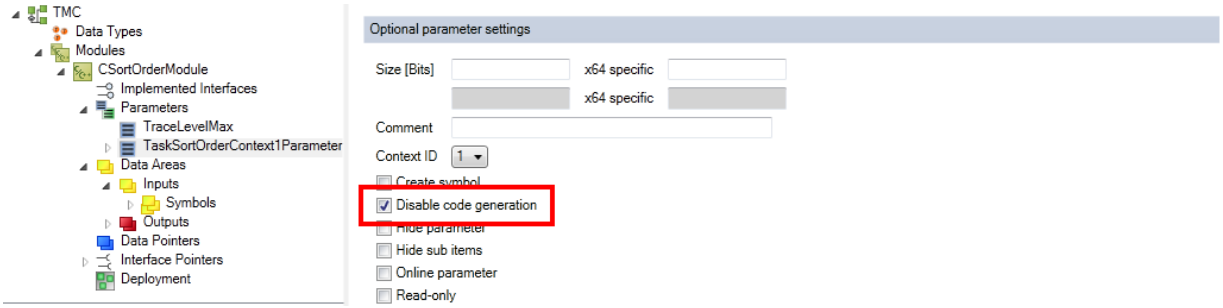
Das Beispiel verstehen

✓ Ein TcCOM C++ Modul mit zyklischem IO.

1. Das Modul benötigt einen kontextbasierten Parameter Sort order of task, der automatisch „PID_Ctx_TaskSortOrder“ als Namen auswählen wird.
   Beachten Sie, dass der Parameter ein Alias (Spezifikation) zum Datentyp UDINT sein muss:



2. Starten Sie den TMC Code Generator, um die Standardimplementierung zu erhalten.
3. Da der Code im nächsten Schritt geändert werden wird, deaktivieren Sie die Kodegenerierung für diesen Parameter jetzt.



4. Vergewissern Sie sich, dass Sie die Änderungen übernehmen, bevor Sie den TMC Code Generator erneut starten:
   Werfen Sie einen Blick auf das CPP Modul (SortOrderModule.cpp im Beispiel). Die Instanz des Smart Pointers des zyklischen Aufrufers beinhaltet Informationsdaten, zu denen ein Feld für die Sortierreihenfolge gehört. In diesem Feld wird der Parameterwert gespeichert.

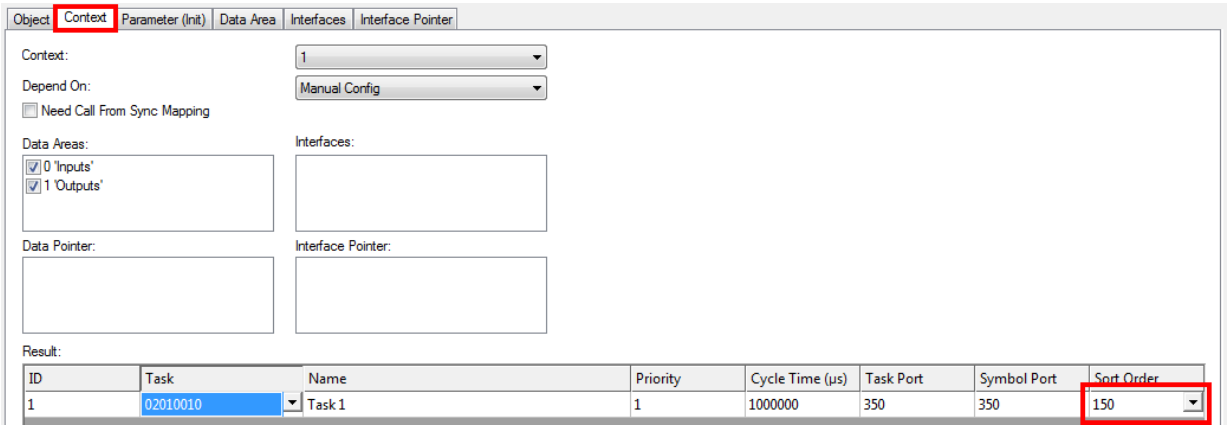
```
////////////////////////////////////////////////////////////////////////////////
// Set parameters of CSortOrderModule
BEGIN_SETOBJPARA_MAP(CSortOrderModule)
    SETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="SetObjectParameterMap">
    SETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    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. In diesem Beispiel werden Objekt-Name, -Id und SortOrder zyklisch verfolgt:

```
    // 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. Die Sortierreihenfolge kann auch als 4. Parameter der Methode ITcCyclicCaller::AddModule() übergeben werden, die in CModuleA::AddModuleToCaller() verwendet wird.
7. Weisen Sie den Instanzen dieses Moduls eine Task mit **langsamen Zyklusintervall** (z.B. 1000ms) zu, um die Verfolgungsmeldungen an das TwinCAT Engineering System zu begrenzen.
8. Weisen Sie jeder Instanz eine andere Sortierreihenfolge über den TwinCAT Module Instance Configurator [▶ 135] zu:



Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340348299.zip

15.21 Beispiel30: Zeitmessung

In diesem Artikel wird die Implementierung eines TwinCAT 3 C++ Moduls, das Zeitmessungsfunktionalitäten beinhaltet, beschrieben.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S30-Timing

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.

4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Dieses Beispiel befasst sich ausschließlich mit Zeitmessung wie

- Abfrage der Taskzyklenzeit in Nanosekunden
- Abfrage der Taskpriorität
- Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC)
- Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000
- Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC)

Siehe auch

ITcTask Schnittstelle [▶ 193]

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340349963.zip

15.22 Beispiel31: Funktionsbaustein TON in TwinCAT3 C++

Dieser Artikel beschreibt die Implementierung eines Verhaltens in C++, das vergleichbar mit einem TON Funktionsbaustein von SPS / IEC-61131-3 ist.

Source

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S31-CTON

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Verhalten dieses Moduls ist vergleichbar mit einem Modul, das mit dem Cyclic IO-Assistenten erstellt wurde. m_input.Value wird dem m_Output.Value hinzugefügt. Im Gegensatz zum Modul Cyclic IO, fügt dieses Modul lediglich m_input.Value zu m_Output.Value hinzu, wenn die definierte Zeitspanne (1000ms) abgelaufen ist.

Dies wird mit Hilfe einer CTON Klasse erzielt, die mit dem TON Funktionsbaustein von SPS / 61131 vergleichbar ist.

Das Beispiel verstehen

Die C++ Klasse CTON (TON.h/.cpp) stellt das Verhalten eines TON Funktionsbausteins von SPS / 61131 zur Verfügung. Die Methode Update() ist vergleichbar mit dem Rumpf des Funktionsbausteins, der regelmäßig aufgerufen werden muss.

Die Methode Update() erhält zwei „in“ Parameter:

- IN1: Startet die Zeitschaltuhr mit steigender Flanke, setzt die Zeitschaltuhr mit fallender Flanke zurück.
- PT: Beschreibt die abzuwartende Zeit, bevor Q gesetzt wird.

Und zwei „out“ Parameter:

- Q: Ist TRUE, wenn PT Sekunden nach IN eine steigende Flanke aufwies.
- ET: Bezeichnet die abgelaufene Zeit.

Darüber hinaus muss ITcTask zur Abfrage der Zeitbasis bereitgestellt werden.

Siehe auch

Beispiel30: Zeitmessung [▶ 319]

ITcTask Schnittstelle [▶ 193]

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340351627.zip

15.23 Beispiel35: Ethernet Zugriff

Dieser Artikel beschreibt die Implementierung von TwinCAT 3 C++ Modulen, die direkt über eine Ethernet-Karte kommunizieren. Der Beispielcode fragt eine Hardware-Adresse (MAC) von einem Kommunikationspartner mittels zyklischem Senden und Empfangen von ARP-Paketen ab.

Dieses Beispiel zeigt den direkten Zugriff auf die Ethernet-Karte. Die Function TF6311 TCP/UDP RT stellt einen Zugriff auf Ethernet-Karten auf Basis TCP und UDP bereit, sodass auf Basis dieses Beispiels eine Implementierung eines Netzwerkstacks nicht notwendig ist.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S35-AccessEthernet

1. Öffnen Sie die enthaltene zip-Datei in TwinCAT 3 mit einem Klick auf **Open Project …**.
2. Wählen Sie Ihr Zielsystem aus.
3. Bauen Sie das Beispiel auf Ihrer lokalen Maschine (z.B. **Build->Build Solution**).
4. Beachten Sie die unter **Konfiguration** auf dieser Seite aufgeführten Handlungsschritte.
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel beinhaltet eine Instanz des Moduls TcEthernetSample, das ARP-Pakete zwecks Bestimmung der fernen Hardware-Adresse (MAC) sendet und empfängt.

Die CycleUpdate Methode implementiert eine rudimentäre Zustandsmaschine für das Versenden von ARP-Paketen und das Warten auf eine Antwort mit einer Zeitüberschreitung.

Das Beispiel verwendet zwei Ethernet-Komponenten von TwinCAT:

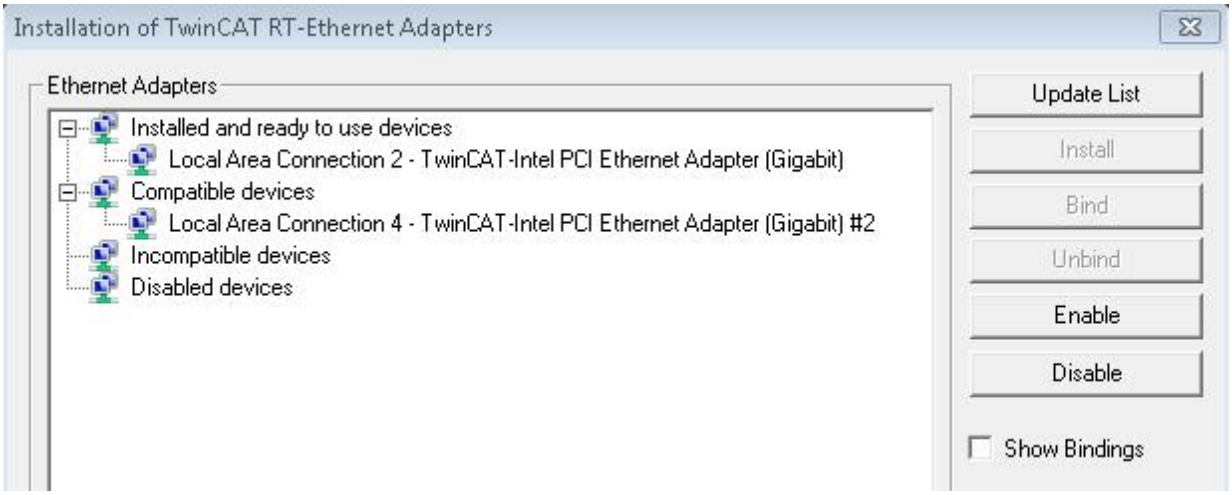
1. Ein **ITcEthernetAdapter** (Instanzenname im Beispiel ist m_spEthernetAdapter) stellt einen RT Ethernet Adapter dar. Er ermöglicht den Zugriff auf die Adapterparameter wie Hardware-MAC-Adresse, Verknüpfungsgeschwindigkeit, Verknüpfungsfehler. Er kann für das Senden von Ethernet-Frames verwendet werden und ermöglicht einer Modulinstanz, sich als ein ITcIoEthProtocol über die Methode registerProtocol anzumelden.
2. Das **ITcIOoEthProtocol** wird um das Abtastmodul erweitert, das dafür sorgt, dass eine Notifizierung bei Ethernet-Ereignissen über den **ITcEthernetAdapter** stattfindet.

Konfiguration

Das heruntergeladene TwinCAT Projekt muss für die Ausführung in einer Netzwerkumgebung konfiguriert sein. Bitte führen Sie die folgenden Schritte aus:

✓ Das Beispiel verlangt, dass die Ethernet-Karte den TwinCAT Treiber verwendet.

1. Starten Sie TcRteInstall.exe entweder vom XAE über das Menü **TwinCAT->Show Realtime Ethernet compatible devices…** oder von der Festplatte auf den XAR Systemen.



2. Möglicherweise müssen Sie den Treiber mit Hilfe der Schaltflächen installieren und aktivieren.
3. TwinCAT muss wissen, welche Ethernet-Karte verwendet werden soll. Öffnen Sie das Projekt in **XAE** und klicken Sie auf **I/O / Devices / Device 1 (RT-Ethernet Adapter) auswählen**.
4. Klicken Sie auf die Registerkarte **Adapter** und wählen Sie mit **Search** den Adapter aus.
5. TcEthernetSample_Obj1 muss konfiguriert werden. Öffnen Sie das Instanzenfenster und legen Sie die folgenden Werte fest:
   Parameter (Init): SenderIpAddress (IP von in Schritt 2 konfiguriertem Netzwerkadapter)
   Parameter (Init): TargetIpAddress (IP von Ziel-Host)
   Schnittstellenzeiger: EthernetAdapter muss auf I/O / Devices / Device 1 (RT-Ethernet Adapter) zeigen.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340353291.zip

15.24 Beispiel37: Daten archivieren

Das Beispiel TcCOM Object Archiv beschreibt das Wiederherstellen und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung.

TwinCAT unterstützt Retain Daten

TwinCAT unterstützt auch Retain Daten um das NOVRAM eines Gerätes zur Persistierung von Daten zu nutzen.

Download

Hier erhalten Sie den Quellcode für dieses Sample:

https://github.com/Beckhoff/TC1300_Samples/tree/main/S37-ArchiveData

1. Öffnen Sie mittels eines Visual Studios, in dem TwinCAT installiert ist, das Projekt über **Open Project ...**.
2. Konfigurieren Sie das Signieren für dieses Projekt, indem Sie auf der Seite unter Rechtsklick auf dem **Projekt->Properties->Tc Sign** die TwinCAT Signierung anschalten und Ihr Zertifikat und ggf. Passwort konfigurieren.
   Weitere Informationen zur Signierung der C++ Projekte im Kapitel TwinCAT [▶ 24].
3. Wählen Sie Ihr Zielsystem aus.
4. Bauen Sie das Beispiel (z. B. **Build->Build Solution**).
5. Aktivieren Sie die Konfiguration mit einem Klick auf  .

⇨ Das Beispiel ist einsatzbereit.

Beschreibung

Das Beispiel TcCOM Object Archiv beschreibt das Wiederherstellen und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung. Der Zustand der Beispielklasse CModuleArchive entspricht dem Wert des Zählers CModuleArchive::m_counter.

Beim Übergang von PREOP zu SAFEOP, d. h. beim Aufruf der Methode CModuleArchive::SetObjStatePS(), wird der Objektarchivserver (ITComObjArchiveServer) für die Erstellung eines Objektarchivs zum Lesen benutzt, auf das über die Schnittstelle ITComArchiveOp zugegriffen wird. Diese Schnittstelle stellt Überladungen von operator>>() zur Verfügung um im Archiv zu lesen.

Beim Übergang von SAFEOP zu PREOP, d. h. beim Aufruf der Methode CModuleArchive::SetObjStateSP(), wird der TCOM Objektarchivserver für die Erstellung eines Objektarchivs zum Schreiben benutzt, auf das über die Schnittstelle ITComArchiveOp zugegriffen wird. Diese Schnittstelle stellt Überladungen von operator<<() zur Verfügung um im Archiv zu schreiben.

Die hier verwendete Schnittstelle ist nicht für den Echtzeit-Kontext [▶ 48] entwickelt, sodass die Schnittstelle nur im Non Real-Time Kontext genutzt werden kann.

Dokumente hierzu

https://infosys.beckhoff.com/content/1031/TC3_C/Resources/10340367755.zip

15.25 TcCOM Beispiele

Module zwischen SPS und C++ können kommunzieren, so dass sowohl auf Seiten der SPS beschrieben wird, wie mit C++ Modulen umgegangen werden kann, wie auch auf C++ Seite, wie mit der SPS umgegangen werden kann.
An dieser Stelle sind die TcCOM Beispiele zur Kommunikation mit der SPS dargestellt.

Im Beispiel TcCOM Sample01 [▶ 323] wird dargestellt, wie eine TcCOM-Kommunikation zwischen zwei SPS stattfinden kann. Dabei werden aus der einen SPS heraus Funktionalitäten der anderen SPS direkt aufgerufen.

Im Beispiel TcCOM Sample02 [▶ 333] wird dargestellt, wie eine SPS-Applikation Funktionalitäten einer existierenden Instanz einer TwinCAT C++ Klasse nutzen kann. Eigene in C++ (oder Matlab®) geschriebene Algorithmen lassen sich so leicht in der SPS verwenden.
Bei Verwendung eines existierenden TwinCAT C++ Moduls bedarf es zwar der TwinCAT C++ Lizenz auf dem Zielsystem, eine C++ Entwicklungsumgebung muss jedoch weder auf dem Zielsystem noch auf dem Entwicklungsrechner vorhanden sein.

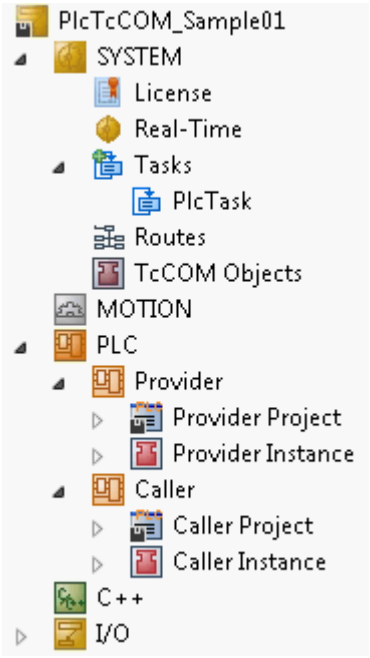
Im Beispiel TcCOM Sample03 [▶ 337] wird dargestellt, wie eine SPS-Applikation Funktionalitäten einer TwinCAT C++ Klasse nutzt, indem zugleich eine Instanz der C++ Klasse erzeugt wird. Dies kann im Vergleich zum vorherigen Sample eine erhöhte Flexibilität bieten.

15.25.1 TcCOM_Sample01_PlcToPlc

Dieses Beispiel beschreibt eine TcCOM-Kommunikation zwischen zwei SPS.

Funktionalitäten, die von einem Funktionsbaustein in der ersten SPS (im Beispiel auch „Provider“ genannt) bereitgestellt werden, werden aus der zweiten SPS (im Beispiel auch „Caller“ genannt) heraus aufgerufen. Dazu muss der Funktionsbaustein oder dessen Programmcode nicht kopiert werden, sondern es wird direkt mit der Objektinstanz, die sich in der ersten SPS befindet, gearbeitet.

Beide SPS müssen sich in einer TwinCAT-Laufzeit befinden. Ein Funktionsbaustein bietet hierbei seine Methoden über eine global definierte Schnittstelle systemweit an und stellt selbst ein TcCOM-Objekt dar. Wie jedes TcCOM-Objekt wird auch ein solcher Funktionsbaustein zur Laufzeit im Knoten **TcCOM Objects** gelistet.



Die Vorgehensweise wird in folgenden Unterkapiteln erläutert:

- Erstellen eines FBs in der ersten SPS, der seine Funktionalität global bereitstellt [▶ 325]
- Erstellen eines FBs in der zweiten SPS, der als einfacher Proxy diese Funktionalität dort ebenfalls anbietet [▶ 329]
- Ausführung des Beispielprojektes [▶ 331]

Download des Beispiels: https://infosys.beckhoff.com/content/1031/TC3_C/Resources/2343046667.zip

Race Conditions bei Multi-Tasking (Multi-Threading)-Verwendung

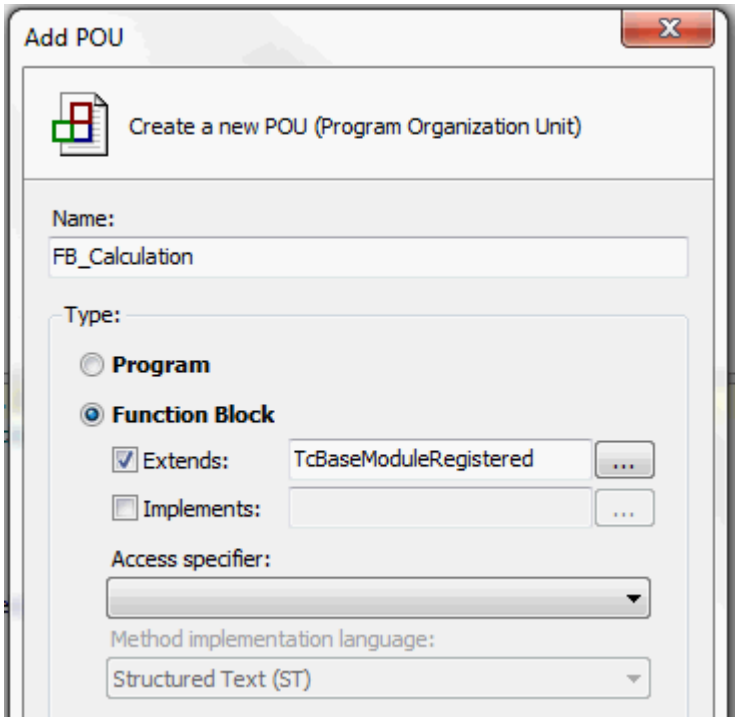
Der Funktionsbaustein, der seine Funktionalität global zur Verfügung stellt, wird in der ersten SPS instanziiert. Dort kann er wie jeder Funktionsbaustein verwendet werden. Wenn er außerdem aus einer anderen SPS (oder bspw. einem C++ Modul) verwendet wird, achten Sie darauf, dass die angebotenen Methoden thread-sicher sind, da die verschiedenen Aufrufe je nach Systemkonfiguration zeitgleich aus unterschiedlichen Taskkontexten erfolgen oder sich gegenseitig unterbrechen könnten. In diesem Fall dürfen die Methoden nicht auf Membervariablen des Funktionsbausteins oder globale Variablen der ersten SPS zugreifen. Sollte dies zwingend notwendig sein, beugen Sie einem zeitgleichen Zugriff vor. Beachten Sie die Funktion TestAndSet() aus der Tc2_System Bibliothek.

Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, Arm®	Tc3_Module

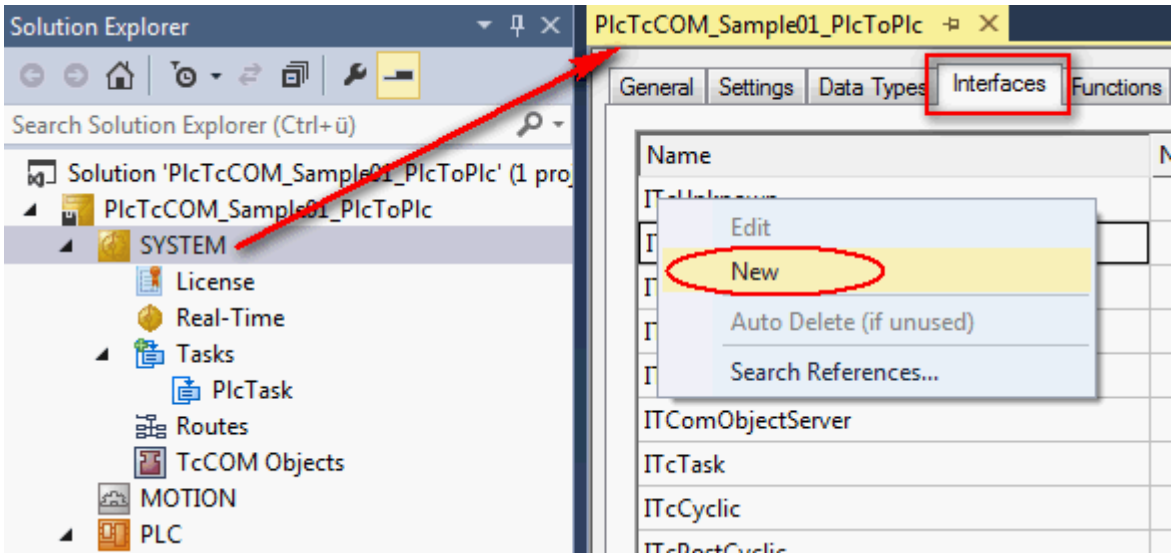
15.25.1.1 Erstellen eines FBs in der ersten SPS, welcher seine Funktionalität global bereitstellt

1. Legen Sie eine SPS an und erstellen Sie einen neuen Funktionsbaustein (FB) (hier: FB_Calculation). Leiten Sie den Funktionsbaustein von der Klasse TcBaseModuleRegistered ab, damit eine Instanz dieses Funktionsbausteins nicht nur in der gleichen SPS verfügbar, sondern auch aus einer zweiten SPS heraus erreichbar ist. Alternativ können Sie auch einen FB in einer bestehenden SPS modifizieren.

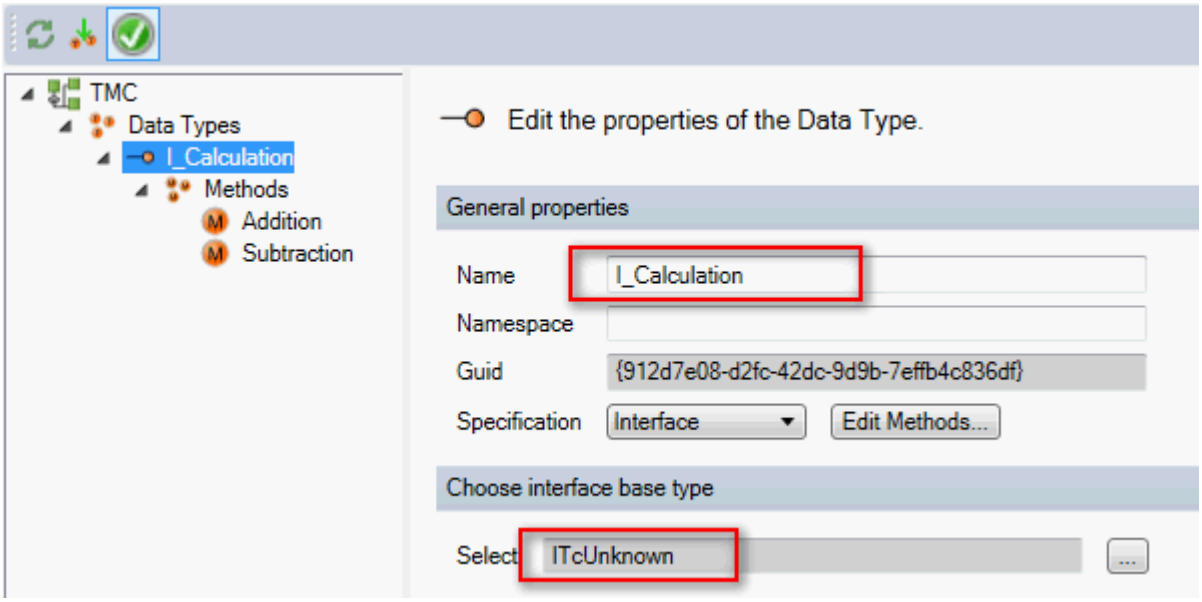


2. Der Funktionsbaustein muss seine Funktionalität mittels Methoden anbieten. Diese werden in einer globalen Schnittstelle definiert, deren Typ systemweit und programmiersprachenunabhängig bekannt ist. Um ein globales Interface anzulegen, öffnen Sie im Reiter „Interface“ der Systemeigenschaften das Kontextmenü und wählen Sie die Option „New“ aus.

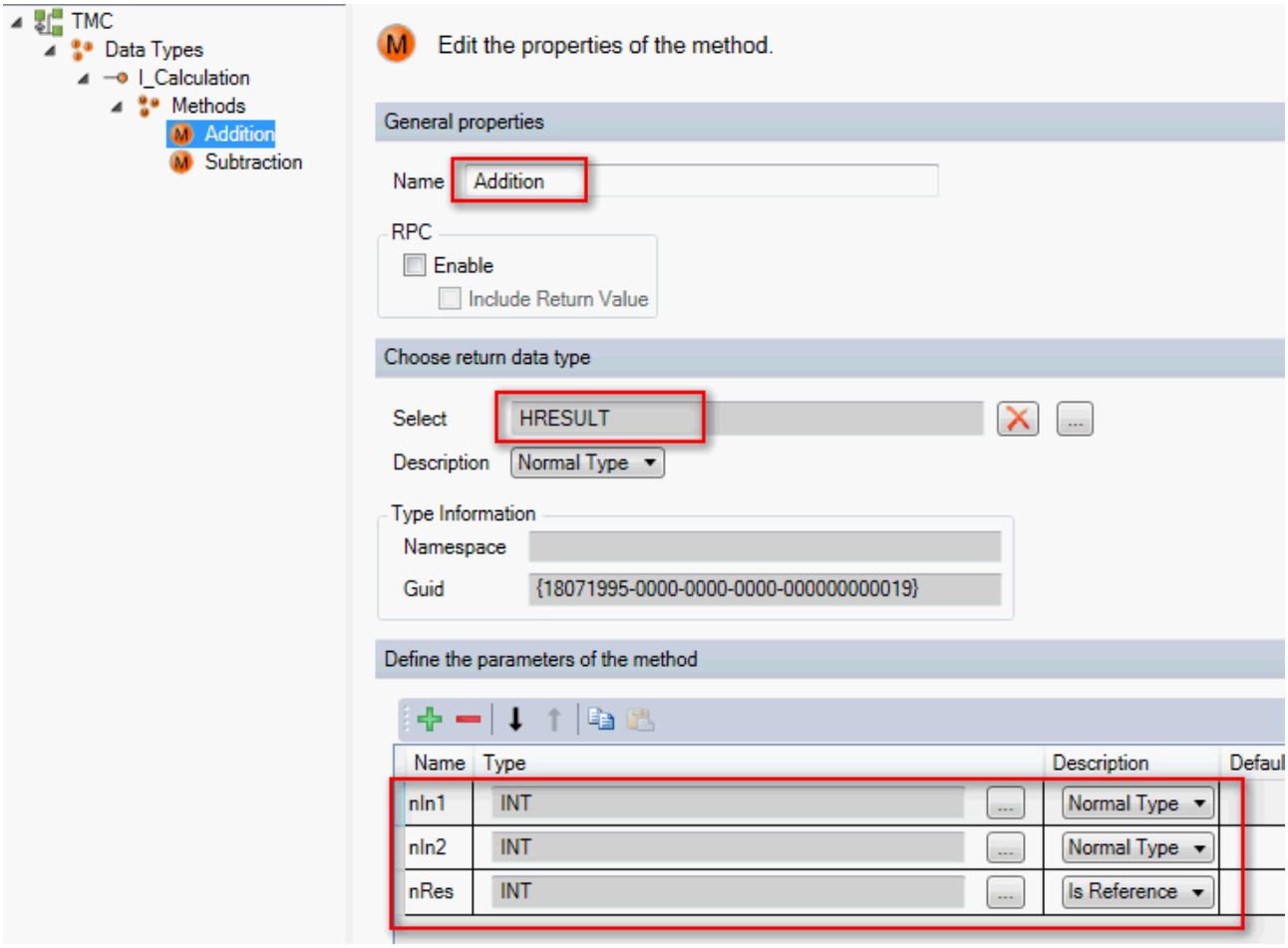
   ⇨ Es öffnet sich der TMC Editor, welcher Sie darin unterstützt ein globales Interface anzulegen.



3. Spezifizieren Sie den Namen (hier: I_Calculation) und fügen Sie die gewünschten Methoden an. Das Interface wird automatisch von ITcUnknown abgeleitet, um dem TwinCAT TcCOM-Modulkonzept gerecht zu werden.



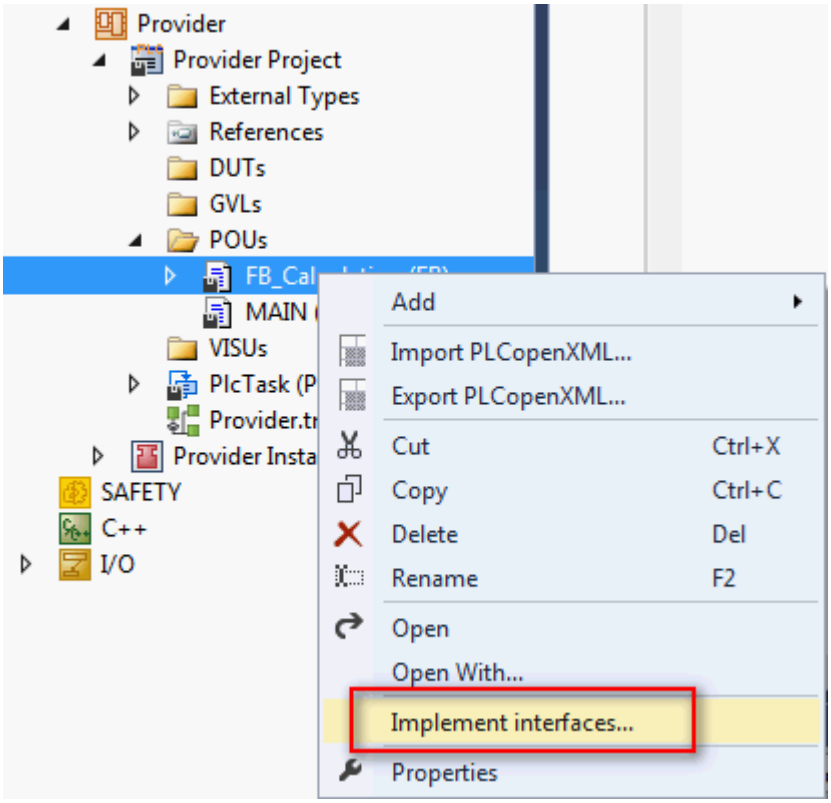
4. Geben Sie analog den Namen der Methoden an (hier: Addition() und Subtraction()) und wählen Sie als Rückgabedatentyp HRESULT. Dieser Rückgabetyp ist zwingend vorgeschrieben, wenn diese Art der TcCOM-Kommunikation implementiert werden soll.
5. Spezifizieren Sie zuletzt die Methodenparameter und schließen dann den TMC Editor.



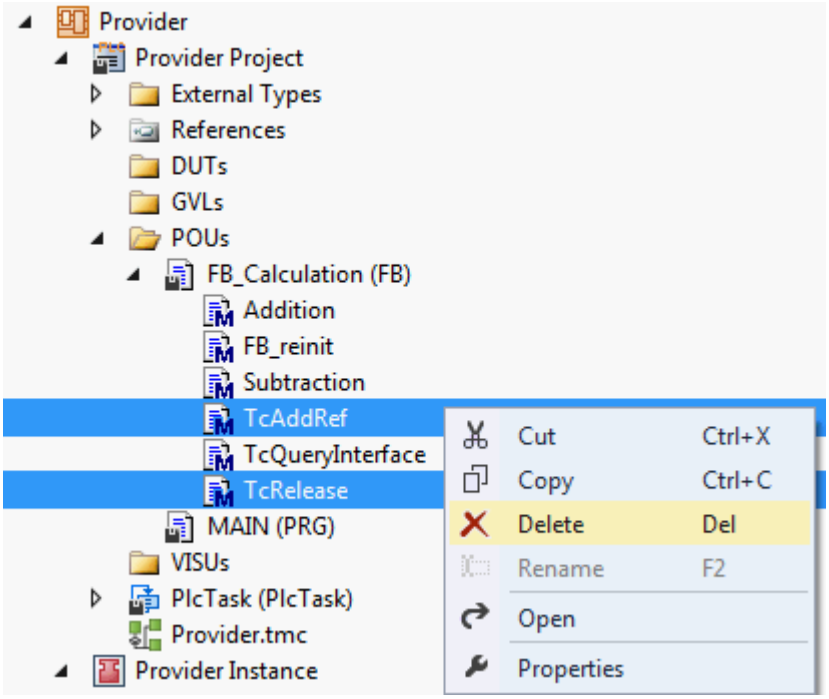
6. Implementieren Sie nun im Funktionsbaustein FB_Calculation die Schnittstelle I_Calculation und fügen Sie das Attribut c++_compatible an.

```
{attribute 'c++_compatible'}
FUNCTION_BLOCK FB_Calculation EXTENDS TcBaseModuleRegistered IMPLEMENTS I_Calculation

VAR
END_VAR
```

7. Wählen Sie im Kontextmenü des Funktionsbausteins die Option „Implement interfaces...“ aus, um die zu dieser Schnittstelle gehörenden Methoden zu erhalten.



8. Löschen Sie die beiden Methoden TcAddRef() und TcRelease(), weil hiervon die bereits vorhandene Implementierung der Basisklasse verwendet werden soll.



9. Legen Sie für den Funktionsbaustein FB_Calculation die Methode FB_reinit() an und rufen Sie die Basisimplementierung auf. Hierdurch wird gewährleistet, dass die Methode FB_reinit() der Basisklasse beim Online Change durchlaufen wird. Dies ist zwingend notwendig.

FB_Calculation.FB_reinit

```
1  METHOD FB_reinit : BOOL
2  VAR_INPUT
3  END_VAR
4
```

```
1  SUPER^.FB_reinit();
2
```

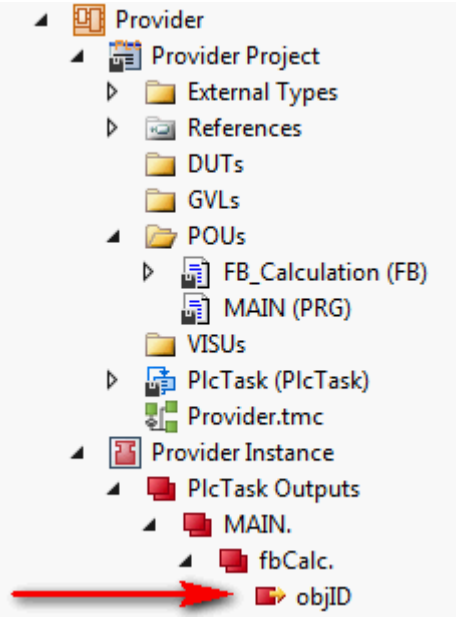
10. Implementieren Sie die Methode TcQueryInterface() der Schnittstelle ITcUnknown [▶ 197]. Über diese Methode ist es anderen TwinCAT Komponenten möglich, einen Schnittstellenzeiger auf eine Instanz dieses Funktionsbausteines zu erhalten und damit Methodenaufrufe zu tätigen. Der Aufruf von TcQueryInterface ist erfolgreich, wenn der Funktionsbaustein oder seine Basisklasse die mittels iid (Interface-ID) angefragte Schnittstelle bereitstellt. Für diesen Fall wird dem übergebenen Schnittstellenzeiger die Adresse auf den Funktionsbaustein typgewandelt zugewiesen und der Referenzzähler mittels TcAddRef() erhöht.
11. Füllen Sie die beiden Methoden Addition() und Subtraction() mit entsprechendem Code, um die Funktionalität zu erbringen: `nRes := nIn1 + nIn2` und `nRes := nIn1 - nIn2`
12. Fügen Sie eine oder mehrere Instanzen dieses Funktionsbausteins im Programmbaustein MAIN oder in einer globalen Variablenliste hinzu.

    ⇨ Die Implementierung in der ersten SPS ist vollständig.

MAIN*

```
1  PROGRAM MAIN
2  VAR
3      m : UDINT;
4
5      fbCalc : FB_Calculation('MAIN.fbCalc');
6  END_VAR
7
```

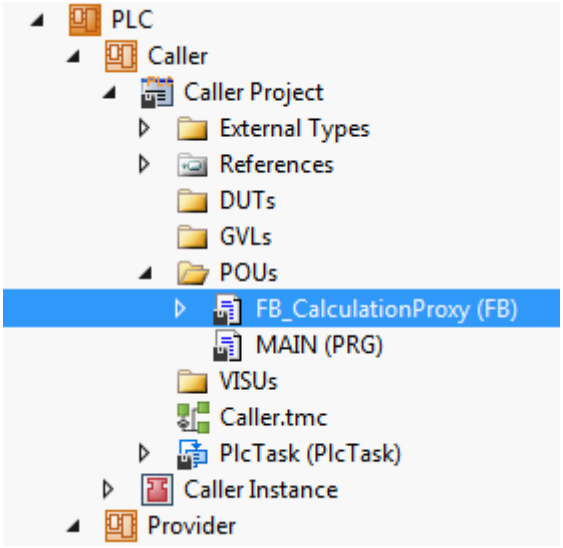
⇨ Nach dem Kompilieren der SPS ist im Prozessabbild die Objekt-ID des TcCOM-Objektes, welches die Instanz von FB_Calculation representiert, als Ausgang verfügbar.



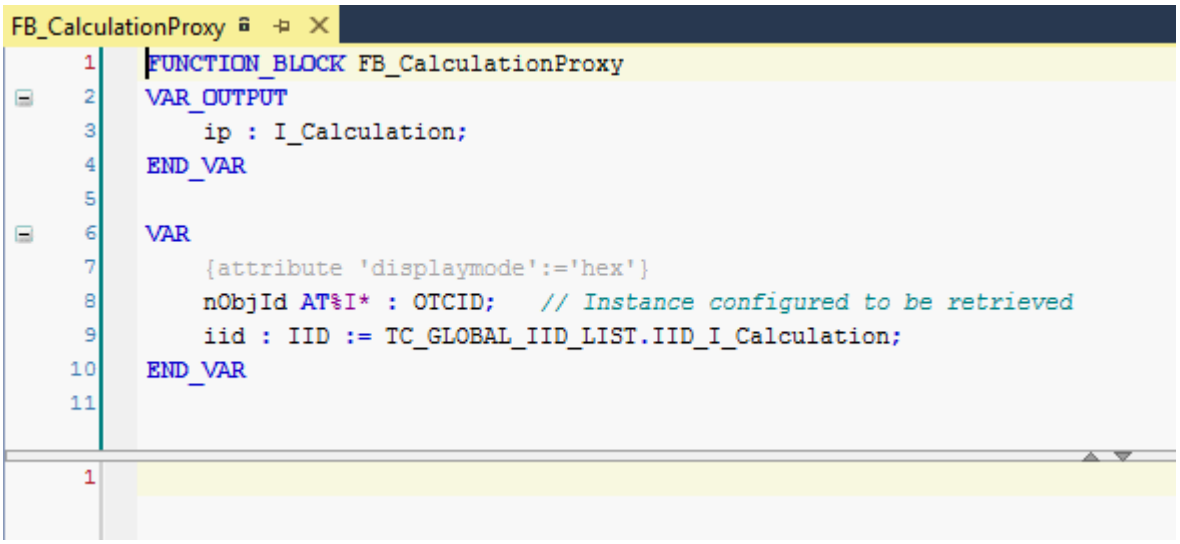
15.25.1.2 Erstellen eines FBs in der zweiten SPS, welcher als einfacher Proxy diese Funktionalität dort ebenfalls anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.

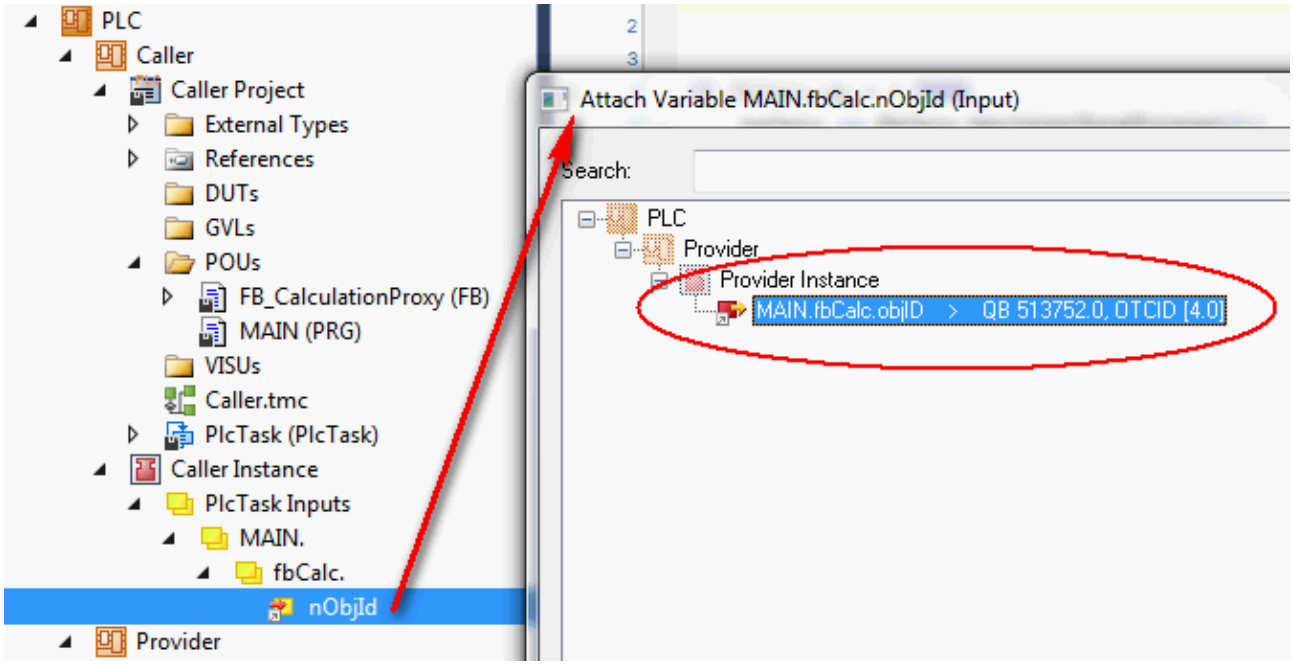
   ⇨ Dieser Proxy-Baustein soll die Funktionalität bereitstellen, welche in der ersten SPS programmiert wurde. Dies kann er über einen Schnittstellenzeiger vom Typ der globalen Schnittstelle I_Calculation.



2. Deklarieren Sie im Deklarationsteil des Funktionsbaustein als Ausgang einen Schnittstellenzeiger auf die globale Schnittstelle, welche später die Funktionalität nach außen bereitstellt.



3. Legen Sie zudem die Objekt-ID und die Schnittstellen-ID als lokale Membervariablen an.
   Während die Schnittstellen-ID über eine globale Liste bereits verfügbar ist, wird die Objekt-ID über eine Verknüpfung im Prozessabbild zugewiesen.



4. Implementieren Sie den SPS Proxy-Baustein. Zuerst fügen Sie dem Baustein die Methode GetInterfacePointer() hinzu.
   Der Schnittstellenzeiger wird auf die spezifizierte Schnittstelle des spezifizierten TcCOM-Objektes mit Hilfe der Funktion FW_ObjMgr_GetObjectInstance() geholt. Dies wird nur ausgeführt, wenn die Objekt-ID gültig und der Schnittstellenzeiger nicht bereits zugewiesen ist. Das Objekt selbst zählt einen Referenzzähler hoch.

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. Es ist zwingend notwendig die verwendete Referenz wieder freizugeben. Rufen Sie hierzu die Funktion FW_SafeRelease() im FB_exit Destruktor des Bausteines auf.

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
```

⇨ Damit ist die Implementierung des Proxy-Funktionsbausteines bereits abgeschlossen.

6. Instanziieren Sie in der Applikation den Proxy-Funktionsbaustein FB_CalculationProxy und rufen Sie dessen Methode GetInterfacePointer() auf, um einen gültigen Schnittstellenzeiger zu erhalten.
   Zum Aufruf der über die Schnittstelle bereitgestellten Methoden wird in der Applikation eine Instanz des Proxy-Bausteines deklariert. Die Aufrufe selbst finden alle über den als Ausgang des Bausteines definierten Schnittstellenzeiger statt. Wie bei Zeigern typisch muss eine Überprüfung auf Null vorausgehen. Daraufhin können die Methoden, auch mittels Intellisense, direkt aufgerufen werden.

MAIN*

```
1  PROGRAM MAIN
2  VAR
3      fbCalc : FB_CalculationProxy;
4      hrCalc : HRESULT;
5      a : INT := 10;
6      b : INT := 7;
7      nSum : INT; // a + b
8      nDiff : INT; // a - b
9  END_VAR
10
```

```
1  IF fbCalc.ip = 0 THEN
2      hrCalc := fbCalc.GetInterfacePointer();
3  END_IF
4  IF fbCalc.ip <> 0 THEN
5      hrCalc := fbCalc.ip.Addition(a,b,nSum);
6      hrCalc := fbCalc.ip.Subtraction(a,b,nDiff);
7  END_IF
8
```

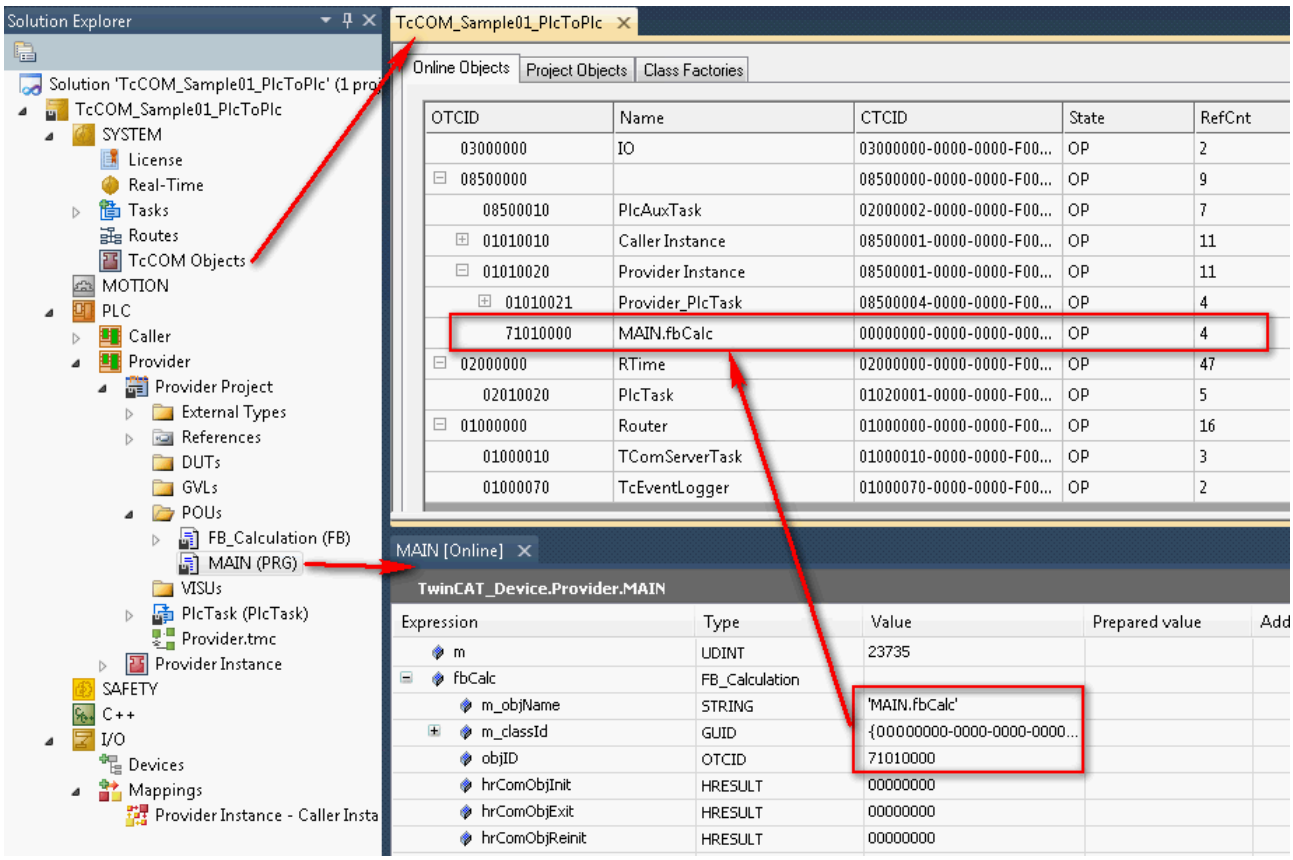
⇨ Das Beispiel ist bereit zum Test.

Reihenfolge irrelevant

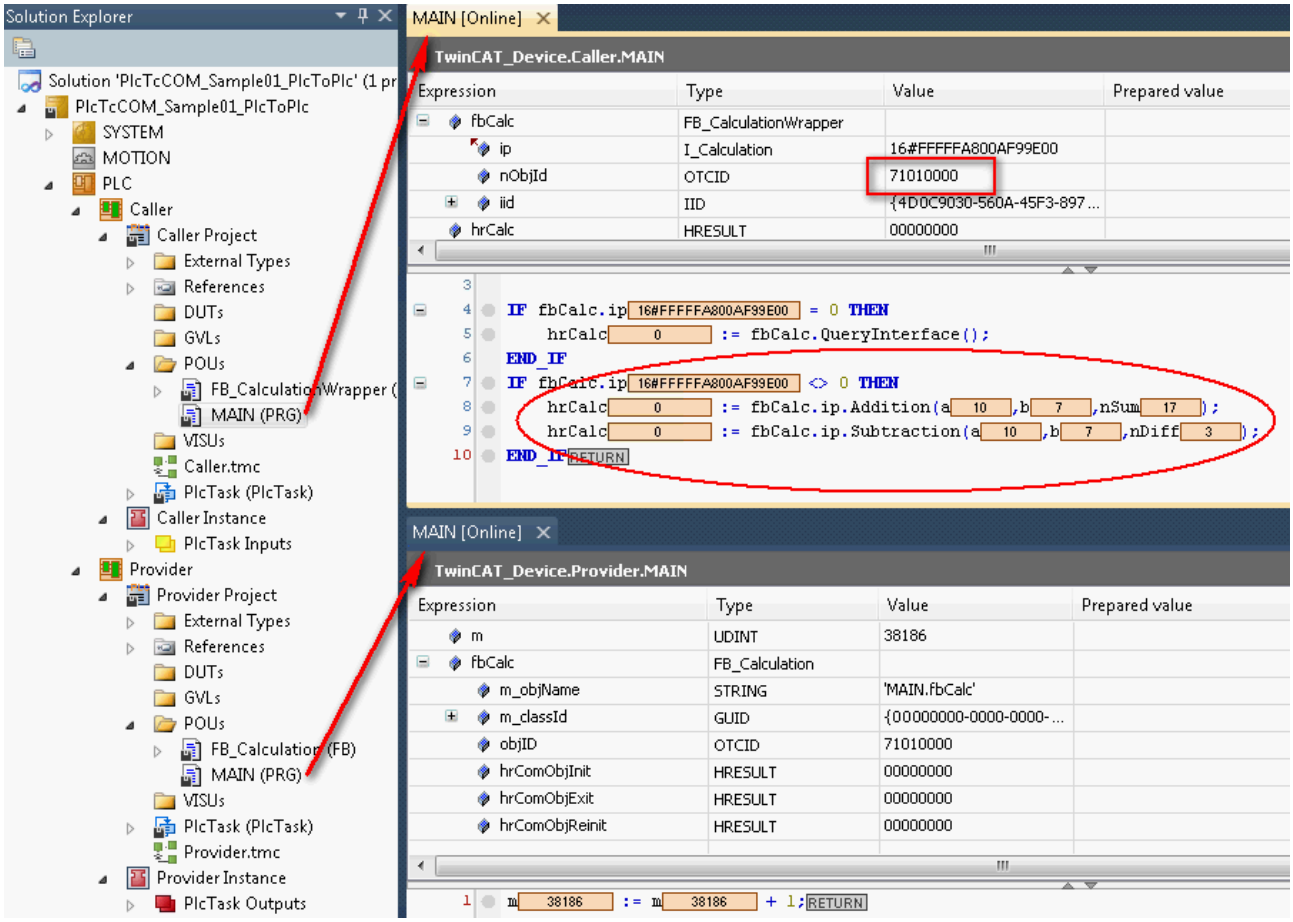
In welcher Reihenfolge die zwei SPS später starten, ist bei dieser Implementierung irrelevant.

15.25.1.3 Ausführung des Beispielprojektes

1. Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
2. Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten beider SPSn aus.
   ⇨ In der Onlineansicht der SPS-Applikation „Provider“ ist die generierte Objekt-ID des C++ Objektes im SPS Baustein FB_Calculation ersichtlich. Der Projektknoten „TcCOM Objekte“ führt das erzeugte Objekt mit dieser Objekt-ID und dem gewählten Namen in seiner Liste.



⇨ In der Onlineansicht der SPS Applikation „Caller“ hat der Proxy-Funktionsbaustein die gleiche Objekt-ID über das Prozessabbild zugewiesen bekommen. Der Schnittstellenzeiger hat einen gültigen Wert und die Methoden werden ausgeführt.



15.25.2 TcCOM_Sample02_PlcToCpp

Dieses Beispiel beschreibt eine TcCOM-Kommunikation zwischen SPS und C++. Hierbei nutzt eine SPS-Applikation Funktionalitäten einer existierenden Instanz einer TwinCAT C++ Klasse. Eigene in C++ geschriebene Algorithmen lassen sich so leicht in der SPS verwenden.
Bei Verwendung eines existierenden TwinCAT C++ Moduls bedarf es zwar der TwinCAT C++ Lizenz auf dem Zielsystem, eine C++ Entwicklungsumgebung muss jedoch weder auf dem Zielsystem noch auf dem Entwicklungsrechner vorhanden sein.

Ein bereits gebauter C++ Modul stellt eine oder mehrere Klassen zur Verfügung, deren Schnittstellen in der TMC-Beschreibungsdatei hinterlegt und somit in der SPS bekannt sind.

Die Vorgehensweise wird in folgenden Unterkapiteln erläutert:

1. Instanziieren einer TwinCAT C++ Klasse als TwinCAT TcCOM Objekt [▶ 333]

2. Erstellen eines FBs in der SPS, welcher als einfacher Wrapper die Funktionalität des C++ Objektes anbietet [▶ 334]

3. Ausführung des Beispielprojektes [▶ 336]

Download des Beispiels: https://infosys.beckhoff.com/content/1031/TC3_C/Resources/2343048971.zip

Systemvoraussetzungen

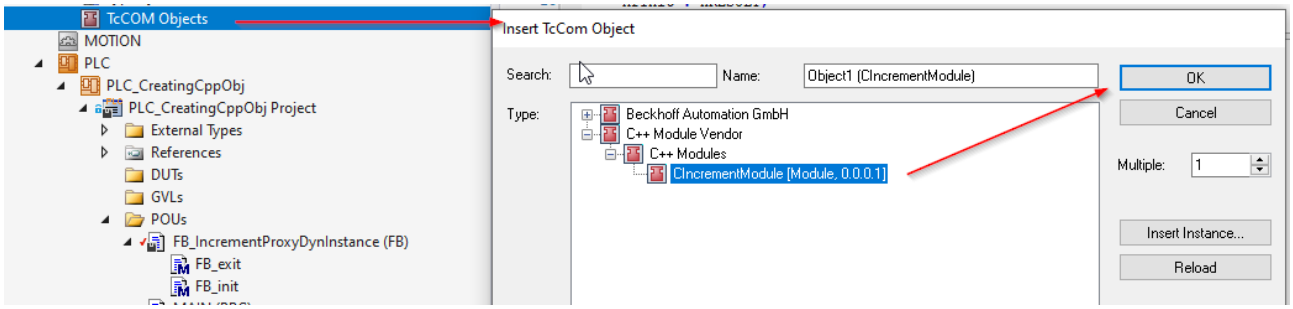
TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64	Tc3_Module

15.25.2.1 Instanziieren einer TwinCAT C++ Klasse als TwinCAT TcCOM Objekt

Der binäre TwinCAT C++ Projekt muss auf dem Engineering-System zur Verfügung stehen, sodass es zusammen mit dem SPS-Projekt bei der Aktivierung auf das Zielsystem übertragen werden kann.

TwinCAT bietet für die Verteilung zwischen Engineering-Systemen einen Deployment-Mechanismus, der unter Versionierte C++ Projekte [▶ 53] beschrieben ist.
(Dieses Beispiel beinhaltet im Download das entsprechende TMX, da TwinCAT diese automatisch in das Archiv platziert, falls die Class Factory verwendet wird.)

1. Öffnen Sie ein TwinCAT Projekt oder legen Sie ein neues Projekt an.
2. Fügen Sie in der Solution unter dem Knotenpunkt **TcCOM Objekte** eine Instanz der Klasse CIncrementModule hinzu.

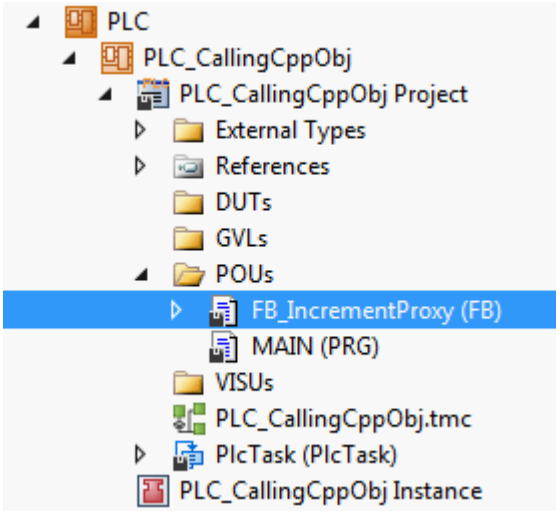


Erstellung der C++ Module

In der Dokumentation zu TwinCAT C++ [▶ 12] wird ausführlich erläutert, wie C++ Module für TwinCAT erstellt werden.

15.25.2.2 Erstellen eines FBs in der SPS, der als einfacher Proxy die Funktionalität des C++ Objektes anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.
   Dieser Proxy-Baustein soll die Funktionalität bereitstellen, die in C++ programmiert wurde. Dies kann er über einen Schnittstellenzeiger, der von der C++ Klasse definiert wurde und aufgrund der TMC-Beschreibungsdatei in der SPS bekannt ist.



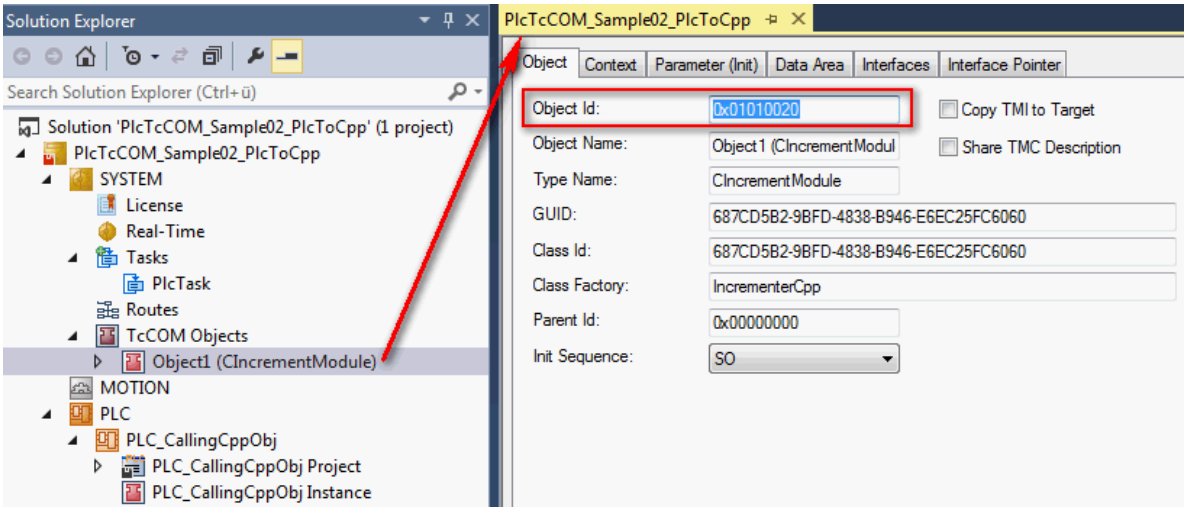
2. Deklarieren Sie im Deklarationsteil des Funktionsbausteins als Ausgang einen Schnittstellenzeiger auf die Schnittstelle, der später die Funktionalität nach außen bereitstellt.
3. Legen Sie die Objekt-ID und die Schnittstellen-ID als lokale Membervariablen an.
   Während die Schnittstellen-ID über eine globale Liste bereits verfügbar ist, wird die Objekt-ID über die TwinCAT-Symbol-Initialisierung zugewiesen. Das Attribut TcInitSymbol sorgt dafür, dass die Variable in einer Liste auftaucht, die der externen Symbolinitialisierung dient. Zugewiesen werden soll die Objekt-ID des angelegten C++ Objektes.

FB_IncrementProxy

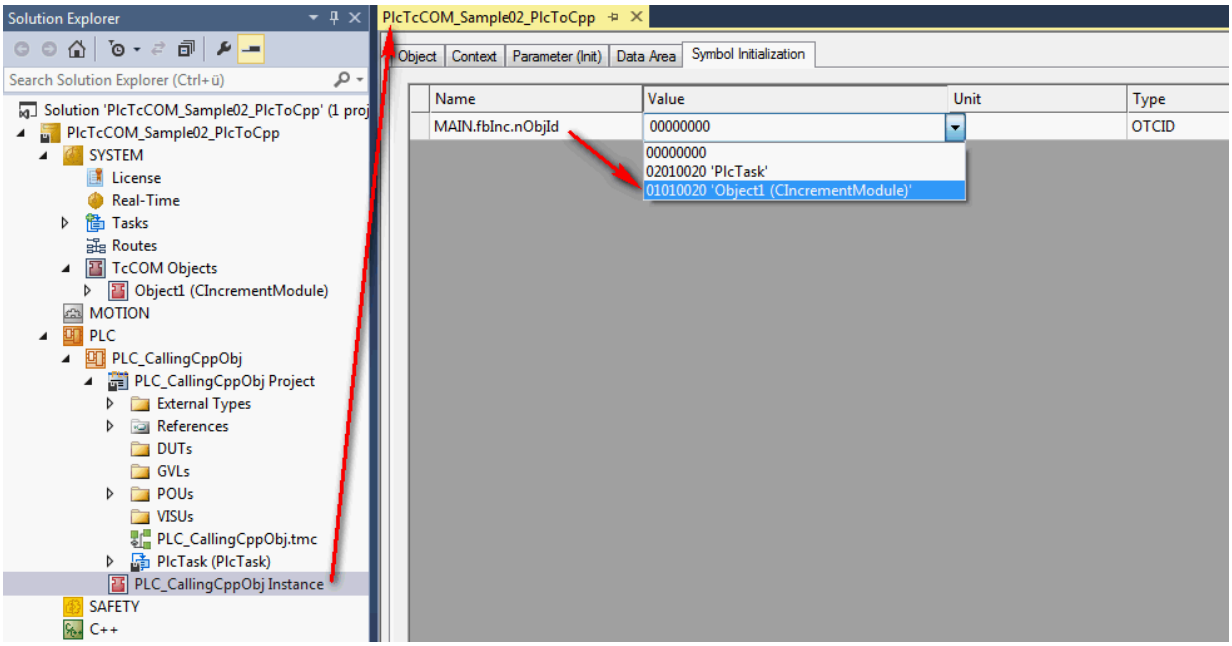
```
1  FUNCTION_BLOCK FB_IncrementProxy
2  VAR_OUTPUT
3      ip : IIncrement;
4  END_VAR
5
6  VAR
7      {attribute 'TcInitSymbol'}
8      {attribute 'displaymode':='hex'}
9      nObjId : OTCID;     // Instance configured to be retrieved
10     iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
11     hrInit : HRESULT;
12 END_VAR
13
```

```
1
```

⇨ Die Objekt-ID wird bei Anwahl des Objektes unter dem Knoten **TcCOM-Objekte** angezeigt. Die Liste der Symbol-Initialisierungen befindet sich - sofern das Attribut TcInitSymbolverwendet wurde - im Knotenpunkt der SPS-Instanz im Reiter **Symbol Initialisierung**.



4. Weisen Sie hier dem Symbolnamen der Variablen per Dropdown eine vorhandene Objekt-ID zu. Beim Download der SPS wird dieser Wert zugewiesen, um so bereits vor der SPS-Laufzeit festgelegt zu sein. Neue Symbolinitialisierungen oder Änderungen werden demnach mit einem neuen Download der SPS eingespielt.



Die Übergabe der Objekt-ID könnte alternativ auch per Prozessabbildverknüpfung wie im ersten Beispiel implementiert werden (TcCOM_Sample01_PlcToPlc [▶ 323]).

5. Implementieren Sie den SPS Proxy-Baustein.
Zuerst wird dem Baustein die FB_init Konstruktormethode hinzugefügt. Für den Fall, dass es sich nicht um einen OnlineChange sondern um die Initialisierung des Bausteins handelt, wird der Schnittstellenzeiger auf die spezifizierte Schnittstelle des spezifizierten TcCOM-Objektes mit Hilfe der

Funktion FW_ObjMgr_GetObjectInstance() geholt. Hierbei zählt das Objekt selbst einen Referenzzähler hoch.

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. Es ist zwingend notwendig, die verwendete Referenz wieder freizugeben. Rufen Sie hierzu die Funktion FW_SafeRelease() im FB_exit Destruktor des Bausteins auf.

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
```

⇨ Damit ist die Implementierung des Proxy-FunktionsBausteins bereits abgeschlossen.

7. Deklarieren Sie zum Aufruf der über die Schnittstelle bereitgestellten Methoden in der Applikation eine Instanz des Proxy-Bausteins.
   Die Aufrufe selbst finden alle über den als Ausgang des Bausteins definierten Schnittstellenzeiger statt. Wie bei Zeigern typisch, muss eine Überprüfung auf Null vorrausgehen. Daraufhin können die Methoden, auch mittels Intellisense, direkt aufgerufen werden.

MAIN*

```
PROGRAM MAIN
VAR
    fbInc : FB_IncrementProxy;
    nValue : UDINT;
END_VAR
```

```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(4, ADR(nValue));
END_IF
```

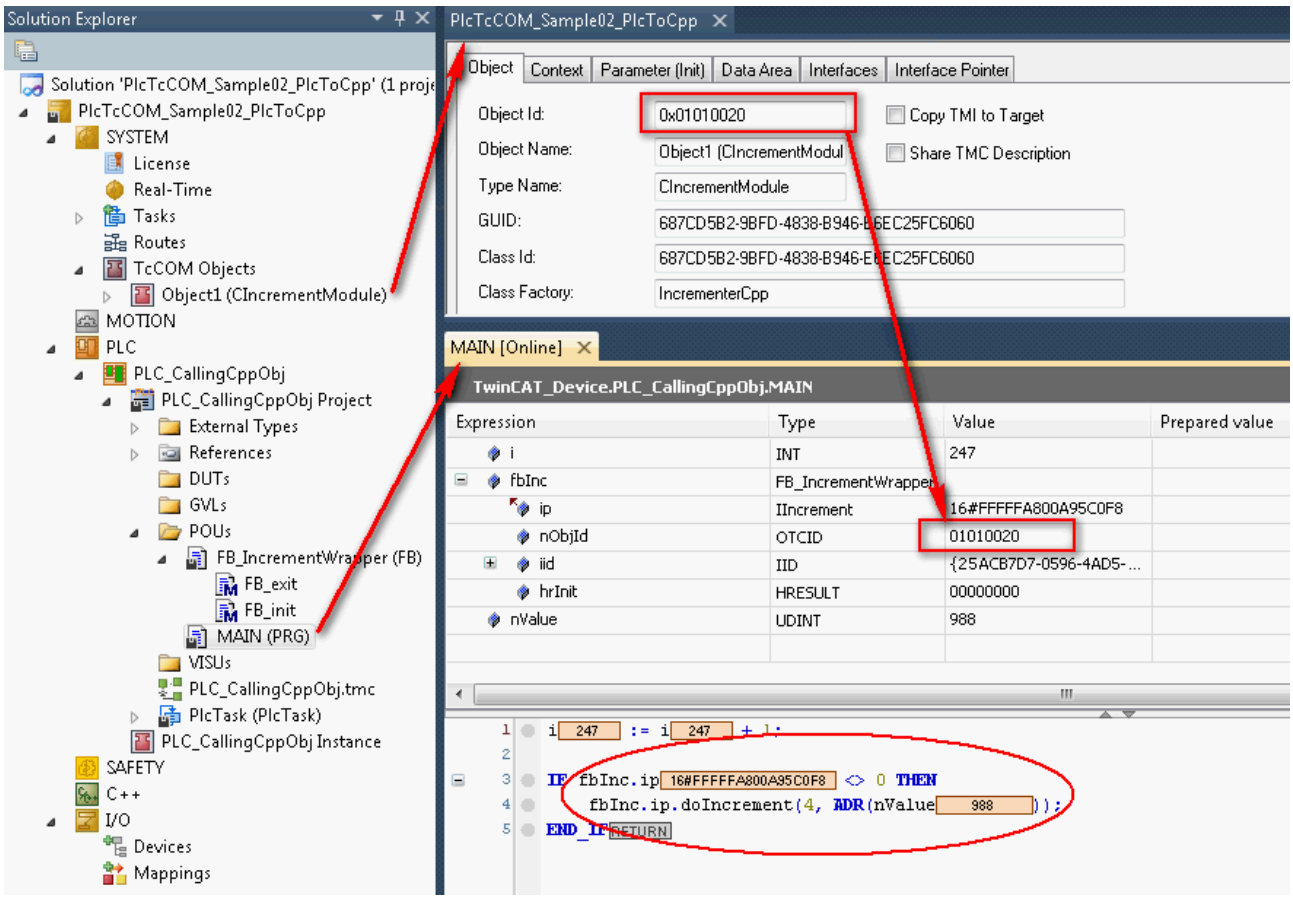
⇨ Das Beispiel ist bereit zum Test.

15.25.2.3 Ausführung des Beispielprojektes

1. Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
2. Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten der SPS aus.

3. Dieses C++ TcCOM Modul verlangt den Windows Test-Mode. Aktivieren Sie diesen, um das Modul und damit das Sample nutzen zu können.
4. Beim ersten Aktivieren der TwinCAT-Konfiguration ist das Test-Zertifikat zu akzeptieren:

⇨ In der Onlineansicht der SPS-Applikation ist die zugewiesene Objekt-ID des C++ Objektes im SPS Proxy-Bausteins ersichtlich. Der Schnittstellenzeiger hat einen gültigen Wert und die Methode wird ausgeführt.



15.25.3 TcCOM_Sample03_PlcCreatesCpp

Dieses Beispiel beschreibt, ebenso wie Sample02, eine TcCOM-Kommunikation zwischen SPS und C++. Hierbei nutzt eine SPS Applikation Funktionalitäten einer TwinCAT C++ Klasse. Die benötigten Instanzen dieser C++ Klasse werden in diesem Beispiel von der SPS selbst angelegt. Eigene in C++ geschriebene Algorithmen lassen sich so leicht in der SPS verwenden.
Bei Verwendung eines existierenden TwinCAT C++ Treibers bedarf es zwar der TwinCAT C++ Lizenz auf dem Zielsystem, eine C++ Entwicklungsumgebung muss jedoch weder auf dem Zielsystem noch auf dem Entwicklunsgrechner vorhanden sein.

Ein bereits gebauter C++ Treiber stellt eine oder mehrere Klassen zur Verfügung, deren Schnittstellen in der TMC-Beschreibungsdatei hinterlegt und somit in der SPS bekannt sind.

Die Vorgehensweise wird in folgenden Unterkapiteln erläutert:

1. Bereitstellen eines TwinCAT C++ Treibers und seiner Klassen [▶ 338]

2. Erstellen eines FBs in der SPS, welcher das C++ Objekt anlegt und dessen Funktionalität anbietet [▶ 339]

3. Ausführung des Beispielprojektes [▶ 341]

Download des Beispiels: https://infosys.beckhoff.com/content/1031/TC3_C/Resources/2343051531.zip

Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64	Tc3_Module

15.25.3.1 Bereitstellen des binären C++ Projekts (TMX) und seiner Klassen

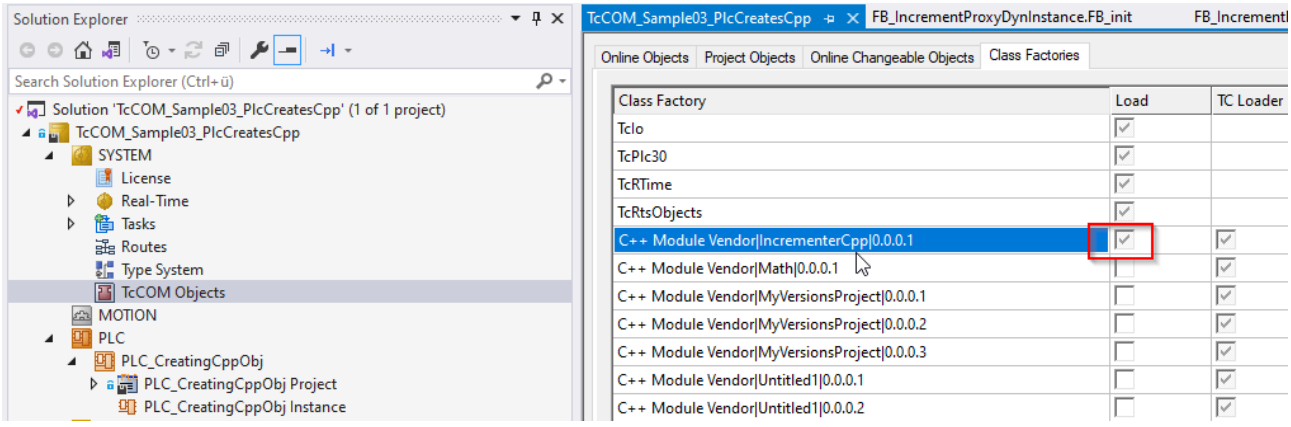
Der binäre TwinCAT C++ Projekt muss auf dem Engineering-System zur Verfügung stehen, sodass es zusammen mit dem SPS-Projekt bei der Aktivierung auf das Zielsystem übertragen werden kann.

TwinCAT bietet für die Verteilung zwischen Engineering-Systemen einen Deployment-Mechanismus, der unter Versionierte C++ Projekte [▶ 53] beschrieben ist.
(Dieses Beispiel beinhaltet im Download das entsprechende TMX, da TwinCAT diese automatisch in das Archiv platziert, falls die Class Factory verwendet wird.)

Öffnen Sie ein TwinCAT-Projekt oder legen Sie ein neues Projekt an.

1. Wählen Sie in der Solution unter dem Knotenpunkt **TcCOM-Objekte** im Reiter **Class Factories** den benötigten C++ aus. Die Checkbox kann auch von TwinCAT automatisch gesetzt werden, wenn Sie dieses (wie hier im Beispiel) entsprechend umsetzten.

⇨ So wird sichergestellt, dass der Treiber beim Starten von TwinCAT auf dem Zielsystem übertragen und geladen wird.

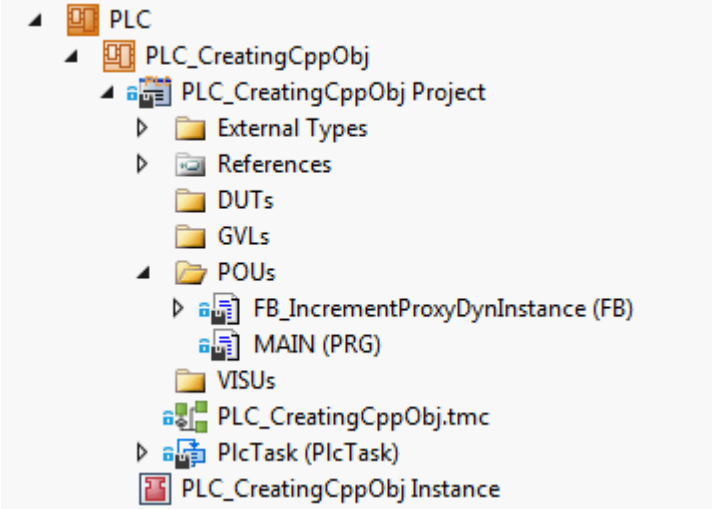


Erstellung des binären C++ Projektes

In der Dokumentation zu TwinCAT C++ [▶ 12] wird ausführlich erläutert, wie die TMX erstellt wird. Für Sample03 ist zu beachten, dass TwinCAT C++ Module, deren Klassen dynamisch instanziiert werden sollen, als „TwinCAT Module Class for RT Context“ definiert sein müssen. Der C++ Wizard bietet hierfür ein spezielles Template an.
Des Weiteren verwendet dieses Beispiel eine TwinCAT C++ Klasse, die ohne TcCOM-Initialisierungsdaten und ohne TcCOM-Parameter auskommt.

15.25.3.2 Erstellen eines FBs in der SPS, der das C++ Objekt anlegt und dessen Funktionalität anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.
   Dieser Proxy-Baustein soll die Funktionalität bereitstellen, die in C++ programmiert wurde. Dies kann er über einen Schnittstellenpointer, der von der C++ Klasse definiert wurde und aufgrund der TMC-Beschreibungsdatei in der SPS bekannt ist.

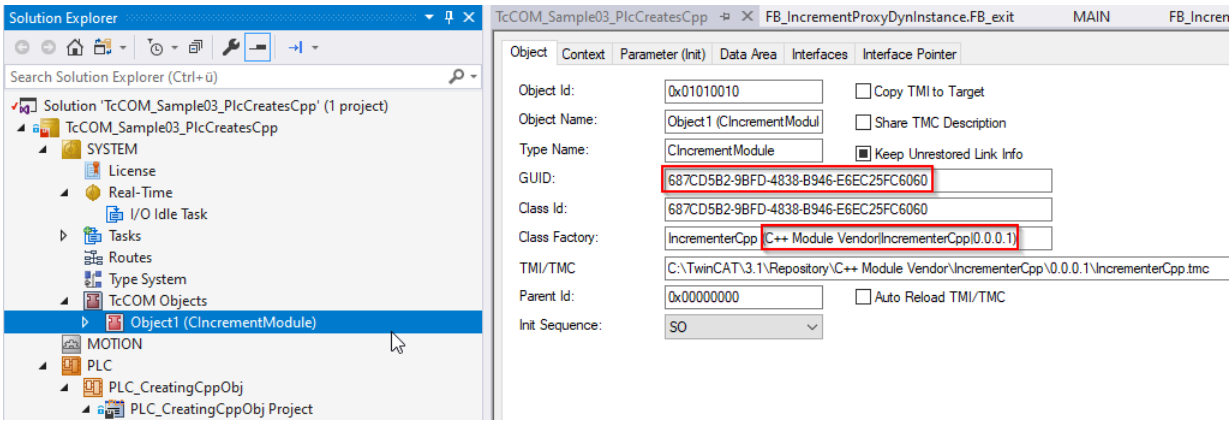


2. Deklarieren Sie im Deklarationsteil des Funktionsbausteins als Ausgang einen Schnittstellenzeiger auf die Schnittstelle (IIncrement), die später die Funktionalität nach außen bereitstellt.

```
{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. Legen Sie eine Library-ID, Klassen-ID und die Schnittstellen-ID als Membervariablen an, wie im vorherigen Schritt gezeigt.
   Während die Schnittstellen-ID über eine globale Liste verfügbar ist, wird die Library-ID und Klassen-ID - sofern sie noch nicht bekannt sein sollte - über einen anderen Weg ermittelt. Ein möglicher Weg hierfür ist eine Instanz zwischenzeitig anzulegen und die Informationen aus dem Dialog zu entnehmen, bevor

sie wieder gelöscht werden kann:



4. Fügen Sie dem SPS Proxy-Baustein die FB_init Konstruktormethode hinzu.
   Für den Fall, dass es sich nicht um einen Online Change sondern um die Initialisierung des Bausteins handelt, wird ein neues TcCOM-Objekt (Klasseninstanz der spezifizierten Klasse) angelegt und der Schnittstellenzeiger auf die spezifizierte Schnittstelle geholt.
   Als erstes wird mittels der Methode F_GetClassIdVersioned() aus der Library-ID sowie der Klassen-ID die versionierte Klassen-ID berechnet. Danach wird der verwendeten Funktion FW_ObjMgr_CreateAndInitInstance() auch der Name und der Zielzustand des TcCOM-Objektes mitgegeben. Diese zwei Parameter werden hier als Eingangsparameter der FB_init Methode deklariert, weshalb sie bei Instanziierung des Proxy-Bausteins anzugeben sind. Die zu instanziierende TwinCAT C++ Klasse kommt ohne TcCOM-Initialisierungsdaten und ohne TcCOM-Parameter aus. Bei diesem Funktionsaufruf zählt das Objekt selbst einen Referenzzähler hoch.

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. Es ist zwingend notwendig, die verwendete Referenz wieder freizugeben und das Objekt zu löschen, sofern es nicht mehr verwendet wird. Rufen Sie hierzu die Funktion FW_ObjMgr_DeleteInstance() im FB_exit Destruktor des Bausteins auf.

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
```

⇨ Damit ist die Implementierung des Proxy-Funktionsbausteins bereits abgeschlossen.

6. Deklarieren Sie zum Aufruf der über die Schnittstelle bereitgestellten Methoden in der Applikation eine Instanz des Proxy-Bausteins. Die Aufrufe selbst finden alle über den als Ausgang des Bausteins definierten Schnittstellenzeiger statt. Wie bei Zeigern typisch, muss eine Überprüfung auf Null vorrausgehen. Daraufhin können die Methoden, auch mittels Intellisense, direkt aufgerufen werden.

```
MAIN | FB_IncrementProxyDynInstance.FB_exit | FB_IncrementProxyDynInstance.FB_init | FB_IncrementProxyDynInstance

1  PROGRAM MAIN
2  VAR
3      fbInc : FB_IncrementProxyDynInstance(    sObjName:='CIncrementModule:fbInc',
4                                               eObjState:=TCOM_STATE.TCOM_STATE_OP);
5      nValue : UDINT;
6  END_VAR
7
```

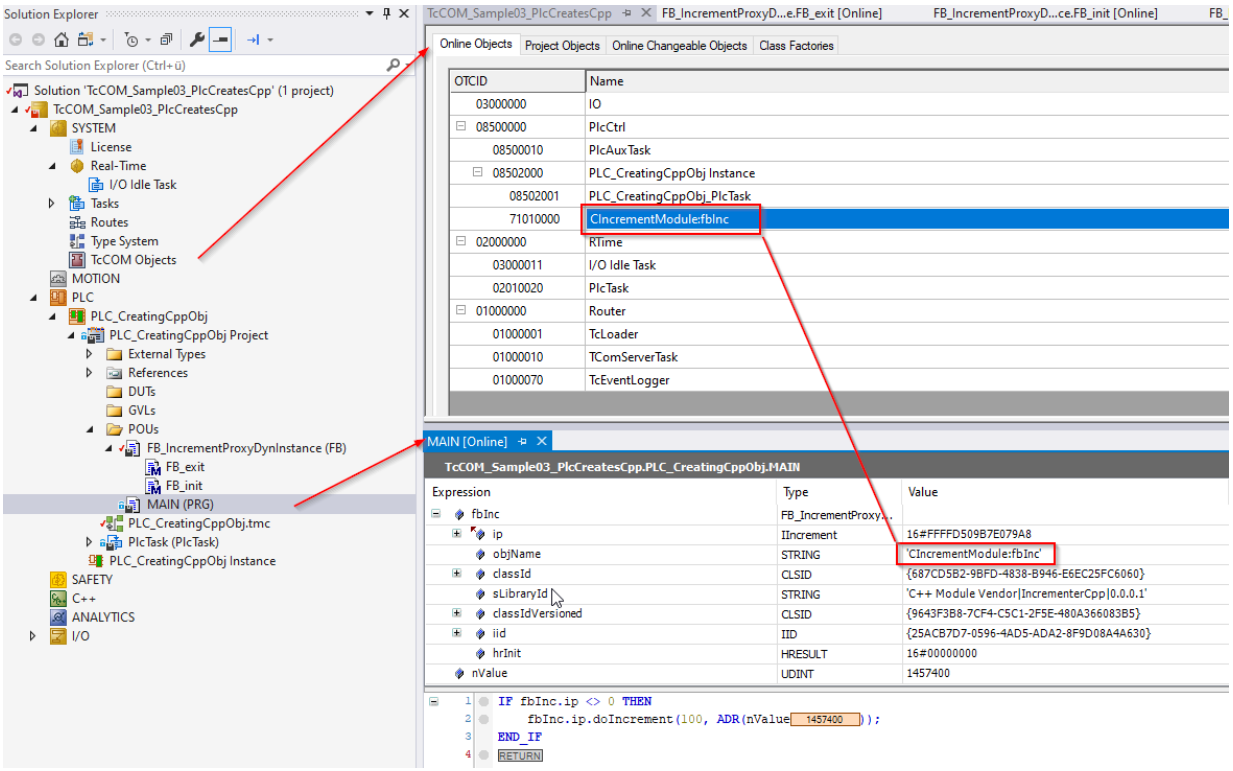
```
1  IF fbInc.ip <> 0 THEN
2      fbInc.ip.doIncrement(100, ADR(nValue));
3  END_IF
4
```

⇨ Das Beispiel ist bereit zum Test.

15.25.3.3 Ausführung des Beispielprojektes

1. Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
2. Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten der SPS aus.
3. Dieses C++ TcCOM Modul verlangt den Windows Test-Mode. Aktivieren Sie diesen, um das Modul und damit das Sample nutzen zu können.
4. Beim ersten Aktivieren der TwinCAT-Konfiguration ist das Test-Zertifikat zu akzeptieren:

⇨ In der Onlineansicht der SPS-Applikation ist der gewünschte TcCOM-Objektname im SPS-Proxy-Baustein ersichtlich. Der Projektknoten **TcCOM-Objekte** führt das erzeugte Objekt mit der generierten ID und dem gewünschten Namen in seiner Liste. Der Schnittstellenzeiger hat einen gültigen Wert und die Methode wird ausgeführt.



16 Anhang

- Die ADS Return Codes [▶ 343] sind im gesamten TwinCAT 3 Bereich wichtig, hier insbesondere, wenn ADS-Kommunikation [▶ 200] selbst implementiert wird.
- Die Retain Daten [▶ 348] (auf NOVRAM Speicher) sind auf ähnliche Art aus der SPS und auch C++ nutzbar.
- Neben dem Konzept der TcCOM Module [▶ 39] ist das TwinCAT 3 Typsystem eine wichtige Grundlage für das Verständnis.
- Die folgenden Seiten stammen aus der Doku des Automation Interface.
  Beim Umgang mit dem Automation Interface empfiehlt es sich, die dortige Dokumentation zu beachten.
  - Erstellung von und Umgang mit C++ Projekten und Modulen [▶ 351]
  - Erstellung von und Umgang mit TcCOM Modulen [▶ 354]

16.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 343]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 344]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 345]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 347]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet, nicht erreichbar oder nicht installiert.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet. Mehrere Ursachen sind möglich. Beispielsweise beim Anlegen von Routen, dass ein falsches Passwort angegeben wurde.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert. Mehrere Ursachen sind möglich. Beispielsweise, dass eine Unidirectionale ADS Route in die umgekehrte Richtung verwendet wird.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.

Hex	Dec	HRESULT	Name	Beschreibung
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel® VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel® VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel® VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel® VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

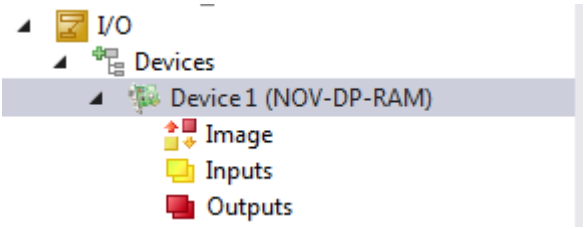
Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

16.2 Retain Daten

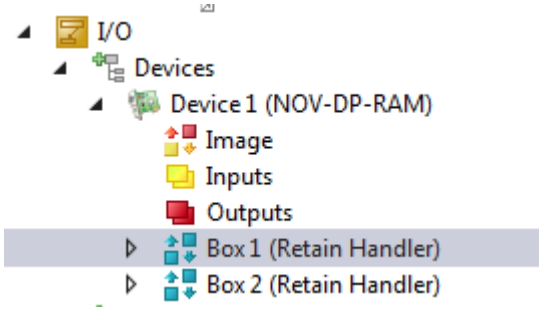
Dieser Bereich beschreibt die Möglichkeit, Daten über einen geordneten oder spontanen Neustart einer Anlage bereitzuhalten. Hierfür wird das NOV-RAM eines Gerätes verwendet. Die EL6080 kann für diese Retain Daten nicht verwendet werden, da die entsprechenden Daten erst übertragen werden müssen, welches zu entsprechenden Laufzeiten führt.
Im Folgenden wird der Umgang mit dem Retain Handler, welcher die Daten speichert und wieder bereitstellt, sowie die Verwendung aus den unterschiedlichen TwinCAT 3 Programmiersprachen beschrieben.

Konfiguration eines Retain Gerätes

1. Die Retain Daten werden dabei von einem Retain Handler, der Teil des NOV-DP-RAM-Geräts im IO Bereich der TwinCAT Solution ist, gespeichert und bereitgestellt. Legen Sie hierfür im IO Bereich der Solution ein NOV-RAM DP Device an.

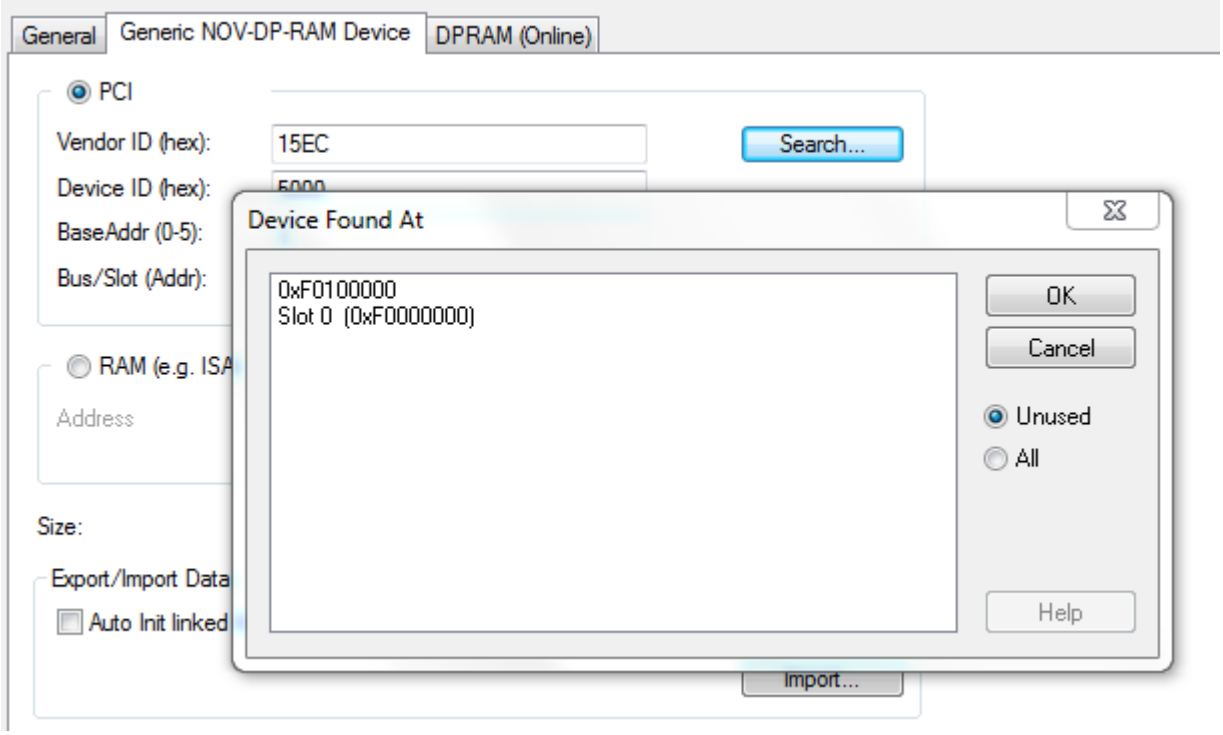


2. Legen Sie unterhalb dieses Gerätes ein oder mehrere Retain Handler an.



Speicherort: NOVRAM

3. Konfigurieren Sie das NOV-DP-RAM-Gerät. Definieren Sie im Tab **Generic NOV-DP-RAM Device** über **Search…** den zu verwendenden Bereich.



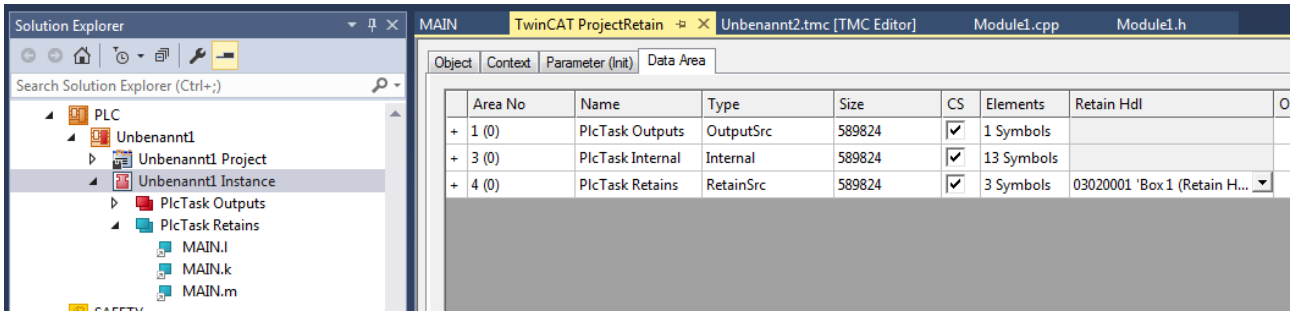
4. Für die Symbolik wird zusätzlich im Boot/-Verzeichnis von TwinCAT ein Retain/-Verzeichnis angelegt.

Nutzung des Retain Handlers mit einem PLC Projekt

In einem SPS-Projekt werden die Variablen entweder in einem VAR RETAIN Bereich angelegt, oder mit dem Attribut TcRetain versehen.

```
PROGRAM MAIN
VAR RETAIN
  l: UINT;
  k: UINT;
END_VAR
VAR
  {attribute 'TcRetain':='1'}
  m: UINT;
  x: UINT;
END_VAR
```

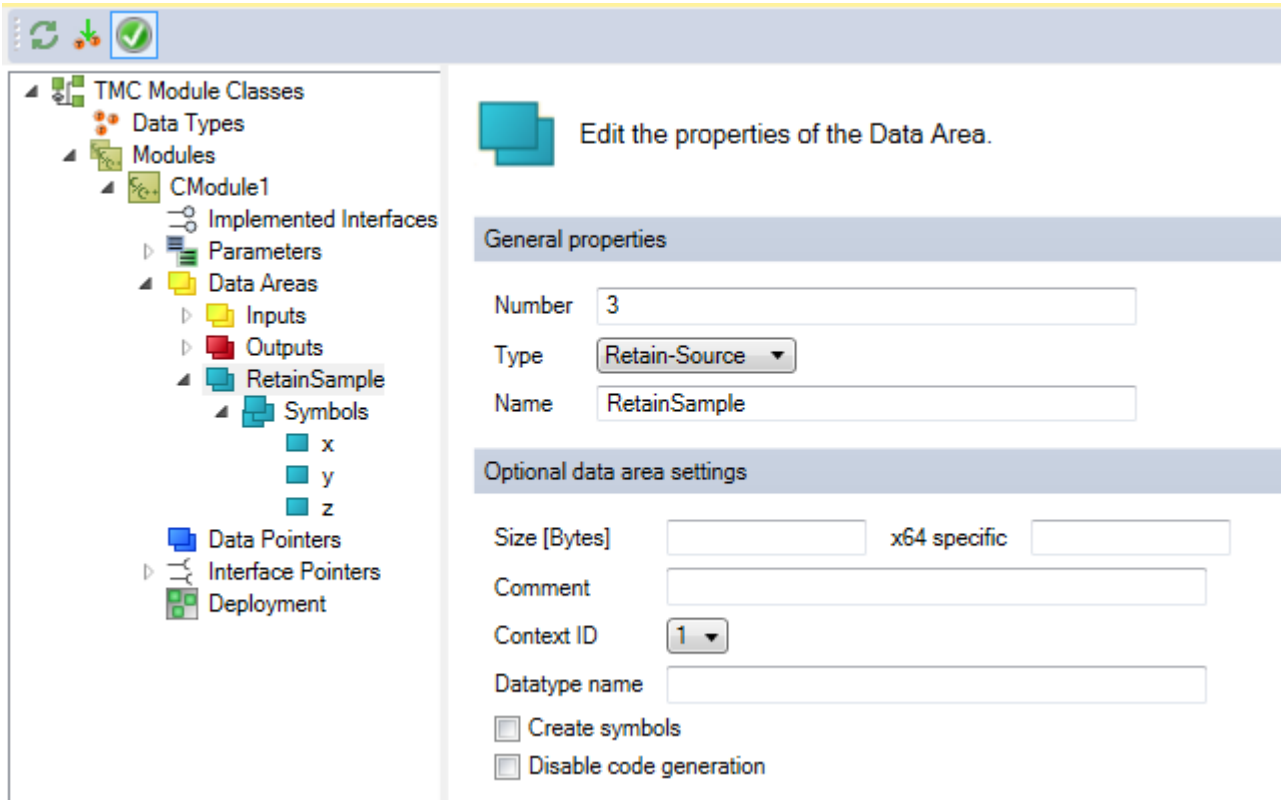
Nach einem „Build“ werden entsprechende Symbole angelegt.
Die Zuordnung zu dem Retain Handler des NOV-DP-RAM Gerätes wird in der Spalte **Retain Hdl** vorgenommen.



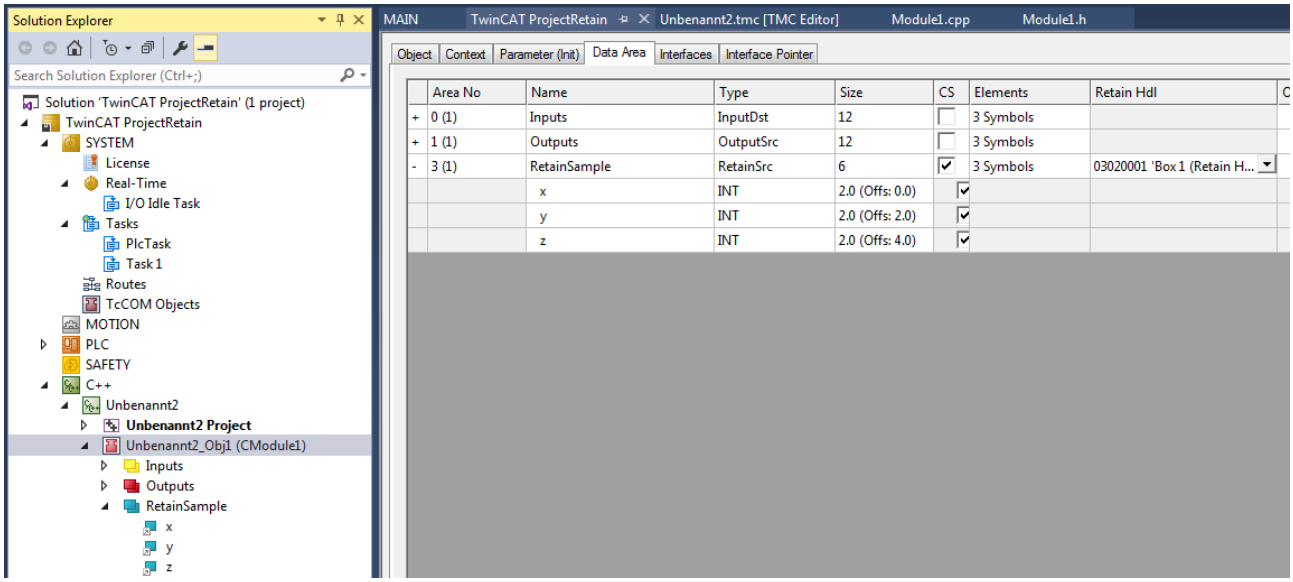
Wenn selbstangelegte Datentypen (DUTs) als Retain verwendet werden, müssen die Datentypen im TwinCAT Typsystem vorhanden sein. Hierfür kann entweder die Option **Convert to Global Type** verwendet werden oder Strukturen können direkt als `STRUCT RETAIN` angelegt werden, womit dann jedoch alle Vorkommen der Struktur über den Retain Handler behandelt werden.
Für POUs (Funktionsbausteine) als Ganzes ist die Verwendung von Retain-Daten nicht möglich. Einzelne Elemente eines POUs können hingegen verwendet werden.

Nutzung des Retain Handlers mit einem C++ Modul

In einem C++ Modul wird eine Data Area vom Typ Retain-Source angelegt, die die entsprechenden Symbole enthält.

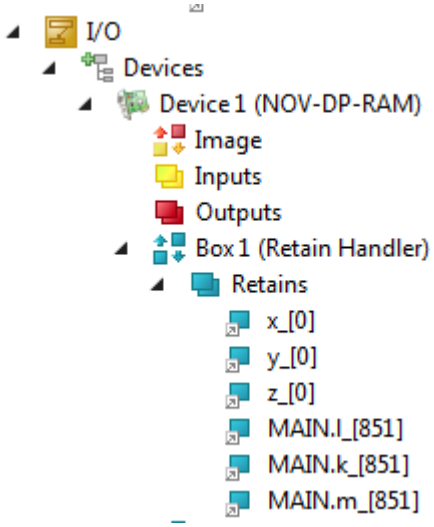


An den Instanzen des C++ Moduls wird ein zu verwendender Retain Handler des NOV-DP-RAM-Gerätes für diese Data Area in der Spalte **Retain Hdl** vorgenommen.



Fazit

Sobald ein Retain Handler in dem jeweiligen Projekt als Ziel ausgewählt wurde, wird nach einem ‚Build' automatisiert sowohl die Symbole unterhalb des Retain Handlers angelegt, wie auch ein Mapping erzeugt.



16.3 Erstellung von und Umgang mit C++ Projekten und Modulen

In diesem Kapitel wird die Erstellung von, der Zugriff auf und der Umgang mit TwinCAT C++ Projekten ausführlich erklärt. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über C++ Projekte
- Neue C++ Projekte erstellen
- Neues Modul innerhalb eines C++ Projekts erstellen
- Bestehende C++ Projekte öffnen
- Modulinstanzen erstellen
- TMC Code Generator aufrufen
- Publish Modules Befehl aufrufen
- C++ Projekteigenschaften einstellen
- Projekt aufbauen

Allgemeine Informationen über C++ Projekte

C++ Projekte werden durch ihre sogenannten Projektvorlagen spezifiziert, die vom „TwinCAT C++ Projekt-Assistenten“ verwendet werden. Innerhalb eines Projekts können verschiedene Module durch Modulvorlagen spezifiziert werden, die vom „TwinCAT Klassenassistenten“ verwendet werden.

TwinCAT-definierte Vorlagen sind im Abschnitt C++ / Assistenten [▶ 89] dokumentiert.

Der Kunde kann eigene Vorlagen definieren, was in dem entsprechenden Unterabschnitt C++ Abschnitt / Assistenten [▶ 139] dokumentiert ist.

C++ Projekte erstellen

Um ein neues C++ Projekt mit Hilfe des Automation Interface zu erstellen, müssen Sie zum C++ Knoten navigieren und dann die CreateChild()-Methode mit entsprechenden Vorlagendatei als Parameter ausführen.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (PowerShell):

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

Zur Instanziierung eines Treiber-Projekts verwenden Sie "TcVersionedDriverWizard" als `pathToTemplateFile`.

Neues Modul innerhalb eines C++ Projekts erstellen

Innerhalb eines C++ Projekts wird ein TwinCAT Modul-Assistent verwendet, damit der Assistent aus einer Vorlage ein Modul erstellt.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (PowerShell):

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

Als Beispiel zur Instanziierung eines Cyclic IO-Modulprojekts verwenden Sie "TcModuleCyclicCallerWizard" als `pathToTemplateFile`.

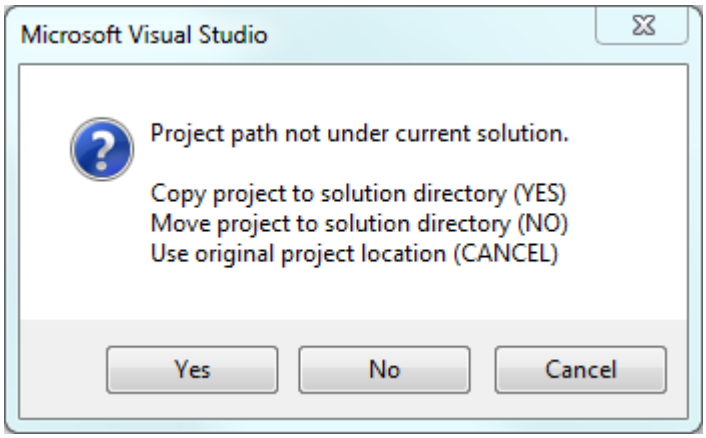
Bestehende C++ Projekte öffnen

Zum Öffnen eines bestehenden C++ Projekts mit Hilfe von Automation Interface, müssen Sie zum C++ Knoten navigieren und dann die CreateChild()-Methode mit dem Pfad der entsprechenden C++ Projektdatei als Parameter ausführen.

Sie können drei verschiedene Werte als SubType verwenden:

- 0: Projekt zum Solution-Verzeichnis kopieren
- 1: Projekt zum Solution-Verzeichnis verschieben
- 2: Verwenden Sie den Original-Projektspeicherort (spezifizieren Sie “” als NameOfProject Parameter)

Grundsätzlich repräsentieren diese Werte die Funktionalitäten (Ja, Nein, Abbrechen) von der folgenden MessageBox in TwinCAT XAE:



Anstelle der Vorlagendatei müssen Sie den Pfad zum C++ Projekt (bzw. dessen vcxproj Datei) verwenden, das hinzugefügt werden muss. Alternativ können Sie auch ein C++ Projektarchiv (tczip Datei) verwenden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (PowerShell):

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

Achten Sie darauf, dass C++ Projekte nicht umbenannt werden können, somit muss der Original-Projektname spezifiziert werden. (vergl. Umbenennen von TwinCAT-C++ Projekten [▶ 230])

Modulinstanzen erstellen

TcCOM Module können am System -> TcCOM Modulknoten erstellt werden. Siehe dort diesbezügliche Dokumentation [▶ 354].

Das gleiche Verfahren kann auch für den C++ Projektknoten gelten, um die TcCOM Instanzen an der Stelle hinzuzufügen (`$newProject` im Code oben auf dieser Seite).

TMC Code Generator aufrufen

Der TMC-Code-Generator kann aufgerufen werden, um den C++ Code nach den Änderungen der TMC-Datei oder dem C++ Projekt zu erstellen.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (PowerShell):

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

Publish Modules Befehl aufrufen

Die Veröffentlichung umfasst den Aufbau des Projekts für alle Plattformen. Das kompilierte Modul wird für den Export bereitgestellt, wie dies im Abschnitt Modulumgang bei C++ beschrieben ist.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (PowerShell):

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

C++ Projekteigenschaften einstellen

C++ Projekte bieten verschiedene Optionen für den Aufbau- und Bereitstellungsprozess.
Diese sind über das Automation Interface einstellbar.

Code-Ausschnitt (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);
```

Code-Ausschnitt (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)
```

Für die BootProjectEncryption sind die Werte „None“ und „Target“ gültig.
Beide anderen Einstellungen sind „false“ und „true“ Werte.

Projekt aufbauen

Zum Aufbau des Projekts oder der Solution können Sie die entsprechenden Klassen und Methoden der Visual Studio API verwenden.

16.4 Erstellung von und Umgang mit TcCOM Modulen

In diesem Kapitel wird beschrieben, wie bestehende TcCOM-Module einer bestehenden TwinCAT-Konfiguration hinzugefügt werden und diese parametrisiert werden. Die folgenden Themen werden in diesem Kapitel kurz behandelt:

- Erwerb einer Referenz zu “TcCOM Objects” Knoten
- TcCOM-Module hinzufügen
- Durch hinzugefügte TcCOM-Module iterieren
- CreateSymbol Flag für Parameter einstellen
- CreateSymbol Flag für Datenbereiche einstellen
- Kontext (Aufgaben) einstellen
- Variablen verknüpfen

Erwerb einer Referenz zu “TcCOM Objects” Knoten

In einer TwinCAT-Konfiguration befindet sich der “TcCOM Objects” Knoten unter “SYSTEM^TcCOM Objects”. Daher können Sie eine Referenz zu diesem Knoten erfassen, indem Sie die Methode ITcSysManager::LookupTreeItem() auf folgende Weise verwenden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

The code above assumes that there is already a systemManager objects present in your AI code.

Adding existing TcCOM modules

Der vorstehende Code geht davon aus, dass bereits ein SystemManager Objekt in Ihrem AI-Code vorhanden ist.

TcCOM-Module hinzufügen

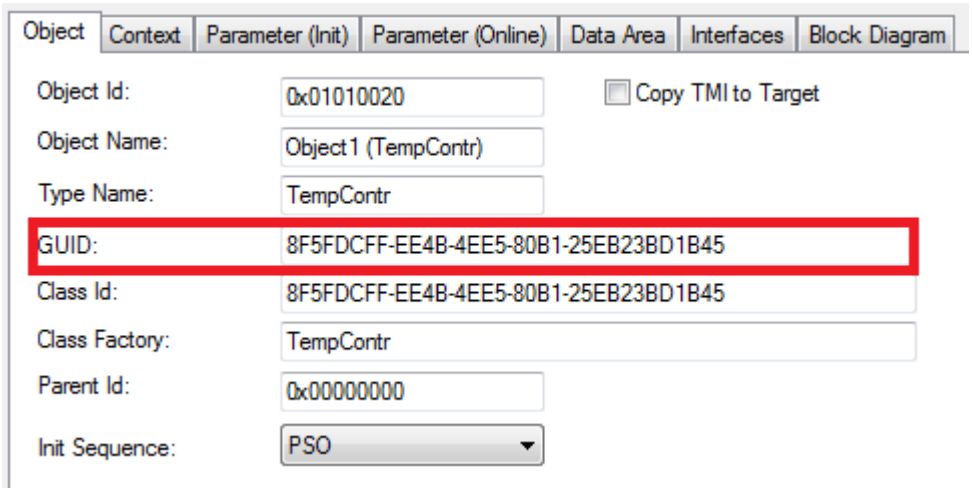
Um bestehende TcCOM-Module zu Ihrer TwinCAT-Konfiguration hinzufügen, müssen diese Module für TwinCAT erkennbar sein. Dies kann durch eine der folgenden Vorgehensweisen erreicht werden:

- TcCOM-Module zum Ordner %TWINCAT3.XDIR"\CustomConfig\Modules\ kopieren
- %TWINCAT3.XDIR"\Config\Io\TcModuleFolders.xml bearbeiten, um einen Pfad zu einem Ordner Ihrer Wahl hinzuzufügen und die Module innerhalb dieses Ordners zu platzieren

Both ways will be sufficient to make the TcCOM modules detectable by TwinCAT.

A TcCOM module is being identified by its GUID or name:

- This GUID can be used to add a TcCOM module to a TwinCAT configuration via the ITcSmTreeItem::CreateChild() method. The GUID can be determined in TwinCAT XAE via the properties page of a TcCOM module.



Alternativ können Sie die GUID über die TMC-Datei des TcCOM-Moduls bestimmen.

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

Angenommen, wir verfügen bereits über ein TcCOM-Modul, das in TwinCAT registriert ist und für TwinCAT erkennbar ist. Wir möchten nun dieses TcCOM-Modul, das den GUID {8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45} hat, zur TwinCAT-Konfiguration hinzufügen. Dies kann wie folgt geschehen:

Code-Ausschnitt (C#):

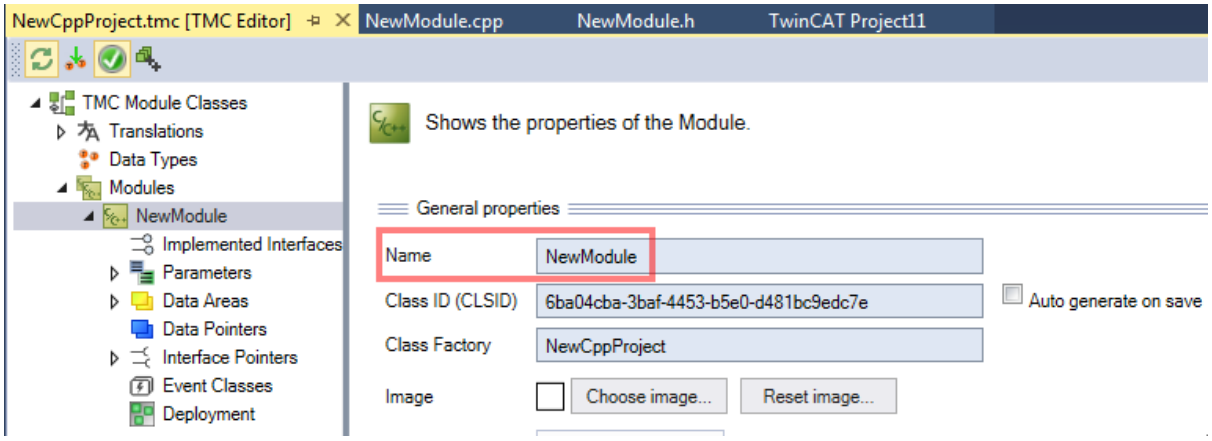
```
Dictionary<string,Guid> tcomModuleTable = new Dictionary<string,Guid>();
tcomModuleTable.Add("TempContr",Guid.Parse("{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}"));
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 0, "",
tcomModuleTable["TempContr"]);
```

Code-Ausschnitt (Powershell):

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

Please note that the vInfo parameter of the method ItcSmTreeItem::CreateChild() contains the GUID of the TcCOM module which is used to identify the module in the list of all registered TcCOM modules in that system.

- This name can be used to add a TcCOM module to a TwinCAT configuration via the ITcSmTreeItem::CreateChild() method. The name can be determined in TwinCAT XAE via the TMC Editor.



- This can be done by the following way:

Code Snippet (C#):

```
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 1, "", "NewModule");
```

Code Snippet (Powershell):

```
$tempController = $tcComObjects.CreateChild("Test", 0, "", "NewModule")
```

Durch hinzugefügte TcCOM-Module iterieren

Zur Iteration durch alle hinzugefügte TcCOM-Modulinstanzen können Sie die ITcModuleManager2 Schnittstelle verwenden. Der folgende Code-Ausschnitt zeigt, wie Sie diese Schnittstelle verwenden.

Code-Ausschnitt (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;
}
```

Code-Ausschnitt (Powershell):

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

Denken Sie daran, dass jedes Modulobjekt ebenfalls als ein ITcSmTreeItem interpretiert werden kann, daher wäre die folgende Typumwandlung gültig:

Code-Ausschnitt (C#):

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

Bitte beachten: Powershell verwendet standardmäßig dynamische Datentypen.

CreateSymbol Flag für Parameter einstellen

Das CreateSymbol (CS) Flag für Parameter eines TcCOM-Moduls kann über ihre XML-Beschreibung eingestellt werden. Der folgende Code-Ausschnitt zeigt, wie das CS Flag für den Parameter „CallBy“ aktiviert wird.

Code-Ausschnitt (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);
```

Code-Ausschnitt (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)
```

CreateSymbol Flag für Datenbereiche einstellen

Das CreateSymbol (CS) Flag für Datenbereiche eines TcCOM-Moduls kann über ihre XML-Beschreibung eingestellt werden. Der folgende Code-Ausschnitt zeigt, wie das CS Flag für den Datenbereich „Input“ aktiviert wird. Es sei darauf hingewiesen, dass dieses Verfahren dem für Parameter sehr ähnlich ist.

Code-Ausschnitt (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);
```

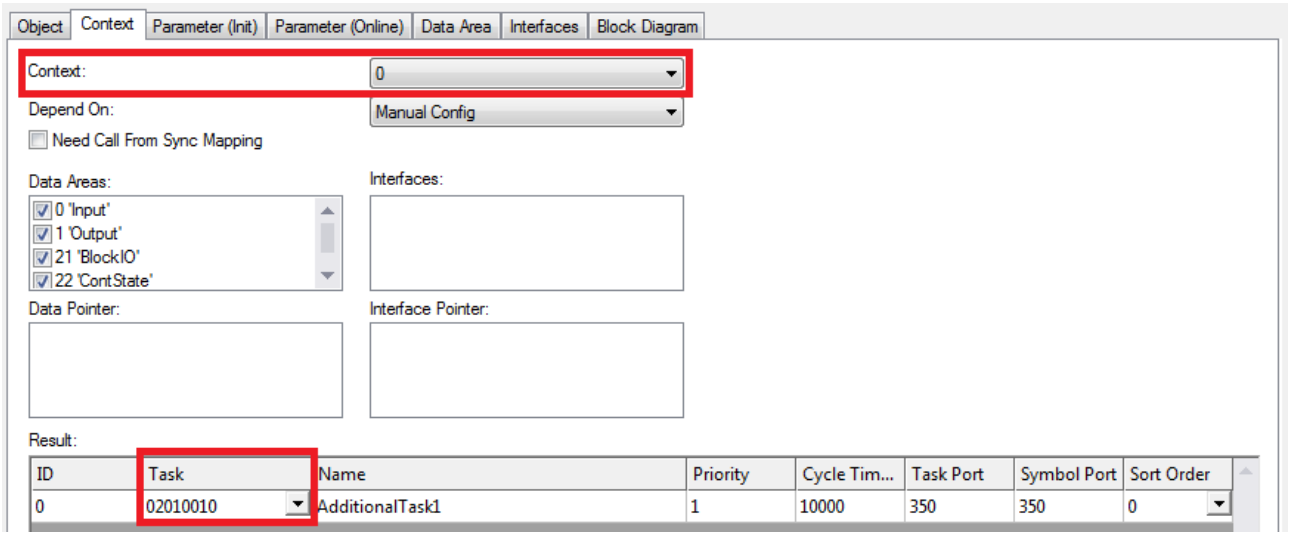
```
XmlNode sourceDataAreas = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
DataAreas");

// 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);
```

Code-Ausschnitt (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)
```

Kontext (Aufgaben) einstellen

Jede TcCOM-Modulinstanz muss in einem spezifischen Kontext (Aufgabe) laufen. Dies kann über die ITcModuleInstance2::SetModuleContext() Methode erfolgen. Diese Methode erwartet zwei Parameter: ContextId und TaskObjectId. Beide sind äquivalent zu den entsprechenden Parametern in TwinCAT XAE:



Denken Sie daran, dass die TaskObjectId in TwinCAT XAE hexadezimal wiedergegeben wird.

Code-Ausschnitt (C#):

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

Sie können die TaskObjectId über die XML-Beschreibung der entsprechenden Aufgabe bestimmen, z. B.:

Code-Ausschnitt (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);
```

Variablen verknüpfen

Die Verknüpfung von Variablen einer TcCOM-Modulinstanz zu SPS/IO oder anderen TcCOM-Modulen kann vorgenommen werden, indem die regulären Mechanismen des Automation Interface verwendet werden, z. B. ITcSysManager::LinkVariables().

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

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Mehr Informationen:
www.beckhoff.de/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com