



**Handbuch**

## **TC3 C++**

**TwinCAT 3**

**Version:** 1.5  
**Datum:** 22.06.2017  
**Bestell-Nr.:** TC1300

**BECKHOFF**



# Inhaltsverzeichnis

<b>1 Vorwort .....</b>	<b>7</b>
1.1 Hinweise zur Dokumentation .....	7
1.2 Sicherheitshinweise .....	8
<b>2 Übersicht .....</b>	<b>9</b>
<b>3 Einleitung .....</b>	<b>10</b>
3.1 Von der klassischen Usermode-Programmierung zur Echtzeit-Programmierung in TwinCAT.....	12
<b>4 Anforderungen.....</b>	<b>18</b>
<b>5 Vorbereitung - nur einmal!.....</b>	<b>19</b>
5.1 „Microsoft Windows Driver Kit (WDK)“ Installation .....	19
5.2 Visual Studio - TwinCAT XAE Base-Symbolleiste.....	21
5.3 Visual Studio vorbereiten - Konfigurations- und Plattformsymbolleiste .....	22
5.4 x64: Treibersignierung .....	22
5.4.1 Treiber signieren .....	23
5.4.2 Testsignierung.....	23
5.4.3 Testzertifikat löschen .....	25
5.4.4 Kundenzertifikate .....	27
<b>6 Module .....</b>	<b>29</b>
6.1 Das TwinCAT Component Object Model (TcCOM) Konzept.....	29
6.1.1 TwinCAT-Modul Eigenschaften .....	31
6.1.2 TwinCAT-Modul Zustandsmaschine .....	38
6.2 Modul zu Modul Kommunikation.....	40
<b>7 Module - Handhabung .....</b>	<b>43</b>
7.1 Module exportieren .....	43
7.2 Module importieren .....	44
<b>8 TwinCAT C++ Entwicklung .....</b>	<b>47</b>
<b>9 Schnellstart .....</b>	<b>49</b>
9.1 TwinCAT 3 Projekt erstellen .....	49
9.2 TwinCAT 3 C++ Projekt erstellen .....	50
9.3 TwinCAT 3 C++ Projekt implementieren .....	54
9.4 TwinCAT 3 C++ Projekt kompilieren / bauen.....	55
9.5 TwinCAT 3 C++ Modulinstanz erstellen .....	56
9.6 TwinCAT Task erstellen und auf Modulinstanz anwenden.....	58
9.7 TwinCAT 3 C++ Debugger aktivieren .....	60
9.8 TwinCAT 3 Projekt aktivieren .....	61
9.9 TwinCAT 3 C++ Projekt debuggen .....	62
<b>10 Debuggen .....</b>	<b>67</b>
10.1 Einzelheiten zu den bedingten Haltepunkten .....	70
10.2 Visual Studio Werkzeuge.....	72
<b>11 Assistenten .....</b>	<b>75</b>
11.1 TwinCAT C++ Projekt-Assistent .....	75
11.2 TwinCAT Module Klassenassistent .....	76
11.3 TwinCAT Module Class Editor (TMC).....	79
11.3.1 Übersicht.....	81

11.3.2	Grundlegende Informationen .....	82
11.3.3	Datentypen.....	82
11.3.4	Module .....	100
11.4	TwinCAT Module Instance Configurator.....	123
11.4.1	Objekt.....	124
11.4.2	Kontext.....	125
11.4.3	Parameter (Init).....	125
11.4.4	Data Area.....	126
11.4.5	Schnittstellen.....	126
11.4.6	Schnittstellenzeiger.....	126
11.4.7	Datenzeiger.....	127
11.5	Kunden-spezifische Projekt-Templates .....	127
11.5.1	Übersicht.....	127
11.5.2	Beteiligte Dateien .....	128
11.5.3	Transformationen .....	129
11.5.4	Hinweise zum Handling .....	130
<b>12</b>	<b>Programmierreferenz .....</b>	<b>133</b>
12.1	Dateibeschreibung.....	134
12.1.1	Ablauf der Kompilierung.....	136
12.2	Limitierungen .....	136
12.3	Speicherallokierung .....	137
12.4	Schnittstellen .....	138
12.4.1	Schnittstelle ITcCyclic .....	139
12.4.2	Schnittstelle ITcCyclicCaller.....	140
12.4.3	Schnittstelle ITc FileAccess .....	142
12.4.4	Schnittstelle ITc FileAccessAsync .....	151
12.4.5	Schnittstelle ITcIoCyclic .....	152
12.4.6	Schnittstelle ITcIoCyclicCaller.....	154
12.4.7	Schnittstelle ITComObject.....	156
12.4.8	Schnittstelle ITComObject (C++ Convenience) .....	161
12.4.9	Schnittstelle ITcPostCyclic .....	162
12.4.10	Schnittstelle ITcPostCyclicCaller.....	163
12.4.11	Schnittstelle ITcRTTimeTask .....	165
12.4.12	Schnittstelle ITcTask .....	166
12.4.13	Schnittstelle ITcTaskNotification .....	170
12.4.14	Schnittstelle ITcUnknown .....	171
12.5	Runtime Library (RtlR0.h) .....	173
12.6	ADS-Kommunikation .....	175
12.6.1	AdsReadDeviceInfo .....	175
12.6.2	AdsRead .....	177
12.6.3	AdsWrite.....	179
12.6.4	AdsReadWrite .....	181
12.6.5	AdsReadState .....	184
12.6.6	AdsWriteControl .....	185
12.6.7	AdsAddDeviceNotification .....	187
12.6.8	AdsDelDeviceNotification .....	189
12.6.9	AdsDeviceNotification .....	191
12.7	Mathematische Funktionen.....	193
12.8	Zeitfunktionen .....	195
12.9	STL / Container.....	195
12.10	Fehlermeldungen - Verständnis .....	196
12.11	Modul-Nachrichten zum Engineering (Logging / Tracing) .....	196

<b>13 How to...?</b>	<b>200</b>
13.1 Verwendung des Automation Interface.....	200
13.2 Windows 10 als Zielsystem .....	200
13.3 Module veröffentlichen.....	200
13.4 Module veröffentlichen auf der Kommandozeile.....	201
13.5 Clone .....	201
13.6 Umbenennen von TwinCAT-C++ Projekten .....	202
13.7 Zugriff auf Variablen über ADS.....	204
13.8 TcCallAfterOutputUpdate für C++ Module.....	204
13.9 Reihenfolgebestimmung der Ausführung in einem Task .....	204
13.10 Stack Size > 4kB verwenden .....	205
13.11 Setzen von Version/Herstellerinformationen .....	206
13.12 Modul löschen.....	207
13.13 Initialisierung von TMC-Membervariablen .....	208
13.14 SPS-Zeichenketten als Methodenparameter verwenden .....	208
13.15 Bibliotheken von Drittanbietern.....	209
13.16 Verknüpfungen mittels TMC Editor (TcLinkTo).....	210
<b>14 Fehlersuche</b>	<b>212</b>
14.1 Build - „Cannot open include file ntddk.h“ .....	212
14.2 Build - „The target ... does not exist in the project“ .....	212
14.3 Debug - „Unable to attach“ .....	213
14.4 Activation – „invalid object id“ (1821/0x71d) .....	214
14.5 Fehlermeldung – VS2010 und LNK1123/COFF .....	214
14.6 Verwendung von C++ Klassen in TwinCAT C++ Modulen .....	214
14.7 Verwendung von afxres.h .....	215
<b>15 C++-Beispiele</b>	<b>216</b>
15.1 Übersicht.....	216
15.2 Beispiel01: Zyklisches Modul mit IO .....	219
15.3 Beispiel02: Zyklische C++ Logik, die IO vom IO Task verwendet .....	219
15.4 Beispiel03: C++ als ADS Server.....	220
15.4.1 Beispiel03: Der in C++ geschriebene TC3 ADS Server.....	221
15.4.2 Beispiel03: ADS Client UI in C# .....	225
15.5 Beispiel05: C++ CoE Zugriff über ADS.....	229
15.6 Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch .....	230
15.7 Beispiel07: Empfang von ADS Notifications .....	235
15.8 Beispiel08: Anbieten von ADS-RPC .....	236
15.9 Beispiel10: Modulkommunikation: Verwendung von Datenzeigern.....	239
15.10 Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf .....	240
15.10.1 Methoden zur Verfügung stellendes TwinCAT 3 C++ Modul.....	241
15.10.2 SPS um Methoden aufzurufen, die von einem anderen Modul angeboten werden.....	255
15.11 Beispiel11a: Modulkommunikation: C-Modul ruft eine Methode eines anderem C-Moduls auf ..	267
15.12 Beispiel12: Modulkommunikation: Verwendet IO Mapping .....	268
15.13 Beispiel13: Modulkommunikation: C-Modul ruft SPS-Methoden auf .....	269
15.14 Beispiel19: Synchroner Dateizugriff.....	272
15.15 Beispiel20: FileIO-Write .....	273
15.16 Beispiel20a: FileIO-Cyclic Read / Write .....	273

15.17	Beispiel22: Automation Device Driver (ADD): Zugang DPRAM .....	275
15.18	Beispiel23: Strukturierte Ausnahmebehandlung (SEH).....	276
15.19	Beispiel25: Statische Bibliothek.....	278
15.20	Beispiel26: Ausführungsreihenfolge in einem Task.....	279
15.21	Beispiel30: Zeitmessung.....	281
15.22	Beispiel31: Funktionsbaustein TON in TwinCAT3 C++ .....	282
15.23	Beispiel35: Ethernet Zugriff .....	283
15.24	Beispiel37: Daten archivieren .....	284
15.25	TcCOM Beispiele.....	285
15.25.1	TcCOM_Sample01_PlctoPlc .....	285
15.25.2	TcCOM_Sample02_PlctoCpp.....	295
15.25.3	TcCOM_Sample03_PlccreatesCpp .....	300
<b>16</b>	<b>Anhang .....</b>	<b>305</b>
16.1	ADS Return Codes .....	305
16.2	Retain Daten.....	308
16.3	Typsystem .....	311
16.3.1	Projektbasiertes Typsystem .....	311
16.3.2	Arten von Datentypen .....	312
16.3.3	Handhabung von Datentypen .....	313
16.3.4	Verwaltung und Identifizierung von Datentypen .....	314
16.3.5	Alignment von Datentypen .....	316
16.3.6	Dateien im Zusammenhang mit dem Typsystem.....	317
16.4	Erstellung von und Umgang mit C++ Projekten und Modulen.....	317
16.5	Erstellung von und Umgang mit TcCOM Modulen .....	321

# 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, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentlichte 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 weiter entwickelt.

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®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® 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 Bezeichnungen führen.

### Patente

Die EtherCAT Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:

EP1590927, EP1789857, DE102004044764, DE102007017835

mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

Die TwinCAT Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:

EP0851348, US6167425 mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.



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

### Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.

Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

## 1.2 Sicherheitshinweise

### Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!

Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

### 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.

### Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

 <b>GEFAHR</b>	<b>Akute Verletzungsgefahr!</b> Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!
 <b>WARNUNG</b>	<b>Verletzungsgefahr!</b> Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!
 <b>VORSICHT</b>	<b>Schädigung von Personen!</b> Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!
 <b>Achtung</b>	<b>Schädigung von Umwelt oder Geräten</b> Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.
 <b>Hinweis</b>	<b>Tipp oder Fingerzeig</b> Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

## 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 \[▶ 18\]](#) und [Vorbereitung \[▶ 19\]](#). Limitierungen sind [hier \[▶ 136\]](#) dokumentiert.
- **Schnellstart [▶ 49]**  
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 [▶ 31]**  
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 [▶ 75]**  
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 [▶ 133]**  
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 ...? [▶ 200]“** Kapitel enthält nützliche Tipps für das Arbeiten mit TwinCAT C++.
- **Beispiele [▶ 216]**  
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, complexes 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.

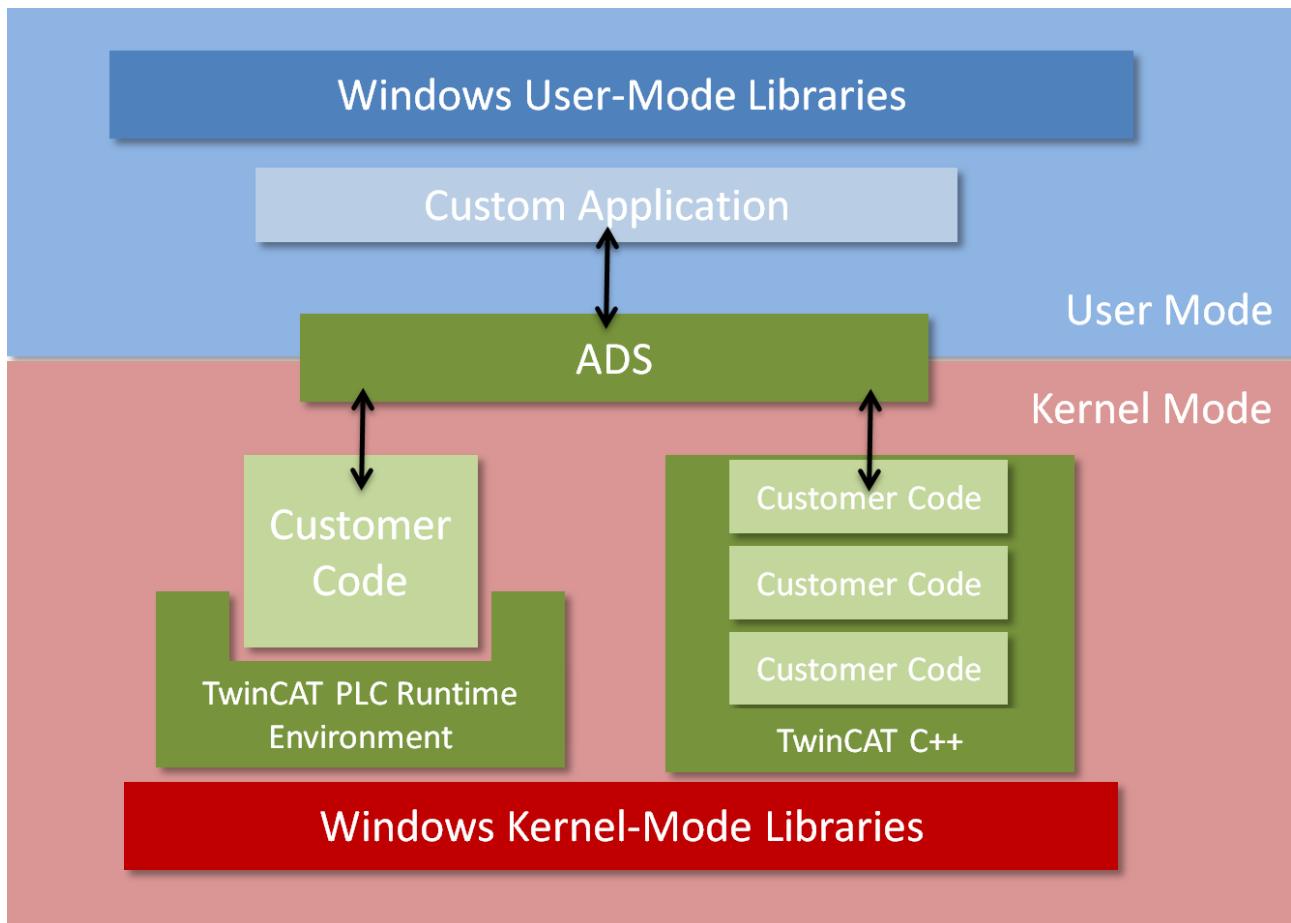
Durch die Definition dieser Schnittstellen und die Standardisierung der entsprechenden Parameter und Prozessdaten entsteht ein signifikantes Profil. Da die einzelnen Module in eine Laufzeit implementiert sind, sind auch direkte Aufrufe anderer Module möglich - wiederum über entsprechende Standard-Schnittstellen.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

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 instanziert, die lediglich so dimensioniert sein muss, dass der Bedarf an Ressourcen für alle instanzierten 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.

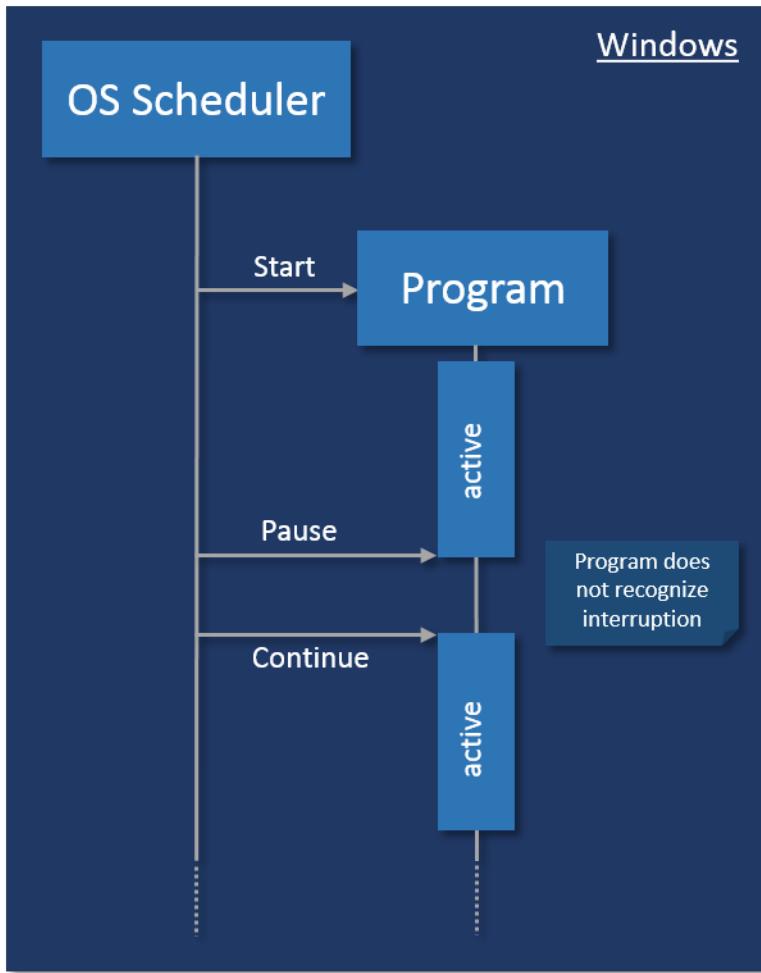


Im Gegensatz zur SPS, wo das benutzerdefinierte Programm in einer Laufzeitumgebung ausgeführt wird, sind TwinCAT C++ Module nicht in so einer gehosteten Umgebung. Dadurch werden TwinCAT C++ Module als Kernel-Module (.sys) ausgeführt, sie werden also demzufolge aus Kernel-Modus-Bibliotheken erstellt.

### 3.1 Von der klassischen Usermode-Programmierung zur Echtzeit-Programmierung in TwinCAT

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.

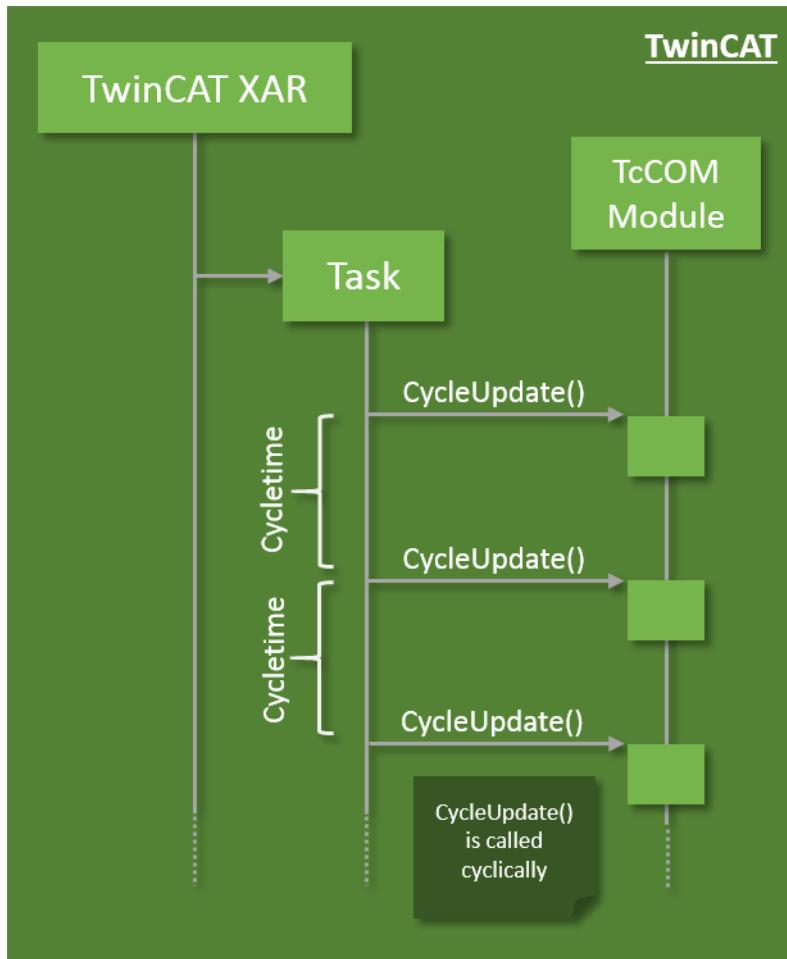
Dieser Artikel bezieht sich dabei insbesondere auf die TwinCAT-Echtzeit-Programmierung mit TwinCAT C++, da gerade hier auf der einen Seite wesentliche Vorkenntnisse der C++ Programmierung eingebracht werden können, auf der anderen Seite die Ablaufeigenschaften des TwinCAT Echtzeitsystems Berücksichtigung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



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

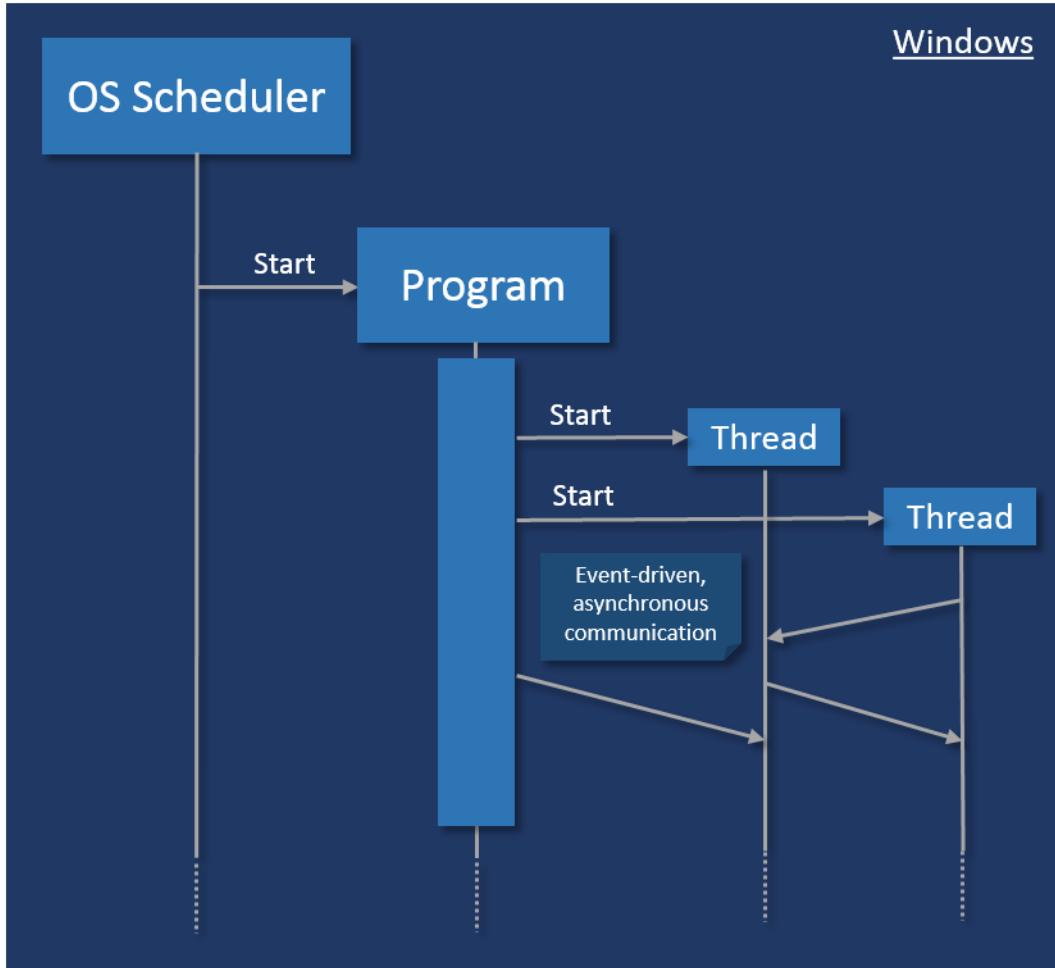
Das Programm wird hierbei vom Betriebssystem gestartet und kann selbstständig ablaufen, d.h. auch es hat die volle Kontrolle auf seinen eigenen Ablauf wie Threading und Speicherverwaltung. Um Multitasking zu ermöglichen unterbricht das Betriebssystem ein solches Programm zu einem beliebigen Zeitpunkt 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 selber muss sich auf die bereitstehenden Ressourcen verlassen können um Echtzeiteigenschaften (Antwortgarantien) zusichern zu können.



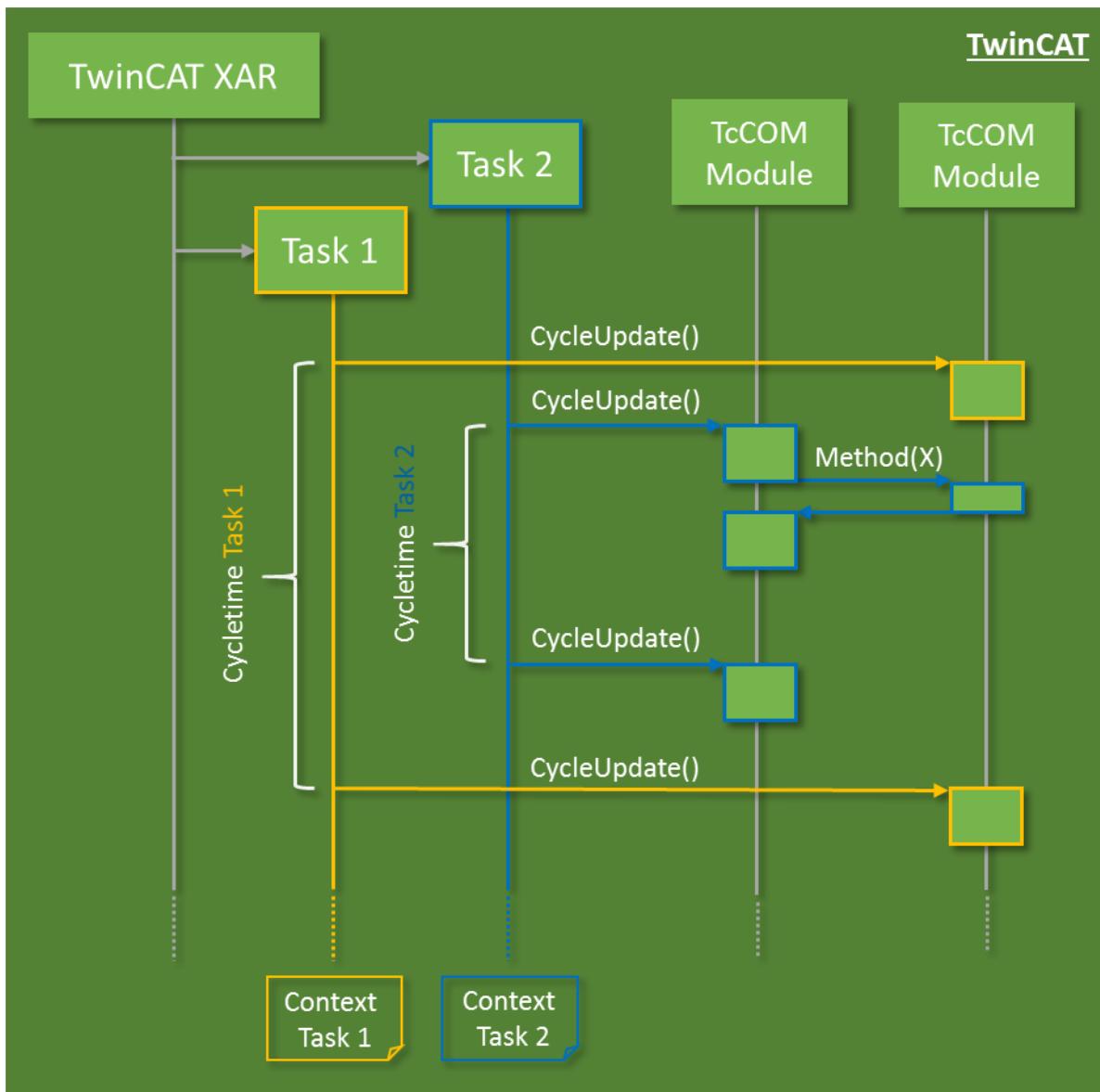
Für TwinCAT C++ wird deswegen der Grundgedanke aus der PLC aufgenommen: Das TwinCAT Echtzeitsystem verwaltet die Echtzeit Tasks, übernimmt das Scheduling und ruft einen Einstiegspunkt in dem Programmcode zyklisch auf. Das Programm muss innerhalb der bereitstehenden Zykluslänge abgearbeitet sein und die 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 selber 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 ein höher priorer Task einen niedriger prioren 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 „Windows Kernel Kontext“ und „TwinCAT Echtzeit-Kontext“ und nicht als UserMode-Anwendung ausgeführt.

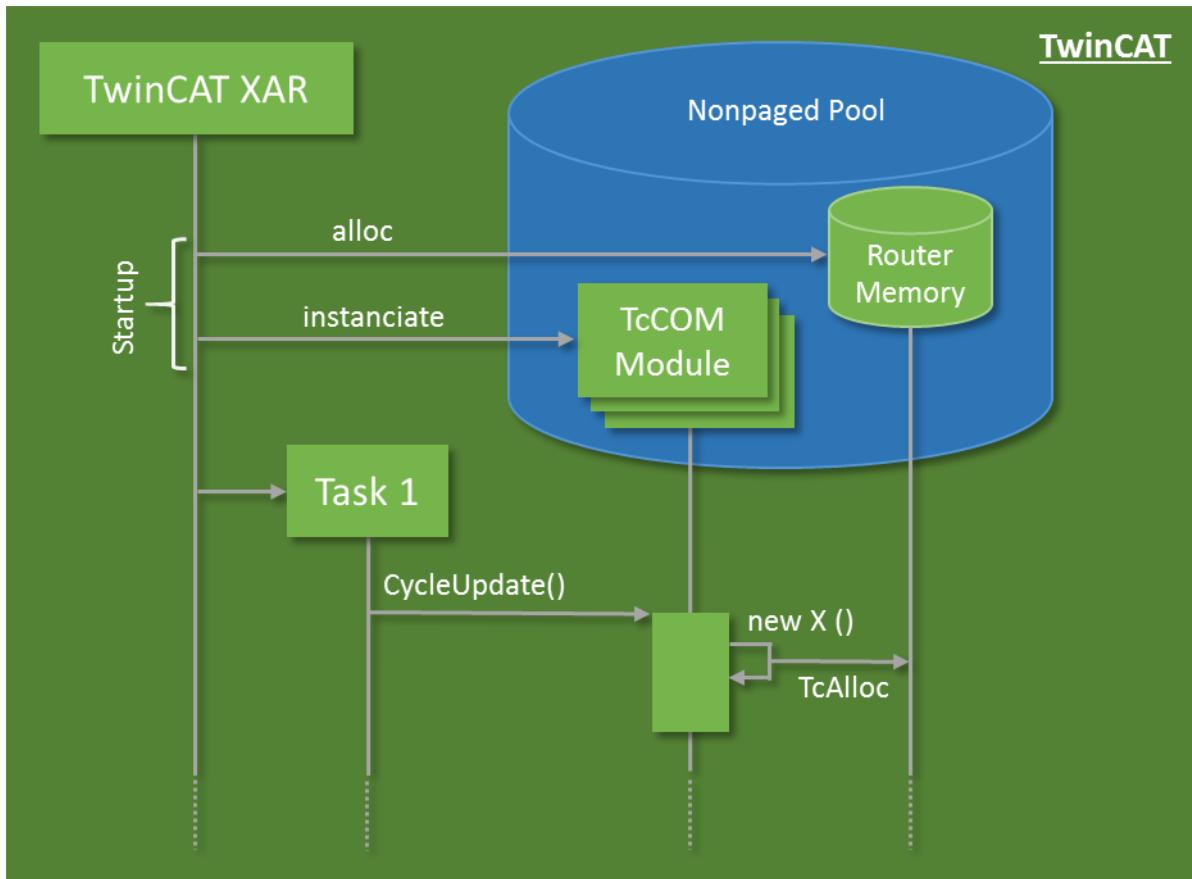
Beim Hoch-/Runterfahren der Module wird Code zur (De)Initialisierung zunächst im Windows Kernel Kontext ausgeführt; erst die letzte Phase sowie die zyklischen Aufrufe werden im TwinCAT Echtzeit-Kontext ausgeführt.

Details sind im Kapitel [Modul-State machine \[▶ 38\]](#) 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 instanziert.

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.

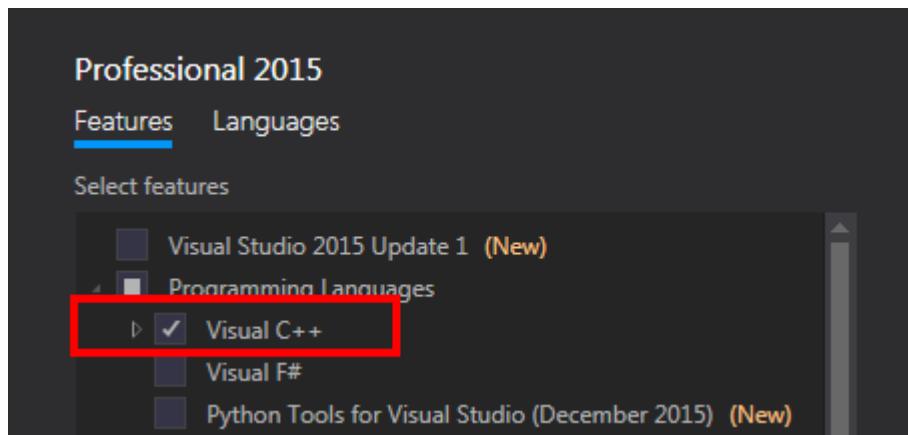
## 4 Anforderungen

### Übersicht der Mindestanforderungen

Die Implementierung und das Debuggen von TwinCAT 3 C++ Modulen erfordert:

#### Auf dem Engineering PC muss folgendes installiert sein:

- Microsoft Visual Studio 2010 (mit Service Pack 1), 2012, 2013 oder 2015 Professional, Premium oder Ultimate
  - **HINWEIS!** Unter Windows 7 bitte VS2010 / 2012 / 2013 mit Rechtsklick auf „Run As Admin...“ installieren (Selbst dann, wenn Sie der Admin auf der Maschine sind!)
  - **HINWEIS!** Bei der Installation von Visual Studio 2015 muss manuell während der Installation die Option zur Visual C++ Entwicklung selektiert werden, da diese Option bei der automatischen Installation nicht selektiert ist:



- Microsoft „Windows Driver Kit“ (WDK)  
Wenn Sie C++ Module implementieren und debuggen wollen, muss Microsoft „Windows Driver Kit“ (WDK) installiert sein: „[Windows Driver Kit](#)“ (WDK) installieren [▶ 19]
- TwinCAT 3 Installation (XAE Engineering)

#### Auf dem Laufzeit-PC:

- IPC oder Embedded CX PC mit Microsoft Betriebssystem (Windows XP oder Windows 7 oder höher)
- Microsoft Visual Studio muss **nicht** installiert werden
- Microsoft „Windows Driver Kit“ (WDK) muss **nicht** installiert werden.  
(Für die Integration und Nutzung bestehender, binärer C++ Module in eine TwinCAT 3 SPS-Umgebung ist keine zusätzliche Installation erforderlich.)
- TwinCAT 3 Installation (XAR Laufzeit)

#### Limitierungen auf dem Laufzeit-PC

- TwinCAT 3.0 unterstützt als Zielplattform (Laufzeit-PC) nur 32-Bit-Betriebssysteme.  
Auf x64 PCs kann TC3.0 als Engineering-Plattform verwendet werden: das Programm kann über Netzwerk auf einen 32bit (x86) Remote-PC übertragen und dort ausgeführt werden.
- TC3.1 unterstützt auch x64-Bit-Betriebssysteme als Zielplattform (Laufzeit-PC). Hierfür müssen die Treiber signiert werden, wie [hier](#) [▶ 22] dokumentiert, was für einen Produktivbetrieb ein Zertifikat voraussetzt [▶ 27].
- Die Ziellaufzeit muss auf „Windows NT Kernel“ basieren, wie Windows XP, Windows 7 oder die embedded Versionen wie Windows XP Embedded, Windows Embedded Standard 2009, Windows Embedded Standard 7

## 5 Vorbereitung - nur einmal!

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

- Microsoft Windows Driver Kit (WDK) [▶ 19] muss installiert werden und
- TwinCAT Basis [▶ 21] sowie Konfigurations- und Plattform [▶ 22]-Symbolleiste müssen konfiguriert werden
- Auf x64 PCs müssen Module signiert sein, damit sie ausgeführt werden können.  
Siehe Dokumentation zum Setup einer Testsignierung [▶ 22].

### 5.1 „Microsoft Windows Driver Kit (WDK)“ Installation

#### Übersicht

Die Implementierung von TwinCAT 3 C++ Modulen erfordert Teile des „Windows Driver Kit“ (WDK).

**⚠ HINWEIS! Die Installation ist nur für die TwinCAT 3 Engineering Umgebung erforderlich, um C++ Module erstellen und bearbeiten zu können. Für die Zielplattform der TwinCAT 3 Runtime wird sie nicht benötigt.**

1. „Windows Driver Kit 7.1“ aus dem Microsoft Download Center herunterladen <http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=36a2630f-5d56-43b5-b996-7633f2ec14ff>



#### Windows Driver Kit Version 7.1.0

Language: English

Download

The Windows Driver Kit (WDK) Version 7.1.0 is an update to the WDK 7.0.0 release and contains the tools, code samples, documentation, compilers, headers and libraries with which software developers create drivers for Windows 7, Windows Vista, Windows XP, Windows Server 2008 R2, Windows Server 2008, and Windows Server 2003.

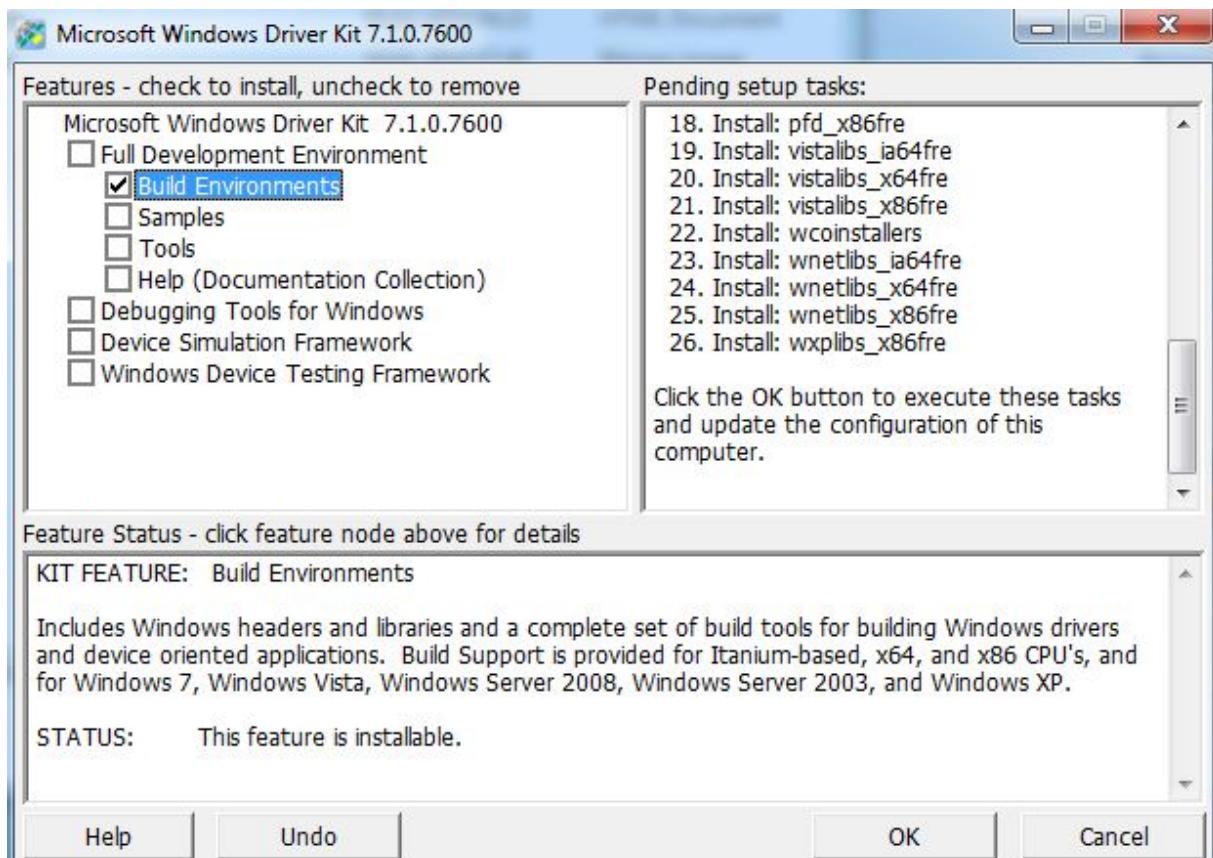
(+) Details

(+) System Requirements

(+) Install Instructions

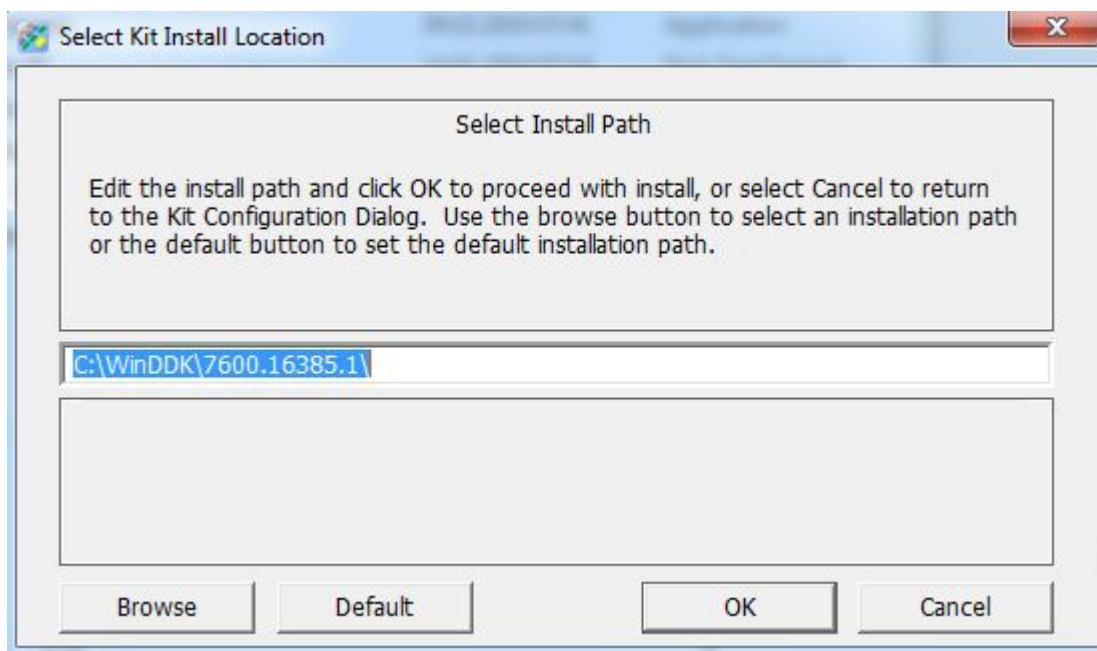
2. Brennen Sie sich nach dem Download entweder eine CD des heruntergeladenen ISO-Abbilds oder verwenden Sie ein virtuelles (softwarebasiertes) CD-Laufwerk.
3. „KitSetup.exe“ auf CD / heruntergeladenem ISO-Abbildung starten (auf Windows7 PCs Installation mit „Run As Administrator...“ starten)

4. Option „Build Environment“ wählen - alle anderen Komponenten werden von TwinCAT 3 nicht benötigt - und „OK“ klicken um fortzufahren.



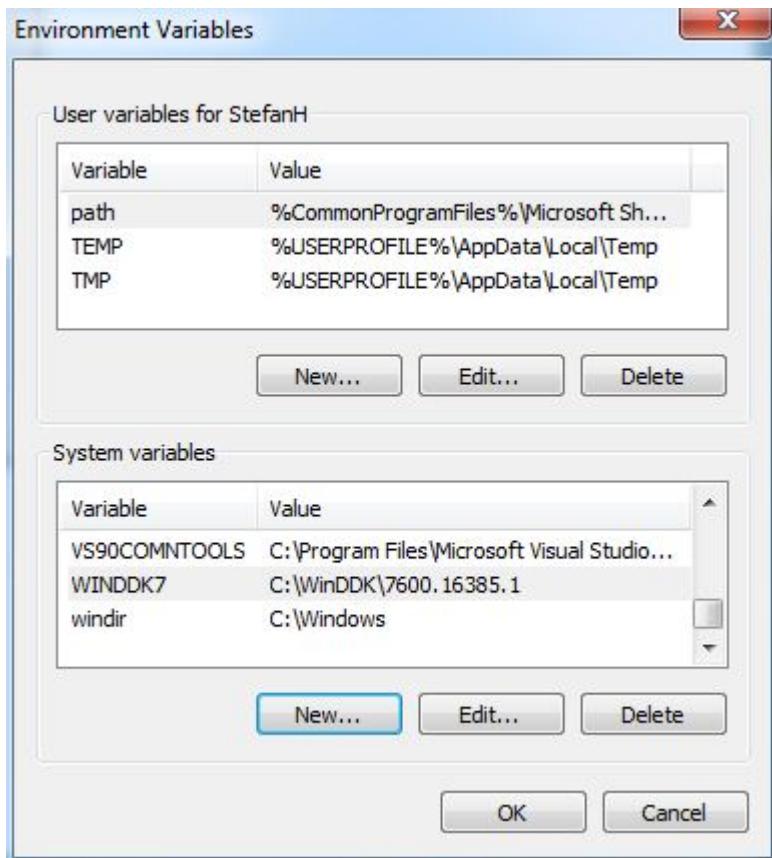
5. Nach Zustimmung zur Microsoft EULA Lizenz Zielordner für die Installation wählen.  
Standardmäßig wird der Stammordner „C:\“ ausgewählt - also wird „C:\WinDDK\7600.16385.1“ vorgeschlagen  
**! HINWEIS! Die Ziffern „7600....“ sind im Falle einer neueren Version von „Windows Driver Kit“ möglicherweise andere.**

6. Installation mit „OK“ starten



7. Zukünftig übernimmt TwinCAT 3 den folgenden Schritt. Jetzt muss er noch einmal manuell erledigt werden.

8. Navigieren Sie zu „Start“-> „Control Panel“ -> „System“ und wählen „Advanced system settings“
9. Wählen Sie den Karteireiter „Advanced“ und klicken dann auf „Environment Variables...“
10. Im unteren Bereich von „System variables“ wählen Sie „New...“ und fügen folgende Information ein:  
Variablenname WINDDK7  
Variablenwert C:\WinDDK\7600.16385.1  
Der Pfad kann bei einer anderen Version von Windows Driver Kit oder bei Angabe eines anderen Installationspfads abweichen.



11. Nach der Installation neu anmelden oder PC neu starten, um die neuen Umgebungsvariableneinstellungen zu übernehmen.

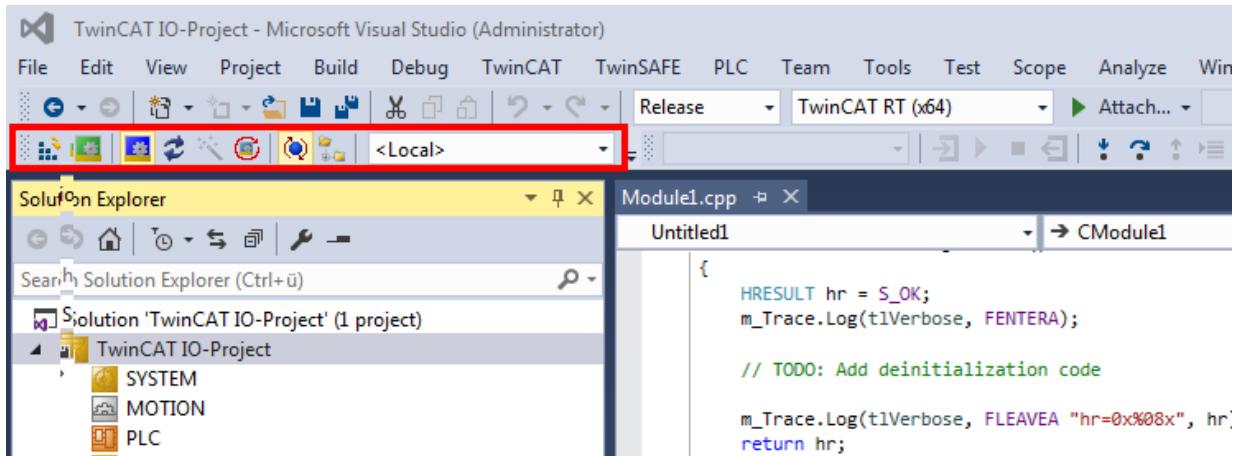
## 5.2 Visual Studio - TwinCAT XAE Base-Symbolleiste

### Fügen Sie für ein effizientes Engineering die „TwinCAT XAE Base“-Symbolleiste hinzu

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, müssen Sie folgendes tun:

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

⇒ Die ausgewählte Symbolleiste erscheint unter dem Menü.



## 5.3 Visual Studio vorbereiten - Konfigurations- und Plattformsymbolleiste

### Symbolleiste „Solution Configuration and Solution Platform“ hinzufügen

Mit der „Configuration and Platform“-Symbolleiste können Sie die Zielplattform für das Erstellen Ihres Projekts bestimmen. Diese Symbolleiste wird vom TwinCAT 3 Setup automatisch zum Visual Studio-Menü hinzugefügt. Wenn Sie sie allerdings manuell hinzufügen möchten, müssen Sie folgendes tun:

1. Öffnen Sie das Menü „View“ und wählen „Toolbars\Customize“
  2. Navigieren Sie zum Karteireiter „Commands“
  3. Aktivieren Sie das Optionsfeld „Toolbar“ und wählen dann aus der Liste der Symbolleisten „Standard“
  4. Klicken Sie auf „Add Command...“
  5. Wählen Sie die Kategorie „Build“, wählen den Befehl „Solution Configurations“ aus und klicken dann auf „Ok“.
  6. Wiederholen Sie den letzten Schritt für den Befehl „Solution Platforms“
  7. Zum Abschluss auf „Close“ klicken
- ⇒ Nun erscheinen beide Befehle unter der Menüleiste



## 5.4 x64: Treibersignierung

Auf x64 PCs müssen TwinCAT C++ Module signiert werden, damit sie ausgeführt werden können.

**HINWEIS!** Lediglich die Ausführung erfordert Zertifikate - das Engineering auf einer x64-Maschine mit Ausführung auf einer x86-Maschine erfordert keine Signierung.

Da ein veröffentlichtes Modul auf verschiedenen PCs lauffähig sein soll, ist für eine Veröffentlichung („Publish“) immer eine Signierung erforderlich.

## 5.4.1 Treiber signieren

### Übersicht

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

Die Signatur, die beim TwinCAT 3 Build-Prozess automatisch generiert 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 \[▶ 23\]](#)“ 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 \[▶ 23\]](#)
- [\(Test-\)Zertifikate löschen \[▶ 25\]](#)

### Weitere Referenzen:

MSDN, Testzertifikate (Windows Treiber),

[http://msdn.microsoft.com/en-us/library/windows/hardware/ff553457\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff553457(v=vs.85).aspx)

MSDN, MakeCert Testzertifikate (Windows Treiber),

[http://msdn.microsoft.com/en-us/library/windows/hardware/ff548693\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff548693(v=vs.85).aspx)

## 5.4.2 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.

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

Einer der folgenden Befehle muss ausgeführt werden

- **Eingabeaufforderung von 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“)

- normale Eingabeaufforderung (Start->Command Prompt) **mit Administratorrechten**, dann ins Verzeichnis %WINDDK7%\\bin\\x86\ wechseln, das die entsprechenden Tools enthält.

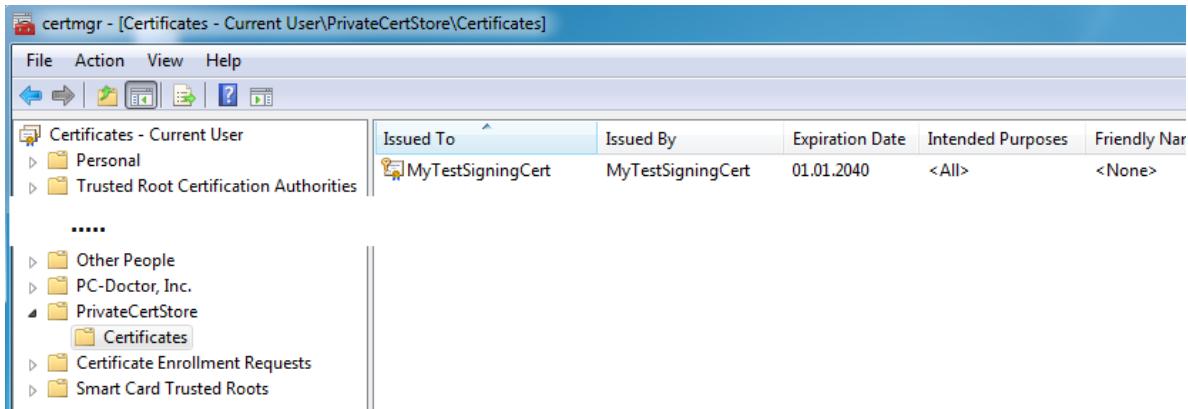
### 1. Auf XAE:

Im Engineering System folgenden Befehl in die Visual Studio 2010 / 2012 Eingabeaufforderung mit Administratorrechten eingeben (siehe Hinweis oben):

```
makecert -r -pe -ss PrivateCertStore -n CN=MyTestSigningCert  
MyTestSigningCert.cer
```

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

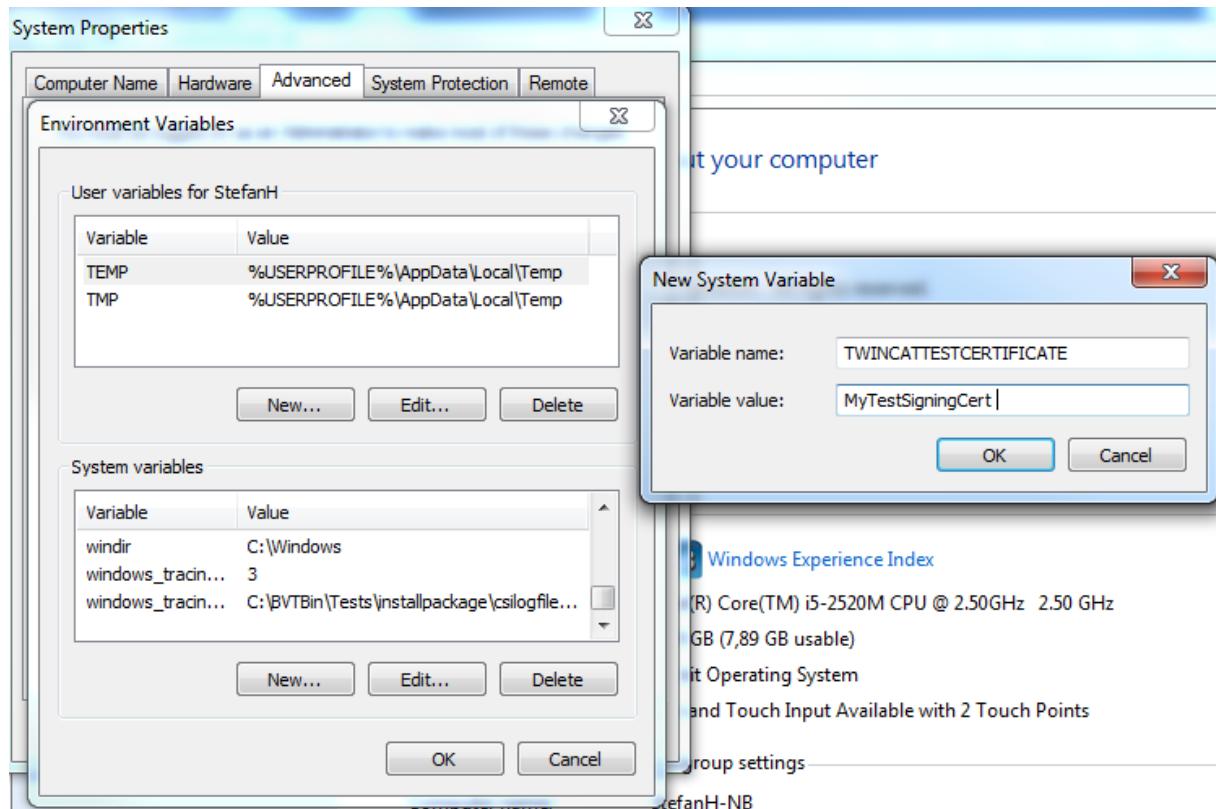
⇒ Das Ergebnis kann mit mmc (Use File->Add/Remove Snap-in->Certificates) überprüft werden:



### 2. Auf XAE:

Zertifikat konfigurieren, sodass es von TwinCAT XAE auf dem Engineering System erkannt wird. Umgebungsvariable „TWINCATTESTCERTIFICATE“ auf „MyTestSigningCert“ im Engineering System setzen oder jeweils den Post Build Step von „Debug|TwinCAT RT (x64)“ und „Release|TwinCAT RT (x64)“ bearbeiten.

**Der Name der Variablen ist NICHT der Name der Zertifikatsdatei, sondern der CN-Name (in diesem Falle MyTestSigningCert).**



### 3. Auf XAR (und XAE, wenn lokaler Test)

Signiermodus aktivieren, damit Windows die selbstsignierten Zertifikate akzeptieren kann. Das ist auf jedem System möglich, das die Module starten kann - Engineering System oder XAR (Laufzeit-)

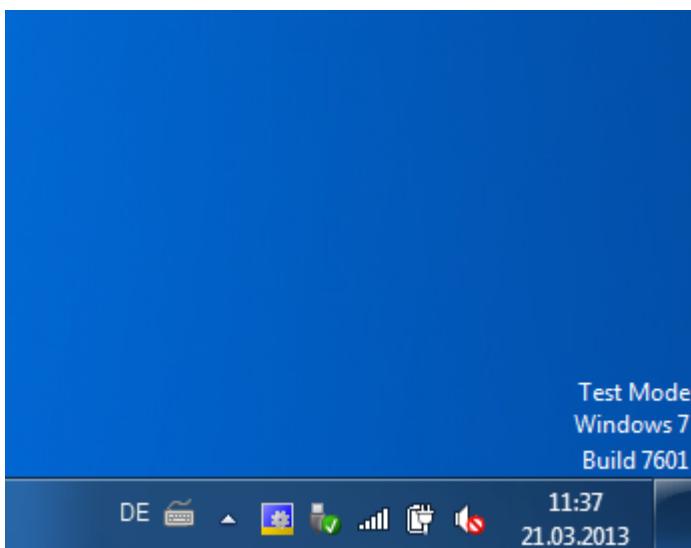
Systeme.

Mittels Eingabeaufforderung folgendes ausführen:

```
bcdedit /set testsigning yes
```

und Zielsystem neu starten.

- ⇒ Wenn der Testsigniermodus aktiviert ist, wird dies rechts unten auf dem Desktop angezeigt.  
Der PC akzeptiert nun alle signierten Treiber zur Ausführung.



4. 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, Testzertifikate \(Windows Treiber\)](#)

[MSDN, MakeCert Testzertifikate \(Windows Treiber\),](#)

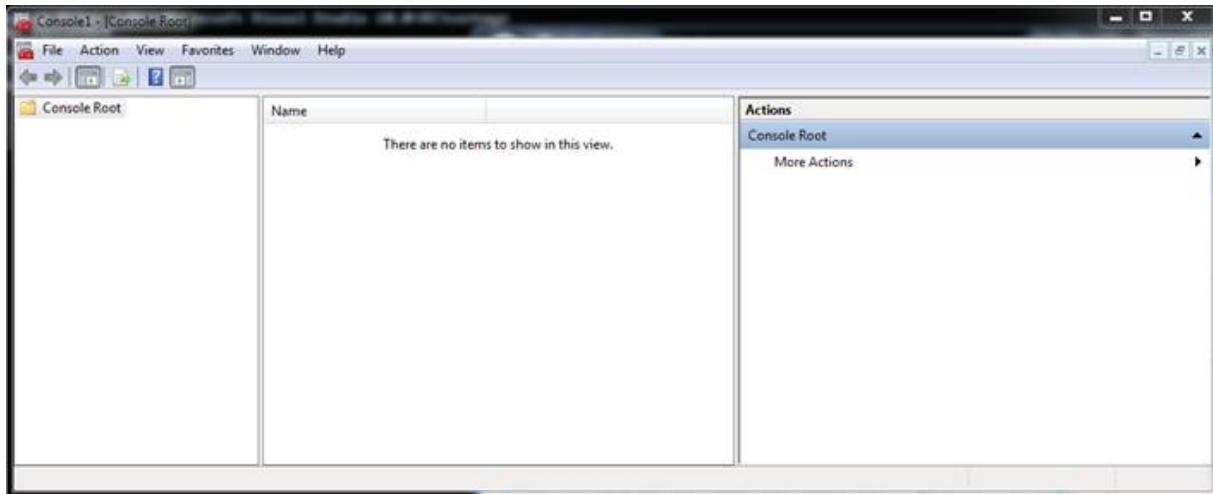
### 5.4.3 Testzertifikat löschen

Dieser Artikel beschreibt, wie ein Testzertifikat gelöscht wird.

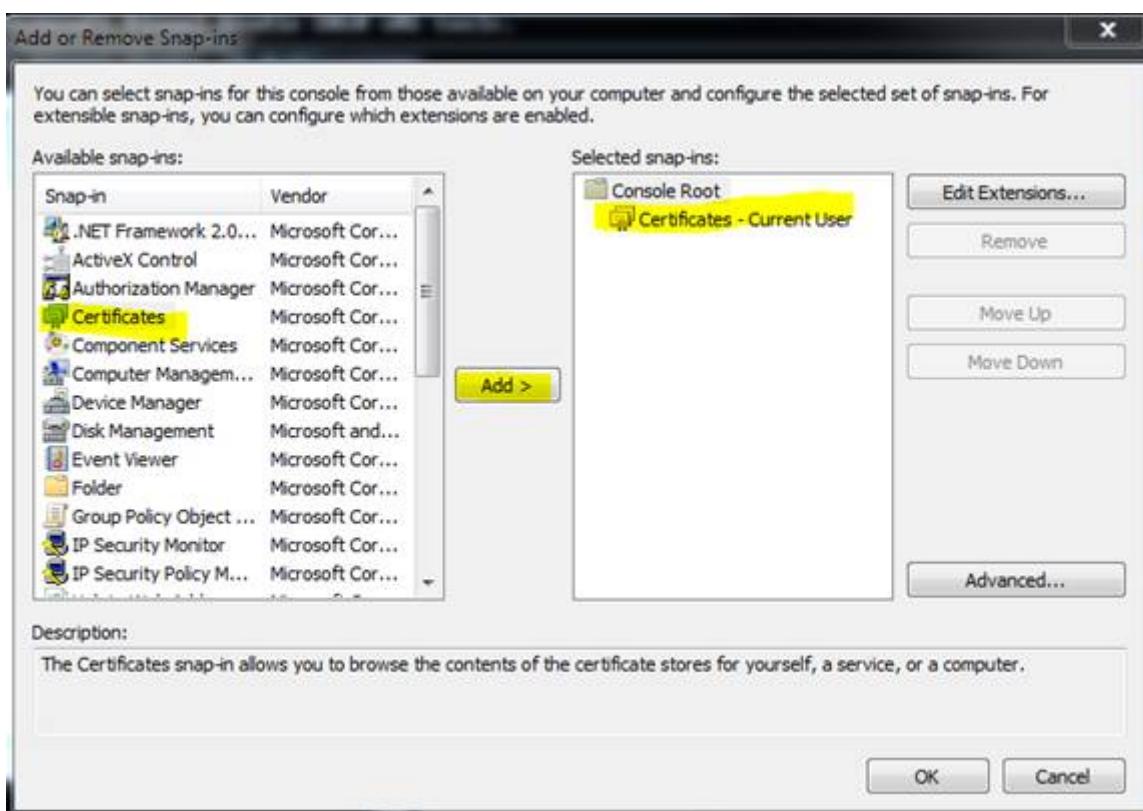
## Übersicht

Ein Zertifikat kann mit der Microsoft Management Console gelöscht werden:

1. Management console MMC.exe über Start-Menü oder Oberfläche starten

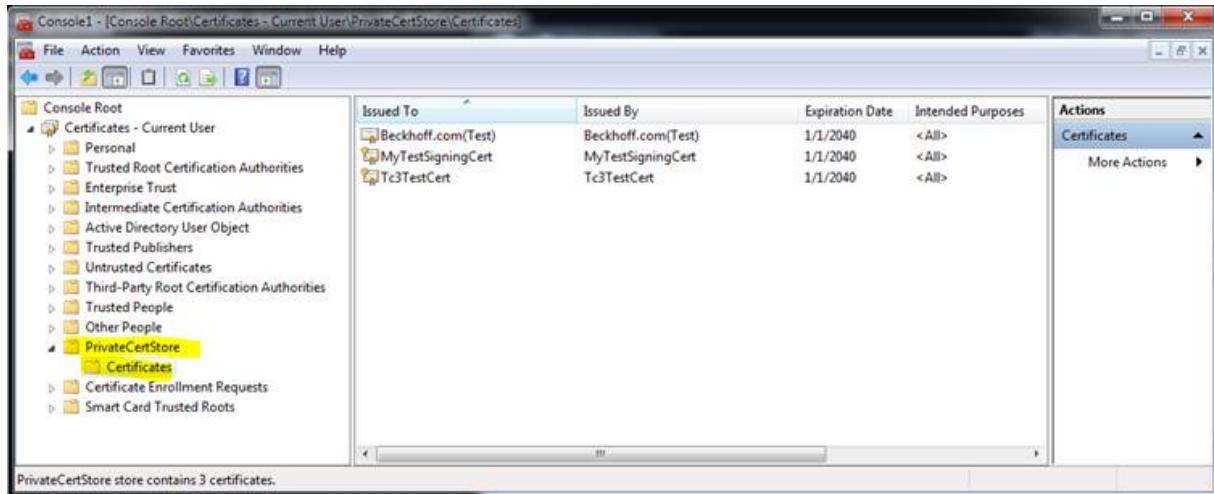


2. Im Menü auf „File“ -> „Add/Remove Snap-in..“ klicken und Zertifikat-Snap-In für den aktuellen Nutzer wählen - mit „OK“ abschließen



⇒ Die Zertifikate sind im Knoten unter „PrivateCertStore/Certificates“ aufgelistet.

3. Zu lösches Zertifikat auswählen.

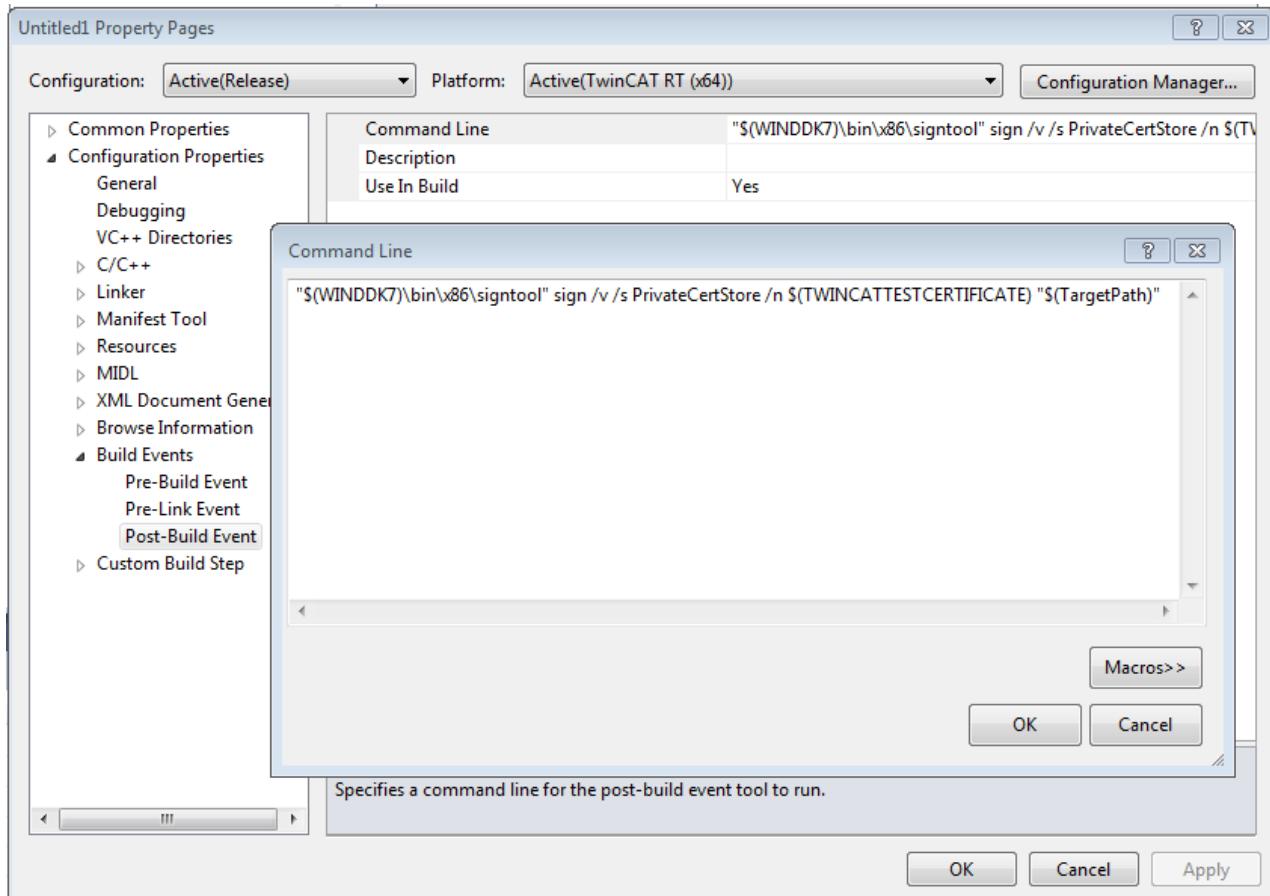


#### 5.4.4 Kundenzertifikate

Wenn der TwinCAT C++ Klassenassistent genutzt wird, wird das Projekt unter Verwendung des zuvor beschriebenen Testzertifikatsverfahren auf x64-Zielen vorbereitet.

Dieses Testsigniersystem kann für den gesamten Engineering- und Testprozess verwendet werden.

Wenn man eine Infrastruktur erstellen und die Kernel-Treiber mit offiziellen „Microsoft trusted“-Zertifikaten unterzeichnen möchte, bieten die Postbuild-Prozeduren der Projekteigenschaften den Einsprungpunkt.



Der Kunde kann einfach den Wert der Umgebungsvariablen **TWINCATTESTCERTIFICATE** ersetzen oder ein anderes zu verwendendes Zertifikat bestimmen.

Der Kunde kann auch das ganze Signierverfahren mit dem SignierTool ändern.

## 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

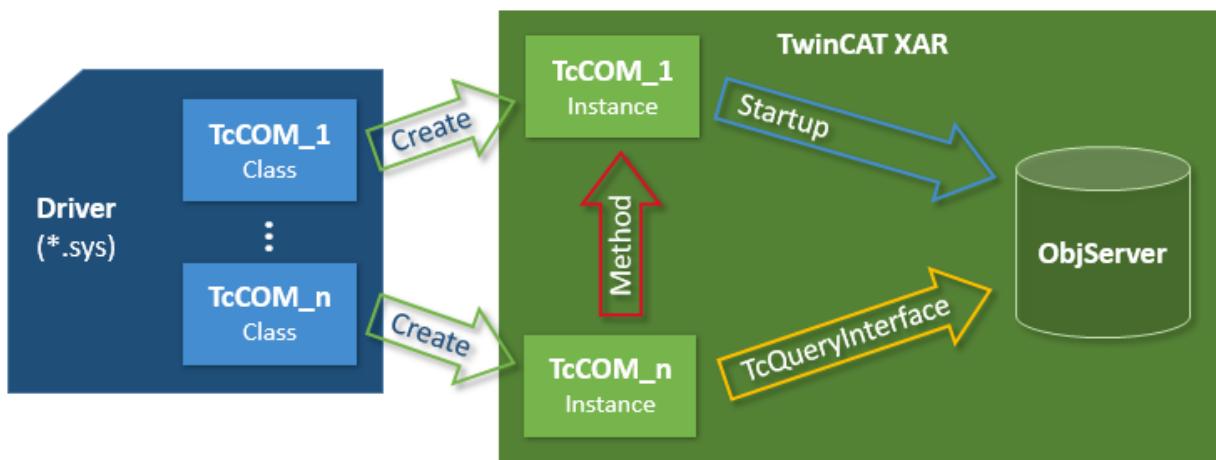
Das TwinCAT Component Object Model definiert die Eigenschaften und das Verhalten der Module. Das vom „Component Object Model“ (COM) von Microsoft Windows abgeleitete Modell beschreibt die Art und Weise, wie verschiedene Software-Komponenten, die unabhängig voneinander entwickelt und kompiliert wurden, 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 Modulinstantz 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-DeviceNotifications zu erhalten.
- Die ITcTask-Schnittstelle, die von den Tasks des Echtzeitssystems implementiert wird, stellen Informationen bezüglich der Zykluszeit, der Priorität und anderer Informationen zum 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.

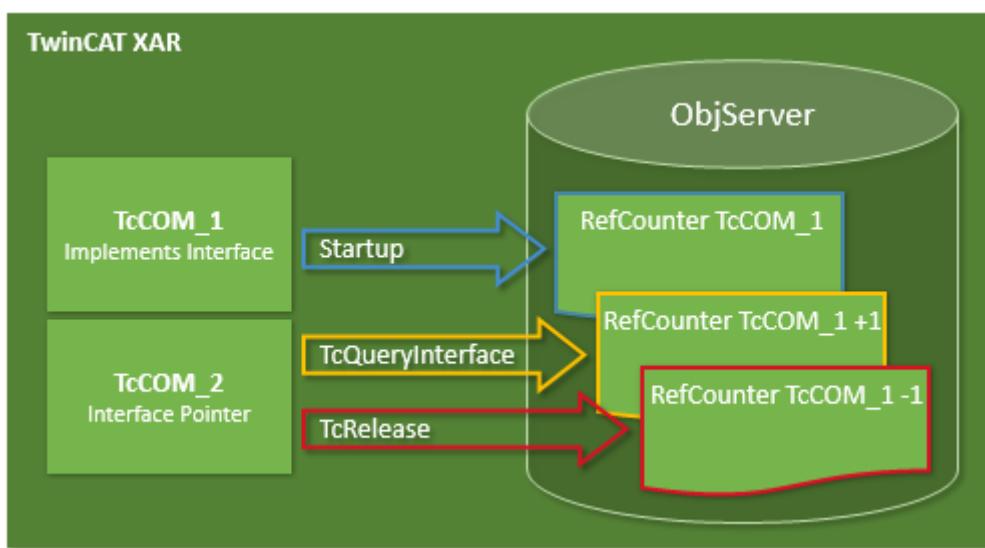
## Class Factories

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.



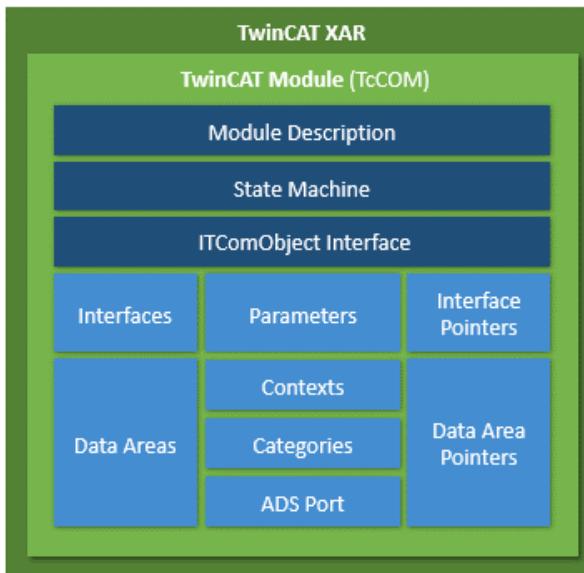
### 6.1.1 TwinCAT-Modul Eigenschaften

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.

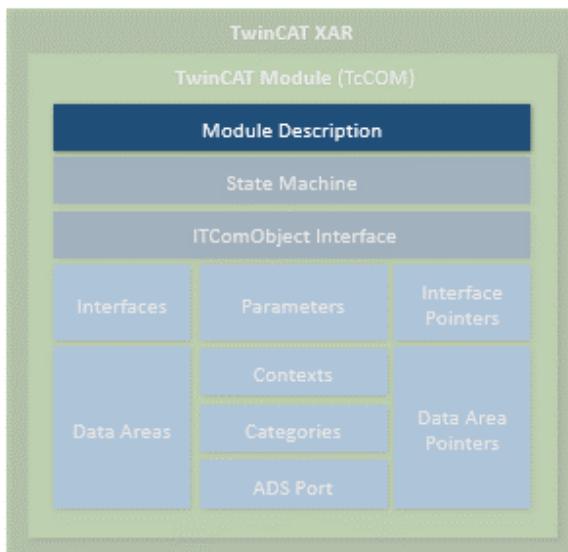
Wenn ein Modul in der TwinCAT Runtime instanziiert wird, dann meldet es sich selber bei einer zentralen Systeminstanz, dem ObjectServer, an. Dadurch wird es für andere Module und auch für allgemeine Tools erreichbar und parametrierbar. Module können unabhängig voneinander kompiliert und demzufolge auch 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 instanziert. 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 Datentypsysteem aufgenommen werden. Damit werden also alle Schnittstellen der im System vorhandenen TMC Beschreibungen für alle Module nutzbar.

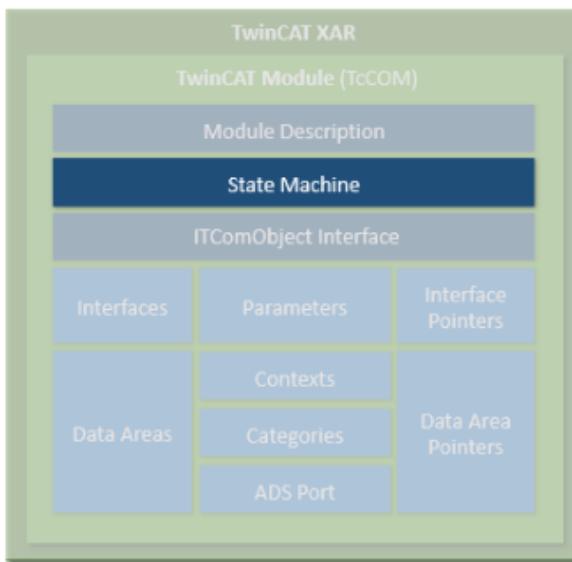
Auch komplexere Konfigurationen mehrerer Module können in den Klassenbeschreibungsdateien beschrieben werden, die für eine spezifische Anwendung vorkonfiguriert und verbunden sind. Demzufolge kann ein Modul für eine komplexe Maschineneinheit, die intern aus einer Reihe von Untermodulen besteht, bereits 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 parametrisiertes Modul in einem neuen Projekt erneut verwendet werden soll.

### Zustandsmaschine

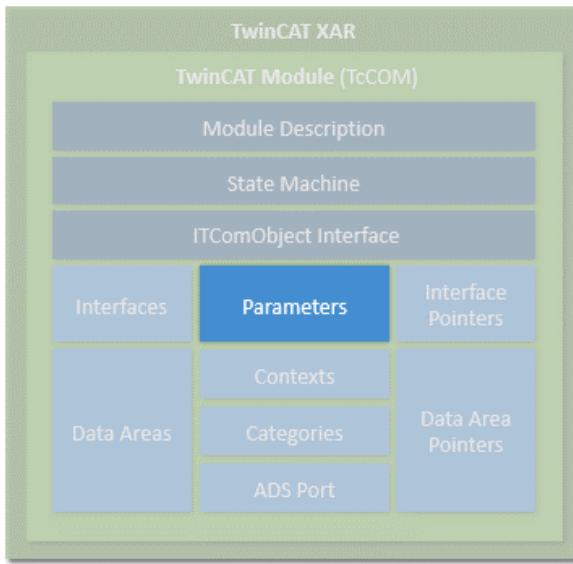


Jedes Modul enthält eine Zustandsmaschine, die den Initialisierungszustand des Moduls und die Mittel, mit denen dieser Zustand von außen verändert werden kann, beschreibt. Diese Zustandsmaschine beschreibt die Zustände, die beim Starten und Beenden des Moduls durchlaufen werden. Dieses betrifft die Modulerzeugung,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

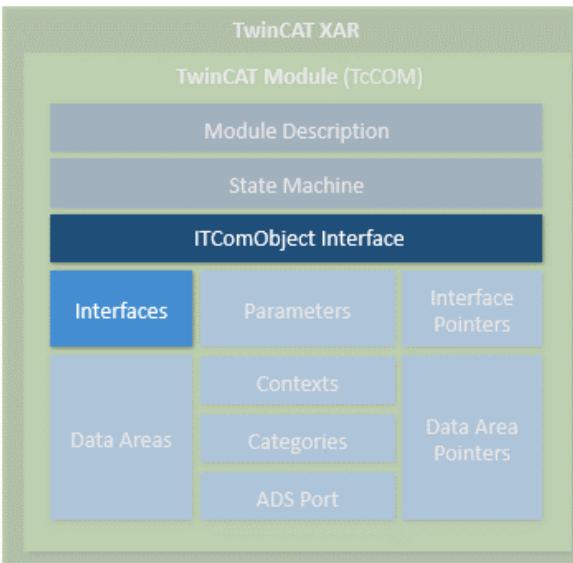
Die Zustandsmaschine ist im Detail separat [beschrieben \[▶ 38\]](#).

## Parameter



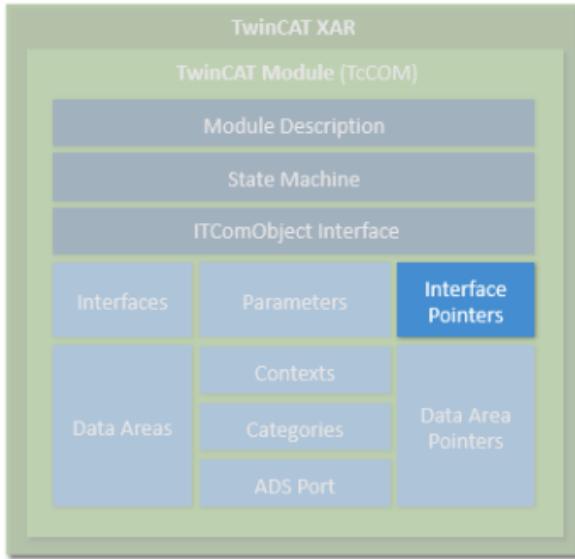
Module können Parameter haben, die während der Initialisierung oder später während der Laufzeit (OP-Zustand) gelesen oder geschrieben werden können. Jeder Parameter ist durch eine Parameter-ID gekennzeichnet. Die Eindeutigkeit der Parameter-ID kann global, eingeschränkt global oder modulspezifisch sein. 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.

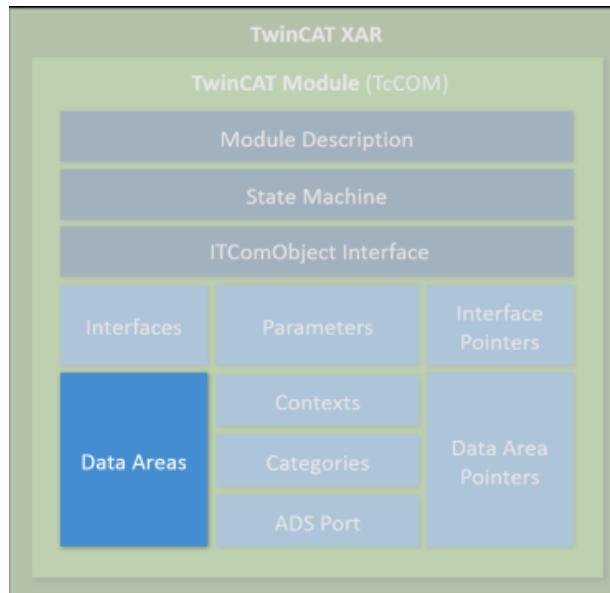
## Schnittstellenzeiger



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.

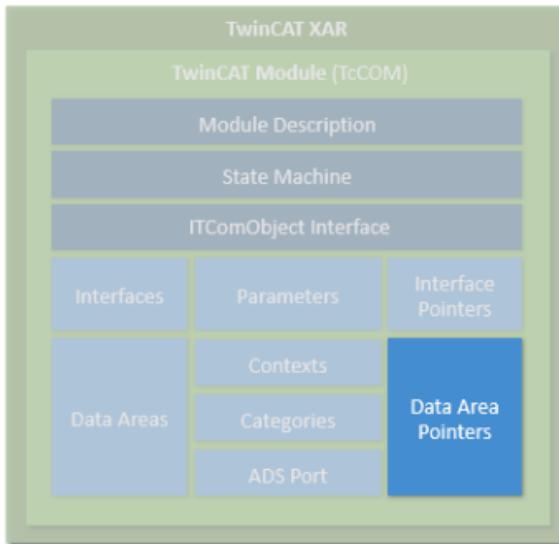
Schnittstellenzeiger werden normalerweise beim Start der Zustandsmaschine gesetzt. Beim Übergang von INIT zu PREOP (IP) empfängt das Modul die Objekt-ID des anderen Moduls mit der entsprechenden Schnittstelle, beim Übergang PREOP zu SAFEOP (PS) oder SAFEOP zu OP (SO) wird die Instanz des anderen Moduls 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



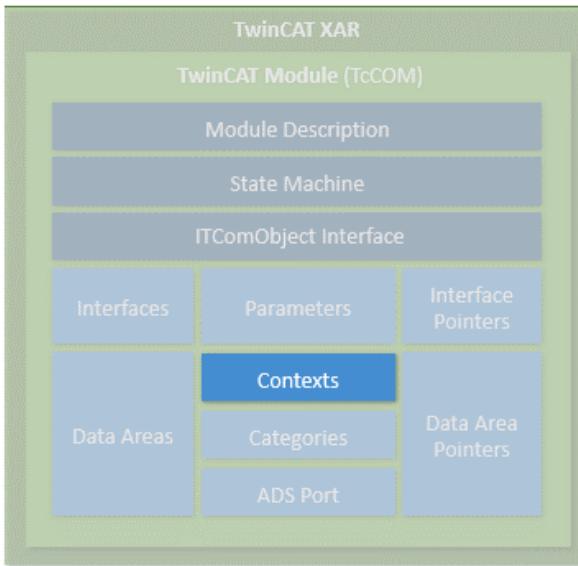
Module können Datenbereiche enthalten, die von der Umgebung verwendet werden können (z.B. von anderen Modulen oder zum IO Bereich von TwinCAT). Diese Datenbereiche können beliebige Daten enthalten. Sie werden oft für Prozessabbilddaten (Ein- und Ausgänge) genutzt. Die Struktur der Datenbereiche wird 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



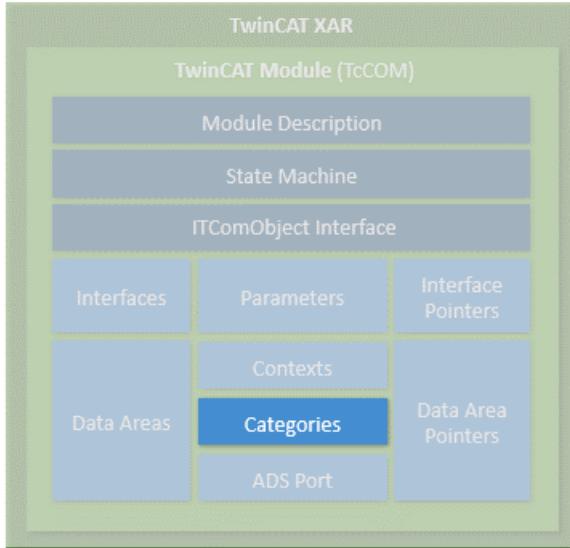
Wenn ein Modul auf den Datenbereich anderer Module zugreifen möchte, kann es Datenbereichszeiger enthalten. Diese werden normalerweise während der Initialisierung der Zustandsmaschine auf Datenbereiche oder Datenbereichsabschnitte anderer Module gesetzt. Es handelt sich dabei um einen direkten Zugriff 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



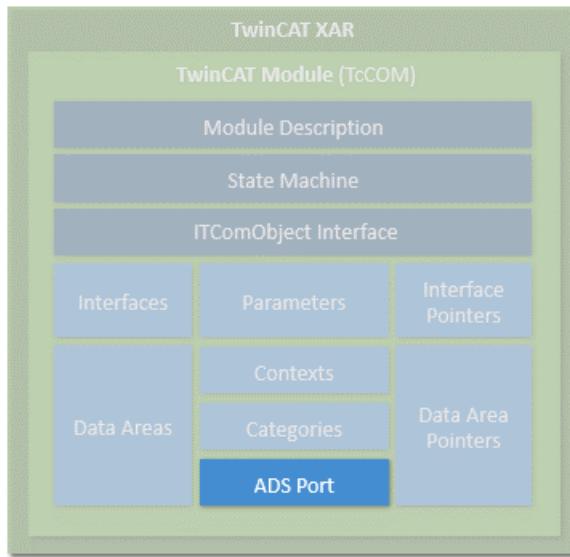
In diesem Zusammenhang ist Kontext als Echtzeit-Task Kontext zu verstehen. Kontexte werden u.a. für die Konfiguration der Module benötigt. Einfache Module arbeiten normalerweise in einem einzigen Zeitkontext, demzufolge muss er nicht näher spezifiziert werden. Andere Module können teilweise in mehreren Kontexten aktiv sein (z.B. ein EtherCAT Master kann mehrere unabhängige Echtzeit Tasks unterstützen, 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 eigenen 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++.

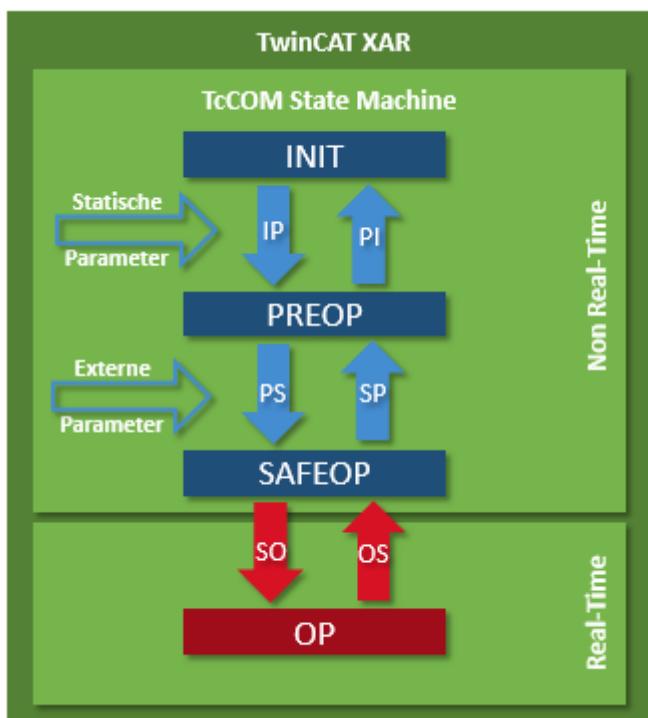
Die SPS-Module melden sich ebenfalls selber beim ObjectServer an und sind demzufolge über ihn erreichbar. Die Komplexität von SPS-Modulen ist unterschiedlich. Es macht keinen Unterschied, ob nur ein kleines Filtermodul generiert oder ein komplettes SPS-Programm in ein Modul hineingepackt wird. Jedes 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 selber 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 alloziert 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 alloziert 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 alloziert werden. Alle allozierten 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 hergestellt.

## 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 alloziert 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.

## 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

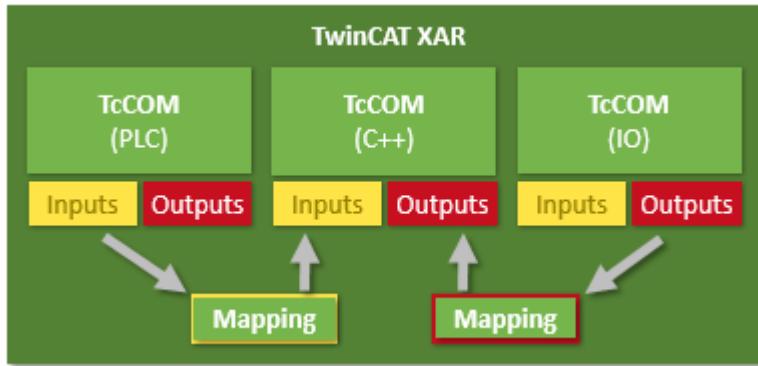
Diese vier Arten werden nun beschrieben.

### 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 [▶ 111] angelegt, die entsprechende Ein-/Ausgänge beschreiben. Diese werden in der TwinCAT Solution dann verknüpft.

Durch ein Mapping werden dabei die Daten zu Task-Begin (Inputs) bzw. am Task-Ende (Outputs) bereitgestellt bzw. übernommen. Die Datenkonsistenz wird hierbei durch synchrones bzw. asynchrones Mapping sichergestellt.

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



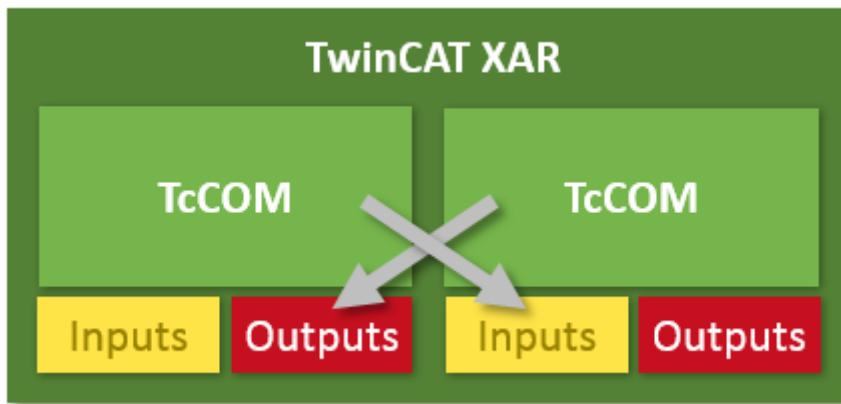
Folgendes Beispiel zeigt die Realisierung:

Beispiel12: Modulkommunikation: Verwendet IO Mapping [▶ 268]

### IO Data Pointer

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



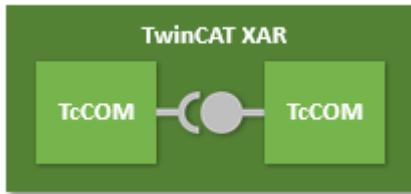
Folgendes Beispiel zeigt die Realisierung:

[Beispiel10: Modulkommunikation: Verwendung von Datenzeigern \[▶ 239\]](#)

#### 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 [▶ 102]) 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 \[▶ 240\]](#)

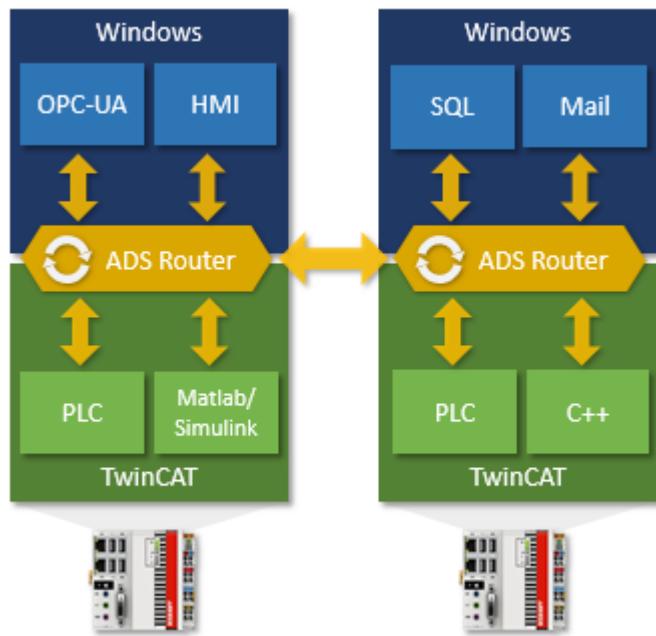
[Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an \[▶ 267\]](#)

[Für die Kommunikation mit der PLC existieren weitere Beispiele \[▶ 285\].](#)

#### 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 (Anbieter) sein. Die implementierende Sprache (PLC, C++, Matlab) spielt dabei keine Rolle.



Folgende Beispiele zeigen die Realisierung:

Beispiel03: C++ als ADS Server [▶ 220]

Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch [▶ 230]

Beispiel07: Empfang von ADS Notifications [▶ 235]

Beispiel08: Anbieten von ADS-RPC [▶ 236]

## 7 Module - Handhabung

TcCOM Module werden definiert und implementiert. Anschließend können sie

- ausgetauscht werden: [Module exportieren \[► 43\]](#), [Module importieren \[► 44\]](#)
- gestartet werden

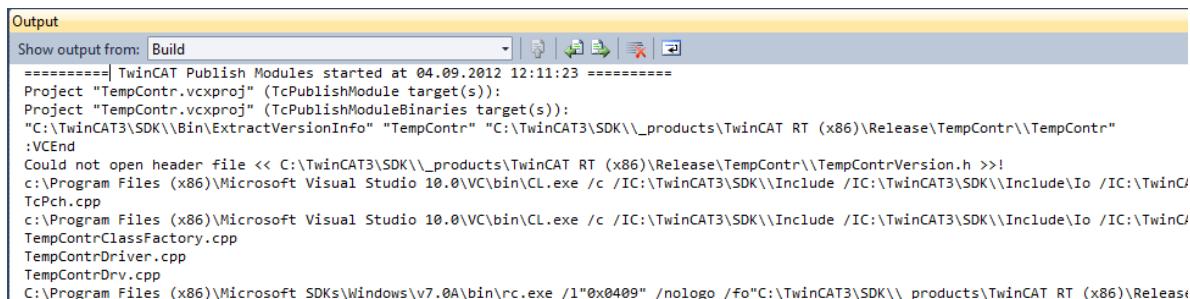
In diesem Abschnitt wird der Umgang mit Modulen beschrieben.

### 7.1 Module exportieren

Dieser Artikel beschreibt, wie ein TwinCAT 3 Modul, das auf jedem anderen TwinCAT PC laufen kann, exportiert wird.

Folgende Schritte sind zu erledigen

1. Implementierung eines TwinCAT 3 C++ Projekts auf einem mit Visual Studio-Version ausgestatteten Engineering PC. Siehe [Schnellstartbeispiel \[► 49\]](#) „Ein TwinCAT 3 Projekt erstellen“, die TwinCAT-Module wie beschrieben implementieren, kompilieren und das Modul vor dem Export testen.
2. Da das Ergebnismodul in der Lage sein sollte, auf jeder Maschine verwendet zu werden, wird TwinCAT eine 32-Bit- und eine 64-Bit-Version des Moduls generieren.  
Da x64bit-Module unterzeichnet sein müssen, muss auf der Maschine, die das Modul exportiert, ein Zertifikat installiert sein. Siehe [x64: Treibersignierung \[► 22\]](#), wie ein Zertifikat zu generieren und installieren ist.  
(Schritt 3 kann auf einem Engineering oder 32-Bit-System ausgelassen werden)
3. Um ein TC 3 C++ Modul zu exportieren, klicken Sie einfach mit der rechten Maustaste auf das Modulprojekt im Solution-Baum und wählen „TwinCAT Publish Modules“ aus.  
⇒ Daraufhin wird das Modul kompiliert (Rebuild) - der erfolgreiche Export wird im Ausgabefenster „Build“ angezeigt.



```
Output
Show output from: Build
=====
TwinCAT Publish Modules started at 04.09.2012 12:11:23 =====
Project "TempContr.vcxproj" (TcPublishModule target(s)):
Project "TempContr.vcxproj" (TcPublishModuleBinaries target(s)):
"C:\TwinCAT3\SDK\Bin\ExtractVersionInfo" "TempContr" "C:\TwinCAT3\SDK\_\_products\TwinCAT RT (x86)\Release\TempContr\TempContr"
:VCEnd
Could not open header file << C:\TwinCAT3\SDK\_\_products\TwinCAT RT (x86)\Release\TempContr\TempContrVersion.h >>!
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /IC:\TwinCAT3\SDK\Include /IC:\TwinCAT3\SDK\Include\Io /IC:\TwinCAT3\SDK\Include\TempContr\TempContrClassFactory.cpp
TempContrDriver.cpp
TempContrDrv.cpp
C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\rc.exe /l"0x0409" /nologo /fo"C:\TwinCAT3\SDK\_\_products\TwinCAT RT (x86)\Release\TempContr\TempContrVersion.h"
```

Most important is the success report at the end:

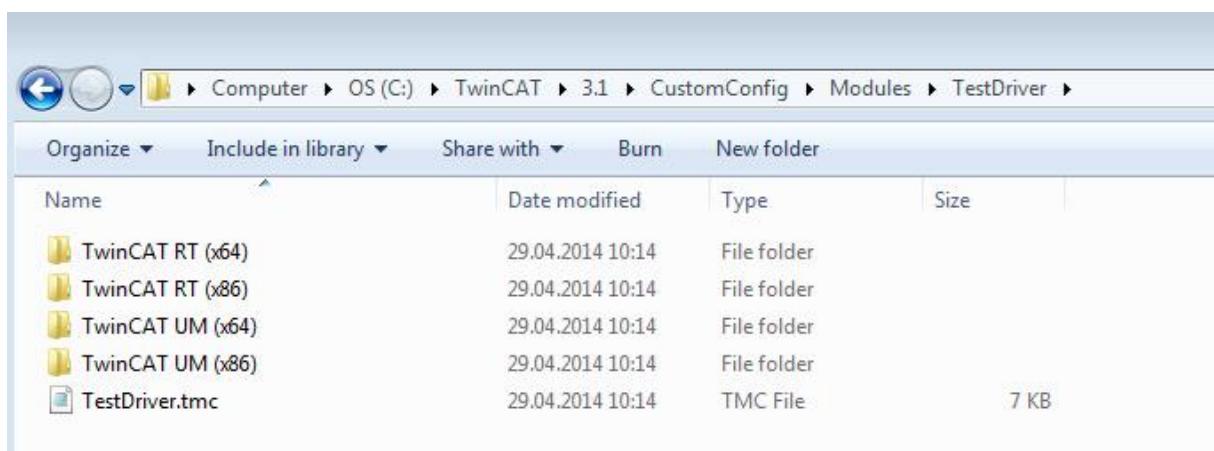
**Output**

Show output from: Build

```
TcPch.cpp
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /IC:\TwinCAT3\SDK\Include
TempContrClassFactory.cpp
TempContrCtrl.cpp
TempContrDrv.cpp
TempContrW32.cpp
C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\rc.exe /D _UNICODE /D UNICODE /I"0x0401
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\link.exe /ERRORREPORT:QUEUE /OUT:"C:
"C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContr\TcPch.obj"
"C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContr\TempContrClassFactory.obj"
"C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContr\TempContrCtrl.obj"
"C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContr\TempContrDrv.obj"
"C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContr\TempContrW32.obj"
    Creating library C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContrW32.lib and ol
TempContr.vcxproj -> C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContrW32.dll
C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\mt.exe /nologo /verbose /out:"C:\TwinCAT3\SDK\_\products\TwinCAT UM (x86)\Release\TempContrW32.lib"
Done building project "TempContr.vcxproj".
Project "TempContr.vcxproj" (TcPublishAdditionalFiles target(s)):
Done building project "TempContr.vcxproj".
Done building project "TempContr.vcxproj".
===== TwinCAT Publish Modules finished at 04.09.2012 12:11:29 =====
```

Die binären Dateien und die TMC-Modulbeschreibung werden in den Ordner „TempContr“ unter „C:\TwinCAT\3.x\CustomConfig\Modules“ exportiert.

- Kopieren Sie für den Import lediglich den Ordner „TempContr“ auf jede andere TwinCAT 3 Maschine.



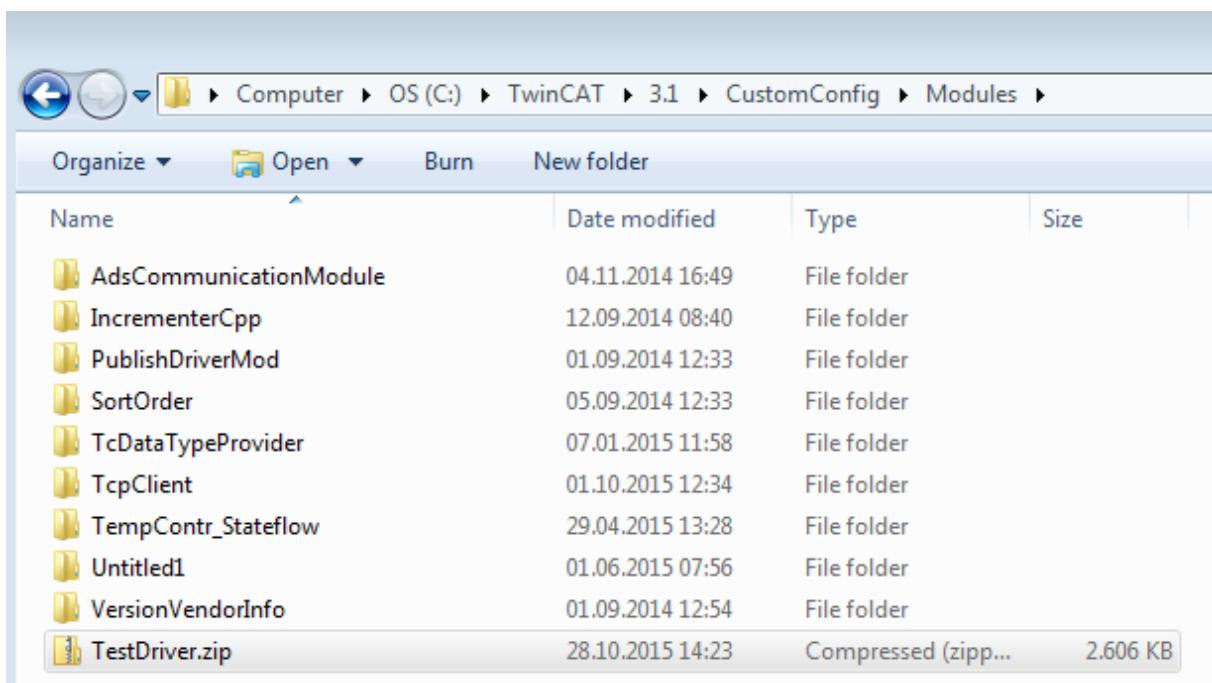
## 7.2 Module importieren

In diesem Artikel ist beschrieben, wie ein binäres TC3-Modul in eine „PC/IPC“-Steuerung mit TwinCAT 3 XAE (ohne Vollversion von Visual Studio), importiert und integriert werden kann.

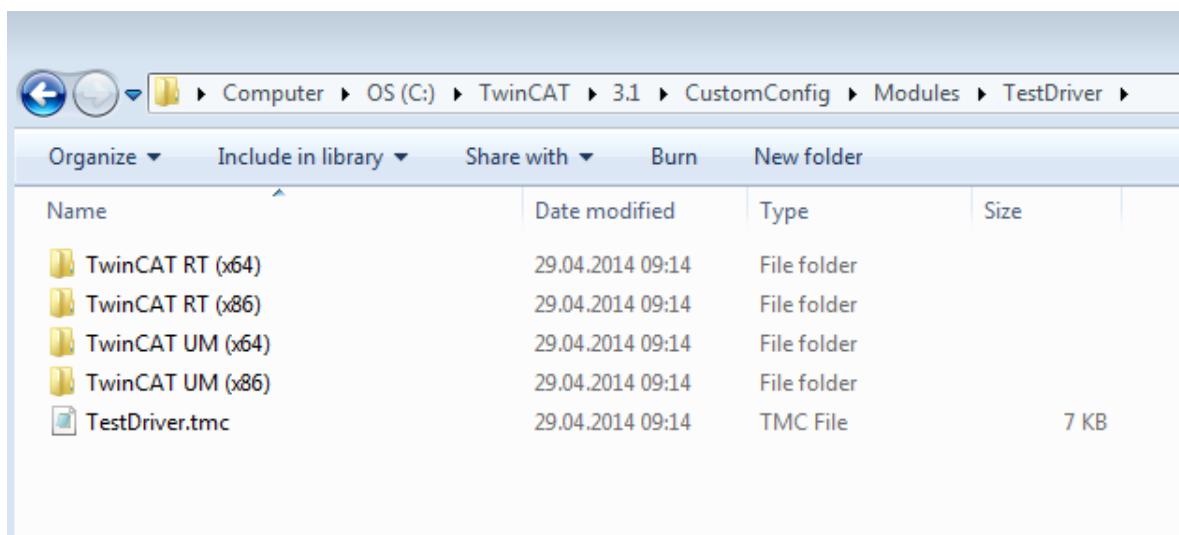
Das binäre TC3-Modul wurde vorher auf einem anderen PC implementiert und exportiert.

Folgende Schritte sind zu erledigen

1. Auf dem zweiten IPC mit TwinCAT XAE ohne Vollversion von Visual Studio binäres Modul in den Zielordner „..\\TwinCAT\\3.x\\CustomConfig\\Modules“ kopieren. In diesem Beispiel wird das Archiv „TestDriver.zip“ entpackt.

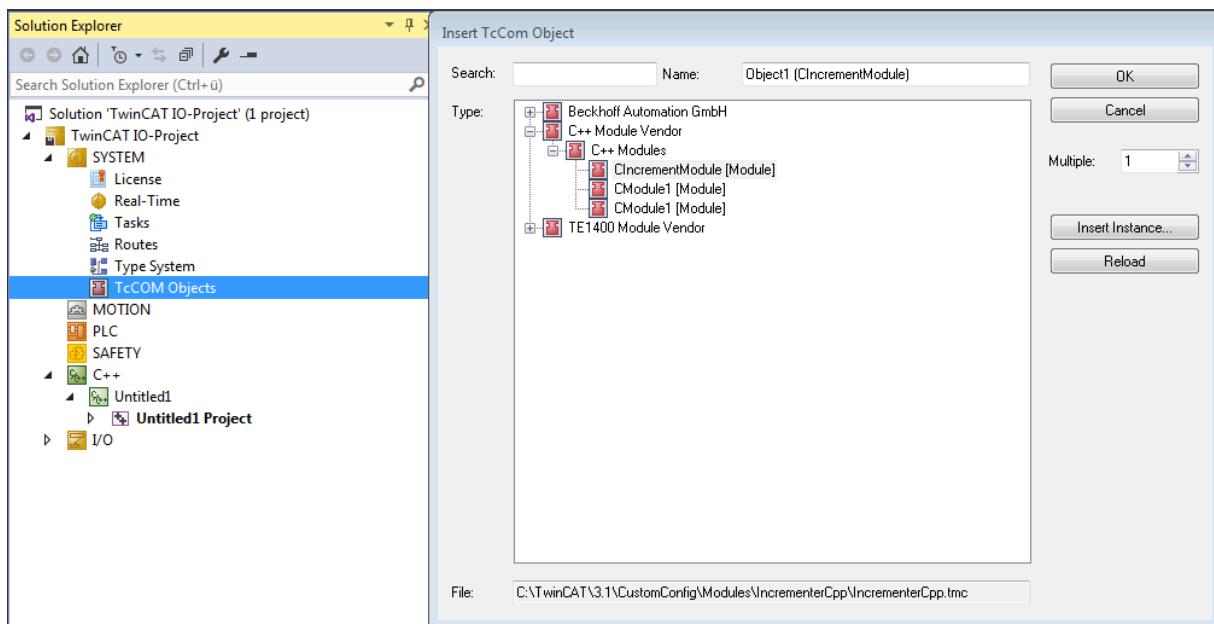


⇒ Daraufhin stellt das „TestDriver“-Modul die binären Module (in den Unterordnern RT und UM) und die entsprechende TwinCAT Modul Class \*.tmc Datei „TestDriver.tmc“ zur Verfügung.



2. Starten Sie die TwinCAT XAE Umgebung und erstellen ein TwinCAT 3 Projekt.

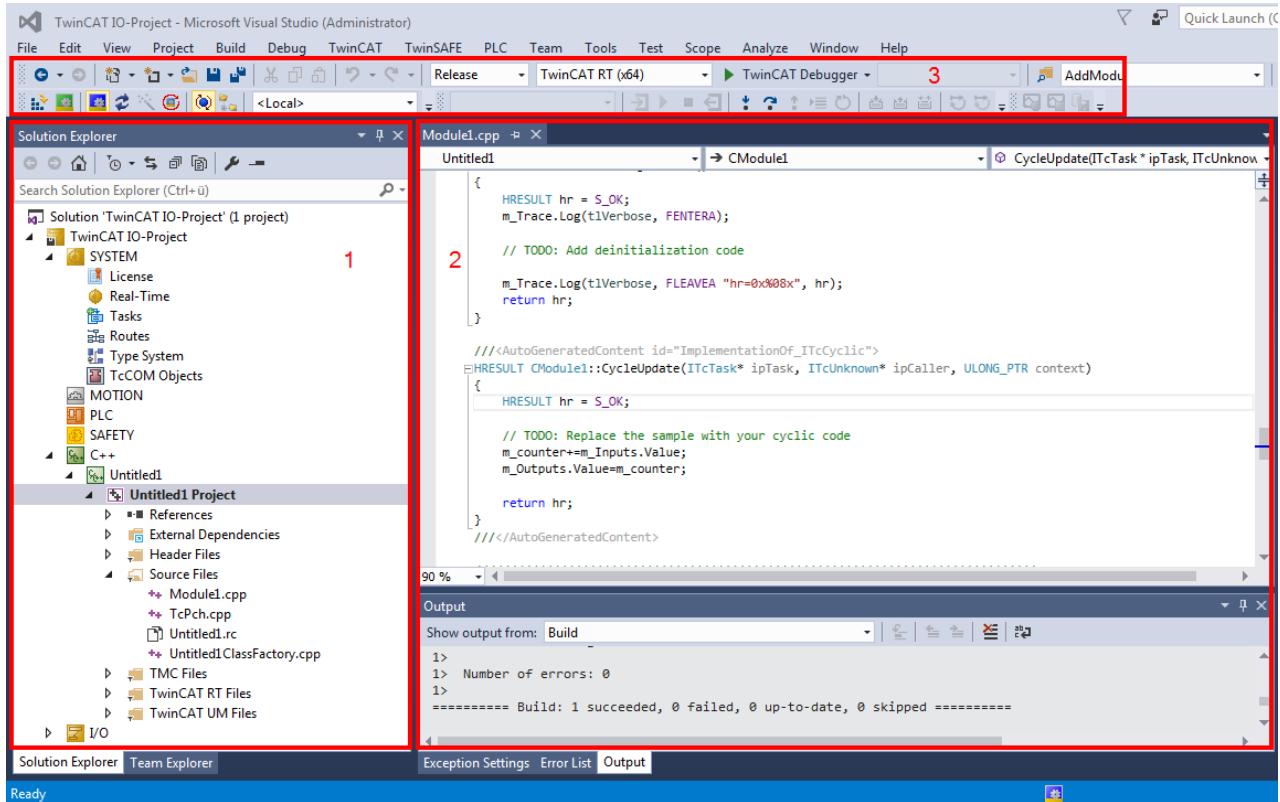
3. Klicken Sie mit der rechten Maustaste auf „System->TcCOM Objects“ und wählen „Add New Item...“ aus.



4. Im eingeblendeten Dialog wird das neue CTestModule Modul aufgelistet. Erstellen Sie eine Modulinstanz, indem Sie den Modulnamen auswählen und mit „OK“ fortfahren.
5. Die Instanz des **TestModule** Moduls erscheint nun unter „**TcCom Objects**“.
6. Die nächsten Schritte wie gehabt: Einen neuen Task erzeugen und
7. Zu „Context“ der Modulinstanz gehen und C++ Modulinstanz mit dem zuvor hinzugefügten „**Task 1**“ verknüpfen.
8. Konfiguration aktivieren

## 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 [▶ 29] 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 [▶ 40], 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) [▶ 196]) umgeschaltet werden. In den Editoren können die üblichen Features wie Breakpoints (vgl. Debuggen [▶ 67]) 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 Knöpfe geben schnellen Zugriff auf häufig genutzte Funktionen. Der „TwinCAT Debugger“ ist der Knopf um die Verbindung zum Zielsystem in Bezug auf C++ Module aufzubauen (die PLC 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 Breakpoints 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 \[▶ 49\]](#) geht die gemeinsamen Schritte im Einzelnen durch.

#### 1. Typen-Deklaration und Modultyp:

Der [TwinCAT Module Class Editor \(TMC\) \[▶ 79\]](#) 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.

Bearbeiten und den Code Generator starten kann immer und immer wieder 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 \[▶ 67\]](#) des benutzerdefinierten Codes innerhalb der Code-Vorlage verwendet.

#### 3. [Module \[▶ 29\]](#) instanziieren

Das Programm beschreibt eine Klasse, die als Objekte instanziert wird. Der [TwinCAT Module Instance Configurator \[▶ 123\]](#) 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) (siehe [Module exportieren \[▶ 43\]](#) / [Module importieren \[▶ 44\]](#))

Die Veröffentlichung eines Moduls erstellt die Treiber für alle Plattformen und bereitet es für die Verteilung vor. Das erzeugte Verzeichnis kann verteilt werden, ohne dass der Quellcode übergeben werden muss. Es wird ausschließlich Binärkode mit Schnittstellenbeschreibung übergeben.

#### 7. Signatur (siehe [x64: Treibersignierung \[▶ 22\]](#))

Die TwinCAT-Treiber müssen für x64-Laufzeiten signiert sein, weil 64-Bit-Windows Versionen fordern, dass Kernel-Module signiert sind. Also gilt dies sowohl für die x64-Erstellung als auch für das Veröffentlichen von Modulen, weil diese Module die x64-Binärcodes enthalten (falls nicht deaktiviert, wie [hier \[▶ 200\]](#) beschrieben).

Der Signaturprozess kann wie [hier \[▶ 27\]](#) beschrieben benutzerdefiniert 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 \[▶ 67\]](#) in Echtzeit, sowie das Setzen von (bedingten) Haltepunkten ist im Falle von TwinCAT C++ Modulen möglich.

⇒ Das Modul läuft unter Echtzeitbedingungen.

## 9 Schnellstart

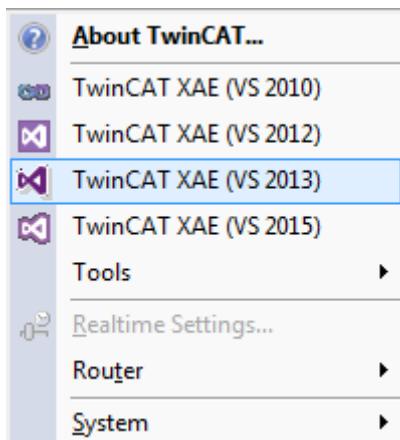
Dieser Schnellstart zeigt, wie man in kurzer Zeit mit dem TwinCAT C++ Modul Engineering vertraut werden kann.

Jeder Schritt der Erstellung eines im Echtzeitkontext laufenden Moduls ist ausführlich beschrieben.

### 9.1 TwinCAT 3 Projekt erstellen

#### TwinCAT Engineering Umgebung (XAE) starten

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



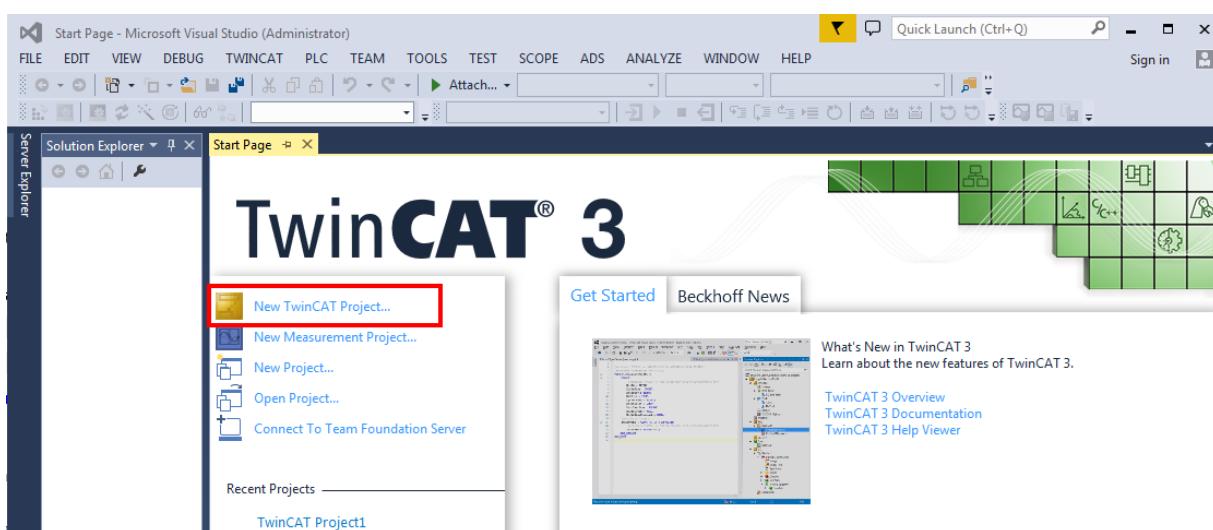
Dabei werden die bei der Installation erkannten und von TwinCAT unterstützten Visual Studio Versionen angeboten.

Alternativ kann das Visual Studio auch über das Start-Menü gestartet werden.

#### TwinCAT 3 C++ - Projekt erstellen

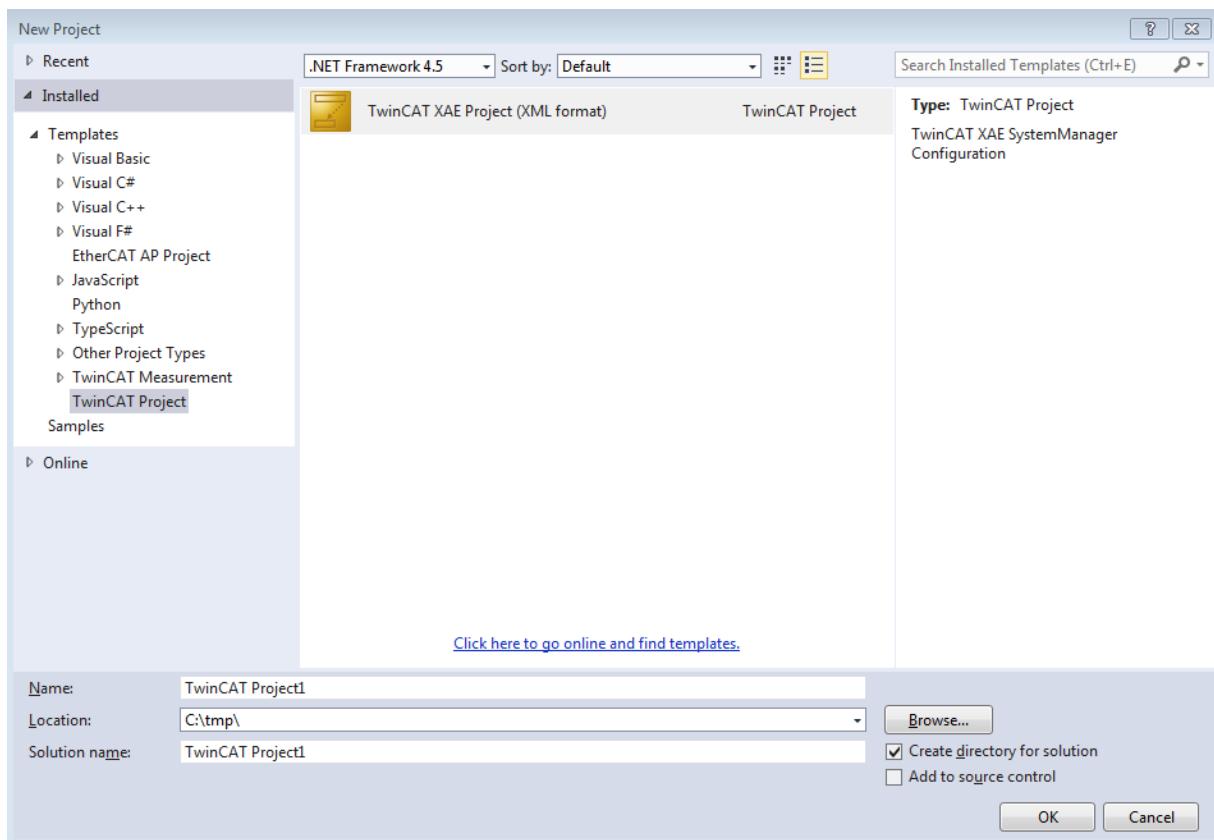
Um ein TwinCAT C++ Projekt zu erstellen, folgende Schritte ausführen:

1. Über die Start Page „New TwinCAT Project ...“ auswählen.

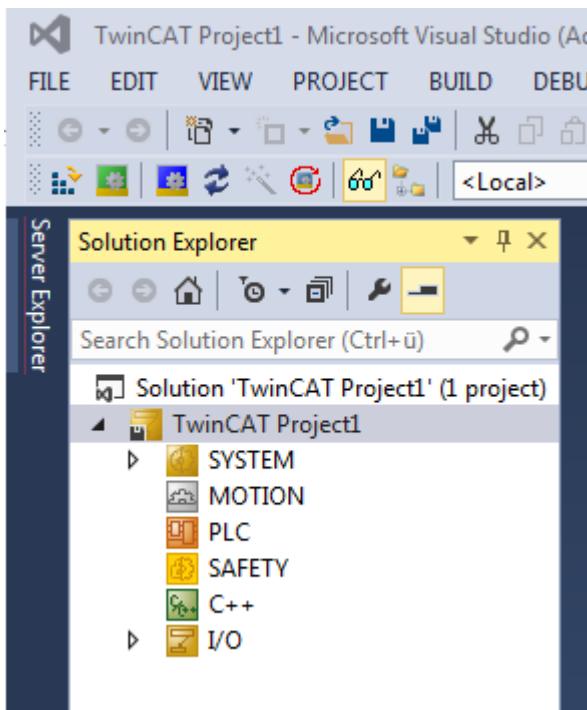


2. Alternativ kann auch mit Klick auf: File -> New -> Project ein Projekt angelegt werden  
⇒ Es werden alle bestehenden Projektvorlagen angezeigt.
3. Wählen Sie „TwinCAT XAE Project“ - optional kann ein passender Projektname eingegeben werden

4. Klick auf „OK“. Sie können den Namen des Verzeichnisses ab sofort nicht mehr wählen oder umbenennen. Behalten Sie die Standardeinstellungen bei (gewählte Option „Create directory for solution“).



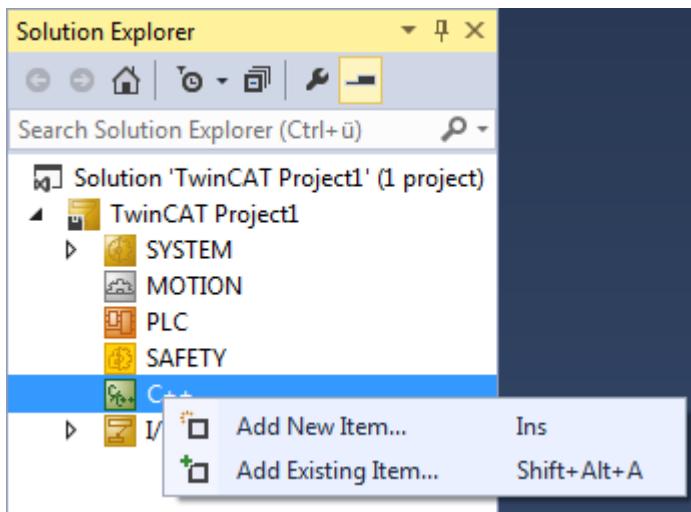
⇒ Daraufhin zeigt der Visual Studio Solution Explorer das TwinCAT 3 Projekt.



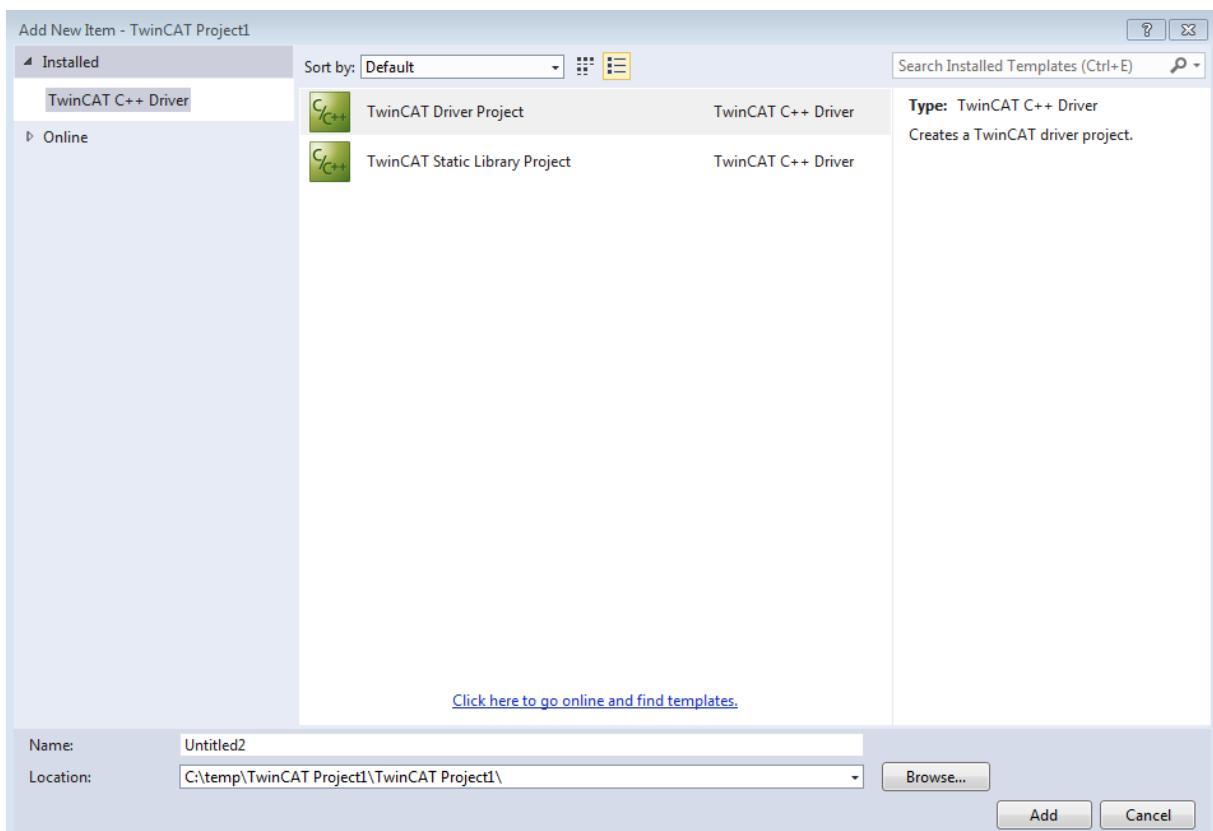
## 9.2 TwinCAT 3 C++ Projekt erstellen

Nach der Erstellung eines TwinCAT 3 Projekts, den „C++“ Knoten öffnen und den folgenden Schritten folgen:

- 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. [Anforderungen \[► 18\]](#)).

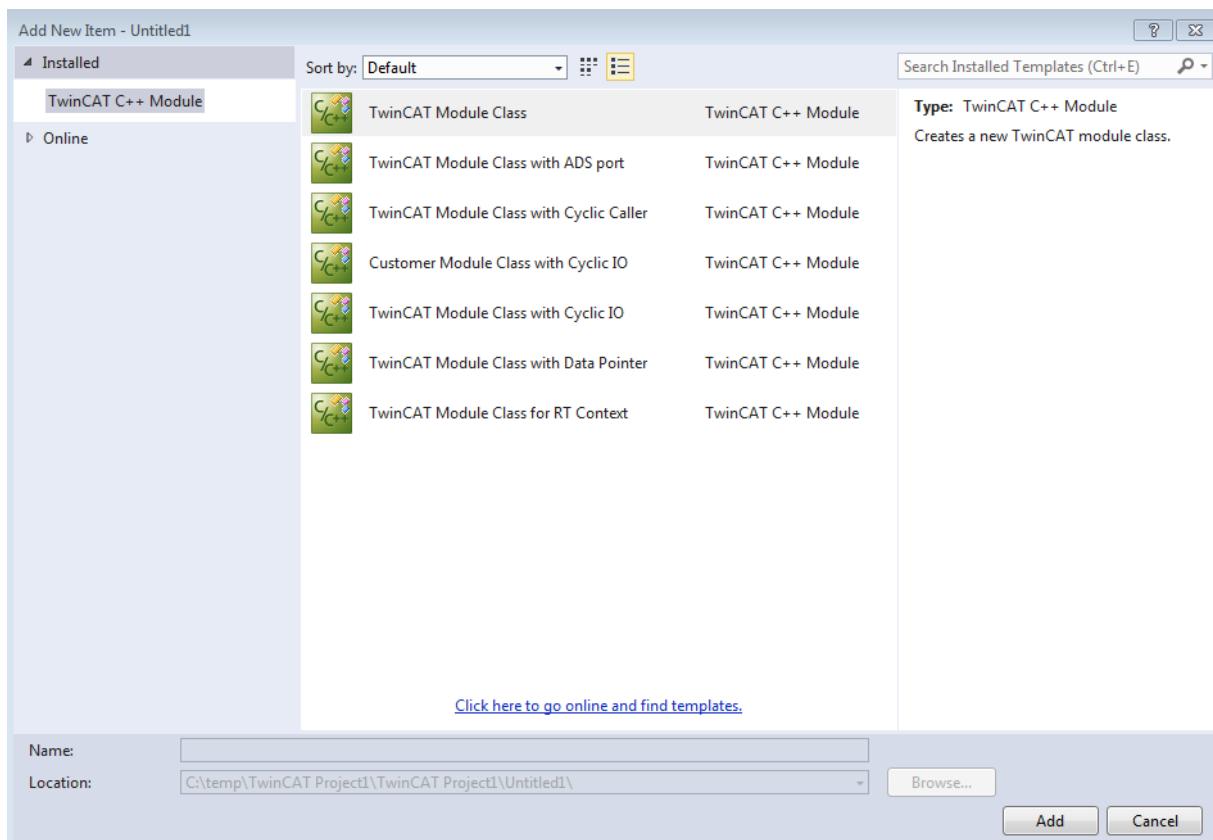


- ⇒ Der „[TwinCAT C++ Projekt-Assistent \[► 75\]](#)“ wird eingeblendet und alle bestehenden Projektvorlagen werden aufgeführt.
- 2. A) Wählen Sie „TwinCAT Driver Project“ aus, geben optional einen verwandten Projektnamen ein und klicken auf „OK“.  
B) Alternativ verwenden Sie das „[TwinCAT Static Library Project](#)“, das eine Umgebung für das Programmieren von statischen TC-C++ Bibliotheken bereitstellt (siehe [Beispiel 25 \[► 278\]](#))

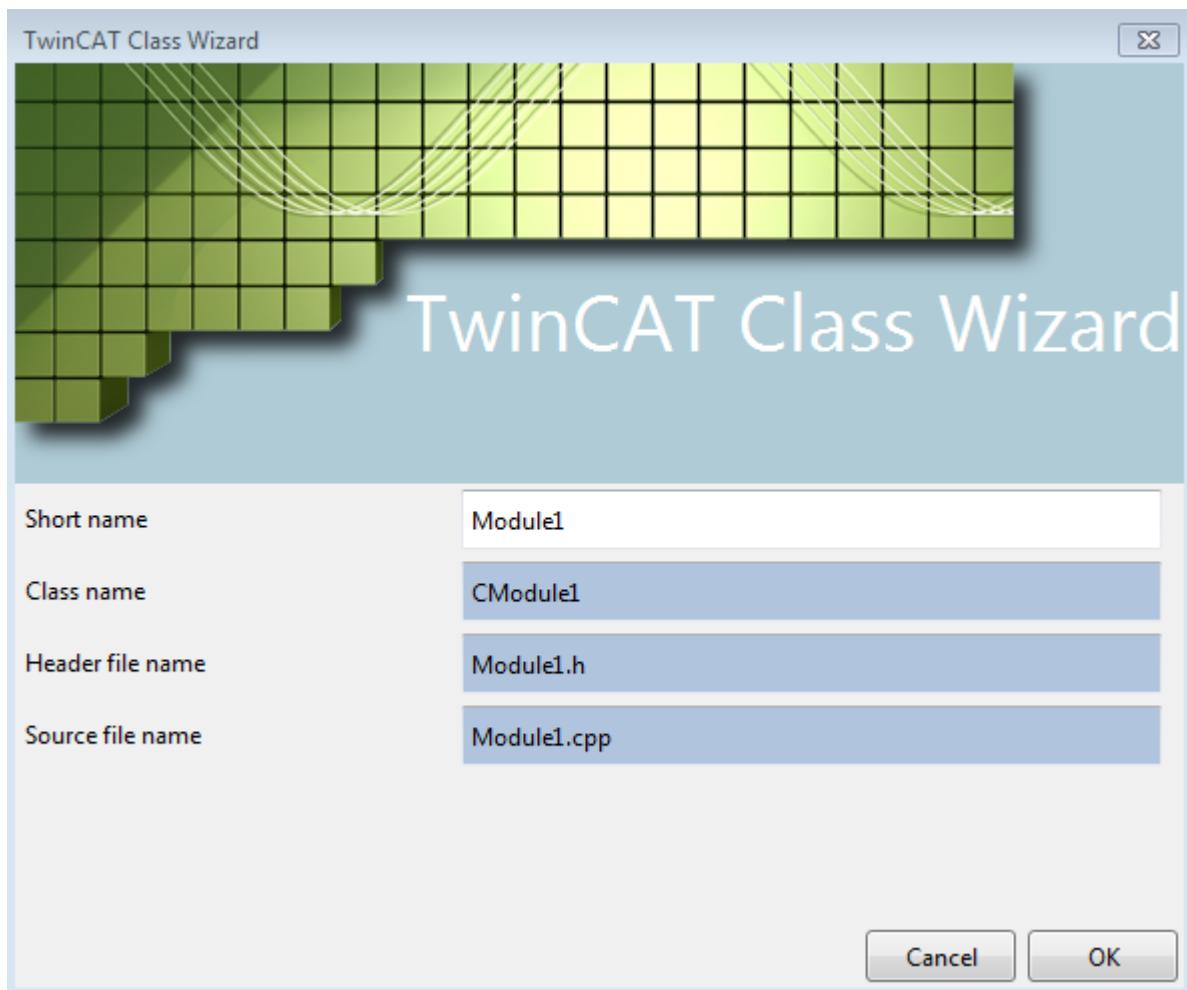


- ⇒ Der „[TwinCAT Modul-Assistent \[► 76\]](#)“ wird eingeblendet.

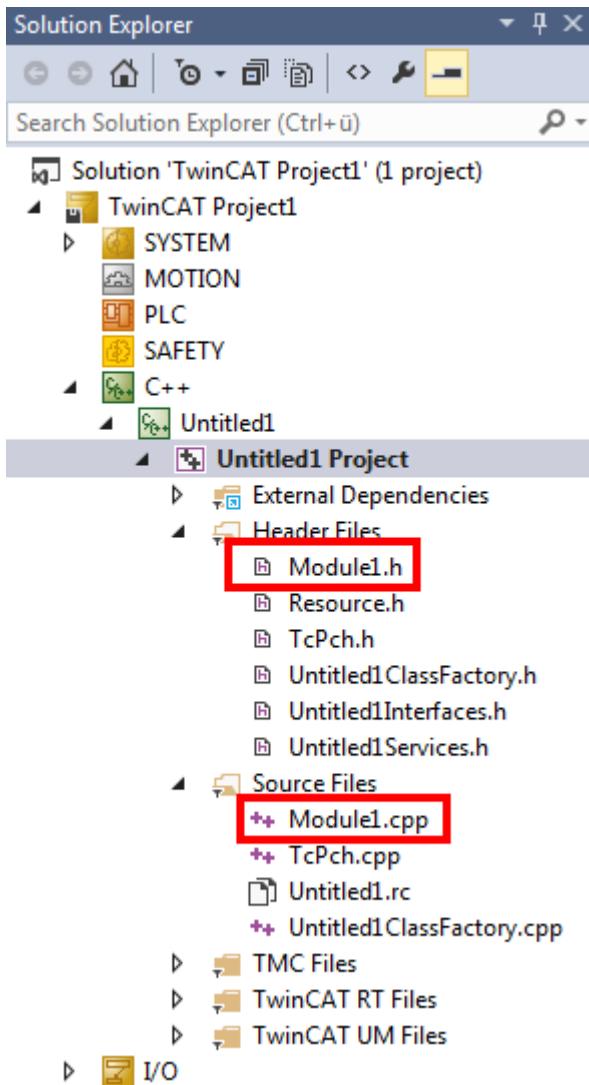
3. Wählen Sie in diesem Fall „TwinCAT Module Class with Cyclic I/O“ aus und klicken auf „OK“. Ein Name ist nicht erforderlich und kann hier auch nicht eingegeben werden.



4. Geben Sie im Dialogfenster „TwinCAT Class Wizard“ einen eindeutigen Namen ein oder fahren Sie mit dem Vorschlag „Object1“ fort.



⇒ Daraufhin wird ein TwinCAT 3 C++ Projekt mit einem Treiber auf Basis des ausgewählten Templates erstellt:

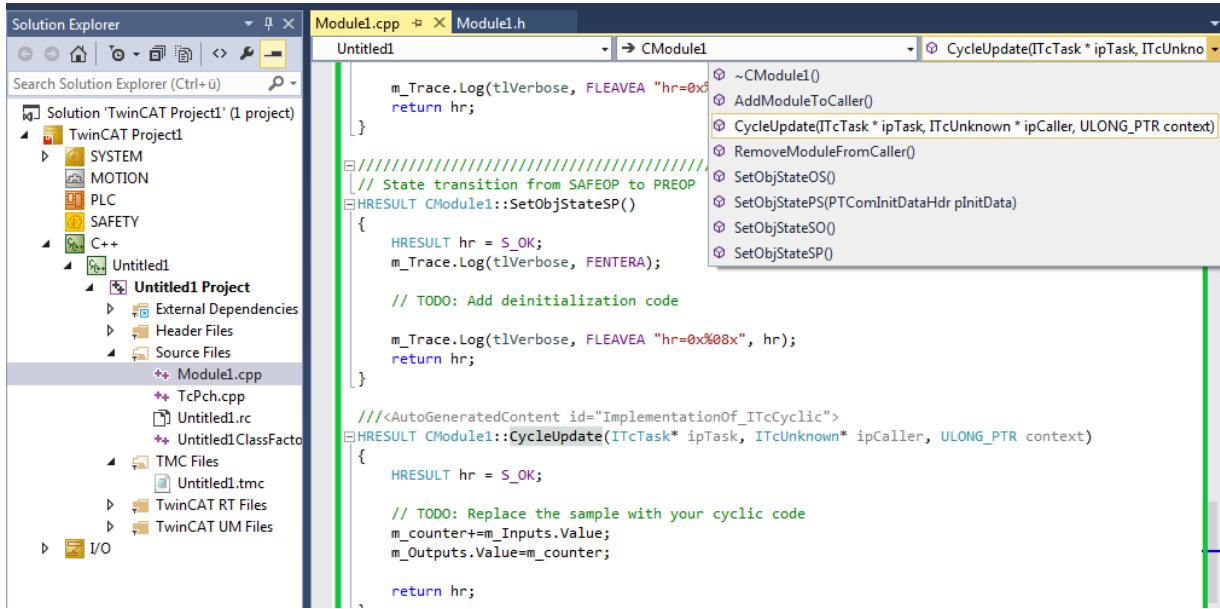


## 9.3 TwinCAT 3 C++ Projekt implementieren

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.

- Die Methode `<MyClass>::CycleUpdate()` wird zyklisch aufgerufen - das ist die Stelle, wo die zyklische Logik zu positionieren ist. An dieser Stelle wird der gesamte zyklische Code hinzugefügt. Zur Navigation können die Dropdown Menüs am oberen Rand des Editors verwendet werden, wie im Screenshot gezeigt



- 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

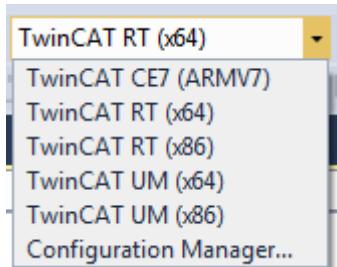
`m_counter++;`

- Speichern Sie die Änderungen.

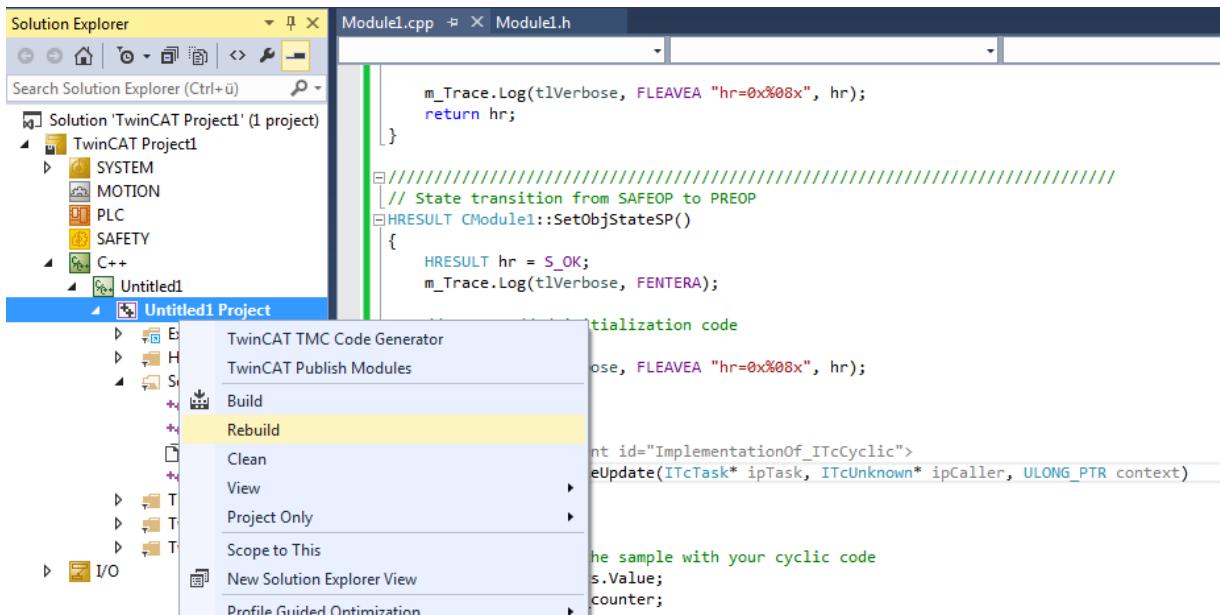
## 9.4 TwinCAT 3 C++ Projekt kompilieren / bauen

In diesem Artikel wird beschrieben, wie eine bereits implementierte C++ Modulklasse erstellt (kompiliert) wird.

- Wählen Sie die Zielplattform aus, entsprechend der die Kompilierung vorgenommen werden soll. TwinCAT wird diese Einstellung bei Auswahl eine Zielsystems überprüfen und ggf. nach einer Nachfrage ändern. Ebenso wird das Projekt deaktiviert, falls eine nicht unterstützte Zielplattform ausgewählt wird.



2. Rechtsklick auf das TwinCAT 3 C++ Projekt und Auswahl von „Build“ oder „Rebuild“



- ⇒ Wenn der Code korrekt erstellt wurde (d.h. keine Syntaxfehler), dann muss das Compiler-Ausgabefenster so aussehen:

```
Output
Show output from: Build
=====
1> Build succeeded.
1> Time Elapsed 00:00:03.61
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
```

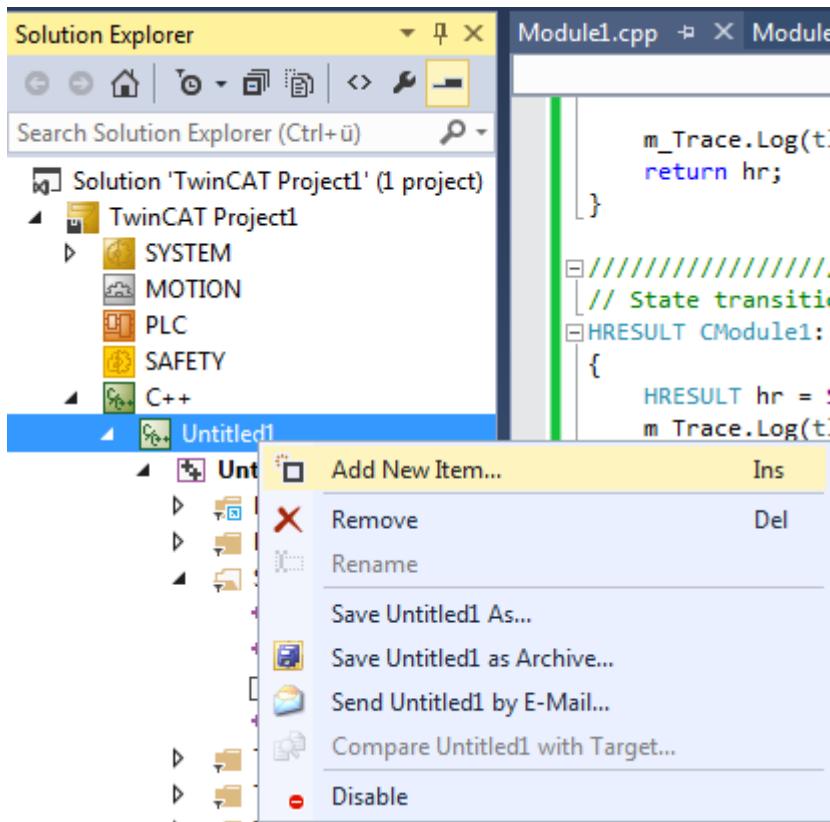
- ⇒ Nach erfolgreicher Kompilation/Erstellung wird das neue TwinCAT C++ Modul im Projektverzeichnis im Unterordner „\_Deployment“ Zielplattform-spezifisch bereitgestellt.

## 9.5 TwinCAT 3 C++ Modulinstanz erstellen

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

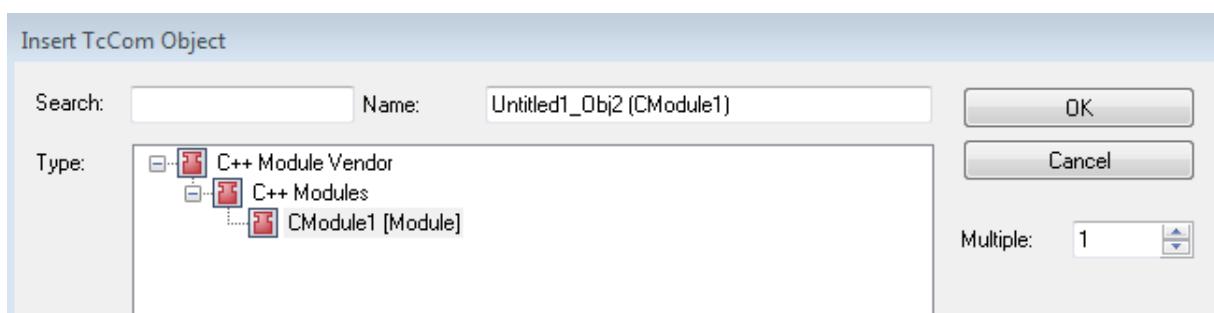
**Nach Erstellen eines TwinCAT C++ Moduls, Knoten „C++ - Configuration“ öffnen und dann diesen Schritten folgen, um eine Instanz zu erstellen.**

1. Rechtsklick auf das C++ Modul (in diesem Fall „Untitled1“) und Auswahl von „Add New Item...“



⇒ Alle bestehenden C++ Module werden aufgelistet.

2. Wählen Sie ein C++ Modul aus. Es kann die Standardbezeichnung verwendet oder optional ein neuer Instanzennname eingegeben und mit „OK“ bestätigt werden (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“)

```

Solution Explorer Untitled1Services.h Module1.cpp Module1.h
Search Solution Explorer (Ctrl+ü) [x] [x] [x]
Solution 'TwinCAT Project1' (1 project)
  ▾ TwinCAT Project1
    ▾ SYSTEM
    ▾ MOTION
    ▾ PLC
    ▾ SAFETY
  ▾ C++
    ▾ Untitled1
      ▾ Untitled1 Project
        ▾ External Dependencies
      ▾ Header Files
        Module1.h
        Resource.h
        TcPch.h
        Untitled1ClassFa
        Untitled1Interface
      ▾ Untitled1Service
  ▾ Source Files
  100 %
  
```

```

} Module1Parameter, *PModule1Parameter;

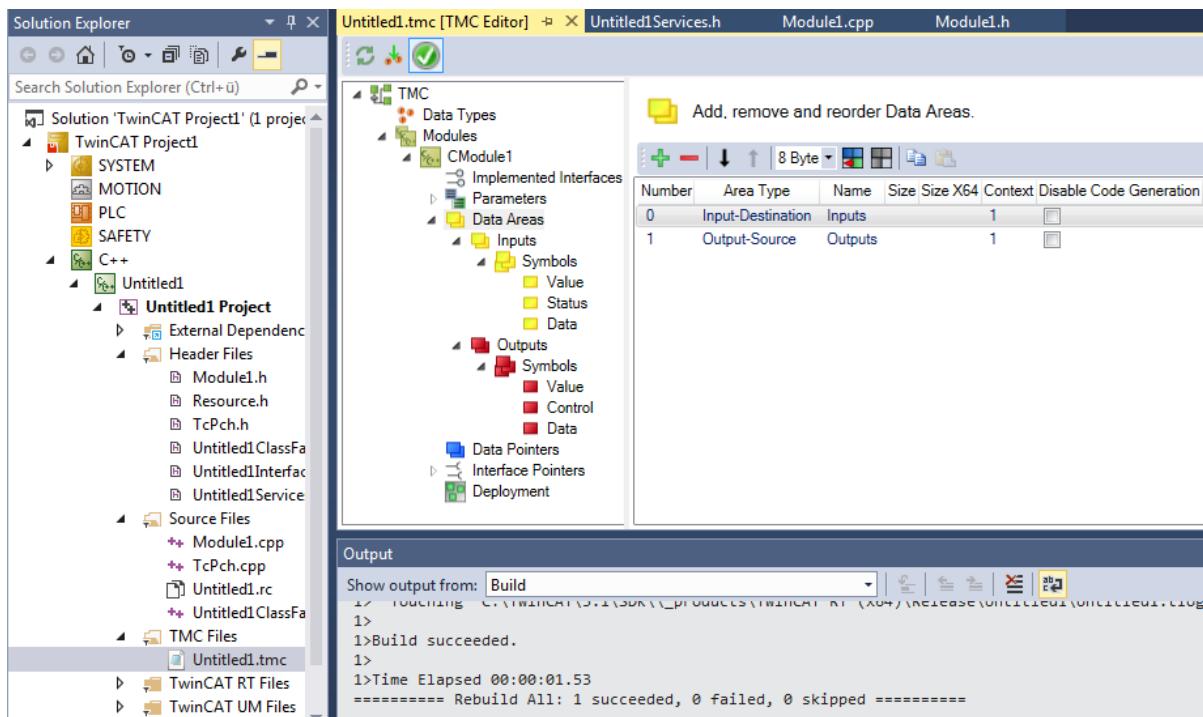
typedef struct _Module1Inputs
{
    ULONG Value;
    ULONG Status;
    ULONG Data;
} Module1Inputs, *PModule1Inputs;

typedef struct _Module1Outputs
{
    ULONG Value;
    ULONG Control;
    ULONG Data;
} Module1Outputs, *PModule1Outputs;

///</AutoGeneratedContent>

///<AutoGeneratedContent id="DataAreaIDs">
  
```

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



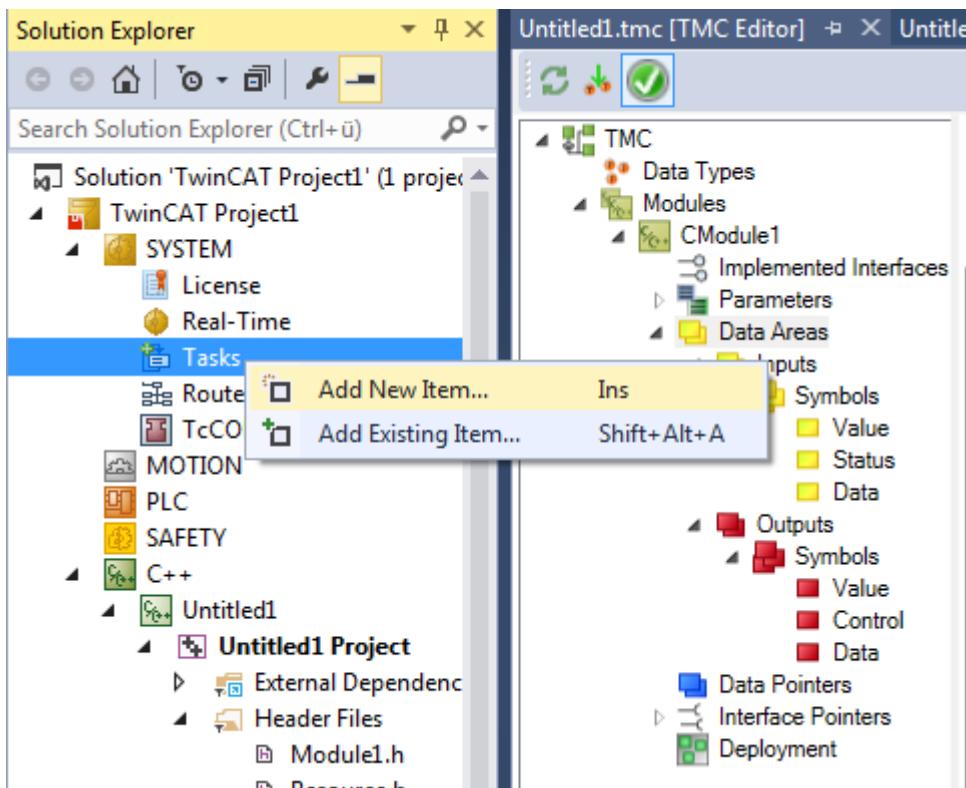
## 9.6 TwinCAT Task erstellen und auf Modulinstanz anwenden

Diese Seite beschreibt die Verknüpfung von Modulinstanz zu einem 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 kein neuer Task konfiguriert werden.

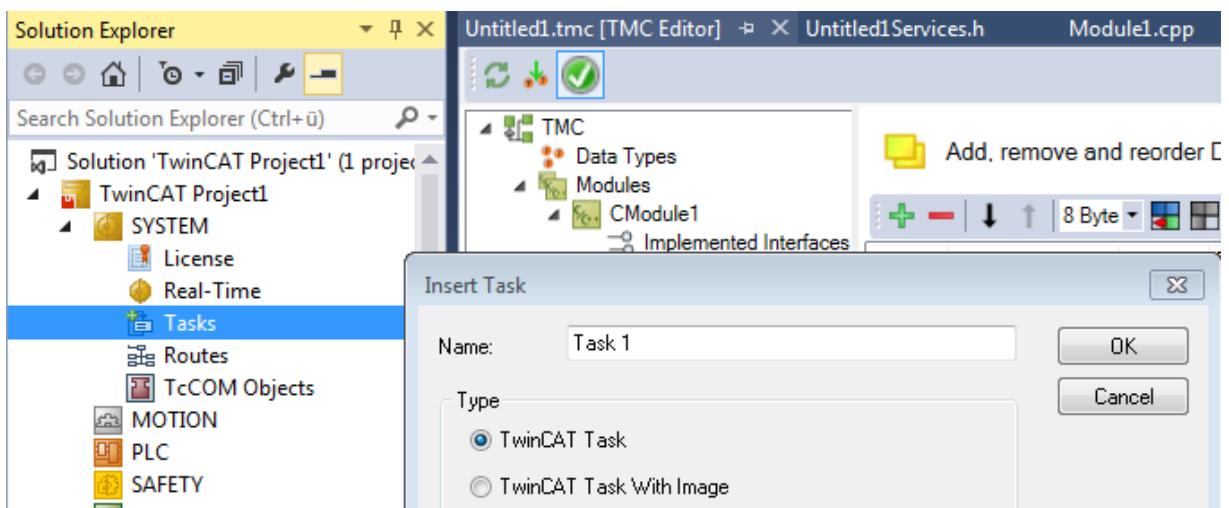
## TwinCAT 3 Task erstellen

- „System“ öffnen, Rechtsklick auf „Tasks“ und „Add New Item...“ auswählen.



- Eindeutigen Namen für den Task eingeben (oder Standard beibehalten).

In diesem Beispiel wird die I/O-Abildschnittstelle durch eine C++ Modulinstanz bereitgestellt, so dass für das Auslösen der Ausführung der C++ Modulinstanz kein Abbild (Image) an der Task nötig ist.

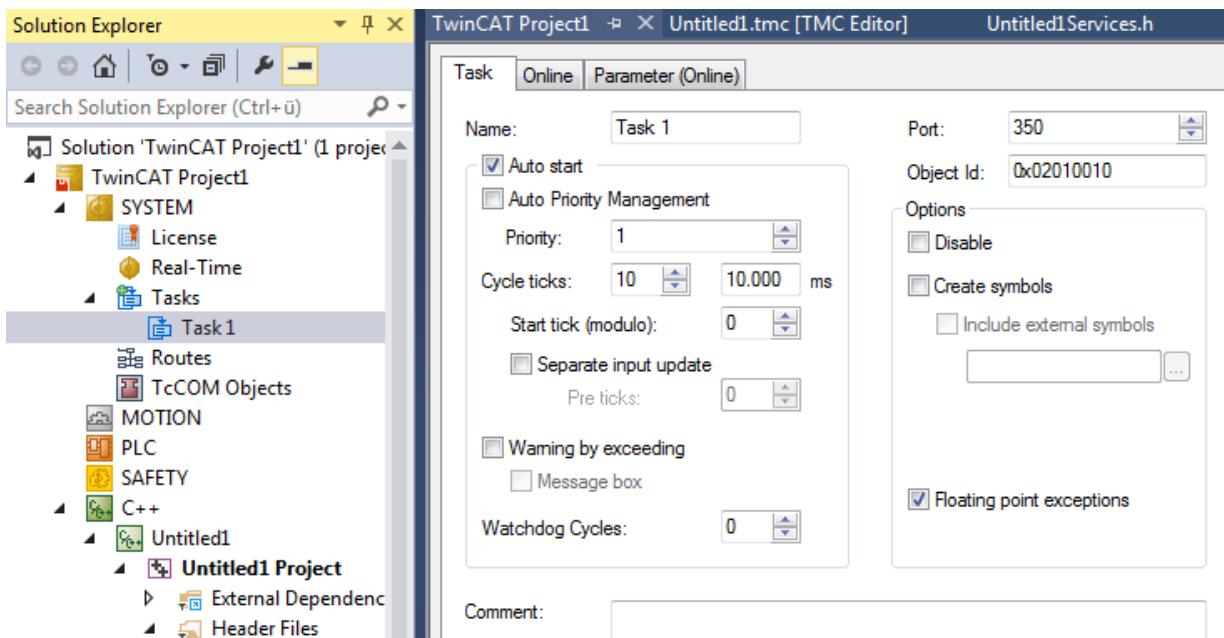


⇒ Der neue Task mit Namen „Task 1“ ist angelegt.

- Nun kann der Task konfiguriert werden, hierfür Doppelklick auf den Task.

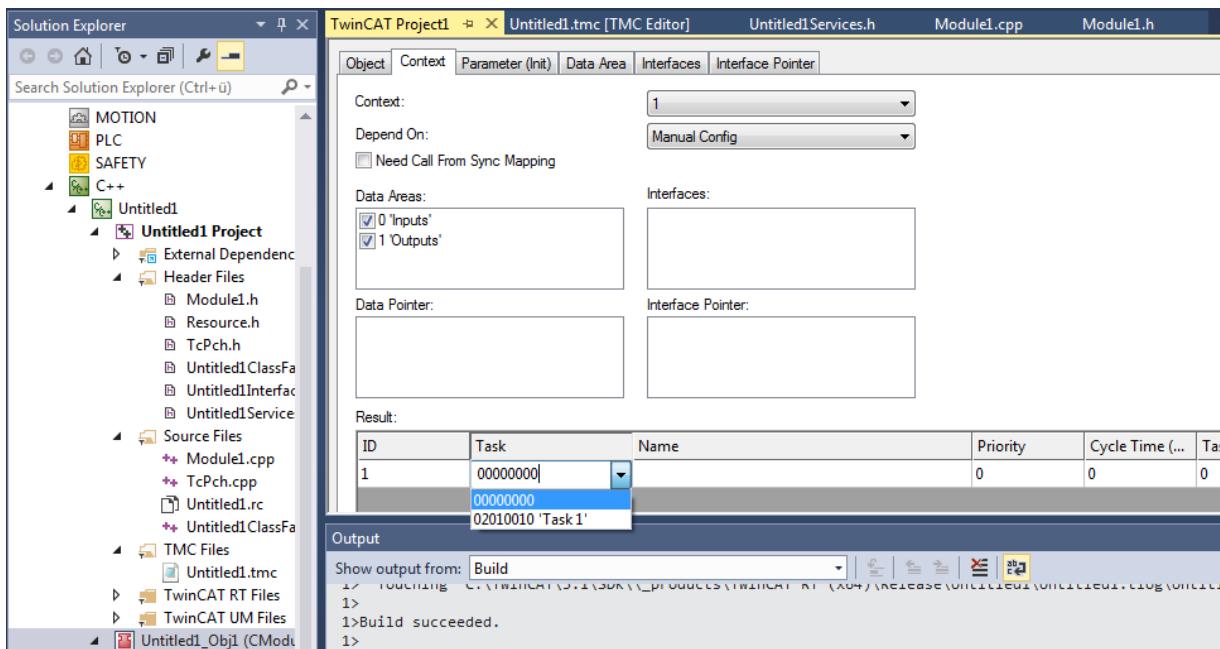
Die wichtigsten Parameter sind „Auto start“ und „Priority“:

„Auto start“ muss aktiviert werden, um einen zyklisch auszuführenden 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 dem Task aufgerufen wird

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

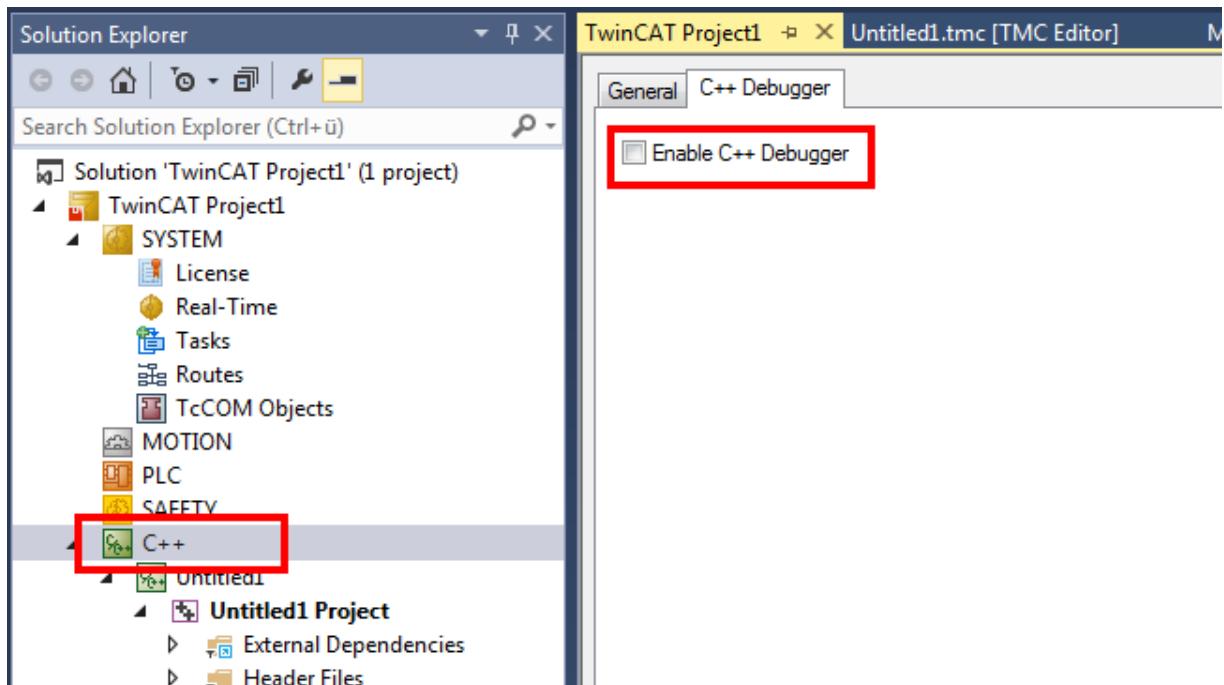


⇒ Mit Abschluss dieses Schritts ist der „Interface Pointer“ als „CyclicCaller“ konfiguriert. Die Konfiguration ist jetzt abgeschlossen!

## 9.7 TwinCAT 3 C++ Debugger aktivieren

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

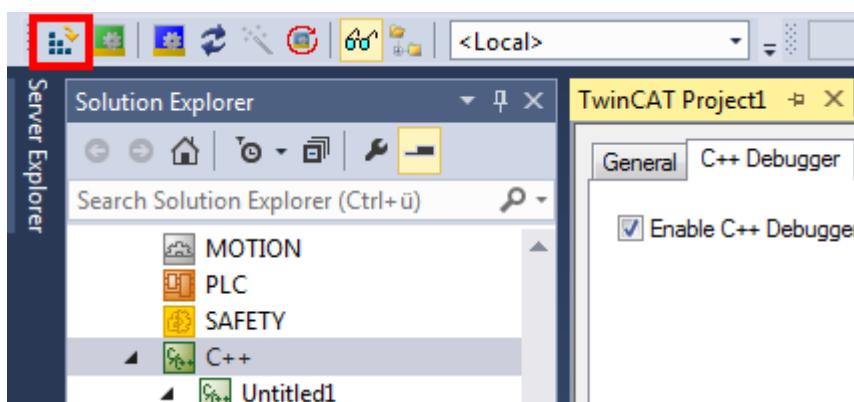
- Auf dem C++ Knoten der Solution Registerkarte „C++ Debugger“ wählen und „Enable C++ Debugger“ auswählen und „Enable C++ Debugger“ anschalten



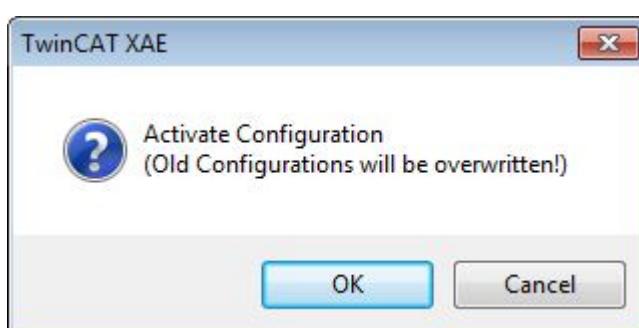
## 9.8 TwinCAT 3 Projekt aktivieren

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

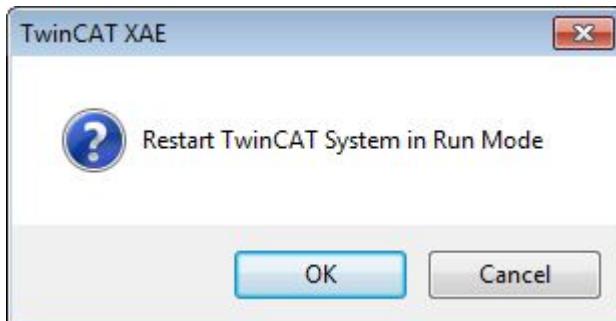
- Klicken Sie auf das Symbol „Activate Configuration“ – alle benötigten Dateien für das TwinCAT Projekt werden auf das Zielsystem übertragen:



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



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.



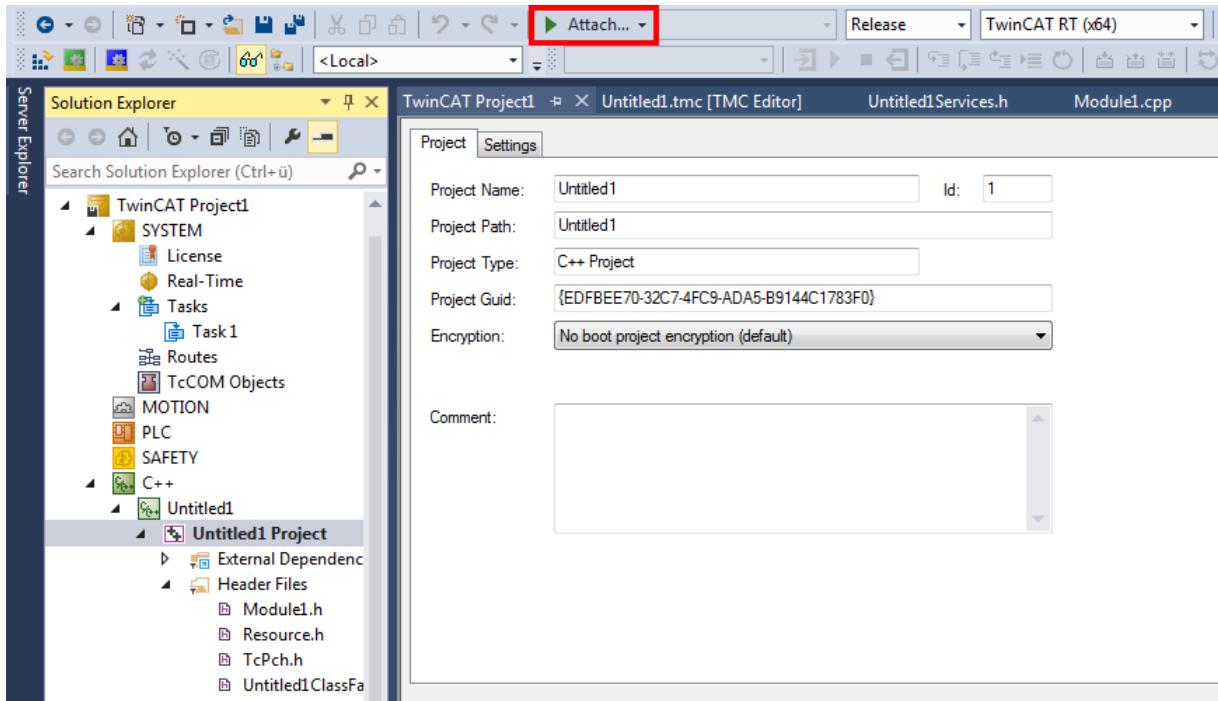
## 9.9 TwinCAT 3 C++ Projekt debuggen

Dieser Artikel beschreibt das Debuggen des TwinCAT 3 C++ Beispielprojekts.

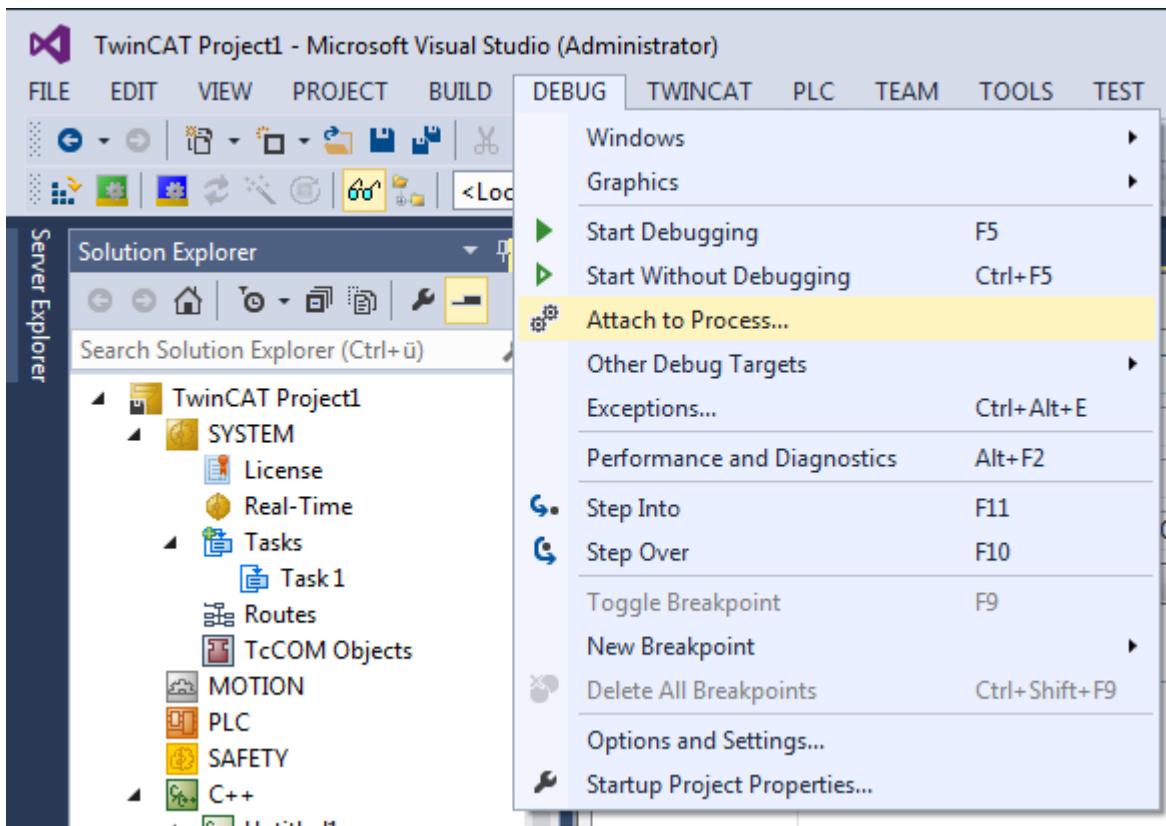
### An C++ Laufzeit anhängen

Nachdem in dem TwinCAT Projekt das C++ Debugging angeschaltet und das gesamte Projekt aktiviert wurde, kann das TwinCAT Engineering (XAE) nun genutzt werden, um sich mit dem Zielsystem zu verbinden und zu debuggen.

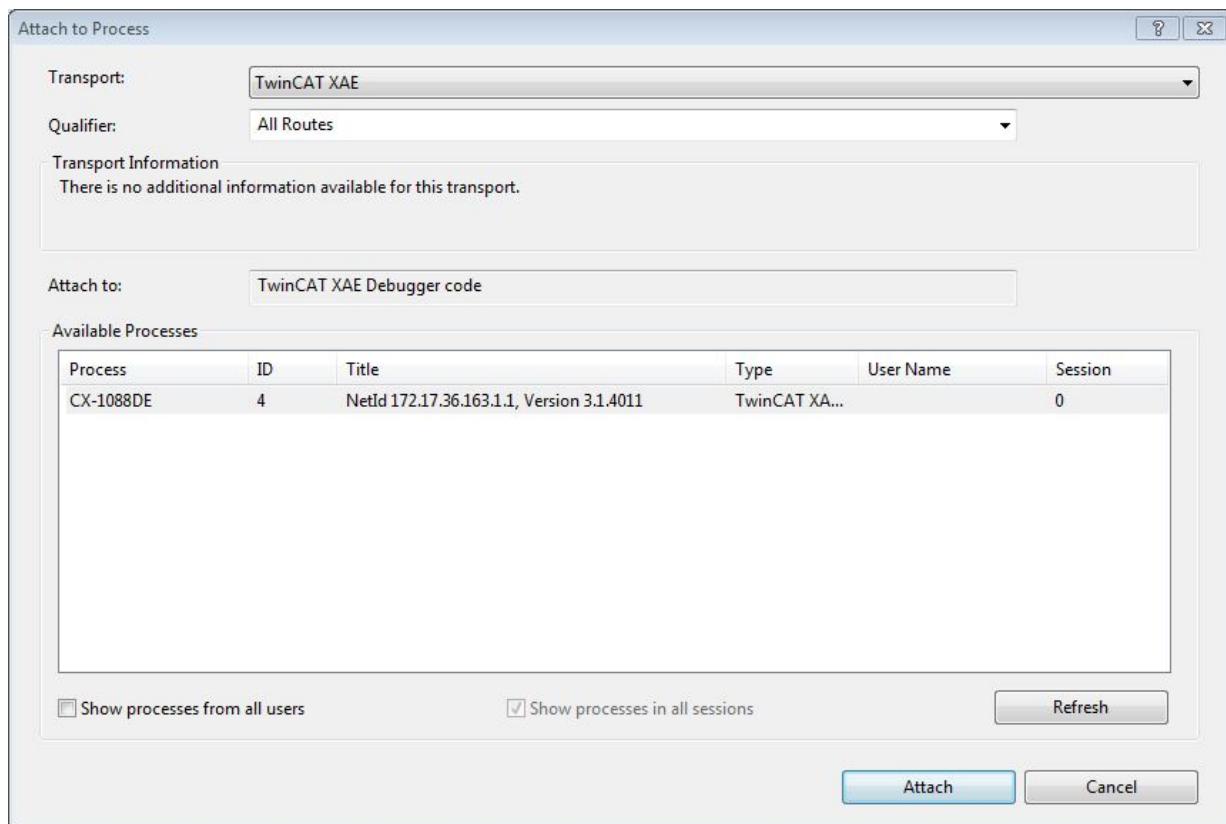
1. A) Klicken Sie auf den aus dem Visual Studio bekannten „Attach...“ Button, um sich mit dem TwinCAT Debugger des Zielsystems zu verbinden:



- B) Alternativ wählen Sie „Debug“ -> „Attach to process...“ in der Visual Studio Umgebung:



2. Für den Transport wählen Sie nicht die „Default“-Einstellung von Visual Studio, sondern „TwinCAT XAE“. Zielsystem (oder „All Routes“) als Qualifizierer und verbinden Sie per „Attach“.

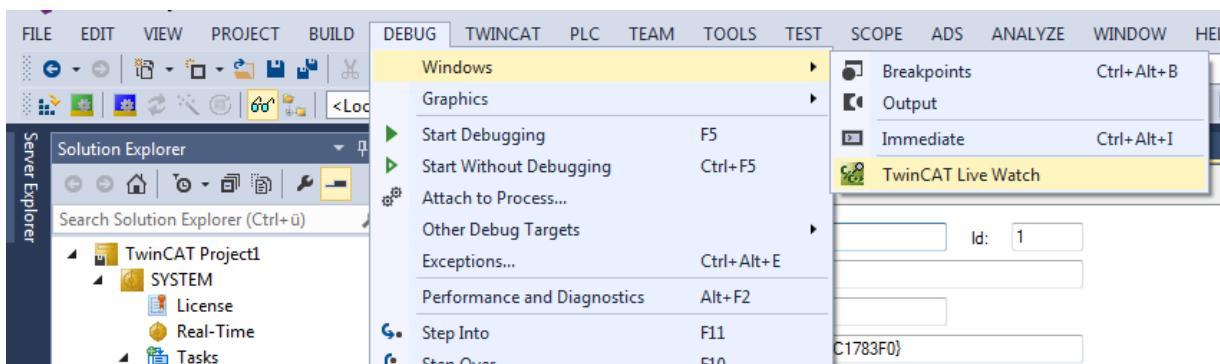


### C++ Modul Membervariablen überwachen (ohne Haltepunkte)

Es steht der „normale“ Debugging-Mechanismus von Visual Studio zur Verfügung - mit Haltepunkt setzen, schrittweise Ausführung, usw. Dessen Verwendung hängt vom zu überwachenden Prozess ab:  
Wenn TwinCAT auf einer realen Maschine mit Achsenbewegungen läuft, wird der Nutzer nur zur Überwachung von Variablen wahrscheinlich keinen Haltepunkt setzen wollen. Bei Erreichen eines Haltepunktes würde die Ausführung des Tasks angehalten und je nach Konfiguration die Achse sofort stoppen oder, was möglicherweise noch schlimmer wäre, sich unkontrolliert weiterbewegen - eine sehr unglückliche Situation.

**TwinCAT 3 bietet deswegen die Möglichkeit, Prozessvariablen ohne das Setzen von Haltepunkten zu überwachen:**

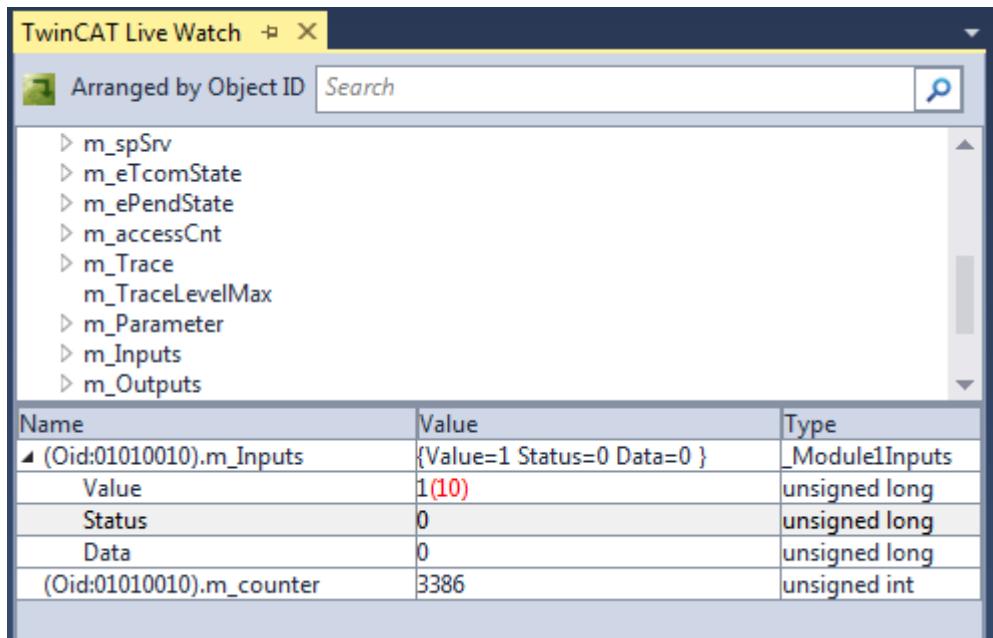
1. Wählen Sie „Debug“ -> „Windows“ -> „TwinCAT Live Watch“



⇒ Die „TwinCAT Live Watch“-Fenster zeigen eine Liste aller Variablen im Modul. Per Drag & Drop in die Überwachungsliste (watch list) gezogene Variablen werden ohne das Setzen eines Haltepunktes überwacht.

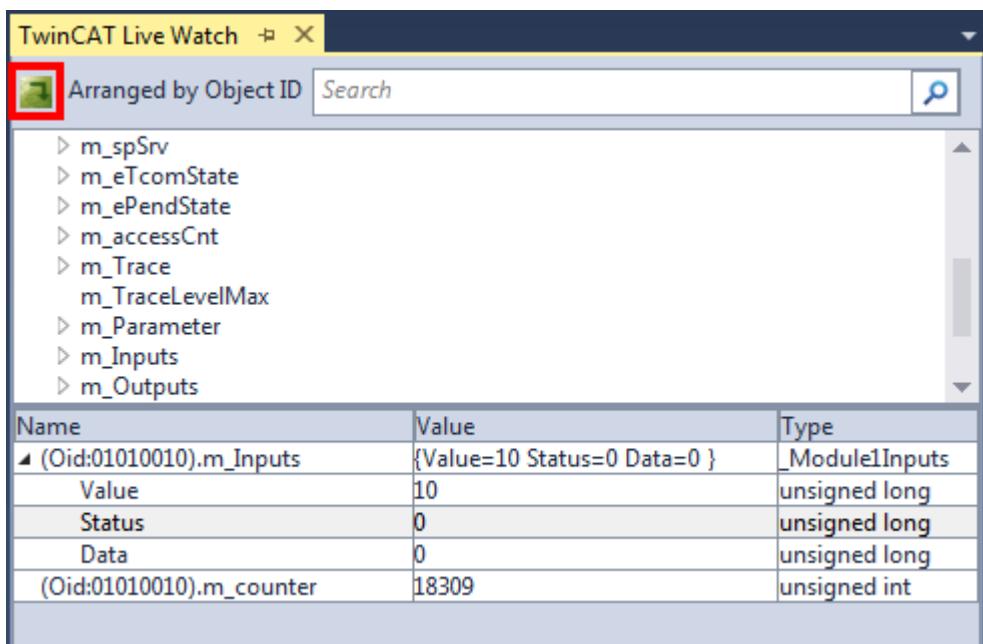
2. Um den Wert einer Überwachungsvariablen zu verändern, einfach einen neuen Wert eingeben.

⇒ Der neue Wert wird in rot und in Klammern angezeigt.



3. Klicken Sie auf das grüne Symbol

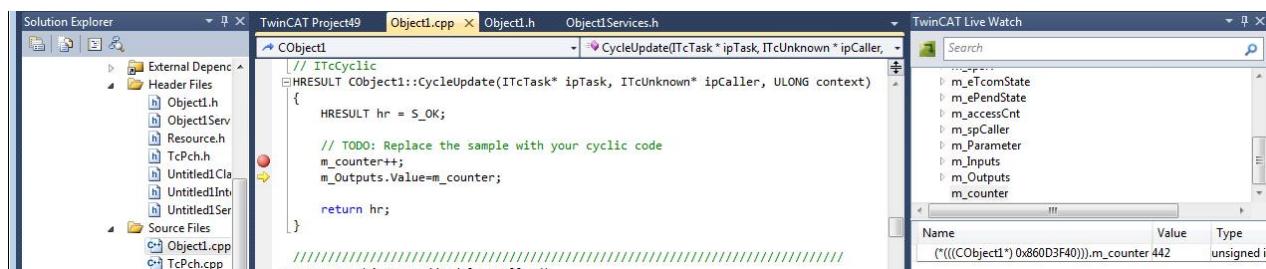
⇒ der neue Wert wird in den Prozess geschrieben.



### Haltepunkte setzen

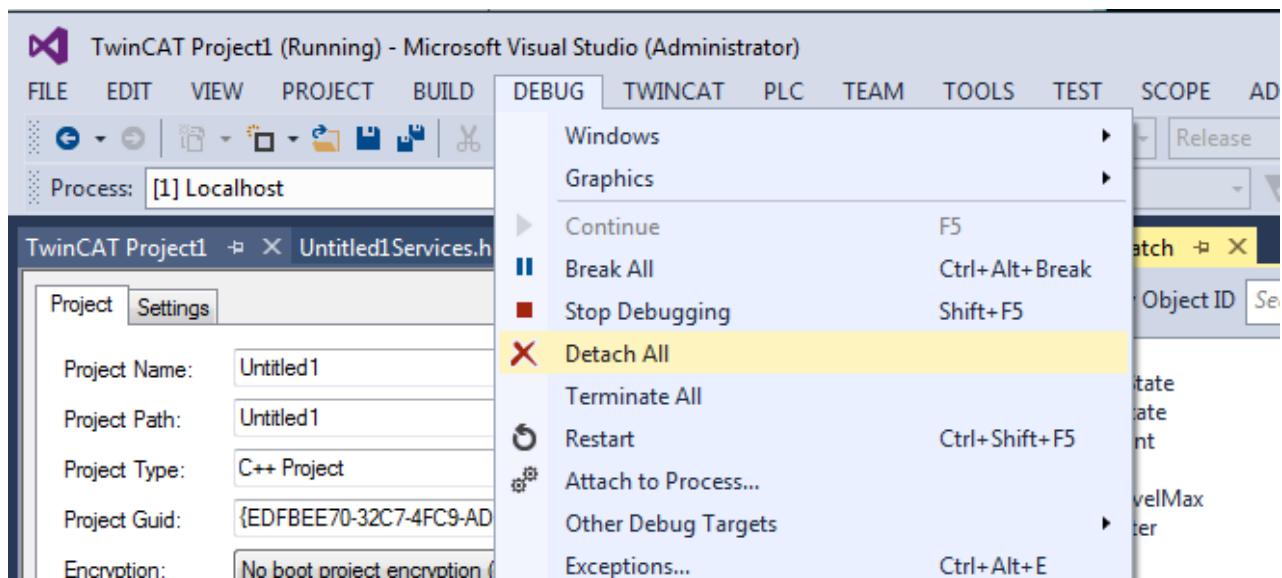
Das Setzen von Haltepunkten auf herkömmliche Art und Weise ist ebenfalls möglich.

**HINWEIS! Beachten Sie die Folgen für die Maschine!**



## Debugger vom Prozess trennen

Klicken Sie auf „Debug“ -> „Detach All“.



## Sehen Sie dazu auch

- 📄 TwinCAT 3 C++ Projekt debuggen [▶ 62]
- 📄 TwinCAT 3 C++ Projekt debuggen [▶ 64]
- 📄 TwinCAT 3 C++ Projekt debuggen [▶ 65]
- 📄 TwinCAT 3 C++ Projekt debuggen [▶ 66]

## 10 Debuggen

TwinCAT C++ bietet verschiedene Mechanismen für das Debuggen der unter Echtzeitbedingungen laufenden TwinCAT C++ Module.

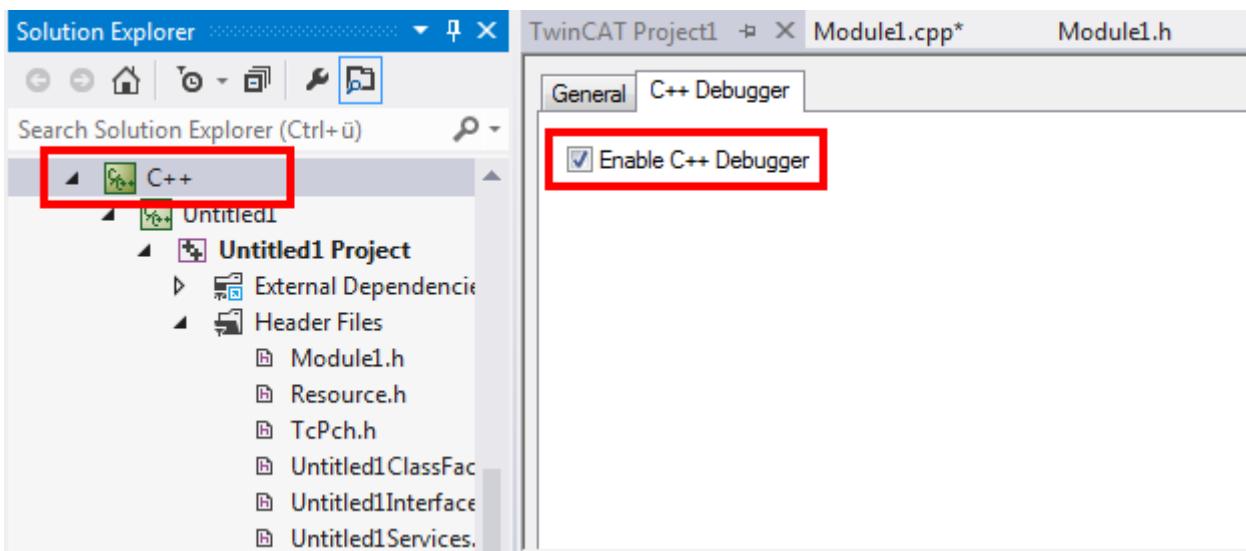
Die meisten von ihnen entsprechen denjenigen, die von der normalen C++ Entwicklungsumgebung bekannt sind. Die Automatisierungswelt benötigt zusätzliche, leicht abweichende Debugging-Mechanismen, die hier dokumentiert sind.

Zusätzliche 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 kann über den C++ Knoten der Solution konfiguriert werden:

Doppelklick auf den C++ Knoten und Wechsel zum „C++ Debugger“ Reiter, um auf das Prüfkästchen zuzugreifen.



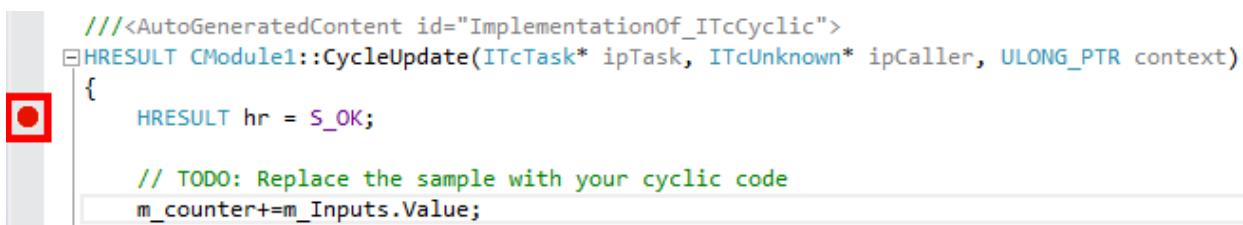
Für jegliches Debuggen in TwinCAT C++ muss das TwinCAT Engineering mit dem Laufzeitsystem (XAR) über die „TwinCAT Debugger“ Schaltfläche verbunden sein:



### 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.



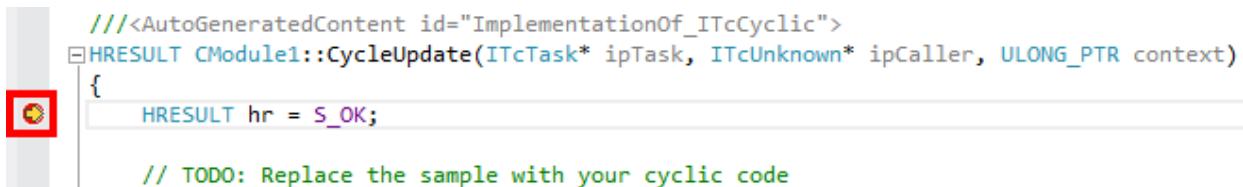
```

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

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

```

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



```

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

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

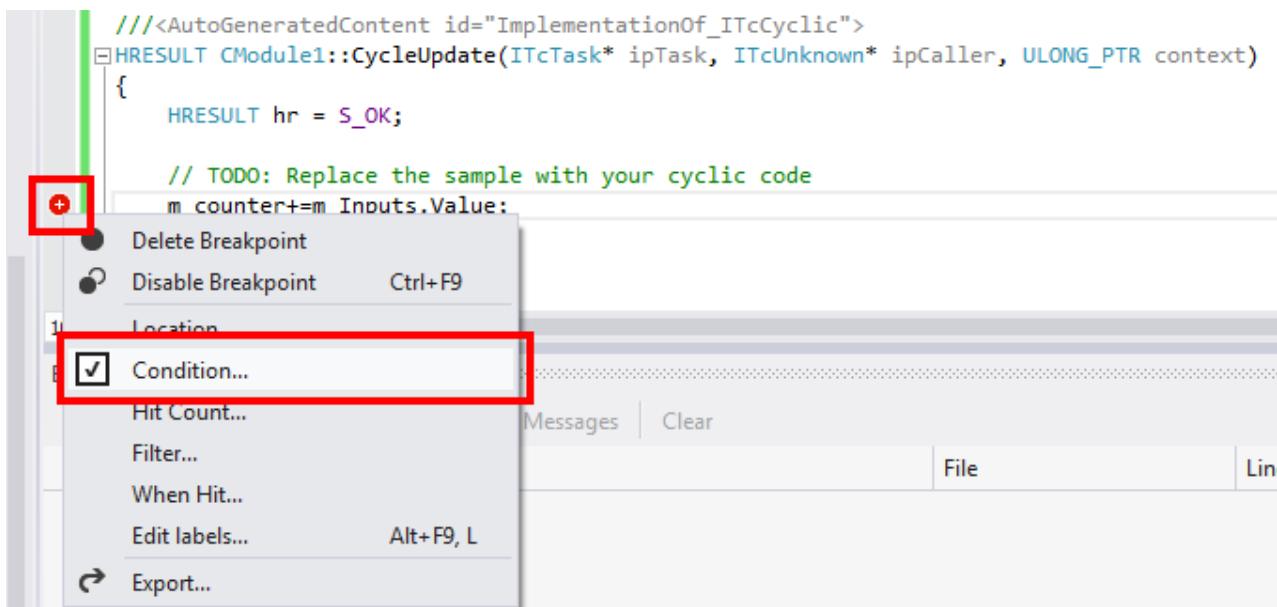
```

Durch Drücken auf „Step Over“ (Debug-Menü, Toolbar oder Hotkey F10) kann man den Code schrittweise ausführen. Auch stehen die von Visual Studio bekannten „Step in“ (F11) und „Step out“ (Shift+F11) zur Verfügung.

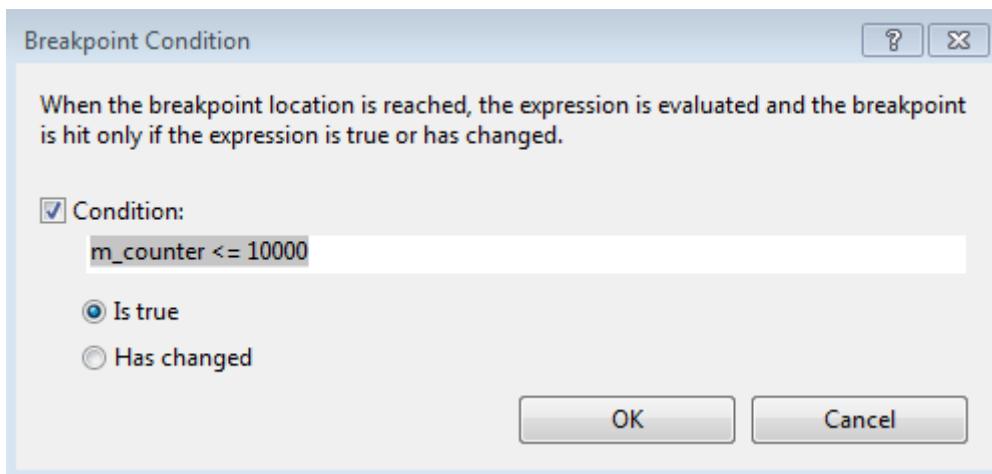
### Bedingte Haltepunkte

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.



Wählen Sie „Condition...“ um das Bedingungsfenster zu öffnen:



Einzelheiten zu den Bedingungen und wie diese zu formulieren sind, finden Sie [hier \[► 70\]](#).

## Live Watch

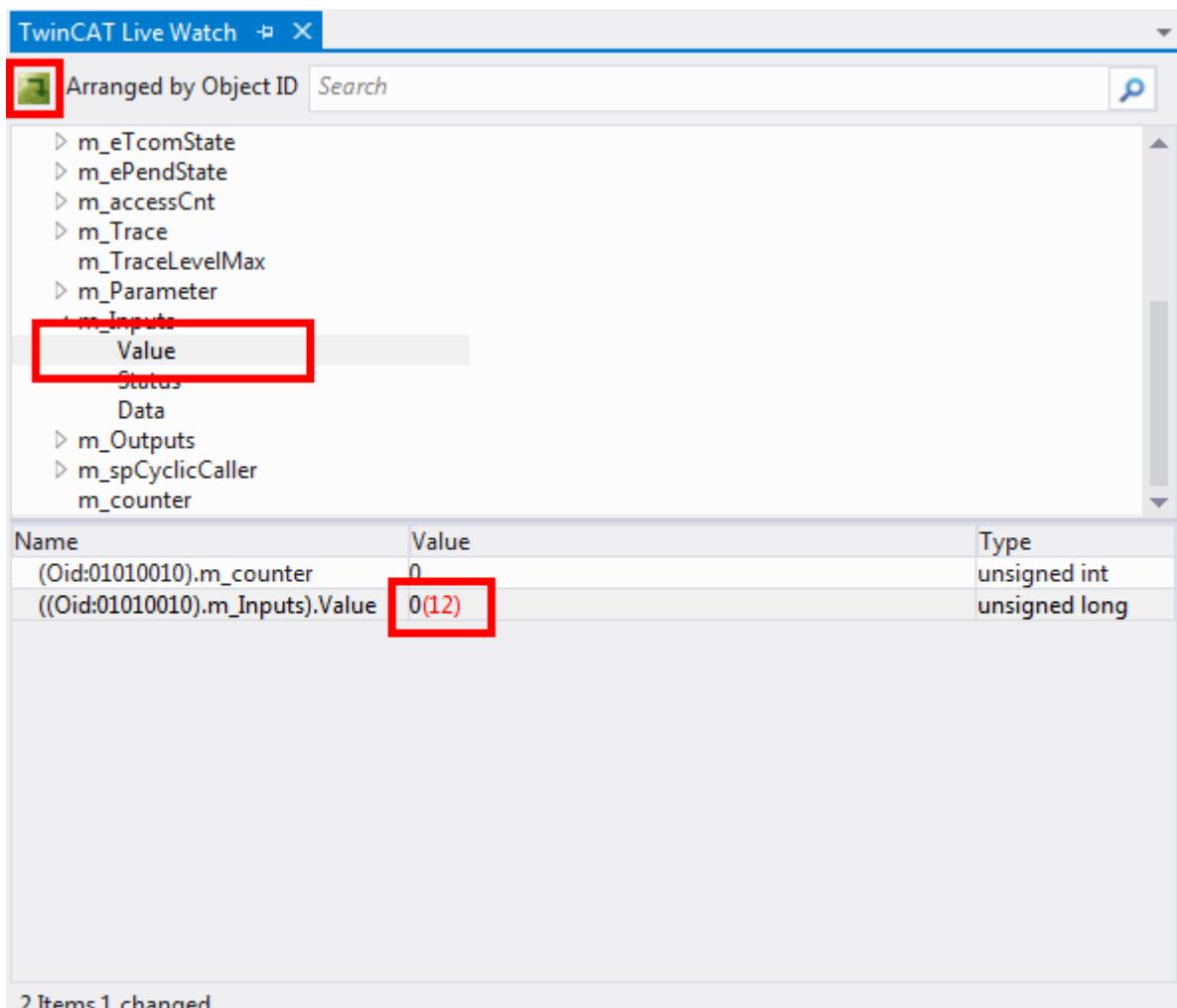
Bei dem Engineering und der Entwicklung von Maschinen ist es nicht immer ratsam, das System über einen Haltepunkt anzuhalten, weil dies sich auf das Verhalten auswirken wird.

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, zunächst eine Verbindung mit dem Echtzeitsystem herstellen (drücken der „TwinCAT Debugger“-Schaltfläche), woraufhin Visual Studio zur Debug-Ansicht wechselt, ansonsten können keine Daten bereitgestellt werden.



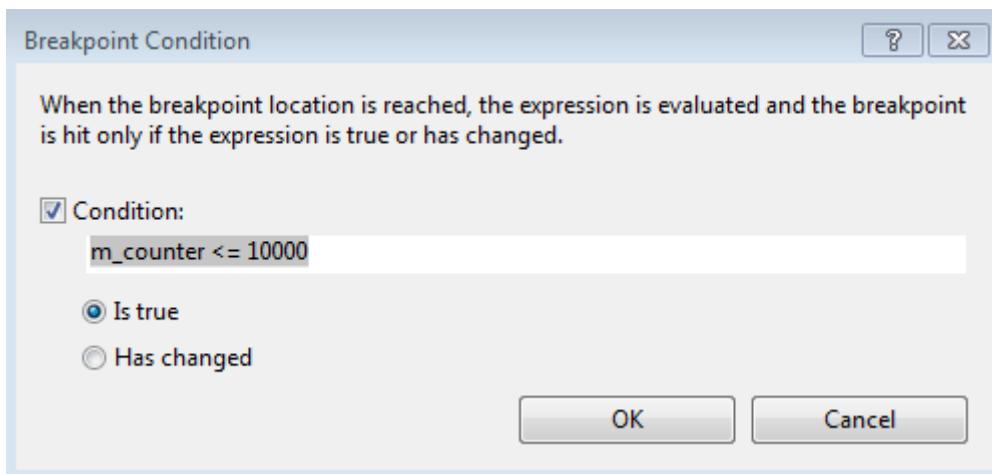
Das TwinCAT Live Watch Fenster ist in zwei Bereiche unterteilt.

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

Sie können diese Werte mittels Klicken auf dem Wert im „Value“-Feld bearbeiten. Der neue Wert wird zwischen Klammern gesetzt und rot markiert. Um den Wert nun zu schreiben, drücken Sie auf das Symbol oben links.

## 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.

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 „Maxterms“ gebildet:

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

wobei jedes Maxterm eine Kombination von `||` verbundenen Bedingungen darstellt:

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

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

Für die Bestimmung der verfügbaren Variablen siehe 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
```

Weitere Anmerkungen:

- 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 des aufrufenden Tasks 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
```

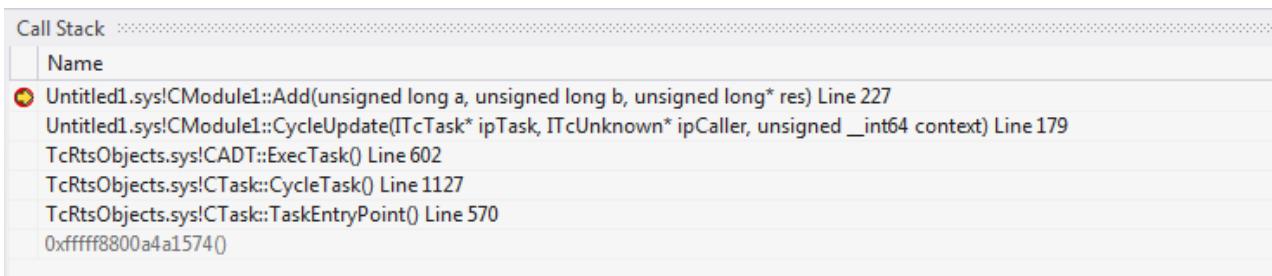
```
m_counter && m_counter2
```

## 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, wenn ein Breakpoint erreicht wurde, wird durch das Toolwindow „Call Stack“ dargestellt.



The screenshot shows the 'Call Stack' tool window in Visual Studio. The title bar says 'Call Stack'. The main area has a header 'Name' with a small red circle icon. Below it is a list of function calls:

- Untitled1.sys!CModule1::Add(unsigned long a, unsigned long b, unsigned long\* res) Line 227
- Untitled1.sys!CModule1::CycleUpdate(ITcTask\* ipTask, ITcUnknown\* ipCaller, unsigned \_\_int64 context) Line 179
- TcRtsObjects.sys!CADT::ExecTask() Line 602
- TcRtsObjects.sys!CTask::CycleTask() Line 1127
- TcRtsObjects.sys!CTask::TaskEntryPoint() Line 570
- 0xfffff8800a4a1574()

### Autos / Locals und Watch

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

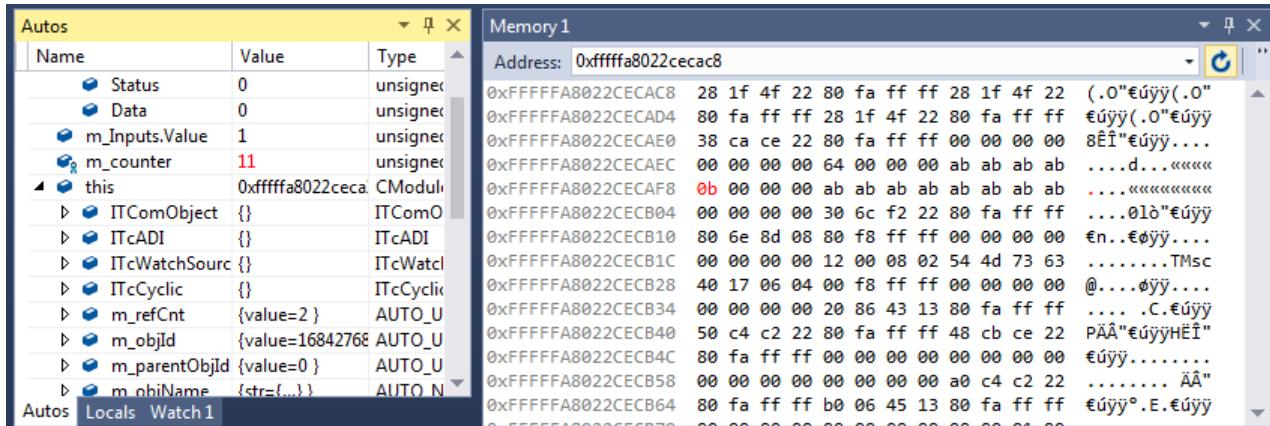
Locals		
Name	Value	Type
this	0xfffffa801a1d57b0	CModule1*
ITComObject	{}	ITComObject
ITcADI	{}	ITcADI
ITcWatchSource	{}	ITcWatchSource
ITcCyclic	{}	ITcCyclic
Calc	{}	Calc
m_refCnt	{value=2 }	AUTO ULONG
m_objId	{value=16842768 }	AUTO ULONG
m_parentObjId	{value=0 }	AUTO ULONG
m_objName	{str={...} }	AUTO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

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

Watch 1		
Name	Value	
a	123	
b	108117	
*(res)	108117	

## Memory View

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



# 11 Assistenten

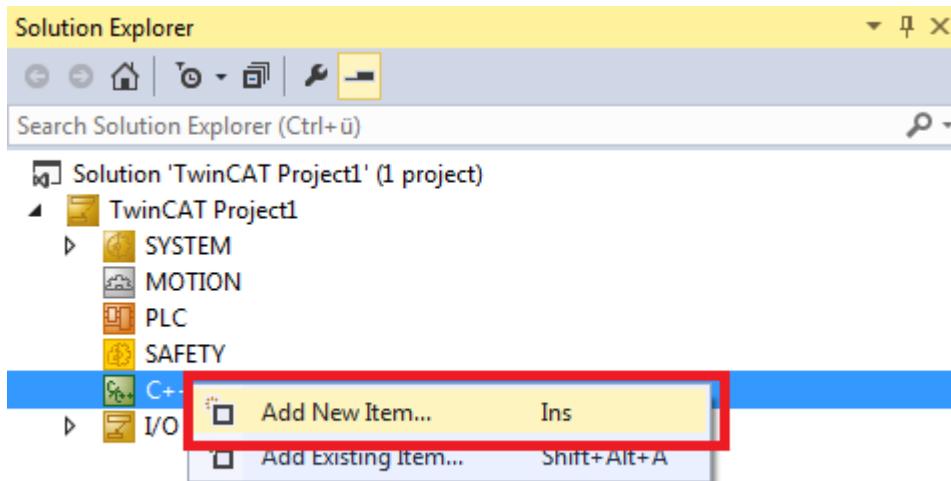
Um den Einstieg in das Engineering des TwinCAT C++ Systems zu vereinfachen, stehen Assistenten zur Verfügung.

- Der [TwinCAT Projekt-Assistent](#) [► 75] erstellt ein TwinCAT C++ Projekt. Im Falle von Treiber-Projekten wird anschließend der TwinCAT Class Wizard gestartet.
- Der [TwinCAT Module Class Wizard](#) [► 76] 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](#) [► 79] (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](#) [► 123] konfiguriert werden können.

## 11.1 TwinCAT C++ Projekt-Assistent

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

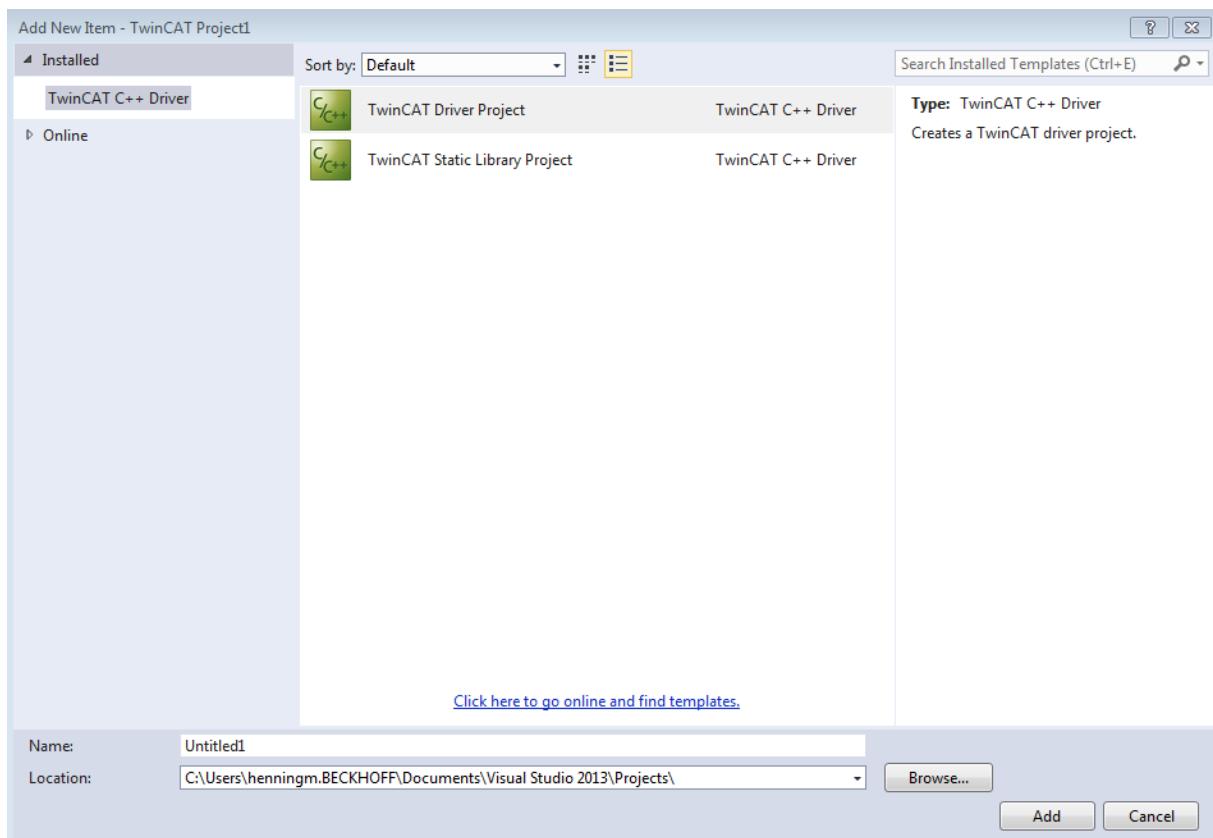
1. Rechtsklicken auf C++ Symbol und „Add new Item...“ auswählen, um den C++ Projektassistenten zu starten.



TwinCAT bietet zwei C++ Projekte an:

Treiber-Projekt: Projekte, die ein oder mehrere ausführbare Module beinhalten

Statische Bibliothek: Projekte mit C++ Funktionen, die von (verschiedenen) TwinCAT C++ Treibern verwendet werden.

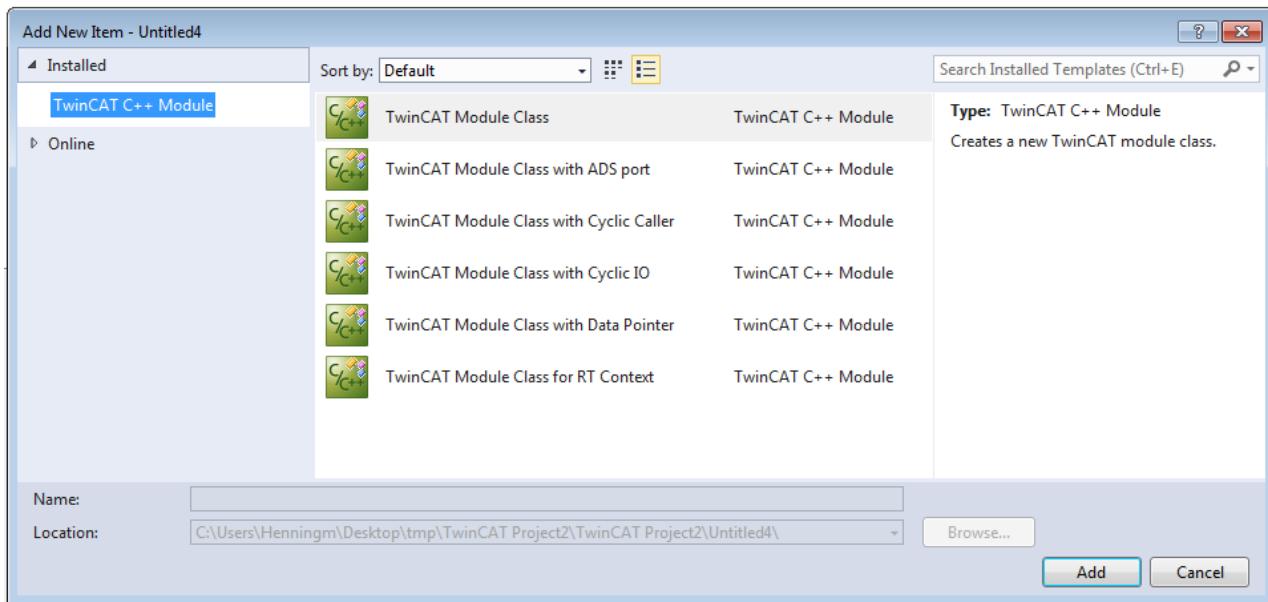


2. Wählen Sie eine der Projektvorlagen, geben einen Namen und einen Speicherort an.
  - ⇒ Das TwinCAT C++ Projekt wird erstellt
  - ⇒ Im Falle eines Treibers wird der TwinCAT C++ Klassenassistent [► 76] gestartet.

## 11.2 TwinCAT Module Klassenassistent

TwinCAT 3 bietet verschiedene Klassenvorlagen

- TwinCAT Modules Class
- TwinCAT Modules Class mit ADS Port
- TwinCAT Modules Class mit zyklischem Aufrufer
- TwinCAT Modules Class mit zyklischem Ein-/Ausgang
- TwinCAT Modules Class mit Datenzeiger
- TwinCAT Modules Class für Echtzeitkontext



## 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 \[▶ 240\]](#)

## 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 \[▶ 220\]](#), [ADS-Kommunikation \[▶ 175\]](#)

## TwinCAT Modules Class mit zyklischem Aufrufer

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

Dieses 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, welche die zyklisch aufrufende Schnittstelle implementiert und die 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 \[▶ 49\]](#), [Beispiel01 \[▶ 219\]](#)

## 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 \[▶ 123\]](#) konfiguriert.

Siehe [Beispiel10 \[▶ 239\]](#)

## TwinCAT Modules Class für Echtzeitkontext

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

Wie [hier \[▶ 38\]](#) 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) State machine können auch verwendet werden um direkt beim Starten von TC instanziert zu werden. In diesem Fall werden die Transitionen wie bei einem normalen Modul ausgeführt.

Das Beispiel TcCOM 03 [▶ 300] zeigt die Verwendung eines solchen Moduls.

## 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 mit deren Verwendung, bereitgestellte und implementierte Schnittstellen, sowie Datenbereiche und Datenzeiger.

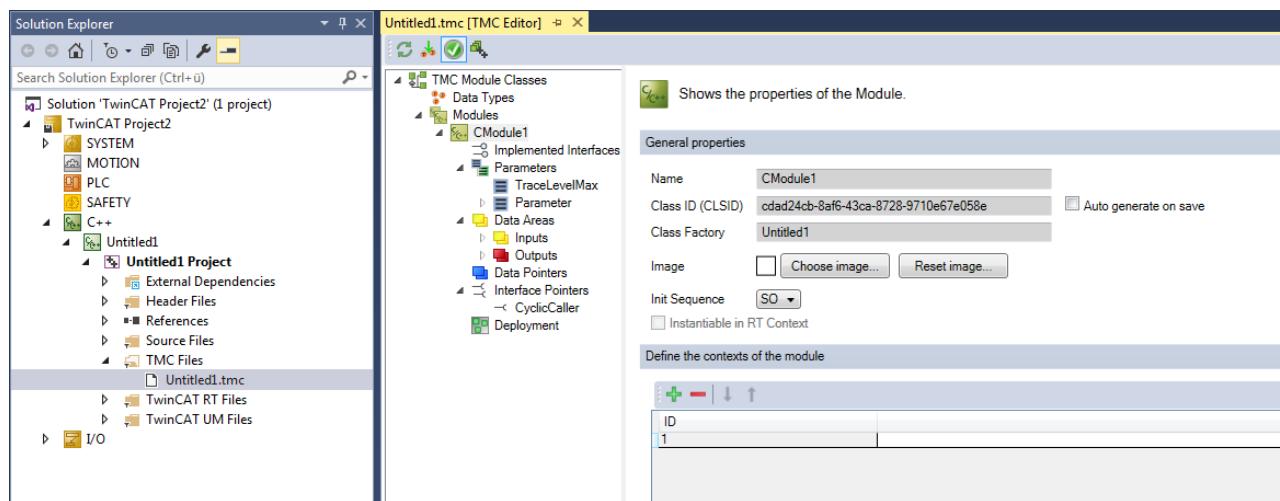
Kurz: Alles was von außen zu sehen ist, muss mit diesem Editor definiert werden.

Die Grundidee ist:

1. Mit dem „TMC Editor“ verändern Sie die Modul-Beschreibungsdatei (TMC-Datei). Diese enthält alle Informationen, die im TwinCAT System selber 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 mittels Doppelklick auf die TMC-Datei eines Moduls. Der grafische Editor wird geöffnet:



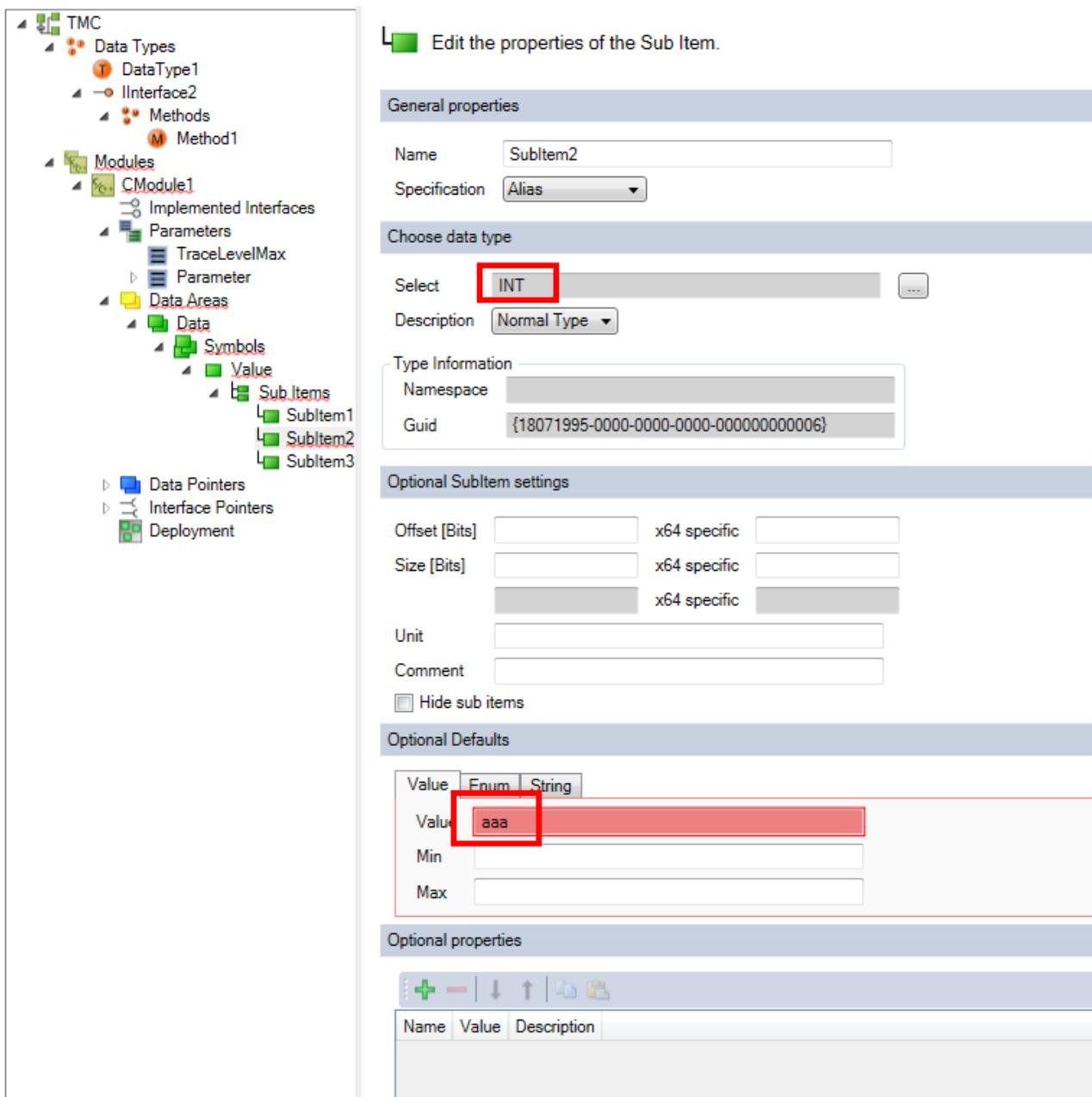
Funktionalitäten des TMC Editors sind:

- Symbole in den Datenbereichen, z.B. wie 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.

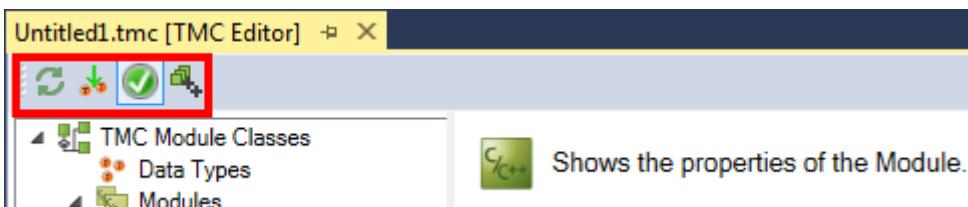
Zum Beispiel bei Problemen (Alignment, ungültige Standarddefinitionen, ...) innerhalb des TMC, wird der Nutzer über die Anzeige von roten Markierungen innerhalb des TMC-Baums zur entsprechenden Stelle geführt:



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

## Werkzeuge

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

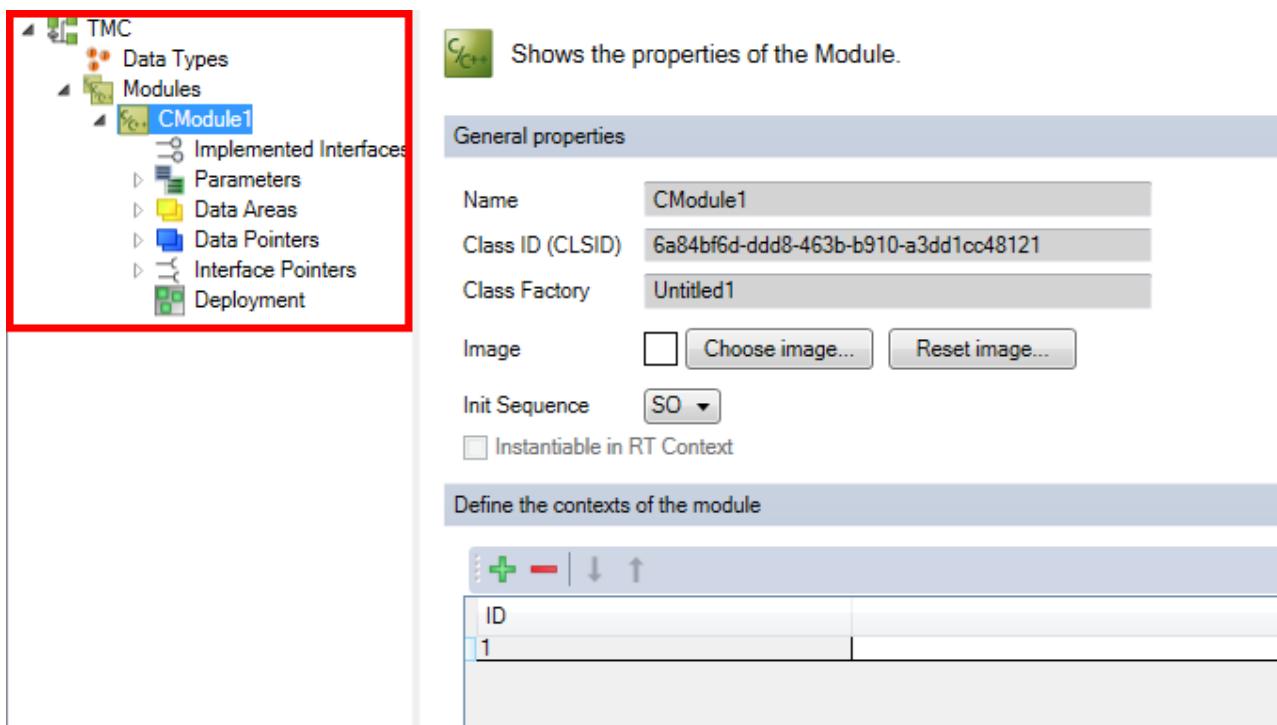


- Erneutes Laden der TMC Datei sowie der Typen aus dem Typsystem.
- Aktualisierung der überlagerten Datentypen
- An- / Ausschalten der Nutzer-Hilfen (vgl. oben)
- Start des „TwinCAT TMC Code Generator“:

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

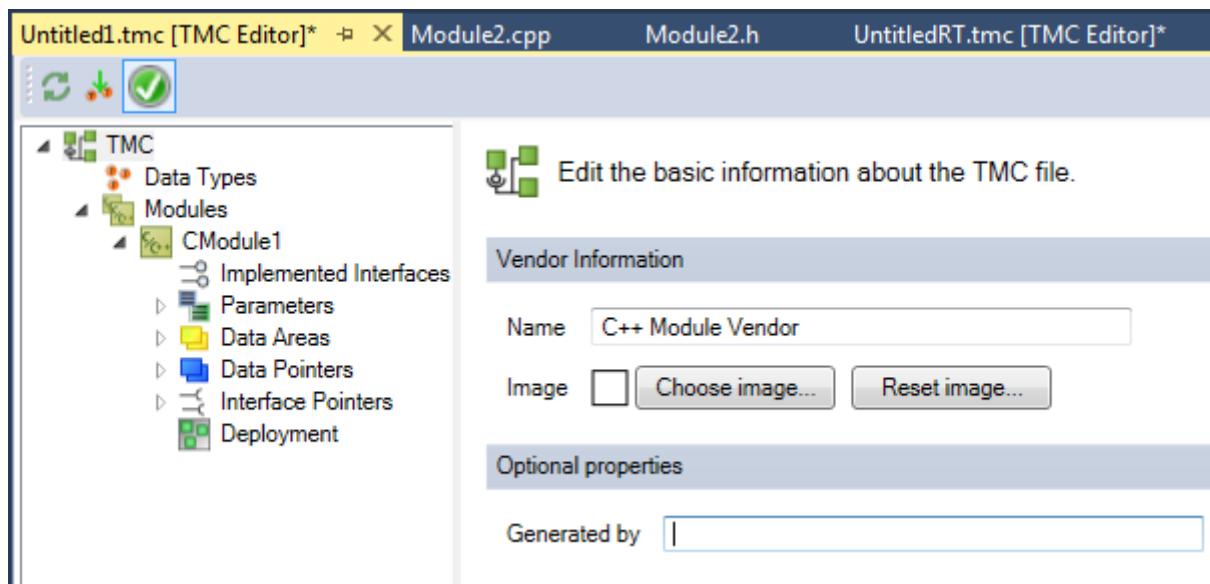
- [TMC \[► 82\]](#): Die grundlegenden Angaben des Herstellers des C++ Moduls bearbeiten und ein Bild hinzufügen
- [Data Types \[► 83\]](#): Datentypen hinzufügen, entfernen und neu ordnen
- [Modules \[► 100\]](#): Zeigt die Module des Treibers
- [Implemented Interfaces \[► 102\]](#): Zeigt die implementierten Schnittstellen des Moduls.
- [Parameters \[► 103\]](#): Ihre Parameter hinzufügen, entfernen und neu ordnen
  - [TraceLevelMax \[► 110\]](#): Parameter, der die Menge an protokollierten Nachrichten steuert - für (fast) jedes Modul vordefiniert.
- [Data Areas \[► 111\]](#): Datenbereiche hinzufügen, entfernen und neu ordnen
- [Data Pointers \[► 118\]](#): Datenzeiger hinzufügen, entfernen und neu ordnen
- [Interface Pointers \[► 120\]](#): Datenzeiger hinzufügen, entfernen und neu ordnen
- [Deployment \[► 121\]](#): Bestimmt die Dateien, die bereitgestellt werden

#### Sehen Sie dazu auch

□ Datentypen hinzufügen / bearbeiten / löschen [▶ 84]

## 11.3.2 Grundlegende Informationen

Grundlegende Informationen bezüglich der TMC-Datei



### Informationen zum Anbieter

**Name:** Modulnamen bearbeiten

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

**Reset image:** Modulbild auf Standardwert zurücksetzen

### 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 Felds Änderungen nicht mehr möglich sind (deaktiviert alle Bearbeitungsvorgänge) im TMC Editor.

## 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.

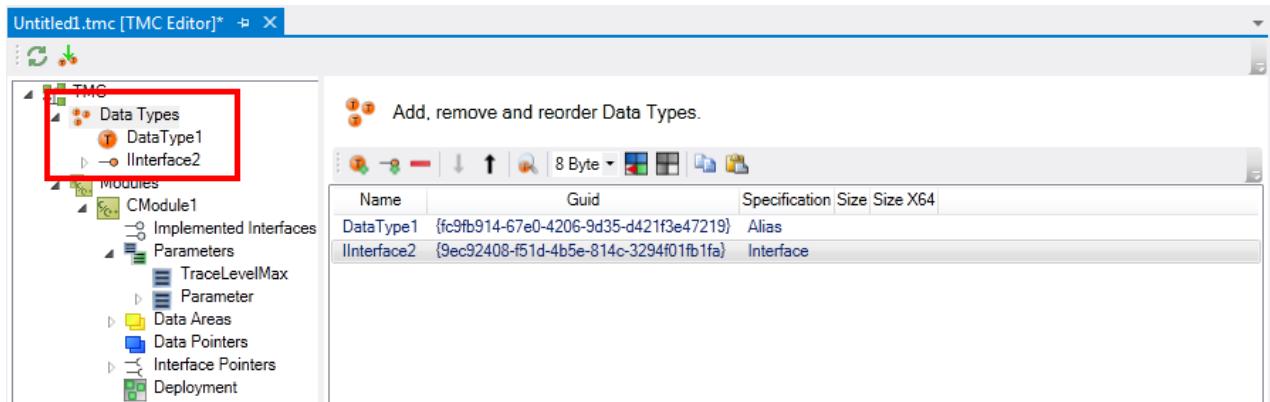
Das TwinCAT Engineering System (XAE) veröffentlicht diese Datentypen gegenüber allen anderen verschachtelten Projekten des TwinCAT Projekts, so dass diese auch z.B. in SPS-Projekten verwendet werden können (wie [hier \[▶ 240\]](#) beschrieben).

<span style="font-size: 2em;">i</span> <b>Hinweis</b>	<b>Namenskonflikt</b> Bitte verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen. Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.
--	---

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
	Löscht den ausgewählten Typ
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben
	Sucht nicht verwendete Typen
	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:** Benutzerdefinierte 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

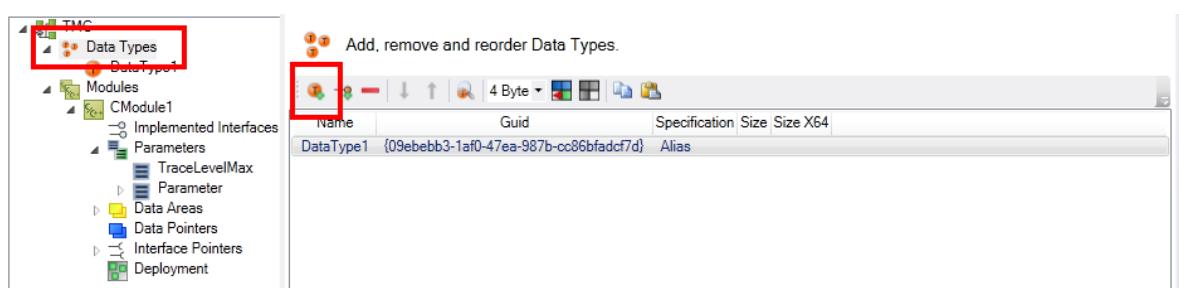
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Datentypen, die von TwinCAT C++ Modulen verwendet werden, hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

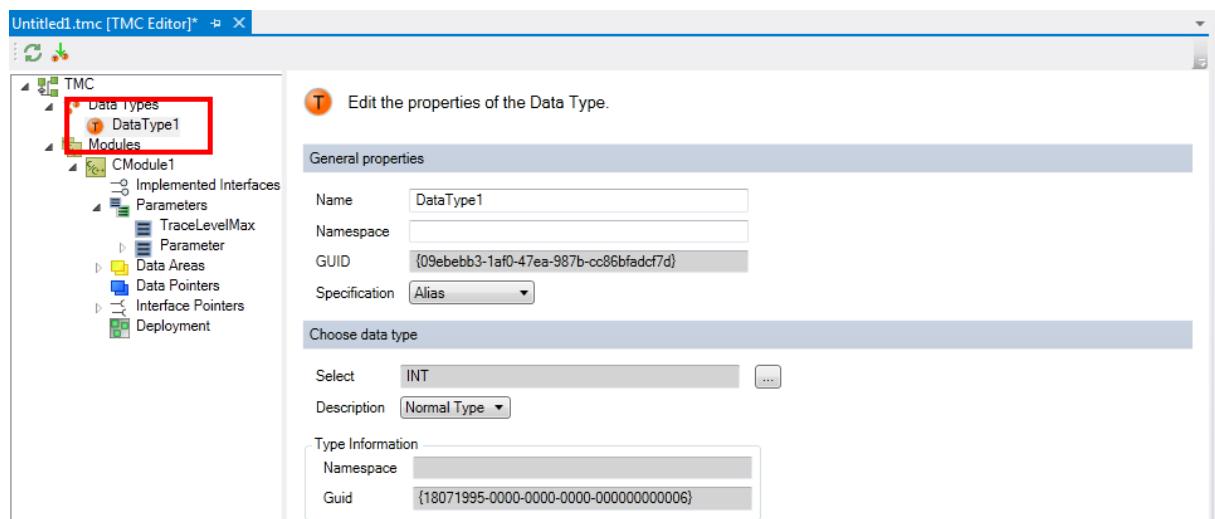
- Schritt 1: Einen neuen Datentyp [► 84] in der TMC-Datei erstellen
- Schritt 2: TwinCAT TMC Code Generator starten [► 87], um C++ Code auf der Grundlage einer Modulbeschreibung in der TMC-Datei zu generieren
- Die Datentypen verwenden [► 100]

#### Schritt 1: Einen neuen Datentyp erzeugen

1. Nach dem Starten des TMC Editors den Knoten „**Data Types**“ auswählen.
2. Die Liste der Datentypen und Schnittstellen wird mit einem neuen Datentyp durch Klicken auf die „+“ Schaltfläche „Add a new data area“ erweitert.  
⇒ Daraufhin wird ein neuer „**Datentyp**“ als neuer Eintrag aufgeführt:



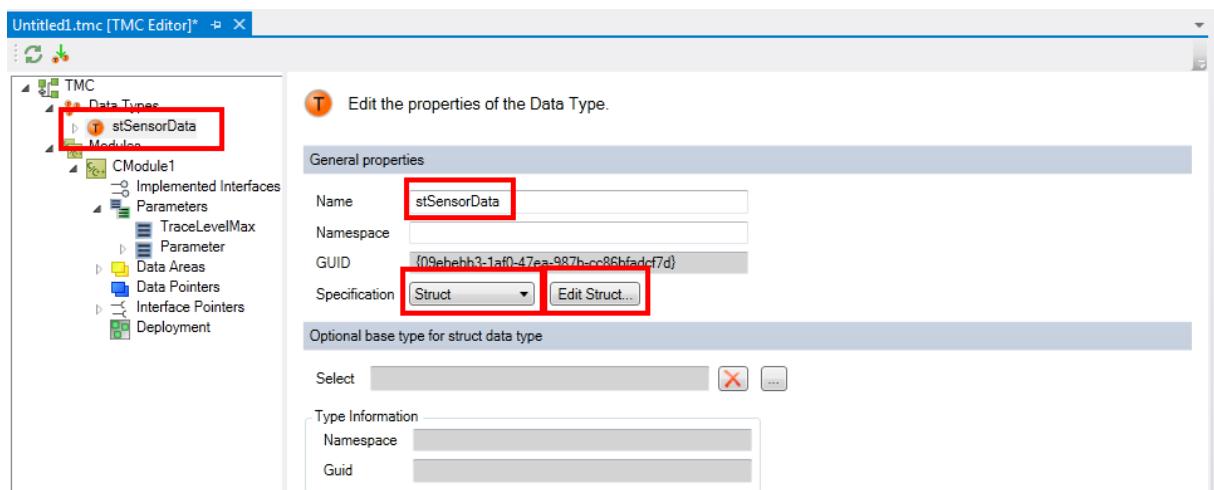
3. Wählen Sie den generierten „Data Type1“ um Einzelheiten zum neuen Datentyp zu erhalten.



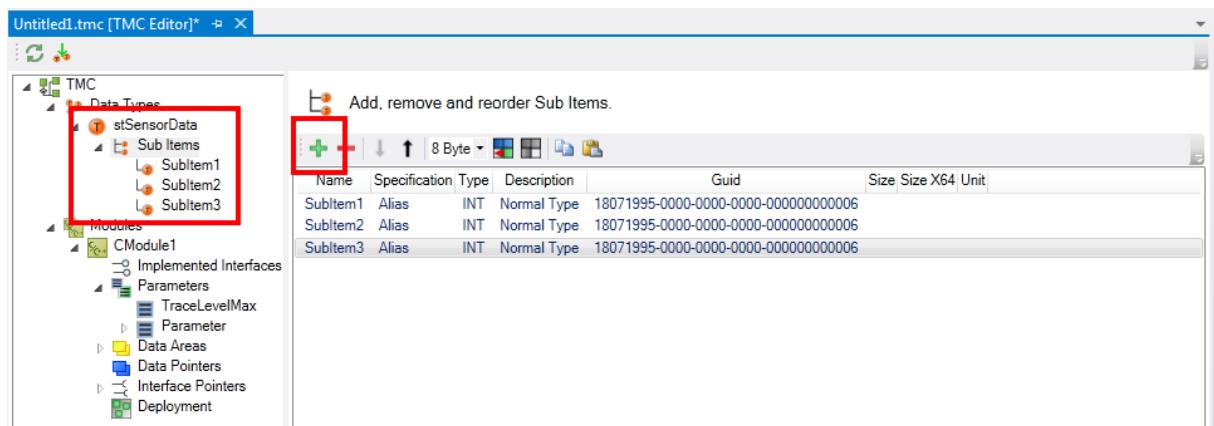
4. Den Datentyp spezifizieren.  
Genaueres siehe [hier \[► 94\]](#).

5. Den Datentyp umbenennen.

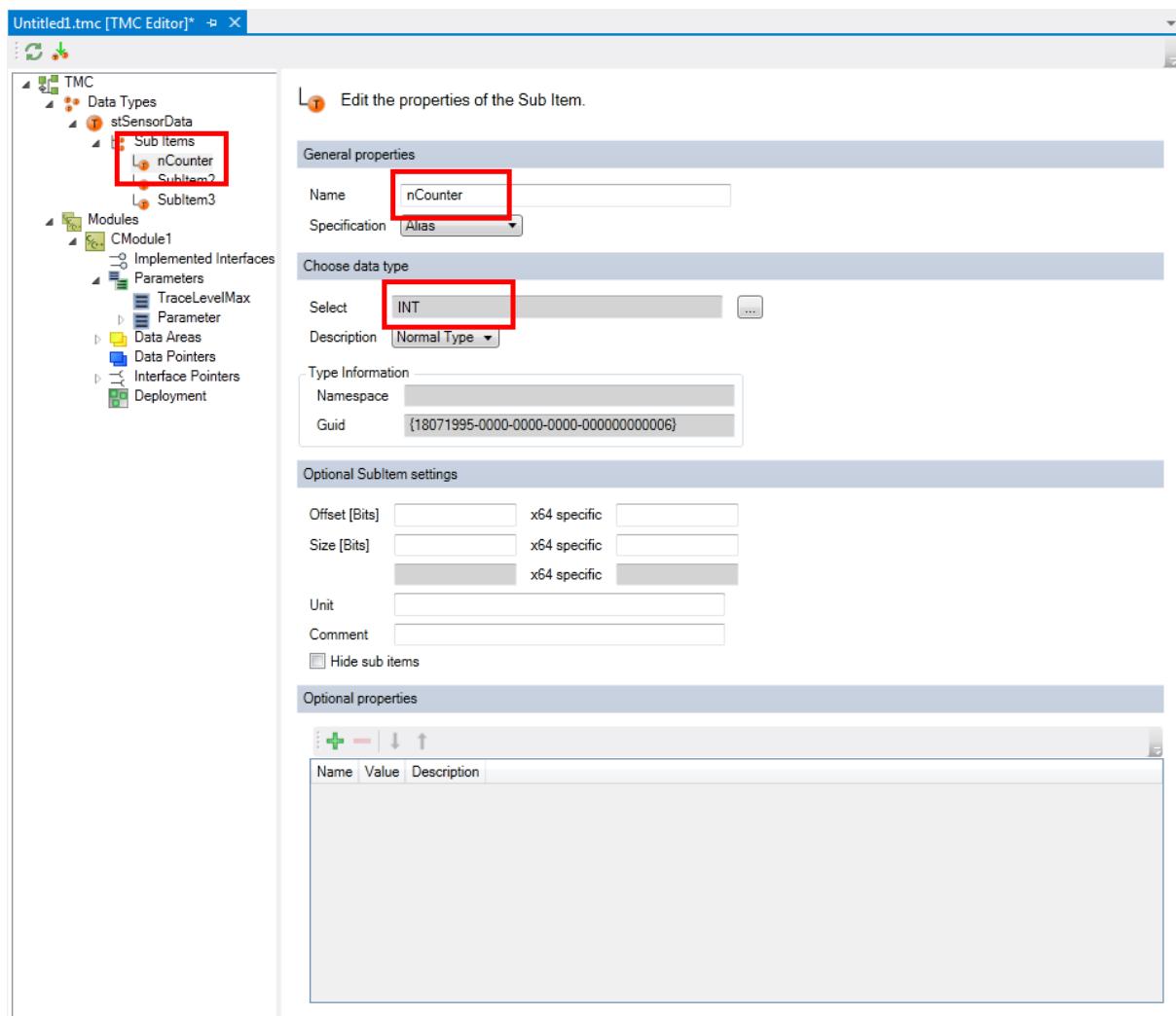
In diesem Beispiel „**stSensorData**“ wählen Sie die Spezifikation „STRUCT“ und klicken auf „Edit Struct“.



6. Fügen Sie neue Unterelemente in die Struktur ein mittels Klicken auf die „Add a new sub item“-Schaltfläche.



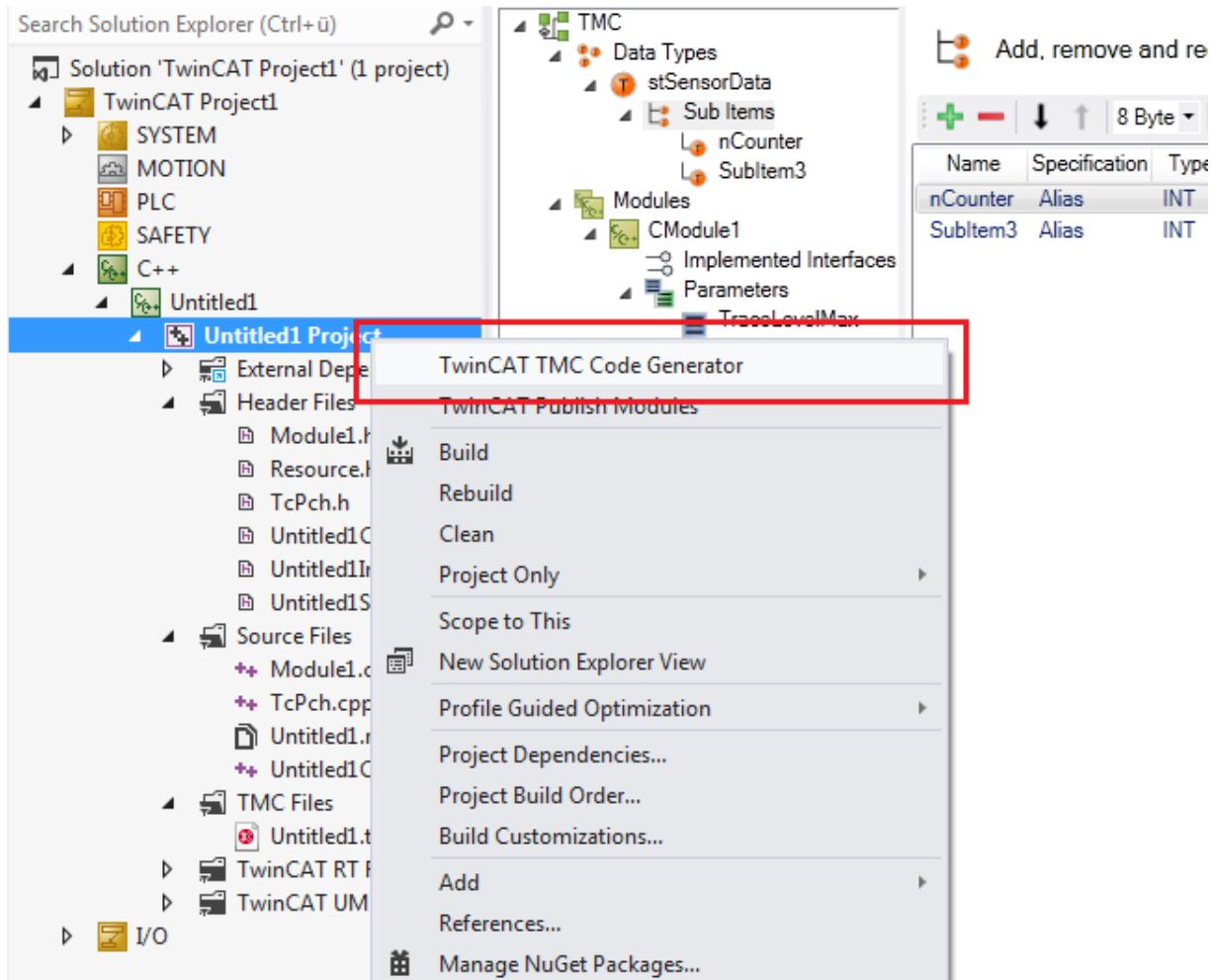
7. Mittels Doppelklick auf das Unterelement können Sie die Eigenschaften bearbeiten. Geben Sie dem Unterelement einen neuen Namen und wählen Sie einen geeigneten Datentyp



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.**

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



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

```

// Untitled1Services.h
#ifndef _TC_TYPE_D6526A1E_8A6F_48EA_A3AD_C2CAC8D56B04_INCLUDED_
#define _TC_TYPE_D6526A1E_8A6F_48EA_A3AD_C2CAC8D56B04_INCLUDED_
#pragma pack(push, 1)

typedef struct _stSensorData
{
    SHORT nCounter;
    unsigned int tTimeStamp;
} stSensorData, *PstSensorData;
#pragma pack(pop)

typedef struct _Module1Parameter
{
    ULONG data1;
    ULONG data2;
} Module1Parameter, *PModule1Parameter;
#endif // !defined(_TC_TYPE_D6526A1E_8A6F_48EA_A3AD_C2CAC8D56B04_INCLUDED_)

```

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

### Sehen Sie dazu auch

- Array [▶ 97]
- Enum [▶ 98]
- Struct [▶ 98]
- Schnittstellen [▶ 99]

### 11.3.3.3 Schnittstellen hinzufügen / bearbeiten / löschen

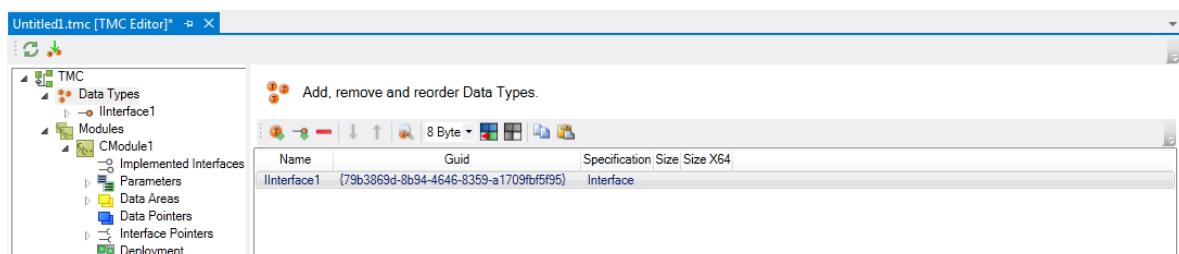
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Schnittstellen eines TwinCAT Moduls hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

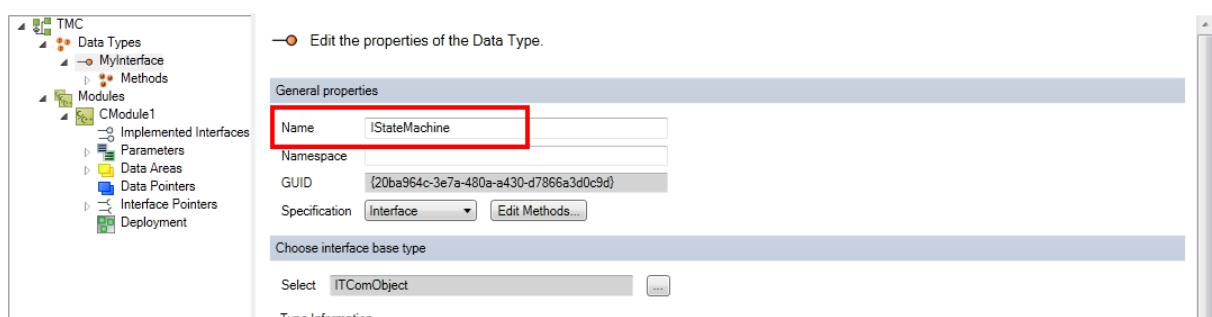
- Schritt 1: Eine neue Schnittstelle [▶ 88] in der TMC-Datei erstellen
- Schritt 2: [▶ 89]Der Schnittstelle in der TMC-Datei Methoden hinzufügen
- Schritt 3: Verwenden Sie die Schnittstelle [▶ 90], indem sie diese zu „Implemented Interfaces“ des Moduls hinzufügen.
- Schritt 4: Starten Sie den TwinCAT TMC [▶ 92] Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.
- Optionale Änderung der Schnittstelle [▶ 92]

#### Schritt 1: Eine neue Schnittstelle erzeugen

1. Nach dem Starten des TMC Editors den Knoten „Data Types“ auswählen.
2. Die Liste der Schnittstellen wird mittels Klicken auf „Add a new interface“ um eine neue Schnittstelle erweitert.  
⇒ Daraufhin wird „IInterface1“ als neuer Eintrag aufgeführt:



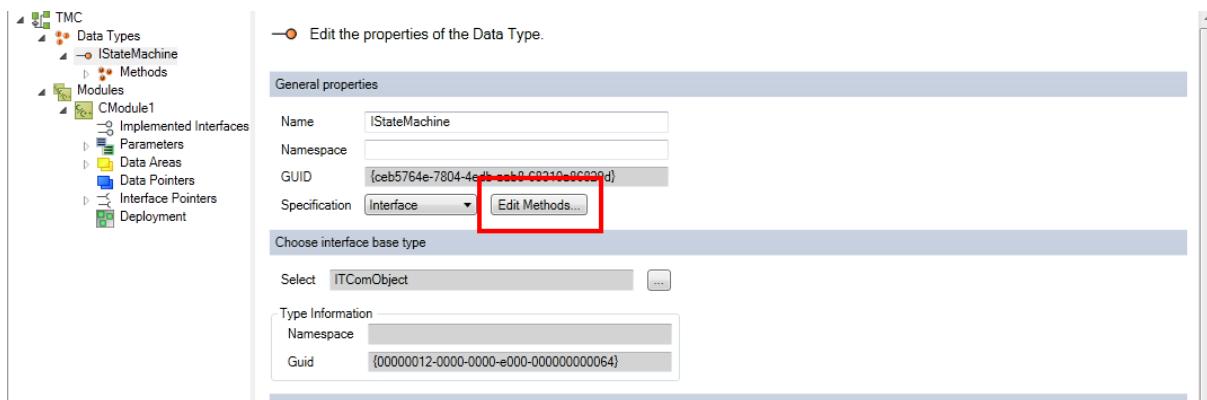
3. Entweder wählen Sie den entsprechenden Knoten im Baum oder machen einen Doppelklick auf die Zeile in der Tabelle, um die Einzelheiten zu öffnen



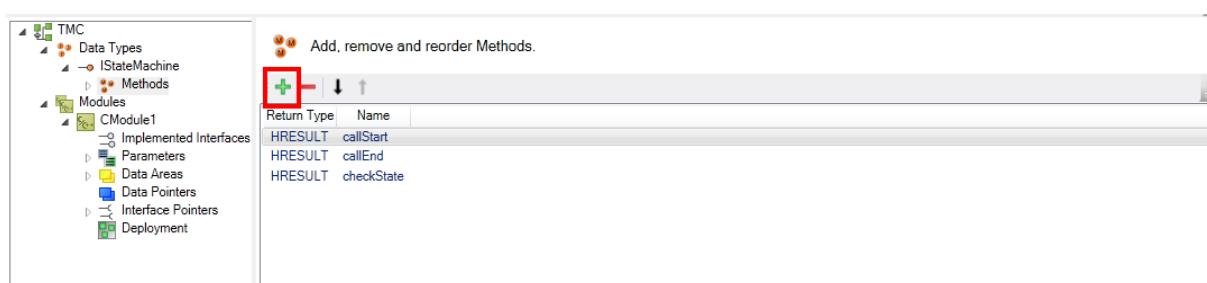
4. Geben Sie einen aussagekräftigeren Namen ein - in diesem Beispiel „IStateMachine“.

## Schritt 2: Fügen Sie der Schnittstelle Methoden hinzu

5. Klicken Sie auf „Edit Methods...“, um eine Liste der Methoden dieser Schnittstelle zu erhalten:



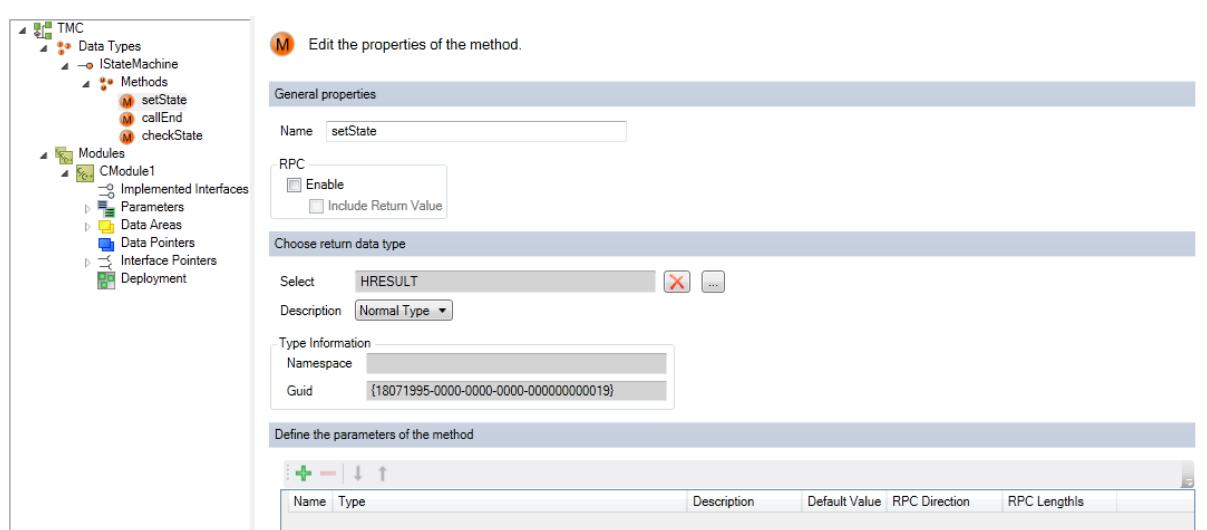
6. Klicken Sie auf die „+“ Schaltfläche um eine neue standardmäßige Methode „Method1“ zu erzeugen:



7. Doppelklicken Sie auf die Methode oder wählen den Knoten im Baum aus, um Einzelheiten zu öffnen

8. Geben Sie der standardmäßigen „Method1“ einen aussagekräftigeren Namen.

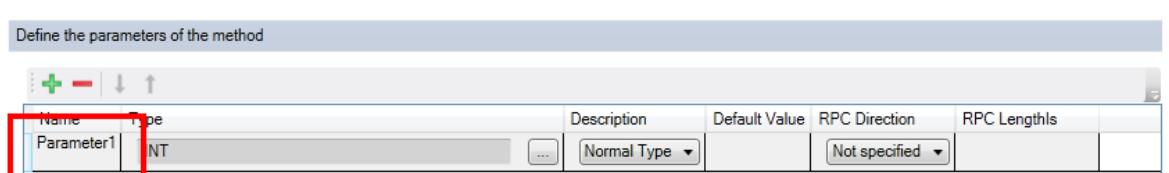
9. Anschließend können Sie mit einem Klick auf „Add a new parameter“ Parameter hinzufügen / Parameter der Methode „SetState“ bearbeiten



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

10. Durch einen Klick auf den Namen „Parameter1“ kann dieser bearbeitet werden.

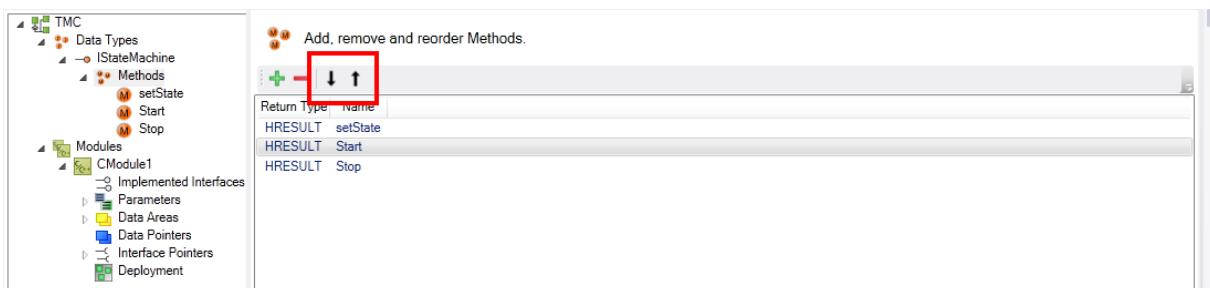
⇒ Der „Normal Type“ kann auch in „Pointer“ geändert werden usw. - auch kann der Datentyp selber ausgewählt werden



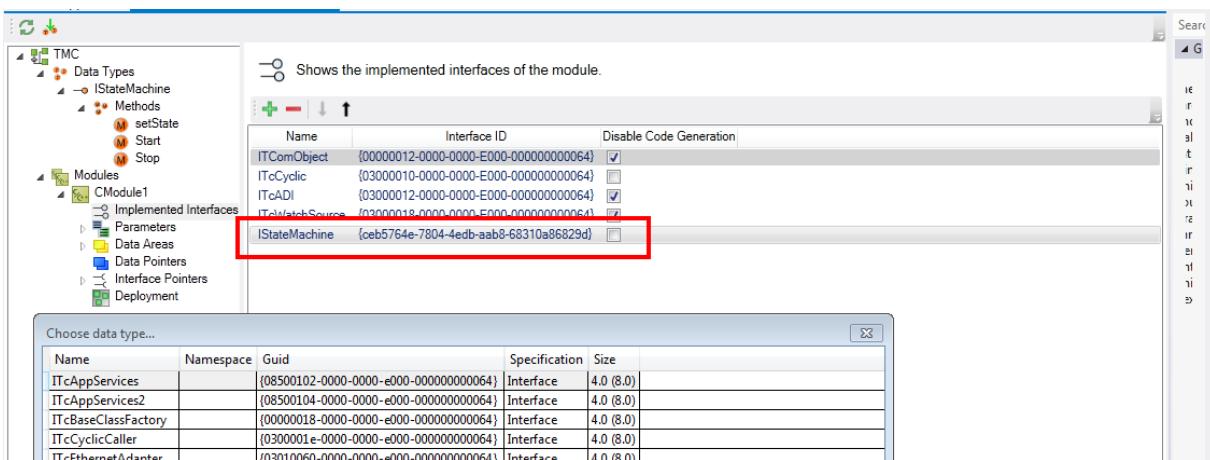
⇒ In diesem Falle ist „NewState“ der neue Name - die übrigen Einstellungen werden nicht geändert

Name	Type	Description	Default Value	RPC Direction	RPC Lengths
NewState	INT		Normal Type	Not specified	

11. Durch Wiederholen des Schritts 2 „Methoden zur Schnittstelle hinzufügen“ werden alle Methoden aufgelistet - mit Hilfe der „nach oben“ / „nach unten“ Schaltfläche können die Methoden neu geordnet werden.



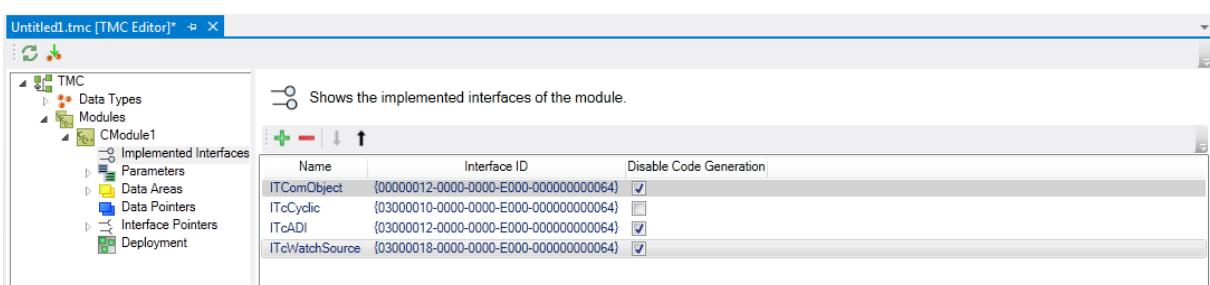
12. Die Schnittstelle ist bereit durch Ihr Modul implementiert zu werden.



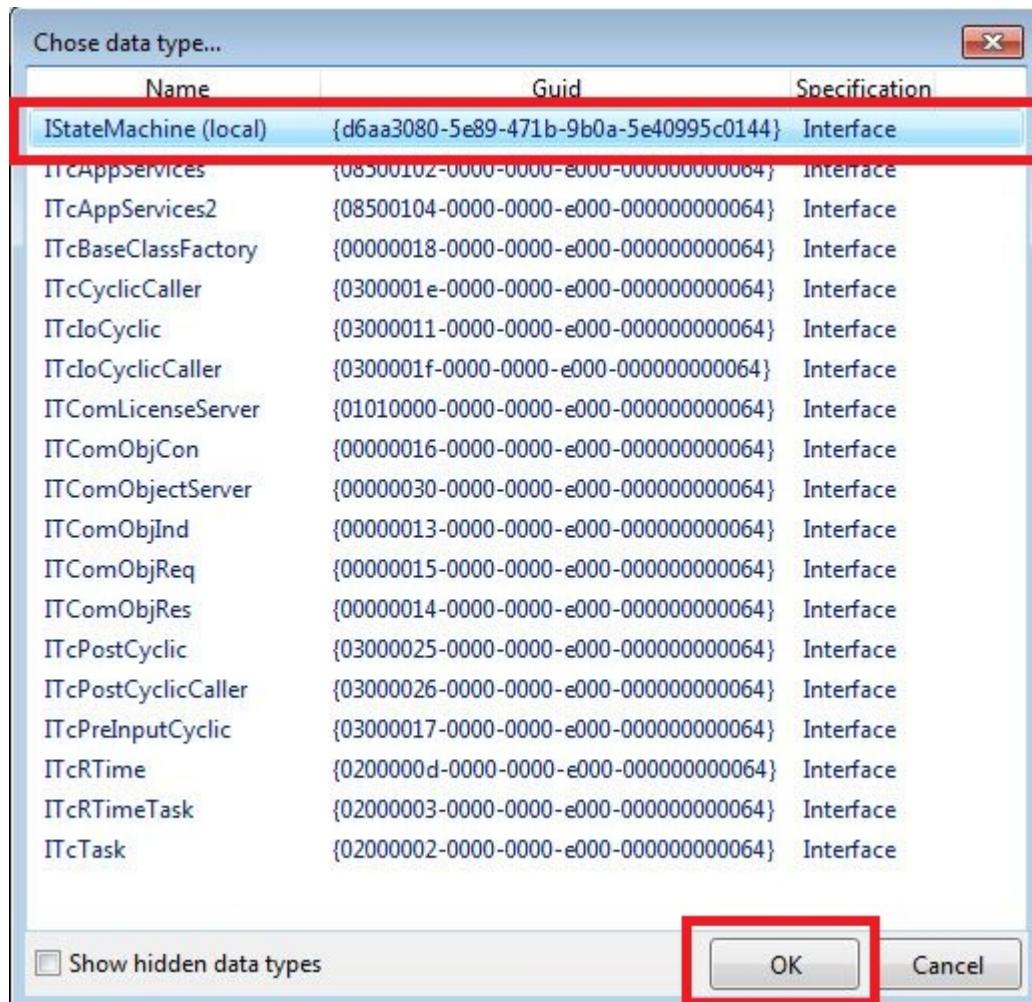
### Schritt 3: Die neue Schnittstelle zu „Implemented Interfaces“ hinzufügen

13. Wählen Sie das Modul, das mit der neuen Schnittstelle erweitert werden soll - in diesem Falle wählen Sie das Ziel „Modules->CModule1“.

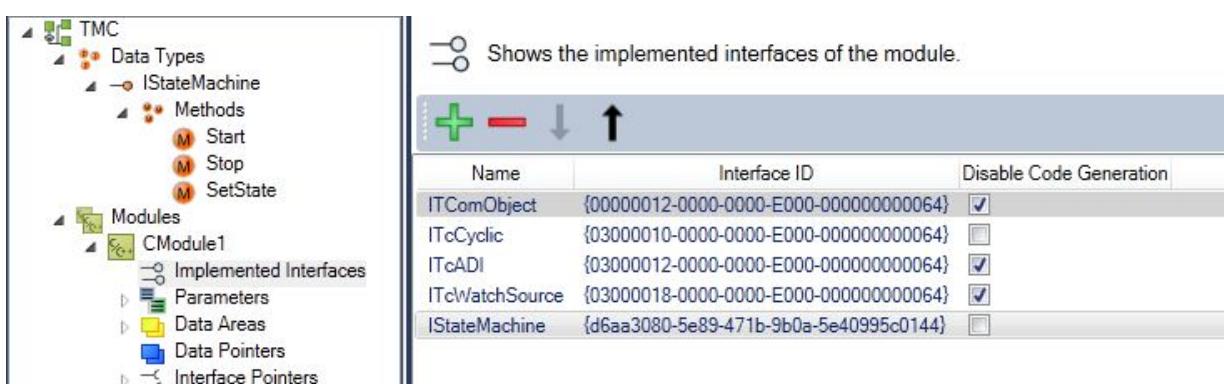
14. Die Liste der implementierten Schnittstellen wird mittels Klicken auf die „+“ Schaltfläche mit „Add a new interface to the module“ um eine neue Schnittstelle erweitert.



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

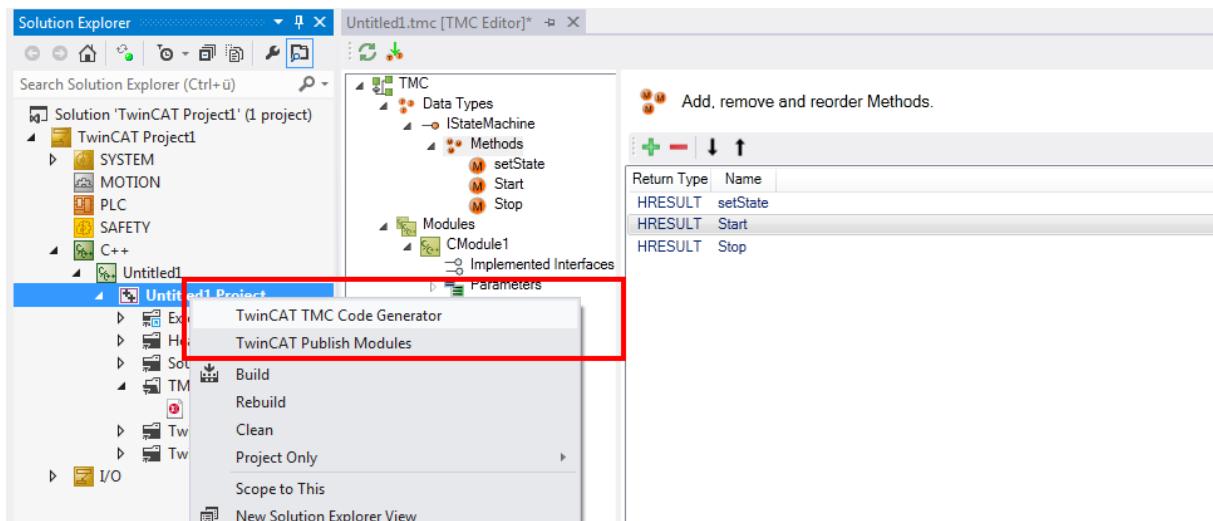


⇒ In Folge dessen ist nun die neue Schnittstelle „**IStateMachine**“ Teil der Modulbeschreibung.



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

- 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

**HINWEIS!** Im Falle 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>
```

**Sehen Sie dazu auch**

Implementierte Schnittstellen [▶ 102]

### 11.3.3.4 Datentypeigenschaften

#### Die Eigenschaften von Datentypen bearbeiten

The screenshot shows the 'Edit the properties of the Data Type' dialog for 'DataType1' in the TMC software. The dialog is organized into several sections:

- General properties:** Name: DataType1, Namespace: (empty), Guid: {ecccd91-3635-43a3-987b-8dbf355f1a55}, Specification: Alias.
- Choose data type:** Select: INT, Description: Normal Type.
- Type Information:** Namespace: (empty), Guid: {18071995-0000-0000-000000000006}.
- Optional data type settings:** Size [Bits]: (empty), x64 specific: (empty).
- C/C++ Name:** default: (empty), x64 specific: (empty).
- Unit:** (empty).
- Comment:** (empty).
- Optional Defaults:** Value: 4, Enum: (selected), String: (disabled). Min: 1, Max: 5.
- Optional properties:** A table with columns Name, Value, and Description. It contains a single row with a '+' icon and a '-' icon.
- Datatype Hides:** A table with a single row containing the GUID: {ecccd91-3635-43a3-987b-8dbf355f1a55}.

The left sidebar shows a tree view of the project structure: TMC > Data Types > DataTypes > CModule1 > Implemented Interfaces, Parameters, Data Areas, Data Pointers, Interface Pointers, Deployment.

## Allgemeine Eigenschaften

**Name:** Benutzerdefinierter Name des Datentyps

 <b>Hinweis</b>	<p><b>Namenskonflikt</b></p> <p>Bitte verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen. Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.</p>
---	---

**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“:

 Edit the properties of the Data Type.

General properties	
Name	ASampleEnum
Namespace	A
GUID	{41d4a207-3a09-4316-9d89-0dd1881ab8c4}
Specification	Enumeration

Der folgende Code wird generiert:

```
///<AutoGeneratedContent id="DataTypes">
#ifndef _TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_
#define _TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_
enum A_ASampelEnum : 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:

```
#ifndef _TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_
#define _TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_
enum B_BSampelEnum : 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:** Ein Alias eines Standarddatentyps (z.B. INT) erzeugen
- **Array [▶ 97]:** Ein benutzerdefiniertes Array erstellen
- **Enumeration [▶ 98]:** Eine benutzerdefinierte Aufzählung erstellen
- **Struct [▶ 98]:** Eine benutzerdefinierte Struktur erzeugen
- **Interface [▶ 99]:** 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 verwendet 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 Falle 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.

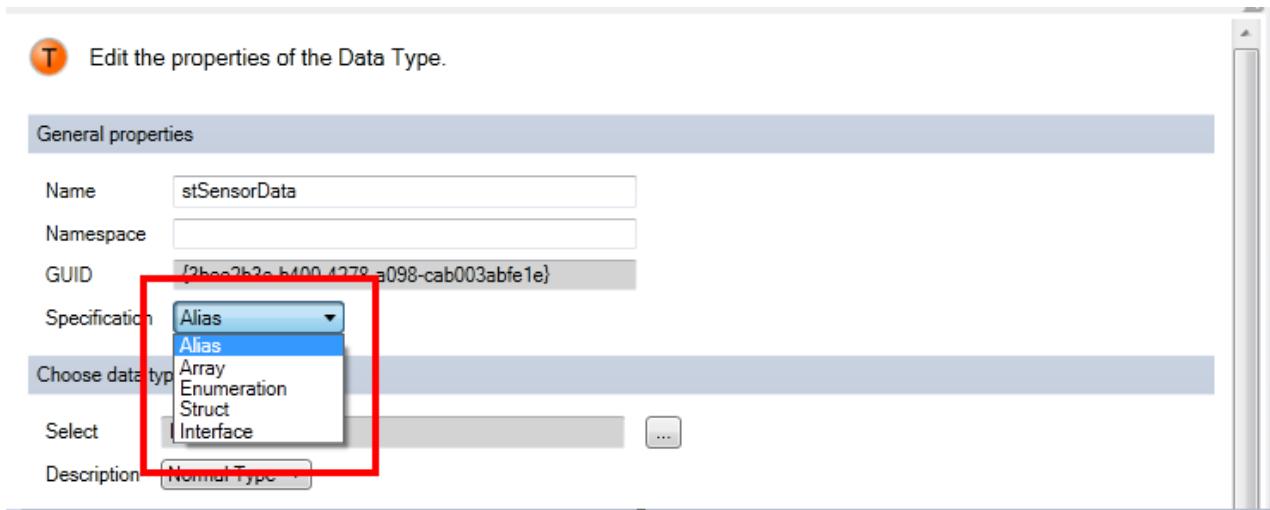
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

#### 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.

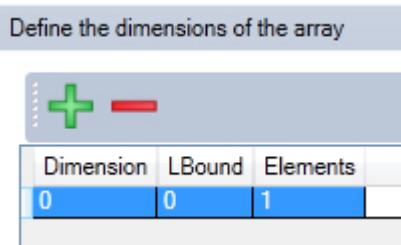
### 11.3.3.5 Spezifikation

In diesem Abschnitt wird die Spezifikation von Datentypen beschrieben.



### 11.3.3.5.1 Array

**Array:** Ein benutzerdefiniertes Array erstellen



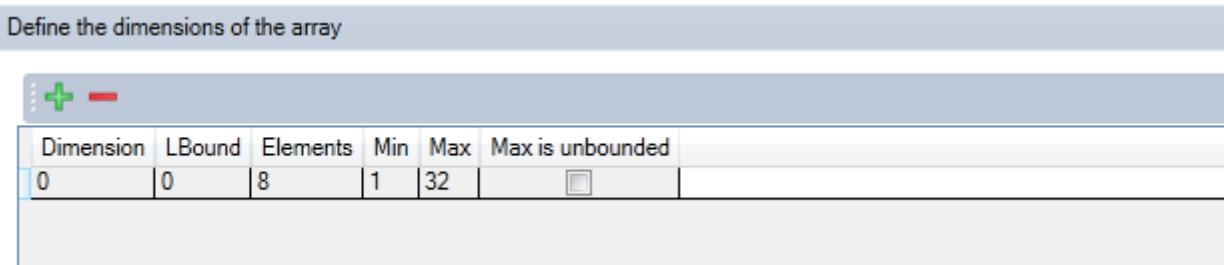
Es wird ein neuer Dialog eingeblendet, um Array-Elemente hinzuzufügen (+) oder zu entfernen (-).

**Dimension:** Dimension des Array

**LBound:** Linke Grenze des Array (Standardwert = 0)

**Elemente:** Menge der Elemente

### Dynamische Arrays für Parameter und Datenzeiger



Im Falle von [Parameter \[▶ 103\]](#) und [Datenzeigern \[▶ 118\]](#) unterstützt TwinCAT 3 Arrays mit dynamischer Größe.

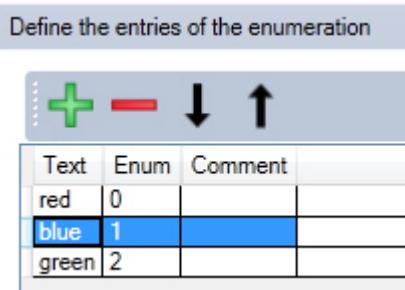
**Min:** Mindestgröße des Arrays

**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.

 <b>Hinweis</b>	<b>Eindeutige Namen für Aufzählungselemente erforderlich</b> Beachten Sie, dass die Aufzählungselemente eindeutig benannt sein müssen, da ansonsten der generierte C++ Code ungültig ist.
---	--

**Text:** Aufzählungselement

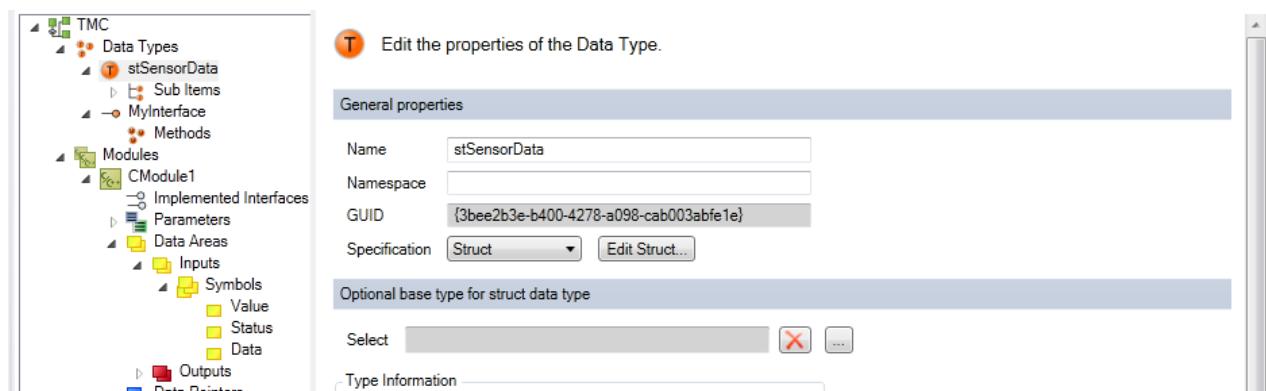
**Enum:** Geeigneter Integerwert

**Comment:** Optionaler Kommentar

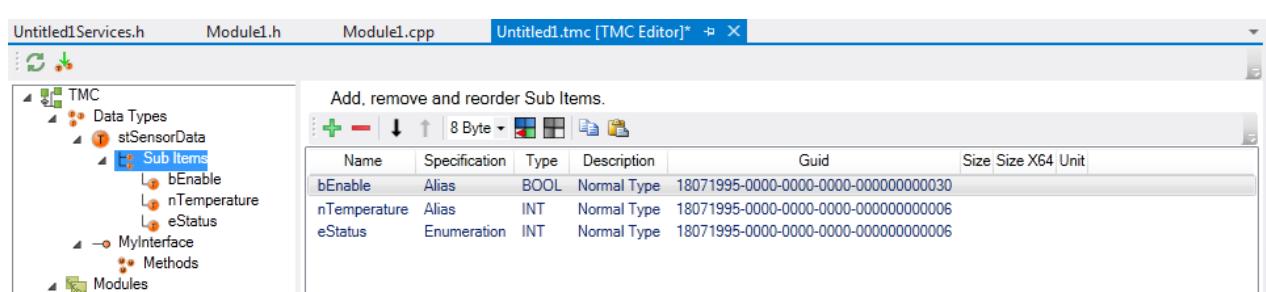
### 11.3.3.5.3 Struct

**Struct:** Eine benutzerdefinierte Struktur erstellen

Wählen Sie bitte 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.

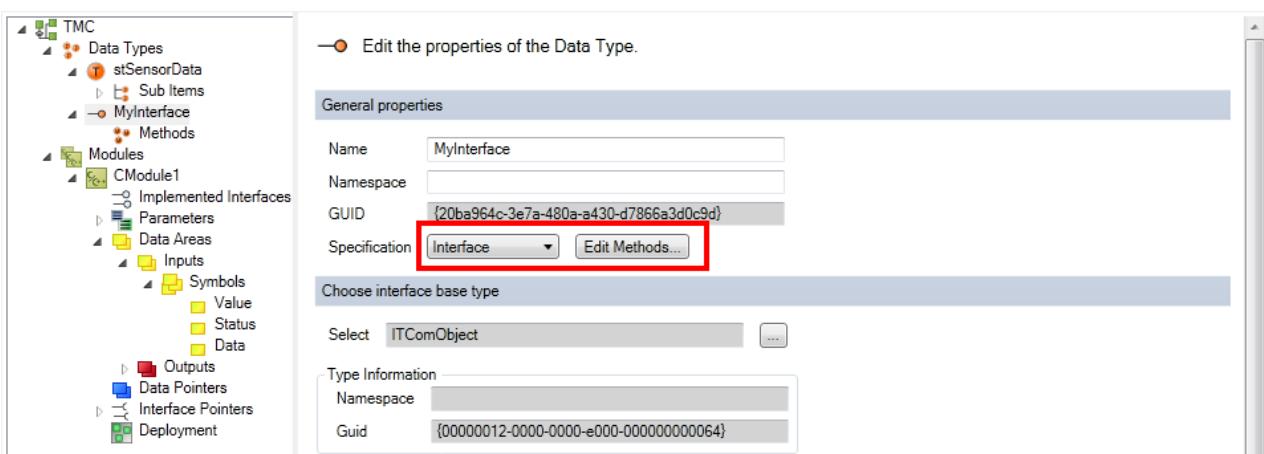
**Size X64:** Andere Größe für x64 Plattform wird zusätzlich bereitgestellt.

**Unit:** Optionale Einheit

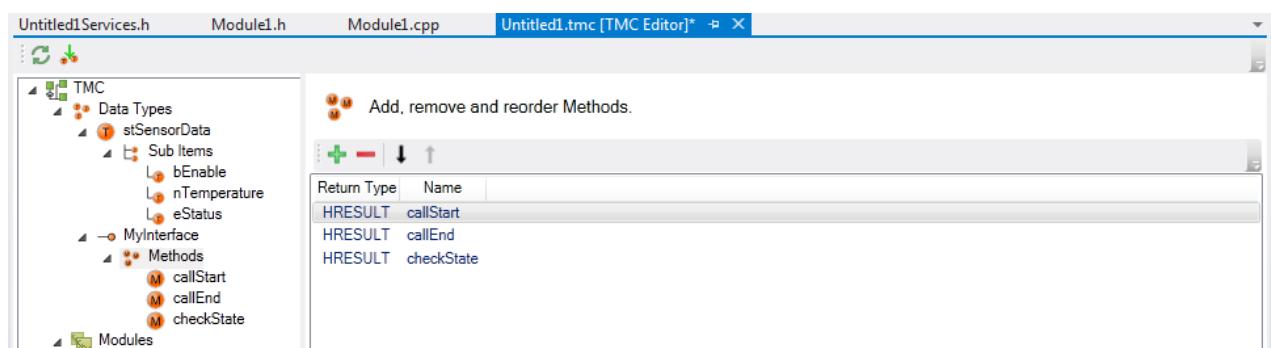
Mittels Auswahl des Datentyps oder Doppelklick auf den Tabelleneintrag werden die Details der Konfigurationsseite des Unterelements eingeblendet. Ähnlich wie [Datentypeigenschaften \[► 94\]](#).

#### 11.3.3.5.4 Schnittstellen

**Interfaces:** Eine benutzerdefinierte Schnittstelle erstellen.

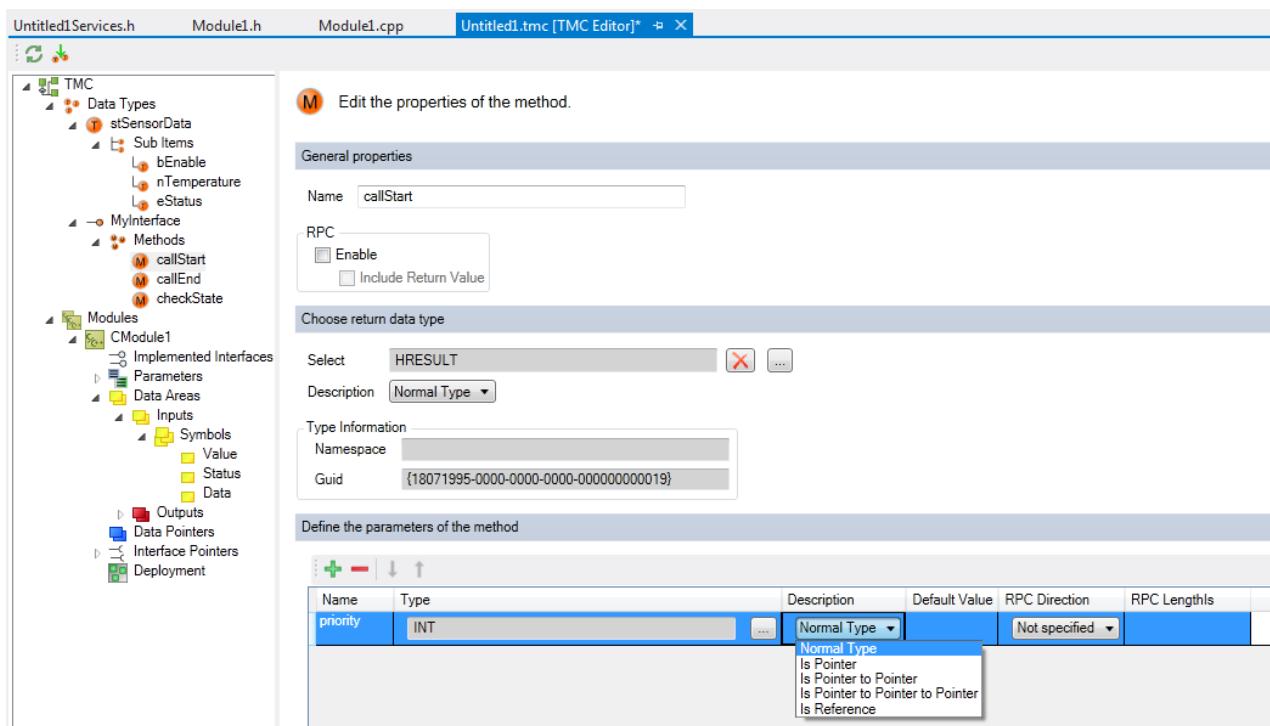


Wählen Sie bitte den „Methods“-Knoten oder klicken Sie auf die „Edit Methods“-Schaltfläche, um zu dieser Tabelle zu wechseln:



#### 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 „Remoteprozedaaufrufen“ von außerhalb dieser Methode.

- **Include Return Value:** Freigabe der Weitergabe des Rückgabewerts der Methode

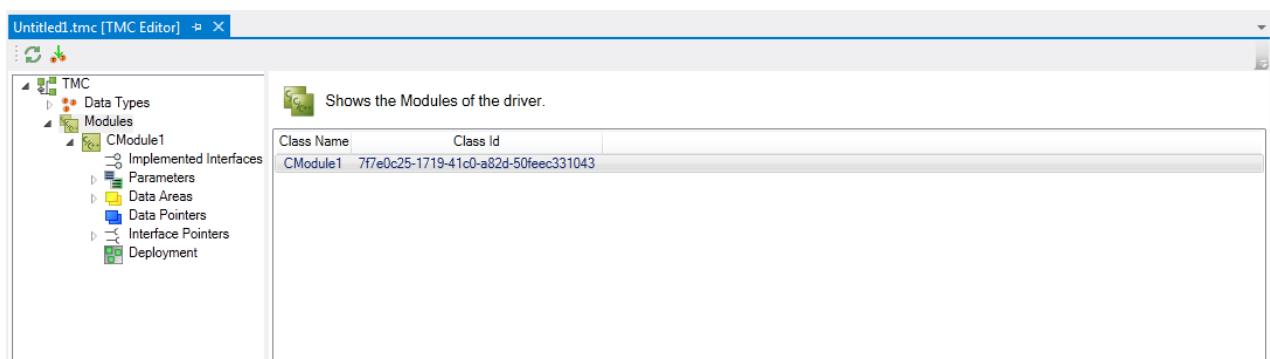
Felder entsprechen denjenigen der [Datentypeigenschaften \[▶ 94\]](#).

#### Parameter der Methode definieren

- Name
- Type: Bekannt aus den [Datentypeigenschaften \[▶ 94\]](#)
- Description: Bekannt aus den [Datentypeigenschaften \[▶ 94\]](#)
- 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 Remoteprozederaaufrufen (RPC) ignoriert wird.

### 11.3.4 Module

**Modules:** Zeigt die Module des Treibers

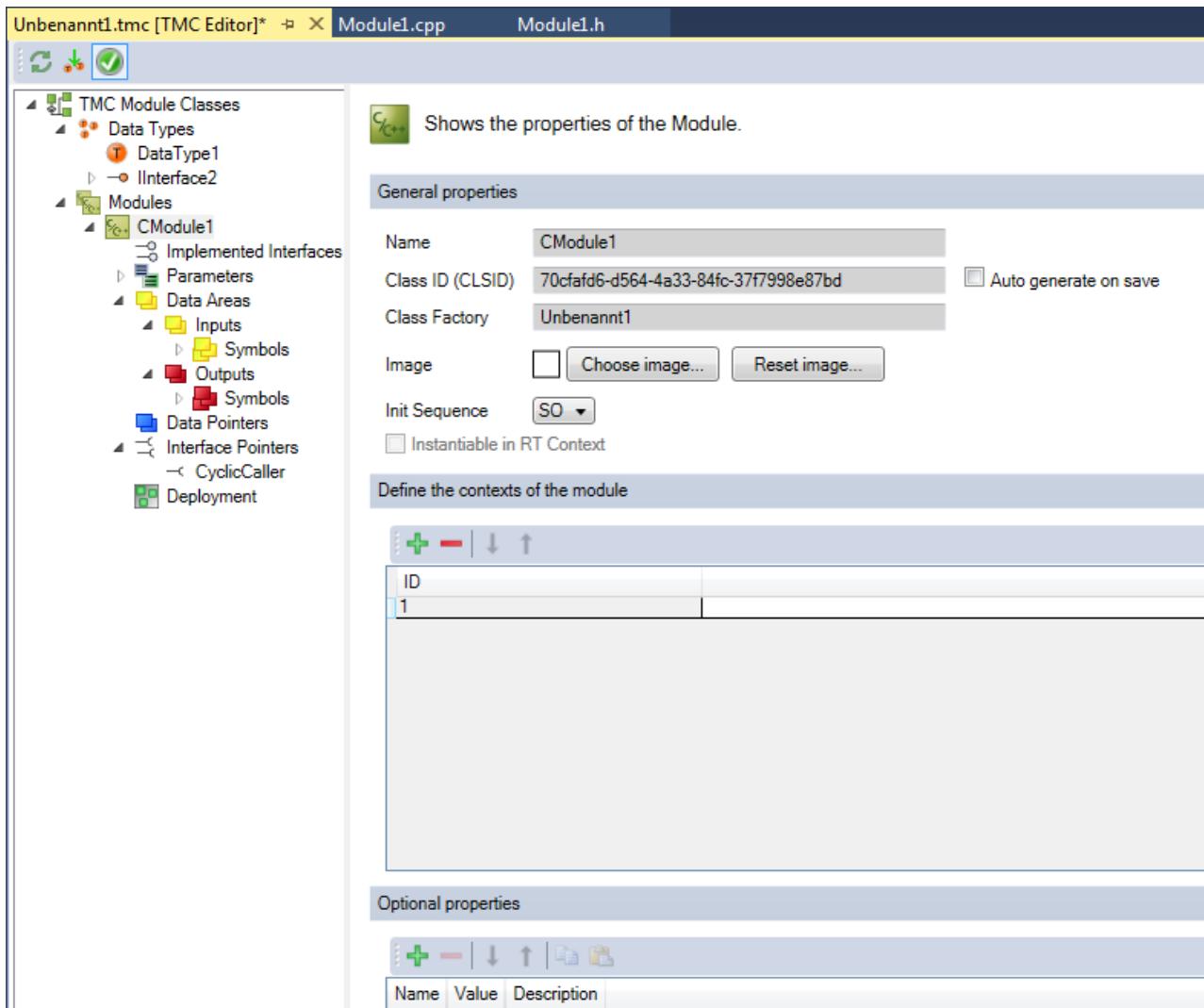


**Class Name:** Name des Moduls

**Class ID:** Eindeutige ID des Moduls

### Module Properties:

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



**Shows the properties of the Module.**

#### General properties

Name	CModule1	<input type="checkbox"/> Auto generate on save
Class ID (CLSID)	70cfaf6-d564-4a33-84fc-37f7998e87bd	
Class Factory	Unbenannt1	
Image	<input type="checkbox"/> Choose image... <input type="button" value="Reset image..."/>	
Init Sequence	SO	
<input type="checkbox"/> Instantiable in RT Context		

#### Define the contexts of the module



ID	
1	

#### Optional properties



Name	Value	Description

## Allgemeine Eigenschaften

**Name:** Name des Moduls

**Class ID:** Eindeutige ID des Moduls

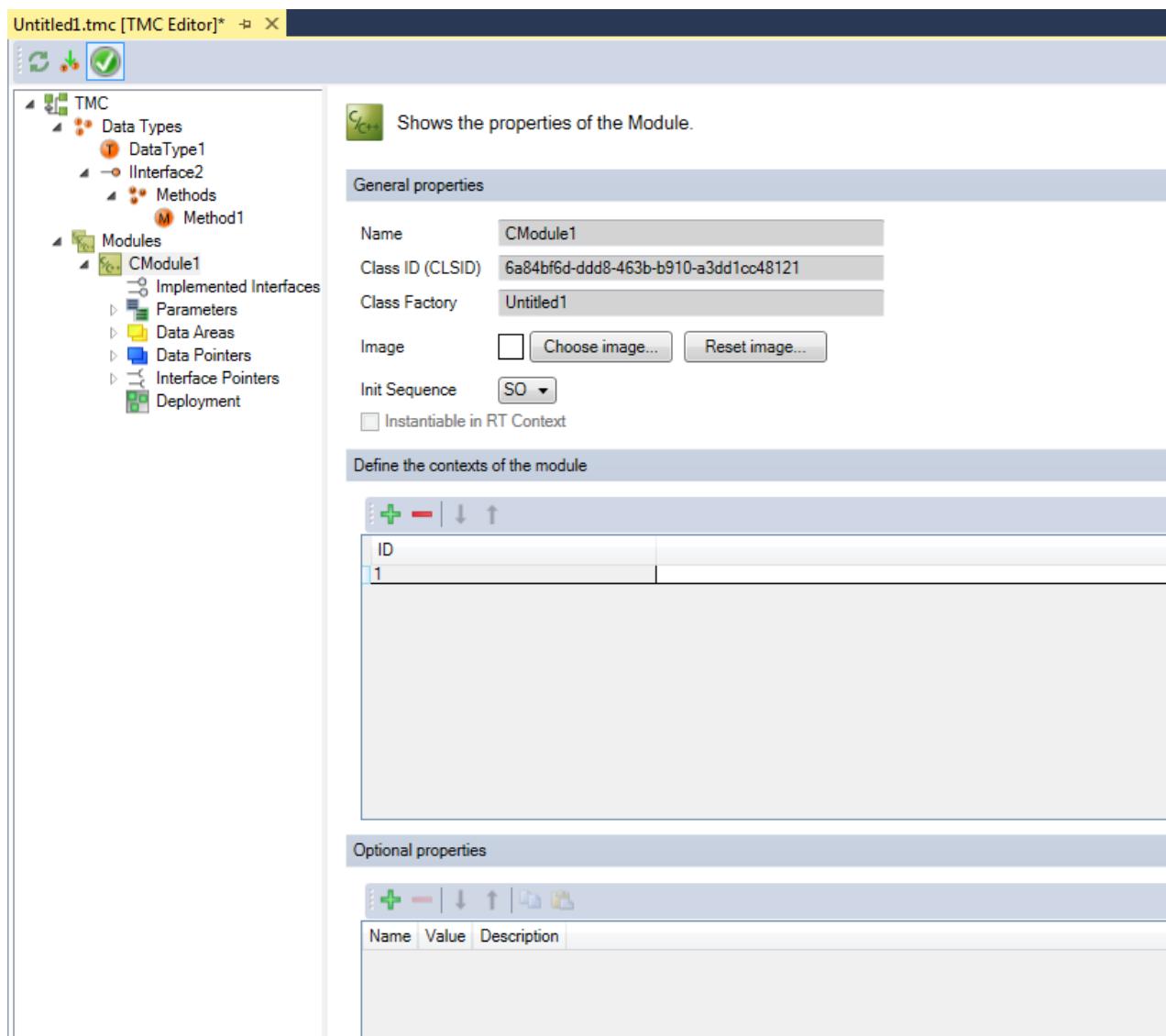
**Auto generate on save:** Ermöglicht, dass TwinCAT die ClassID anhand der Modul-Parameter beim Speichern generiert. Die Änderung der ClassID führt beim Importieren der binären Module dazu, dass die entsprechenden ClassIDs angepasst werden müssen. Somit kann TwinCAT die Interface-Änderung erkennen.

**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 [▶ 124] des Instance Configurators für weitere Auskünfte.)

**Instantiable in RT Context:** Zeigt an, ob dieses Modul unter Echtzeitkontext instanziert werden kann, siehe [TwinCAT Module Klassenassistent \[▶ 76\]](#)



### 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.

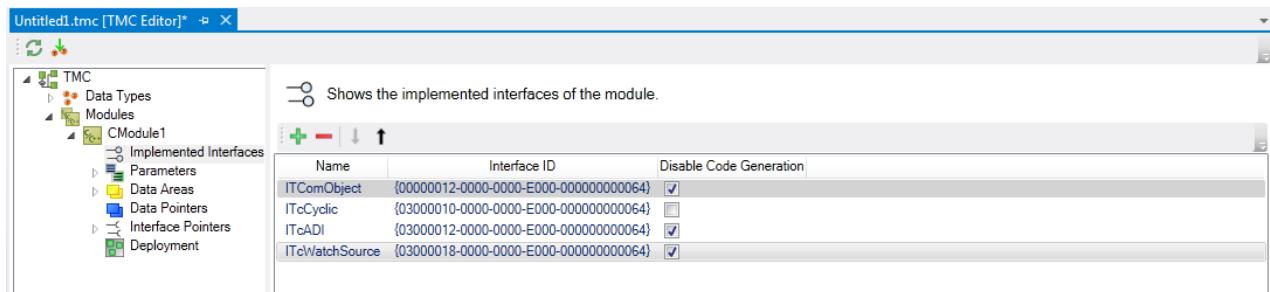
### Optionale Eigenschaften

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

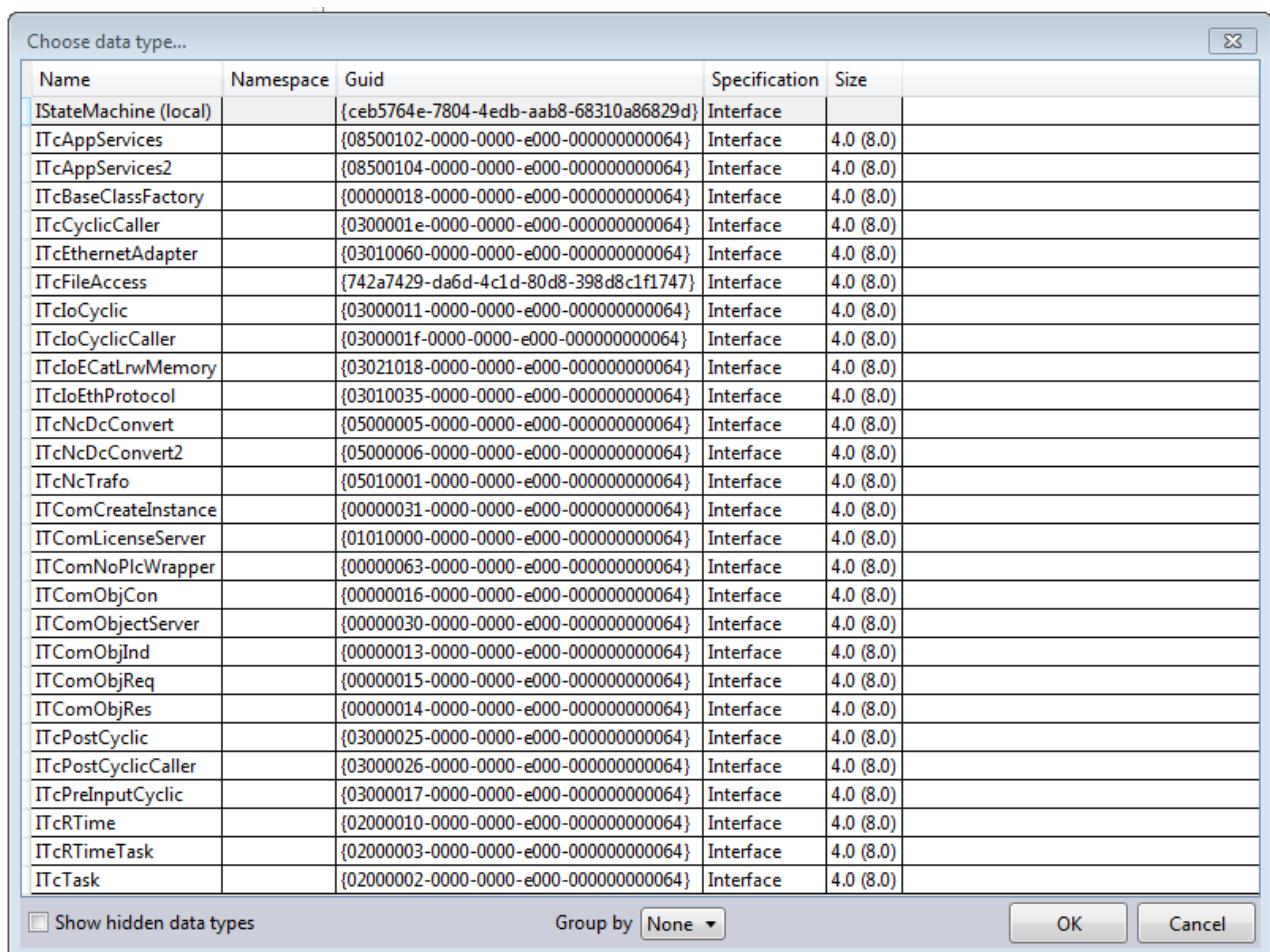


**Name:** Name der Schnittstelle

**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.

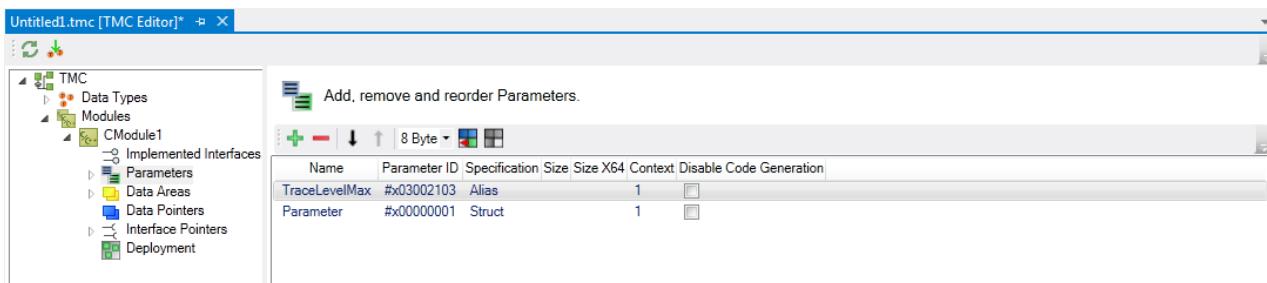
TwinCAT unterstützt drei Arten von „Parameter IDs“ (PTCID) im Abschnitt „Configure the parameter ID“.

- „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 [▶ 125] 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

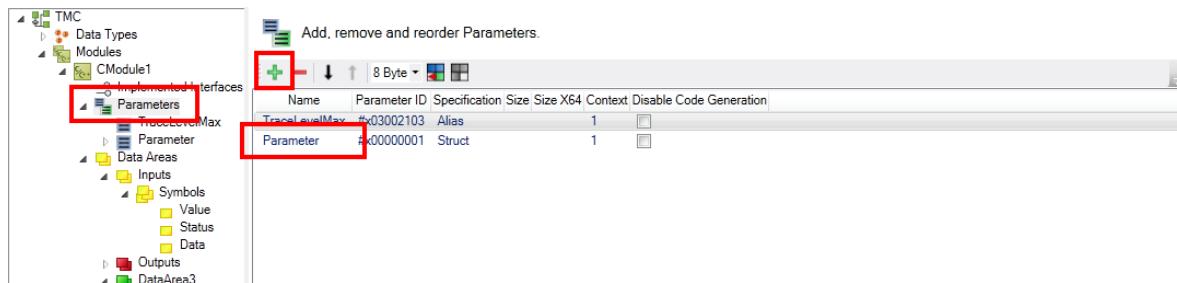
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Eigenschaften und Funktionalitäten einer TwinCAT-Klasse hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

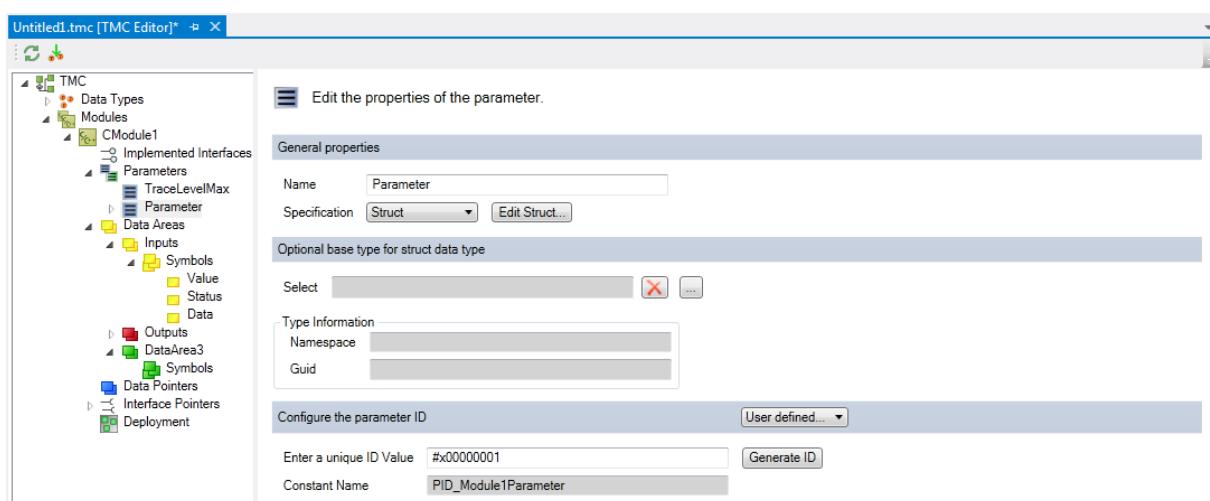
- Schritt 1: Einen neuen Parameter [▶ 105] in der TMC-Datei erstellen
- Schritt 2: Starten Sie den TwinCAT TMC Code Generator [▶ 106], um Code für die Modulbeschreibung in der TMC-Datei zu erzeugen
- Schritt 3: Übergänge der Zustandsmaschine [▶ 107]

## Schritt 1: Neuen Parameter erzeugen

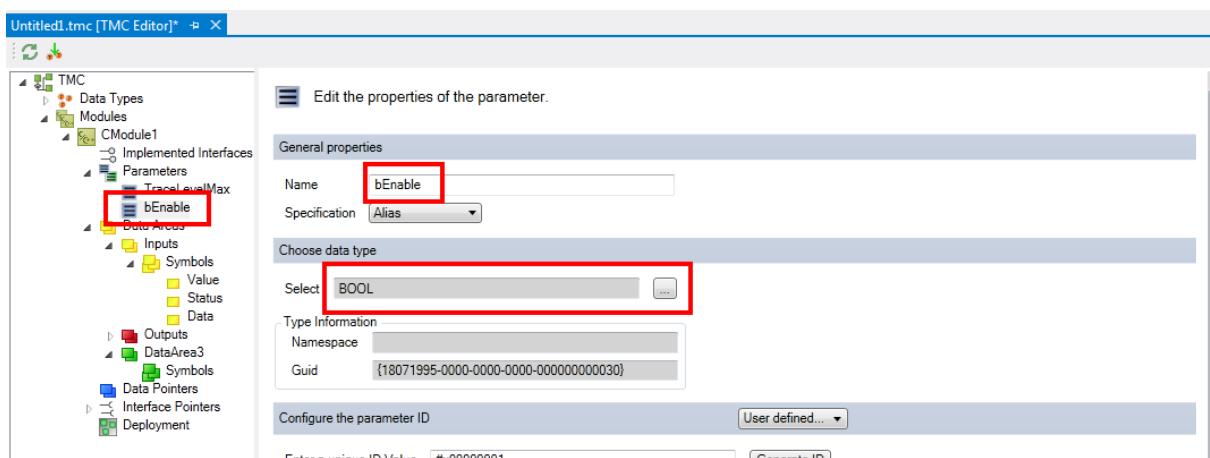
1. Nach dem Starten des TMC Editors das Ziel „Parameters“ auswählen.
2. Die Liste der Parameter wird mittels Klicken auf die „+“ Schaltfläche „Add a new parameter“ um einen neuen Parameter erweitert.  
⇒ Daraufhin wird ein neuer „Parameter“ als neuer Eintrag aufgeführt:



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.



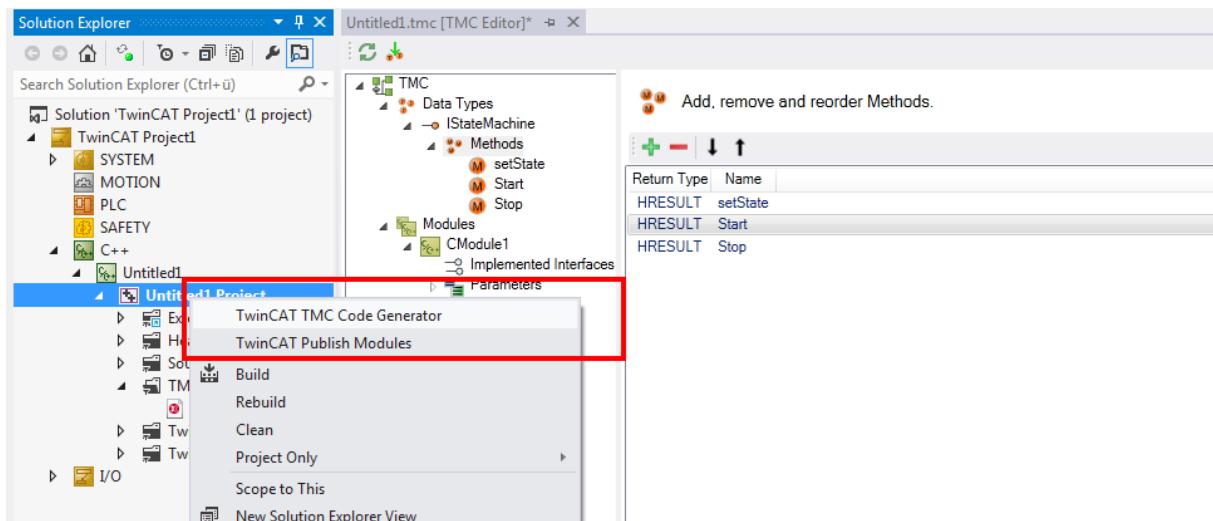
4. Parameter sowie die Datentypen [► 82] konfigurieren
5. Geben Sie diesem einen aussagekräftigeren Namen - in diesem Beispiel „**bEnable**“ - und wählen den Datentyp „BOOL“.



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.

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



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

```
///<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)

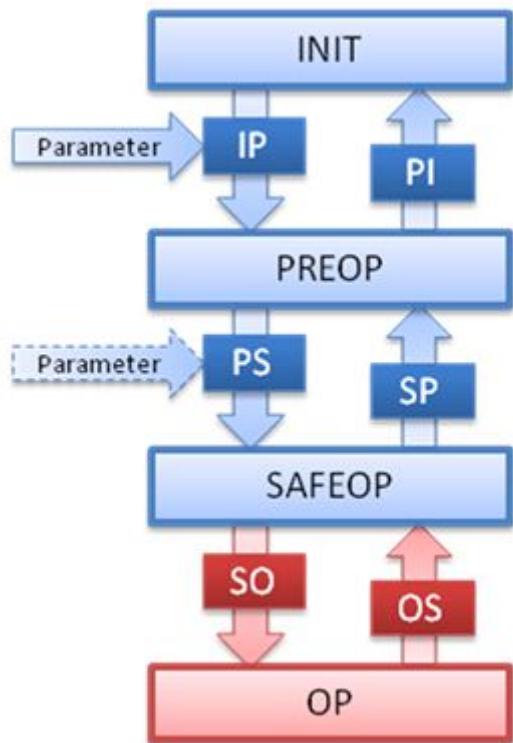
// Set parameters of CModule1
BEGIN_SETOBJPARA_MAP(CModule1)
    SETOBJPARA_DATAAREA_MAP()
    ///<AutoGeneratedContent id="SetObjectParameterMap">
        SETOBJPARA_VALUE(PID_TraceLevel, m_traceLevelMax)
        SETOBJPARA_VALUE(PID_Module1bEnable, m_bEnable)
        SETOBJPARA_TTEPTR(PTD_Ctx_TaskId, m_spCyclicCaller)
    ///</AutoGeneratedContent>
END_SETOBJPARA_MAP()

// Get parameters of CModule1
BEGIN_GETOBJPARA_MAP(CModule1)
    GETOBJPARA_DATAAREA_MAP()
    ///<AutoGeneratedContent id="GetObjectParameterMap">
        GETOBJPARA_VALUE(PID_TraceLevel, m_traceLevelMax)
        GETOBJPARA_VALUE(PID_Module1bEnable, m_bEnable)
        GETOBJPARA_TTEPTR(PTD_Ctx_TaskId, m_spCyclicCaller)
    ///</AutoGeneratedContent>
END_GETOBJPARA_MAP()
```

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

### Schritt 3: Übergänge der Zustandsmaschine

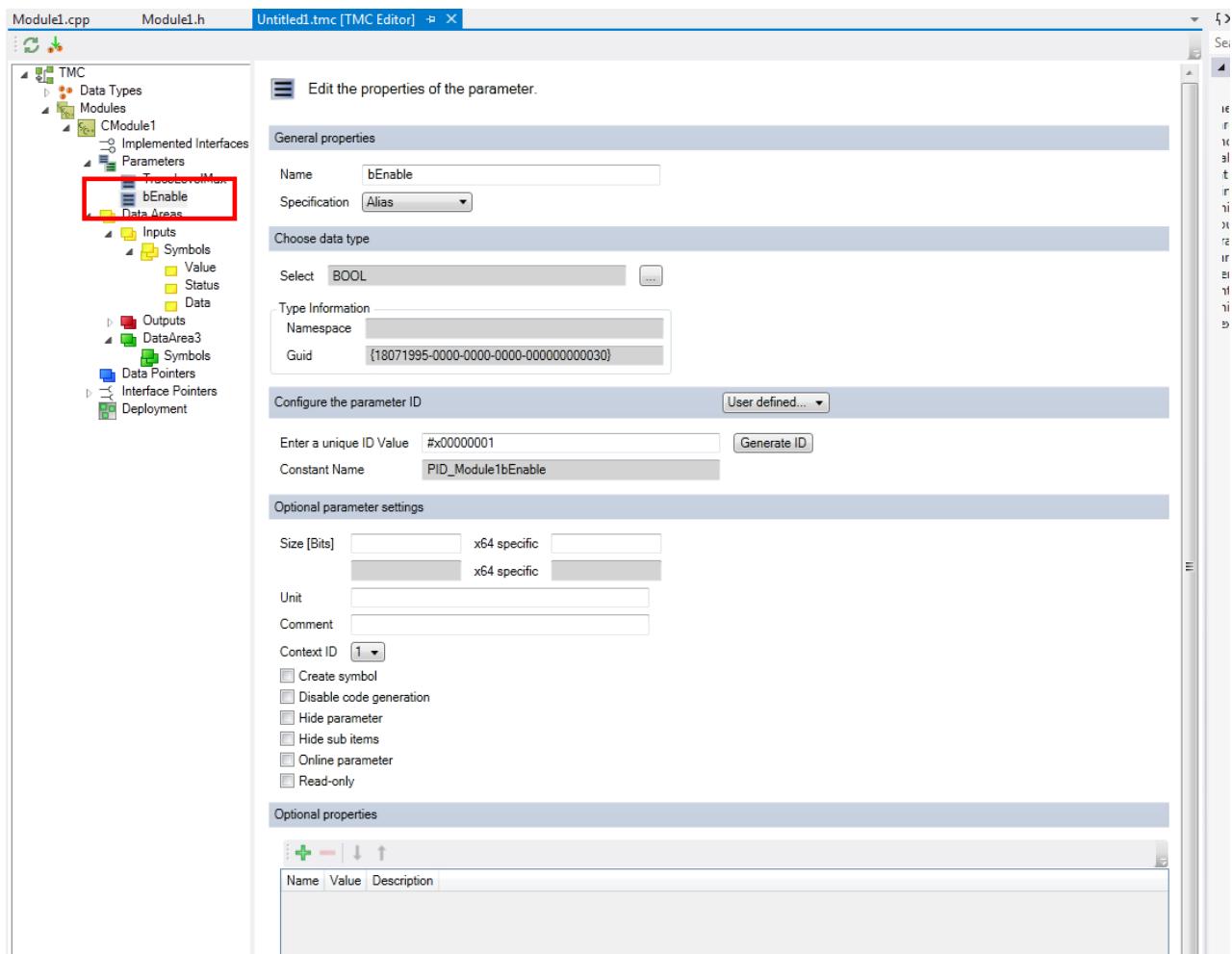
Beachten Sie die verschiedenen Zustandsübergänge Ihrer Zustandsmaschine [▶ 38]:



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 \[► 103\]](#).

**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:** Wechselt 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.

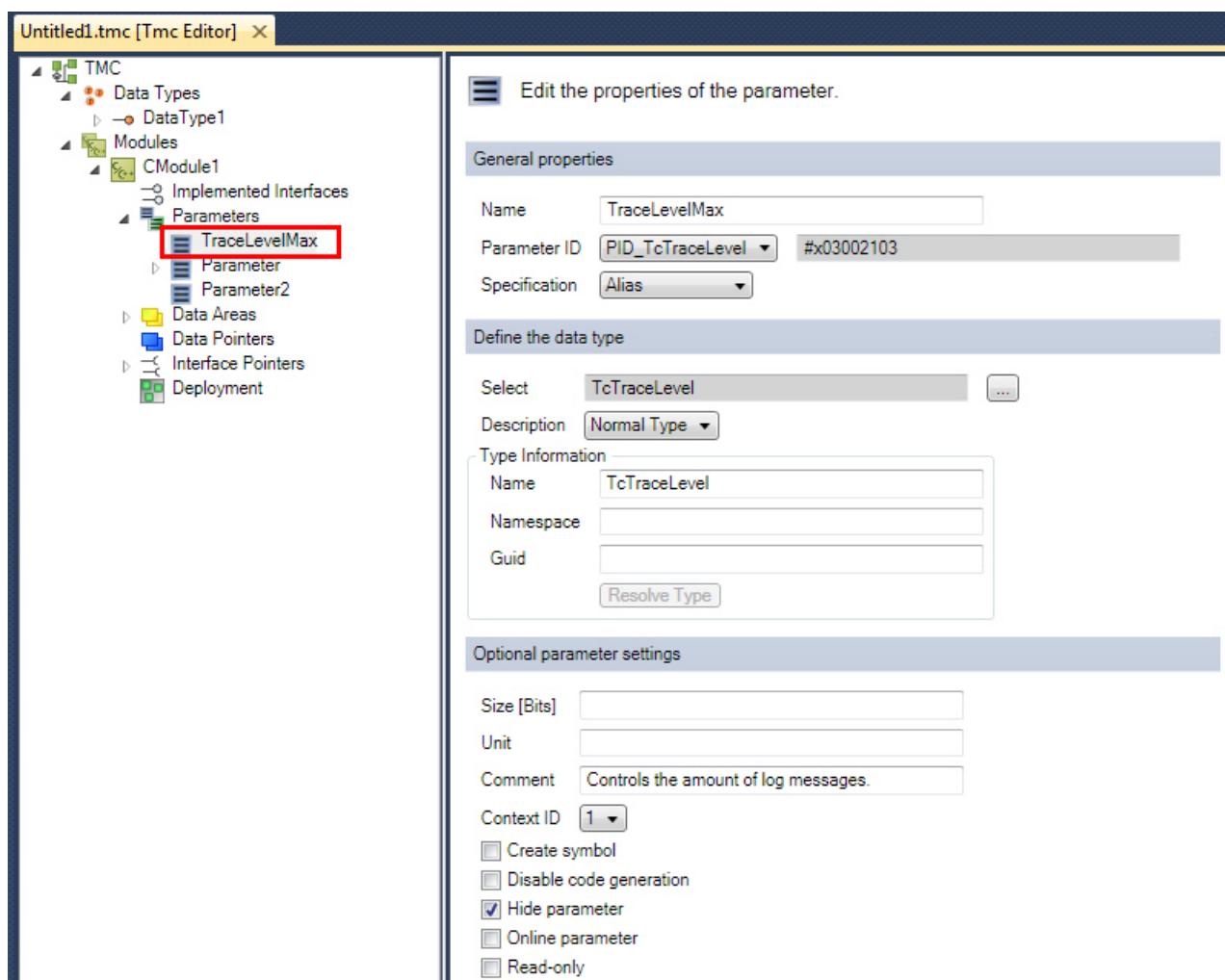
#### 11.3.4.2.3 TraceLevelMax

**TraceLevelMax:** Parameter der den Tracelevel festlegt.

Dies ist ein vordefinierter Parameter, den die meisten TwinCAT-Modulvorlagen bereitstellen (außer die leere TwinCAT-Modulvorlage).

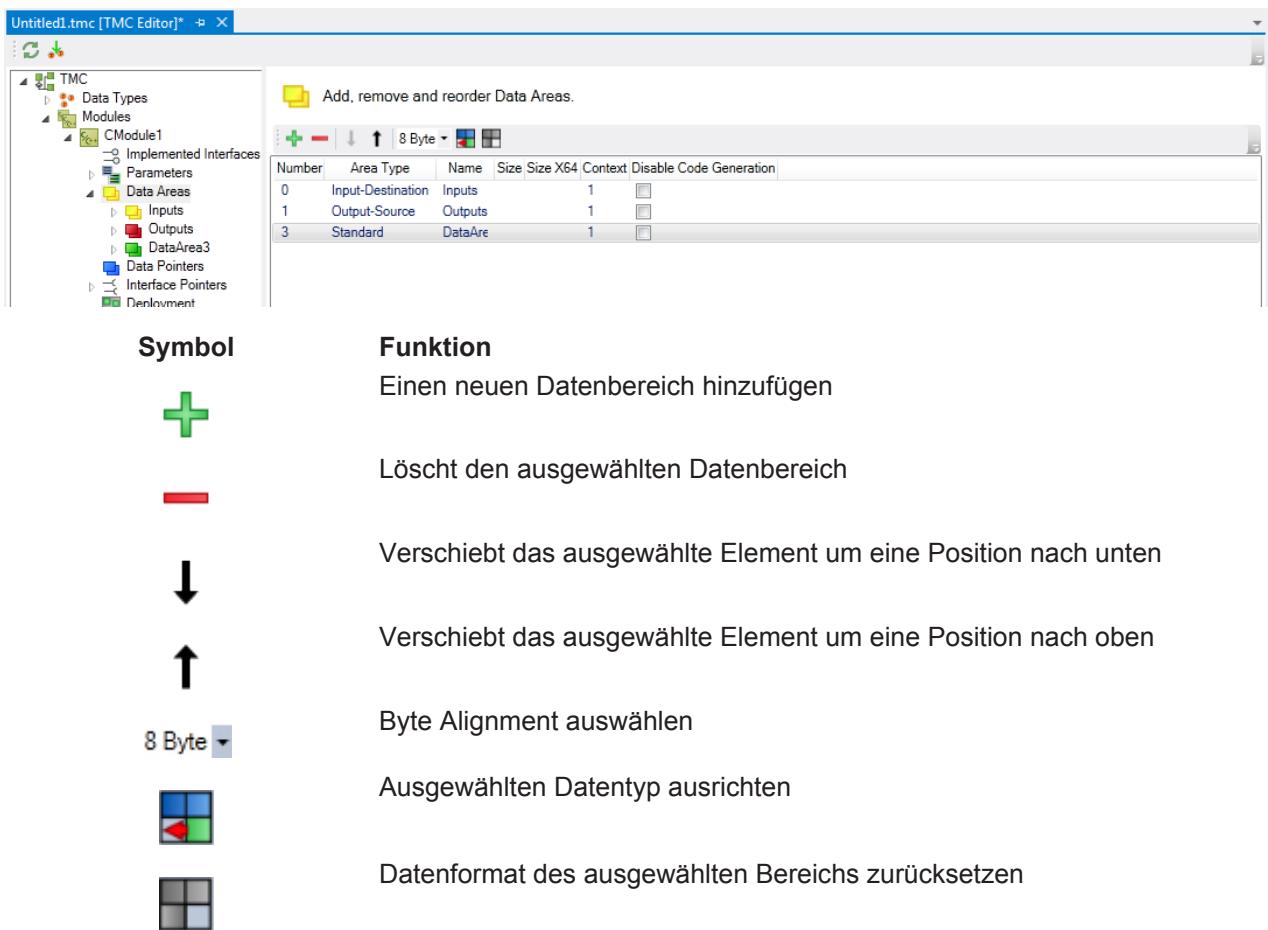
Die Einstellungen dieses Parameters sollten nicht geändert werden.

Siehe [Modul-Nachrichten zum Engineering \(Logging / Tracing\) \[▶ 196\]](#)



### 11.3.4.3 Datenbereiche

**Data Areas:** Dialog zum Bearbeiten der Datenbereiche Ihres Moduls



Hinweis

#### Rekursion bei Festlegung eines Alignment

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

**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.

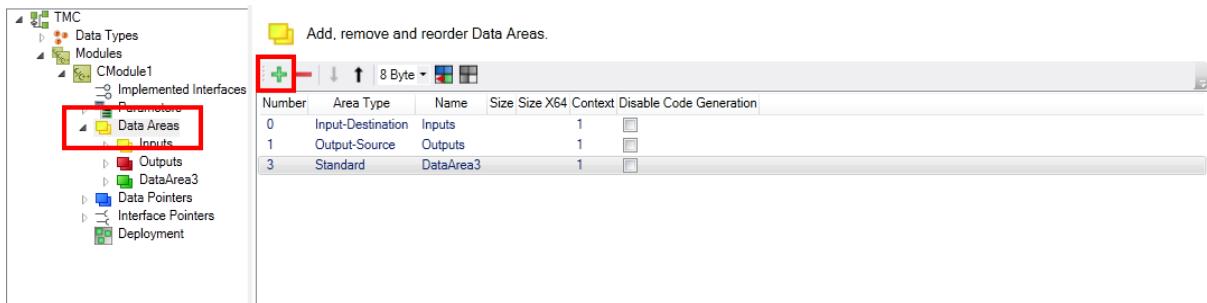
Dieser Artikel beschreibt:

- Einen neuen Datenbereich in der TMC-Datei erstellen
- Neue Variablen in einen Datenbereich erzeugen [▶ 117]

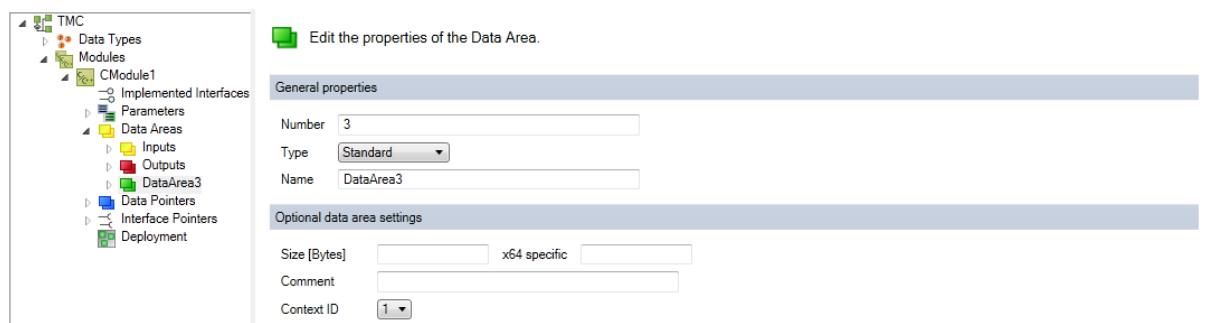
- Z.B. den Namen oder Datentyp [▶ 117] von in der TMC-Datei bestehenden Variablen bearbeiten
- In der TMC-Datei bestehende Variablen löschen [▶ 118]

### Einen neuen Datenbereich erzeugen

1. Nach dem Starten des TMC Editors den Knoten „Data Areas“ des Moduls auswählen
2. Durch Klicken auf die + Schaltfläche wird ein neuer Datenbereich erzeugt



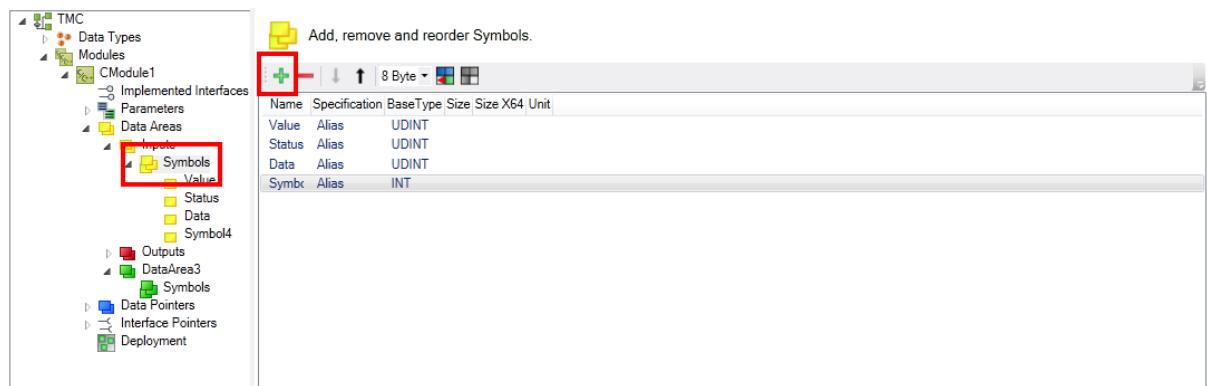
3. Um die Eigenschaften des Datenbereichs zu erhalten, auf die Tabelle doppelklicken oder auf den Knoten klicken



4. Den Datenbereich umbenennen

### Eine neue Variable erzeugen

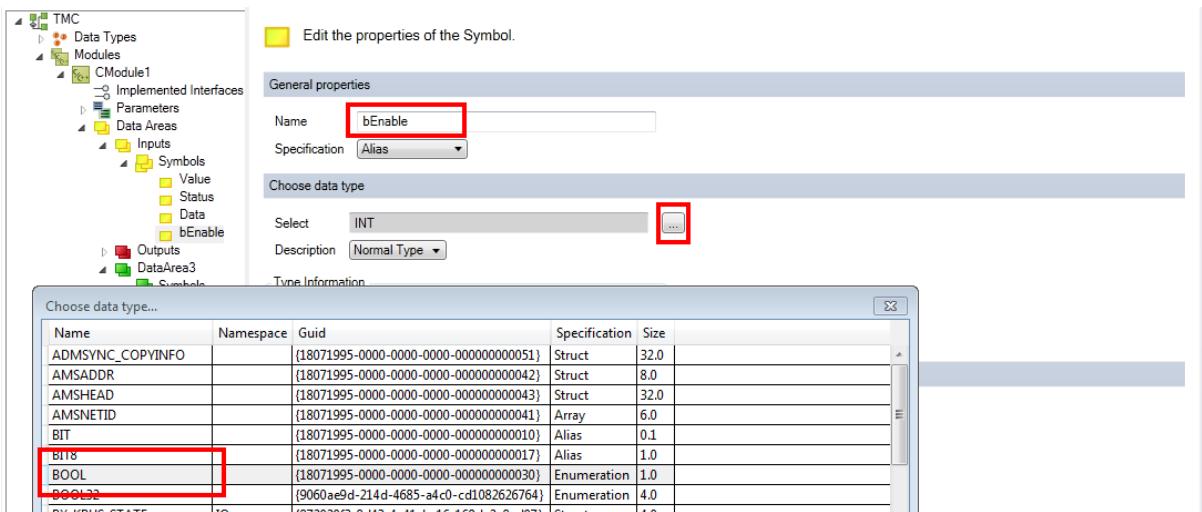
5. Den Unterknoten „Symbols“ des Datenbereichs auswählen.
6. Dieser Datenbereich kann mit einem Klick auf die „+“ Schaltfläche um eine neue Variable erweitert werden. Daraufhin wird „Symbol4“ als neuer Eintrag aufgeführt.



### Name oder Datentyp von bestehenden Variablen bearbeiten

7. Wählen Sie den Unterknoten „Symbol4“ oder doppelklicken Sie auf die Zeile. Die Variableneigenschaften werden eingeblendet

8. Geben Sie einen neuen Namen, z.B. „bEnableJob“, ein und ändern den Typ in BOOL

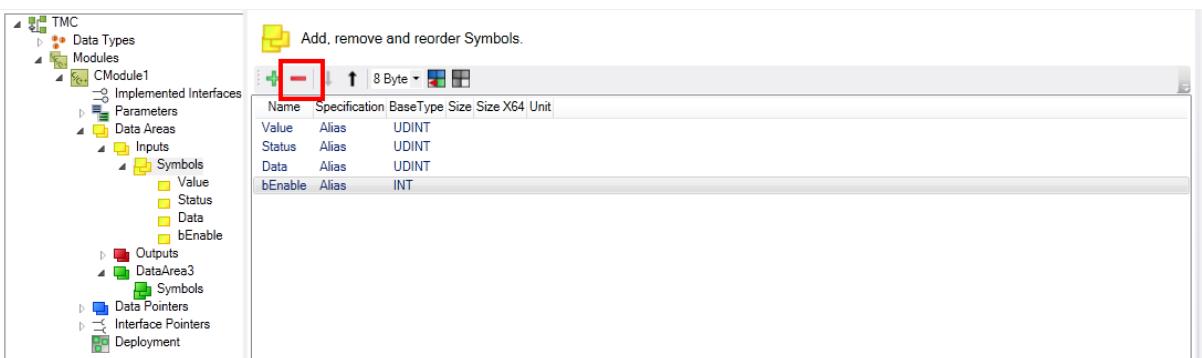


⇒ Letztendlich wird die neue Variable „bEnableJob“ im Datenbereich „Input“ erzeugt.

#### **HINWEIS! Nicht vergessen, den TMC Code Generator erneut auszuführen**

#### Bestehende Variablen löschen

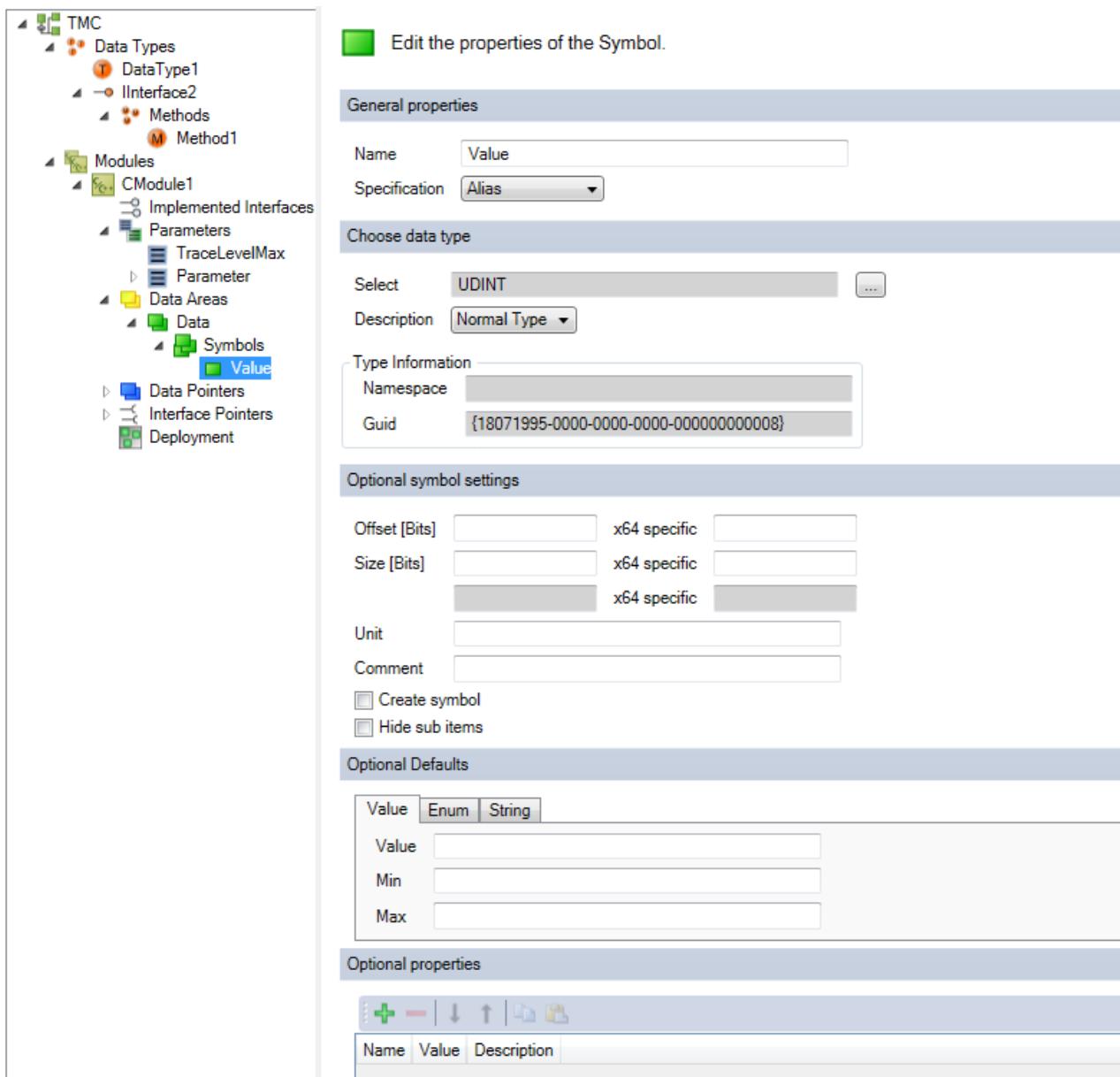
- Zum Löschen bestehender Variablen im Datenbereich die Variable auswählen und dann auf das Löschen-Symbol klicken:  
In diesem Beispiel „MachineStatus1“ auswählen und auf „Delete symbol“ klicken



- Den TMC Code Generator erneut ausführen

#### 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 \[▶ 308\]](#))
- 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

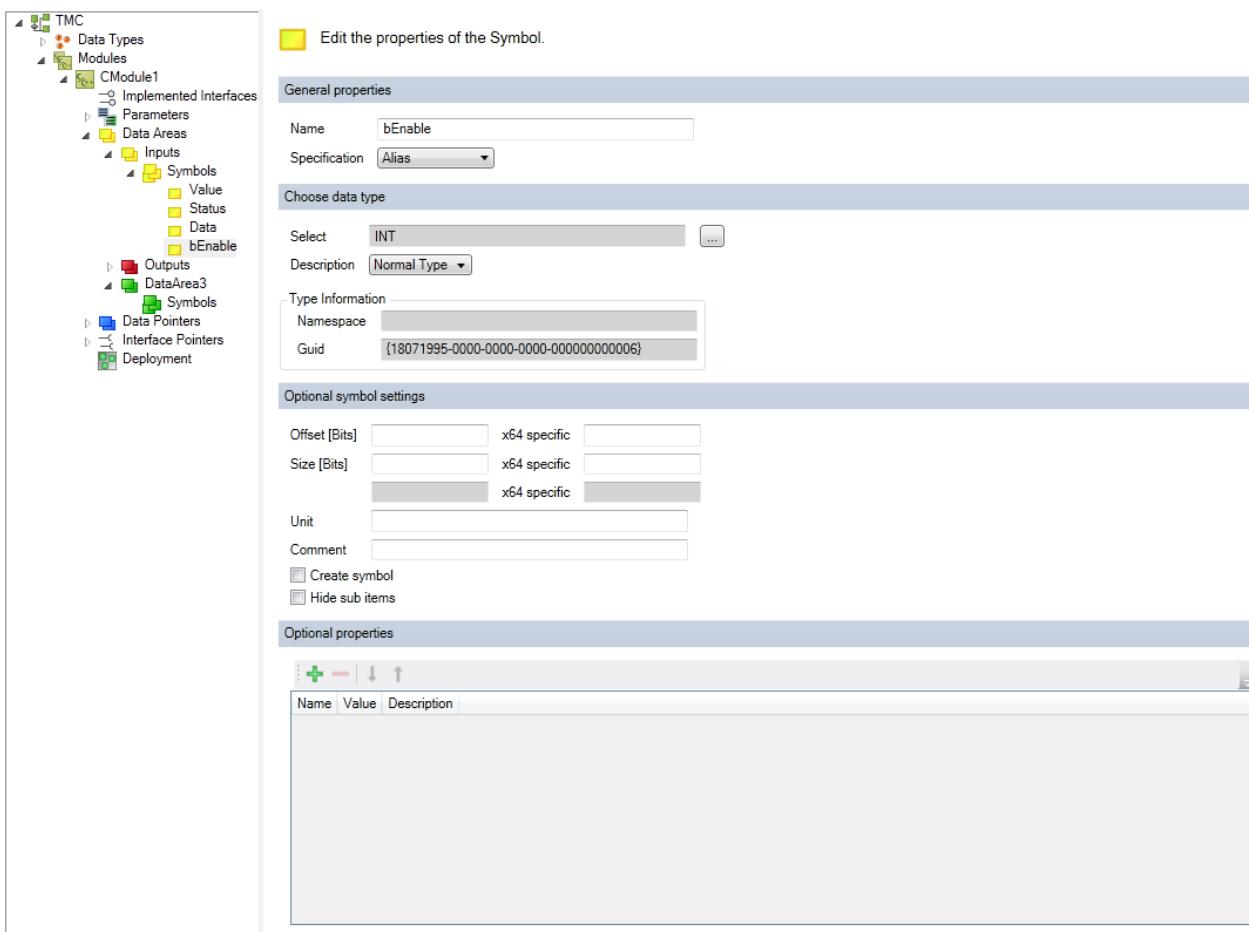
Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Datenbereichs.

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 \[► 94\]](#)

### 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 anderes Offset festgelegt werden.

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

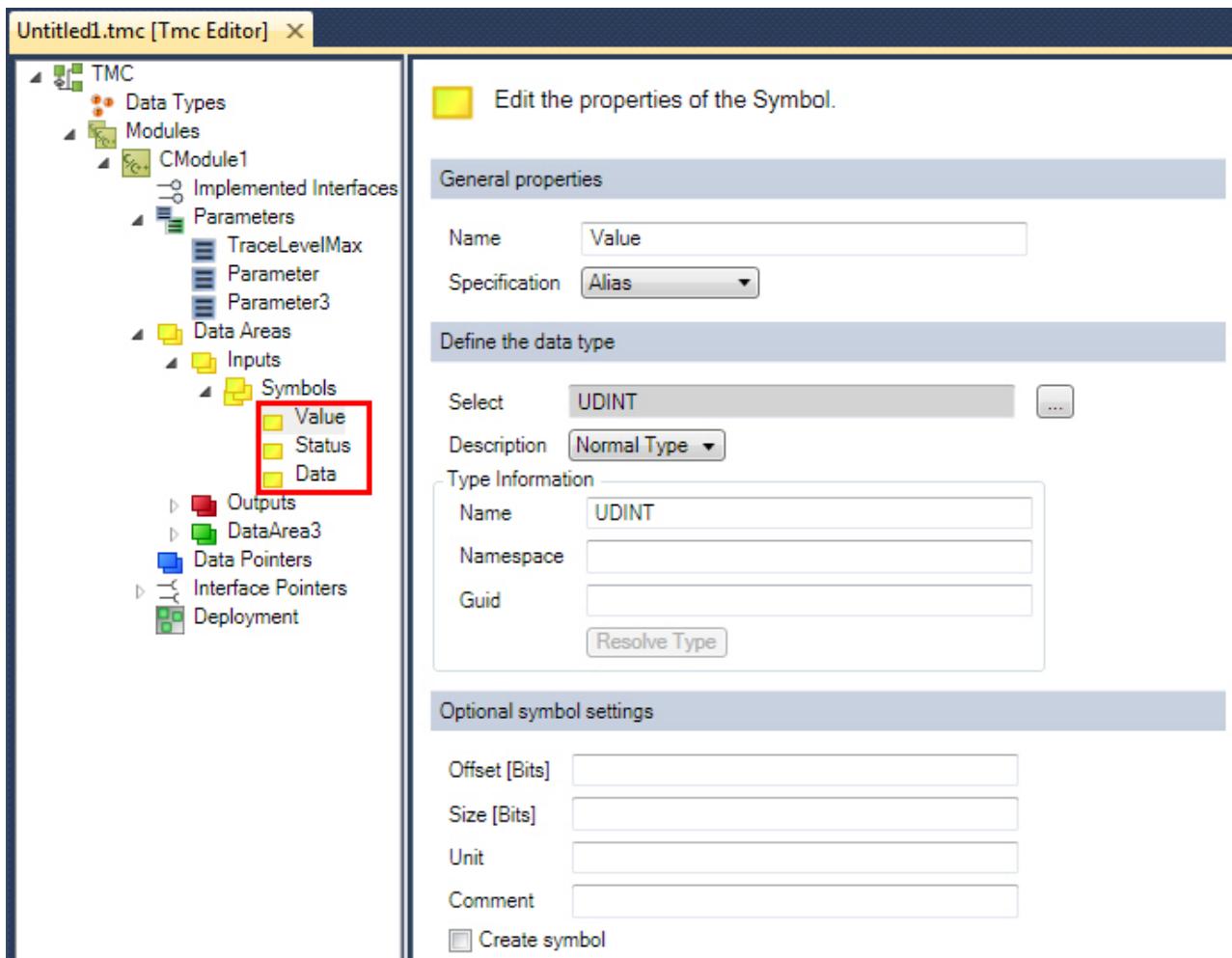
**Comment:** Optionaler Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

**Create Symbol:** Standardeinstellung für ADS-Symbolerstellung

**Hide sub items:** Wenn 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 TwinCAT Module Class Editor - Data Areas Symbols Properties

**Data Areas Symbols Properties:** Dialog zum Bearbeiten der Datenbereichssymboleigenschaften



#### 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

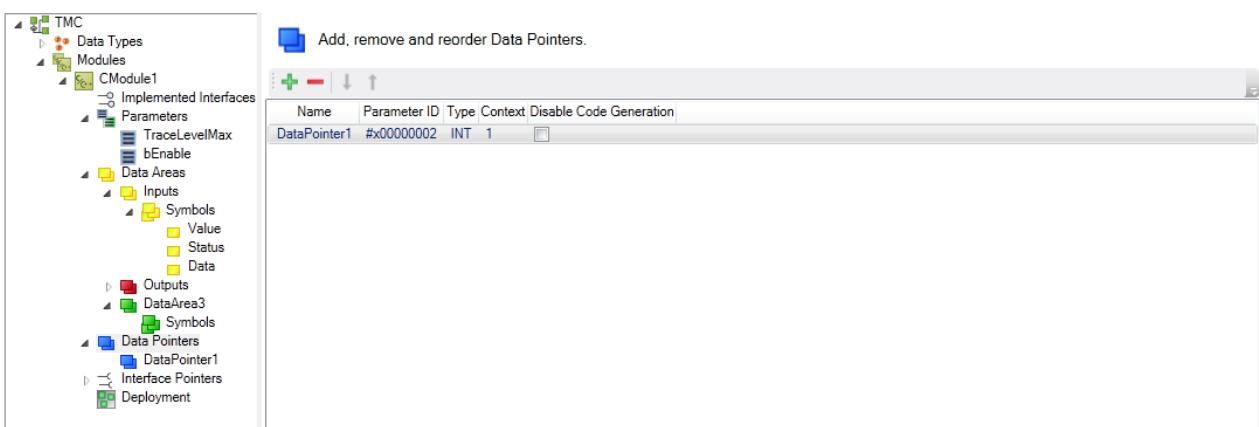
**Unit:** Optional

**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

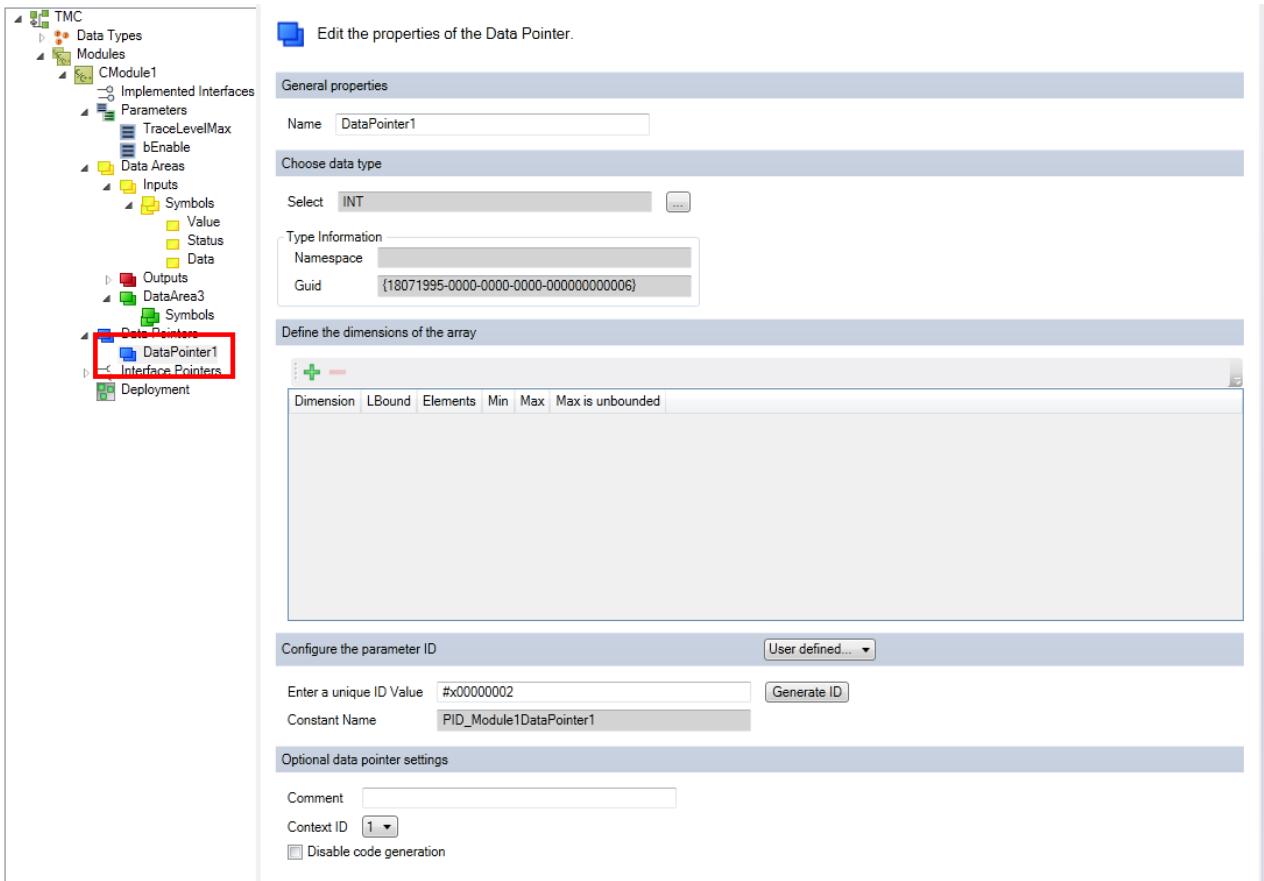
**Type:** Definiert den Zeigertyp

**Context:** Zeigt die Kontext-ID an

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

### 11.3.4.4.1 Datenzeigereigenschaften

**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 Array definieren

Siehe [hier \[▶ 97\]](#).

#### Die Parameter-ID konfigurieren

**Enter a unique ID Value:** Einen eindeutigen ID-Wert eingeben. Siehe [Parameter \[▶ 103\]](#).

**Constant Name:** Quellcodename der Parameter-ID

#### Optionale Datenzeigereinstellungen

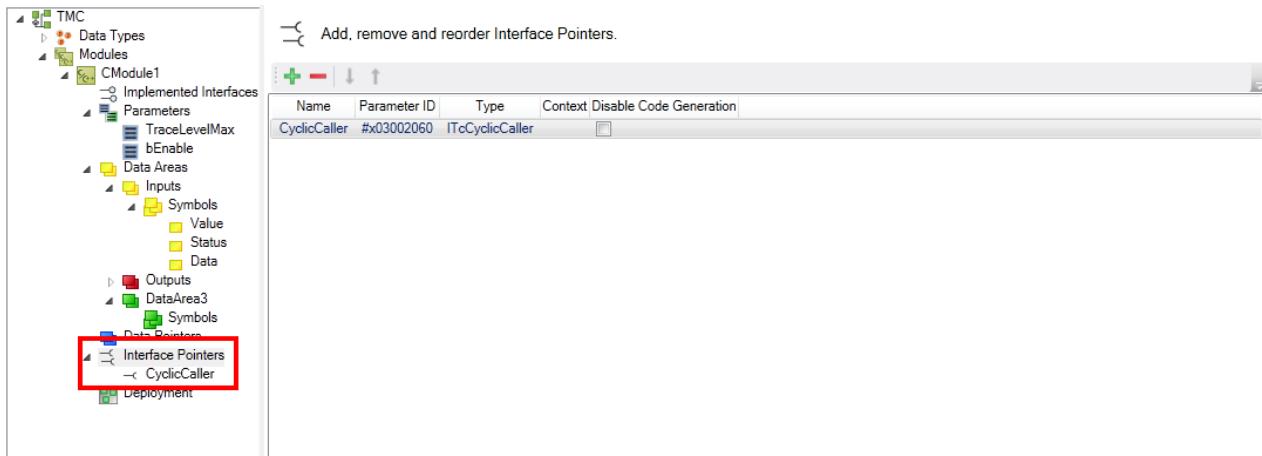
**Comment:** Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

**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

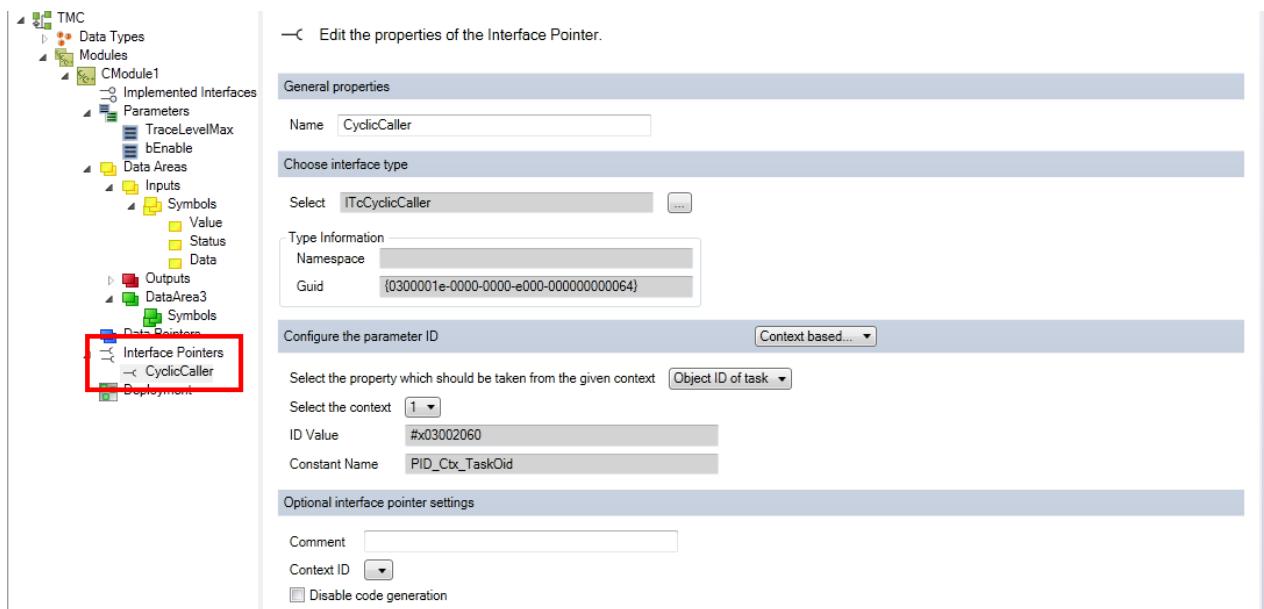
**Type:** Typ des Schnittstellenzeigers

**Context:** Kontext der Schnittstelle

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

#### 11.3.4.5.1 Eigenschaften des Schnittstellenzeigers

**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 \[► 103\]](#).

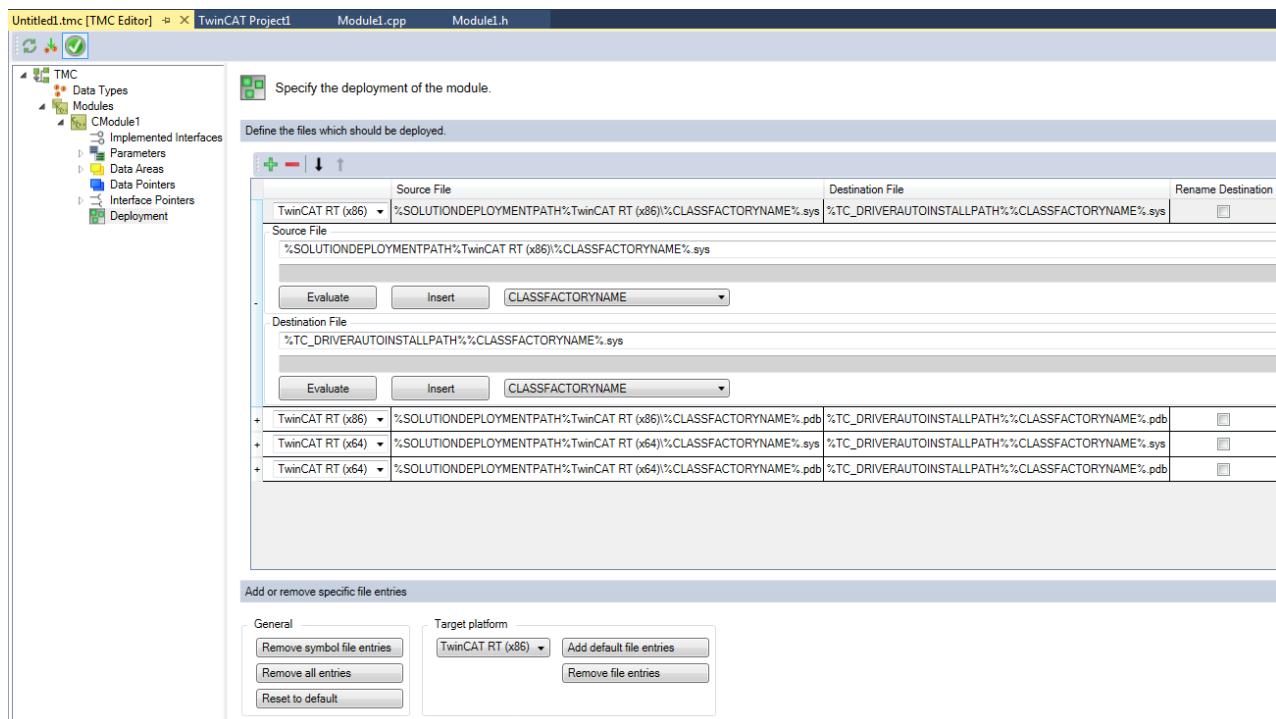
**Comment:** Optional

**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



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:** Zielfile wird umbenannt, bevor neue Datei übertragen wird.

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

	<b>Windows 10</b> Für Windows 10 Zielsysteme wird die Option „Rename Destination“ benötigt.
Hinweis	

**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 Taste	Defaultwert
	\HKLM\Software\Beckhoff\Twin-CAT3	
%TC_INSTALLPATH%	InstallDir	C:\TwinCAT\3.x\
%TC_CONFIGPATH%	ConfigDir	C:\TwinCAT\3.x\Config\
%TC_TARGETPATH%	TargetDir	C:\TwinCAT\3.x\Target\
%TC_SYSTEMPATH%	SystemDir	C:\TwinCAT\3.x\System\
%TC_BOOTPRJPATH%	BootDir	C:\TwinCAT\3.x\Boot\
%TC_RESOURCEPATH%	ResourceDir	C:\TwinCAT\3.x\Target\Resource\
%TC_REPOSITORYPATH%	RepositoryDir	C:\TwinCAT\3.x\Repository\
%TC_DRIVERPATH%	DriverDir	C:\TwinCAT\3.x\Driver\
%TC_DRIVERAUTINSTALLPATH%	DriverAutoInstallDir	C:\TwinCAT\3.x\Driver\AutoInstall\
%TC_SYSSRVEXE PATH%		C:\TwinCAT\3.x\SDK\Bin\TwinCAT UM (x86)\
%CLASSFACTORYNAME%		<Name der Class Factory>

(wobei „x“ durch die installierte TwinCAT-Version ersetzt wird)

## 11.4 TwinCAT Module Instance Configurator

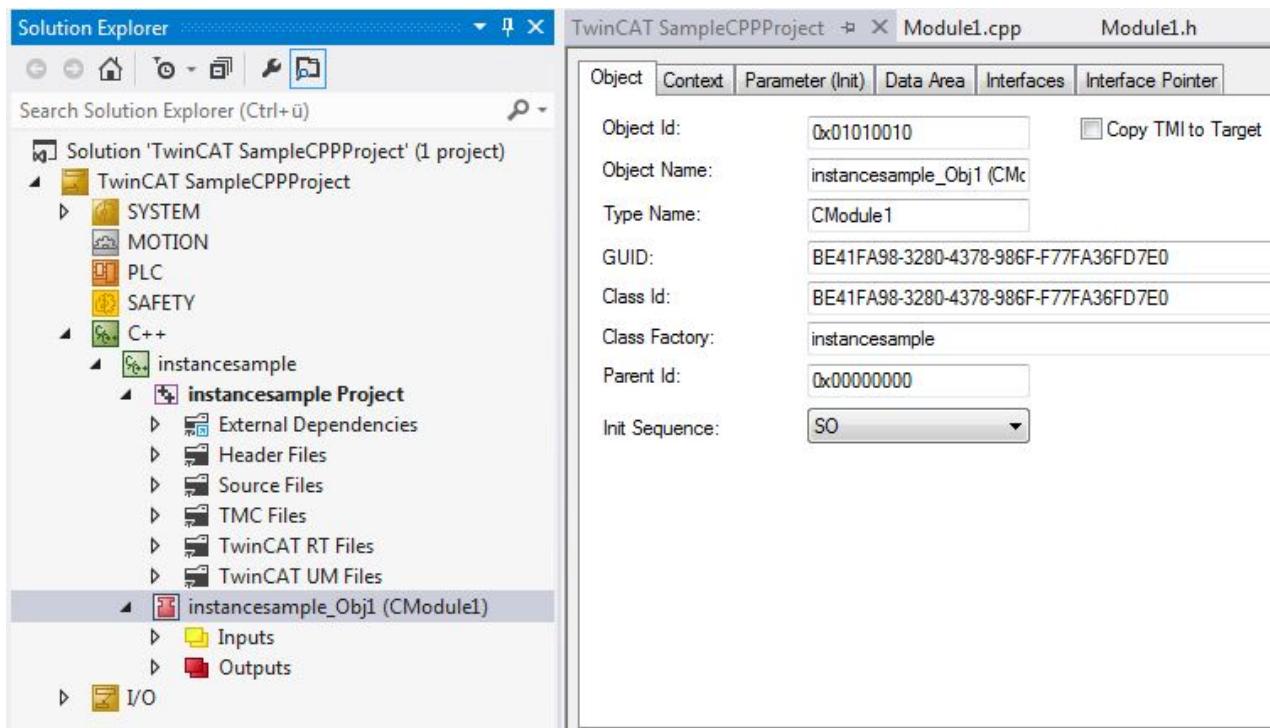
Der oben beschriebene TwinCAT 3 Module Class (TMC) Editor definiert Treiber auf Klassenebene. Diese werden instanziert und müssen über den TwinCAT 3 Instance Configurator konfiguriert werden.

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

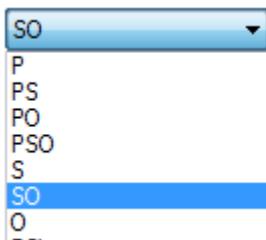
Instanzen von C++ Klassen werden durch Rechtsklick auf den C++ Projektordner erstellt, siehe Schnellstart. In diesem Kapitel wird die Konfiguration dieser Instanzen ausführlich beschrieben.

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 hier [38] für 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.

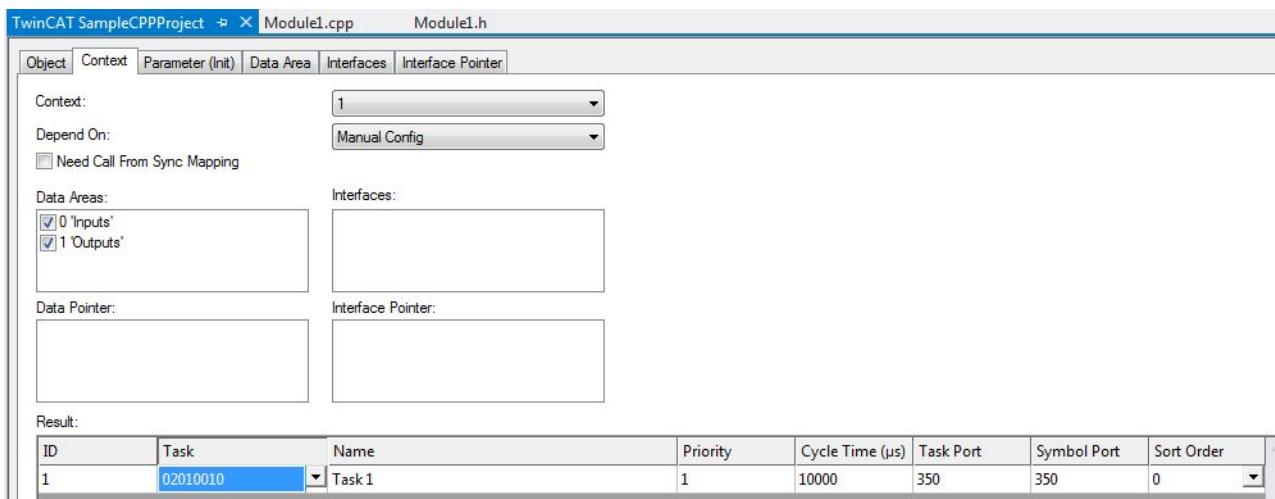
Der Name einer Init Sequenz besteht aus dem Kurznamen der TcCOM Zustandsmaschine. Wenn der Kurzname eines Zustands (I,P,S,O) im Namen der Init Sequenz enthalten ist, dann werden die Module in diesem Zustand warten, bis alle anderen Module zumindest diesen Zustand erreicht haben. Beim nächsten Übergang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

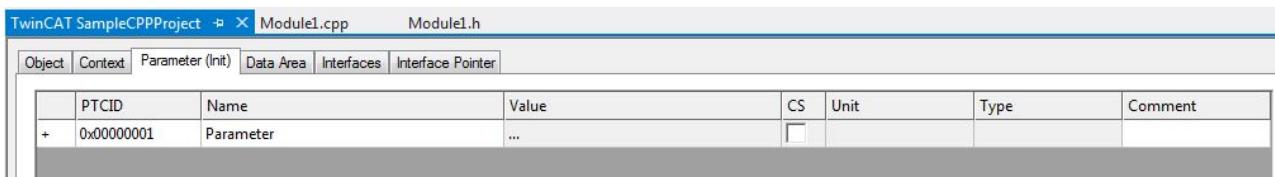
- **Copy TMI to target:** Die TMI (TwinCAT Module Instance) Datei generieren und an das Ziel übergeben.

## 11.4.2 Kontext



- **Context:** Den zu konfigurierenden Kontext auswählen (siehe TMC Editor für Hinzufügen verschiedener Kontexte).
- **HINWEIS! 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:** Liest der IDs, die konfiguriert werden müssen. Zumindest muss der Kontext („Task“-Spalte) dem 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 [hier \[▶ 103\]](#) beschrieben konfiguriert.

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

## 11.4.4 Data Area

TwinCAT SampleCPPProject Module1.cpp Module1.h								
Object	Context	Parameter (Init)	Data Area	Interfaces	Interface Pointer			
-	0		Inputs	InputDst	12	<input type="checkbox"/>	3 Symbols	
			Value	UDINT	4.0 (Offs: 0.0)	<input type="checkbox"/>		
			Status	UDINT	4.0 (Offs: 4.0)	<input type="checkbox"/>		
			Data	UDINT	4.0 (Offs: 8.0)	<input type="checkbox"/>		
-	1		Outputs	OutputSrc	12	<input type="checkbox"/>	3 Symbols	
			Value	UDINT	4.0 (Offs: 0.0)	<input type="checkbox"/>		
			Control	UDINT	4.0 (Offs: 4.0)	<input type="checkbox"/>		
			Data	UDINT	4.0 (Offs: 8.0)	<input type="checkbox"/>		

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

TwinCAT SampleCPPProject Module1.cpp Module1.h							
Object	Context	Parameter (Init)	Data Area	Interfaces	Interface Pointer		
	IID			Name			
	00000012-0000-0000-E000-000000000064			ITComObject			
	03000010-0000-0000-E000-000000000064			ITcCyclic			
	03000012-0000-0000-E000-000000000064			ITcADI			
	03000018-0000-0000-E000-000000000064			ITcWatchSource			

Liste aller implementierter Schnittstellen (wie in TMC definiert).

## 11.4.6 Schnittstellenzeiger

TwinCAT SampleCPPProject Module1.cpp Module1.h					
Object	Context	Parameter (Init)	Data Area	Interfaces	Interface Pointer
PTCID	Name	OTCID	Object Name	IID	Type
0x03002060	CyclicCaller	0201010	Task 1	0300001E-0000-0000...	ITcCyclicCaller

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 [hier \[▶ 103\]](#) beschrieben konfiguriert.

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

## 11.4.7 Datenzeiger

Object		Context	Parameter (Init)	Data Area	Interfaces	Interface Pointer	Data Pointer			
PTCID	Name	OTCID		Object Name	Area No	Offset	Size			
0x00000003	DataPointer1	0x00000000			0	0	0			

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 [hier \[▶ 103\]](#) beschrieben konfiguriert.

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

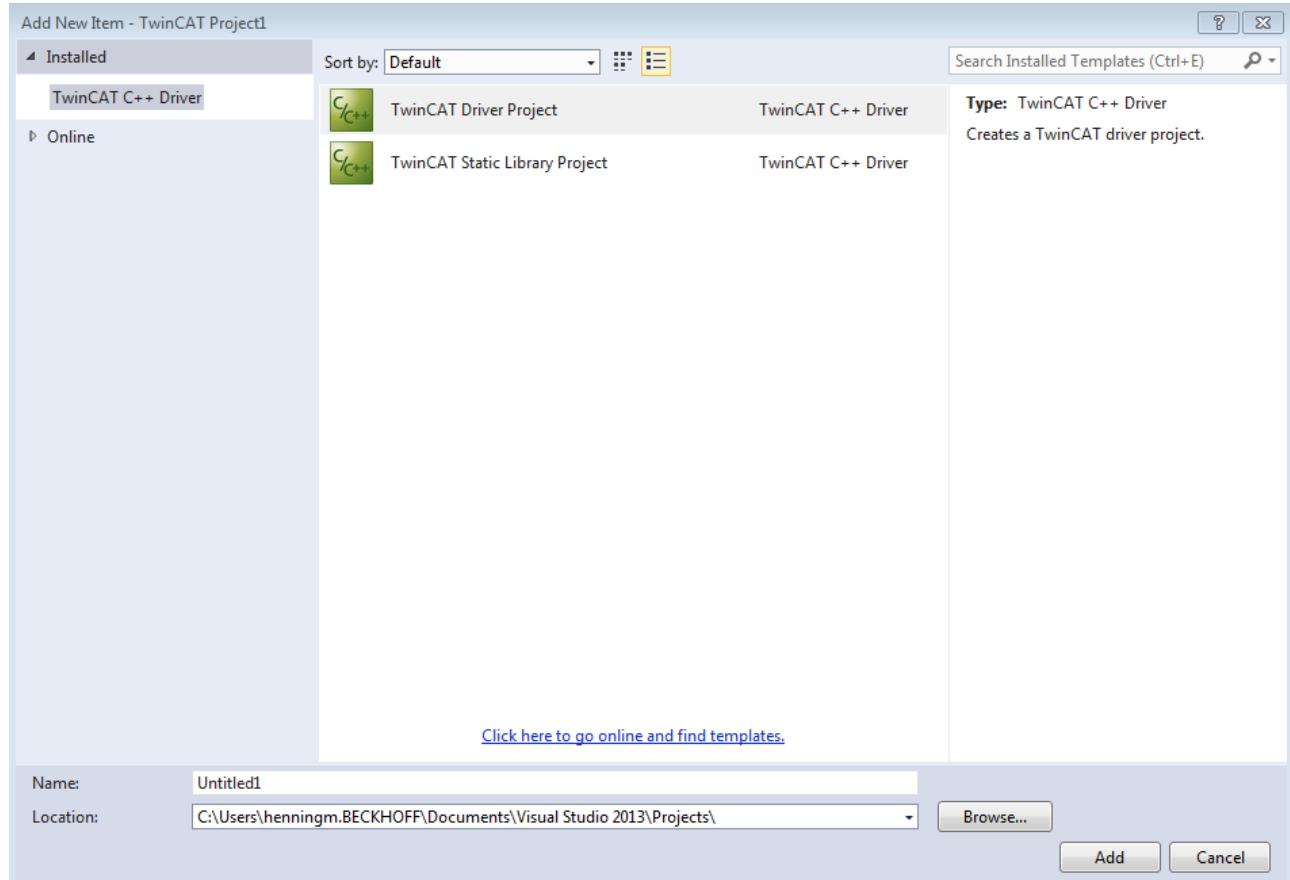
## 11.5 Kunden-spezifische Projekt-Templates

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).

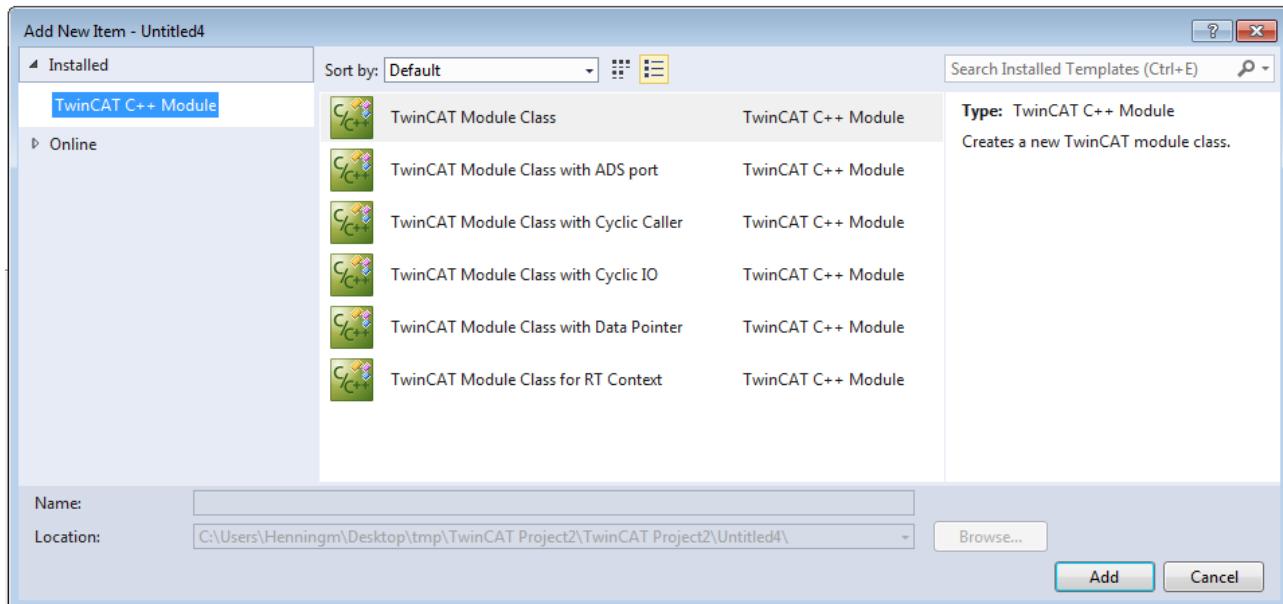
Dieser Abschnitt der Dokumentation beschreibt, wie Kunden eigene Projekt-Templates einbringen können.

### 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 einen TwinCAT Treiber. Dieses Rahmengerüst dient der Registrierung eines TwinCAT Treibers im System. Die eigentliche Funktion des Treibers wird in TwinCAT Modulen implementiert.



Der TwinCAT Class Wizard wird automatisch nach dem Anlegen eines neuen Treiberprojektes gestartet, um das erste TwinCAT Treiber 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.x\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 „TriggerAddModule“ 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 Zielfile 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 Attribut 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 bereit gestellt werden:
 

```
<TargetFile symbolName="CustomerFileName">[ !output SYMBOLNAME ].txt</TargetFile>
```
3. Wichtige Symbole werden vom System selber 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}

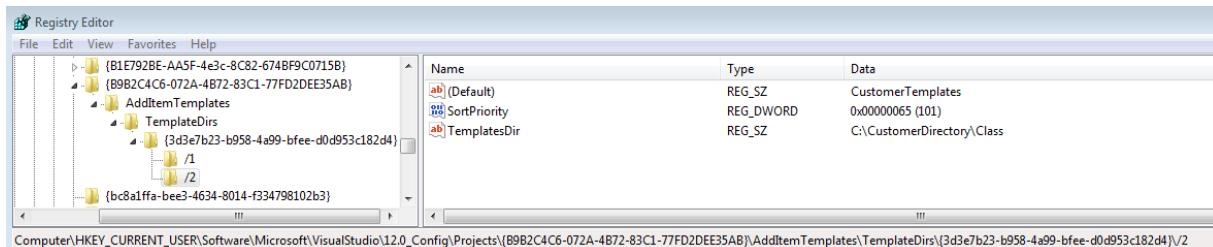
## 11.5.4 Hinweise zum Handling

### Template in Kunden-spezifischem Verzeichnis

Templates können auch außerhalb vom TwinCAT üblichen Verzeichnis abgelegt werden.

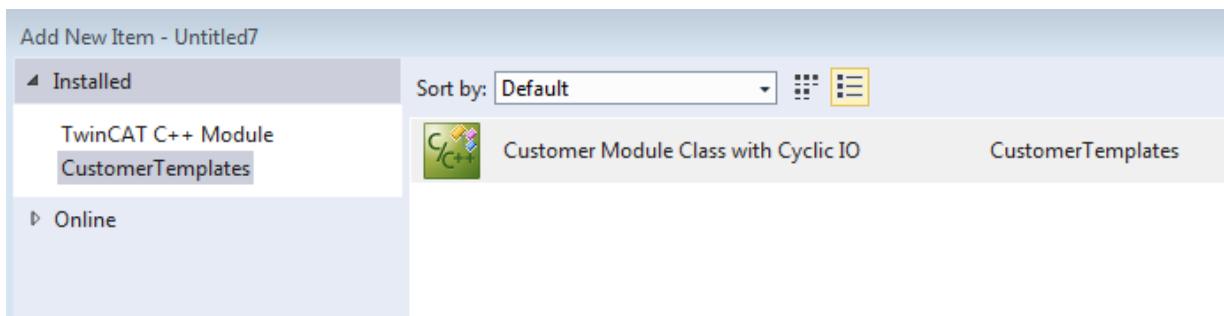
- Erweitern Sie in der Registry den Suchpfad (hier V12.0, also für VS 2013), in dem der Knoten /2 angelegt wird:

**Registry Key:** HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\12.0\_Config\Projects\{B9B2C4C6-072A-4B72-83C1-77FD2DEE35AB}\AddItemTemplates\TemplateDirs\{3d3e7b23-b958-4a99-bfee-d0d953c182d4}\



- Erhöhen Sie die SortPriority.
- 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.
- 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 Einstiegsplatz: <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“.

- Nehmen Sie ein existierendes Modul-Template als Kopiervorlage  
In C:\TwinCAT\3.x\Components\Base\CppTemplate\Templates

CustomerModuleCyclicIO	20.08.2015 10:29	File folder
TcDriverWizard	21.07.2015 13:02	File folder
TcModuleAdsPort	21.07.2015 13:02	File folder
TcModuleCyclicCaller	21.07.2015 13:02	File folder
TcModuleCyclicIO	21.07.2015 13:02	File folder
TcModuleDataPointer	21.07.2015 13:02	File folder
TcModuleEmpty	21.07.2015 13:02	File folder
TcModuleRT	21.07.2015 13:02	File folder
TcStaticLibrary	21.07.2015 13:02	File folder

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

CustomerModuleCyclicIOConfig.xml	10.06.2015 11:14	XML Document	1 KB
TcModuleCyclicIO.cpp	10.06.2015 11:14	C++ Source	7 KB
TcModuleCyclicIO.h	10.06.2015 11:14	C/C++ Header	2 KB
TcModuleCyclicIO.tmc	10.06.2015 11:14	TMC File	5 KB

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

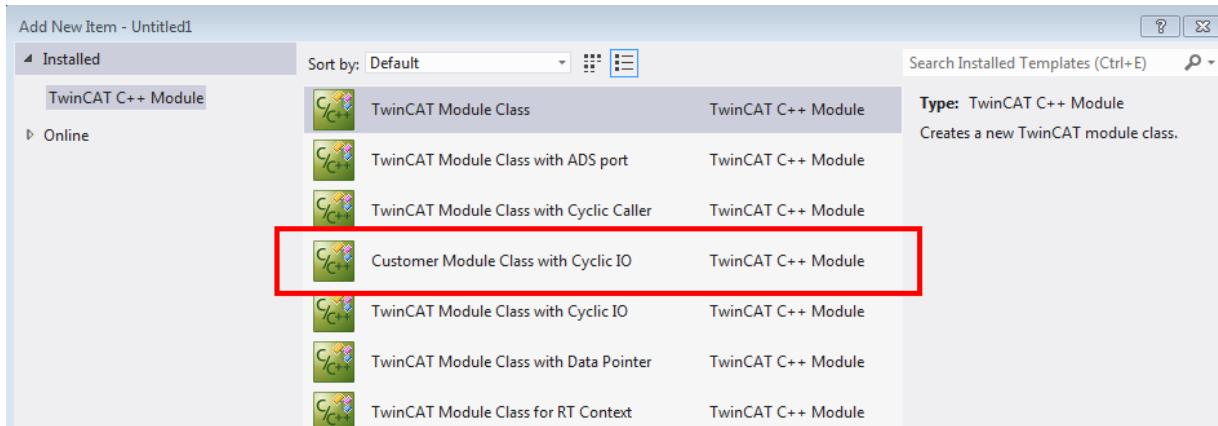
CustomerModuleCyclicIOWizard.ico	10.06.2015 11:14	Icon	265 KB
CustomerModuleCyclicIOWizard.vsdir	20.08.2015 10:35	VSDIR File	1 KB
CustomerModuleCyclicIOWizard.vsz	10.06.2015 11:14	Visual Studio Wiza...	1 KB
TcModuleAdsPortWizard.ico	10.06.2015 11:14	Icon	265 KB
TcModuleAdsPortWizard.vsdir	10.06.2015 11:14	VSDIR File	1 KB
TcModuleAdsPortWizard.vsz	10.06.2015 11:14	Visual Studio Wiza...	1 KB
TcModuleCyclicCallerWizard.ico	10.06.2015 11:14	Icon	265 KB

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.

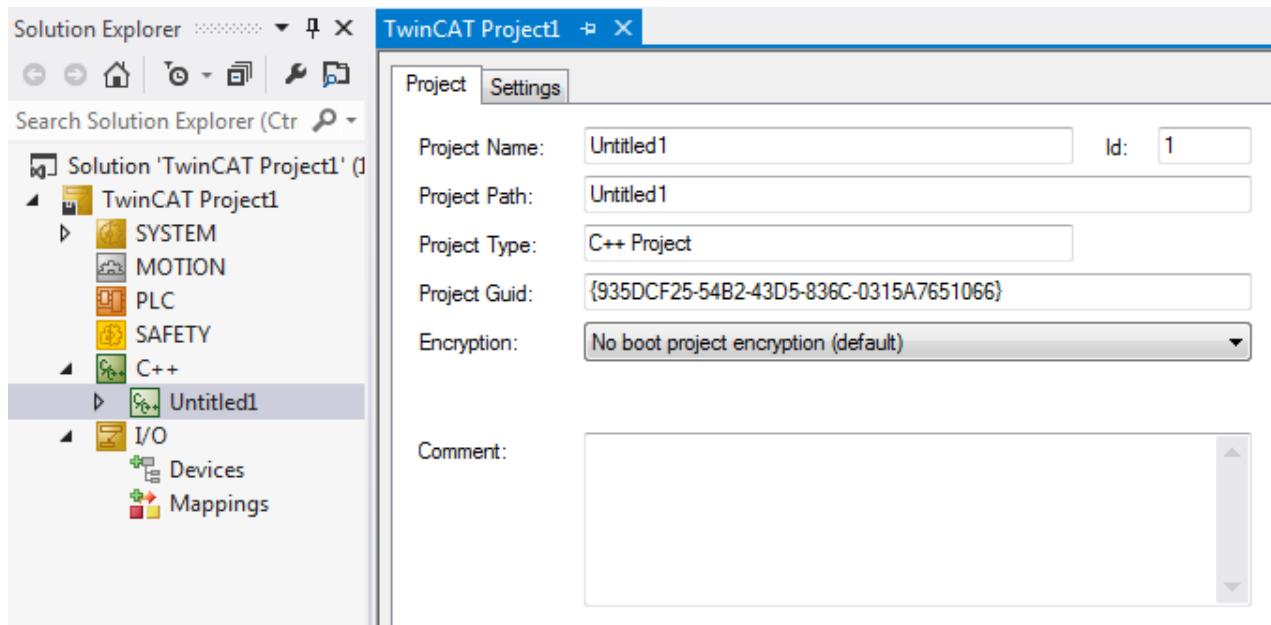
## 12 Programmierreferenz

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

Es besteht eine Vielzahl an [C++ Beispielen \[▶ 216\]](#), die wertvolle Informationen über die Handhabung der Module und Schnittstellen beinhalten.

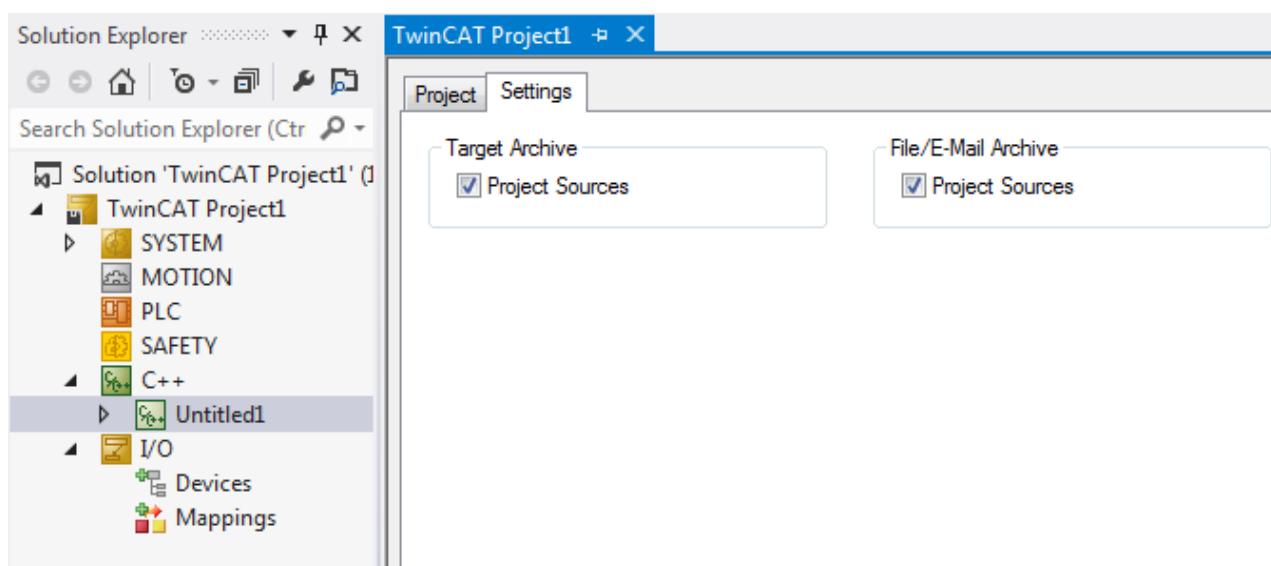
### C++ Projekteigenschaften

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



Ein Umbenennen ist hier nicht möglich (vgl. [Umbenennen von TwinCAT-C++ Projekten \[▶ 202\]](#))

Die Encryption ist für C++ Projekte nicht implementiert.



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

Bei Deselektion werden entsprechend leere Archive erzeugt.

## 12.1 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
<b>Engineering / XAE</b>		
*.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 <a href="#">TwinCAT Module Class Editor (TMC) [► 79]</a>
*.rc	Resourcendatei	Siehe <a href="#">Setzen von Version/ Herstellerinformationen [► 206]</a>
*.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%\CustomConfig\Modules\*	Veröffentlichtes TwinCAT Treiberpaket normalerweise C:\TwinCAT\3.x\CustomConfig\Modules\*	Siehe <a href="#">Module exportieren [► 43]</a>
<b>Laufzeit / XAR</b>		
%TC_BOOTPRJPATH%\CurrentConfig\*	Derzeitige Konfigurationssetup normalerweise C:\TwinCAT\3.x\Boot	
%TC_DRIVERAUTINSTALLPATH% \*.sys/pdb	Kompilierter, plattformspezifischer Treiber <ul style="list-style-type: none"> <li>• C:\TwinCAT\3.x\Driver\AutoInstall (System geladen)</li> <li>• C:\TwinCAT\3.x\Driver\AutoLoad (TcLoader geladen)</li> </ul>	
%TC_BOOTPRJPATH% \TM\OBJECTID.tmi	TwinCAT Module Instance Datei Beschreibt Variablen des Treibers Dateiname lautet „ObjectID.tmi“ Normalerweise C:\TwinCAT\3.x\Boot\TMI\OTCID.tmi	
<b>Temporäre Dateien</b>		
*.sdf	IntelliSense Datenbank	
*.suo / *.v12.suo	Benutzerspezifische und Visual Studio spezifische Dateien	

Datei	Beschreibung	Weitere Informationen
*.tsproj.bak	Automatisch generierte Sicherungsdatei von tsproj	
ipch/	Für vorkompilierter Header erstelltes Zwischen-Verzeichnis	

## 12.1.1 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 dedizierte Einstellung der Build Konfiguration.

Bei Auswahl von „Build“ oder ein „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 in C:\TwinCAT\3.1\ sdk\\_products\ z.B. in C:\TwinCAT\3.1\ sdk\\_products\TwinCAT RT (x64)\Debug\<ProjectName>
3. Der Linkvorgang legt danach die .sys/.pdb Datei ebenfalls architektur-spezifisch in C:\TwinCAT \3.1\ sdk\\_products\ z.B. in C:\TwinCAT\3.1\ sdk\\_products\TwinCAT RT (x64)\Debug\
4. Es wird eine Kopie der .sys/.pdb in das Unterverzeichnis \_Deployment/ des Projekt-Verzeichnisses gelegt. z.B. nach Projektverzeichnis/\_Deployment/TwinCAT RT (x64)\
5. Ein Druck auf den „Activate Configuration“ Button führt dazu, dass .sys/.pdb von \_Deployment/ des Projekt-Verzeichnisses auf das Zielsystem übertragen werden (ggf. ist das eine lokale Kopie)

## 12.2 Limitierungen

TwinCAT 3 C++ Module [▶ 29] werden im Windows Kernel-Modus ausgeführt. Deswegen müssen die Entwickler einige Limitierungen beachten:

- Win32 API ist nicht im Kernel-Modus verfügbar (siehe unten [▶ 137]).
- 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 [▶ 209])
- Speicherplatz für dynamische Allozierung im Echtzeit-Kontext ist durch den Router-Speicher begrenzt (kann im Verlauf des Engineerings konfiguriert werden). (siehe Speicherallozierung [▶ 137])
- Untermenge der C++ Laufzeit-Bibliotheksfunktionen (CRT) wird unterstützt
- C++ Ausnahmen (C++ Exceptions) werden nicht unterstützt.
- Laufzeit-Typinformation (RTTI, Runtime Type Information) wird nicht unterstützt (siehe unten [▶ 137])
- Untermenge von STL wird unterstützt (siehe STL / Container [▶ 195])

- Unterstützung für Funktionen aus math.h durch TwinCAT Implementierung (siehe [Mathematische Funktionen \[▶ 193\]](#))

### 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	<a href="#">Tracing [▶ 196]</a>
Datei-I/O	Sieh <a href="#">Schnittstelle ITc FileAccess [▶ 142]</a> , <a href="#">Schnittstelle ITc FileAccess Async [▶ 151]</a> und <a href="#">Beispiel19: Synchroner Dateizugriff [▶ 272]</a> , <a href="#">Beispiel20: FileIO-Write [▶ 273]</a> , <a href="#">Beispiel20a: FileIO-Cyclic Read / Write [▶ 273]</a>
Synchronisation	Siehe <a href="#">Beispiel11a: Modulkommunikation: C-Modul ruft eine Methode eines anderem C-Moduls auf [▶ 267]</a>
Visual C CRT	Siehe <a href="#">RtlR0.h</a>

### 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 \[▶ 171\]](#)

## 12.3 Speicherallozierung

Im Allgemeinen wird empfohlen, Speicher mit Hilfe von Member-Variablen der Modulklasse zu reservieren. Dies wird automatisch für im TMC Editor definierte Datenbereiche gemacht.

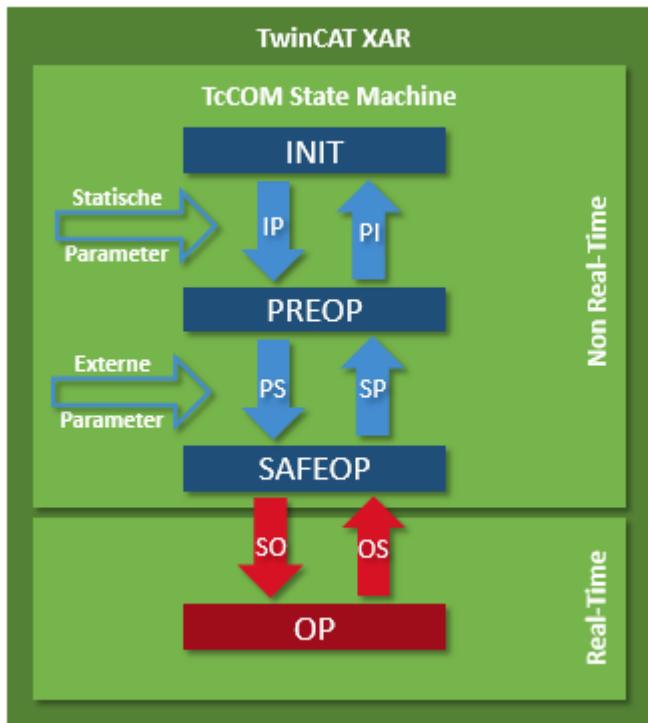
Es besteht auch die Möglichkeit, Speicherplatz dynamisch zu allozieren und freizugeben.

- Operator `new` / `delete`
- `TcMemAllocate` / `TcMemFree`

Diese Speicherallozierung kann in den [Transitionen \[▶ 38\]](#) oder dem OP-State der Statemachine genutzt werden.

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

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. allozieren in PS und Freigeben in SP.



### Globalen Instanzen von Klassen

Globale Instanzen dürfen im Echtzeitkontext allozierten Speicher nicht erst im Destruktor freigeben.

TwinCAT unterstützt bis zu 32 globale Instanzen von Klassen.

Als Globale Instanzen von Klassen werden die folgenden Konstrukte verstanden:

- Definition im globalen Scope
- Definition als statische Klassenvariablen
- Lokale, statische Variablen in Methoden

## 12.4 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
<a href="#">ITcUnknown [▶ 171]</a>	ITcUnknown definiert die Referenzzählung, sowie das Abfragen einer Referenz auf eine spezifischere Schnittstelle.
<a href="#">ITComObject [▶ 156]</a>	Die ITComObject Schnittstelle wird von jedem TwinCAT Modul implementiert.
<a href="#">ITcCyclic [▶ 139]</a>	Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus aufgerufen werden.
<a href="#">ITcCyclicCaller [▶ 140]</a>	Schnittstelle zum Anmelden oder Abmelden der ITcCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.
<a href="#">ITcFileAccess [▶ 142]</a>	Schnittstelle zum Zugriff auf das Dateisystem
<a href="#">ITcFileAccessAsync [▶ 151]</a>	Asynchroner Zugriff auf Dateioperationen.
<a href="#">ITcPostCyclic [▶ 162]</a>	Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus im Anschluss an die Ausgang-Aktualisierung aufgerufen werden.
<a href="#">ITcPostCyclicCaller [▶ 163]</a>	Schnittstelle zum Anmelden oder Abmelden der ITcPostCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.
<a href="#">ITcloCyclic [▶ 152]</a>	Diese Schnittstelle wird von TwinCAT Modulen implementiert, die bei Eingang-Aktualisierung und bei Ausgang-Aktualisierung innerhalb eines Taskzyklus aufgerufen werden.
<a href="#">ITcloCyclicCaller [▶ 154]</a>	Schnittstelle zum Anmelden oder Abmelden der ITcloCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.
<a href="#">ITcRTIME_Task [▶ 165]</a>	Abfrage von erweiterten TwinCAT Taskinformationen.
<a href="#">ITcTask [▶ 166]</a>	Abfrage von Zeitstempel und taskspezifischen Informationen eines TwinCAT Tasks.
<a href="#">ITcTaskNotification [▶ 170]</a>	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.x\ sdk\ include gefunden werden können.

- Das TcCOM Framework wird hier bereitgestellt (insb. TcInterfaces.h und TcServices.h).
- Tasks und Datenbereichs-Zugriffe werden über die TcloInterfaces.h bereitgestellt.
- SDK-Funktionen sind die [mathematischen Funktionen \[▶ 193\]](#).
- Untermenge der [STL \[▶ 195\]](#).
- TwinCAT Runtime [RtlR0.h \[▶ 173\]](#)
- Methoden zur [ADS Kommunikation \[▶ 175\]](#)
- Klassen / Funktionen, die mit „Os“ beginnen, dürfen nicht im Echtzeitkontext genutzt werden.

## Sehen Sie dazu auch

▀ [Mathematische Funktionen \[▶ 193\]](#)

### 12.4.1 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

Symbol	Name	Beschreibung
	<a href="#">CycleUpdate [▶ 140]</a>	Wird ein Mal pro Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen Aufrufer angemeldet ist.

## Anmerkungen

Die Schnittstelle ITcCyclic wird von TwinCAT Modulen implementiert. Diese Schnittstelle wird der Methode ITcCyclicCaller::AddModule() übergeben, wenn sich ein Modul bei einem Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen.

### 12.4.1.1 Methode ITcCyclic:CyclicUpdate

Die Methode CyclicUpdate wird normalerweise von einem TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

#### Syntax

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

#### Parameter

**ipTask:** (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

**ipCaller:** (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

**Context:** (Typ: ULONG\_PTR) Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddModule() übergeben wurde.

#### Rückgabewert

Es wird empfohlen, stets S\_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

#### 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.

#### Sehen Sie dazu auch

 [Schnittstelle ITcCyclicCaller \[▶ 140\]](#)

### 12.4.2 Schnittstelle ITcCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.

#### Syntax

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

Benötigtes include: TcIoInterfaces.h

## Methoden

Sym- bol	Name	Beschreibung
	<a href="#">AddModule [▶ 141]</a>	Modul anmelden, das die ITcCyclic Schnittstelle implementiert.
	<a href="#">RemoveModule [▶ 142]</a>	Die zuvor angemeldete ITcCyclic Schnittstelle eines Moduls abmelden.

## Anmerkungen

Die ITcCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle, um seine ITcCyclic Schnittstelle bei einem 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 vom Task aufgerufen wird.

### 12.4.2.1 Methode ITcCyclicCaller:AddModule

Meldet die ITcCyclic Schnittstelle eines Moduls bei einem zyklischen Aufrufer, z.B. einem TwinCAT Task an.

#### Syntax

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

#### Parameter

**pEntry:** (Typ: STcCyclicEntry) [in] Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen Aufrufers eingefügt wird. Siehe auch Beschreibung.

**ipMod:** (Typ: ITcCyclic) [in] Schnittstellenzeiger, der vom zyklischen Aufrufer verwendet wird

**context:** (Typ: ULONG\_PTR) [optional] ein Kontextwert, der bei jedem Aufruf an die ITcCyclic::CyclicUpdate() Methode übergeben wird.

**sortOrder:** (Typ: ULONG) [optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

#### Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S\_OK zurück. Wenn der zyklische Aufrufer, d.h. der TwinCAT Task, nicht im OP-Zustand ist, wird der Fehler ADS\_E\_INVALIDSTATE zurückgegeben.

#### Beschreibung

Eine TwinCAT-Modulklasse verweist normalerweise mit einem Smart Pointer auf den zyklischen Aufrufer Typ ITcCyclicCallerPtr. Die Objekt-ID des Tasks 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;
}
}
}
}

```

### Sehen Sie dazu auch

- „ Schnittstelle ITcCyclic [▶ 139]
- „ Schnittstelle ITcCyclicCaller [▶ 154]

## 12.4.2.2 Methode ITcCyclicCaller::RemoveModule

Eine Modulinstanz vom Aufruf durch einen zyklischen Aufrufer abmelden.

### Syntax

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

### Parameter

**pEntry:** (Typ: STcCyclicEntry) verweist auf den Listeneintrag, der aus der internen Liste des zyklischen Aufrufer zu entfernen ist.

### Rückgabewert

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E\_FAIL zurück.

### Beschreibung

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;

```

## 12.4.3 Schnittstelle ITcFileAccess

Schnittstelle für Zugriff auf Dateisystem von TwinCAT C++ Modulen aus

### Syntax

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

Benötigtes include: TcFileAccessInterfaces.h

## Methoden

Sym- bol	Name	Beschreibung
☰	<a href="#">FileOpen [▶ 143]</a>	Öffnet eine Datei
☰	<a href="#">FileClose [▶ 144]</a>	Schließt eine Datei
☰	<a href="#">FileRead [▶ 145]</a>	Liest aus einer Datei
☰	<a href="#">FileWrite [▶ 145]</a>	Schreibt in eine Datei
☰	<a href="#">FileSeek [▶ 146]</a>	Setzt Position in einer Datei
☰	<a href="#">FileTell [▶ 146]</a>	Fragt Position in einer Datei ab
☰	<a href="#">FileRename [▶ 147]</a>	Nennt eine Datei um
☰	<a href="#">FileDelete [▶ 147]</a>	Löscht eine Datei
☰	<a href="#">FileGetStatus [▶ 147]</a>	Erhält den Zustand einer Datei
☰	<a href="#">FileFindFirst [▶ 148]</a>	Sucht nach einer Datei, erste Iteration
☰	<a href="#">FileFindNext [▶ 149]</a>	Sucht nach einer Datei, nächste Iteration
☰	<a href="#">FileFindClose [▶ 150]</a>	Schließt eine Dateisuche
☰	<a href="#">MkDir [▶ 150]</a>	Erstellt ein Verzeichnis
☰	<a href="#">RmDir [▶ 150]</a>	Löscht ein Verzeichnis

## Anmerkungen

Die ITc FileAccess 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 [ITc FileAccess Async \[▶ 151\]](#) fügt eine Check() Methode hinzu, die stattdessen verwendet werden kann.

Siehe [Beispiel20a: FileIO-Cyclic Read / Write \[▶ 273\]](#).

Die Schnittstelle wird über die Modulklasse CID\_Tc FileAccess implementiert.

### 12.4.3.1 Methode ITc FileAccess:FileOpen

Öffnet eine Datei

#### Syntax

```
virtual HRESULT TCOMAPI FileOpen(PCCH szFileName, Tc FileAccessMode AccessMode, PTc FileHandle phFile);
```

#### Parameter

**szFileName:** (Typ: PCCH) [in] der Name der zu öffnenden Datei

**AccessMode:** (Typ: Tc FileAccessMode) [in] Art des Zugriffs auf die Datei, siehe *Tc FileAccess Services.h*

**phFile:** (Typ: Tc FileHandle) [out] zurückgegebener Datei-Handle

#### Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S\_OK zurück.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-ZustandsCodes \[▶ 305\]](#) sind möglich.

### **Beschreibung**

Die Methode gibt einen Handle für den Zugriff auf die Datei zurück, dessen Name in szFileName definiert ist.

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;
```

### **12.4.3.2 Methode ITcFileAccess::FileClose**

Schließt eine Datei

#### **Syntax**

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

#### **Parameter**

**phFile:** (Typ: TcFileHandle) [out] zurückgegebener Datei-Handle

#### **Rückgabewert**

Typ: HRESULT

Bei Erfolg gibt die Methode S\_OK zurück.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-ZustandsCodes \[▶ 305\]](#) sind möglich.

### **Beschreibung**

Die Methode schließt eine mittels phFile definierte Datei.

#### **Sehen Sie dazu auch**

[ADS Return Codes \[▶ 305\]](#)

### 12.4.3.3 Methode ITcFileAccess:FileRead

Liest Daten aus einer Datei.

#### Syntax

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

#### Parameter

**hFile:** (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

**pData:** (Typ: PVOID) [out] Speicherort der zu lesenden Daten

**cbData:** (Typ: PVOID) [in] maximale Größe der zu lesenden Daten (Größe des Speichers hinter pData)

**pcbRead:** (Typ: PUINT) [out] Größe der gelesenen Daten

#### Rückgabewert

Typ: HRESULT

Wenn Daten gelesen werden können, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-Zustandscodes \[► 305\]](#) sind möglich.

#### Beschreibung

Diese Methode fragt Daten von einer mit Dateihandle definierten Datei ab. Die Daten werden in pData gespeichert, wobei pcbRead die Länge der Daten angibt.

### 12.4.3.4 Methode ITcFileAccess:FileWrite

Daten in eine Datei schreiben.

#### Syntax

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

#### Parameter

**hFile:** (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

**pData:** (Typ: PCVOID) [in] Speicherort der zu schreibenden Daten

**cbData:** (Typ: PCVOID) [in] Größe der zu schreibenden Daten (Größe des Speichers hinter pData)

**pcbRead:** (Typ: PUINT) [out] Größe der geschriebenen Daten

#### Rückgabewert

Typ: HRESULT

Wenn Daten geschrieben werden können, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-Zustandscodes \[► 305\]](#) sind möglich.

## Beschreibung

Diese Methode schreibt Daten in eine mit Dateihandle definierte Datei. Die Daten werden in pData gelesen, wobei pcbRead die Länge der Daten angibt.

### 12.4.3.5 Methode ITcFileAccess::FileSeek

Setzt Position in einer Datei.

#### Syntax

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

#### Parameter

**hFile:** (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

**uiPos:** (Typ: UINT) [in] Position auf die zu setzen ist

#### Rückgabewert

Typ: HRESULT

Wenn die Position gesetzt werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [► 305] sind möglich.

## Beschreibung

Diese Methode legt die Position innerhalb einer Datei für weitere Aktionen fest.

### 12.4.3.6 Methode ITcFileAccess::FileTell

Fragt Position in einer Datei ab.

#### Syntax

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

#### Parameter

**hFile:** (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

**puiPos:** (Typ: PUINT) [out] Speicherort der zurückzugebenden Position

#### Rückgabewert

Typ: HRESULT

Wenn die Position abgefragt werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [► 305] sind möglich.

## Beschreibung

Diese Methode fragt die derzeit gesetzte Position innerhalb einer Datei ab.

### 12.4.3.7 Methode ITcFileAccess:FileRename

Nennt eine Datei um

#### Syntax

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

#### Parameter

**szOldName:** (Typ: PCCH) [in] der umzubenenende Dateiname

**szNewName:** (Typ: PCCH) [in] der neue Dateiname

#### Rückgabewert

Typ: HRESULT

Wenn die Datei umbenannt werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-ZustandsCodes](#) [► 305] sind möglich.

#### Beschreibung

Diese Methode benennt eine Datei mit einem neuen Namen um.

### 12.4.3.8 Methode ITcFileAccess:FileDelete

Löscht eine Datei.

#### Syntax

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

#### Parameter

**szFileName:** (Typ: PCCH) [in] Name der zu löschen Datei

#### Rückgabewert

Typ: HRESULT

Wenn die Datei gelöscht werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-ZustandsCodes](#) [► 305] sind möglich.

#### Beschreibung

Diese Methode löscht eine Datei aus dem Dateisystem

### 12.4.3.9 Methode ITcFileAccess:FileGetStatus

Fragt den Zustand einer Datei ab.

#### Syntax

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

## Parameter

**szFileName:** (Typ: PCCH) [in] der Name der fraglichen Datei

**pFileStatus:** (Typ: PTcFileStatus) [out] der Zustand der Datei. Vergleiche mit *TcFileAccessServices.h*

## Rückgabewert

Typ: HRESULT

Wenn Zustand zurückgegeben werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [► 305] sind möglich.

## Beschreibung

Diese Methode fragt Zustandsinformationen bezüglich eines gegebenen Dateinamen 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;
```

### 12.4.3.10 Methode ITcFileAccess:FileFindFirst

Möglichkeit, die Dateien eines Verzeichnisses zu durchlaufen.

## Syntax

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

## Parameter

**szFileName:** (Typ: PCCH) [in] Verzeichnis oder Pfad, und Name der gesuchten Datei. Der Dateiname kann Platzhalterzeichen wie Sternchen (\*) oder Fragezeichen (?) enthalten.

**pFileFindData:** (Typ: PTcFileFindData) [out] die Beschreibung der ersten Datei. Vergleiche mit *TcFileAccessServices.h*

**phFileFind:** (Typ: PTcFileFindHandle) [out] Handle um weiter mit FileFindNext zu suchen.

## Rückgabewert

Typ: HRESULT

Wenn eine Datei gefunden werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-Zustandscodes \[► 305\]](#) sind möglich.

## 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;
```

### 12.4.3.11 Methode ITcFileAccess:FileFindNext

Die Dateien eines Verzeichnisses weiter durchlaufen.

#### Syntax

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

#### Parameter

**hFileFind:** (Typ: PTcFileFindHandle) [in] Handle um weiter mit FileFindNext zu suchen.

**pFileFindData:** (Typ: PTcFileFindData) [out] die Beschreibung der nächsten Datei. Vergleiche mit *TcFileAccessServices.h*

#### Rückgabewert

Typ: HRESULT

Wenn eine Datei gefunden werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-Zustandscodes \[► 305\]](#) sind möglich.

## 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;
```

### 12.4.3.12 Methode ITcFileAccess:FileFindClose

Die Suche nach Dateien in einem Verzeichnis schließen.

#### Syntax

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

#### Parameter

**hFileFind:** (Typ: PTcFileFindHandle) [in] Handle um die Suche zu schließen

#### Rückgabewert

Typ: HRESULT

Wenn die Dateisuche geschlossen werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-ZustandsCodes \[► 305\]](#) sind möglich.

#### Beschreibung

Diese Methode schließt die Suche nach Dateien in einem Verzeichnis.

### 12.4.3.13 Methode ITcFileAccess:MkDir

Ein Verzeichnis in einem Dateisystem erstellen.

#### Syntax

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

#### Parameter

**szDir:** (Typ: PCCH) [in] zu erstellendes Verzeichnis

#### Rückgabewert

Typ: HRESULT

Wenn Verzeichnis erstellt werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere [ADS-ZustandsCodes \[► 305\]](#) sind möglich.

#### Beschreibung

Diese Methode erstellt ein Verzeichnis entsprechend der Definition im szDir Parameter.

### 12.4.3.14 Methode ITcFileAccess:RmDir

Ein Verzeichnis aus dem Dateisystem entfernen.

#### Syntax

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

## Parameter

**szDir:** (Typ: PCCH) [in] zu lösches Verzeichnis

## Rückgabewert

Typ: HRESULT

Wenn Verzeichnis gelöscht werden konnte, wird S\_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS\_E\_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [► 305] sind möglich.

## Beschreibung

Diese Methode löscht ein Verzeichnis entsprechend der Definition im szDir Parameter.

## 12.4.4 Schnittstelle ITc FileAccessAsync

Asynchroner Zugriff auf Dateioperationen.

Diese Schnittstelle erweitert ITc FileAccess [► 142].

### Syntax

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

Benötigtes include: TcFileAccessInterfaces.h

### Methoden

Symbol	Name	Beschreibung
	C [► 152] heck	Zustand der Dateioperation abfragen

### Schnittstellenparameter

Sym- bol	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

### Anmerkungen

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 eines Tasks, das 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 \[▶ 273\]](#)

#### **12.4.4.1 Method ITc FileAccessAsync::Check()**

Zustand der Dateioperation abfragen

##### **Syntax**

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

##### **Parameter**

keiner

##### **Rückgabewert**

Typ: HRESULT

Gibt S\_OK zurück, wenn Dateioperation abgeschlossen ist

Besonders interessante Fehlercodes:

- ADS\_E\_PENDING, wenn die Dateioperation nicht abgeschlossen ist.
- ADS\_E\_TIMEOUT, wenn die Zeitüberschreitung für die Dateioperation abgelaufen ist.

Weitere [ADS-Zustandscodes \[▶ 305\]](#) sind möglich.

##### **Beschreibung**

Diese Operation überprüft den Zustand der zuvor aufgerufenen Dateioperation

#### **12.4.5 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**

<b>Sym- bol</b>	<b>Name</b>	<b>Beschreibung</b>
	<a href="#">InputUpdate [▶ 153]</a>	Wird zu Beginn eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.
	<a href="#">OutputUpdate [▶ 153]</a>	Wird am Ende eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.

## Anmerkungen

ITcloCyclic kann dazu verwendet werden, ein TwinCAT Modul zu implementieren, das als Feldbustreiber oder I/O-Filtermodul agiert.

Diese Schnittstelle wird der Methode ITcloCyclicCaller::AddIoDriver übergeben, wenn sich ein Modul bei einem Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung werden die Methoden InputUpdate() und OutputUpdate() der Modulinstantz je einmal pro Taskzyklus aufgerufen.

### 12.4.5.1 Methode ITcloCyclic:InputUpdate

Die Methode InputUpdate wird normalerweise von einem TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

#### Syntax

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

#### Parameter

**ipTask:** (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

**ipCaller:** (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

**dwStateIn:** (Typ: DWORD) zukünftigen Erweiterungen vorbehalten, derzeit ist dieser Wert immer 0

**context:** (Typ: ULONG\_PTR) Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddIoDriver() übergeben wurde.

#### Rückgabewert

Es wird empfohlen, stets S\_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

#### 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 Eingabequelle des Moduls verwendet werden.

### 12.4.5.2 Methode ITcloCyclic:OutputUpdate

Die Methode OutputUpdate wird normalerweise von einem TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

#### Syntax

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

#### Parameter

**ipTask:** (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

**ipCaller:** (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

**pdwStateOut:** (Typ: DWORD) [out] zukünftigen Erweiterungen vorbehalten, derzeit wird der Rückgabewert ignoriert.

**context:** (Typ: ULONG\_PTR) Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddIoDriver() übergeben wurde.

## Rückgabewert

Es wird empfohlen, stets S\_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

## Beschreibung

In einem Taskzyklus wird für alle angemeldeten Modulinstanzen die Methode OutputUpdate() aufgerufen. Deswegen muss diese Methode für die Aktualisierung der Datenbereiche vom Typ Eingabequelle des Moduls verwendet werden.

## 12.4.6 Schnittstelle ITcloCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcloCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.

### Syntax

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

Benötigtes include: TcIoInterfaces.h

### Methoden

Symbole	Name	Beschreibung
≡	AddIoDriver <a href="#">► 154</a>	Modul anmelden, das die ITcloCyclic Schnittstelle implementiert.
≡	RemoveIoDriver <a href="#">► 155</a>	Zuvor angemeldete ITcloCyclic Schnittstelle eines Moduls abmelden.

### Anmerkungen

Die ITcloCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle um seine ITcloCyclic Schnittstelle bei einem 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 vom Task aufgerufen wird.

### 12.4.6.1 Methode ITcloCyclicCaller::AddIoDriver

Meldet die ITcloCyclic Schnittstelle eines Moduls bei einem zyklischen I/O-Aufrufer, z.B. einem TwinCAT Task an.

### Syntax

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

## Parameter

**pEntry:** (Typ: STcloCyclicEntry) Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen I/O-Aufrufers eingefügt wird. Siehe auch Beschreibung.

**ipDrv:** (Typ: ITcloCyclic) [in] Schnittstellenzeiger, der vom zyklischen I/O-Aufrufer verwendet wird

**context:** (Typ: ULONG\_PTR) [optional] ein Kontextwert, der bei jedem Aufruf an die ITcloCyclic::InputUpdate() und ITcloCyclic::OutputUpdate Methode übergeben wird.

**sortOrder:** (Typ: ULONG) [optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

## Rückgabewert

Typ: HRESULT

## Beschreibung

Eine TwinCAT-Modulklasse verwendet normalerweise einen Smart Pointer, um auf den zyklischen I/O-Aufrufer vom Typ ITcloCyclicCallerPtr 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 ITcloCyclicCaller 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(ITcloCyclic))) )
{
m_spIoCyclicCaller = NULL;
}
}
```

### 12.4.6.2 Methode ITcloCyclicCaller:RemoveloDriver

Eine Modulinstanz vom Aufruf durch einen zyklischen I/O-Aufrufer abmelden.

## Syntax

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

## Parameter

**pEntry:** (Typ: STcloCyclicEntry) verweist auf den Listeneintrag, der aus der internen Liste des zyklischen I/O-Aufrufers zu entfernen ist.

## Rückgabewert

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E\_FAIL zurück.

## Beschreibung

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;
```

## 12.4.7 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

Sym- bol	Name	Beschreibung
≡	<a href="#">TcGetObjectID(OTCID&amp; objId)</a> [▶ 156]	Speichert die Objekt ID mit Hilfe der gegebenen OTCID Referenz.
≡	<a href="#">TcSetObjectID</a> [▶ 157]	Setzt die Objekt-ID des Objekts auf die gegebene OTCID
≡	<a href="#">TcGetObjectName</a> [▶ 157]	Speichert den Objektnamen im Puffer mit der gegebenen Länge
≡	<a href="#">TcSetObjectName</a> [▶ 158]	Setzt den Objektnamen des Objekts auf gegebenen CHAR*
≡	<a href="#">TcSetObjState</a> [▶ 158]	Initialisiert einen Übergang zu einem vorgegebenen Zustand.
≡	<a href="#">TcGetObjState</a> [▶ 159]	Fragt den aktuellen Zustands des Objekts ab.
≡	<a href="#">TcGetObjPara</a> [▶ 159]	Fragt einen mit seiner PTCID identifizierten Objektparameter ab
≡	<a href="#">TcSetObjPara</a> [▶ 159]	Setzt einen mit seiner PTCID identifizierten Objektparameter
≡	<a href="#">TcGetParentObjId</a> [▶ 160]	Speichert die Parent-Objekt ID mit Hilfe der gegebenen OTCID Referenz.
≡	<a href="#">TcSetParentObjId</a> [▶ 160]	Setzt die Parent-Objekt-ID auf die gegebene OTCID.

### Anmerkungen

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt Funktionalitäten zur Verfügung bezüglich der Zustandsmaschine und Informationen vom/an das TwinCAT-System.

### 12.4.7.1 Methode ITcComObject:TcGetObjectId

Die Methode speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

### Syntax

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

### Parameter

**objId:** (Typ: OTCID&) Referenz auf OTCID-Wert

## Rückgabewert

Informiert über Erfolg der OTCID Abfrage.

## Beschreibung

Die Methode speichert Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

## Sehen Sie dazu auch

▀ Schnittstelle ITcCyclicCaller [▶ 140]

### 12.4.7.2 Methode ITcComObject:TcSetObjectId

Die Methode TcSetObjectId setzt die Objekt-ID des Objekts auf die gegebene OTCID.

## Syntax

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

## Parameter

**objId:** (Typ: OTCID) Die zu setzende OTCID.

## Rückgabewert

Es wird empfohlen, stets S\_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

## Beschreibung

Zeigt den Erfolg der id-Änderung an.

## Sehen Sie dazu auch

▀ Schnittstelle ITcCyclicCaller [▶ 140]

### 12.4.7.3 Methode ITcComObject:TcGetObjectName

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

## Syntax

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

## Parameter

**objName:** (Typ: CHAR\*) der zu setzende Name.

**nameLen:** (Typ: ULONG) die maximale, zu schreibende Länge.

## Rückgabewert

Informiert über Erfolg der Namen-Abfrage.

## Beschreibung

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

## Sehen Sie dazu auch

█ Schnittstelle ITcCyclicCaller [▶ 140]

#### 12.4.7.4 Methode ITcComObject:TcSetObjectName

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR\*.

##### Syntax

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

##### Parameter

**objName:** (Typ: CHAR\*) der zu setzende Name des Objekts

##### Rückgabewert

Informiert über Erfolg der Namen-Abfrage.

##### Beschreibung

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR\*.

##### Sehen Sie dazu auch

█ Schnittstelle ITcCyclicCaller [▶ 140]

#### 12.4.7.5 Methode ITcComObject:TcSetObjState

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

##### Syntax

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

##### Parameter

**state:** (Typ: TCOM\_STATE) stellt den neuen Zustand dar

**ipSrv:** (Typ: ITComObjectServer\*) handhabt des Objekts

**pInitData:** (Typ: PTComInitDataHdr) zeigt auf eine Liste von Parametern (optional)  
Siehe Makro IMPLEMENT\_ITCOMOBJECT\_EVALUATE\_INITDATA als Beispiel, wie die Liste iteriert werden kann.

##### Rückgabewert

Zeigt den Erfolg der Zustandsänderung an.

##### Beschreibung

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

##### Sehen Sie dazu auch

█ Schnittstelle ITcCyclicCaller [▶ 140]

### 12.4.7.6 Methode ITcComObject:TcGetObjState

Die Methode TcGetObjState fragt den aktuellen Zustands des Objekts ab.

#### Syntax

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

#### Parameter

**pState:** (Typ: TCOM\_STATE\*) Zeiger auf den Zustand

#### Rückgabewert

Informiert über Erfolg der Zustandsabfrage.

#### Beschreibung

Die Methode TcGetObjState fragt den aktuellen Zustands des Objekts ab.

#### Sehen Sie dazu auch

▀ Schnittstelle ITcCyclicCaller [▶ 140]

### 12.4.7.7 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)
```

#### Parameter

**pid:** (Typ: PTCID) Parameter-ID des Objektparameters

**nData:** (Typ: ULONG&) max. Länge der Daten

**pData:** (Typ: PVOID&) Zeiger auf die Daten

**pgp:** (Typ: PTCGP) für zukünftige Erweiterung vorbehalten, NULL weitergeben

#### Rückgabewert

Informiert über Erfolg der Objektparameterabfrage.

#### Beschreibung

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

#### Sehen Sie dazu auch

▀ Schnittstelle ITcCyclicCaller [▶ 140]

### 12.4.7.8 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)
```

## Parameter

**pid:** (Typ: PTCID) Parameter-ID des Objektparimeters

**nData:** (Typ: ULONG) max. Länge der Daten

**pData:** (Typ: PVOID) Zeiger auf die Daten

**pgp:** (Typ: PTCGP) für zukünftige Erweiterung vorbehalten, NULL weitergeben

## Rückgabewert

Informiert über Erfolg der Objektparameferabfrage.

## Beschreibung

Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparimeter.

## Sehen Sie dazu auch

█ Schnittstelle ITcCyclicCaller [▶ 140]

## 12.4.7.9 Methode ITcComObject:TcGetParentObjId

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

## Syntax

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

## Parameter

**objId:** (Typ: OTCID&) Referenz auf OTCID-Wert

## Rückgabewert

Informiert über Erfolg der parentObjId-Abfrage.

## Beschreibung

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

## Sehen Sie dazu auch

█ Schnittstelle ITcCyclicCaller [▶ 140]

## 12.4.7.10 Methode ITcComObject:TcSetParentObjId

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

## Syntax

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

## Parameter

**objId:** (Typ: OTCID) Referenz auf OTCID-Wert

## Rückgabewert

Es wird empfohlen, stets S\_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

## Beschreibung

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

## Sehen Sie dazu auch

▀ Schnittstelle ITcCyclicCaller [▶ 140]

## 12.4.8 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

Sym- bol	Name	Beschreibung
≡	<a href="#">OTCID TcGetObjectId [▶ 161]</a>	Fragt die Objekt ID vom Objekt ab.
≡	<a href="#">TcTryToReleaseOpState [▶ 162]</a>	Gibt Ressourcen frei; muss implementiert sein

## Anmerkungen

Es existieren weitere Methoden, die hier nicht einzeln aufgeführt werden.

Diese Funktionalität wird durch die Modul-Assistenten bei den Modulen standardmäßig bereitgestellt.

### 12.4.8.1 Methode TcGetObjectId

Die Methode fragt die Objekt-ID des Objektes ab.

## Syntax

```
OTCID TcGetObjectId(void)
```

## Parameter

## Rückgabewert

OTCID: Gibt die OTCID des Objekts zurück.

## Beschreibung

Die Methode speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

**Sehen Sie dazu auch**

„[Schnittstelle ITcCyclicCaller \[▶ 140\]](#)“

**12.4.8.2 Methode TcTryToReleaseOpState**

Die Methode TcTryToReleaseOpState gibt Ressourcen frei, z.B. Datenzeiger, um das Verlassen des OP-Zustands vorzubereiten.

**Syntax**

```
BOOL TcTryToReleaseOpState(void)
```

**Parameter****Rückgabewert**

TRUE bedeutet Freigabe von Ressourcen war erfolgreich.

**Beschreibung**

Die Methode TcTryToReleaseOpState gibt Ressourcen frei, z.B. Datenzeiger, um das Verlassen des OP-Zustands vorzubereiten. Muss implementiert werden, um mögliche wechselseitige Abhängigkeiten von Modulinstanzen aufzuheben. Siehe [Beispiel10 \[▶ 239\]](#).

**Sehen Sie dazu auch**

„[Schnittstelle ITcCyclicCaller \[▶ 140\]](#)“

**12.4.9 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**

<b>Sym- bol</b>	<b>Name</b>	<b>Beschreibung</b>
	<a href="#">PostCycleUpdate [▶ 163]</a>	Wird ein Mal pro Taskzyklus nach der Ausgangs-Aktualisierung aufgerufen, wenn die Schnittstelle bei einem zyklischen Aufrufer angemeldet wurde.

**Anmerkungen**

Die ITcPostCyclic Schnittstelle wird von TwinCAT Modulen implementiert. Diese Schnittstelle wird der Methode `ITcCyclicCaller::AddPostModule()` übergeben, wenn ein Modul sich selber bei einem Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung wird die Methode `PostCycleUpdate()` der Modulinstantz aufgerufen.

## 12.4.9.1 Methode ITcPostCyclic::PostCyclicUpdate

Die normalerweise von einem 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)
```

### Parameter

ipTask: (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

ipCaller: (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

Context: (Typ: ULONG\_PTR) Kontext beinhaltet den Wert, der an die Methode ITcPostCyclicCaller::AddPostModule() übergeben wurde.

### Rückgabewert

Es wird empfohlen, stets S\_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

### 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.

### Sehen Sie dazu auch

▀ Schnittstelle ITcPostCyclicCaller [▶ 163]

## 12.4.10 Schnittstelle ITcPostCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcPostCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.

### Syntax

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

Benötigtes include: TcIoInterfaces.h

### Methoden

Sym- bol	Name	Beschreibung
≡	<a href="#">AddPostModule [▶ 164]</a>	Modul anmelden, das die ITcPostCyclic Schnittstelle implementiert.
≡	<a href="#">RemovePostModule [▶ 165]</a>	Zuvor angemeldete ITcPostCyclic Schnittstelle eines Moduls abmelden.

## Anmerkungen

Die ITcPostCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle um seine ITcPostCyclic Schnittstelle bei einem 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 vom Task aufgerufen wird.

### 12.4.10.1 Methode ITcPostCyclicCaller::AddPostModule

Meldet die ITcPostCyclic Schnittstelle eines Moduls bei einem zyklischen Aufrufer, z.B. einem TwinCAT Task an.

#### Syntax

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

#### Parameter

**pEntry:** (Typ: STcPostCyclicEntry) [in] Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen Aufrufers eingefügt wird. Siehe auch Beschreibung.

**ipMod:** (Typ: ITcPostCyclic) [in] Schnittstellenzeiger, der vom zyklischen Aufrufer verwendet wird

**context:** (Typ: ULONG\_PTR) [optional] ein Kontextwert, der bei jedem Aufruf an die ITcPostCyclic::PostCyclicUpdate() Methode übergeben wird.

**sortOrder:** (Typ: ULONG) [optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

#### Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S\_OK zurück. Wenn der zyklische Aufrufer, d.h. der TwinCAT Task, nicht im OP-Zustand ist, wird der Fehler ADS\_E\_INVALIDSTATE zurückgegeben.

#### Beschreibung

Ein TwinCAT-Modulklasse verwendet einen Smart Pointer, um auf den zyklischen Aufrufer vom Typ ITcPostCyclicCallerPtr zu verweisen. Die Objekt-ID des Tasks 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 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;
        }
    }
}
```

```
}
```

```
}
```

### Sehen Sie dazu auch

- ▀ Schnittstelle ITcPostCyclic [▶ 162]
- ▀ Schnittstelle ITcloCyclicCaller [▶ 154]

## 12.4.10.2 Methode ITcPostCyclicCaller:RemovePostModule

Eine Modulinstanz vom Aufruf durch einen zyklischen Aufrufer abmelden.

### Syntax

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

### Parameter

pEntry: (Typ: STcPostCyclicEntry) verweist auf den Listeneintrag, der aus der internen Liste des zyklischen Aufrufers zu entfernen ist.

### Rückgabewert

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E\_FAIL zurück.

### Beschreibung

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;
```

## 12.4.11 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

Symbol	Name	Beschreibung
	<a href="#">GetCpuAccount [▶ 166]</a>	Abfrage des CPU-Kontos eines TwinCAT Tasks.

### Anmerkungen

Mit dieser Schnittstelle können TwinCAT Taskinformationen abgefragt und verwendet werden.

Siehe [Beispiel30: Zeitmessung \[▶ 281\]](#)

### 12.4.11.1 Methode ITcRTTimeTask::GetCpuAccount()

Abfrage des CPU-Kontos eines TwinCAT Tasks.

#### Syntax

```
virtual HRESULT TCOMAPI GetCpuAccount (PULONG pAccount)=0;
```

#### Parameter

**pAccount:** (Typ: PULONG) [out] TwinCAT Task CPU-Konto ist in diesem Parameter gespeichert.

#### Rückgabewert

E\_POINTER wenn Parameter pAccount gleich NULL, sonst S\_OK

#### Beschreibung

Mit Hilfe der GetCpuAccount() Methode kann die aktuelle, für den 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
ITcRTTimeTaskPtr spRTTimeTask = ipTask;
ULONG nCpuAccountForComputeSomething = 0;
if (spRTTimeTask != NULL)
{
ULONG nStart = 0;
hr = FAILED(hr) ? hr : spRTTimeTask->GetCpuAccount (&nStart);

ComputeSomething();

ULONG nStop = 0;
hr = FAILED(hr) ? hr : spRTTimeTask->GetCpuAccount (&nStop);

nCpuAccountForComputeSomething = nStop - nStart;
}
```

### 12.4.12 Schnittstelle ITcTask

Abfrage von Zeitstempel und taskspezifischen Informationen eines TwinCAT Tasks.

#### Syntax

```
TCOM_DECL_INTERFACE("02000002-0000-0000-e000-000000000064", ITcTask)
struct __declspec(novtable) ITcTask : public ITcUnknown
```

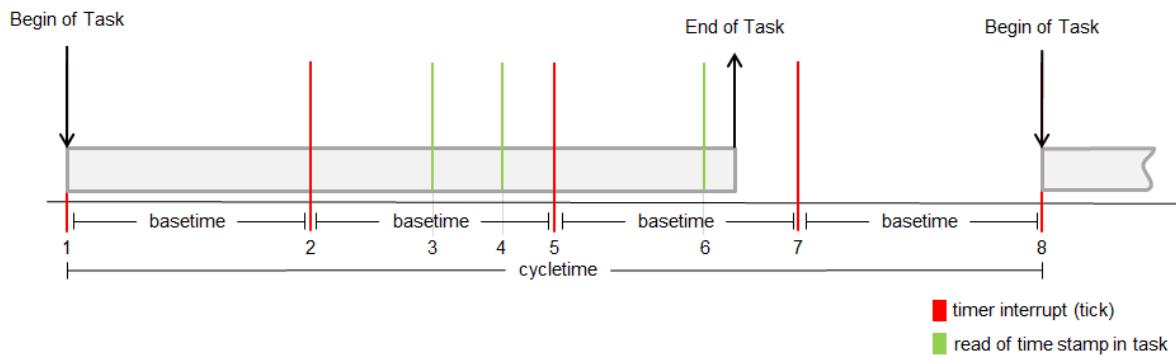
Benötigtes include: TcRtInterfaces.h

## Methoden

Sym- bol	Name	Beschreibung
	<a href="#">GetCycleCounter [► 169]</a>	Anzahl Taskzyklen seit Taskstart abfragen.
	<a href="#">GetCycleTime [► 169]</a>	Abfrage der Taskzykluszeit in Nanosekunden, d.h. Zeit zwischen „begin of task“ und nächstem „begin of task“
	<a href="#">GetPriority [► 167]</a>	Abfrage der Taskpriorität
	<a href="#">GetCurrentSysTime [► 168]</a>	Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).
	<a href="#">GetCurrentDcTime [► 168]</a>	Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000.
	<a href="#">GetCurPentiumTime [► 168]</a>	Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

## Anmerkungen

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

### 12.4.12.1 Methode ITcTask:GetPriority

Abfrage der Taskpriorität

#### Syntax

```
virtual HRESULT TCOMAPI GetPriority(PULONG pPriority)=0;
```

#### Parameter

**pPriority:** (Typ: PULONG) [out] Prioritätswert des Tasks ist in diesem Parameter gespeichert.

**Rückgabewert**

E\_POINTER wenn Parameter pPriority gleich NULL, sonst S\_OK

**Beschreibung**

Beispiel30: Zeitmessung [► 281] veranschaulicht die Verwendung dieser Methode.

### 12.4.12.2 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;
```

**Parameter**

**pSysTime:** (Typ: PLONGLONG) [out] in diesem Parameter ist die aktuelle Systemzeit zu Beginn des Taskzyklus gespeichert.

**Rückgabewert**

E\_POINTER wenn Parameter pSysTime gleich NULL, sonst S\_OK.

**Beschreibung**

Beispiel30: Zeitmessung [► 281] veranschaulicht die Verwendung dieser Methode.

### 12.4.12.3 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;
```

**Parameter**

**pDcTime:** (Typ: PLONGLONG) [out] in diesem Parameter ist die Distributed-Clock-Zeit zu Beginn des Taskzyklus gespeichert.

**Rückgabewert**

E\_POINTER wenn Parameter pDcTime gleich NULL, sonst S\_OK.

**Beschreibung**

Beispiel30: Zeitmessung [► 281] veranschaulicht die Verwendung dieser Methode.

### 12.4.12.4 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;
```

## Parameter

**pCurTime:** (Typ: PLONGLONG) [out] in diesem Parameter wird die aktuelle Zeit (UTC) in 100 Nanosekunden-Intervallen seit dem 1. Januar 1601 gespeichert

## Rückgabewert

E\_POINTER wenn Parameter pCurTime gleich NULL, sonst S\_OK

## Beschreibung

Beispiel30: Zeitmessung [► 281] veranschaulicht die Verwendung dieser Methode.

### 12.4.12.5 Methode ITcTask:GetCycleCounter

Anzahl Taskzyklen seit Taskstart abfragen.

## Syntax

```
virtual HRESULT TCOMAPI GetCycleCounter (PULONGLONG  
pCnt)=0;
```

## Parameter

**pCnt:** (Typ: PULONGLONG) [out] die Anzahl von Taskzyklen seit Task gestartet wurde, wird in diesem Parameter gespeichert

## Rückgabewert

E\_POINTER wenn Parameter pCnt gleich NULL, ansonsten S\_OK

## Beschreibung

Beispiel30: Zeitmessung [► 281] veranschaulicht die Verwendung dieser Methode.

### 12.4.12.6 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;
```

## Parameter

**pCycleTimeNS:** (Typ: PULONG) [out] in diesem Parameter wird die konfigurierte Taskzyklenzeit in Nanosekunden gespeichert.

## Rückgabewert

E\_POINTER wenn Parameter pCnt gleich NULL, sonst S\_OK

## Beschreibung

Beispiel30: Zeitmessung [▶ 281] veranschaulicht die Verwendung dieser Methode.

### 12.4.13 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

Symbol	Name	Beschreibung
	NotifyCycleTimeExceeded	Wird aufgerufen, wenn die Zykluszeit überschritten wurde.

#### Anmerkungen

Beachten Sie, dass der Callback nicht während den Berechnungen, sondern am Ende des Zyklus stattfindet. Also bietet diese Methode keinen Mechanismus, um die Berechnungen sofort zu stoppen.

#### Sehen Sie dazu auch

- ▀ Methode ITcTaskNotification::NotifyCycleTimeExceeded() [▶ 170]
- ▀ Schnittstelle ITc FileAccess [▶ 142]
- ▀ Beispiel20a: FileIO-Cyclic Read / Write [▶ 273]

#### 12.4.13.1 Methode ITcTaskNotification::NotifyCycleTimeExceeded()

Wird aufgerufen, wenn die Zykluszeit zuvor abgelaufen ist

#### Syntax

```
virtual HRESULT TCOMAPI NotifyCycleTimeExceeded ();
```

#### Parameter

**ipTask:** (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

**context:** (Typ: ULONG\_PTR) Kontext

#### Rückgabewert

Typ: HRESULT

Gibt S\_OK zurück, wenn Dateioperation abgeschlossen ist

#### Beschreibung

Wird aufgerufen, wenn die Zykluszeit vorher überschritten war. Also nicht sofort bei Zeitüberschreitung, sondern danach.

#### Sehen Sie dazu auch

- ▀ ADS Return Codes [▶ 305]

## 12.4.14 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

Sym- bol	Name	Beschreibung
	<a href="#">TcAddRef [► 171]</a>	Inkrementiert den Referenzzähler.
	<a href="#">TcQueryInterface [► 171]</a>	Abfrage der Referenz an eine implementierte Schnittstelle über der IID
	<a href="#">TcRelease [► 172]</a>	Dekrementiert den Referenzzähler.

### Anmerkungen

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 muss ein Schnittstellenzeiger nach dem Aufruf von TcRelease() nicht dereferenziert werden.

### 12.4.14.1 Methode ITcUnknown:TcAddRef

Diese Methode inkrementiert den Referenzzähler.

### Syntax

```
ULONG TcAddRef( )
```

### Rückgabewert

Daraus resultierender Referenzzählwert.

### Beschreibung

Inkrementiert den Referenzzähler und gibt den neuen Wert zurück.

### 12.4.14.2 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 )
```

**iid:** (Typ: RITCID) Schnittstelle IID

**pipItf:** (Typ PPVOID) Zeiger auf Schnittstellenzeiger. Wird gesetzt, wenn der verlangte Schnittstellentyp von der entsprechenden Instanz verfügbar ist.

## Rückgabewert

Ein Rückgabewert S\_OK weist auf Erfolg hin.

Wenn die verlangte Schnittstelle nicht verfügbar ist, gibt die Methode ADS\_E\_NOINTERFACE zurück.

## 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(ITcUnknown* 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(ITcUnknown* 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.

### 12.4.14.3 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 selber.

## 12.5 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**

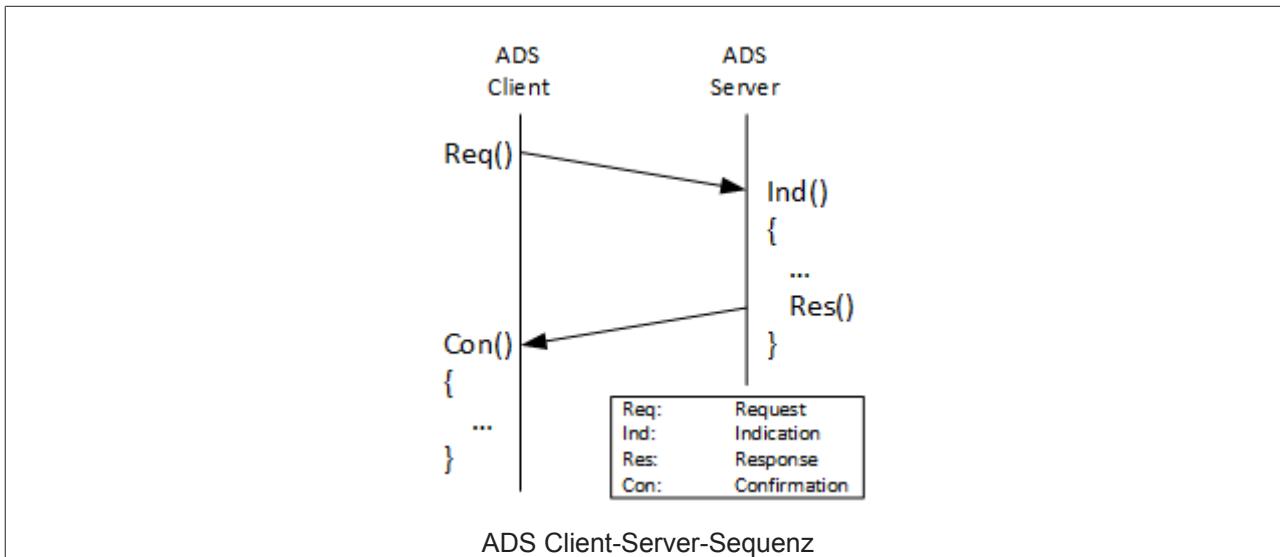
	<b>Name</b>	<b>Beschreibung</b>
≡	abs	Berechnet den absoluten Wert.
≡	atof	Konvertiert einen String (char *buf) in einen Double.
≡	BitScanForward	Sucht vom LSB zum MSB nach einem gesetzten Bit (1).
≡	BitScanReverse	Sucht vom MSB zum LSB nach einem gesetzten Bit (1).
≡	labs	Berechnet den absoluten Wert.
≡	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.
≡	vsprintf	Schreibt formatierte Daten in eine Zeichenfolge.

**Anmerkungen**

Alle Funktionen sind an die C++ Runtime Library angelehnt.

## 12.6 ADS-Kommunikation

Auf Client-Server-Prinzip basierende ADS (Abbildung 1). Eine ADS-Abfrage ruft die entsprechenden Indikationsmethoden auf der Serverseite auf. Die ADS-Antwort ruft die entsprechende Bestätigungs methode auf der Client-Seite auf.



In diesem Abschnitt werden sowohl die ausgehende, als auch die eingehende ADS-Kommunikation für TwinCAT 3 C++ Module beschrieben.

ADS-Befehlssatz	Beschreibung
<a href="#">AdsReadDeviceInfo [► 175]</a>	Mit diesem Befehl können die allgemeinen Geräteinformationen gelesen werden.
<a href="#">AdsRead [► 177]</a>	ADS-Lesebefehl, um Daten von einem ADS-Gerät abzufragen.
<a href="#">AdsWrite [► 179]</a>	ADS-Schreibbefehl, um Daten an ein ADS-Gerät zu übergeben.
<a href="#">AdsReadState [► 184]</a>	ADS-Befehl, um den Zustand von einem ADS-Gerät abzufragen.
<a href="#">AdsWriteControl [► 185]</a>	ADS-Steuerungsbefehl, um den Zustand von einem ADS-Gerät zu ändern.
<a href="#">AdsAddDeviceNotification [► 187]</a>	Variable beobachten. Der Client wird bei einem Ereignis informiert.
<a href="#">AdsDelDeviceNotification [► 189]</a>	Entfernt die Variable, die zuvor verbunden war.
<a href="#">AdsDeviceNotification [► 191]</a>	Wird für die Übermittlung des Gerät-Notification-Ereignisses verwendet.
<a href="#">AdsReadWrite [► 181]</a>	ADS-Schreib-/Lesebefehl. Mit einem Aufruf werden Daten zu einem ADS-Gerät übermittelt (Write) und dessen Antwortdaten gelesen (Read).

Die [ADS Return Codes \[► 305\]](#) gelten für die gesamte ADS-Kommunikation.

Schauen Sie sich als Einstieg das [Beispiel07: Empfang von ADS Notifications \[► 235\]](#)

### 12.6.1 AdsReadDeviceInfo

#### 12.6.1.1 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 );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.1.2 AdsReadDeviceInfoInd

Die Methode AdsDeviceInfoInd verweist auf einen ADS DeviceInfo Befehl zum Lesen der Identifizierung und Versionsnummer eines ADS Servers. Anschließend muss [AdsReadDeviceInfoRes \[► 176\]](#) aufgerufen werden.

## Syntax

```
void AdsReadDeviceInfoInd( AmsAddr& rAddr, ULONG invokeId );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

## Rückgabewert

void

### 12.6.1.3 AdsReadDeviceInfoRes

Die Methode AdsReadDeviceInfoRes sendet eine ADS Read Device Info. AdsReadDeviceInfoCon bildet das Gegenstück und wird anschließend aufgerufen.

## Syntax

```
int AdsReadDeviceInfoRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, CHAR name[ADS_FIXEDNAME-  
SIZE], AdsVersion version );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[► 305\]](#)

**name:** (Typ: char[ADS\_FIXEDNAMESIZE]) [in] enthält den Namen des Geräts.

**version:** (Typ: AdsVersion) [in] struct von Build (int), Revision (Byte) und Version (Byte) des Geräts

### Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes](#)

## 12.6.1.4 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](#) [▶ 175] aufgerufen werden.

### Syntax

```
void AdsReadDeviceInfoCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult,  
CHAR name[ADS_FIXEDNAMESIZE], AdsVersion version );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes](#) [▶ 305]

**name:** (Typ: char[ADS\_FIXEDNAMESIZE]) [in] enthält den Namen des Geräts.

**version:** (Typ: AdsVersion) [in] struct von Build (int), Revision (Byte) und Version (Byte) des Geräts

### Rückgabewert

void

## 12.6.2 AdsRead

### 12.6.2.1 AdsReadReq

Die Methode AdsReadReq ermöglicht das Senden eines ADS-Lesebefehls für die Datenübertragung von einem ADS-Gerät.

Beim Eingang der Antwort wird [AdsReadCon](#) [▶ 179] aufgerufen.

### Syntax

```
int AdsReadReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG  
cbLength );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData).

### Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

## 12.6.2.2 AdsReadInd

Die Methode AdsReadInd ermöglicht den Empfang von einer ADS-Leseanforderung. Die [AdsReadRes \[► 178\]](#) muss aufgerufen werden, um das Ergebnis zu senden.

### Syntax

```
void AdsReadInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG cbLength );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData).

### Rückgabewert

Typ: int

ADS Return Code - siehe [AdsStatuscodes](#)

## 12.6.2.3 AdsReadRes

Die Methode AdsReadRes ermöglicht das Senden von einer ADS-Leseantwort. [AdsReadCon \[► 179\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

### Syntax

```
int AdsReadRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Lesebefehls. Siehe [AdsStatuscodes \[► 305\]](#)

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die Daten sich befinden.

### Rückgabewert

Typ: int

ADS Return Code - siehe [AdsStatuscodes](#)

## 12.6.2.4 AdsReadCon

Die Methode AdsReadCon ermöglicht den Empfang von einer ADS-Lesebestätigung. Das empfangende Modul muss diese Methode bereitstellen.

Das Pendant [AdsReadReq \[▶ 177\]](#) bildet muss vorher aufgerufen worden sein.

### Syntax

```
void AdsReadCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Lesebefehls. Siehe [AdsStatuscodes \[▶ 305\]](#)

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

### Rückgabewert

void

## 12.6.3 AdsWrite

### 12.6.3.1 AdsWriteReq

Die Methode AdsWriteReq ermöglicht das Senden eines ADS-Schreibbefehls, für Übertragung von Daten an ein ADS-Gerät.

Beim Eingang der Antwort wird [AdsWriteCon \[▶ 181\]](#) aufgerufen.

### Syntax

```
int AdsWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG cbLength, PVOID pData );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.3.2 AdsWriteInd

Die Methode AdsWriteInd gibt einen ADS-Schreibbefehl an, um Daten zu einem ADS-Gerät zu übermitteln. Die [AdsWriteRes \[► 180\]](#) muss aufgerufen werden, um den Vorgang zu bestätigen.

#### Syntax

```
void AdsWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG cbLength, PVOID pData );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

## Rückgabewert

void

#### Sehen Sie dazu auch

[ADS Return Codes \[► 305\]](#)

### 12.6.3.3 AdsWriteRes

Die Methode AdsWriteRes sendet eine ADS-Schreibantwort. [AdsWriteCon \[► 181\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

## Syntax

```
int AdsWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[► 305\]](#)

## Rückgabewert

Typ: int

ADS Return Code - siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.3.4 AdsWriteCon

Die Methode AdsWriteCon ermöglicht den Empfang einer ADS-Schreibbestätigung. Das empfangende Modul muss diese Methode bereitstellen.

[AdsWriteReq \[► 179\]](#) bildet das Gegenstück und muss zuvor aufgerufen worden sein.

## Syntax

```
void AdsWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[► 305\]](#)

## Rückgabewert

void

### 12.6.4 AdsReadWrite

#### 12.6.4.1 AdsReadWriteReq

Die Methode AdsReadWriteReq ermöglicht das Senden eines ADS-Lese-/Schreibbefehls für die Datenübergabe an ein und von einem ADS-Gerät. Die [AdsReadWriteCon \[► 183\]](#) wird beim Eingang der Antwort aufgerufen.

## Syntax

```
int AdsReadWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG cbReadLength, ULONG cbWriteLength, PVOID pData );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbReadLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData)

**cbWriteLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.4.2 AdsReadWriteInd

Die Methode AdsReadWriteInd gibt einen ADS-Lese-/Schreibbefehl an, um Daten zu einem ADS-Gerät zu übermitteln. Die [AdsReadWriteRes \[► 184\]](#) 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 );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbReadLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData)

**cbWriteLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

## Rückgabewert

void

### 12.6.4.3 AdsReadWriteRes

Die Methode AdsReadWriteRes ermöglicht den Empfang von einer ADS-Lese-/Schreibbestätigung. [AdsReadWriteCon \[▶ 183\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

#### Syntax

```
int AdsReadWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[▶ 305\]](#)

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

#### Rückgabewert

Typ: int

ADS Return Code - siehe [AdsStatuscodes](#)

### 12.6.4.4 AdsReadWriteCon

Die Methode AdsReadWriteCon ermöglicht den Empfang von einer ADS-Lese-/Schreibbestätigung. Das empfangende Modul muss diese Methode bereitstellen.

[AdsReadWriteReq \[▶ 181\]](#) bildet das Gegenstück und muss zuvor aufgerufen.

#### Syntax

```
void AdsReadWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[▶ 305\]](#)

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

#### Rückgabewert

void

## 12.6.5 AdsReadState

### 12.6.5.1 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 \[▶ 185\]](#) aufgerufen.

#### Syntax

```
int AdsReadStateReq(AmsAddr& rAddr, ULONG invokeId);
```

#### Parameter

**rAddr:** (Typ: AmsAddr) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

#### Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[▶ 305\]](#)

### 12.6.5.2 AdsReadStateInd

Die Methode AdsReadStateInd zeigt einen ADS-Statuslesebefehl an für das Lesen des ADS-Zustands und des Gerätezustands von einem ADS-Gerät. Die [AdsReadStateRes \[▶ 184\]](#) muss aufgerufen werden, um das Ergebnis zu senden.

#### Syntax

```
void AdsReadStateInd( AmsAddr& rAddr, ULONG invokeId );
```

#### Parameter

**rAddr:** (Typ: AmsAddr) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

#### Rückgabewert

void

### 12.6.5.3 AdsReadStateRes

Die Methode AdsWriteRes ermöglicht das Senden von einer ADS-Statusleseantwort. [AdsReadStateCon \[▶ 185\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

#### Syntax

```
int AdsReadStateRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT devi-  
ceState );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[► 305\]](#)

**adsState:** (Typ: USHORT) [in] enthält den ADS-Zustand des Geräts

**deviceState:** (Typ: USHORT) [in] enthält den Gerätetestatus des Geräts

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes](#)

### 12.6.5.4 AdsReadStateCon

Die Methode AdsWriteCon ermöglicht den Empfang von einer ADS-Statuslesebestätigung. Das empfangende Modul muss diese Methode bereitstellen.

[AdsReadStateReq \[► 184\]](#) bildet das Gegenstück und muss zuvor aufgerufen.

## Syntax

```
void AdsReadStateCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT deviceState );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[► 305\]](#)

**adsState:** (Typ: USHORT) [in] enthält den ADS-Zustand des Geräts

**deviceState:** (Typ: USHORT) [in] enthält den Gerätetestatus des Geräts

## Rückgabewert

void

### 12.6.6 AdsWriteControl

#### 12.6.6.1 AdsWriteControlReq

Mit der Methode AdsWriteControlReq kann ein ADS-Schreibsteuerungsbefehl zur Änderung des ADS- und Gerätetestatus eines ADS Servers gesendet werden. Beim Eingang der Antwort wird [AdsWriteControlCon \[► 187\]](#) aufgerufen.

## Syntax

```
int AdsWriteControlReq( AmsAddr& rAddr, ULONG invokeId, USHORT adsState,
                        USHORT deviceState, ULONG cbLength, PVOID pData );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**adsState:** (Typ: USHORT) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**deviceState:** (Typ: USHORT) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der Daten (pData)

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.6.2 AdsWriteControlInd

Mit der Methode AdsWriteControlInd kann ein ADS-Schreibsteuerungsbefehl zur Änderung des ADS- und Gerätestatus eines ADS-Geräts gesendet werden. Die [AdsWriteControlRes \[► 187\]](#) muss aufgerufen werden, um den Vorgang zu bestätigen.

## Syntax

```
void AdsWriteControlInd( AmsAddr& rAddr, ULONG invokeId, USHORT adsState, USHORT deviceState,  
ULONG cbLength, PVOID pDeviceData );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**adsState:** (Typ: USHORT) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**deviceState:** (Typ: USHORT) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**cbLength:** (Typ: ULONG) [in] enthält die Länge, in Bytes, der Daten (pData)

**pData:** (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

## Rückgabewert

void

### 12.6.6.3 AdsWriteControlRes

Die Methode AdsWriteControlRes ermöglicht das Senden von einer ADS-Schreibsteuerungsantwort. [AdsWriteControlCon \[▶ 187\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

#### Syntax

```
int AdsWriteControlRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[▶ 305\]](#)

#### Rückgabewert

Typ: int

ADS Return Code - siehe [AdsStatuscodes \[▶ 305\]](#)

### 12.6.6.4 AdsWriteControlCon

Die Methode AdsWriteCon ermöglicht den Empfang von einer ADS-Schreibsteuerungsbestätigung. Das empfangende Modul muss diese Methode bereitstellen.

[AdsWriteControlReq \[▶ 185\]](#) bildet das Gegenstück und muss zuvor aufgerufen

#### Syntax

```
void AdsWriteControlCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[▶ 305\]](#)

#### Rückgabewert

void

### 12.6.7 AdsAddDeviceNotification

#### 12.6.7.1 AdsAddDeviceNotificationReq

Die Methode AdsAddDeviceNotificationReq ermöglicht das Senden eines ADS-Hinzufügen-Geräte-Notification-Befehls, um eine Geräte-Notification zu einem ADS-Gerät hinzuzufügen.

[AdsAddDeviceNotificationCon \[▶ 189\]](#) wird beim Eingang der Antwort aufgerufen.

## Syntax

```
int AdsAddDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset,
                                AdsNotificationAttrib noteAttrib);
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**noteAttrib:** (Typ: AdsNotificationAttrib) [in] enthält die Spezifikation der Notification-Parameter (cbLength, TransMode, MaxDelay).

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.7.2 AdsAddDeviceNotificationInd

Die Methode AdsAddDeviceNotificationInd sollte das Senden von [AdsDeviceNotification \[► 191\]](#) ermöglichen. Die [AdsAddDeviceNotificationRes \[► 189\]](#) muss aufgerufen werden, um den Vorgang zu bestätigen.

## Syntax

```
void AdsAddDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset,
                                  AdsNotificationAttrib noteAttrib );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**indexGroup:** (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**indexOffset:** (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

**noteAttrib:** (Typ: AdsNotificationAttrib) [in] enthält die Spezifikation der Notification-Parameter (cbLength, TransMode, MaxDelay).

## Rückgabewert

void

### 12.6.7.3 AdsAddDeviceNotificationRes

Die Methode AdsAddDeviceNotificationRes ermöglicht das Senden von einer ADS-Gerät-Hinzufügen-Notification-Antwort. [AdsAddDeviceNotificationCon \[▶ 189\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

#### Syntax

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes **Handle:** (Typ: ULONG) [in] Handle auf generierte Geräte-Notification

#### Rückgabewert

void

### 12.6.7.4 AdsAddDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon bestätigt eine ADS-Gerät-Hinzufügen-Notification-Anforderung. [AdsAddDeviceNotificationReq \[▶ 187\]](#) bildet das Gegenstück und muss zuvor aufgerufen.

#### Syntax

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes **Handle:**

**handle:** (Typ: ULONG) [in] Handle auf generierte Geräte-Notification

#### Rückgabewert

void

### 12.6.8 AdsDelDeviceNotification

#### 12.6.8.1 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 \[▶ 191\]](#) wird beim Eingang der Antwort aufgerufen.

## Syntax

```
int AdsDelDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG hNotification );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen

**hNotification:** (Typ: ULONG) [in] enthält das Handle der zu entfernenden Notification

## Rückgabewert

Typ: int

Fehlercode - Siehe [AdsStatuscodes \[► 305\]](#)

### 12.6.8.2 AdsDelDeviceNotificationInd

Die Methode AdsAddDeviceNotificationCon ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification-Bestätigung. Das empfangende Modul muss diese Methode bereitstellen. Die [AdsDelDeviceNotificationRes \[► 190\]](#) muss aufgerufen werden, um den Vorgang zu bestätigen.

## Syntax

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe [AdsStatuscodes \[► 305\]](#)

## Rückgabewert

void

### 12.6.8.3 AdsDelDeviceNotificationRes

Die Methode AdsAddDeviceNotificationRes ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification. [AdsDelDeviceNotificationCon \[► 191\]](#) bildet das Gegenstück und wird anschließend aufgerufen.

## Syntax

```
int AdsDelDeviceNotificationRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

## Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeId:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Befehls. Siehe [AdsStatuscodes \[▶ 305\]](#)

### Rückgabewert

Int

Gibt das Ergebnis des ADS-Befehls zurück. Siehe [AdsStatuscodes \[▶ 305\]](#)

## 12.6.8.4 AdsDelDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification-Bestätigung. Das empfangende Modul muss diese Methode bereitstellen.

[AdsDelDeviceNotificationReq \[▶ 189\]](#) bildet das Gegenstück und muss zuvor aufgerufen.

### Syntax

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des gesendeten Befehls, die Invokeld wird vom Quellgerät spezifiziert und dient der Identifizierung der Befehle.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls, siehe [AdsStatuscodes \[▶ 305\]](#).

### Rückgabewert

void

## 12.6.9 AdsDeviceNotification

### 12.6.9.1 AdsDeviceNotificationReq

Die Methode AdsAddDeviceNotificationReq ermöglicht das Senden einer ADS-Geräte-Notification, um ein ADS-Gerät zu informieren. Die [AdsDeviceNotificationInd \[▶ 192\]](#) wird auf dem Gegenstück aufgerufen.

### Syntax

```
int AdsDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength, AdsNotificationStream notifications[] );
```

### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis der Geräte-Notification-Anzeige.

**notifications[]:** (Typ: AdsNotificationStream) [in] enthält Informationen der Geräte-Notification(s).

## Rückgabewert

Typ: int

ADS Return Code - siehe [AdsStatuscodes \[▶ 305\]](#)

### 12.6.9.2 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 \[▶ 192\]](#) muss auf Seite der [AdsDeviceNotificationReq \[▶ 191\]](#) aufgerufen werden um die Übermittlung zu überprüfen.

#### Syntax

```
void AdsDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength,  
AdsNotificationStream* pNotifications );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**cbLength:** (Typ: ULONG) [in] enthält die Länge von pNotifications

**pNotifications:** (Typ: AdsNotificationStream\*) [in] Zeiger auf die Notifications. Dieses Array besteht aus AdsStampHeader mit Notification Handle und Daten via AdsNotificationSample.

#### Rückgabewert

void

### 12.6.9.3 AdsDeviceNotificationCon

Mit der Methode AdsAddDeviceNotificationCon kann der Absender die Übermittlung einer ADS-Geräte-Notification überprüfen.

[AdsDeviceNotificationReq \[▶ 191\]](#) muss dafür zuvor aufgerufen werden.

#### Syntax

```
void AdsDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

#### Parameter

**rAddr:** (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

**invokeld:** (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die Invokeld wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

**nResult:** (Typ: ULONG) [in] enthält das Ergebnis der Geräte-Notification-Anzeige.

#### Rückgabewert

void

## 12.7 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 Systemen wird die FPU verwendet.



Hinweis

### 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

Mathematische Funktionen

	<b>Name</b>	<b>Beschreibung</b>
=!	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.
=!	cubic_	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.
<b>Hilfsfunktionen</b>		
=!	finite_	Ermittelt, ob der angegebene Wert endlich ist.
=!	isnan_	Ermittelt, ob der gegebene Wert keine Zahl ist (NAN, „not a number“).

	Name	Beschreibung
=	rands_	Berechnet eine Pseudo-Zufallszahl zwischen 0 und 32767. Der Parameter holdrand wird zufällig gesetzt und bei jedem Aufruf geändert.

### Anmerkungen

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.](#)

## 12.8 Zeitfunktionen

TwinCAT bietet Funktionen für Zeitumwandlung.

Diese Funktionen werden in TcTimeConversion.h, die Teil von TwinCAT SDK ist, deklariert.

	Name	Beschreibung
=	TcDayOfWeek	Ermittelt den Wochentag (Sonntag ist 0)
=	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);	Wandelt die gegebene Systemzeit in eine Dateizeit um.
=	TcFileTimeToSystemTime(const FILETIME *lpFileTime, SYSTEMTIME* lpSystemTime);	Wandelt die gegebene Dateizeit in eine Systemzeit um.
=	TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, ULONGLONG& ul64FileTime);	Wandelt die gegebene Systemzeit in eine Dateizeit um (ULONGLONG Format)
=	TcFileTimeToSystemTime(const ULONGLONG& ul64FileTime, SYSTEMTIME* lpSystemTime);	Wandelt die gegebene Dateizeit (ULONGLONG Format) in eine Systemzeit um.

## 12.9 STL / Container

TwinCAT 3 C++ unterstützt STL bezüglich

- List
- Map
- Set
- Stack
- String
- Vector
- WString
- Algorithm (wie binary\_search)
  - Siehe c:\TwinCAT\3.x\Sdk\Include\Stl\Stl\algorithm für eine konkrete Liste der unterstützten Algorithmen

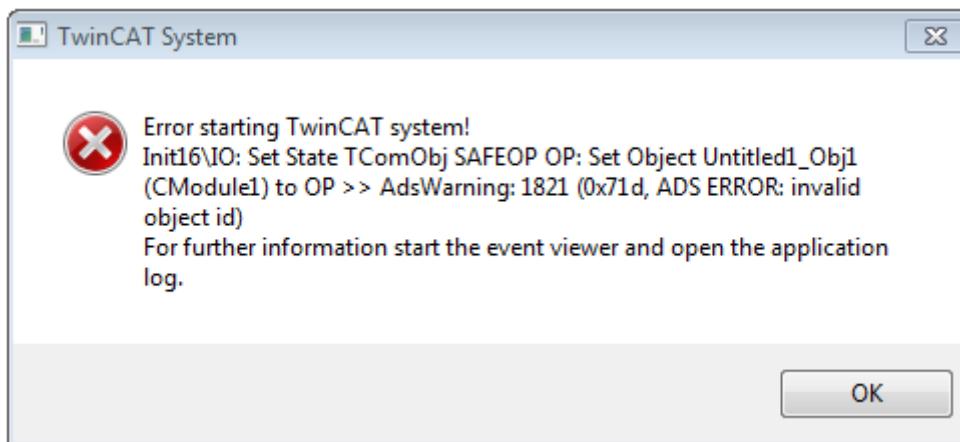
Bitte beachten:

- 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 [hier \[► 137\]](#).

## 12.10 Fehlermeldungen - Verständnis

Es mag den Anschein haben, dass die Fehlermeldungen von TwinCAT überladen sind.



Sie enthalten sehr ausführliche Informationen über den aufgetretenen Fehler.  
So beschreibt z.B. der Screenshot oben Folgendes:

- 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

Die Folge ist, dass man die Methode „SetObjStateSP()“, die für diesen Übergang verantwortlich ist, genauer untersuchen muss. Im Falle des generierten Standardcodes sieht man, dass das Hinzufügen des Moduls dort stattfindet.

Der Grund für das Auftreten dieses Fehlers besteht darin, dass diesem Modul kein Task zugewiesen wurde, also kann das Modul keinen Task haben, in dem es ausgeführt wird.

## 12.11 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.

The screenshot shows the Beckhoff TwinCAT Project Manager interface. The Solution Explorer on the left lists the project structure under 'TwinCAT Project1'. The 'Source Files' folder contains 'Module1.cpp', which is currently open in the code editor. The code in 'Module1.cpp' demonstrates trace logging with various levels (tlAlways, tlError, tlWarning, tlInfo, tlVerbose) based on a cycle counter. Below the code editor is the 'Error List' window, which displays 10 errors, 9 warnings, and 49 messages.

```

// Sample to showcase trace logs
ULONGLONG cnt = 0;
if (SUCCEEDED(ipTask->GetCycleCounter(&cnt)))
{
    if (cnt%500 == 0)
        m_Trace.Log(tlAlways, FNNAME "Level tlAlways: cycle=%llu", cnt);
    if (cnt%510 == 0)
        m_Trace.Log(tlError, FNNAME "Level tlError: cycle=%llu", cnt);

    if (cnt%520 == 0)
        m_Trace.Log(tlWarning, FNNAME "Level tlWarning: cycle=%llu", cnt);

    if (cnt%530 == 0)
        m_Trace.Log(tlInfo, FNNAME "Level tlInfo: cycle=%llu", cnt);

    if (cnt%540 == 0)
        m_Trace.Log(tlVerbose, FNNAME "Level tlVerbose: cycle=%llu", cnt);
}

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

return hr;
}

```

Level	Log Level
Level 0	tlAlways
Level 1	tlError
Level 2	tlWarning
Level 3	tlInfo
Level 4	tlVerbose

## 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 „>>>“
  - FNNAMEA: Wird innerhalb einer Funktion verwendet; drückt den Funktionsnamen
  - FLEAVEA: Wird beim Verlassen einer Funktion verwendet; drückt den Funktionsnamen gefolgt von „<<<“
- Formatspezifizierer

## 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;

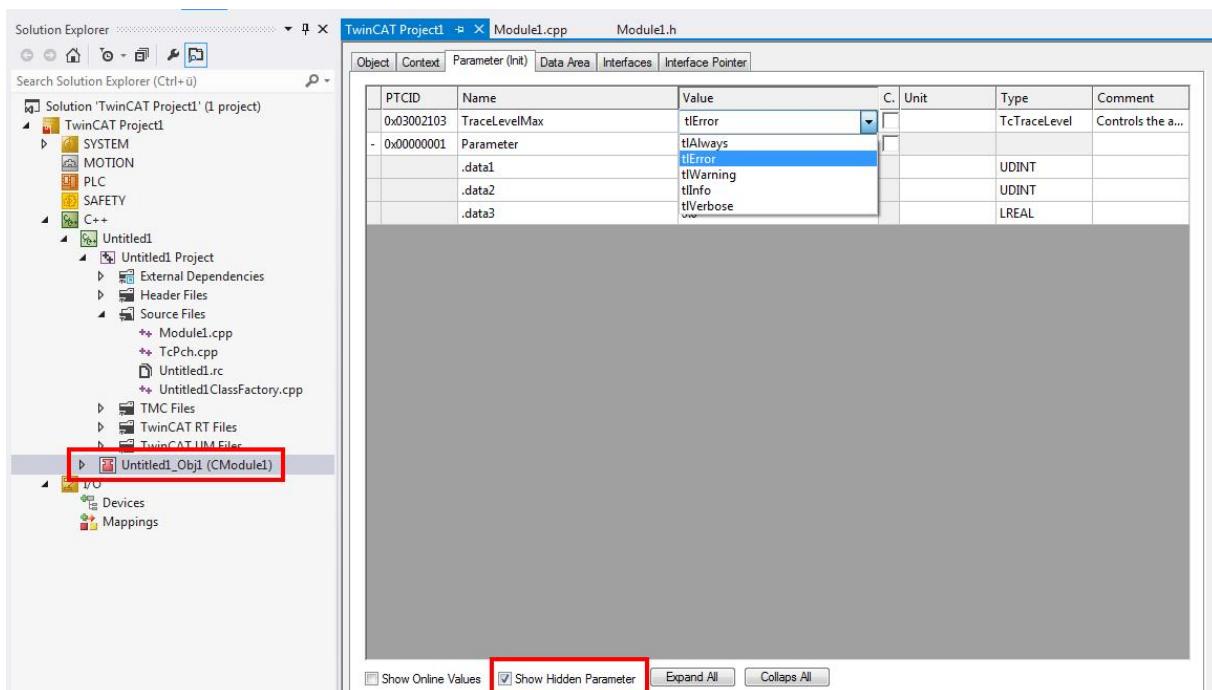
    return hr;
}

```

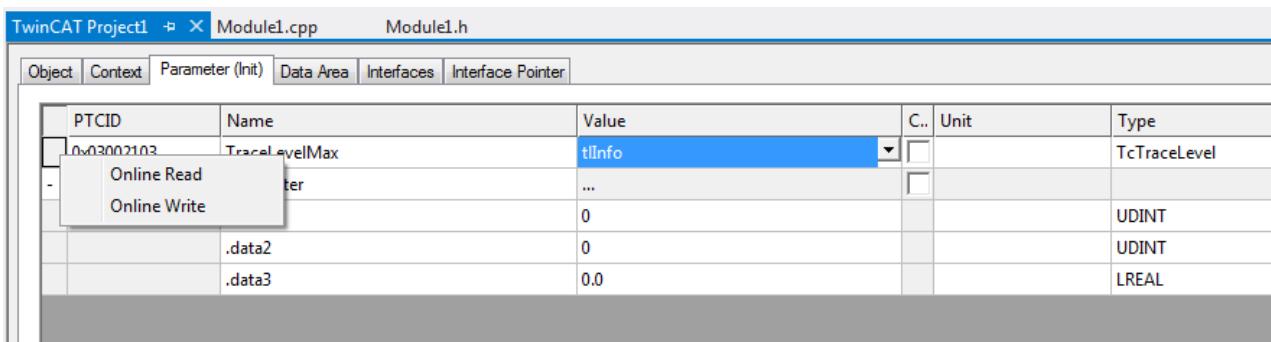
## Verfolgungsebene verwenden

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“.



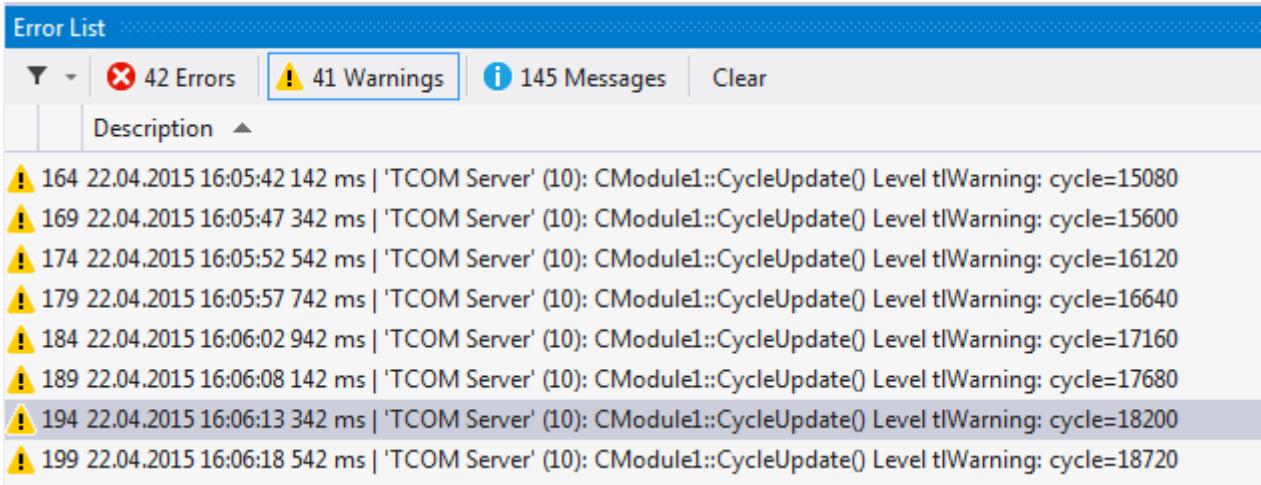
Alternativ dazu kann die Verfolgungsebene auch während der Laufzeit geändert werden, indem man zur Instanz geht, eine Ebene bei „Value“ für TraceLevelMax-Parameter auswählt, dann Rechtsklick vor der ersten Spalte und „Online Write“ auswählen.



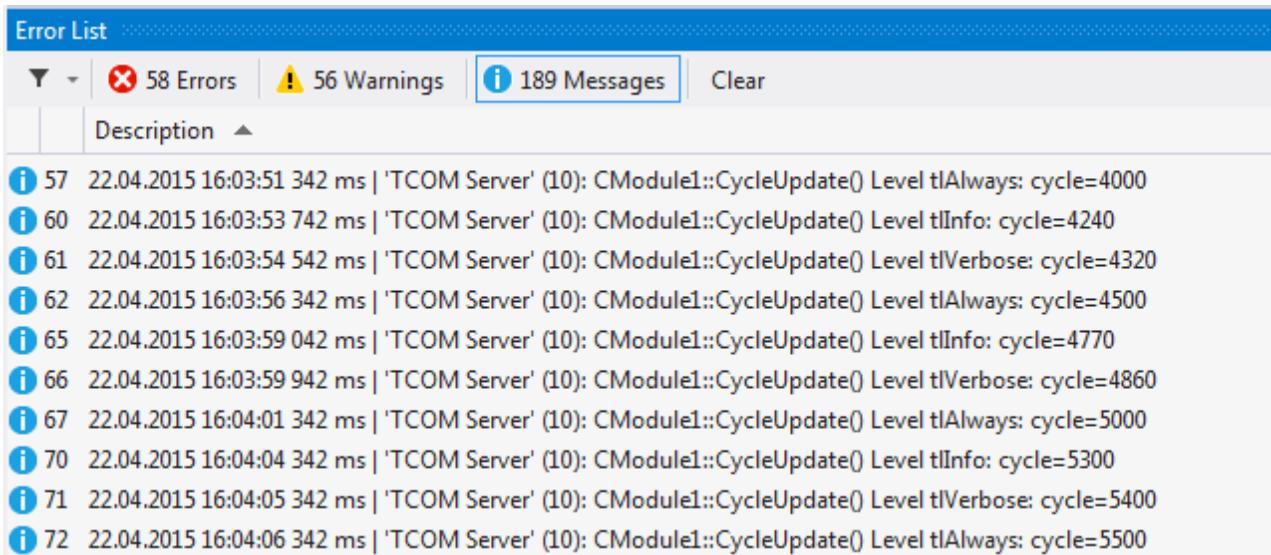
### Meldungskategorien filtern

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...?

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](#) [▶ 75] sowie auch die Nutzung der Assistenten zum [Anlegen von Modulenklassen](#) [▶ 76].

Zusätzlich können die Projekteigenschaften gesetzt werden, der TMC Code Generator und das Publishen von Modulen aufgerufen werden. Die zugehörige [Dokumentation](#) [▶ 317] ist Teil des Automation Interfaces.

Unabhängig von der Programmiersprache kann der [Zugriff auf, die Erstellung von und den Umgang mit TcCOM Modulen](#) [▶ 321] relevant sein.

Von dort aus können gewöhnliche Aufgaben des Systemmanagers wie Verknüpfung von Variablen ausgeführt werden.

### 13.2 Windows 10 als Zielsystem

Bei Windows 10 Zielsystemen können die übertragenen Dateien nicht überschrieben werden, sondern müssen vorher umbenannt werden. Hierfür muss im [Deployment des TMC Editors](#) [▶ 121] die Option „Rename Destination“ aktiviert werden.

### 13.3 Module veröffentlichen

Der Abschnitt [Module exportieren](#) [▶ 43] beschreibt wie TwinCAT-Module veröffentlicht werden, sodass sie auf ein beliebiges TwinCAT System übertragen und [importiert](#) [▶ 44] werden können.

Das Engineering System (XAE) muss sich nicht notwendigerweise auf den gleichen Plattformtyp wie das Ausführungssystem befinden. Hierzu erstellt TwinCAT im Verlauf der Veröffentlichung alle Versionen des Moduls.

Bestimmte Anwendungsfälle erfordern eine Anpassung der Veröffentlichung der Module:

- Beim Arbeiten in einer reinen 32Bit (x86) Umgebung können die x64 Erstellungen übersprungen werden, sodass keine Zertifikate benötigt werden.
- Die User Mode (UM) Erstellungen können übersprungen werden, wenn sie nicht verwendet werden.

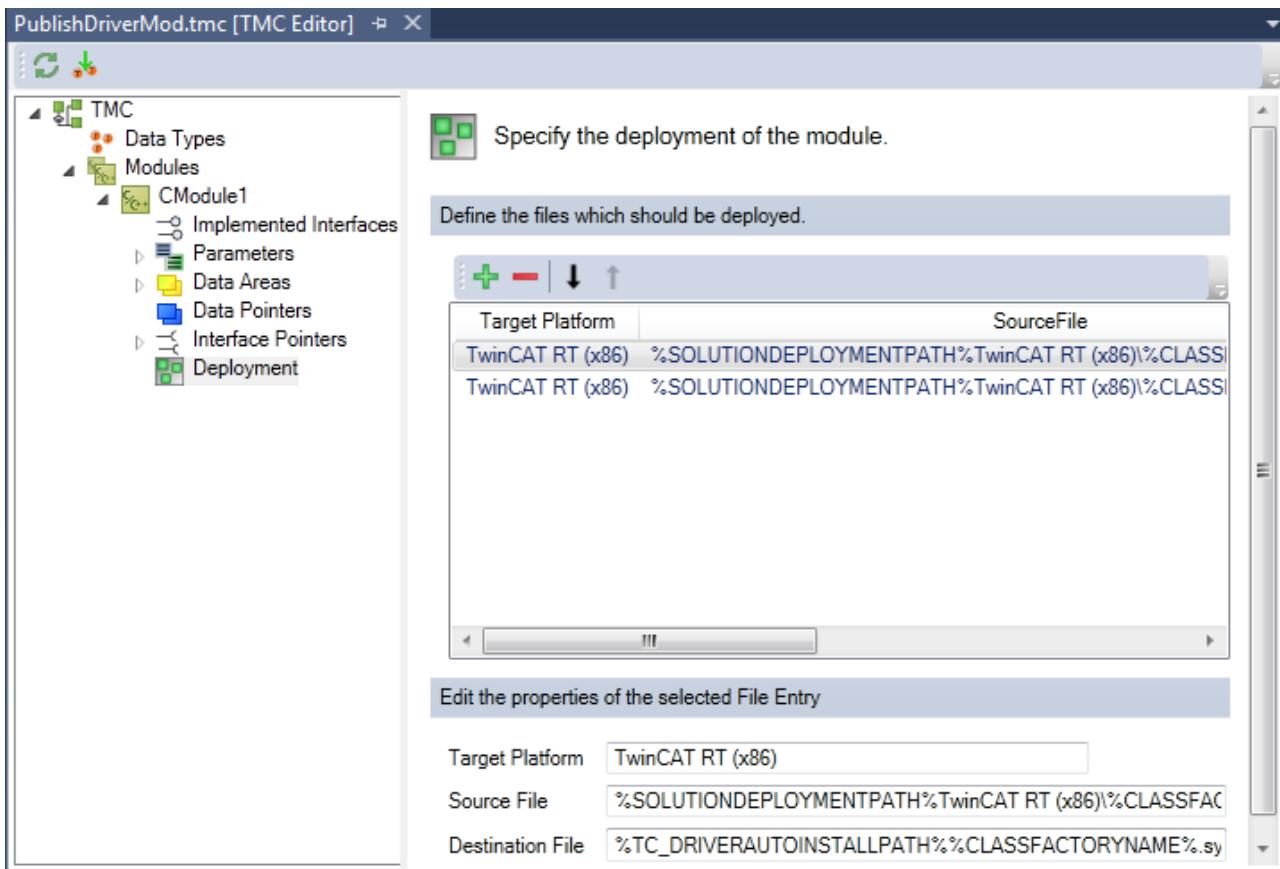


#### Migration

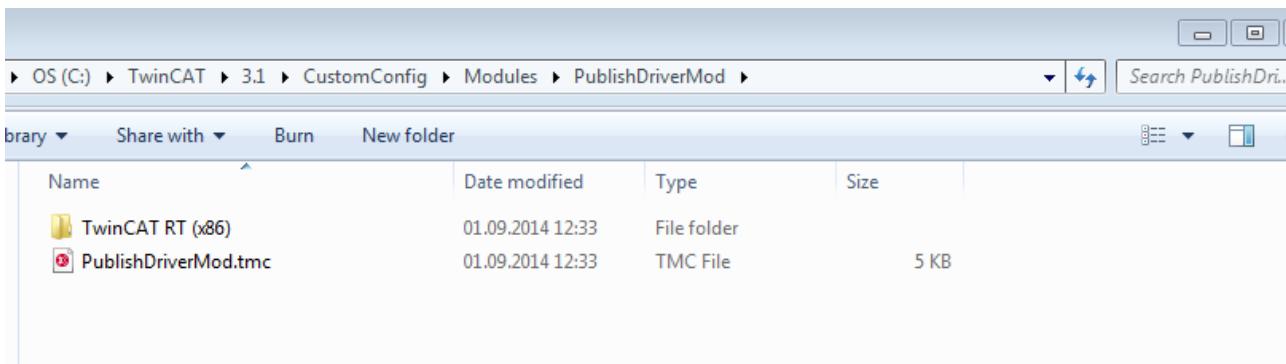
Wenn ein Build nicht im veröffentlichten Modul enthalten ist, kann die entsprechende Plattform nicht als Ausführungssystem verwendet werden.

Bitte die entsprechenden Zielplattformen in [Bereitstellung](#) [▶ 121] des TMC Editors hinzufügen / entfernen.

Folgende Bereitstellungskonfiguration als Beispiel:



stellt ausschließlich die TwinCAT RT (x86) Build zur Verfügung:



## 13.4 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:TcPublishModule /p:TcPublishDestinationBaseFolder=c:\temp
```

Der Parameter `CppProject.vcxproj` muss entsprechend der vorhandenen Projektdatei angepasst werden.

Der Parameter `TcPublishDestinationBaseFolder` ist dabei optional. Wenn er nicht angegeben wird, wird die normale Ablage verwendet (`C:\TwinCAT\3.x\CustomConfig\Modules`).

## 13.5 Clone

Die Laufzeitdateien können mittels Dateikopie von einer zur anderen Maschine übertragen werden, wenn diese 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“.

- ✓ Bitte den Ordner C:\TwinCAT\3.x\Boot auf der Quell-Maschine leeren.
- 1. Das Modul auf der Quellmaschine erstellen (oder aktivieren).
- 2. Den Ordner C:\TwinCAT\3.x\Boot von der Quelle zum Ziel übertragen.
- 3. Den Treiber selber von C:\TwinCAT\3.x\Driver\AutoInstall\\*\*MYDRIVER.sys\*\* übertragen.
- 4. Optional auch \*\*MYDRIVER.pdb\*\* übertragen.
- 5. Falls Treiber neu auf einer Maschine sind:

**■ HINWEIS! TwinCAT muss einmalig eine Registrierung ausführen. TwinCAT dafür per SysTray (Rechts-Klick->System->Start/Restart) in den RUN schalten.**

**Alternativ kann dieser Aufruf verwendet werden (%1 als Treibernamen ersetzen):**

```
sc create %1 binPath= c:\TwinCAT\3.1\Driver\AutoInstall\%1.sys type= kernel
start= auto group= "file system" DisplayName= %1 error= normal
```

- ⇒ Die Zielmaschine kann gestartet werden.

 <b>Hinweis</b>	<b>Umgang mit Lizenzen</b> <p>Beachten Sie, dass die Lizenzen nicht auf diese Weise übertragen werden können. Verwenden Sie bitte vorinstallierte Lizenzen, Volumenlizenzen oder andere Mechanismen, um Lizenzen bereitzustellen.</p>
---	---

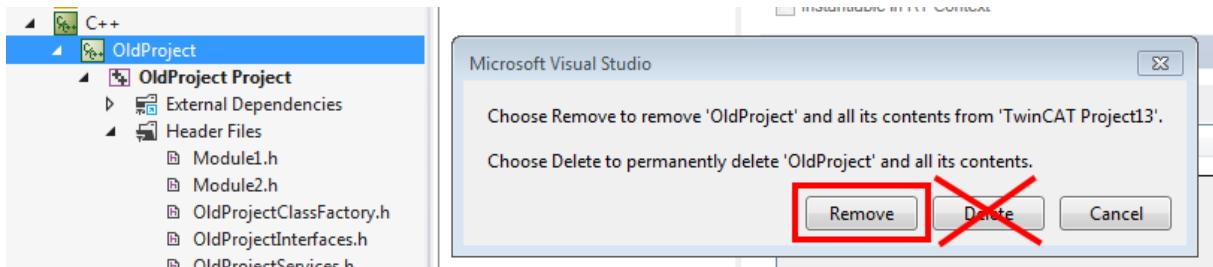
## 13.6 Umbenennen von TwinCAT-C++ Projekten

Das Umbenennen von TwinCAT-C++ Projekten ist nicht automatisiert möglich.

An dieser Stelle wird eine Anleitung gegeben, wie manuell ein Projekt umbenannt werden kann.

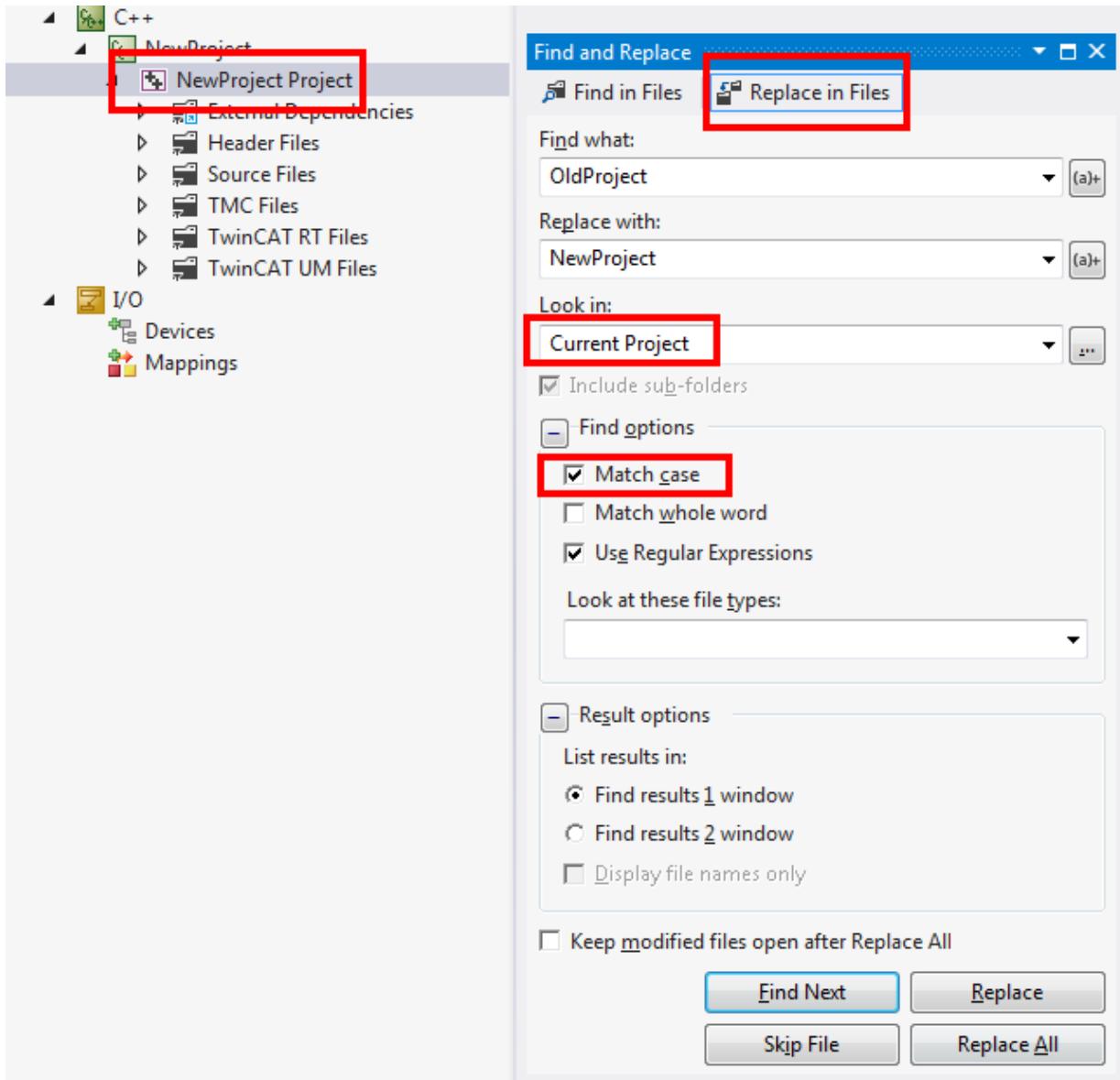
Zusammenfassend kann gesagt werden, dass das C++ Projekt zusammen mit den entsprechenden Dateien umbenannt wird.

- ✓ 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 entfernt („Remove“).



3. Kompilate des „OldProject“ 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 im Visual Studio integriert ist, müssen Sie diesen Schritt also wieder im Versionsverwaltungssystem ausführen. Alle Vorkommen im Quellcode (case-sensitive) ersetzen:  
Aus „OLDPART“ wird „NEWPART“ und  
Aus „OldPart“ wird „NewPart“. Hierfür bietet sich der „Find and Replace“ Dialog des Visual Studios an, wobei das „NewPart Project“ im Solution Explorer (vgl. Screenshot) ausgewählt sein muss.

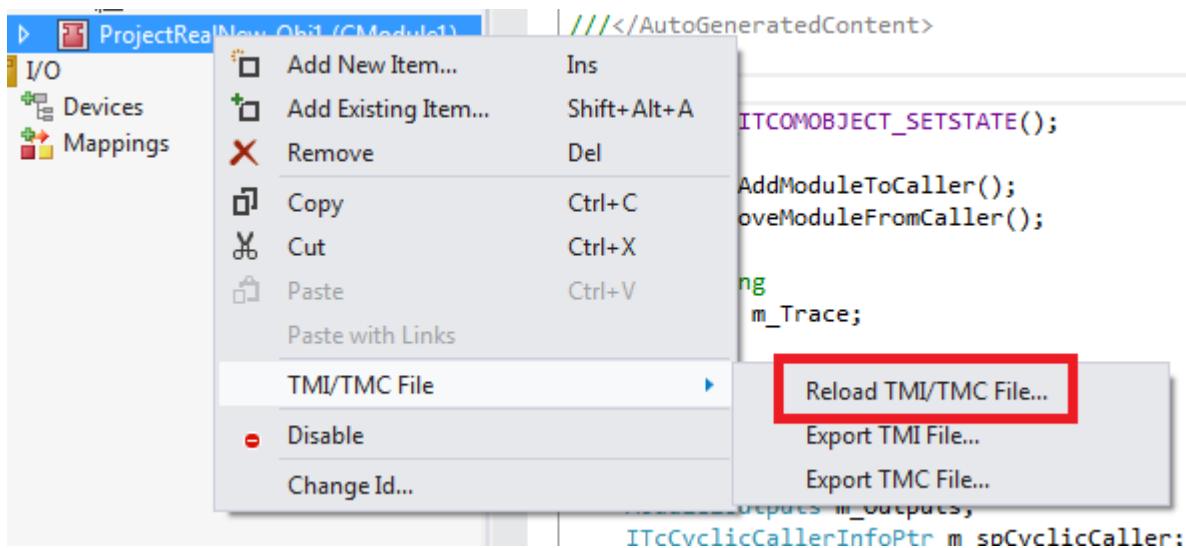


#### Inkorrekter Quellcode

Durch die einfache Umbenennung aller Vorkommen der Zeichenfolge kann es zu inkorrektem Quellcode kommen. Beispielsweise, wenn der Projektnamen 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.

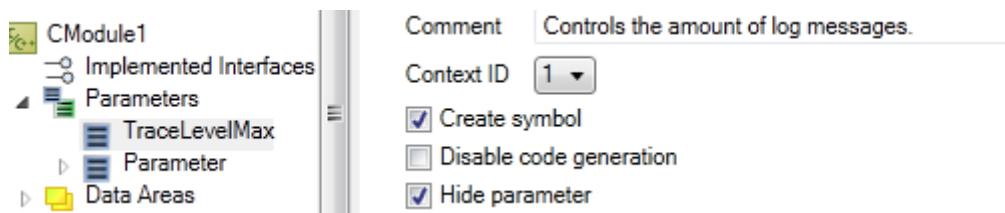


B) Alternativ führen Sie dies über System->TcCOM Objects und den Tab „Project Objects“ durch Rechts-Klick auf die OTCID aus.

2. Verschieben Sie System->TcCOM in das Projekt.
3. Bereinigen Sie das / die Zielsystem(e).  
Löschen Sie die Dateien OldProject.sys/.pdb im C:\TwinCAT\3.x\Driver\AutoInstall.
4. Testen Sie das Projekt.

## 13.7 Zugriff auf Variablen über ADS

Variablen von C++ Modulen sind über ADS erreichbar, wenn die Variable 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
```

## 13.8 TcCallAfterOutputUpdate für C++ Module

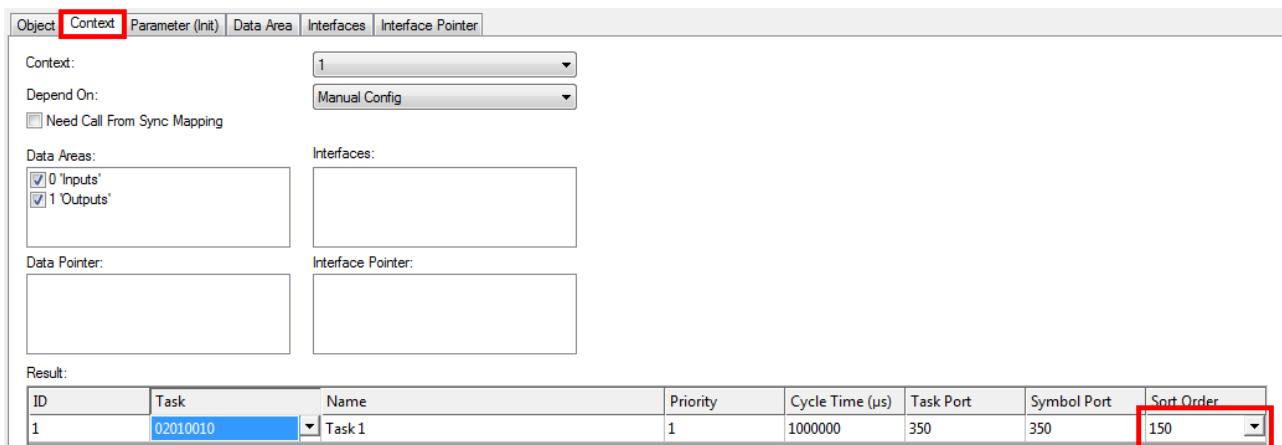
Vergleichbar mit dem SPS Attribut TcCallAfterOutputUpdate können C++ Module nach der Ausgangs-Aktualisierung aufgerufen werden.

Äquivalent zu [ITcCyclic \[▶ 139\]](#) Schnittstelle: Verwenden Sie bitte die [ITcPostCyclic \[▶ 162\]](#) Schnittstelle

## 13.9 Reihenfolgebestimmung der Ausführung in einem Task

Es können verschiedene Modulinstanzen einem Task zugewiesen werden, sodass der Benutzer einen Mechanismus benötigt, um die Reihenfolge der Ausführung im Task festzulegen.

Das Schlüsselwort lautet „Sort Order“, das im [Kontext \[▶ 125\]](#) des [TwinCAT Module Instance Configurator \[▶ 123\]](#) konfiguriert wird.



Siehe [Beispiel26: Ausführungsreihenfolge in einem Task](#) [▶ 279], wie das zu implementieren ist.

## 13.10 Stack Size > 4kB verwenden

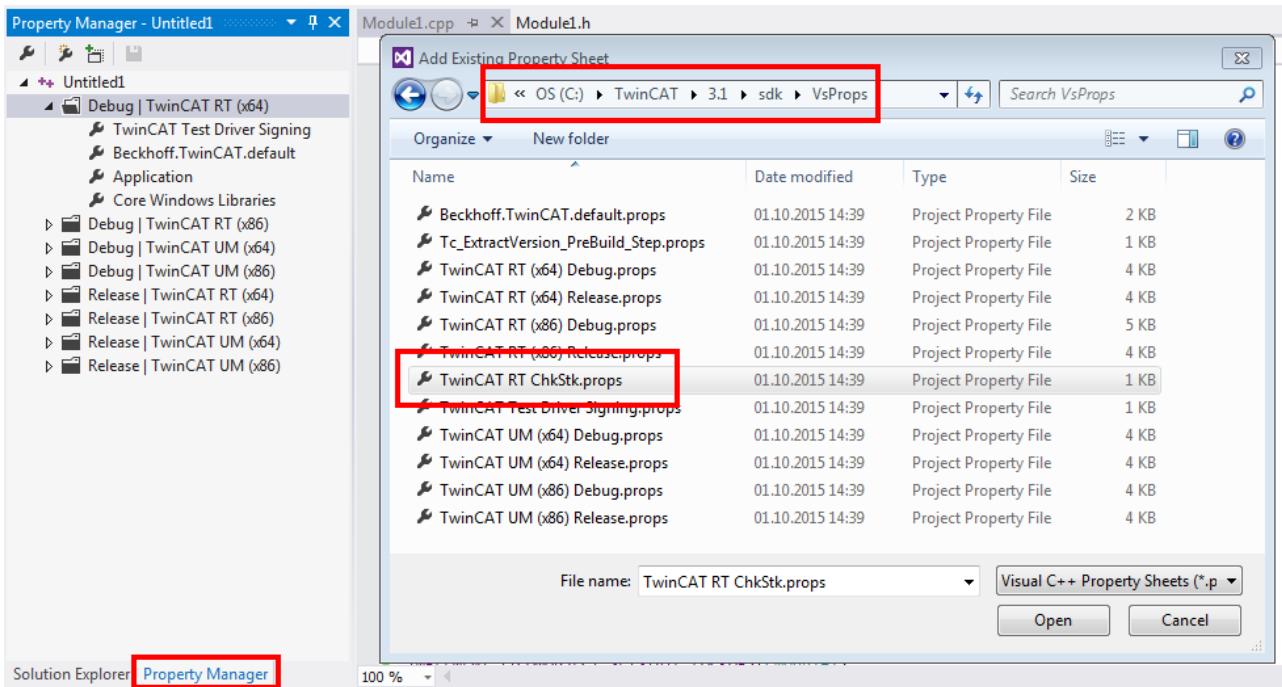
Wird von einer Methode eine Stack-Größe von mehr als 4KB genutzt, wird bei RealTime Modulen zusätzlich die `_chkstk` Funktion aufgerufen.

Aus diesem Grund tritt die Fehlermeldung

Error LNK2001: unresolved external symbol `_chkstk`

auf.

Sie können die von TwinCAT bereit gestellte Bibliothek „TcChkStk.lib“ verwenden, indem Sie die Eigenschaftendatei „TwinCAT RT ChkStk“ (standardmäßig in c:\TwinCAT\3.x\ sdk\VsProps) durch den Property-Manager zu den entsprechenden RT Varianten hinzufügen.



Die TwinCAT Ausführungsumgebung (XAR) hat eine Grenze von 64kB für die Stack-Größe.

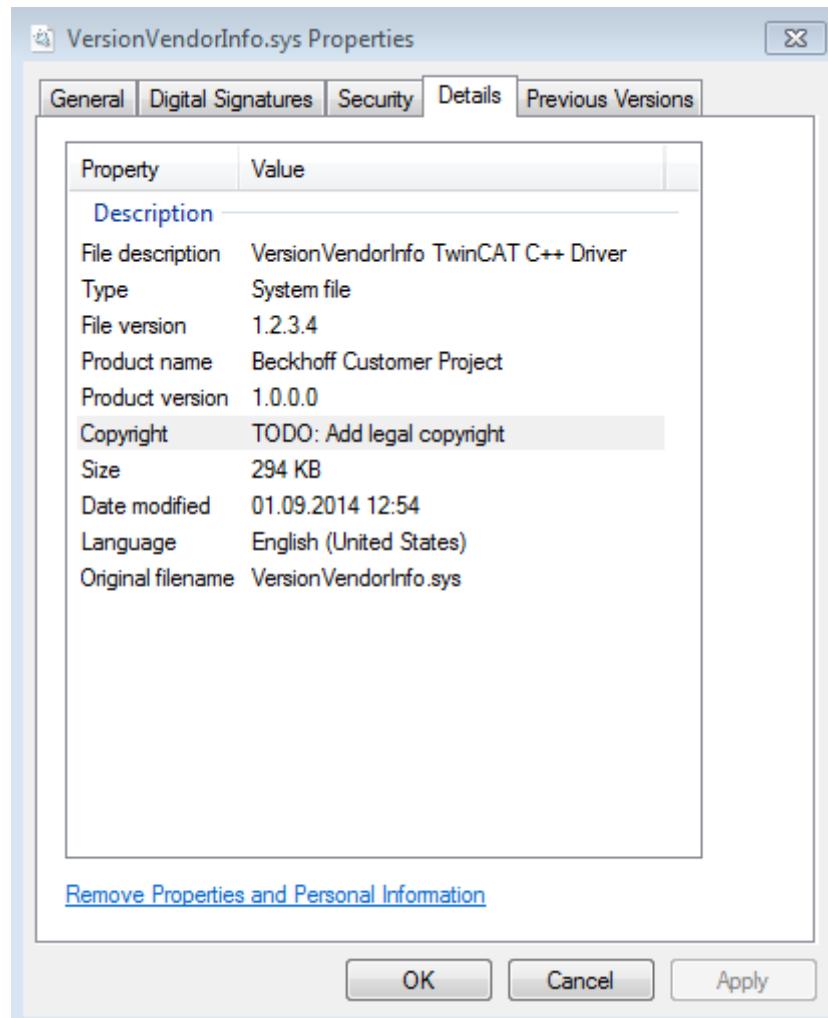
Sollte (lokaler) Speicher benötigt werden, existieren mehrere Möglichkeiten:

- Member Variablen (Header File des Moduls)
- Symbole des Moduls ([TMC Editor](#) [▶ 79]) z.B. in einer zusätzlichen, lokal genutzten DataArea
- Per new() und delete() dynamisch Speicher allozieren (vgl. [Speicherallozierung](#) [▶ 137])

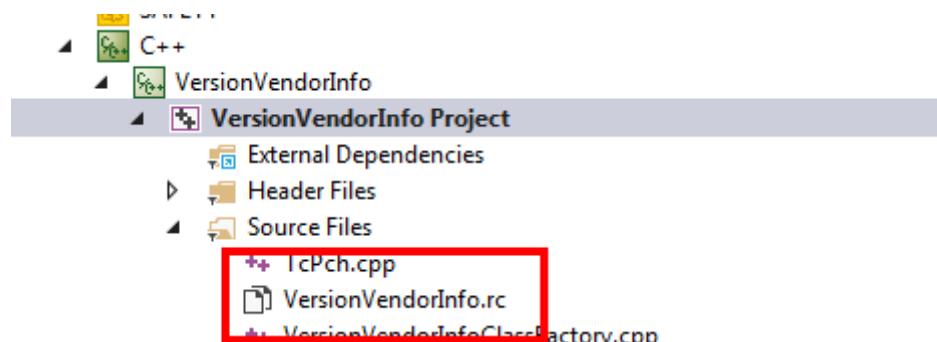
## 13.11 Setzen von Version/Herstellerinformationen

Windows bietet einen Mechanismus, um Hersteller- und Versionsressourcen abzufragen, die im Verlauf einer .rc Datei für die Kompilationszeit definiert sind.

Diese sind z.B. über den Karteireiter Details von jeder Eigenschaften-Datei zugänglich.



TwinCAT bietet dieses Verhalten über die bekannten Windows Mechanismen von .rc Dateien, die im Verlauf der TwinCAT C++ Projekterstellung erzeugt werden.



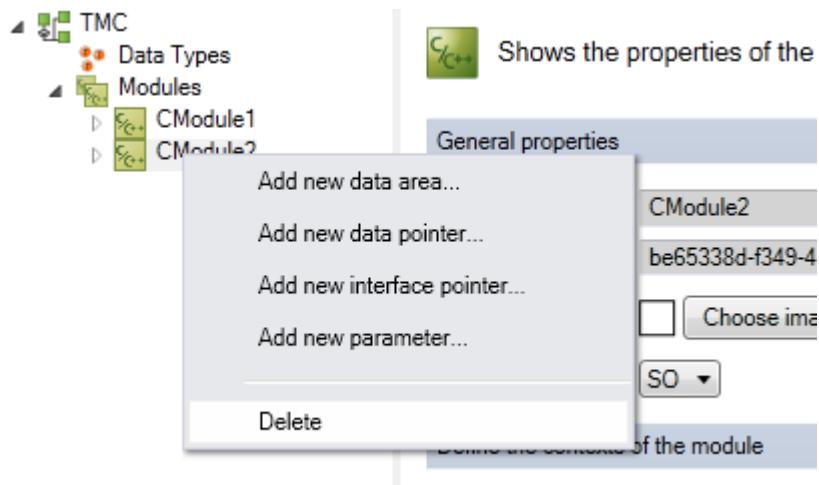
Bearbeiten Sie bitte die .rc-Datei im Source Files Ordner mit dem Resource Editor, um diese Eigenschaften festzulegen:

Key	Value
FILEVERSION	1, 2, 3, 4
PRODUCTVERSION	1, 0, 0, 0
FILEFLAGSMASK	0x17L
FILEFLAGS	0x0L
FILEOS	0x4L
FILETYPE	VFT_APP
FILESUBTYPE	VFT2_UNKNOWN
Block Header	English (United States) (040904b0)
CompanyName	Beckhoff Test Company
FileDescription	VersionVendorInfo TwinCAT C++ Driver
FileVersion	1.2.3.4
InternalName	VersionVendorInfo
LegalCopyright	TODO: Add legal copyright
OriginalFilename	VersionVendorInfo.sys
ProductName	Beckhoff Customer Project
ProductVersion	1.0.0.0

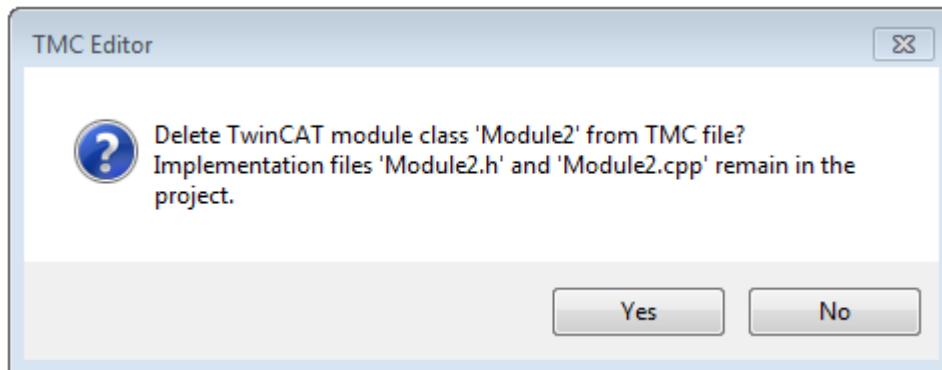
## 13.12 Modul löschen

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. „Delete“ auswählen



### 3. Löschung über TMC bestätigen



- Beachten Sie, dass die .cpp und .h Dateien erhalten bleiben – gegebenenfalls diese manuell löschen. Löschen anderer betreffender Komponenten (z.B. Header- Dateien, Strukturen). Siehe Compiler-Fehlermeldungen für weitere Auskünfte.

## 13.13 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">
```

## 13.14 SPS-Zeichenketten als Methodenparameter verwenden

Um eine Zeichenkette von SPS nach C++ als Methodenparameter zu übergeben, verwenden Sie bitte einen Zeiger mit Längeninformation beim Deklarieren der Methode in TMC:

Name	Type	Description
nStr	UDINT	Normal Type
pStr	SINT	Is Pointer

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
5
6 IF (ipStateMachine <> 0) THEN
7     SetString := ipStateMachine.SetString(SIZEOF(sSent),ADR(sSent));
8 END_IF

```

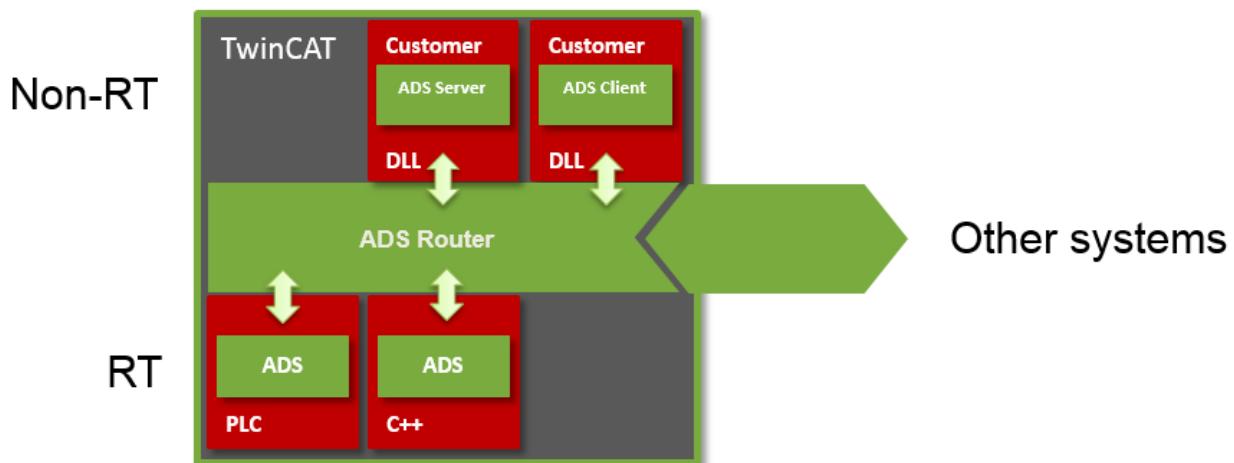
Der Grund ist der unterschiedliche Umgang mit Methodenparametern in beiden Welten:

- PLC: Verwendet den Werteauftrag (call by value) für STRING(nn) Datentypen
- TwinCAT C++ (TMC): Verwendet Verweisauftrag (call by reference)

## 13.15 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 PLC 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 \[► 220\]](#)

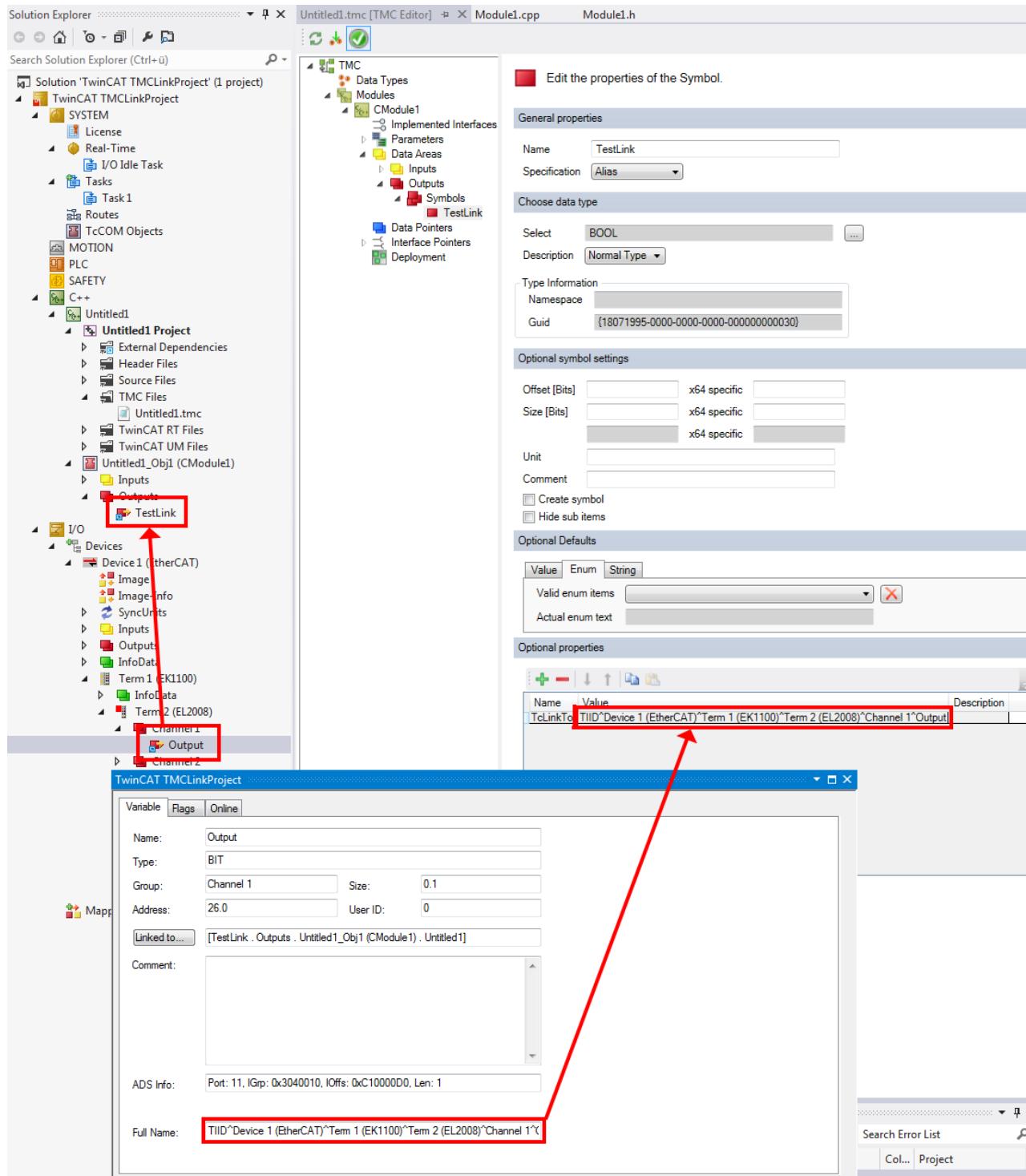
[Beispiel07: Empfang von ADS Notifications \[► 235\]](#)

[Beispiel08: Anbieten von ADS-RPC \[► 236\]](#)

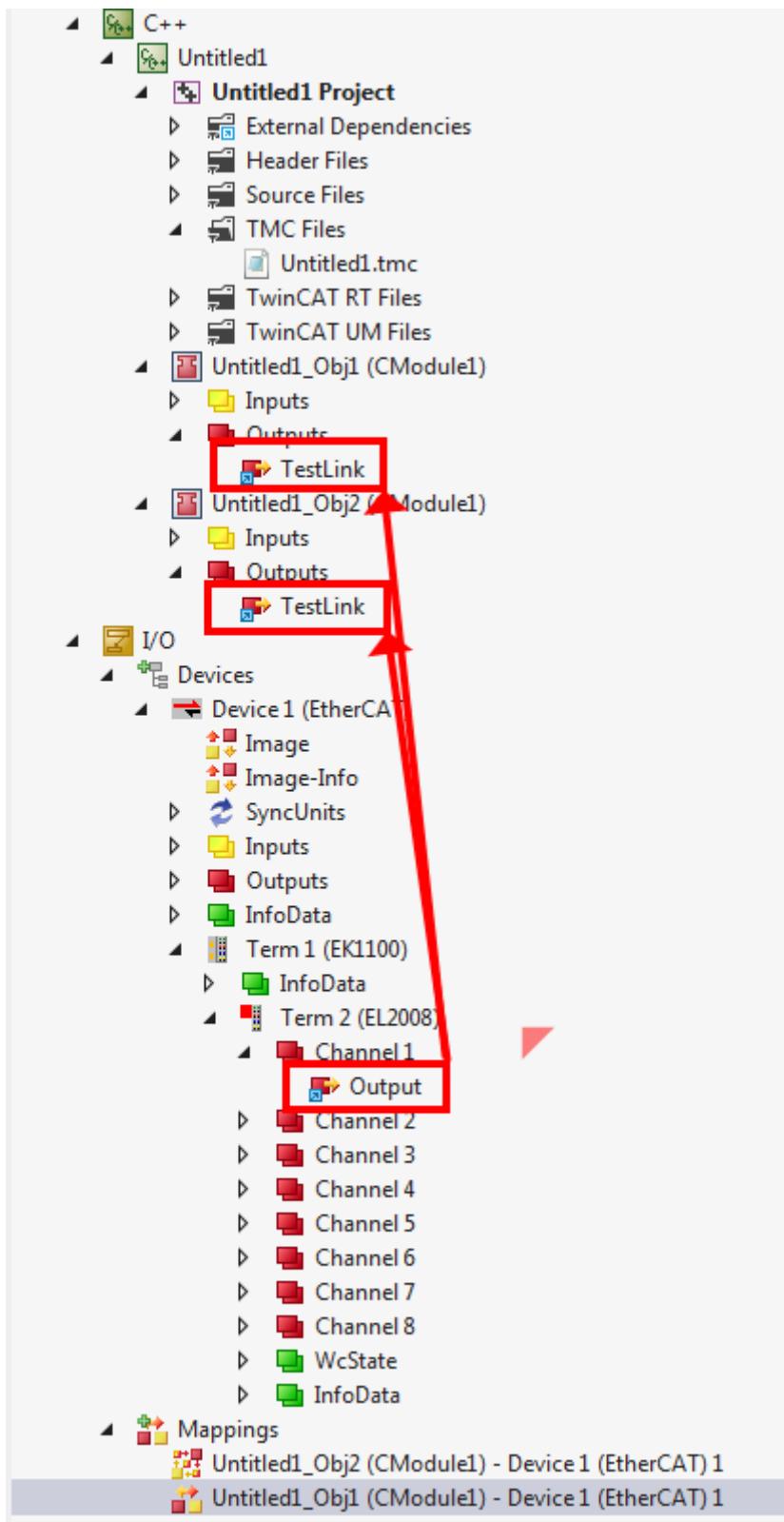
## 13.16 Verknüpfungen mittels TMC Editor (TcLinkTo)

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 Ziels angegeben. Im folgenden Screenshot wird dieses verdeutlicht:



Beachten Sie, dass eine solche Anweisung für alle Instanzen des Moduls gilt:

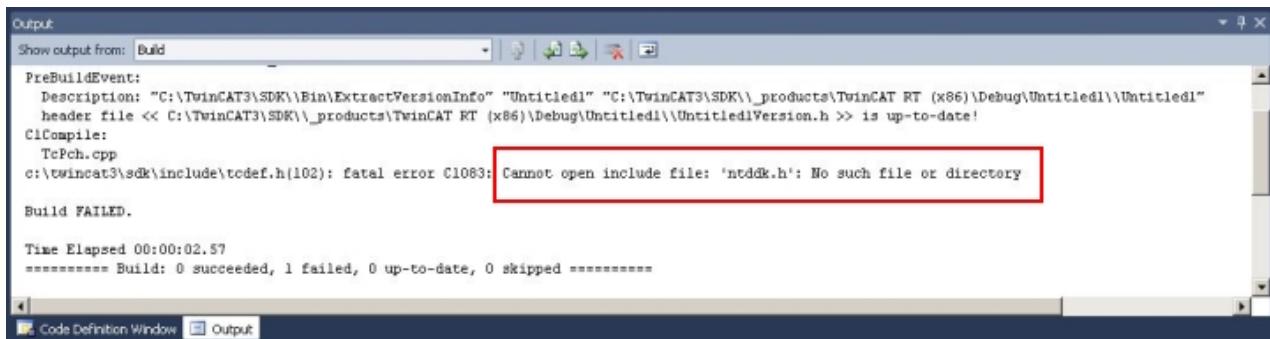


## 14 Fehlersuche

Liste von Fallstricken und Pannen beim Umgang mit TwinCAT C++ Modulen.

### 14.1 Build - „Cannot open include file ntddk.h“

Beim Erstellen eines TwinCAT C++ Projekts zeigt diese Fehlermeldung ein WinDDK-Problem auf Ihrem Engineering Computer.



```
Output
Show output from: Build
PreBuildEvent:
  Description: "C:\TwinCAT3\SDK\Bin\ExtractVersionInfo" "Untitled1" "C:\TwinCAT3\SDK\_products\TwinCAT RT (x86)\Debug\Untitled1\Untitled1"
  header file << C:\TwinCAT3\SDK\_products\TwinCAT RT (x86)\Debug\Untitled1\Untitled1Version.h >> is up-to-date!
C1Compile:
  TePch.cpp
c:\twincat3\sdk\include\tcdef.h(102): fatal error C1083: Cannot open include file: 'ntddk.h': No such file or directory
Build FAILED.

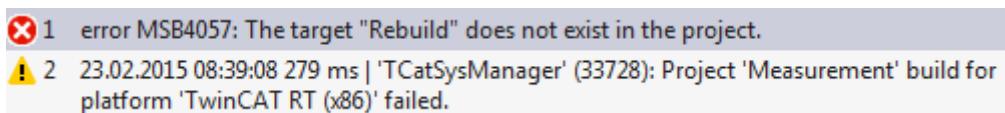
Time Elapsed 00:00:02.57
===== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =====
Code Definition Window Output
```

Bei Auftreten dieser Fehlermeldung prüfen Sie nach [WinDDK-Installationshandbuch \[► 19\]](#):

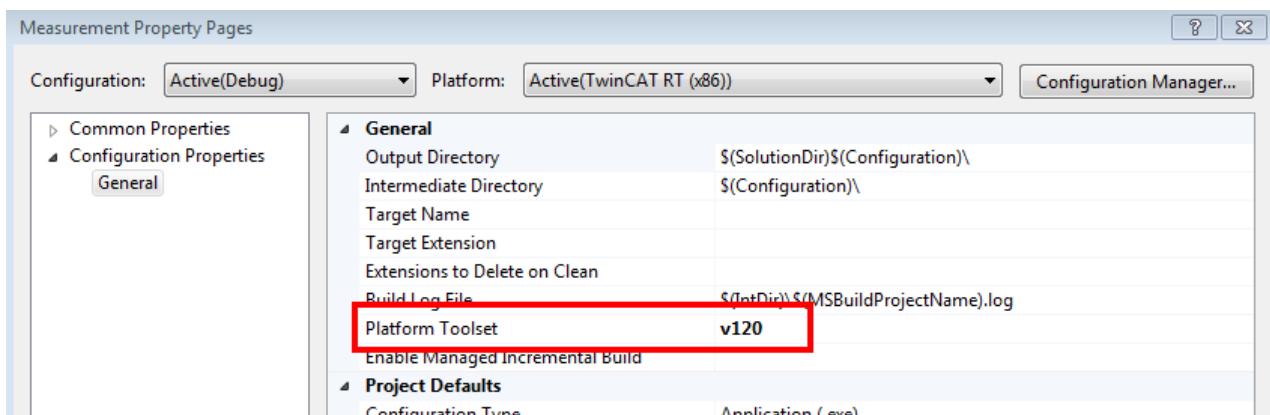
- Vergewissern, dass WinDDK installiert ist.
- Umgebungsvariable „WinDDK7“ und ihr konfigurierter Wert muss existieren, siehe Beschreibung in oben erwähntem Dokument. Der Wert muss dem Pfad entsprechen, wo WinDDK installiert ist, einschließlich des ersten Unterordners. Nach einer Änderung dieses Werts muss der Computer neu gestartet werden.

### 14.2 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.

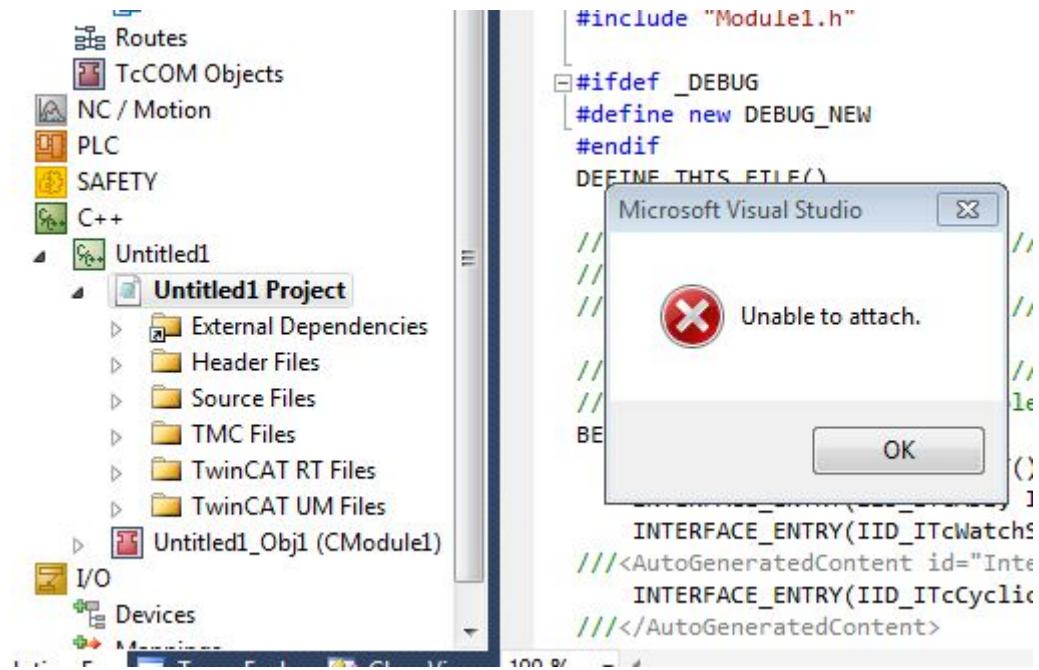


Bitte die Konfiguration von „platform toolset“ des C++ Projekts überprüfen. Muss gegebenenfalls neu konfiguriert werden, wenn Solutions von einer zu anderen Visual Studio Version migrieren:

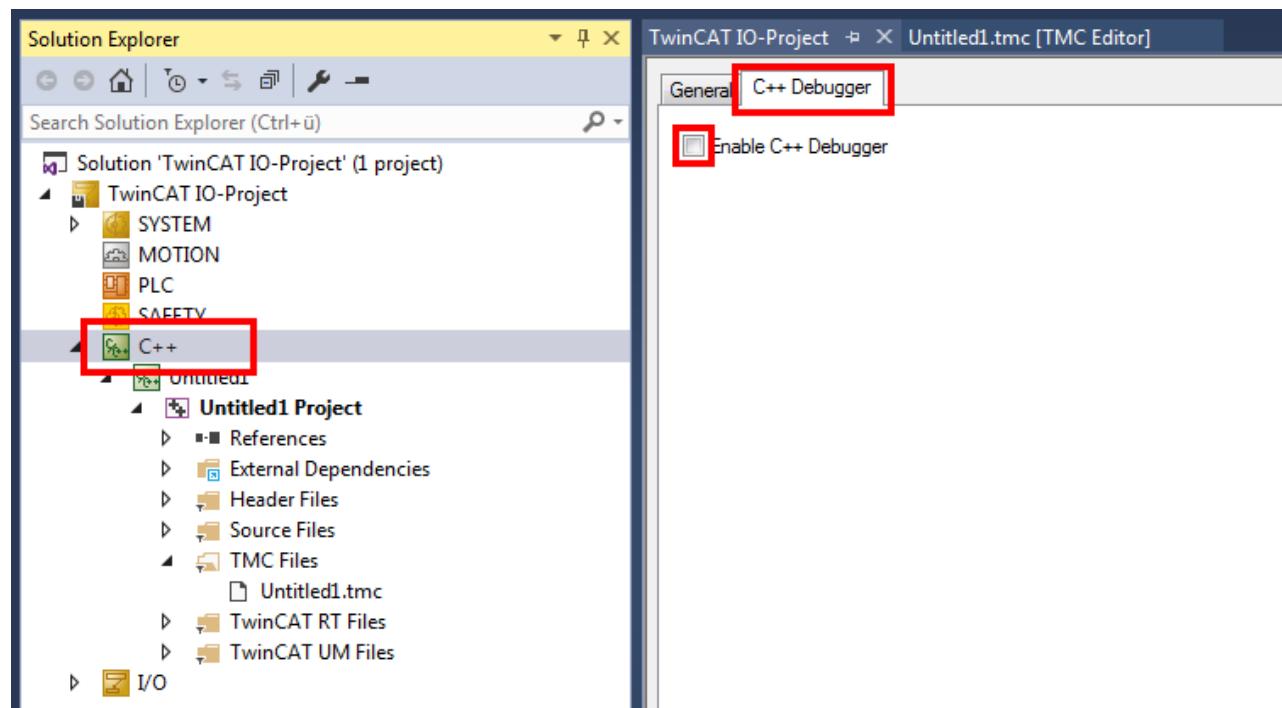


## 14.3 Debug - „Unable to attach“

Wenn diese Fehlermeldung beim Starten des Debuggers zum Debugging eines TwinCAT C++ Projekts erscheint, fehlt ein Konfigurationsschritt:

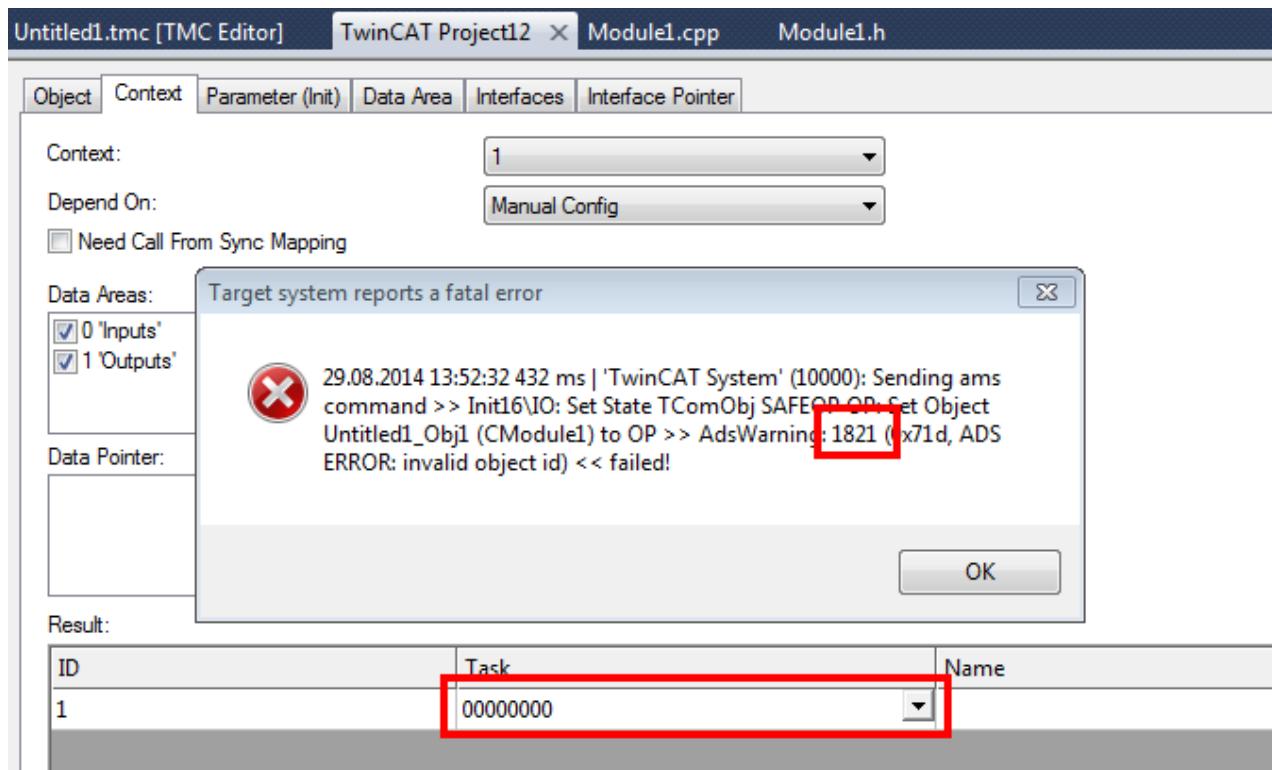


In diesem Fall zu „System -> Real-Time“ navigieren, Registerkarte „C++ Debugger“ wählen und Option „Enable C++ Debugger“ aktivieren.



## 14.4 Activation – „invalid object id“ (1821/0x71d)

Wenn im Verlauf des Starts der ADS Return Code 1821 / 0x71d berichtet wird, bitte den Kontext der Modulinstanz wie in [Schnellstart \[► 58\]](#) beschrieben überprüfen.



## 14.5 Fehlermeldung – VS2010 und LNK1123/COFF

Während der Kompilation eines TwinCAT C++ Moduls, weist die Fehlermeldung

LINK : fatal error LNK1123: failure during conversion to COFF: file invalid or corrupt

darauf hin, dass ein Visual Studio 2010 verwendet wird, aber ohne Service Pack 1, der für TwinCAT C++ Module [erforderlich \[► 18\]](#) ist.

Bitte das [Installationsprogramm](#) für den Service Pack von Microsoft herunterladen.

## 14.6 Verwendung von C++ Klassen in TwinCAT C++ Modulen

Beim Hinzufügen von (nicht TwinCAT) C++ Klassen bei der Verwendung vom Visual Studio-Kontextmenü Add->Class..., der Compiler / Linker meldet:

```
Error 4 error C1010: unexpected end of file while looking for precompiled
header. Did you forget to add '#include <>' to your source?
```

Fügen Sie bitte die folgenden Zeilen am Anfang Ihrer erzeugten Klassendatei ein:

```
#include "TcpCh.h"
#pragma hdrstop
```

## 14.7 Verwendung von afxres.h

In einigen Templates wird die afxres.h includiert, die auf einigen Systemen nicht bereitgestellt wird.

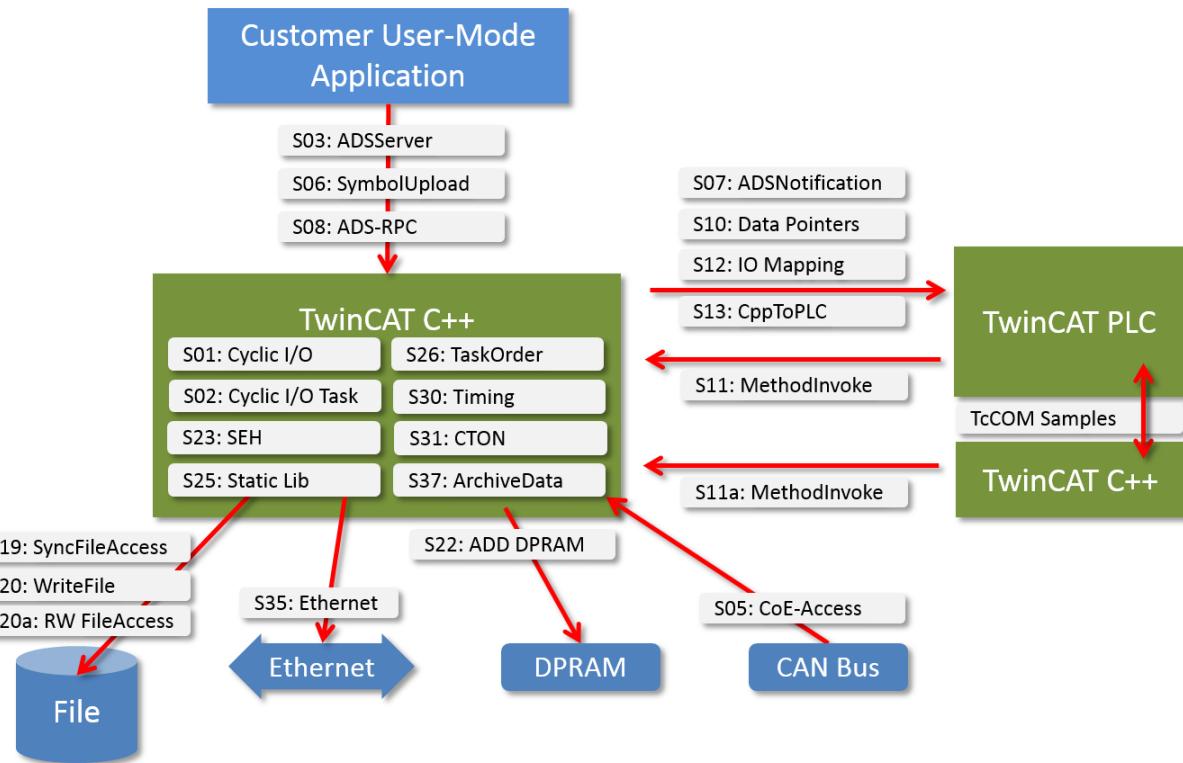
Diese Header Datei kann durch winres.h ersetzt werden.

# 15 C++-Beispiele

## 15.1 Übersicht

Es stehen zahlreiche Beispiele zur Verfügung - weitere Beispiele werden folgen.

Dieses Bild zeigt einen Überblick in grafischer Form und legt dabei den Schwerpunkt auf die Interaktionsmöglichkeiten eines C++ Moduls.



Darüber hinaus ist dies eine Tabelle mit kurzen Beschreibungen der Beispiele.

Nummer	Titel	Beschreibung
01	<a href="#">Beispiel01: Zyklisch mit IO-Modul [► 219]</a>	Dieser Artikel beschreibt die Implementierung eines TC3 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 Modulinstantz zugewiesen werden.
02	<a href="#">Beispiel02: Zyklisch mit IO Task [► 219]</a>	Dieses Beispiel beschreibt die Flexibilität von C++ Code bei der Arbeit mit IOs, welche an dem Task konfiguriert sind. Dank dieser Herangehensweise kann ein abschließend kompiliertes C++ Modul weit flexibler auf verschiedene, mit dem 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	<a href="#">Beispiel03: ADS Server Client [► 220]</a>	Beschreibt den Entwurf und die Implementierung einer eigenen ADS-Schnittstelle in einem C++ Modul. Das Beispiel enthält zwei Teile: - in TC3-C++ implementierter ADS Server mit benutzerspezifischer ADS-Schnittstelle - in C# implementierte ADS Client UI, die benutzerspezifische ADS-Meldungen an den ADS Server sendet
05	<a href="#">Beispiel05: CoE Zugriff über ADS [► 229]</a>	Zeigt wie über ADS auf CoE Register von EtherCAT-Geräten zugegriffen werden kann
06	<a href="#">Beispiel06: ADS C#-Client lädt ADS-Symbole hoch [► 230]</a>	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	<a href="#">Beispiel07: Empfang von ADS Notifications [► 235]</a>	Beschreibt die Implementierung eines TC3 C++ Moduls, das ADS Notifications bezüglich Datenänderungen auf anderen Modulen empfängt.
08	<a href="#">Beispiel08: Anbieten von ADS-RPC [► 236]</a>	Beschreibt die Implementierung von Methoden, welche per ADS über den Task aufrufbar sind.
10	<a href="#">Beispiel10: Modulkommunikation: Verwendung von Datenzeigern [► 239]</a>	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	<a href="#">Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf [► 240]</a>	Dieses Beispiel beinhaltet zwei Teile <ul style="list-style-type: none"> <li>• 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.</li> <li>• Zweites SPS-Modul um mit erstem Modul zu interagieren, indem Methoden vom C++ Modul aufgerufen werden</li> </ul>

11a	<a href="#">Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an [▶ 267]</a>	Dieses Beispiel beinhaltet zwei Klassen in einem Treiber (kann auch zwischen zwei Treibern gemacht werden) <ul style="list-style-type: none"> <li>Ein Modul, das eine Berechnungsmethode bereitstellt. Der Zugriff ist durch eine CriticalSection geschützt.</li> <li>zweites Modul, das als Aufrufer agiert, um die Methode im anderen Modul zu verwenden</li> </ul>
12	<a href="#">Beispiel12: Modulkommunikation: Verwendet IO Mapping [▶ 268]</a>	• 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	<a href="#">Beispiel13: Modulkommunikation: C-Modul ruft SPS-Methoden auf [▶ 269]</a>	• Beschreibt wie ein C++ Modul wie ein TwinCAT C++ Modul per TcCOM Interface Methoden eines Funktionsbausteins der SPS aufruft.
19	<a href="#">Beispiel19: Synchroner Dateizugriff [▶ 272]</a>	Beschreibt, wie die File-IO-Funktionalität bei 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	<a href="#">Beispiel20: FileIO-Write [▶ 273]</a>	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	<a href="#">Beispiel20a: FileIO-Cyclic Read / Write [▶ 273]</a>	Ist ein umfangreicheres Beispiel als S20 und S19. Es beschreibt zyklischen Lese- und/oder Schreibzugriff auf Dateien von einem TC3-C++ Modul aus.
22	<a href="#">Beispiel22: Automation Device Driver (ADD): Zugang DPRAM [▶ 275]</a>	Beschreibt, wie der TwinCAT Automation Device Driver (ADD) für den Zugriff auf die DPRAM zu schreiben ist.
25	<a href="#">Beispiel25: Statische Bibliothek [▶ 278]</a>	Beschreibt, wie die in einem anderen TC3 C++ Modul enthaltene TC3 C++ statische Bibliothek verwendet werden kann.
26	<a href="#">Beispiel26: Ausführungsreihenfolge in einem Task [▶ 279]</a>	Dieser Artikel beschreibt die Bestimmung der Taskausführungsreihenfolge, wenn einem Task mehr als ein Modul zugeordnet ist.
30	<a href="#">Beispiel30: Zeitmessung [▶ 281]</a>	Beschreibt die Messung der TC3 C++ Zyklus- oder Ausführungszeit.
31	<a href="#">Beispiel31: Funktionsbaustein TON in TwinCAT3 C++ [▶ 282]</a>	Dieser Artikel beschreibt die Implementierung eines Verhaltens in C++, das vergleichbar mit einem TON Funktionsbaustein von SPS / 61131 ist.
37	<a href="#">Beispiel37: Daten archivieren [▶ 284]</a>	Beschreibt das Laden und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung.
TcCOM	<a href="#">TcCOM Beispiele [▶ 285]</a>	Mehrere Beispiele die die Modulkommunikation zwischen PLC und C++ verdeutlichen.

## 15.2 Beispiel01: Zyklisches Modul mit IO

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, dessen Modul-IO mit einem physikalischen IO gemappt ist.

### Download

Hier erhalten Sie den [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### 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 [hier \[► 49\]](#) in der Kurzanleitung Schritt für Schritt beschrieben.

## 15.3 Beispiel02: Zyklische C++ Logik, die IO vom IO Task verwendet

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, das ein Image eines IO Tasks verwendet.

### Download

Hier erhalten Sie den [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

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 dem 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 „TcloTaskImageAccessDrv“ mit einer Instanz des Moduls „TcloTaskImageAccessDrv\_Obj1“
- Ein Task „Task1“ mit einem Image, 10 Eingangsvariablen (Var1..Var10) und 10 Ausgangsvariablen (Var11..Var20).

- Sie sind verbunden: Die Instanz wird von dem 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 Helpermethode „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-Abbildung des Systems, um eine Übersicht zu erhalten.

Die Eingangsvariablen können geöffnet und dann mit dem „Online“ Karteireiter gesetzt werden.

## 15.4 Beispiel03: C++ als ADS Server

Dieser Artikel beschreibt:

- Erstellung eines TC3 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 \[▶ 221\]](#)
- Einen C# ADS Client erstellen, um mit 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# \[▶ 225\]](#)

### 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, hartkodiert werden.

Nachteil hier: Das C++ Modul kann nicht mehr als einmal instanziert werden, da das Teilen eines ADS Ports nicht möglich ist.

```

HRESULT IAdsCommunicationModule::SetUpInstanceState(PIOMINITDATA pInitData)
{
    m_Trace.Log(tlVerbose, FENTERA);
    HRESULT hr = S_OK;
    IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA(pInitData);

    hr = SUCCEEDED(hr) ? InitAmsPort(m_spSrv, 0x3039) : hr;

    // cleanup on failure
    if (FAILED(hr))
    {
        ShutdownAmsPort();
    }
    m_Trace.Log(tlVerbose, FLEAVE);
    return hr;
}

// State transition from SAFEOP to OP
// ...

```

## 15.4.1 Beispiel03: Der in C++ geschriebene TC3 ADS Server

In diesem Artikel wird die Erstellung eines als ADS Server agierenden TC3 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 Beispiel](#):

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### 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](#) [▶ 49]

#### 2. Erstellen Sie ein C++ Projekt mit ADS Port

Folgen Sie den Schritten für die [Erstellung eines neuen TwinCAT 3 C++ Projekts](#) [▶ 50].

Wählen Sie im „Class templates“-Dialog „TwinCAT Module Class with ADS port“.

#### 3. Fügen Sie dem Projekt die Beispiellogik hinzu

6. 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;
};

7. 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
    }
}

```

### 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 - fügen Sie die blauen Codezeilen ein, 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
};

```

Fügen Sie die blauen Codezeilen in Ihre <MyClass>::AdsReadWriteInd() Methode ein (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;
}
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

8. Den [TwinCAT TMC Code Generator \[▶ 55\]](#) ausführen, um die Ein-/Ausgänge für das Modul bereitzustellen.
9. Speichern Sie das Projekt
10. Das Projekt [komplizieren \[▶ 55\]](#)
11. [Eine Modulinstanz erstellen \[▶ 56\]](#)
12. Einen [zyklischen Task erstellen \[▶ 58\]](#) und das C++ Modul für die Ausführung in diesem Kontext konfigurieren.
13. Sie können optional die Hardware IO scannen und das Symbol „Value“ von Ausgängen bestimmten Ausgangsklemmen zuordnen.
14. Das TwinCAT Projekt [aktivieren \[▶ 61\]](#)

#### 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:

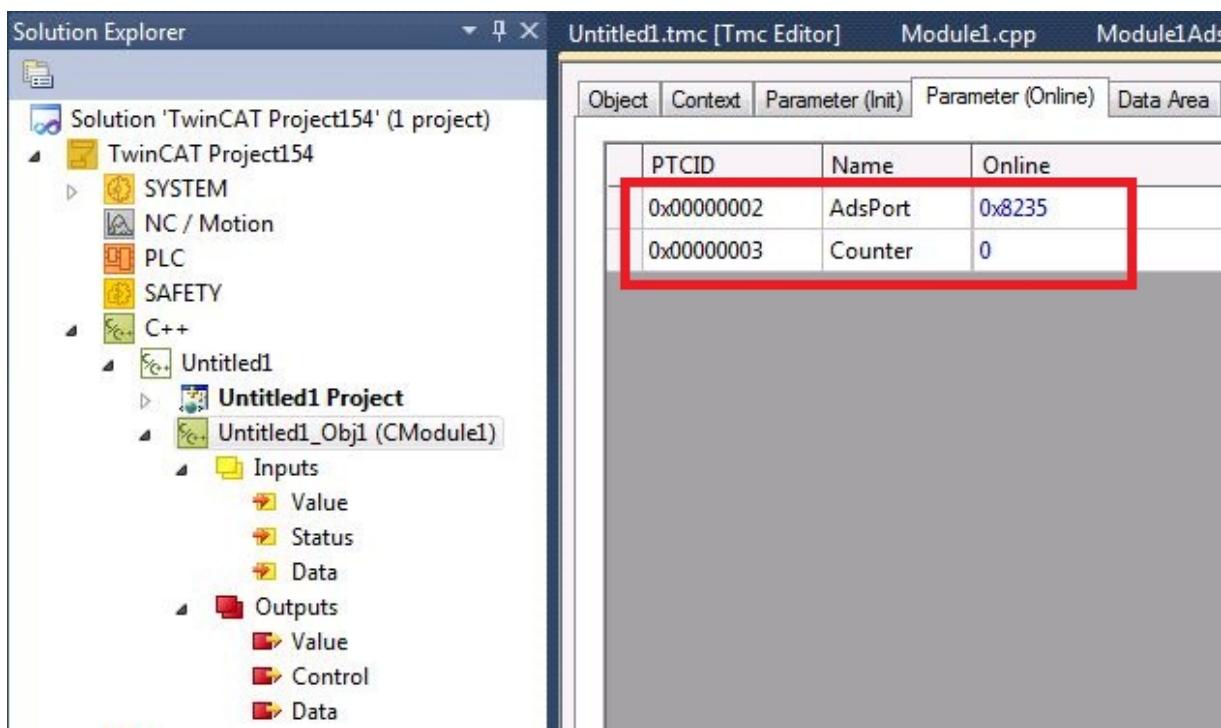
15. Zur Modulinstanz navigieren, Karteireiter „Parameter Online“ auswählen.

**ADSPort**

Es ist 0x8235 oder Dezimal 33333 zugewiesen (das kann in Ihrem Beispiel anders sein). Werden mehr und mehr Instanzen erstellt, erhält jede Instanz ihren eigenen eindeutigen AdsPort.

**Zähler**

Immer noch auf „0“, da wir die ADS-Meldung zum Starten der Inkrementierung nicht gesendet haben.



⇒ Der Server-Teil ist abgeschlossen - fahren wir also fort mit [ADS Client sendet die ADS-Meldungen \[► 225\]](#).

## 15.4.2 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# / PLC / ...), noch von der TwinCAT Version (TwinCAT 2 oder TwinCAT 3) ab.

### Download

**Hier erhalten Sie den [Quellcode für dieses Beispiel](#).**

- ✓ Dieser Code erfordert .NET Framework 3.5 oder höher!
- 1. Die heruntergeladene ZIP-Datei entpacken
- 2. Die enthaltene sln-Datei mit Visual Studio öffnen
- 3. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das Projekt rechtsklicken und „Build“ klicken)
- 4. Programm starten, d.h. 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

Die beiden ADS-Parameter eingeben, um Ihren ADS-Kommunikationspartner zu bestimmen

- NetID:  
127.0.0.1.1 (für ADS-Partner auch mit lokalem ADS Message Router verbunden)

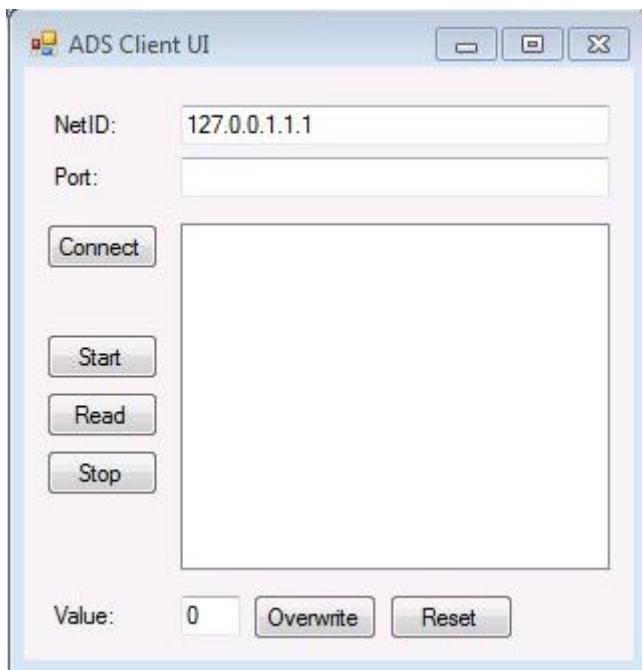
Geben Sie eine andere NetID ein, wenn Sie über das Netzwerk mit einem an einen anderen ADS Router angeschlossenen ADS-Partner kommunizieren möchten.

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 \[► 221\]](#) in diesem Beispiel heraus, der AdsPort war 0x8235 (dez 33333).

#### Verbindung mit Kommunikationspartner herstellen

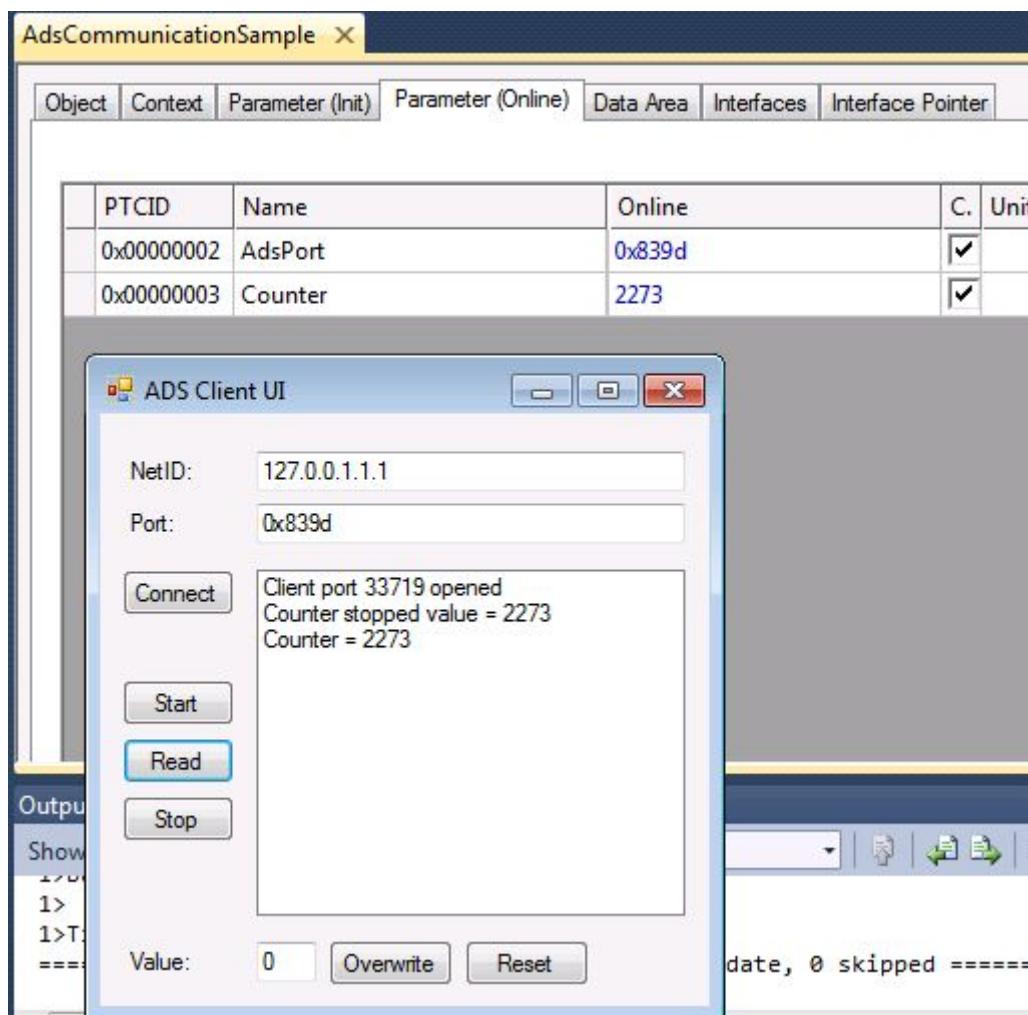
Beim Klicken auf „Connect“ wird die Methode TcAdsClient.Connect zwecks Herstellung einer Verbindung mit dem konfigurierten Port aufgerufen.



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 \[▶ 221\]](#).

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., bitte 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);
    }
}
}
```

## 15.5 Beispiel05: C++ CoE Zugriff über ADS

In diesem Artikel wird die Implementierung eines TC3 C++ Moduls beschrieben, das auf das CoE (CANopen over EtherCAT) Register einer EtherCAT Klemme zugreifen kann.

## Download

**Hier erhalten Sie den Quellcode für dieses Beispiel.**

1. Die heruntergeladene ZIP-Datei entpacken
  2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
  3. Ihr Zielsystem auswählen
  4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
  5. Die Konfiguration aktivieren

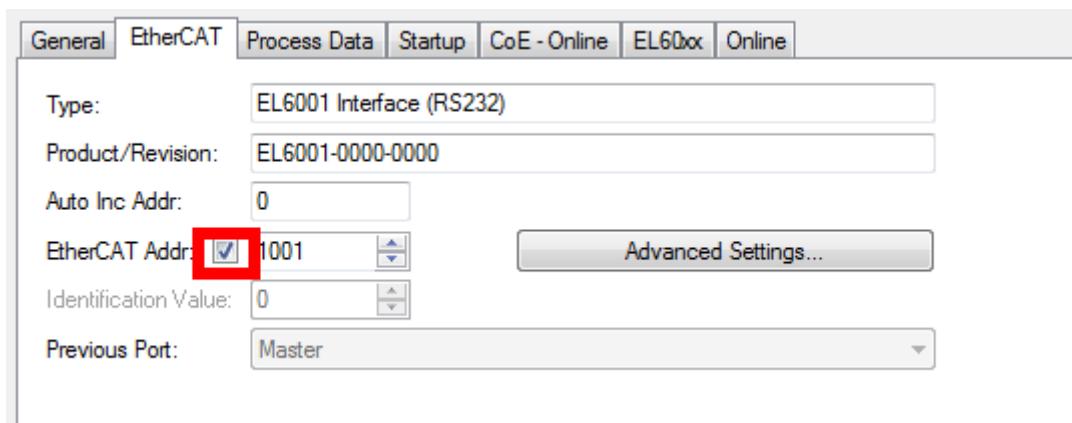
## Beschreibung

Dieses Beispiel beschreibt den Zugriff auf eine EtherCAT-Klemme, der die Hersteller-ID liest und die Baudrate für die serielle Kommunikation festlegt.

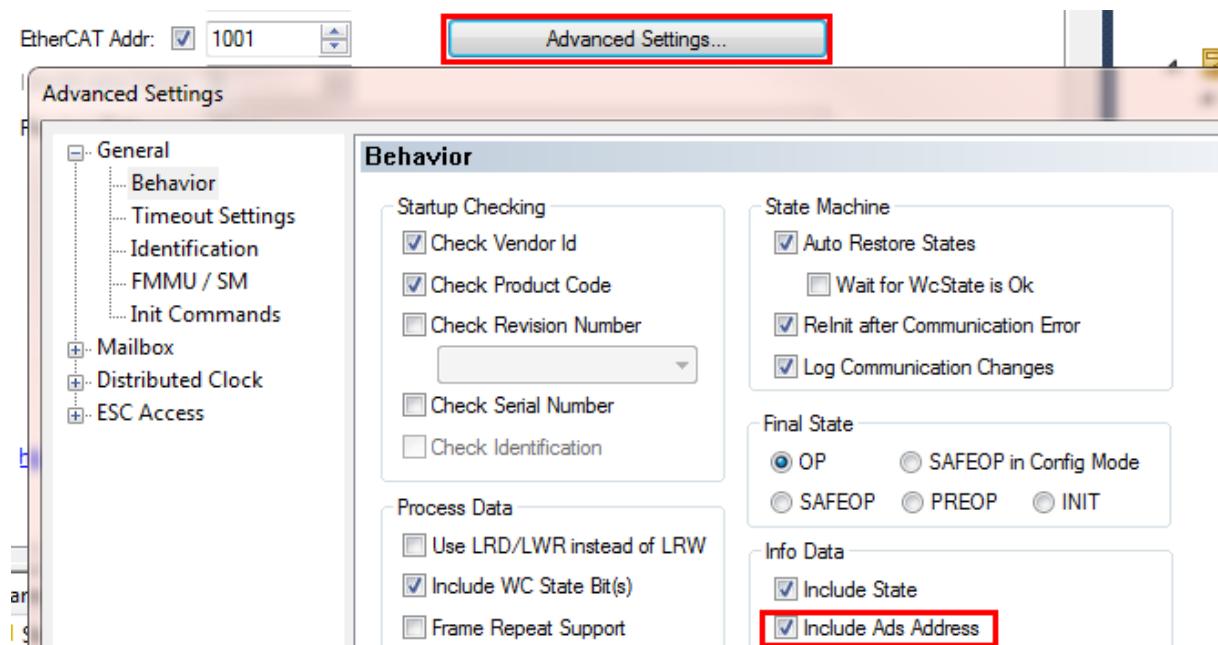
Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++ Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang „Value“ im Datenbereich zuweist.

## Konfiguration

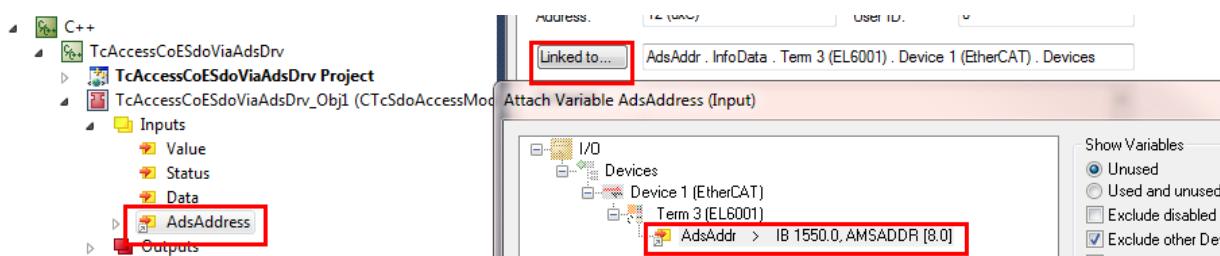
6. Die EtherCAT-Adresse der betreffenden Klemme aktivieren und zuweisen.



7. Den Einschluss der ADS-Adresse in den erweiterten Einstellungen der EtherCAT-Klemme aktivieren:



8. Die ADS-Adresse (einschließlich netId und port) dem Moduleingang AdsAddress zuweisen:



9. Der Modulparameter wird im Verlauf der Initialisierung festgelegt:

	Name	Value	Online
	DefaultAdsPort	0xfffff	0xfffff
	ContextAdsPort	0x015e	0x015e
	BaudRate	0x0005	0x0005
	VendorId	0x00000000	0x00000002
	CoEReadIndex	0x1018	0x1018
	CoEReadSubIndex	0x0001	0x0001
	CoEWriteIndex	0x4073	0x4073
	CoEWriteSubIndex	0x0000	0x0000

## 15.6 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 abonnieren, um Werte „bei Veränderung“ als Callback zu erhalten.

## Download

**Erhalten Sie den** Quellcode für dieses Client-Beispiel:

- ✓ Dieser Code erfordert .NET Framework 3.5 oder höher!
- 1. Die heruntergeladene ZIP-Datei entpacken
- 2. Die enthaltene sln-Datei mit Visual Studio öffnen
- 3. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das Projekt rechtsklicken und „Build“ klicken)
- 4. Programm starten, d.h. Rechtsklick auf Projekt, Debug->Start new instance

Das Client-Beispiel sollte mit Beispiel 03 „C++ als ADS Server“ verwendet werden.

Bitte öffnen Sie Beispiel 03 [► 221] bevor Sie mit diesem Client-seitigen Beispiel beginnen!

## Beschreibung

Anhand von diesem Beispiel werden die Möglichkeiten von ADS beschrieben.

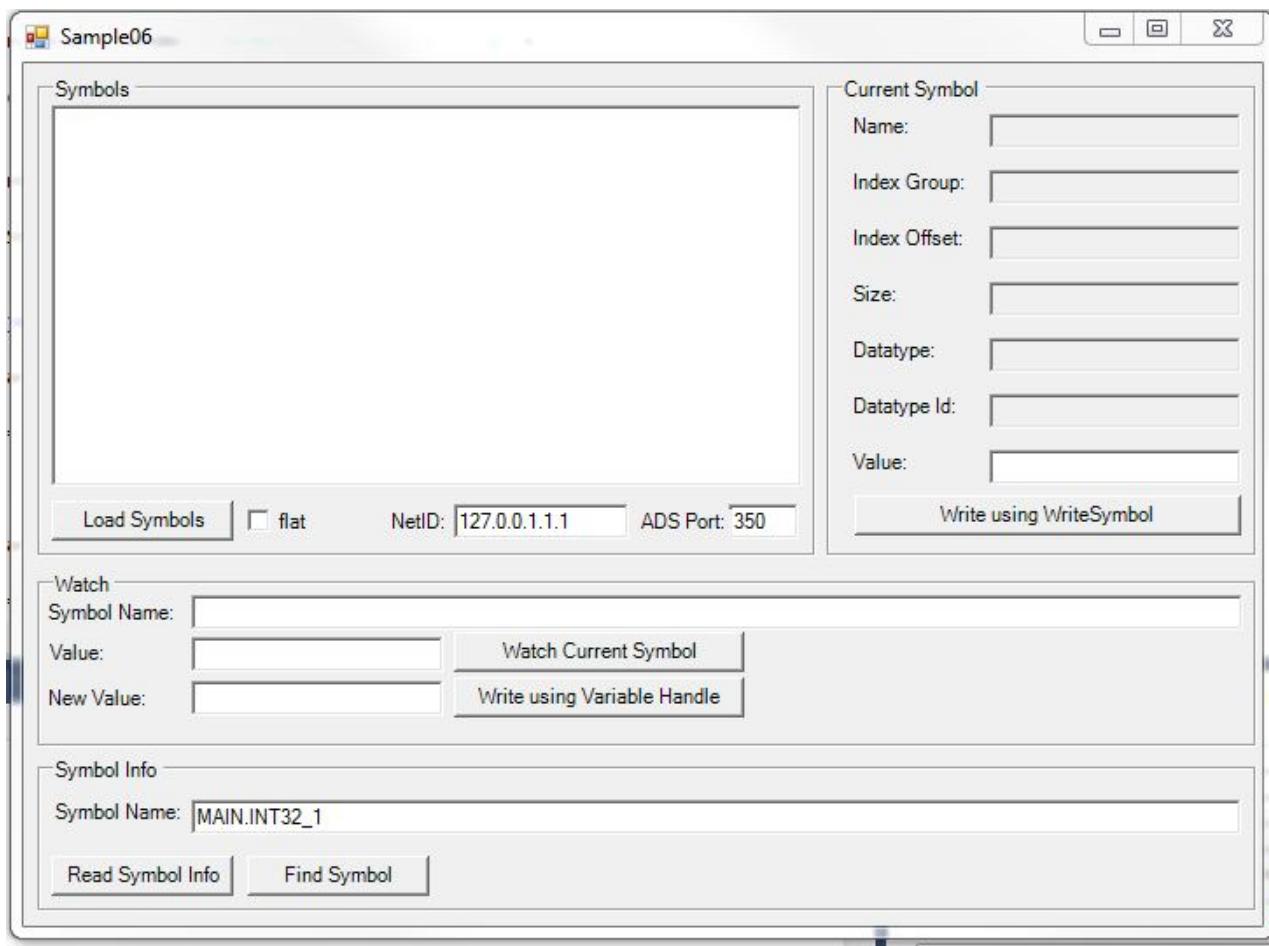
Die Einzelheiten der Implementierung werden in „Form1.cs“, das im Download enthalten ist, beschrieben. Die Verbindung über ADS mit dem Zielsystem wird in der „btnLoad\_Click“ Methode, die beim Klicken auf die „Load Symbols“ Schaltfläche aufgerufen wird, hergestellt. Von dort aus können Sie die verschiedenen 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, SPS-Modul oder 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 (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)



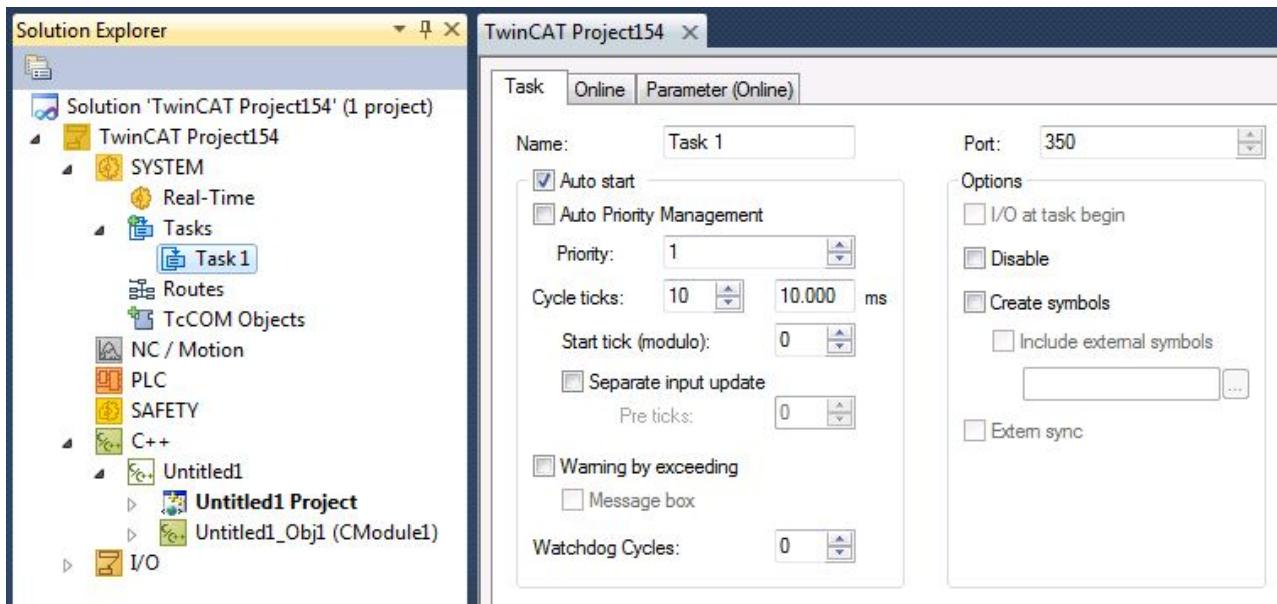
Hinweis

#### Nicht den ADS Server Port mit dem regulären ADS Port verwechseln.

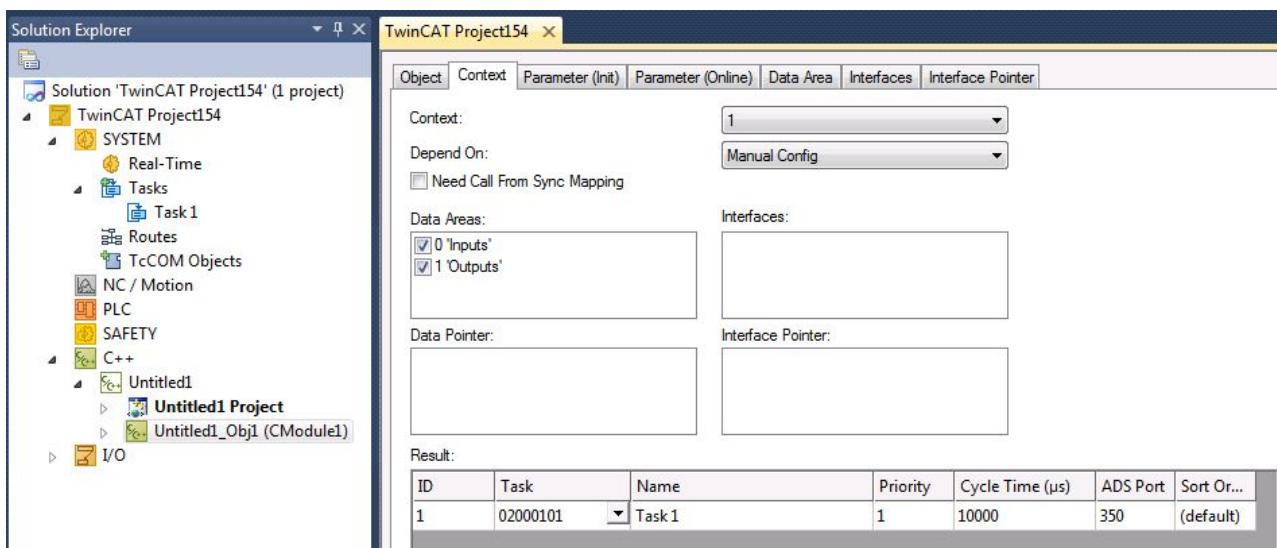
Verwechseln Sie nicht den ADS Server Port (der ausdrücklich in Beispiel 03 implementiert wurde, um Ihren eigenen Message-Handler bereitzustellen) mit dem regulären ADS Port zwecks Zugriff auf Symbole (es gibt nichts zu tun, wird automatisch zur Verfügung gestellt):

Wir benötigen den regulären ADS Port, um auf Symbole zugreifen zu können. Sie können den AdsPort beim IO Task Ihrer Instanz oder bei der Modulinstanz selber herausfinden (weil das Modul im Kontext des IO Tasks ausgeführt wird).

Navigieren Sie zum 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.



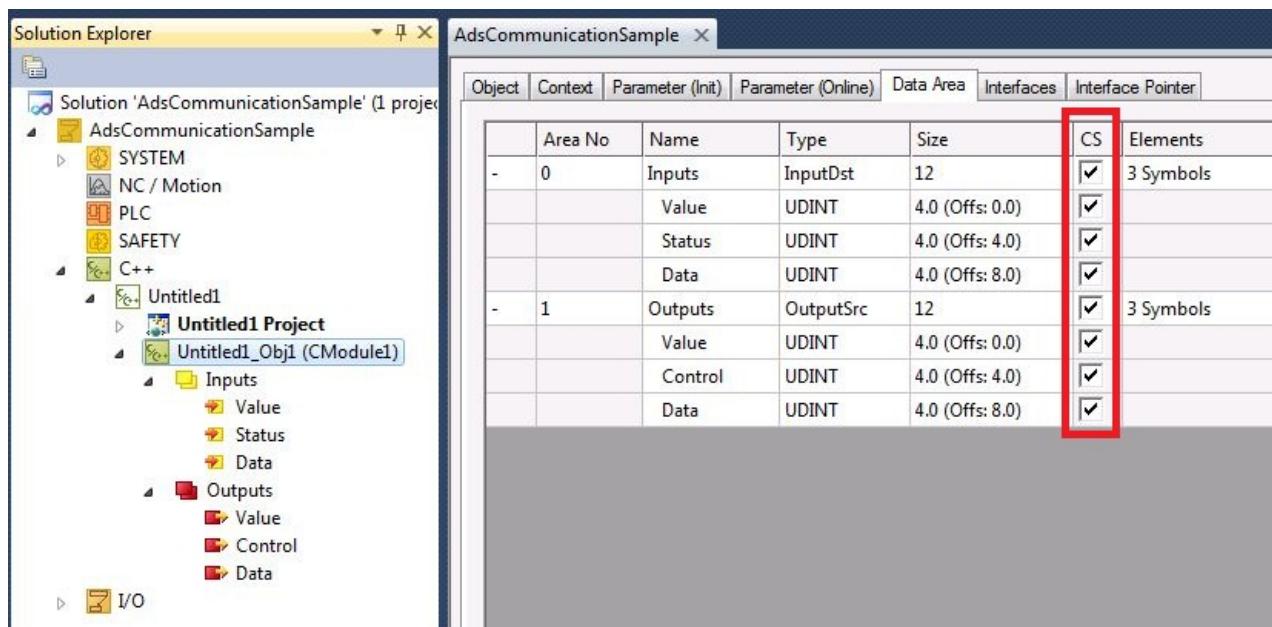
### Aktivierte Symbole für Zugriff über ADS verfügbar

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 bitte die „Konfiguration aktivieren“.



### Symbole laden

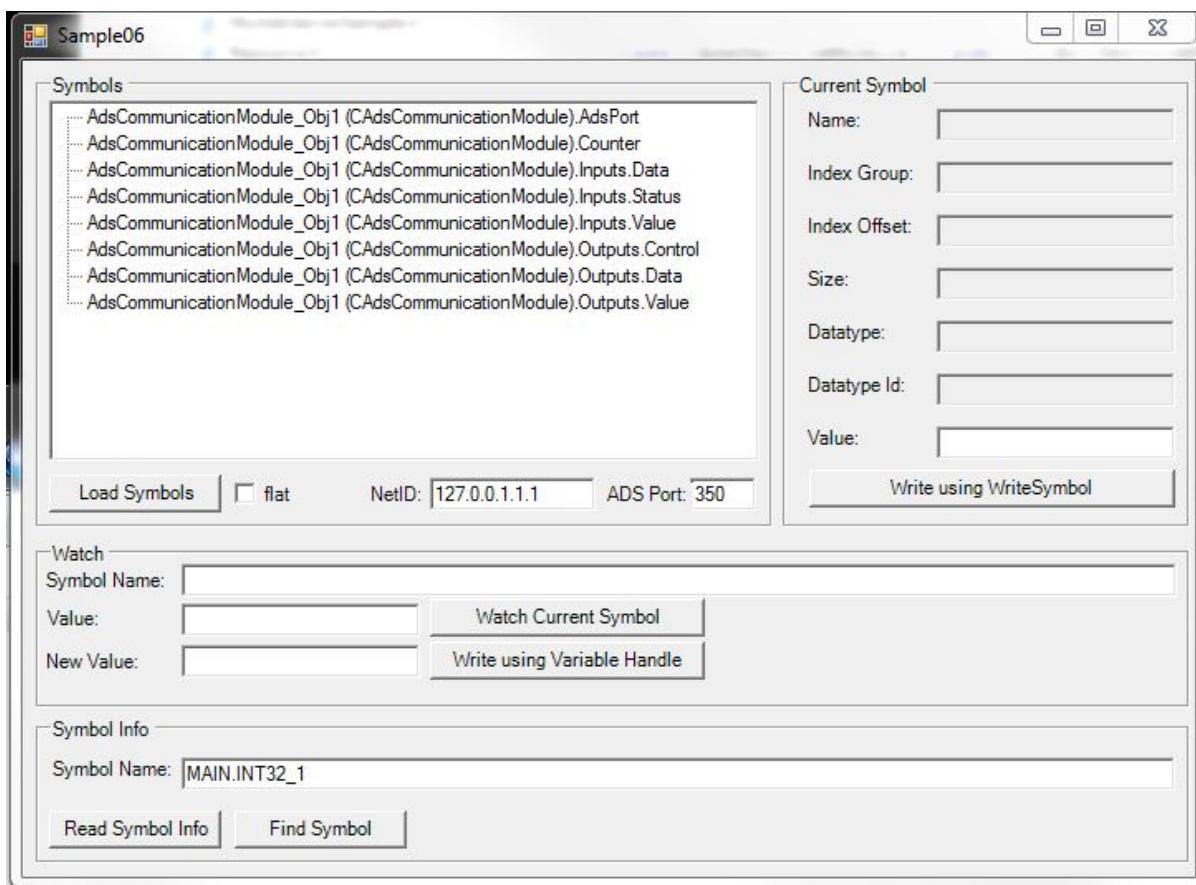
Nach der Einrichtung der NetID und des ADS Port auf die Schaltfläche „Load Symbols“ klicken, um eine Verbindung mit dem Zielsystem herzustellen und die Symbole zu laden.

Daraufhin sind alle verfügbaren Symbole zu sehen. Anschließend können Sie:

- Einen neuen Wert schreiben:  
Ein Symbol im linken Baum auswählen, z.B. „Counter“  
Einen neuen Wert in das Bearbeitungsfeld „Value“ auf der rechten Seite eingeben und auf „Write using WriteSymbol“ klicken.  
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.  
Ein Symbol im linken Baum auswählen, z.B. „Counter“  
Auf „Watch Current Symbol“ klicken



## 15.7 Beispiel07: Empfang von ADS Notifications

Dieser Artikel beschreibt die Implementierung eines TC3 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 C++ Modulen aus betrachtet werden.

### Download

[Hier erhalten Sie den Quellcode für dieses Beispiel:](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### 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 „CycleUpdate“ Methode 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.

 <b>Hinweis</b>	<b>Beginn der Modulentwicklung</b> 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 über schreiben.
---	---

#### Siehe auch

[ADS-Kommunikation \[► 175\]](#)

## 15.8 Beispiel08: Anbieten von ADS-RPC

Dieser Artikel beschreibt die Implementierung von Methoden, welche per ADS über den Task aufrufbar sind.

#### Download

**Hier erhalten Sie den** [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tzip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren
6. Kompilieren und Starten des Visual C++ Projektes zum Testen.

#### 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.

#### Startup

Das Beispiel benötigt nach dem Entpacken keine Konfiguration.

#### Das Beispiel verstehen

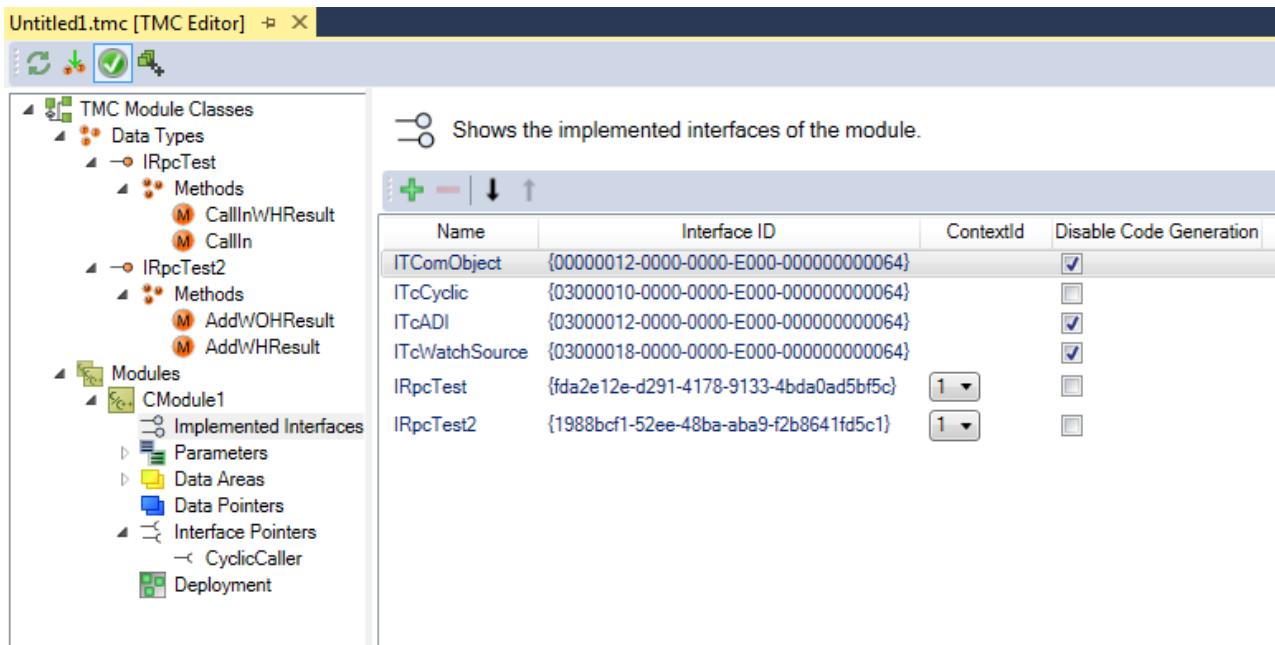
Das Beispiel besteht aus dem TwinCAT C++ Modul, welches die RPC Methoden anbietet und einem C++ Beispielprogramm, welches diese aufruft.

## 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 \[► 79\]](#) sind die Optionen genauer beschrieben.

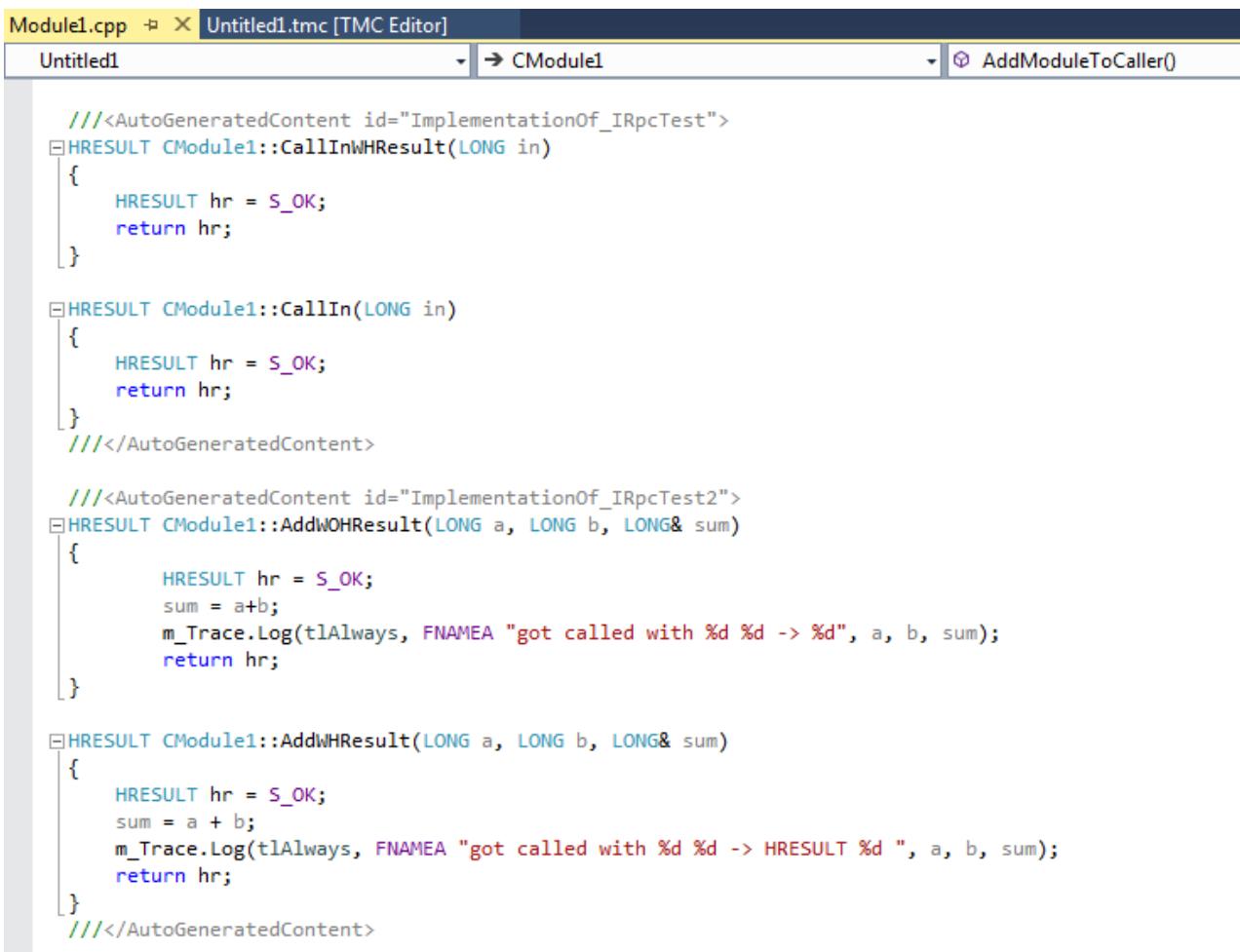
Im hier vorliegenden Modul werden zwei Interfaces beschrieben und implementiert, wie im TMC Editor gesehen werden kann:



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::CallInWHRResult(LONG in)
{
    HRESULT hr = S_OK;
    return hr;
}

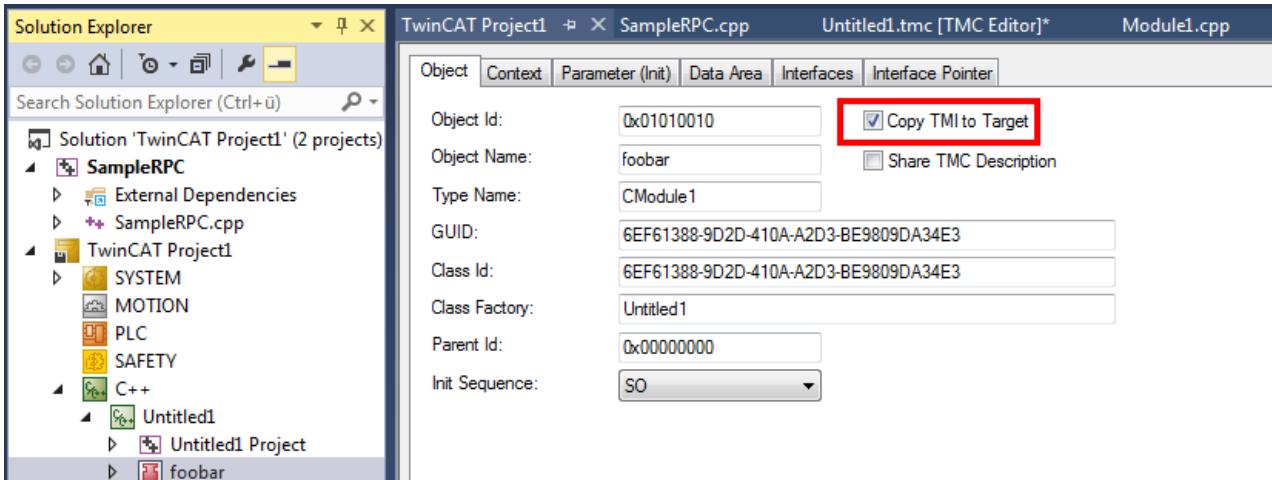
HRESULT CModule1::CallIn(LONG in)
{
    HRESULT hr = S_OK;
    return hr;
}
///</AutoGeneratedContent>

//<AutoGeneratedContent id="ImplementationOf_IRpcTest2">
HRESULT CModule1::AddWHRResult(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:

```
OK: AdsSyncReadWriteReq <getHdl foobar.IRpcTest#CallIn>
OK: AdsSyncReadWriteReq <getHdl foobar.IRpcTest#CallInWHResult>
OK: AdsSyncReadWriteReq <getHdl foobar.IRpcTest2#AddWOHResult>
OK: AdsSyncReadWriteReq <getHdl foobar.IRpcTest2#AddWHResult>

Press key to call all methods

Calling foobar.IRpcTest#CallIn
Send: 0

Calling foobar.IRpcTest#CallInWHResult
Value given: 1
ReturnCode: 0

Calling foobar.IRpcTest2#AddWOHResult
Value given A: 1
Value given B: 2
Value got <(A+B)>: 3

Calling foobar.IRpcTest2#AddWHResult
Value given A: 1
Value given B: 2
ReturnCode: 0
Value got <(A+B)>: 3
```

#### Sehen Sie dazu auch

- TwinCAT Module Instance Configurator [▶ 123]
- TwinCAT Module Klassenassistent [▶ 78]
- TwinCAT Module Instance Configurator [▶ 123]

## 15.9 Beispiel10: Modulkommunikation: Verwendung von Datenzeigern

Dieser Artikel beschreibt die Implementierung von zwei TC3 C++ Modulen, die über einen Datenzeiger kommunizieren.

#### Download

Hier erhalten Sie den [Quellcode für dieses Beispiel](#):

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tzip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

#### Beschreibung

Diese Kommunikation baut auf einen „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 dem selben 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 Synchronisationsmechanismus 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 Ausgangsbereich zur Verfügung und implementiert die Schnittstelle ITclioCyclic. 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 SAFEOEP 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.

## 15.10 Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf

Dieser Artikel beschreibt die Implementierung:

- Eines C++ Moduls [▶ 241], 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 \[▶ 255\]](#), um die Funktionalität des C++ Moduls aufzurufen  
Dass keine hartkodierte 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

**Erhalten Sie den Quellcode für dieses Beispiel:**

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf beide Projekte rechtsklicken und auf „Build“ klicken)
4. Das Programm starten - auf die „activate configuration“ Schaltfläche in TwinCAT 3 drücken

## 15.10.1 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.

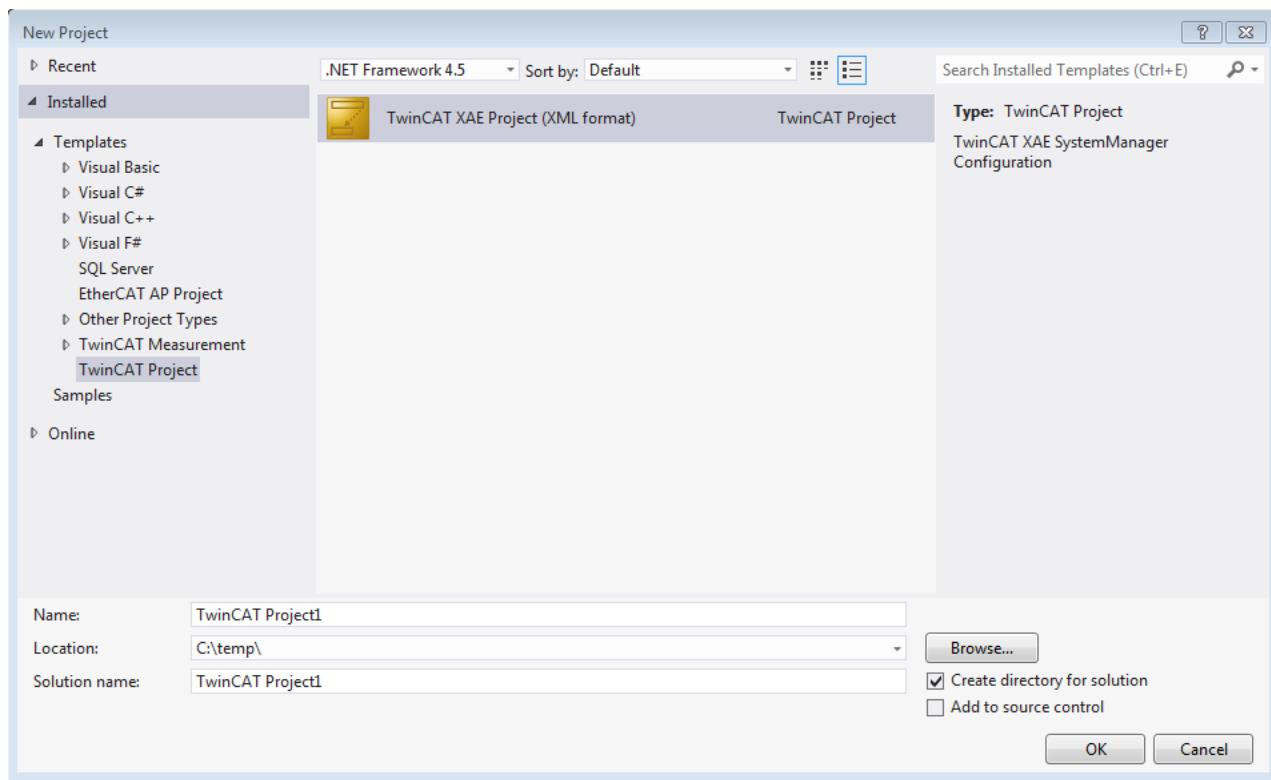
- [Aufruf der Funktionalität von der SPS Logik her \[▶ 240\]](#) - also von der SPS auf C++ Code einwirken
- [Aufruf der Funktionalität von der C++ Logik \[▶ 267\]](#) - also Wechselwirkung zwischen zwei C++ Modulen

Dieser Artikel beschreibt:

- Schritt 1: [Erstellen Sie ein neues TwinCAT 3 Projekt \[▶ 241\]](#)
- Schritt 2: [Erstellen Sie einen neuen TwinCAT 3 C++ Treiber \[▶ 242\]](#)
- Schritt 3: [Eine neue TwinCAT 3 Schnittstelle erzeugen \[▶ 244\]](#)
- Schritt 4: [Fügen Sie der Schnittstelle Methoden hinzu \[▶ 245\]](#)
- Schritt 5: [Neue Schnittstelle zum Modul hinzufügen \[▶ 248\]](#)
- Schritt 6: [Starten Sie den TwinCAT TMC Code Generator um einen Code für die Modulklassenbeschreibung zu erzeugen \[▶ 250\]](#)
- Schritt 7: [Implementierung der Membervariablen und des Konstruktors \[▶ 250\]](#)
- Schritt 8: [Methoden implementieren \[▶ 251\]](#)
- Schritt 9: [Zyklische Aktualisierung implementieren \[▶ 252\]](#)
- Schritt 10: [Code kompilieren \[▶ 253\]](#)
- Schritt 11: [Eine Instanz des C++ Moduls erstellen \[▶ 254\]](#)
- Schritt 12: [Fertig, Ergebnisse überprüfen \[▶ 255\]](#)

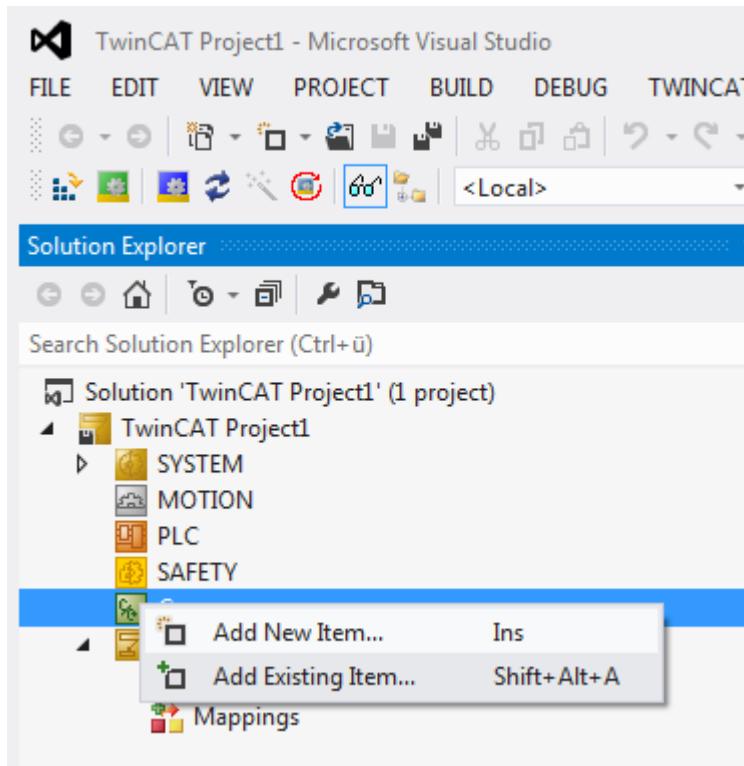
### Schritt 1: TwinCAT 3 Projekt erstellen

Als erstes wird wie gewohnt ein TwinCAT Projekt angelegt

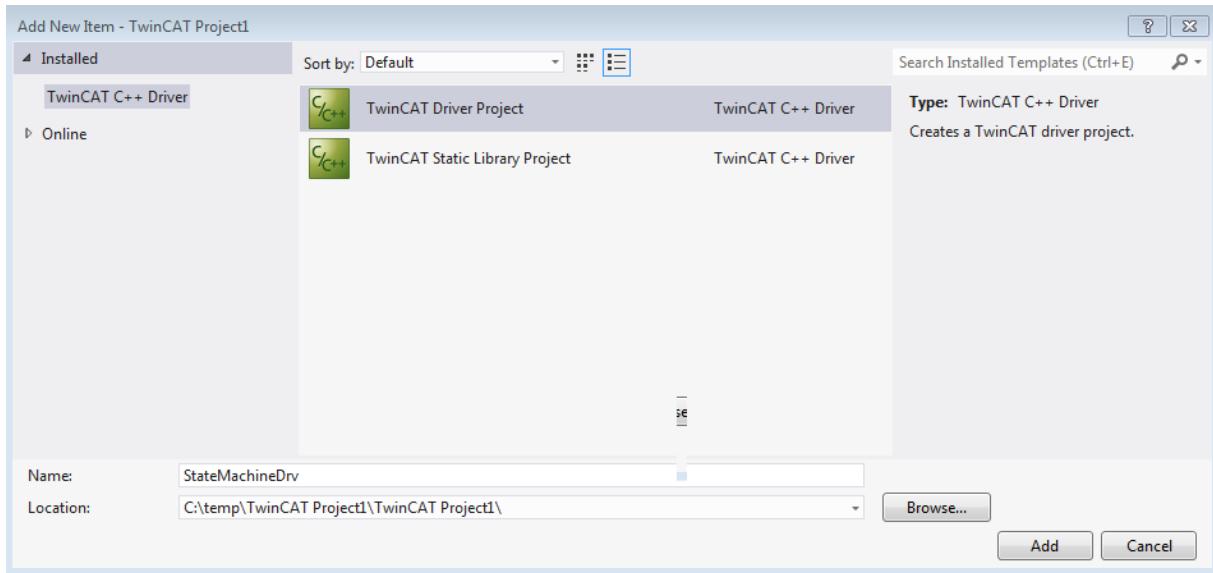


## Schritt 2: Erstellen Sie einen neuen TwinCAT 3 C++ Treiber

1. Klicken Sie mit der rechten Maustaste auf „C++“ und „Add New Item...“ auswählen.



2. Wählen Sie die Vorlage „TwinCAT Driver Project“ und geben einen Treibernamen ein, „StateMachineDrv“ in diesem Beispiel. Klicken Sie auf „Add“ um fortzufahren.

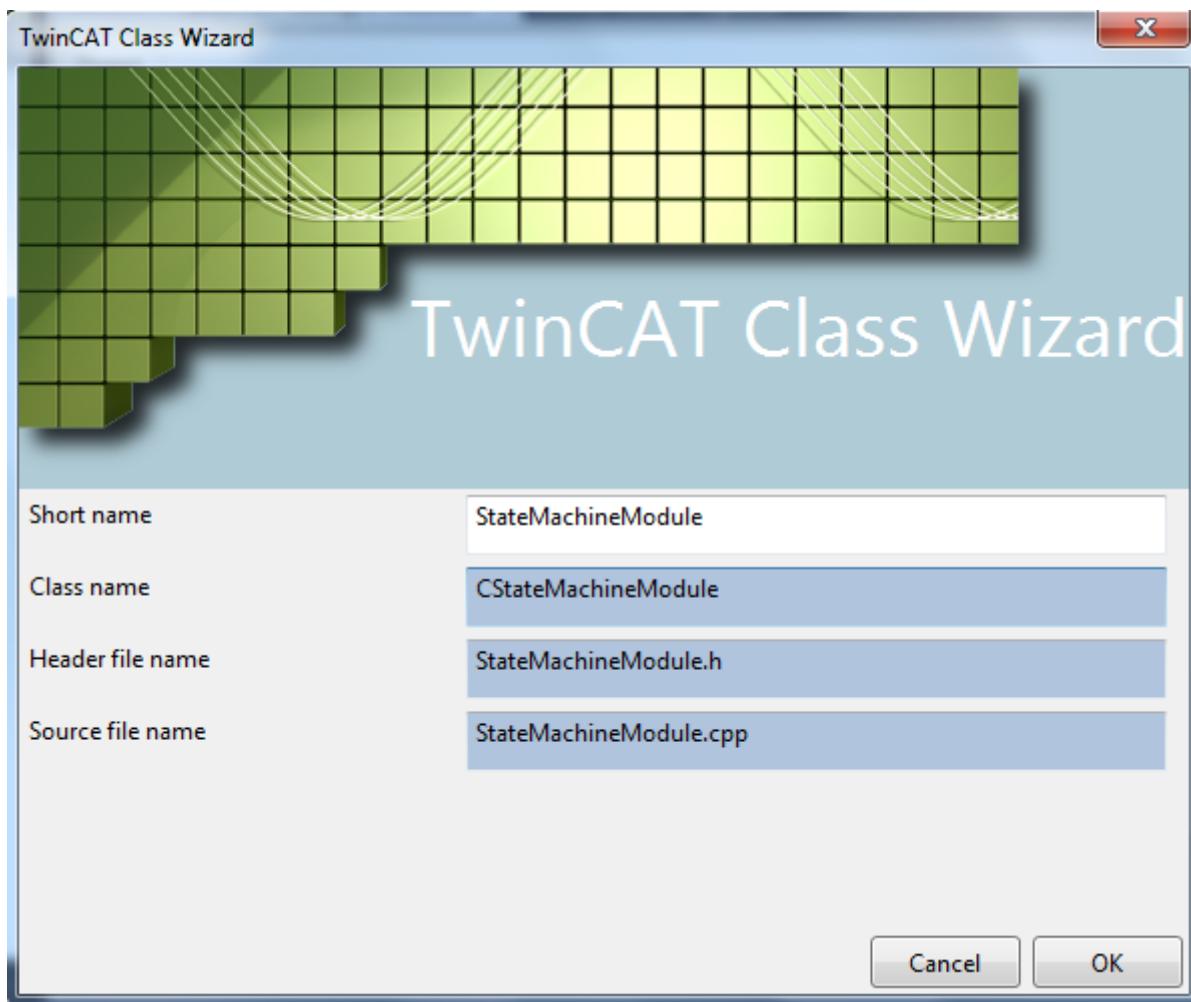


3. Wählen Sie eine für diesen neuen Treiber zu verwendende Vorlage. In diesem Beispiel ist „TwinCAT Module Class with Cyclic IO“ gewählt, da der interne Zähler der Zustandsmaschine verfügbar ist, um den IO zugeordnet zu werden.
4. Klicken Sie auf „Add“ um fortzufahren.



5. Einen Namen für die neue Klasse im C++ Treiber „StateMachineDrv“ angeben.  
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

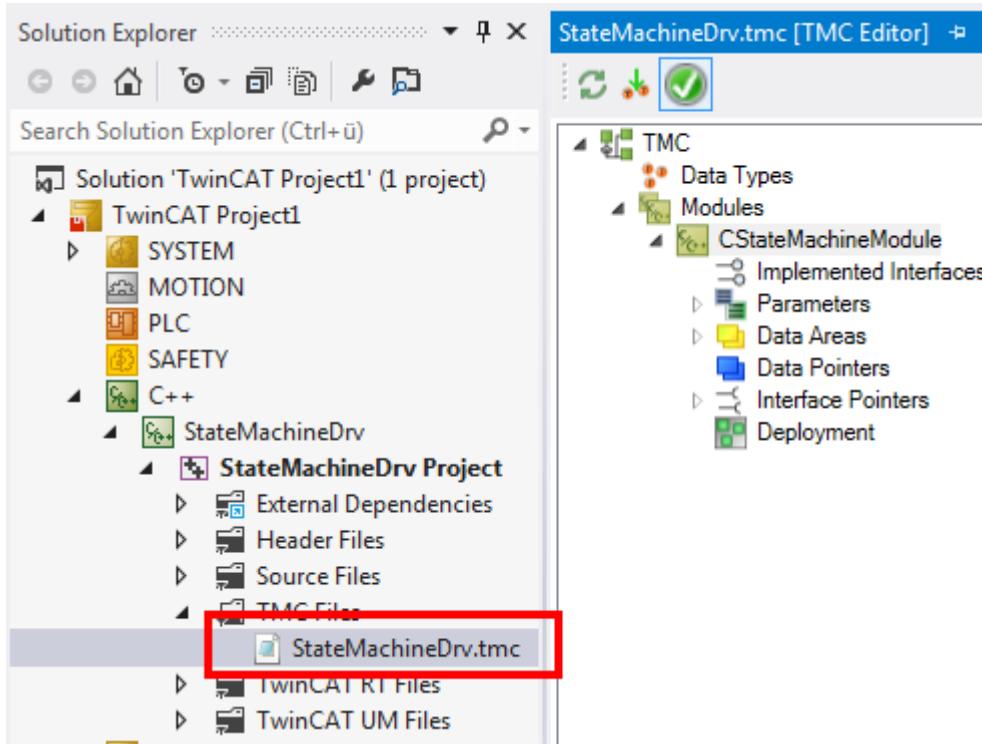


Hinweis

#### Namenskonflikt

Bitte verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen.  
Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.

7. Starten Sie den TMC Editor mittels Doppelklick auf „StateMachineDrv.tmc“.



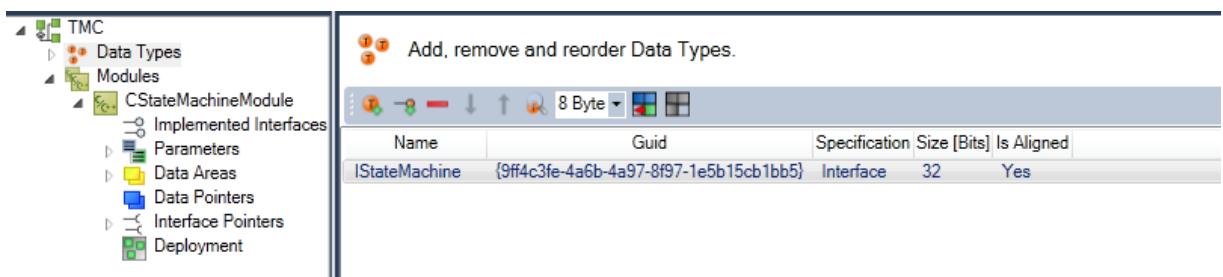
8. Wählen Sie innerhalb des TMC Editors „Data Types“

9. Fügen Sie eine neue Schnittstelle mittels Klicken auf die „Adds a new interface“



10. Daraufhin wird ein neuer Eintrag „IInterface1“ aufgeführt; Öffnen Sie „IInterface1“ (Doppel-Klick) um die Eigenschaften des Interfaces zu verändern.

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



#### Schritt 4: Fügen Sie der Schnittstelle Methoden hinzu

12. 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:

13. Den Standardnamen „Method1“ durch aussagekräftigen Namen ersetzen, in diesem Beispiel „Start“.

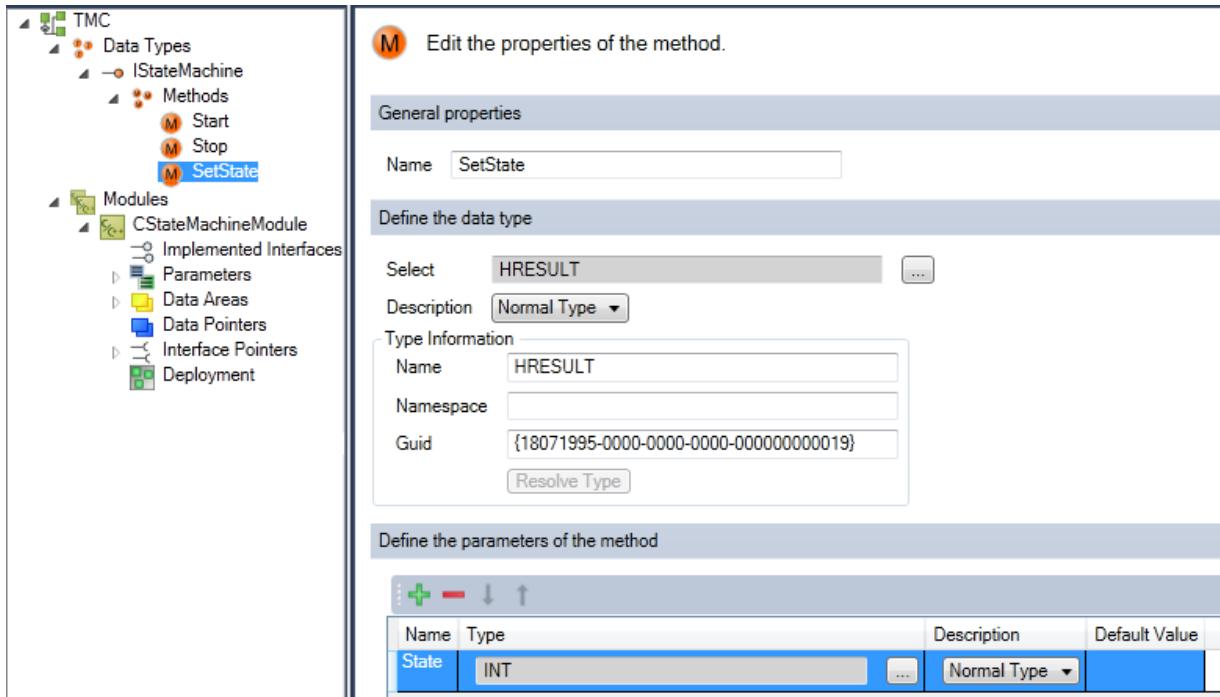
The screenshot shows the TMC configuration interface. On the left, the project tree under 'TMC' shows 'Data Types' with 'IStateMachine' selected, which contains 'Methods' with 'Start' highlighted. On the right, a dialog box titled 'Edit the properties of the method.' is open. It has three main sections: 'General properties' (Name: Start), 'Define the data type' (Select: HRESULT, Description: Normal Type), and 'Type Information' (Name: HRESULT, Namespace: empty, Guid: {18071995-0000-0000-0000-000000000019}). Below these is a section 'Define the parameters of the method' with a table header row: Name, Type, Description, Default Value.

14. Ein zweite Methode hinzufügen und diese „Stop“ nennen.

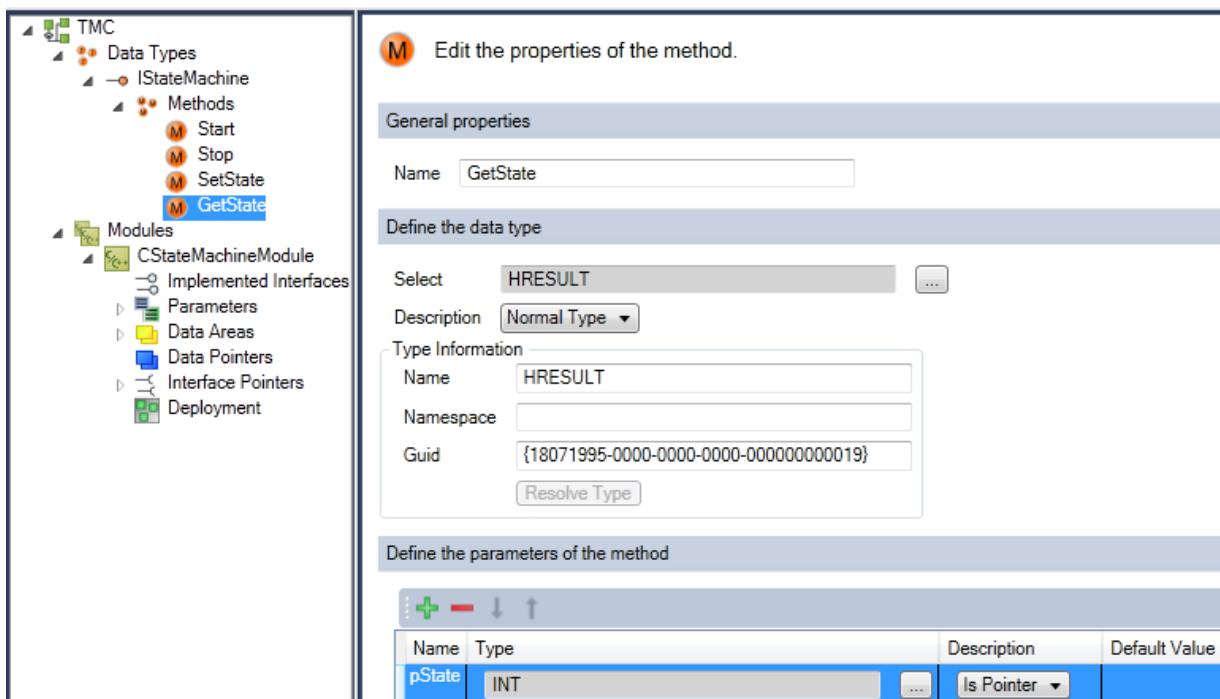
The screenshot shows the TMC configuration interface. The project tree now includes a new 'Stop' method under 'Methods'. The 'Stop' method is selected in the configuration dialog on the right. The 'General properties' section shows 'Name: Stop'. The 'Define the data type' section shows 'Select: HRESULT, Description: Normal Type'. The 'Type Information' section shows 'Name: HRESULT, Namespace: empty, Guid: {18071995-0000-0000-0000-000000000019}'. Below these is a section 'Define the parameters of the method' with a table header row: Name, Type, Description, Default Value.

15. Ein dritte Methode hinzufügen und diese „SetState“ nennen.

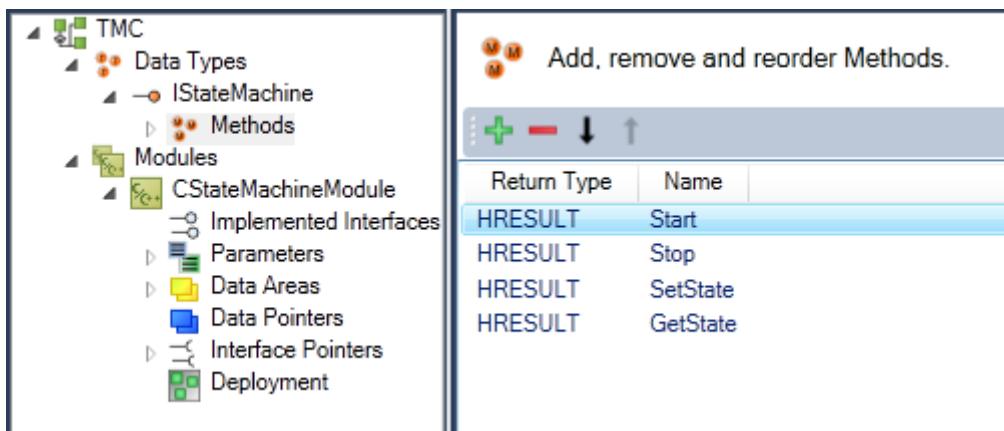
16. Anschließend können Sie mit einem Klick auf „Add a new parameter“ Parameter hinzufügen / Parameter der Methode „SetState“ bearbeiten



17. Standardmäßig wird der neue Parameter „Parameter1“ als „Normal Type“ „INT“ erzeugt.  
 18. Auf den Namen „Parameter1“ klicken, in das Bearbeitungsfeld gehen und den Namen in „State“ ändern.  
 19. Nachdem „Start“, „Stop“ und „SetState“ definiert sind, eine weitere Methode definieren  
 20. In „GetState“ umbenennen  
 21. Einen Parameter hinzufügen und ihn „pState“ nennen (der so konzipiert ist, um später ein Zeiger zu werden)  
 22. „Normal Type“ zu „Is Pointer“ ändern

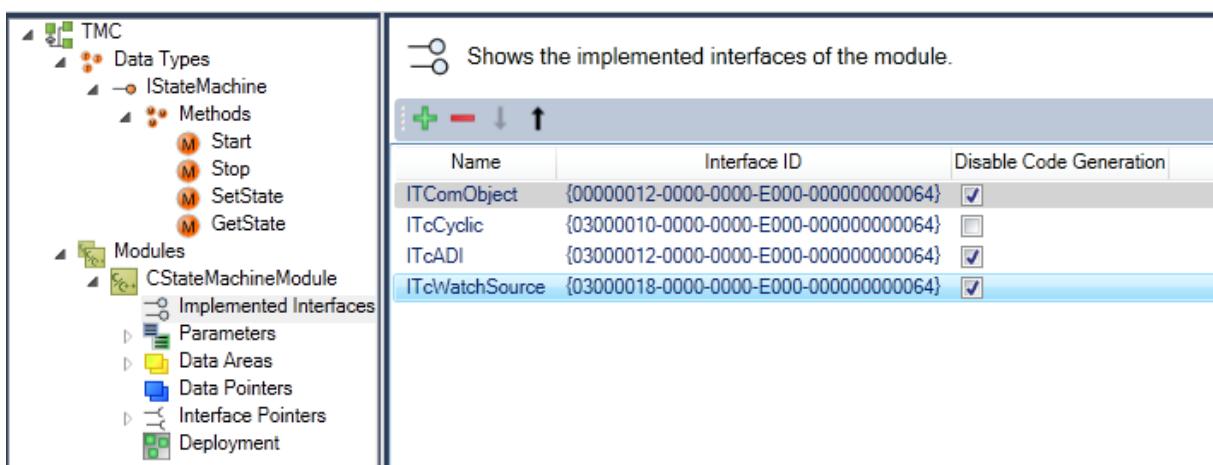


23. Daraufhin erhalten wir eine Liste aller Methoden - mit den „nach oben“ / „nach unten“ Schaltflächen kann die Reihenfolge der Methoden verändert werden.

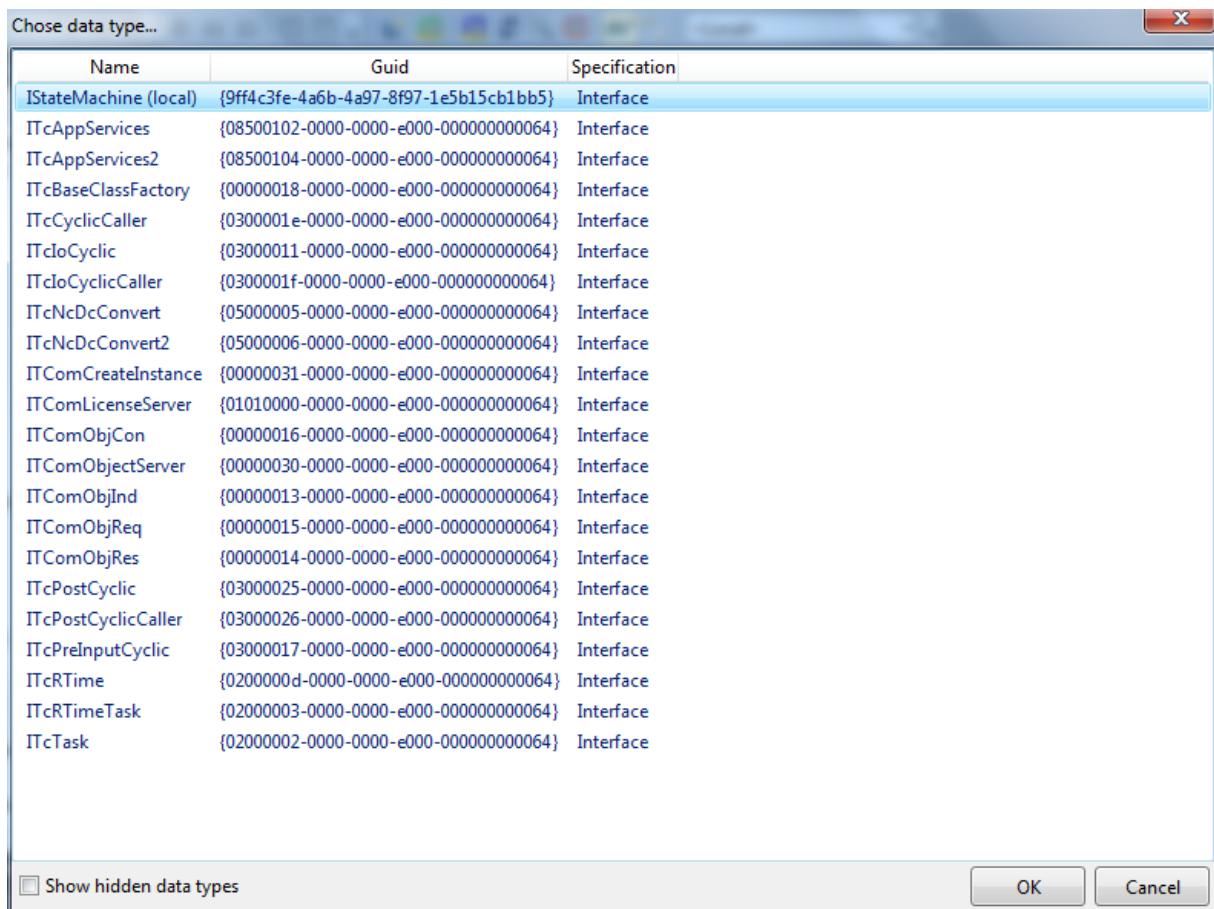


#### Schritt 5: Neue Schnittstelle zum Modul hinzufügen

24. Wählen Sie das Modul, das mit der neuen Schnittstelle erweitert werden soll - in diesem Falle wählen Sie das Ziel „**Modules->CStateMachineModule**“.
25. Die Liste der implementierten Schnittstellen wird mittels Klicken auf die „+“ Schaltfläche mit „Add a new interface to the module“ um eine neue Schnittstelle erweitert.



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



⇒ In Folge dessen ist nun die neue Schnittstelle „**IStateMachine**“ Teil der Modulbeschreibung.

TMC

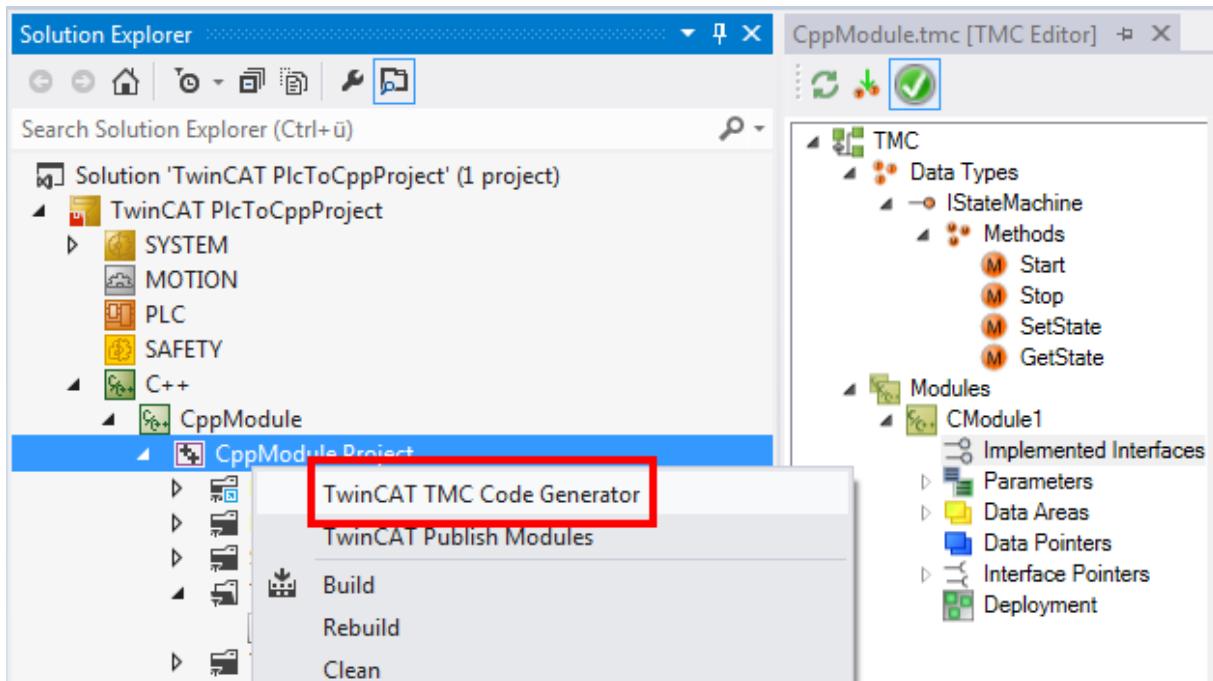
- Data Types
- Modules
  - CStateMachineModule
    - Implemented Interfaces
    - Parameters
    - Data Areas
    - Data Pointers
    - Interface Pointers
    - Deployment

Shows the implemented interfaces of the module.

Name	Interface ID	Disable Code Generation
ITComObject	{00000012-0000-0000-E000-0000000000064}	<input checked="" type="checkbox"/>
ITcCyclic	{03000010-0000-0000-E000-0000000000064}	<input type="checkbox"/>
ITcADI	{03000012-0000-0000-E000-0000000000064}	<input checked="" type="checkbox"/>
ITcWatchSource	{03000018-0000-0000-E000-0000000000064}	<input checked="" type="checkbox"/>
IStateMachine	{9ff4c3fe-4a6b-4a97-8f97-1e5b15cb1bb5}	<input type="checkbox"/>

### Schritt 6: Starten TwinCAT TMC Code Generator

27. Um den C/C++ Code anhand von diesem Modul 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 „**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

Membervariablen in die Header-Datei „**StateMachineModule.h**“ hinzufügen.

The screenshot shows the Microsoft Visual Studio interface. On the left is the Solution Explorer, displaying a 'TwinCAT Project PLC calling C-Method' solution with various project and file nodes under 'TwinCAT Project PLC calling C-Method'. On the right is the Code Editor window titled 'StateMachineModule.h' with '(Global Scope)' selected. The code in the editor is as follows:

```
HRESULT AddModuleToCaller();
VOID RemoveModuleFromCaller();

///<AutoGeneratedContent id="Members">
TcTraceLevel m_TraceLevelMax;
StateMachineModuleParameter m_Parameter;
StateMachineModuleInputs m_Inputs;
StateMachineModuleOutputs m_Outputs;
ITcCyclicCallerInfoPtr m_spCyclicCaller;
///</AutoGeneratedContent>

// Tracing
CTcTrace m_Trace;

// TODO: Custom variable
UINT m_counter;
BOOL m_bRun;
SHORT m_State;
};
```

### Schritt 8: Methoden implementieren

Den Code für die vier Methoden in der StateMachineModule.cpp implementieren:

```
///<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 Zählerwert 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.

```

Solution Explorer ..... × StateMachineModule.h
Search Solution Explorer (Ctrl+ü) 🔍
PLC
  PLC-calling-statemachine
  SAFETY
C++
  StateMachineDrv
    StateMachineDrv Project
      External Dependencies
      Header Files
        Resource.h
        StateMachineDrvCla.h
        StateMachineDrvIntf.h
        StateMachineDrvSer.h
        StateMachineModul.h
        TcPch.h
      Source Files
        StateMachineDrv.rc
        StateMachineDrvCla.h
        StateMachineModul.h
        TcPch.cpp
      TMC Files
      TwinCAT RT Files
StateMachineModule.h
//<AutoGeneratedContent id="Implementationof_ITcCyclic">
HRESULT CStateMachineModule::CycleUpdate(ITcTask * ipTask, ITcUnknown * ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // TODO: Replace the sample with your cyclic code
    if (!m_bRun)
        return hr;

    switch (m_State)
    {
    case 1:
        m_counter++;
        break;

    case 2:
        m_counter--;
        break;

    }

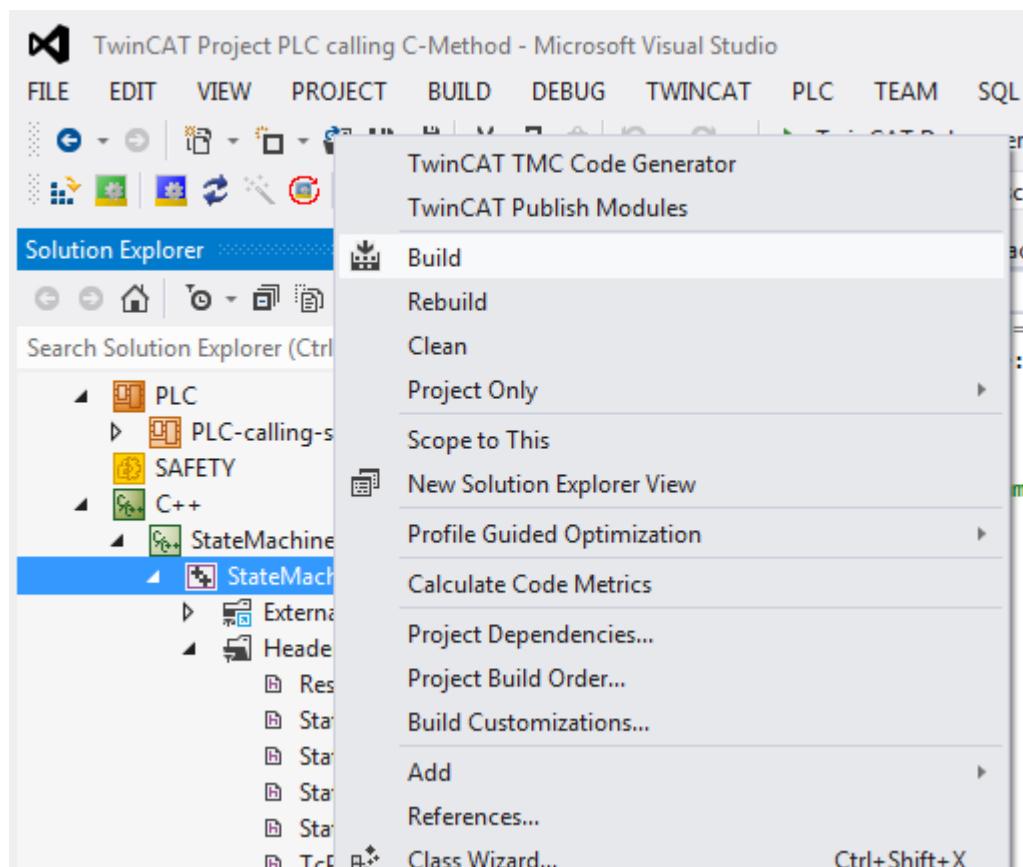
    m_Outputs.Value=m_counter;

    return hr;
}//</AutoGeneratedContent>

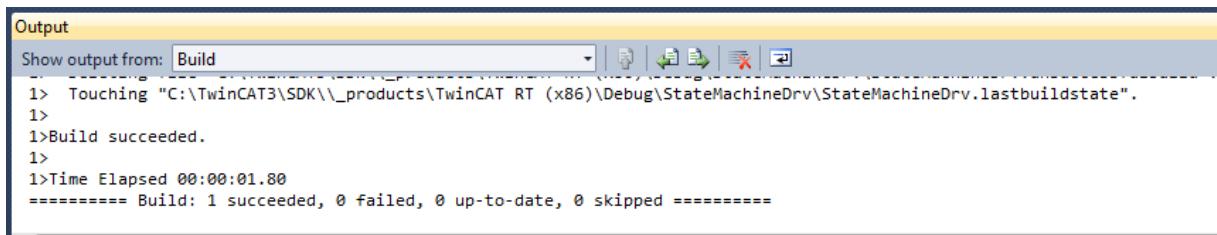
```

## Schritt 10: Code kompilieren

28. Nach der Implementierung aller Schnittstellen ist es Zeit, den Code zu kompilieren - nach Rechtsklick „Build“ wählen.



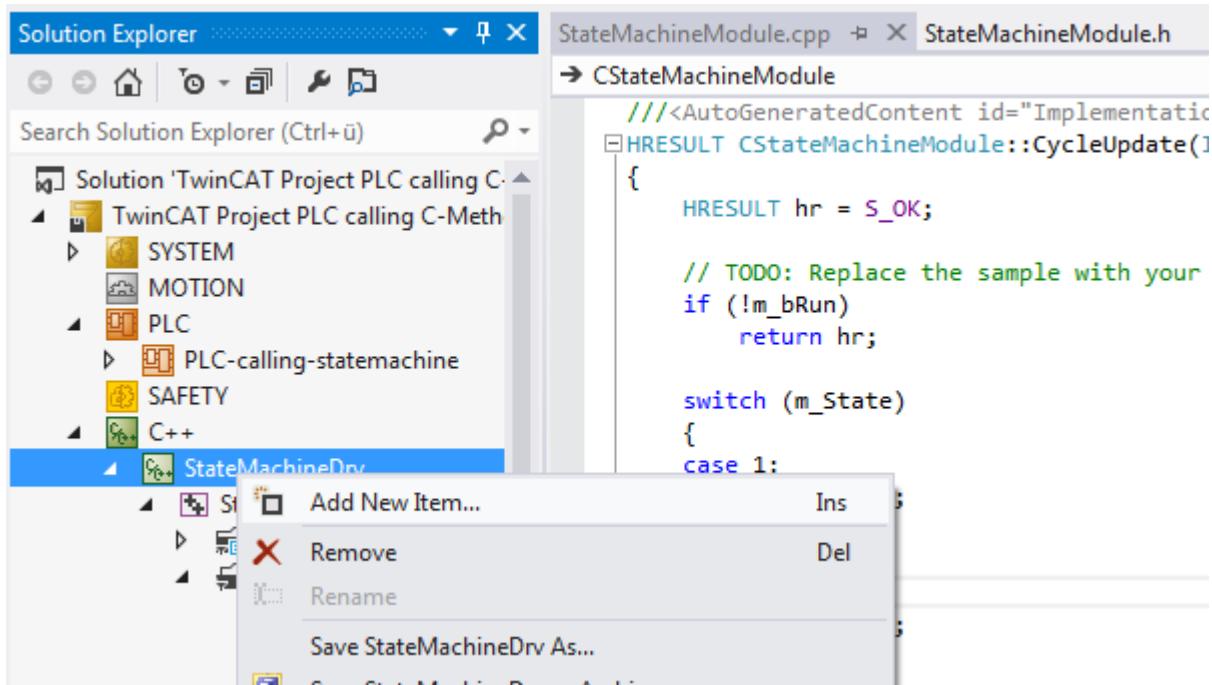
29. Die Kompilation wiederholen und Ihren Code so lange optimieren, bis das Ergebnis so aussieht.



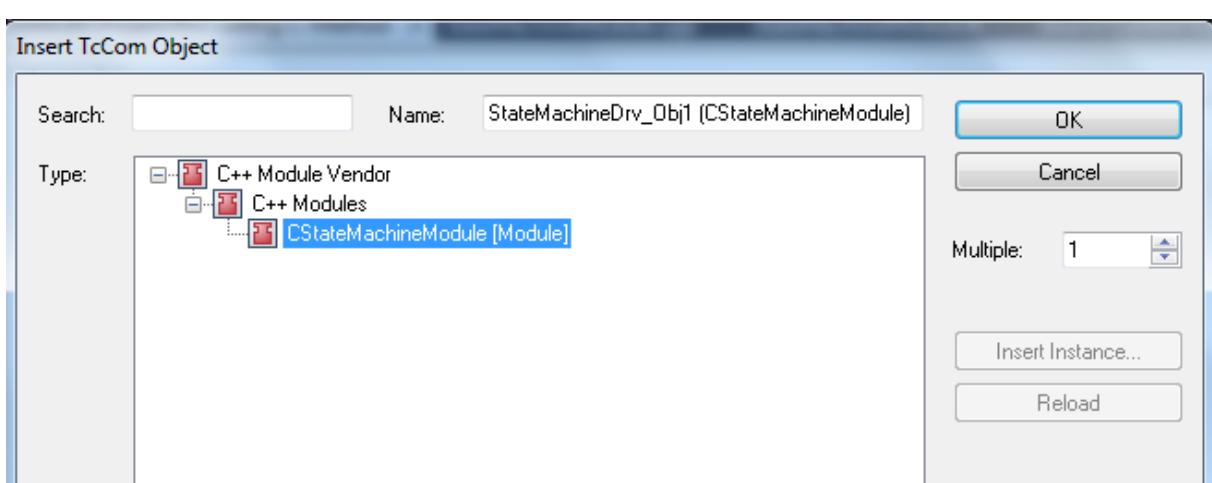
```
Output
Show output from: Build
1> Touching "C:\TwinCAT3\SDK\_products\TwinCAT RT (x86)\Debug\StateMachineDrv\StateMachineDrv.lastbuildstate".
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:01.80
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
```

### Schritt 11: Eine Instanz des C++ Moduls erstellen

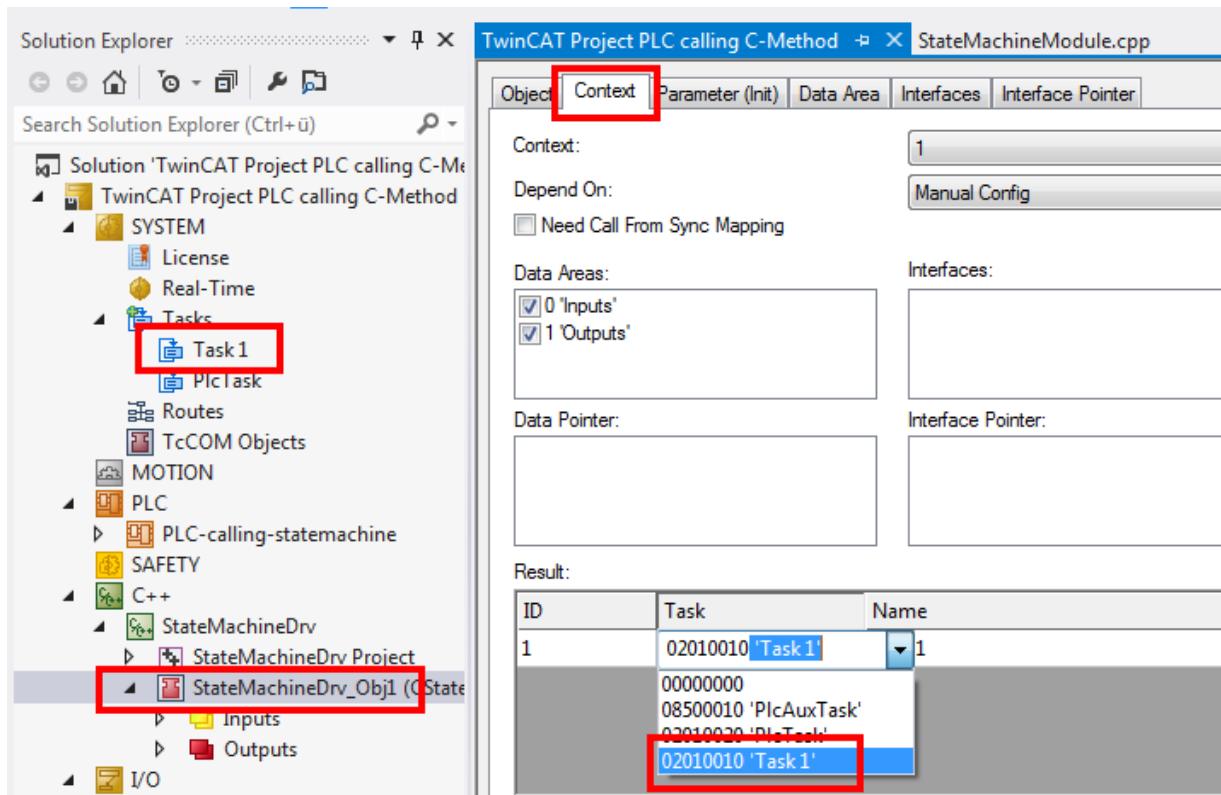
30. Rechtsklick auf das C++ Projekt und Auswahl von „Add New Item...“ um eine neue Modulinstanz zu erstellen



31. Wählen Sie das Modul aus, das als neue Instanz hinzugefügt werden soll - in diesem Falle „CStateMachineModule“



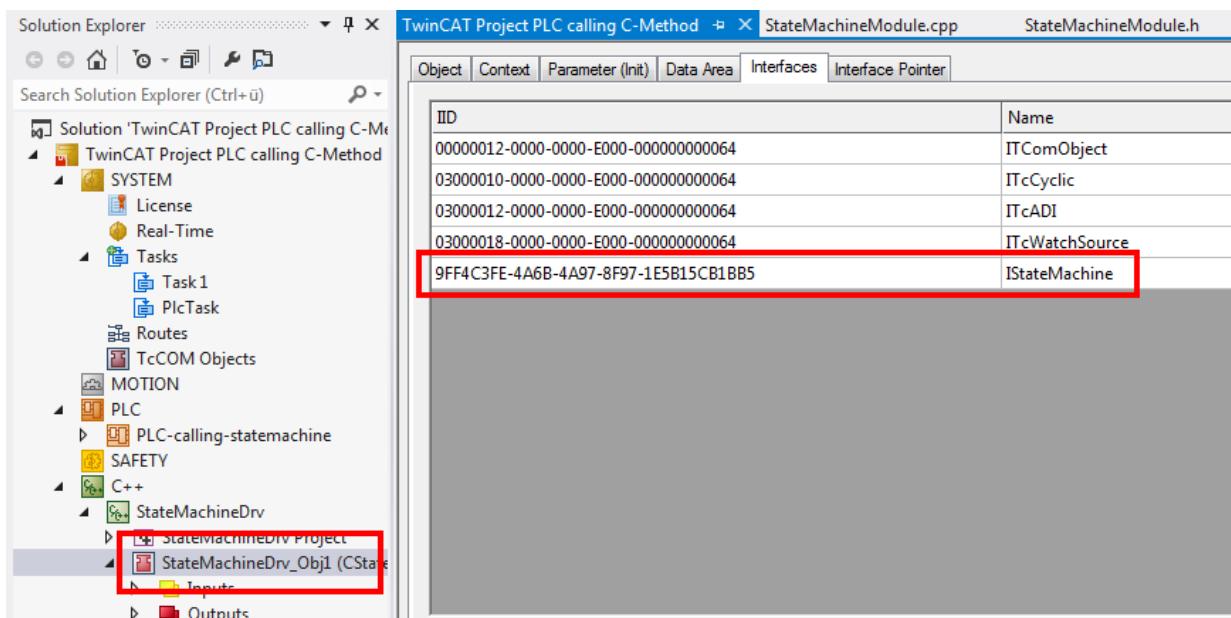
32. Wie üblich, muss die Instanz einem Task zugeordnet werden:



#### Schritt 12: Fertig - Ergebnis überprüfen

33. Navigieren Sie zum neuen im Solution-Baum aufgeführten Modul und wählen die Registerkarte „Interfaces“ auf der rechten Seite.

⇒ Die neue Schnittstelle „IStateMachine“ ist aufgelistet



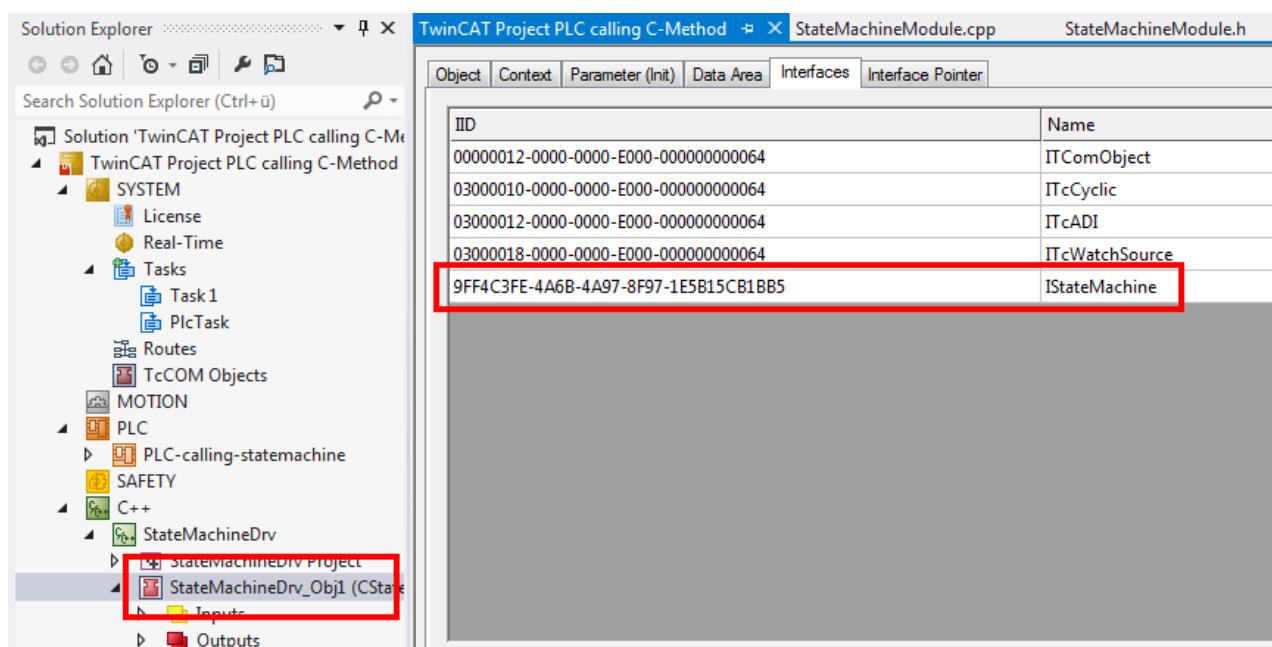
## 15.10.2 SPS um Methoden aufzurufen, die von einem anderen Modul angeboten werden

Dieser Artikel beschreibt wie eine SPS eine Methode, die von einem anderen Modul bereitgestellt wird, aufrufen kann - hier: das zuvor definierte C++ Modul.

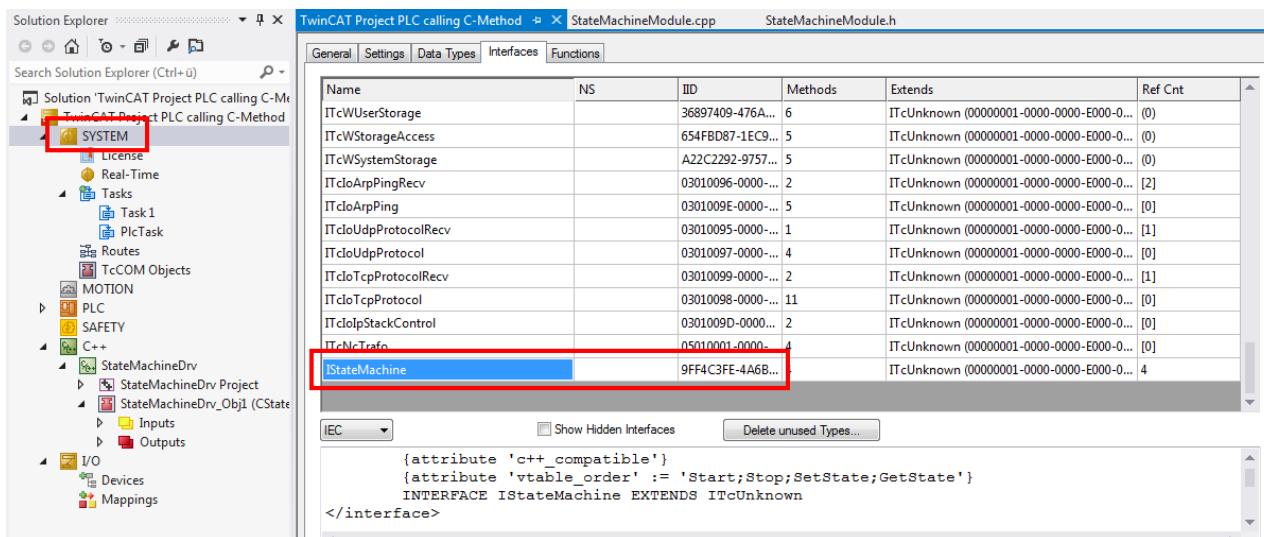
- Schritt 1: [Verfügbare Schnittstellen überprüfen \[▶ 256\]](#)
- Schritt 2: [Ein neues SPS-Projekt erstellen \[▶ 257\]](#)
- Schritt 3: [Eine neue FB-StateMachine hinzufügen \[▶ 258\]](#) (die als der C++ Modulmethoden aufrufende Proxy fungiert)
- Schritt 4: [Funktionsbaustein-Schnittstelle säubern \[▶ 260\]](#)
- Schritt 5: [FB-Methoden „FB init“ und „xit“ hinzufügen \[▶ 261\]](#)
- Schritt 6: [FB-Methoden implementieren \[▶ 262\]](#)
- Schritt 7: [FB-StateMachine in der SPS aufrufen \[▶ 265\]](#)
- Schritt 8: [SPS-Code kompilieren \[▶ 265\]](#)
- Schritt 9: [SPS FB mit C++ Instanz verknüpfen \[▶ 266\]](#)
- Schritt 10: [Die Ausführung von beiden Modulen, SPS und C++, beobachten \[▶ 267\]](#)

### Schritt 1: Verfügbare Schnittstellen überprüfen

Option 1: Zu C++ Modulinstanz navigieren, Karteireiter „Interfaces“ auswählen. Die Schnittstelle „IStateMachine“ befindet sich in der Liste, mit ihrer spezifischen IID (Interface ID)



Option 2: Zu „System“ navigieren, Karteireiter „Interfaces“ auswählen. Die Schnittstelle „IStateMachine“ befindet sich in der Liste, mit ihrer spezifischen IID (Interface ID)

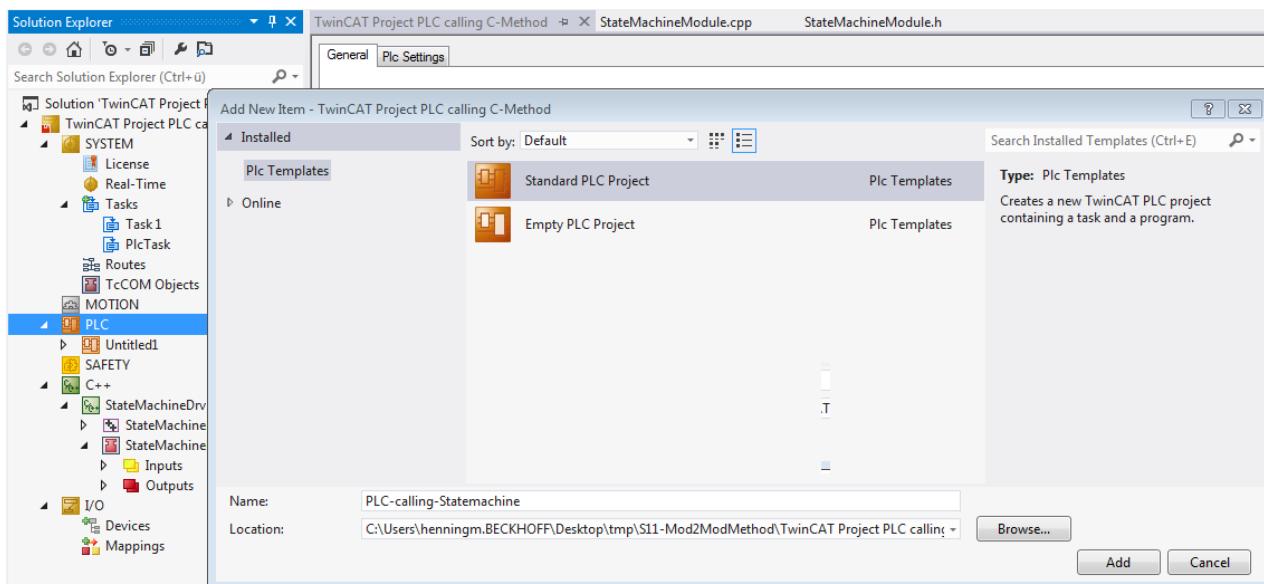


Im unteren Bereich wird der hinterlegte Code in unterschiedlichen Programmiersprachen dargestellt.

## Schritt 2: Neues SPS Projekt anlegen

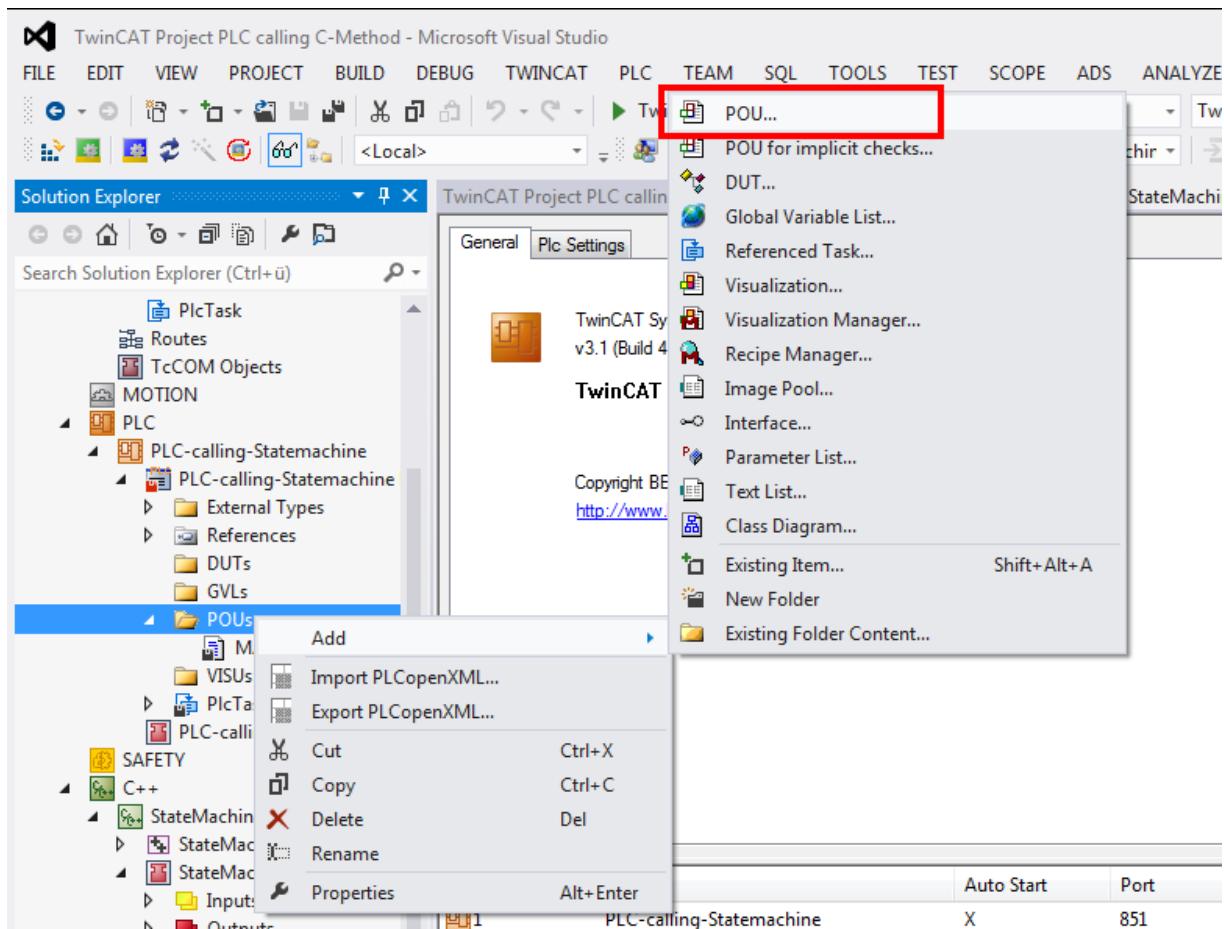
Es wird ein „Standard PLC Project“, genannt „PLC-calling-StateMachine“, angelegt.

1. Rechts-Klick auf den SPS Knoten
2. „Standard PLC Project“ auswählen
3. Namen anpassen

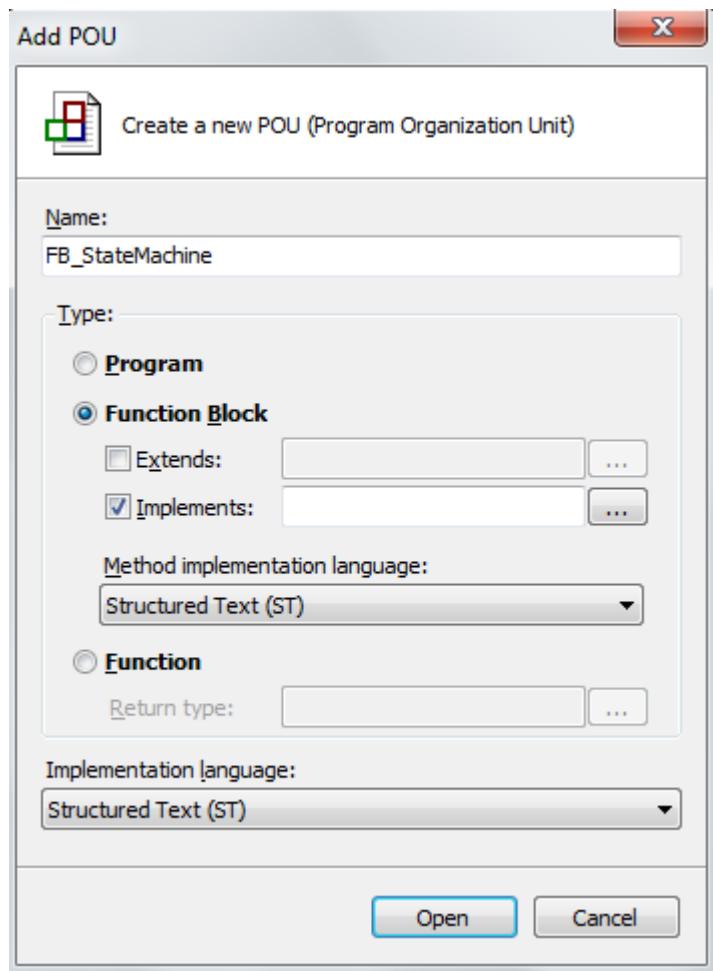


**Schritt 3: Einen Funktionsbaustein (FB) hinzufügen (der als der C++ Modulmethoden aufrufende Proxy fungiert)**

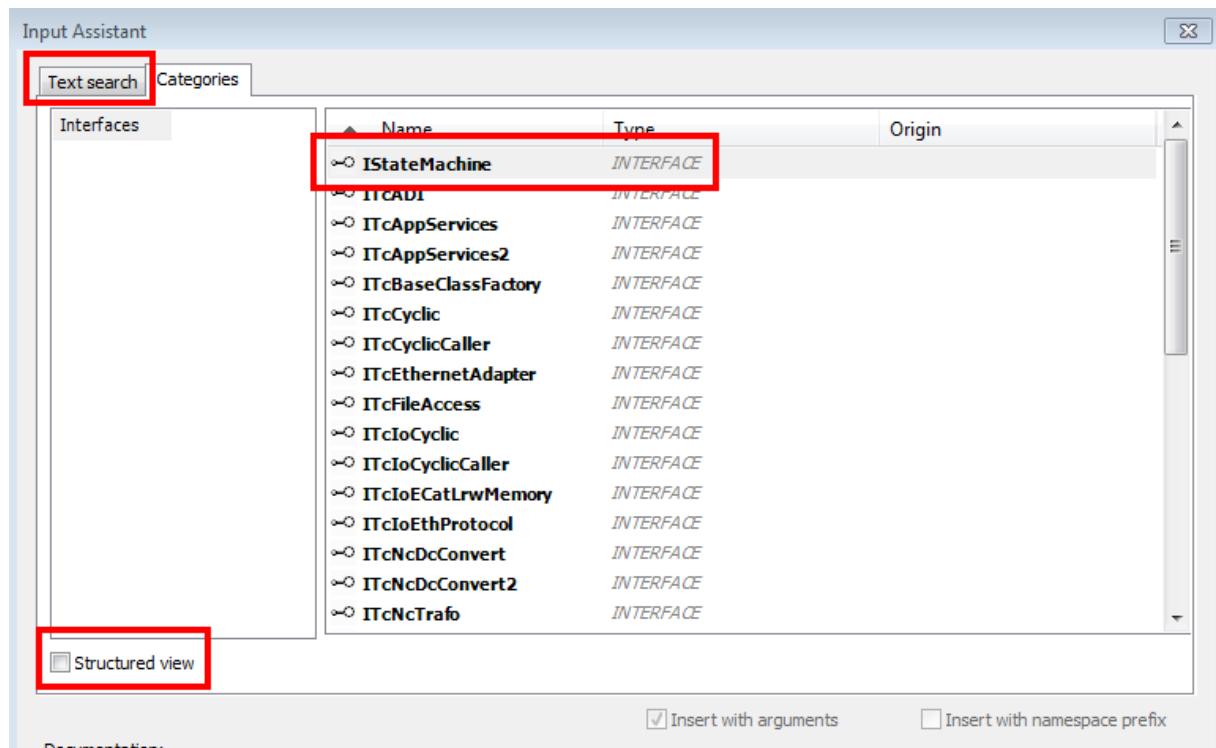
4. Auf POUs rechtsklicken und „Add->POU...“ auswählen.



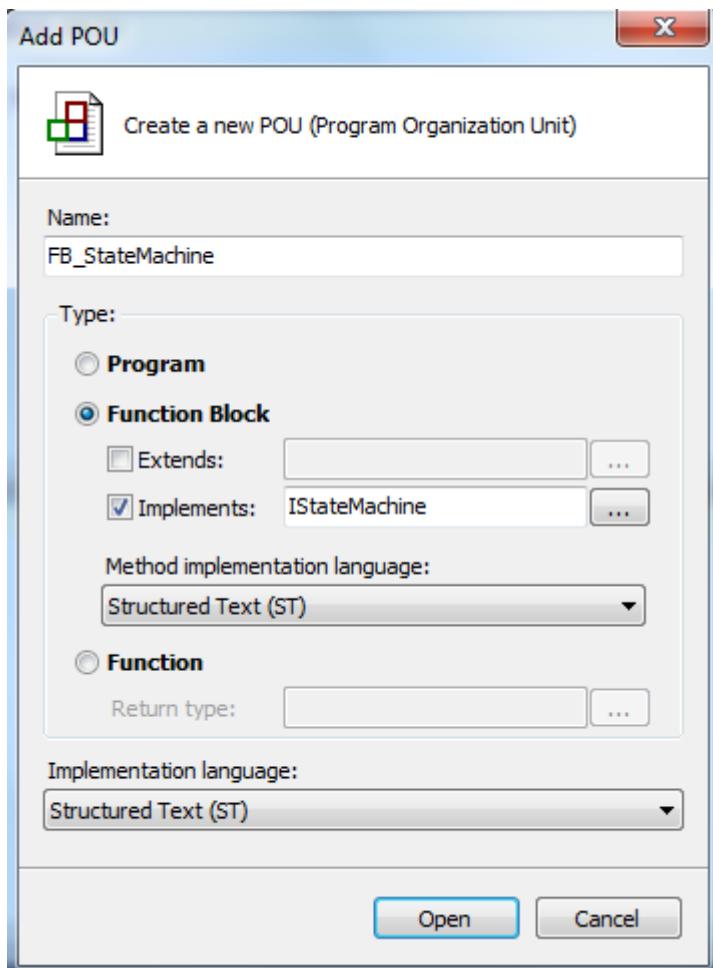
5. Neuen zu erstellenden FB definieren, der später als Proxy zum Aufrufen von C++ Klasse fungiert  
 Name des neuen FB eingeben: „FB\_StateMachine“ und wählen Sie „Function Block“, dann „Implements“ auswählen und schließlich „...“ Schaltfläche auswählen



6. Die Schnittstelle entweder über „Text Search“-Karteireiter oder mittels Aufheben der Auswahl „Structured View“ bei Eingangsassistenten auswählen:



7. „IStateMachine“ auswählen und über „OK“ verlassen  
⇒ Daraufhin wird die Schnittstelle „IStateMachine“ als zu implementierende Schnittstelle aufgelistet.
8. Wählen Sie für „Method implementation language“ „Structured Text (ST)“ aus
9. Wählen Sie für „Implementation language“ „Structured Text (ST)“ aus
10. Diesen Dialog mit „Open“ beenden



#### Schritt 4: Funktionsbaustein-Schnittstelle anpassen

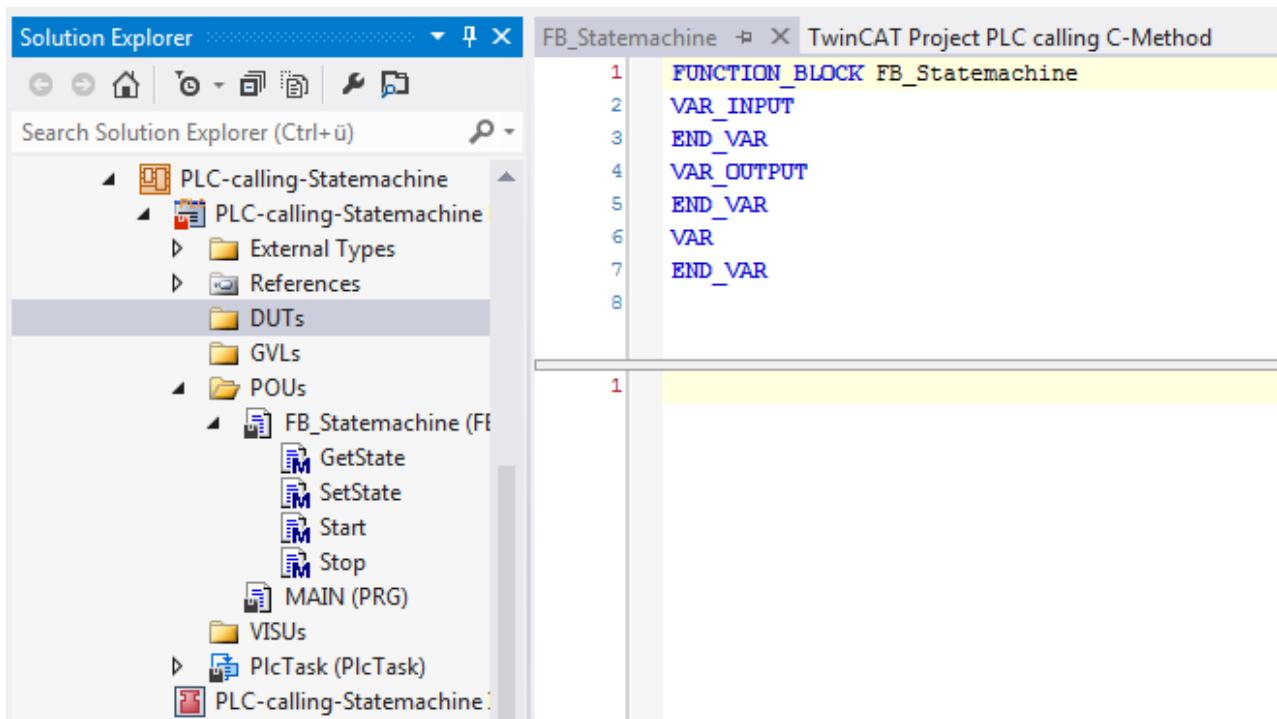
Als Ergebnis der Erstellung eines FB, der die Schnittstelle „IStateMachine“ implementiert, wird der Assistent einen FB mit entsprechenden Methoden erstellen.

Der FB\_StateMachine stellt 4 Methoden zur Verfügung:

- GetState
- SetState
- Start
- Stop

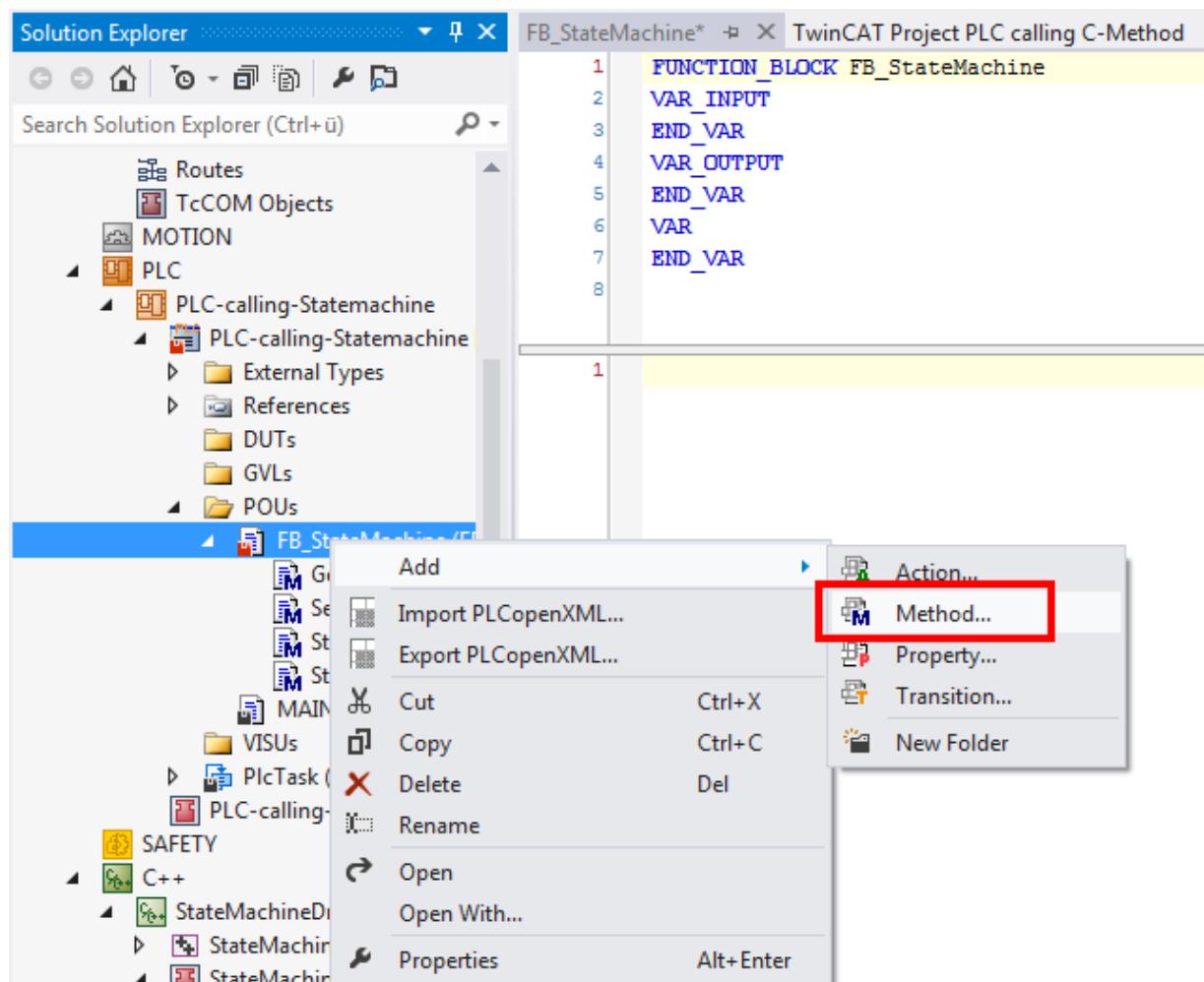
11. Da der Funktionsbaustein als Proxy agieren soll, implementiert er selber nicht das Interface. Also wird das „Implements IStateMachine“ gelöscht
12. Auch die Methoden TcAddRef, TcQueryInterface und TcRelease werden für einen Proxy Funktionsbaustein nicht benötigt und können gelöscht werden.

Es ergibt sich:

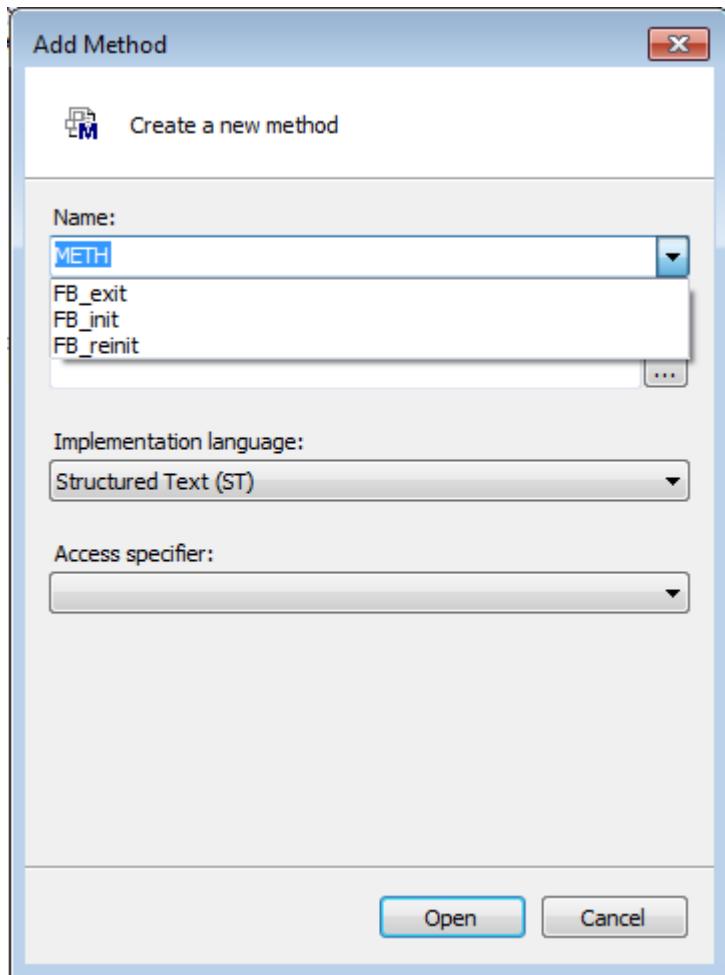


### Schritt 5: FB-Methoden „FB\_init“ (Konstruktor) und „FB\_exit“ (Destruktor) hinzufügen

13. Rechtsklick auf „FB\_Schemamchine“ im Baum und „Add“ --> „Method..“ auswählen

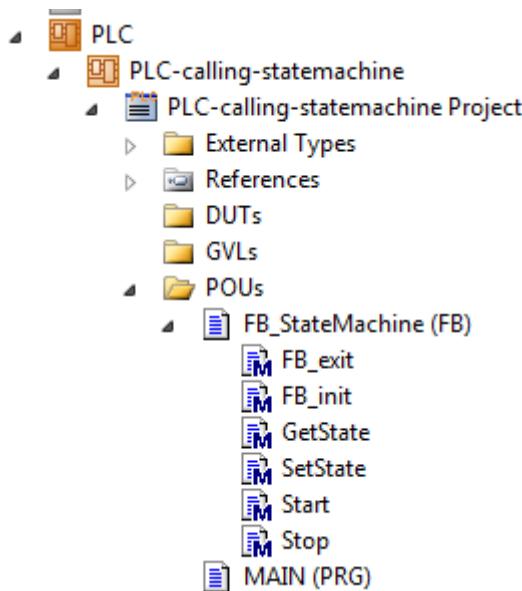


14. Fügen Sie die Methoden „FB\_exit“ und „FB\_init“ hinzu - beide mit „Structured Text (ST)“ als „Implementation language“. Sie stehen als vordefinierter Name zur Verfügung.



15. Dialog jeweils mittels Klicken auf „Open“ verlassen

⇒ Zum Schluss stehen alle notwendigen Methoden zur Verfügung:



#### Schritt 6: FB-Methoden implementieren

Nun müssen alle Methoden mit Code gefüllt werden.



## 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 hier dokumentiert.

1. Variablendeclaration der FB\_Statemachine implementieren. Der FB selber braucht keinen zyklisch auszuführenden Code.

```
FB_StateMachine.FB_exit      FB_StateMachine.FB_init      FB_StateMachine* ➔ X TwinCAT Project PLC calling C-Method
5  END_VAR
6  VAR
7    {attribute 'TcInitSymbol'}
8    oidInstance : OTCID;
9    ipStateMachine : IStateMachine; // interface pointer to the C++ StateMachine module instance
10   hrInit : HRESULT;
11 END_VAR
12
```

- ## 2. Variablen-deklaration und Code-Bereich der Methode „FB\_exit“ implementieren

```
FBStateMachine.FB_exit*  FB_init          FBStateMachine*
```

1 METHOD FB\_exit : BOOL  
2 VAR \_INPUT  
3 bInCopyCode : BOOL; // if TRUE, the exit method is called  
4 END\_VAR  
5

IF NOT bInCopyCode THEN // no online change  
 FW\_SafeRelease(ADR(ipStateMachine));  
END\_IF

- ### 3. Variablen-deklaration und Code-Bereich der Methode „FB\_init“ implementieren

4. Variablen-deklaration und Code-Bereich der Methode „GetState“ implementieren.  
 (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden)

```
FB_StateMachine.GetState* ↗ X FB_StateMachine.FB_exit*           FB_StateMachine.FB_init*
1 METHOD GetState : HRESULT
2 VAR_INPUT
3     pState : POINTER TO INT;
4 END_VAR
5
6
7 IF (ipStateMachine <> 0) THEN
8     GetState:= ipStateMachine.GetState(pState);
9 END_IF
```

5. Variablen-deklaration und Code-Bereich der Methode „SetState“ implementieren  
 (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden)

```
FB_StateMachine.SetState* ↗ X FB_StateMachine.GetState*           FB_StateMachine.FB_exit*
1 METHOD SetState : HRESULT
2 VAR_INPUT
3     State : INT;
4 END_VAR
5
6
7 IF (ipStateMachine <> 0) THEN
8     SetState:= ipStateMachine.SetState(State);
9 END_IF
```

6. Variablen-deklaration und Code-Bereich der Methode „Start“ implementieren  
 (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden)

```
FB_StateMachine.Start ↗ X FB_StateMachine.SetState*           FB_StateMachine.GetState*
1 METHOD Start : HRESULT
2
3
4
5 IF (ipStateMachine <> 0) THEN
6     Start:= ipStateMachine.Start();
7 END_IF
```

7. Variablen-deklaration und Code-Bereich der Methode „Stop“ implementieren  
 (die generierten Pragmas können gelöscht werden, da sie für einen Proxy-FB nicht benötigt werden)

```
FB_StateMachine.Stop ↗ X FB_StateMachine.Start           FB_StateMachine.SetState*
1 METHOD Stop : HRESULT
2
3
4 IF (ipStateMachine <> 0) THEN
5     Stop:= ipStateMachine.Stop();
6 END_IF
```

### Schritt 7: FB in SPS aufrufen

Die Implementierung des „FB\_StateMachine“ der als Proxy zum Aufruf der C++ Modulinstanz agiert, ist abgeschlossen.

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.

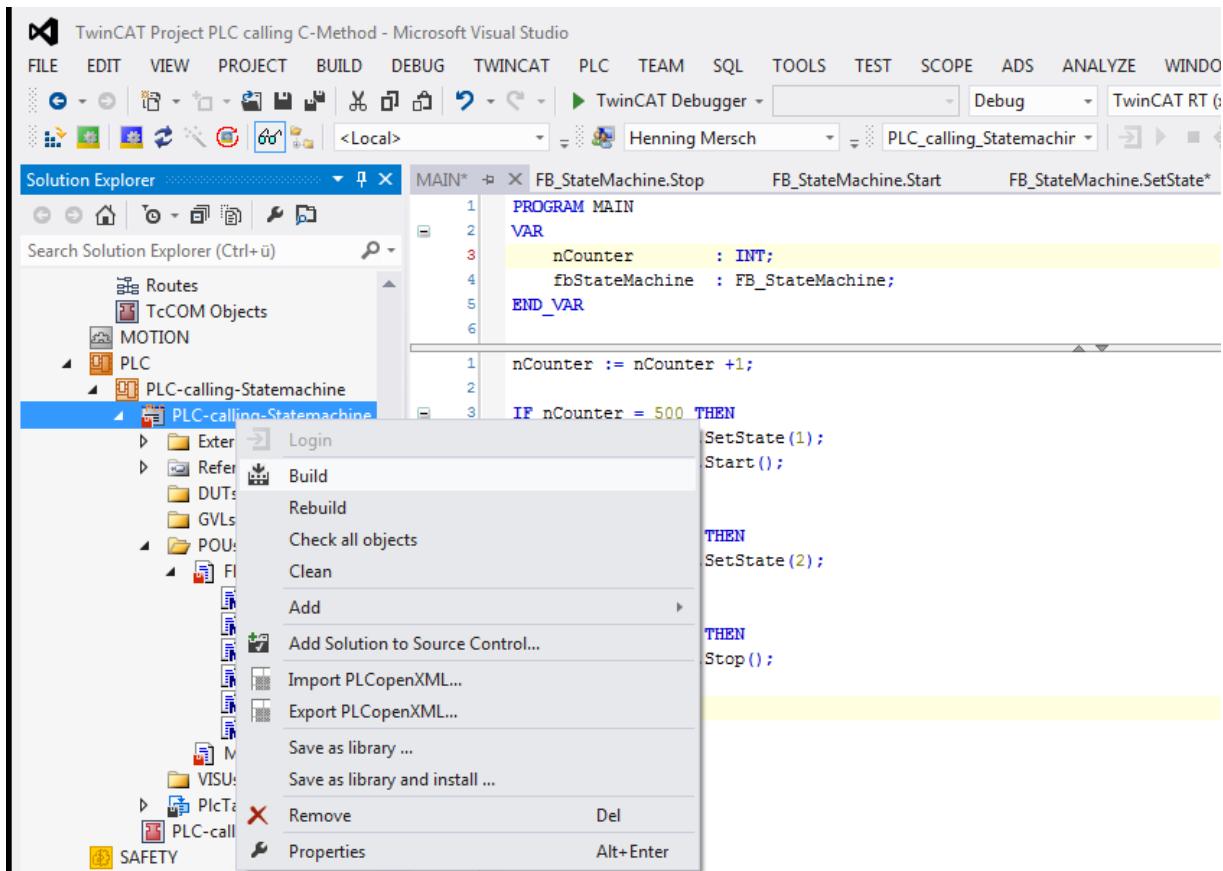
The screenshot shows the TwinCAT Project PLC calling C-Method interface. The project name is "TwinCAT Project PLC calling C-Method". The current file is "MAIN". The code is as follows:

```
MAIN  X TwinCAT Project PLC calling C-Method           FB_StateMachine.GetState
1   PROGRAM MAIN
2
3   VAR
4       nCounter      : INT;
5       ncurrentState : INT;
6       fbStateMachine : FB_StateMachine;
7
8
9   nCounter := nCounter +1;
10
11 IF nCounter = 500 THEN
12     fbStateMachine.SetState(1);
13     fbStateMachine.GetState(ADR(ncurrentState));
14     fbStateMachine.Start();
15 END_IF
16
17 IF nCounter = 1000 THEN
18     fbStateMachine.SetState(2);
19     fbStateMachine.GetState(ADR(ncurrentState));
20 END_IF
21
22 IF nCounter = 1500 THEN
23     fbStateMachine.Stop();
24     nCounter := 0;
25 END_IF
```

### Schritt 8: SPS-Code kompilieren

Nun wurde alles korrekt implementiert. Es muss nun kompiliert („Build“) werden.

## 8. Rechtsklick auf das SPS Projekt und Auswahl von „Build“



⇒ Das Ergebnis der Kompilation zeigt „1 succeeded - 0 failed“.

```

Output
Show output from: Build
-----
Rebuild All started: Project: PlcCaller, Configuration: Debug TwinCAT RT (x64) -----
PLC.PlcCaller : message: The application is up to date
PLC.PlcCaller : message: Build complete -- 0 errors, 0 warnings : ready for download!
generate TMC information...
import symbol information...
PLC.PlcCaller : message: generate boot information...
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====

```

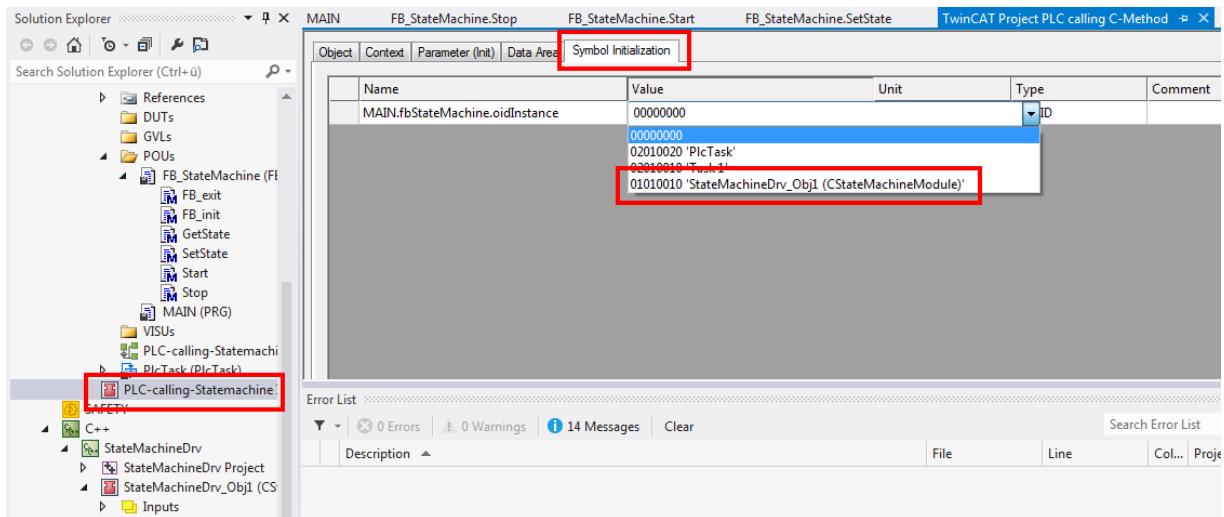
## Schritt 9: SPS FB mit C++ Instanz verknüpfen

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.

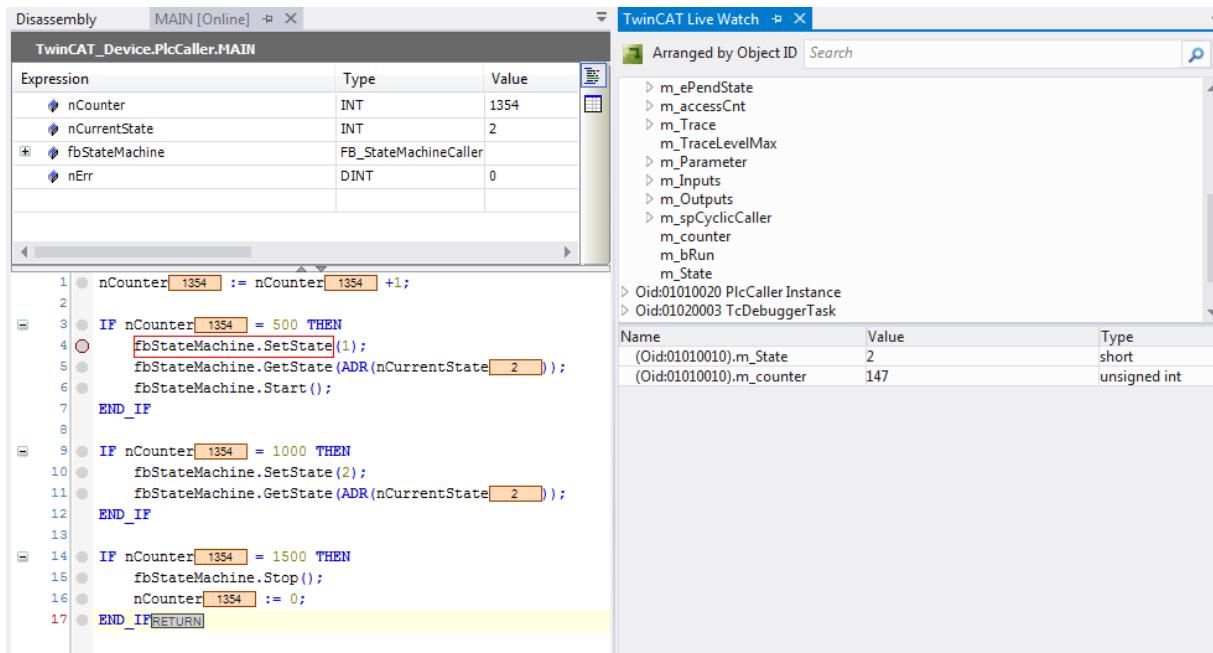
2. Auswahl des Dropdownfelds „Value“ und Auswahl der C++ Modulinstantanz, die mit der FB Instanz zu verknüpfen ist.



### 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:

3. Nach dem „Login“ und „Start“ des SPS-Projekts, befindet sich der Editor bereits im online-Modus (linke Seite der folgenden Abbildung).
4. Um auf online-Variablen des C++ Moduls zugreifen zu können, aktivieren Sie bitte das C++ Debugging [▶ 60] und folgenden Schritten des Schnellstarts [▶ 49], um das Debugging zu starten (rechte Seite der folgenden Abbildung).



## 15.11 Beispiel11a: Modulkommunikation: C-Modul ruft eine Methode eines anderem C-Moduls auf

Dieser Artikel beschreibt, wie TC3 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 Beispiel.](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

## Beschreibung

Das Projekt enthält drei Module:

- Die Instanz der CModuleDataProvider Klasse hostet die Daten und schützt vor dem Zugriff mit Hilfe der „Retrieve“ und „Store“ Methoden durch eine Critical Section.
- Die Instanz der Modulklasse „CModuleDataRead“ liest die Daten aus dem DataProvider über den Aufruf der Retrieve Methode.
- 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 „DataWriteCounterModulo“ 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.

## 15.12 Beispiel12: Modulkommunikation: Verwendet IO Mapping

Dieser Artikel beschreibt, wie zwei TC3 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 Beispiel.](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

## 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 \[▶ 62\]](#))

Letztendlich kann der Nutzer den Eingangs-„Value“ der „Front“ Instanz festlegen und den Ausgangs-„Value“ von "Back" beobachten.

## 15.13 Beispiel13: Modulkommunikation: C-Modul ruft SPS-Methoden auf

Dieser Artikel beschreibt, wie ein TwinCAT C++ Modul per TcCOM Interface Methoden eines Funktionsbausteins der SPS aufruft.

### Download

#### **HINWEIS! Systemvoraussetzung: TwinCAT 3.1 Build 4020**

Hier erhalten Sie den [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### Beschreibung

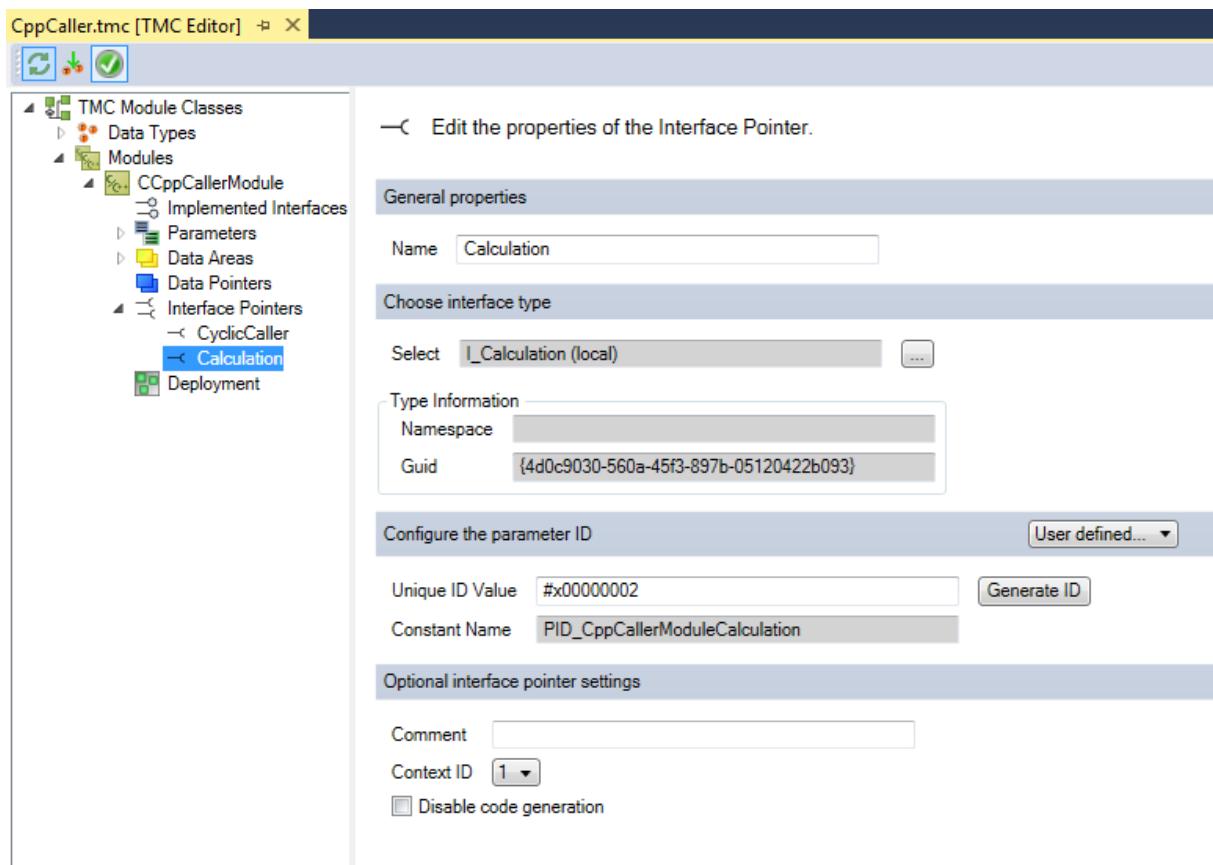
Dieses Beispiel stellt die Kommunikation von einem C++ Modul zu einem Funktionsbaustein einer SPS mittels Methodenaufruf dar. Hierfür wird ein TcCOM-Interface definiert, welches von der SPS angeboten wird und von dem C++ Modul genutzt wird.

Die SPS-Seite als Provider entspricht dabei dem entsprechenden Projekt des Beispiels [TcCOM Sample 01 ▶ 285](#), wo eine SPS nach SPS Kommunikation betrachtet wird. Hier wird nun ein „Caller“ in C++ bereitgestellt, der das gleiche Interface nutzt.

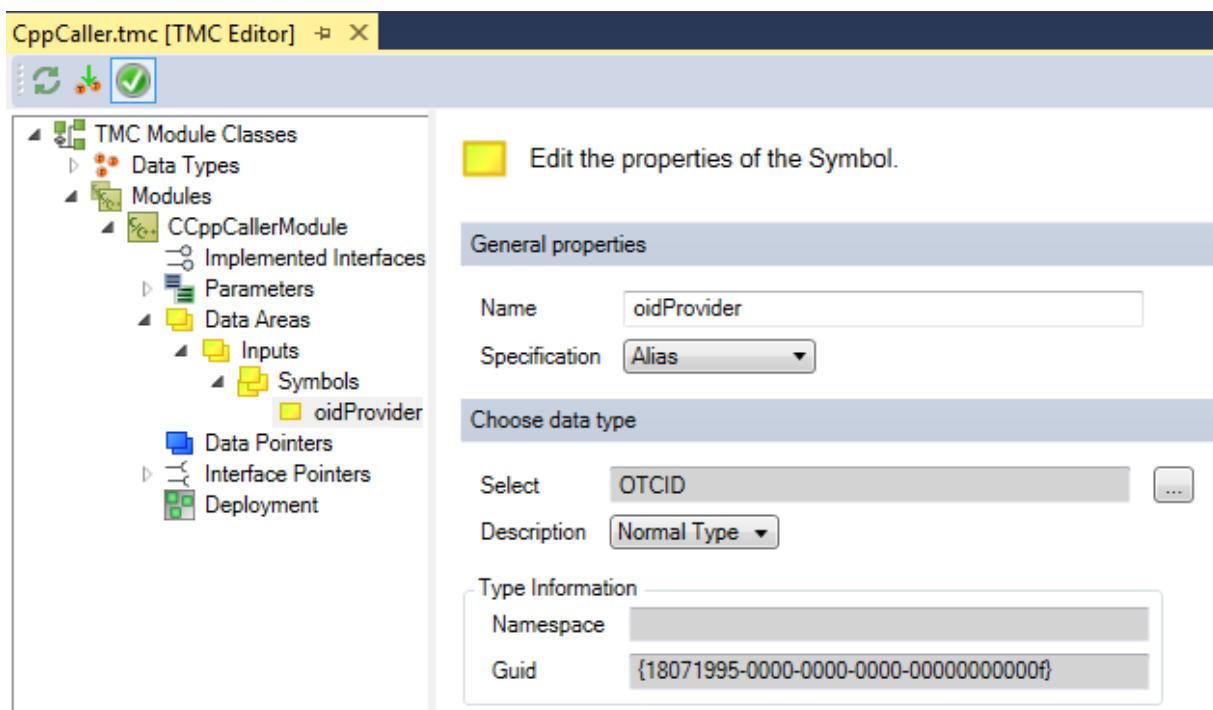
Die SPS-Seite wird von [TcCOM Sample 01 ▶ 285](#) übernommen. Der dort als TcCOM-Modul registrierte Funktionsbaustein bietet die ihm zugewiesene Objekt-ID als Ausgangsvariable an.  
Aufgabe des C++ Moduls ist es, das angebotene Interface dieses Funktionsbausteins zugreifbar zu machen.

- ✓ Es wird von einem C++ Projekt mit einem „Cycle IO“ Modul ausgegangen.

6. Im TMC Editor wird ein Interface-Pointer (Name: „Calculation“) vom Typ „I\_Calculation“ angelegt. Über diesen erfolgt später der Zugriff.



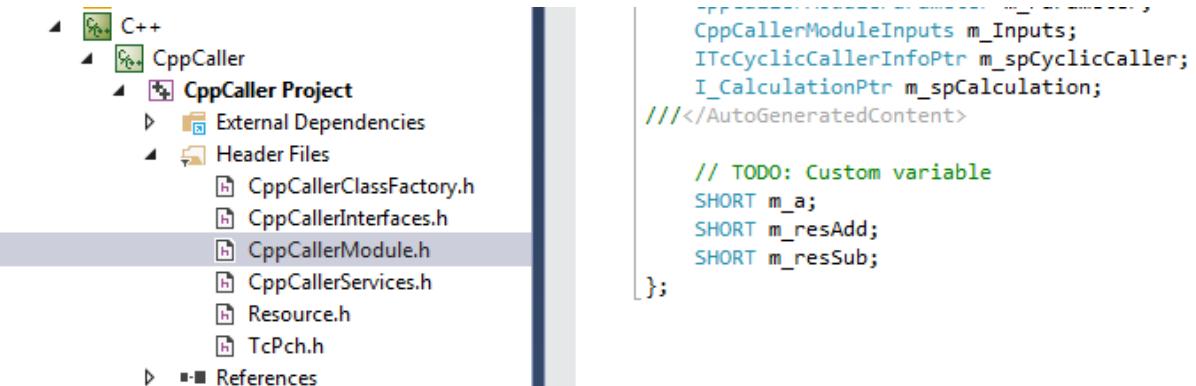
7. Zusätzlich wird im TMC Editor ein Eingang (Name: „oidProvider“) vom Typ „OTCID“ in der „Data Area“ „Inputs“, die mit Typ Input-Destination vom Modul-Wizard schon erstellt wurde, angelegt. Über diesen wird später die Objekt-ID aus der SPS verknüpft.



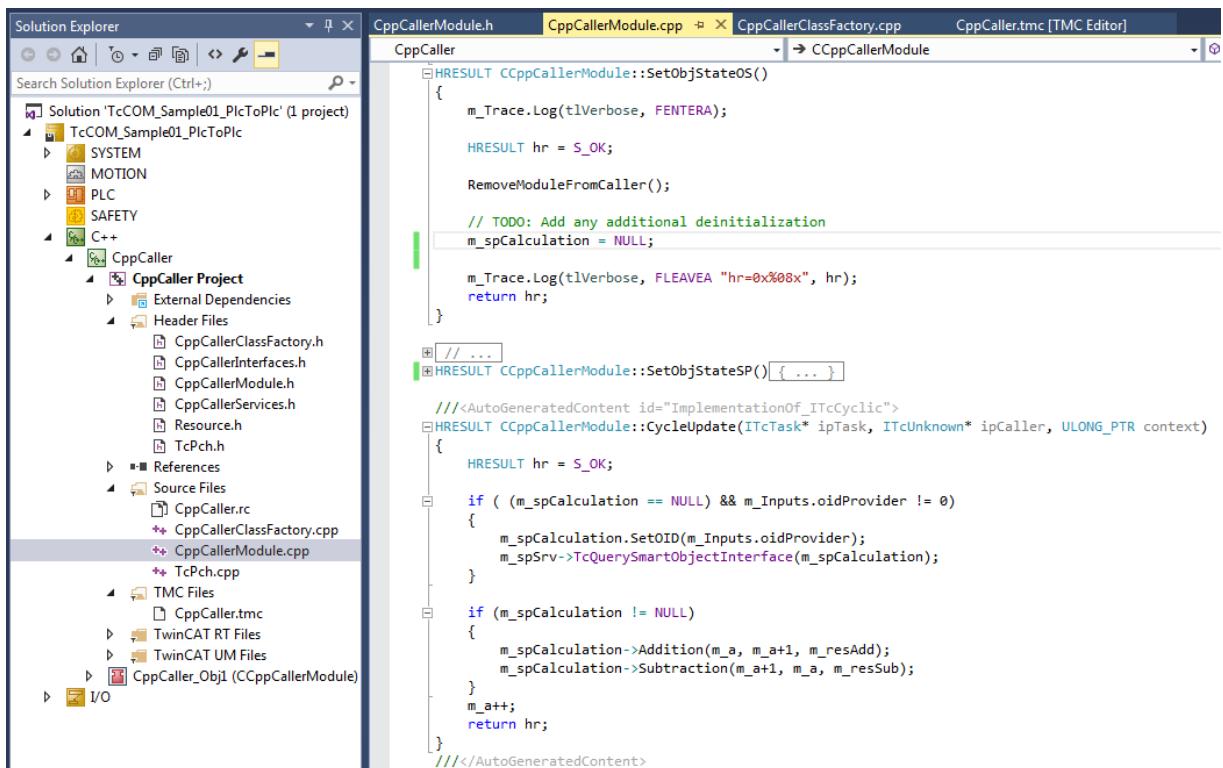
8. Alle weiteren Symbole sind für das Beispiel nicht relevant und können gelöscht werden.

9. Der TMC-Code-Generator bereitet den Code entsprechend vor.

10. In dem Header des Moduls werden einige Variablen angelegt, um die Methoden-Aufrufe später durchzuführen.



11. 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.

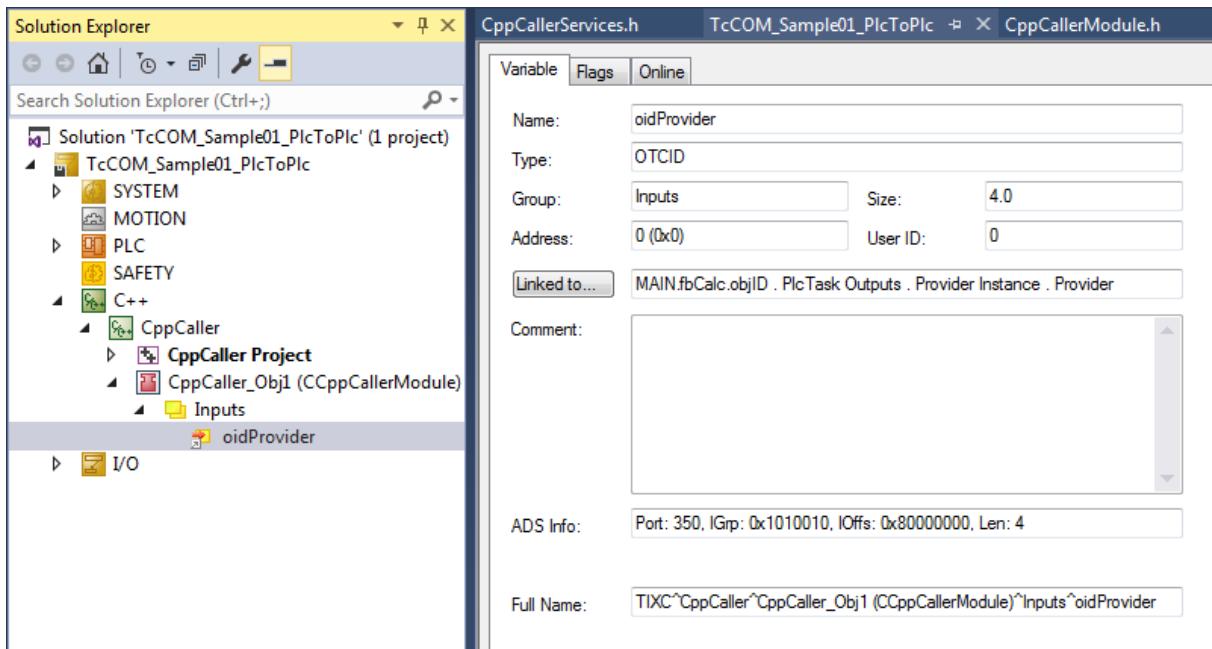


12. Zusätzlich muss, wie oben zu sehen ist, der Interface-Pointer beim Herunterfahren aufgeräumt werden. Dies geschieht in der SetObjStateOS Methode.

13. Das C++ Projekt wird nun einmal gebaut.

14. Es wird eine Instanz des Moduls angelegt.

15. Der Eingang des C++ Moduls muss mit dem Ausgang der SPS verbunden werden.



⇒ 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.

## 15.14 Beispiel19: Synchroner Dateizugriff

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, das auf Dateien auf der Festplatte während des Starts eines Moduls, also in Echtzeitumgebung, zugreifen kann.

### Download

#### Erhalten Sie den [Quellcode für dieses Beispiel](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

Der gesamte Quellcode, der vom Assistenten nicht automatisch generiert wird, wird mit dem Kommentarbeginn-Flag „//sample code“ und dem Kommentarend-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 von diesem Modul „TcFileTestDrv\_Obj1“.

Bei diesem Beispiel findet der Dateizugriff beim Übergang „PREOP to SAFEOP“ statt, also in der „SetObjStatePS()“ Methode.

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.x\Target\DefaultConfig.xml“) auf „%TC\_TARGETPATH %DefaultConfig.xml.bak“ kopiert.

Zwecks 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.

## 15.15 Beispiel20: FileIO-Write

Dieser Artikel beschreibt die Implementierung von TC3 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 Beispiel](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### Beschreibung

Das Beispiel beinhaltet eine Instanz von „TcAsyncWritingModule“, die Daten in die Datei „AsyncTest.txt“ im Verzeichnis BOOTPRJPATH (normalerweise C:\TwinCAT\3.x\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.

## 15.16 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 TC3-C++ Modul aus.

### Download

[Hier erhalten Sie den Quellcode für dieses Beispiel](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

## 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 (normalerweise C:\TwinCAT\3.x\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.

Bitte die „nBytesWritten“ / „nBytesRead“ Ausgangsvariablen des WritingModules / ReadingModules beobachten. 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 (Standardwert: „c:\TwinCAT\3.1\Boot\\*“).

Bitte 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 bitte mit den TcFsmFileReader und TcFsmFileWriter Klassen.

Sollten zu viele Verlaufsverfolgungsmeldungen auftreten, die das Verständnis des Beispiels erschweren, deaktivieren Sie Module!

## Siehe auch

[Beispiel S19 \[▶ 272\]](#)

[Beispiel S20 \[▶ 273\]](#)

[Beispiel S25 \[▶ 278\]](#)

[Schnittstelle ITc FileAccess \[▶ 142\] / Schnittstelle ITc FileAccess Async \[▶ 151\]](#)

## Sehen Sie dazu auch

- „ Beispiel19: Synchroner Dateizugriff [▶ 272]
- „ Beispiel20: FileIO-Write [▶ 273]
- „ Beispiel25: Statische Bibliothek [▶ 278]

## 15.17 Beispiel22: Automation Device Driver (ADD): Zugang DPRAM

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Treibers, der als TwinCAT Automation Device Driver (ADD) mit Zugriff auf DPRAM fungiert.

### Download

Hier erhalten Sie den [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren
6. **Konfigurationsdetails unten lesen** vor Aktivierung

### Beschreibung

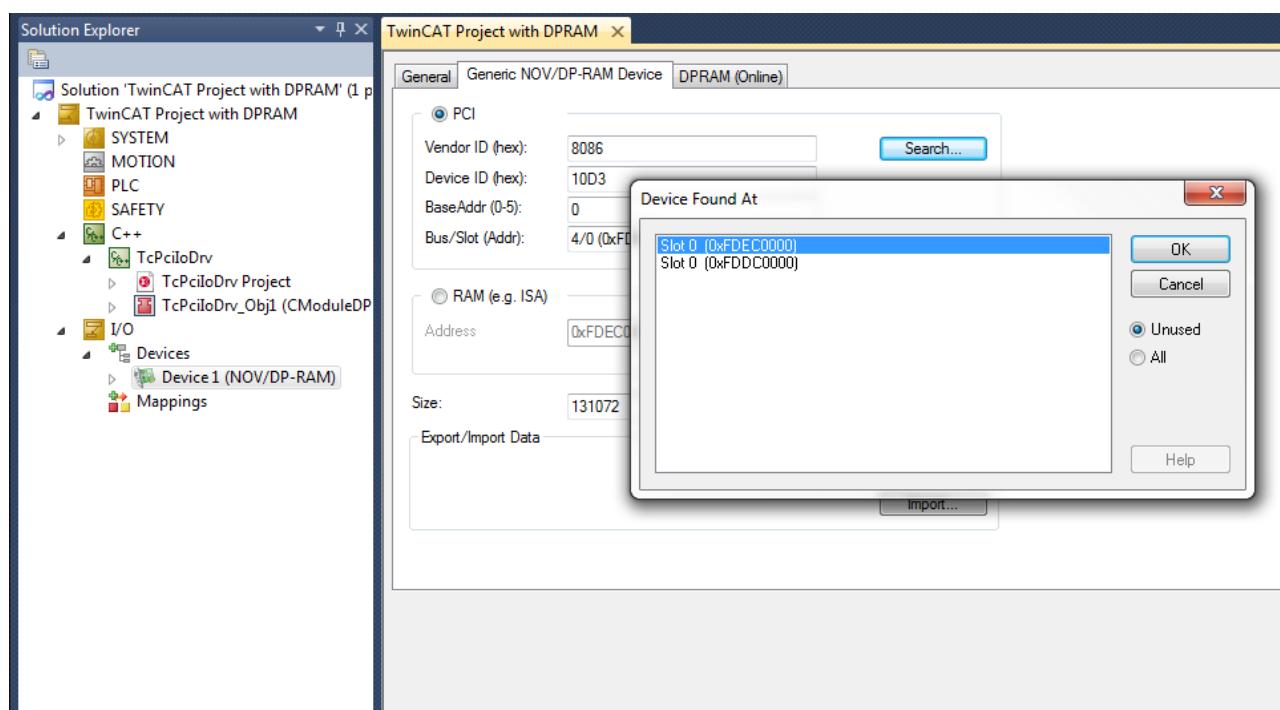
Dieses Beispiel soll den „Link Detect Bit“ des Netzwerkadapters (d.h. von einem CX5010) zyklisch ein- und ausschalten.

Das C++ Modul ist über den Schnittstellenzeiger „PciDeviceAdi“ des C++ Moduls mit dem NOV/DP-RAM Gerät verbunden.

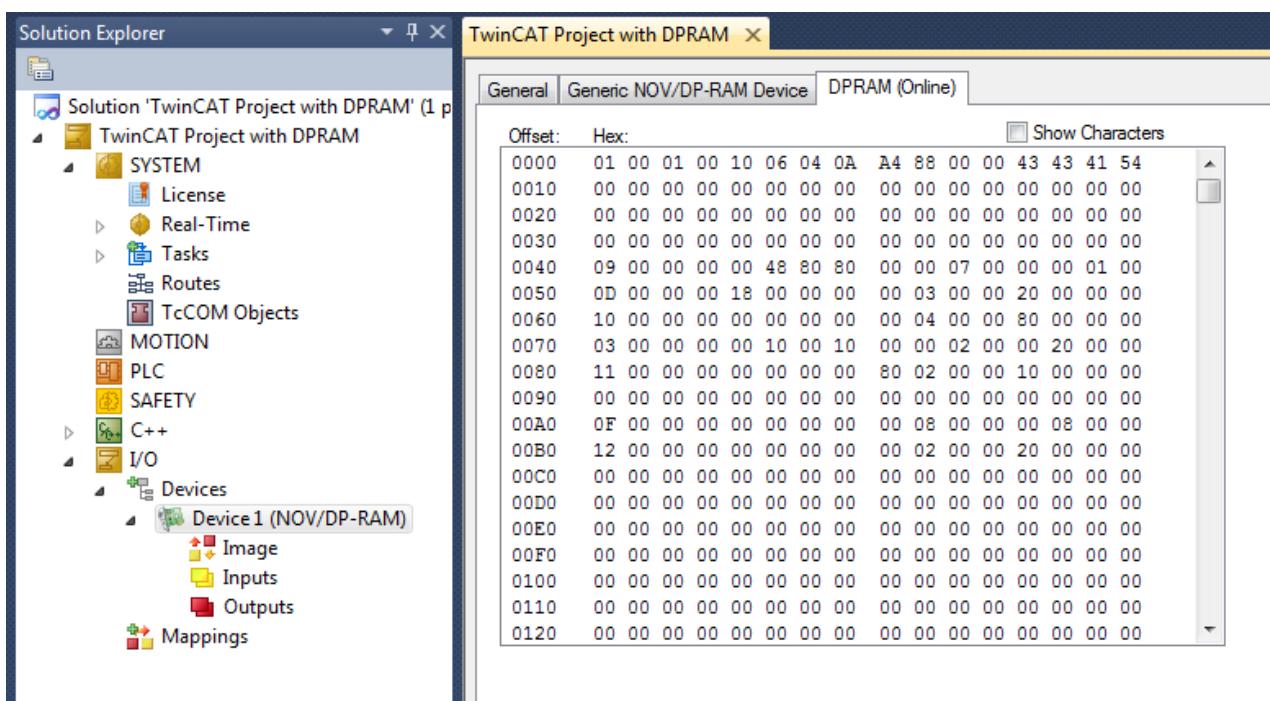
### Konfiguration

Damit das Beispiel funktioniert, müssen Sie die Hardware-Adressen passend zu Ihrer persönlichen Hardware konfigurieren.

Überprüfen Sie bitte die PCI Konfiguration:



Um zu überprüfen, ob die Kommunikation mit NOV/DPRAM korrekt eingerichtet ist, kann die DPRAM (Online) Ansicht verwendet werden:



## 15.18 Beispiel23: Strukturierte Ausnahmebehandlung (SEH)

Dieser Artikel beschreibt die Verwendung von „Structured Exception Handling“ (SEH) anhand von fünf Varianten.

### Download

[Hier erhalten Sie den Quellcode für dieses Beispiel.](#)

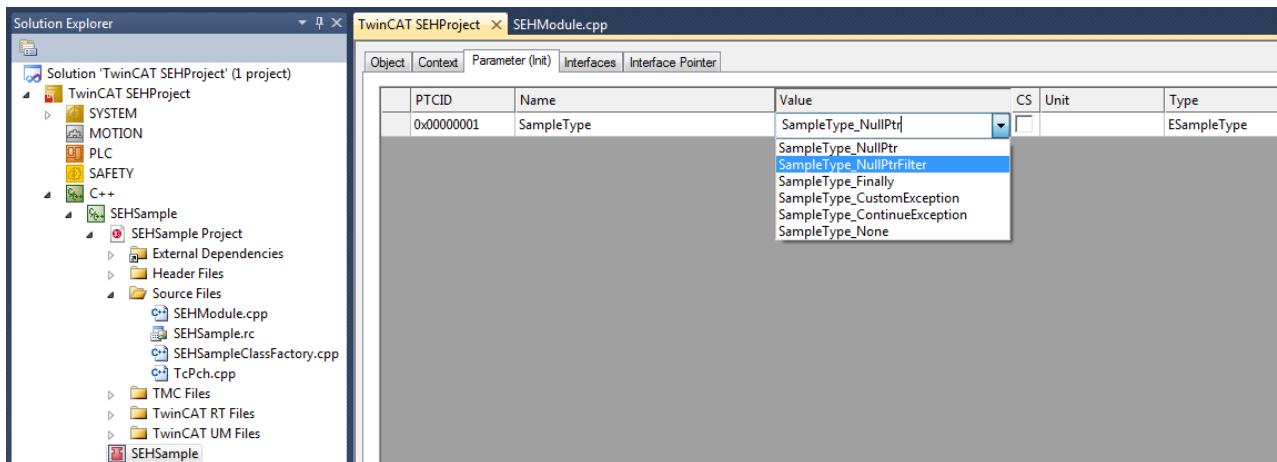
1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tzip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### Beschreibung

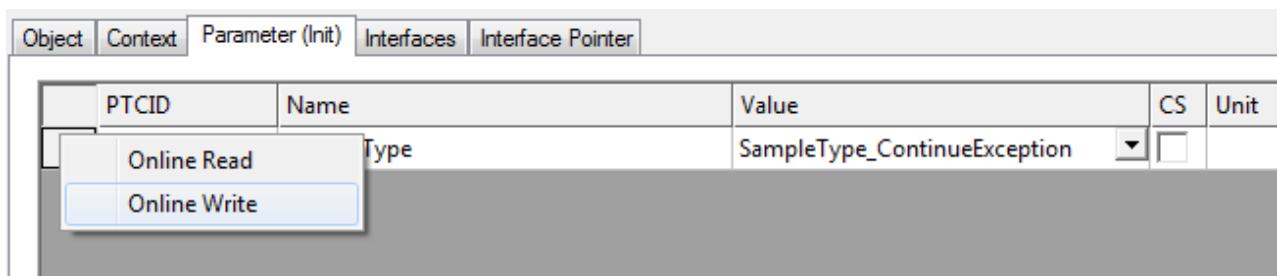
Das Beispiel beinhaltet fünf Varianten, die die Verwendung von SEH im TwinCAT C++ demonstrieren:

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 kann mittels Rechts-Klick auf die vorderste Spalte auch zur Laufzeit der Wert geschrieben werden:



Alle Varianten schreiben zur Verdeutlichung des Verhaltens durch den Trace Mechanismus, so dass Meldungen im TwinCAT Engineering erscheinen:

Error List	
13 Errors   0 Warnings   456 Messages   Clear	
	Description
(i) 33	20.11.2015 11:02:23 621 ms   'TwinCAT System' (10000): Starting COM Server TcOpcUaServer !
(i) 34	20.11.2015 11:02:24 532 ms   'TCOM Server' (10): SEHSample: Simple Exception handling in cycle 95
(i) 35	20.11.2015 11:02:25 482 ms   'TCOM Server' (10): SEHSample: Simple Exception handling in cycle 190
(i) 36	20.11.2015 11:02:26 432 ms   'TCOM Server' (10): SEHSample: Simple Exception handling in cycle 285
(i) 37	20.11.2015 11:02:27 382 ms   'TCOM Server' (10): SEHSample: Simple Exception handling in cycle 380
(i) 38	20.11.2015 11:02:28 332 ms   'TCOM Server' (10): SEHSample: Simple Exception handling in cycle 475
(i) 39	20.11.2015 11:02:29 282 ms   'TCOM Server' (10): SEHSample: Simple Exception handling in cycle 570

## Startup

Das Beispiel benötigt nach dem Entpacken keine Konfiguration.

## 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 auch 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  
Ebenfalls 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  
Ebenfalls wieder ein NULL-Pointer Zugriff mit TcExcept{}, diesmal wird die Exception jedoch nach der Behandlung in der FilterException() Methode weiter gereicht, so dass auch der weitere TcExcept{} die Exception behandelt.

## 15.19 Beispiel25: Statische Bibliothek

Dieser Artikel beschreibt die Implementierung und die Verwendung eines Moduls einer statischen TC3 C++ Bibliothek.

### Download

[Hier erhalten Sie den Quellcode für dieses Beispiel.](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### Beschreibung

Das Beispiel beinhaltet zwei Projekte: Das Projekt „DriverUsingStaticLib“ verwendet den statischen Inhalt des Projekts „StaticLib“.

**StaticLib:** Auf der einen Seite bietet „StaticLib“ eine Funktion „ComputeSomething“ in der StaticFunction.h/.cpp.

Auf der anderen Seite wird eine 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.

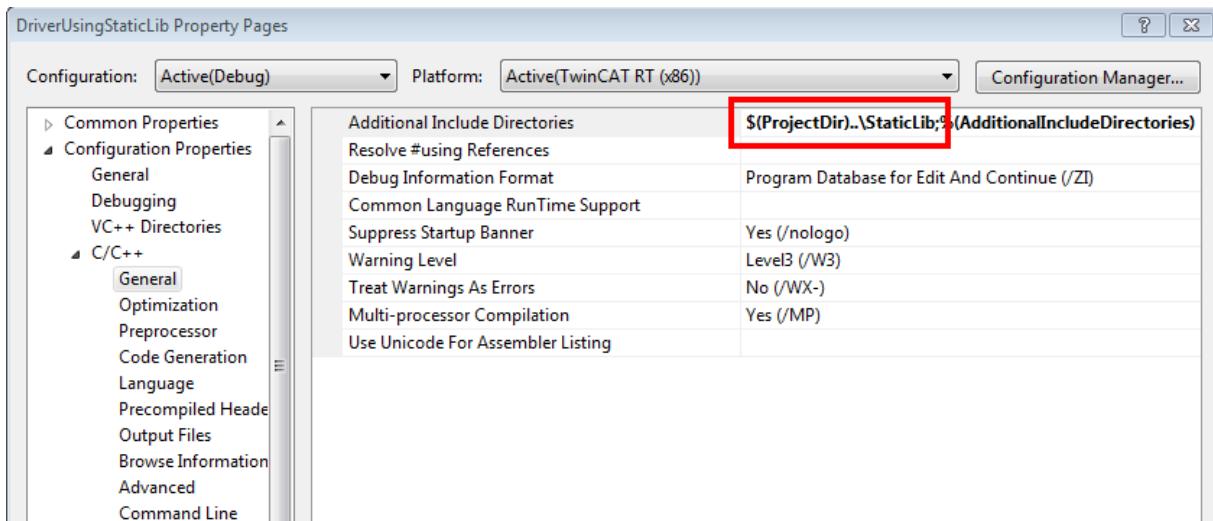
### Das Beispiel verstehen

Durchlaufen Sie bitte folgende Schritte, um eine statische Bibliothek zu erstellen und zu verwenden.

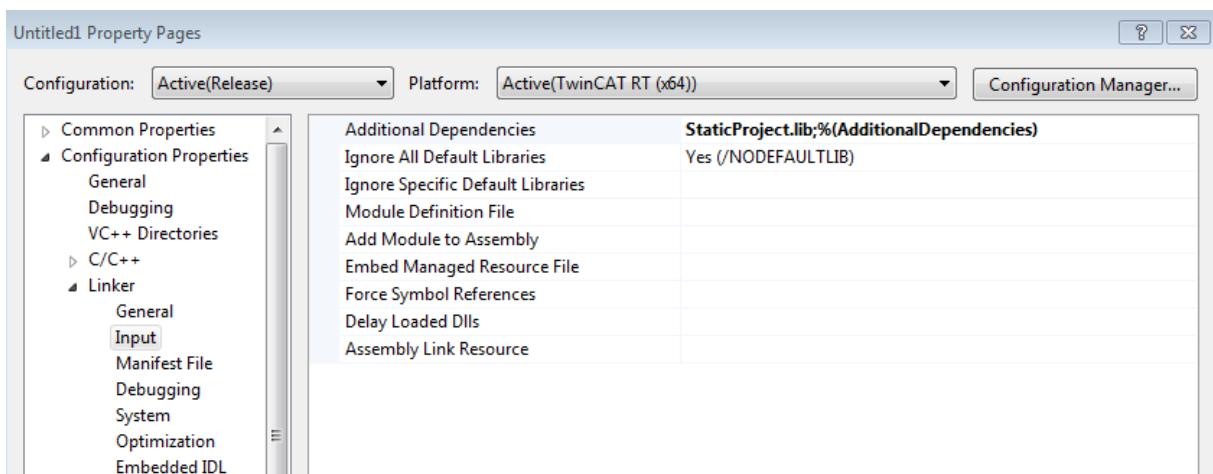
 <b>Hinweis</b>	<b>Manuelle Neukompilierung</b> Beachten Sie, dass Visual Studio die statische Bibliothek bei der Erstellung des Treibers nicht automatisch erneut kompiliert. Führen Sie das bitte manuell aus.
---	---

- ✓ Nutzen Sie bei der Erstellung eines C++ Projekts bitte die „TwinCAT Static Library Project“ Vorlage für die Erstellung einer statischen Bibliothek.
- ✓ ** HINWEIS! Nutzen Sie für die folgenden Schritte den „Edit“-Dialog vom VisualStudio, so dass nachher %(AdditionalIncludeDirectories) bzw. %(AdditionalDependencies) verwendet wird.**

6. Fügen Sie im Treiber dem Compiler das Verzeichnis der statischen Bibliothek unter „Additional Include Directories“ hinzu.



7. Fügen Sie bitte im Treiber, der die statische Bibliothek nutzt, diese als eine zusätzliche Abhängigkeit für den Linker hinzu. Öffnen Sie bitte die Projekteigenschaften des Treibers und fügen die statische Bibliothek hinzu:



## 15.20 Beispiel26: Ausführungsreihenfolge in einem Task

Dieser Artikel beschreibt die Bestimmung der Taskausführungsreihenfolge, wenn einem Task mehr als ein Modul zugeordnet ist.

### Download

Hier erhalten Sie den [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### Beschreibung

Das Beispiel enthält ein Modul „SortOrderModule“ das zwei Mal instanziert wird. Die „Sortierreihenfolge“ bestimmt die Ausführungsreihenfolge, die über den [TwinCAT Module Instance Configurator \[► 123\]](#) konfiguriert werden kann.

Zum Beispiel verfolgt die „CycleUpdate“ Methode den Objekt-Namen und die Objekt-ID zusammen mit der Sortierreihenfolge dieses Moduls. Auf dem Konsolenfenster kann man die Ausführungsreihenfolge sehen:

```
(i) 32 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
(i) 33 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
(i) 34 05.09.2014 13:01:15 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
(i) 35 05.09.2014 13:01:15 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
(i) 36 05.09.2014 13:01:16 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
(i) 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 einem Task zugeordnet sind.

## Startup

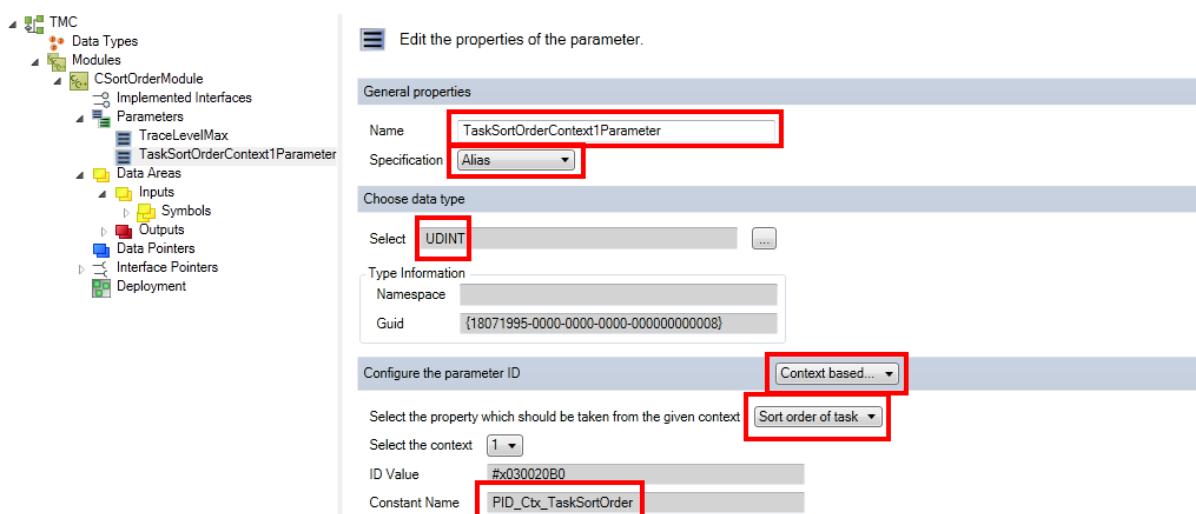
Das Beispiel benötigt nach dem Entpacken keine Konfiguration.

## Das Beispiel verstehen

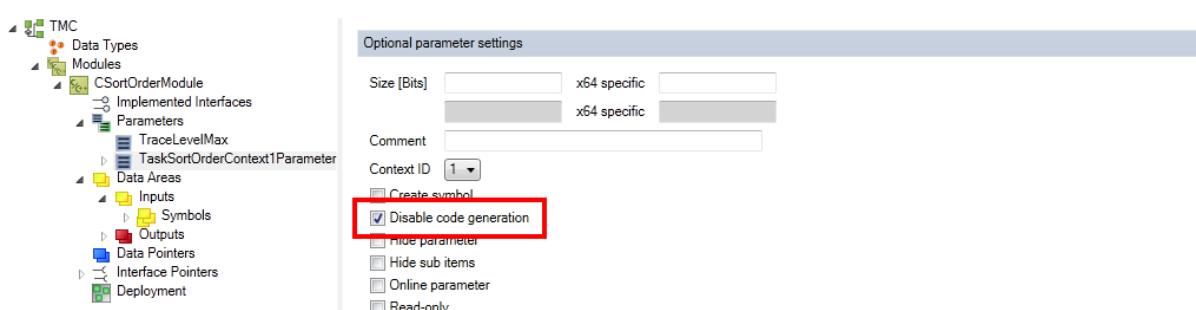
**Annahme:** Ein TcCOM C++ Modul mit zyklischem IO.

6. 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:



7. Bitte starten Sie den TMC Code Generator um die Standardimplementierung zu erhalten.
8. Da der Code im nächsten Schritt geändert werden wird, deaktivieren Sie bitte die Kodegenerierung für diesen Parameter jetzt.



9. Vergewissern Sie sich, dass Sie die Änderungen übernehmen, bevor Sie den TMC Code Generator erneut starten:  
Werfen Sie bitte 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()
```

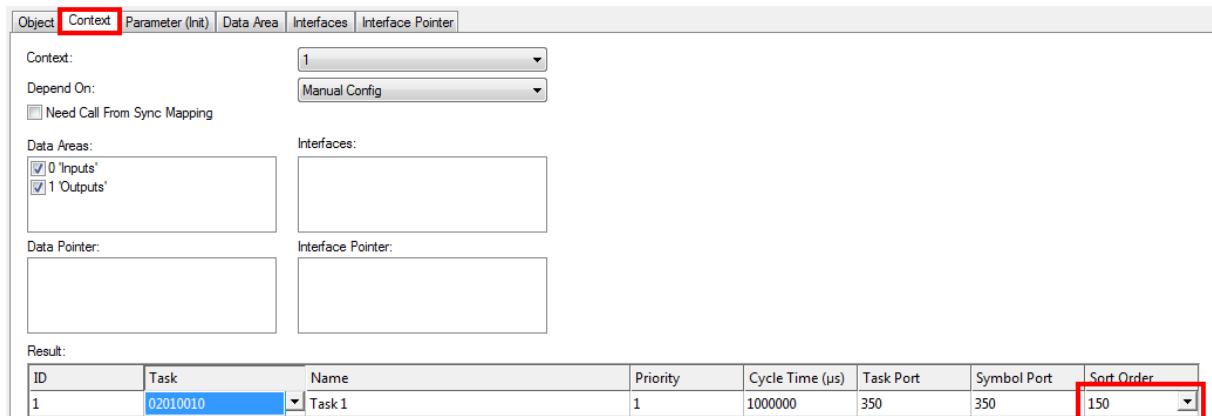
10. In diesem Beispiel werden Objekt-Name, -Id und SortOrder zyklisch verfolgt:

```
// 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->TcGetObjectNa-
me(), this->TcGetObjectId() , m_spCyclicCaller.GetInfo() ->sortOrder); //ADDED
```

11. Die Sortierreihenfolge kann auch als 4. Parameter der Methode ITcCyclicCaller::AddModule() übergeben werden, die in CModuleA::AddModuleToCaller() verwendet wird.

12. Den Instanzen dieses Moduls bitte einen Task mit **langsamem Zyklusintervall** (z.B. 1000ms) zuweisen, um die Verfolgungsmeldungen an das TwinCAT Engineering System zu begrenzen.

13. Jeder Instanz eine andere „Sortierreihenfolge“ über den [TwinCAT Module Instance Configurator \[▶ 123\]](#) zuweisen:



Sehen Sie dazu auch

[TwinCAT Module Klassenassistent \[▶ 78\]](#)

## 15.21 Beispiel30: Zeitmessung

In diesem Artikel wird die Implementierung eines TC3 C++ Moduls, das Zeitmessungsfunktionalitäten beinhaltet, beschrieben.

**Download**

Hier erhalten Sie den [Quellcode für dieses Beispiel](#).

1. Die heruntergeladene ZIP-Datei entpacken

2. Die enthaltene tzip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### 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 \[▶ 166\]](#)

## 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 Beispiel.](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tzip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### 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 zu „Cyclic IO“ Modul, 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: Abzuwartende Zeit, bevor Q gesetzt wird

Und zwei „out“ Parameter:

- Q: ist TRUE, wies PT Sekunden nach IN eine steigende Flanke auf
- ET: abgelaufene Zeit

Darüber hinaus muss ITcTask zur Abfrage der Zeitbasis bereitgestellt werden.

#### Siehe auch

[Beispiel30: Zeitmessung \[► 281\]](#)

[ITcTask Schnittstelle \[► 166\]](#)

## 15.23 Beispiel35: Ethernet Zugriff

Dieser Artikel beschreibt die Implementierung von TC3 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, so dass auf Basis dieses Beispiels eine Implementierung eines Netzwerkstacks nicht notwendig ist.

#### Download

[Hier erhalten Sie den Quellcode für dieses Beispiel.](#)

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren
6. Folgen Sie „Konfiguration“ unten.

#### Beschreibung

Das Beispiel beinhaltet eine Instanz des „TcEthernetSample“ Moduls, 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.

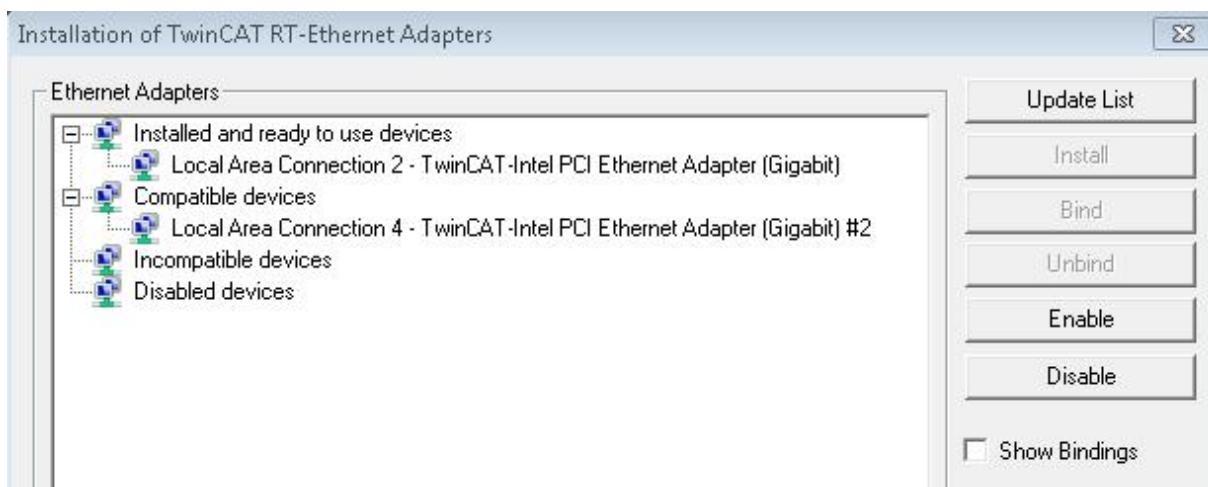
Das Beispiel verwendet zwei Ethernet-Komponenten von TwinCAT:

1. Ein **ITcEthernetAdapter** (Instanzname in 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 **ITcIoEthProtocol** 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 Netzwerkumgebung konfiguriert sein. Bitte führen Sie die folgenden Schritte aus:

7. Das Beispiel verlangt, dass die Ethernet-Karte den TwinCAT Treiber verwendet.  
 Starten Sie „TcRteInstall.exe“ entweder vom XAE über das Menü TwinCAT->“Show Realtime Ethernet compatible devices...“ oder von der Festplatte auf den XAR Systemen.



8. Möglicherweise müssen Sie den Treiber mit Hilfe der Schaltflächen installieren und aktivieren.  
 9. TwinCAT muss wissen, welche Ethernet-Karte verwendet werden sollte. Öffnen Sie das Projekt in XAE und I/O / Devices / Device 1 (RT-Ethernet Adapter) auswählen.  
 10. Die zweite Registerkarte „Adapter“ auswählen und mit „Search“ den Adapter auswählen.  
 11. TcEthernetSample\_Obj1 muss konfiguriert werden. Das Instanzenfenster öffnen und die folgenden Werte festlegen:  
 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.

## 15.24 Beispiel37: Daten archivieren

Das Beispiel TcCOM Object Archiv beschreibt das Wiederherstellen und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung.

**■ HINWEIS! TwinCAT unterstützt auch Retain Daten um das NOVRAM eines Gerätes zur Persistierung von Daten zu nutzen.**

### Download

**Erhalten Sie den Quellcode für dieses Beispiel.**

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene tszip-Datei in TwinCAT 3 mittels „Open Project ...“ öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
5. Die Konfiguration aktivieren

### 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 von 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 von 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.

## 15.25 TcCOM Beispiele

Module zwischen PLC und C++ können kommunizieren, so dass sowohl auf Seiten der PLC beschrieben wird, wie mit C++ Modulen umgegangen werden kann, wie auch auf C++ Seite, wie mit der PLC umgegangen werden kann.

An dieser Stelle sind die TcCOM Beispiele zur Kommunikation mit der PLC dargestellt.

Im [Beispiel TcCOM Sample01 \[▶ 285\]](#) wird dargestellt, wie eine TcCOM-Kommunikation zwischen zwei SPSn stattfinden kann. Dabei werden aus der einen SPS heraus Funktionalitäten der anderen SPS direkt aufgerufen.

Im [Beispiel TcCOM Sample02 \[▶ 295\]](#) 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++ Treibers 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.

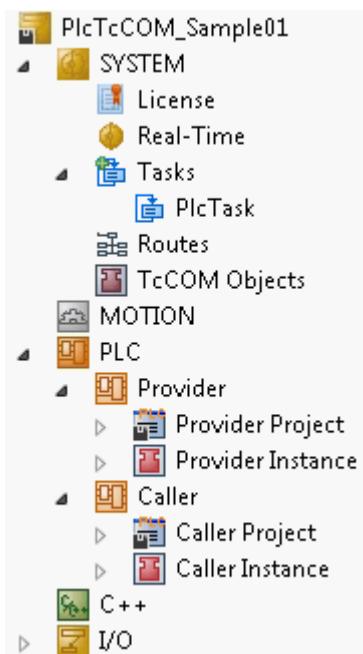
Im [Beispiel TcCOM Sample03 \[▶ 300\]](#) wird dargestellt, wie eine SPS-Applikation Funktionalitäten einer TwinCAT C++ Klasse nutzt, indem zugleich eine Instanz der C++ Klasse erzeugt wird. Dies kann im Vergleich zum vorherigen Sample eine erhöhte Flexibilität bieten.

### 15.25.1 TcCOM\_Sample01\_PlctoPlc

Dieses Beispiel beschreibt eine TcCOM-Kommunikation zwischen zwei SPSn.

Funktionalitäten, welche 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, welche sich in der ersten SPS befindet, gearbeitet.

Die zwei SPSn 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:

1. Erstellen eines FBs in der ersten SPS, welcher seine Funktionalität global bereitstellt [▶ 286]
2. Erstellen eines FBs in der zweiten SPS, welcher als einfacher Proxy diese Funktionalität dort ebenfalls anbietet [▶ 291]
3. Ausführung des Beispielprojektes [▶ 294]

Download des Beispiels: TcCOM\_Sample01\_PlctoPlc.zip ([https://infosys.beckhoff.com/content/1031/TC3\\_C/Resources/zip/2343046667.zip](https://infosys.beckhoff.com/content/1031/TC3_C/Resources/zip/2343046667.zip))

 <b>VORSICHT</b>	<b>Race Conditions bei Multi-Tasking (Multi-Threading) Verwendung</b> <p>Der Funktionsbaustein, welcher seine Funktionalität global zur Verfügung stellt, wird in der ersten SPS instanziert. Dort kann er wie jeder Funktionsbaustein verwendet werden. Wenn er zudem aus einer anderen SPS (oder bspw. einem C++ Modul) verwendet wird, muss darauf geachtet werden, 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, ist einem zeitgleichen Zugriff vorzubeugen. Hierzu sei auf die Funktion TestAndSet() aus der Tc2_System Bibliothek verwiesen.</p>
--	--

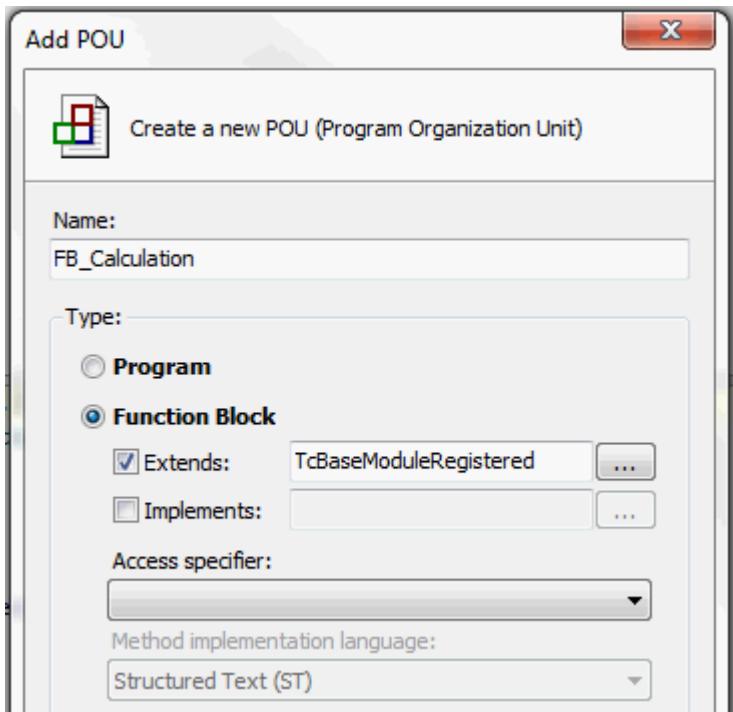
## Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64, ARM	Tc3_Module, Tc3_Interfaces

### 15.25.1.1 Erstellen eines FBs in der ersten SPS, welcher seine Funktionalität global bereitstellt

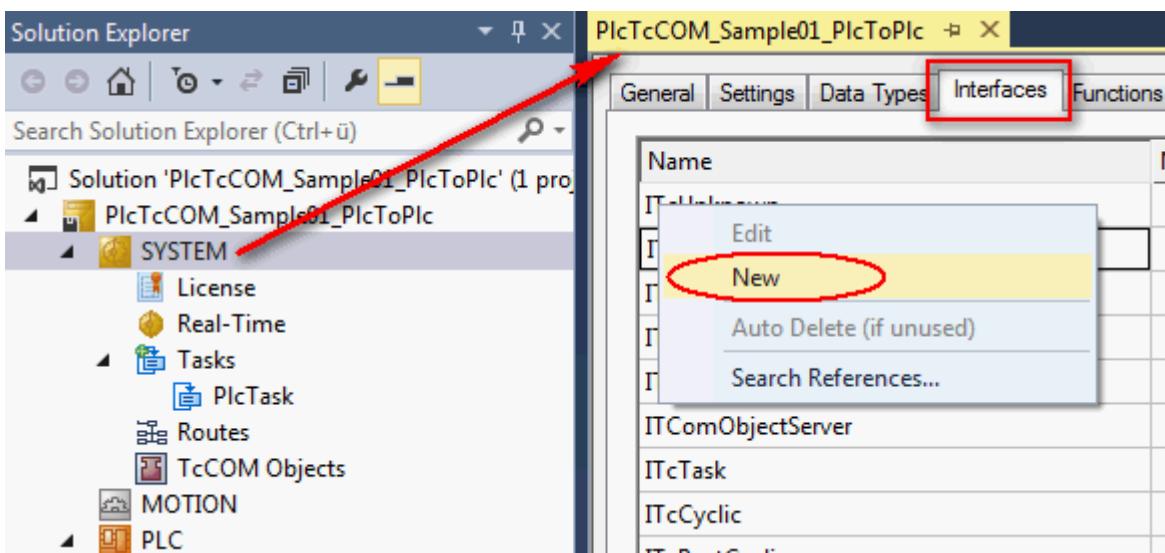
1. Legen Sie eine SPS an und erstellen Sie einen neuen Funktionsbaustein (FB) (hier: FB\_Calculation). Leiten Sie den Funktionsbaustein von der Klasse TcBaseModuleRegistered ab, damit eine Instanz dieses Funktionsbausteins nicht nur in der gleichen SPS verfügbar, sondern auch aus einer zweiten SPS heraus erreichbar ist.

**Hinweis:** Alternativ können Sie auch einen FB in einer bestehenden SPS modifizieren.

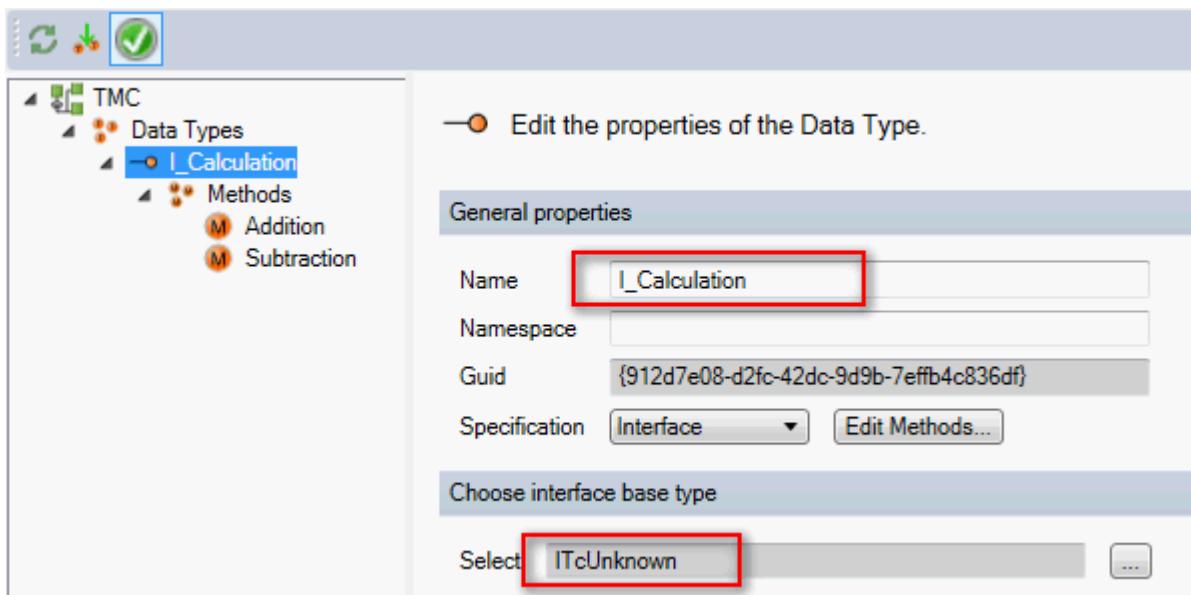


2. Der Funktionsbaustein muss seine Funktionalität mittels Methoden anbieten. Diese werden in einer globalen Schnittstelle definiert, deren Typ systemweit und programmiersprachenunabhängig bekannt ist. Um ein globales Interface anzulegen, öffnen Sie im Reiter „Interface“ der Systemeigenschaften das Kontextmenü und wählen Sie die Option „New“ aus.

⇒ Es öffnet sich der TMC Editor, welcher Sie darin unterstützt ein globales Interface anzulegen.

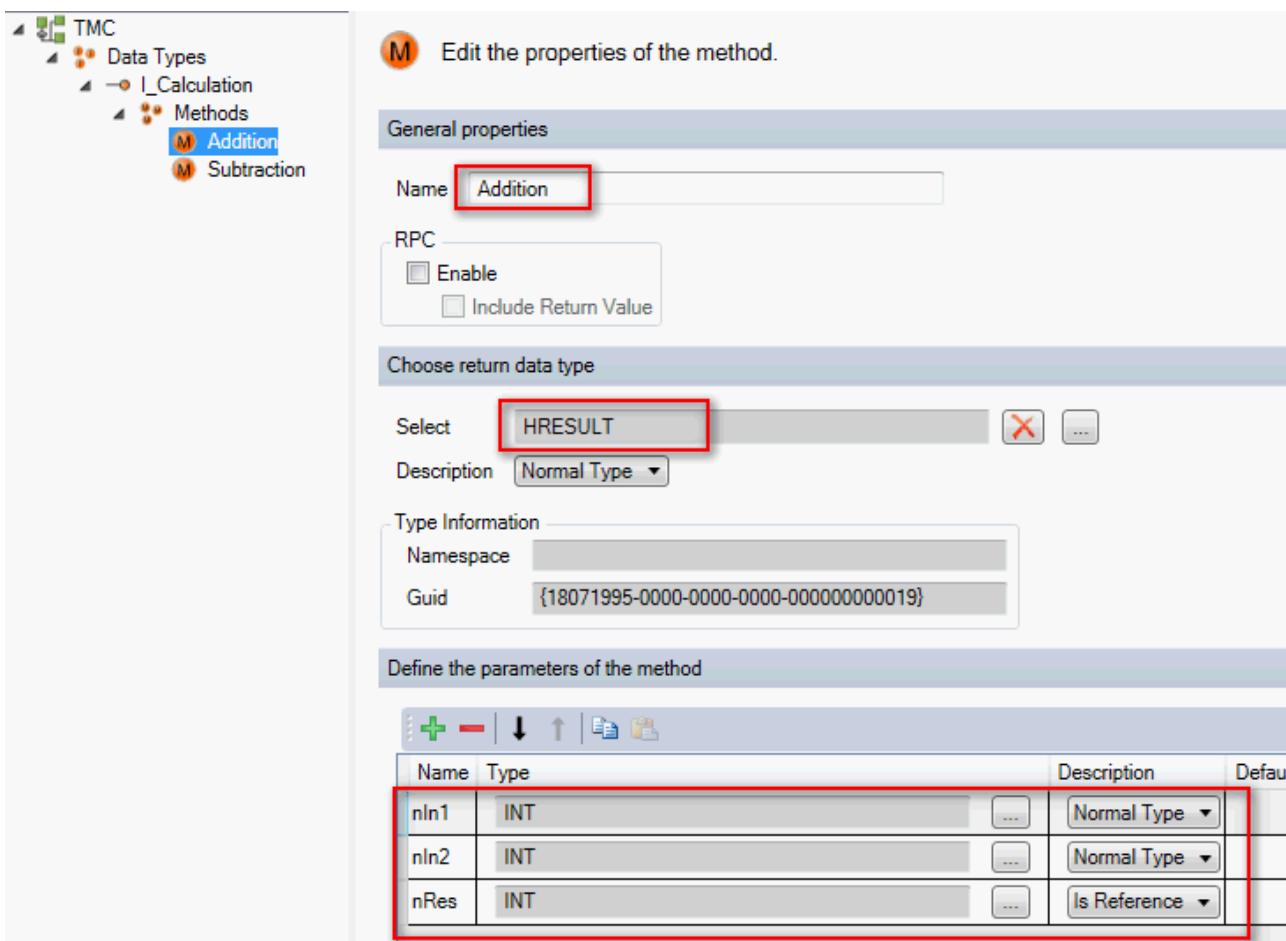


3. Spezifizieren Sie den Namen (hier: I\_Calculation) und fügen Sie die gewünschten Methoden an. Das Interface wird automatisch von ITcUnknown abgeleitet, um dem TwinCAT TcCOM-Modulkonzept gerecht zu werden.



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.



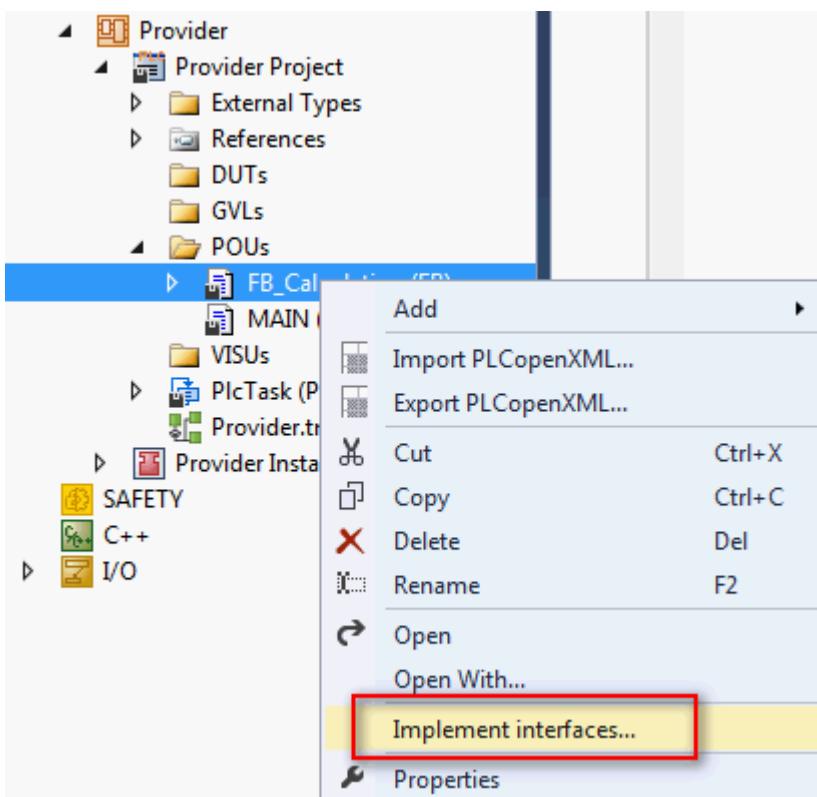
6. Implementieren Sie nun im Funktionsbaustein FB\_Calculation die Schnittstelle I\_Calculation und fügen Sie das Attribut c++\_compatible an.

```

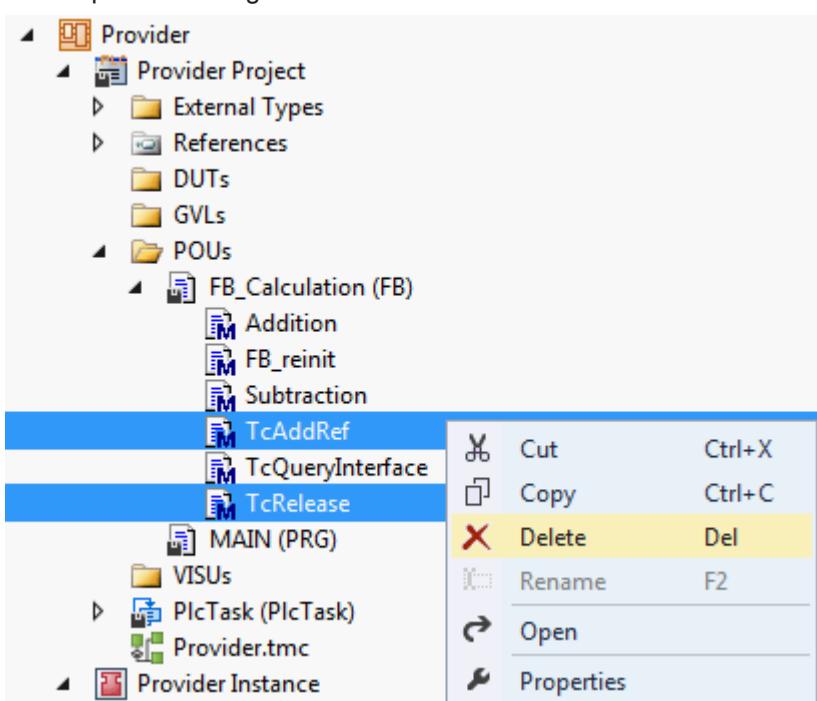
4 [attribute 'c++_compatible']
5 FUNCTION_BLOCK FB_Calculation EXTENDS TcBaseModuleRegistered IMPLEMENTS I_Calculation
6
7 VAR
8 END_VAR
9

```

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 ▶ X
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 [▶ 171]. 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.

```
FB_Calculation.TcQueryInterface ▶ X
1 {attribute 'object_name' := 'TcQueryInterface'}
2 {attribute 'c++_compatible'}
3 {attribute 'signature_flag' := '33554688'}
4 {attribute 'pack_mode' := '4'}
5 {attribute 'show'}
6 {attribute 'minimal_input_size' := '4'}
7 {attribute 'checksuperglobal'}
8 METHOD TcQueryInterface : HRESULT
9 VAR_INPUT
10     iid : REFERENCE TO IID;
11     pipItf : POINTER TO PVOID;
12 END_VAR
13
14 VAR
15     ipCalc : I_Calculation;
16 END_VAR
17
18 IF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_I_Calculation)) THEN
19     ipCalc := THIS^; // cast to interface pointer
20     pipItf^ := ITCUNKNOWN_TO_PVOID(ipCalc);
21     TcAddRef();
22     TcQueryInterface := S_OK;
23 ELSE
24     TcQueryInterface := SUPER^.TcQueryInterface(iid, pipItf);
25 END_IF
26
```

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.

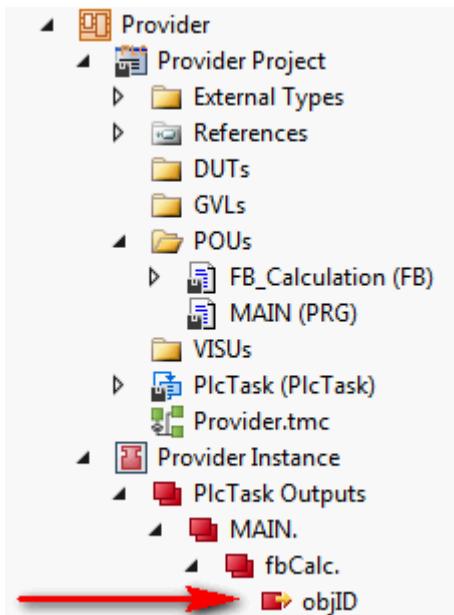
⇒ Die Implementierung in der ersten SPS ist vollständig.

```

MAIN* ▾ X
1 PROGRAM MAIN
2 VAR
3     m : UDINT;
4
5     fbCalc : FB_Calculation('MAIN.fbCalc');
6 END_VAR
7

```

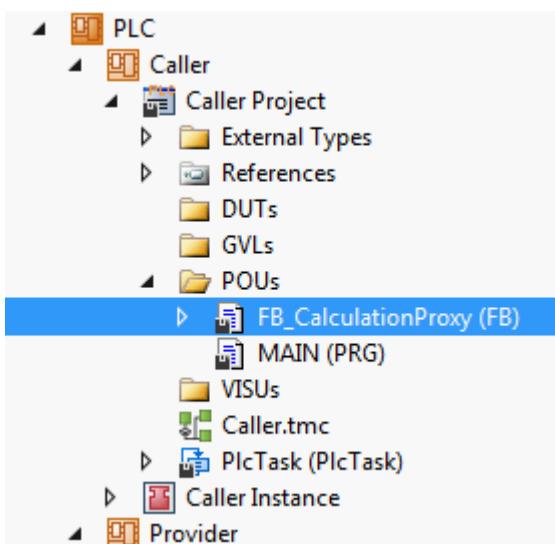
⇒ Nach dem Kompilieren der SPS ist im Prozessabbild die Objekt-ID des TcCOM-Objektes, welches die Instanz von FB\_Calculation representiert, als Ausgang verfügbar.



### 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.



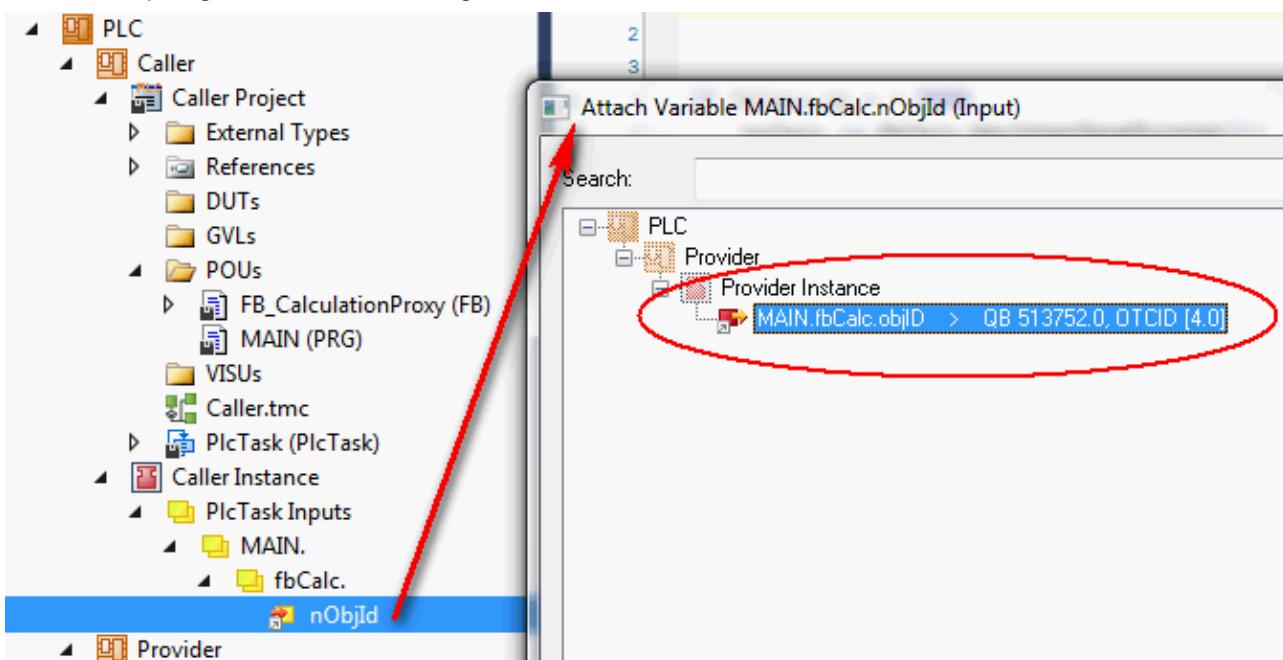
2. Deklarieren Sie im Deklarationsteil des Funktionsbausteins als Ausgang einen Schnittstellenzeiger auf die globale Schnittstelle, welche später die Funktionalität nach außen bereitstellt.

```

FB_CalculationProxy □ ✎ ×
1 | FUNCTION_BLOCK FB_CalculationProxy
2 | VAR_OUTPUT
3 |     ip : I_Calculation;
4 | END_VAR
5 |
6 | VAR
7 |     {attribute 'displaymode':='hex'}
8 |     nObjId AT%I* : OTCID;    // Instance configured to be retrieved
9 |     iid : IID := TC_GLOBAL_IID_LIST.IID_I_Calculation;
10| END_VAR
11|

```

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 # □ X
1 METHOD GetInterfacePointer : HRESULT
2 VAR
3 END_VAR
4
5 IF nObjID <> 0 THEN
6   IF (ip = 0) THEN // only get interface pointer if it is not already existing
7     GetInterfacePointer := FW_ObjMgr_GetObjectInstance(oid:=nObjID, iid:=iid, pipUnk:=ADR(ip));
8   ELSE
9     GetInterfacePointer := E_HRESULTAdsErr.EXISTS;
10  END_IF
11 ELSE
12  GetInterfacePointer := E_HRESULTAdsErr.INVALIDOBJID;
13 END_IF
14
```

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 # □ X FB_CalculationProxy.GetInterfacePointer # □ X
1 [attribute 'hide']
2 METHOD FB_exit : BOOL
3 VAR_INPUT
4   bInCopyCode : BOOL; // if TRUE, the exit method is called by a copy operation
5 END_VAR
6
7 IF NOT bInCopyCode THEN // if not online change
8   FW_SafeRelease(ADR(ip));
9 END_IF
10
```

⇒ 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.

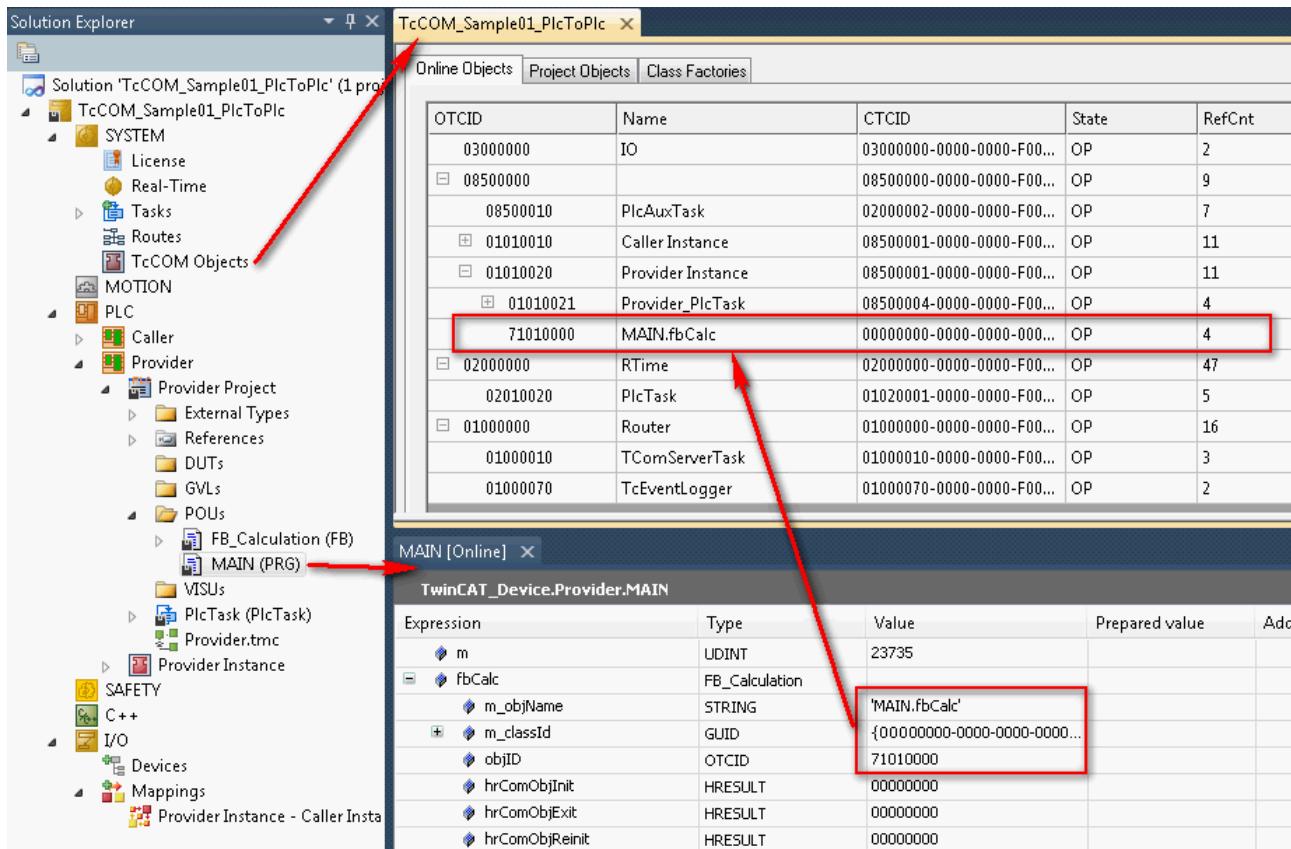
```
MAIN* # □ X
1 PROGRAM MAIN
2 VAR
3   fbCalc : FB_CalculationProxy;
4   hrCalc : HRESULT;
5   a : INT := 10;
6   b : INT := 7;
7   nSum : INT; // a + b
8   nDiff : INT; // a - b
9 END_VAR
10
11 IF fbCalc.ip = 0 THEN
12   hrCalc := fbCalc.GetInterfacePointer();
13 END_IF
14 IF fbCalc.ip <> 0 THEN
15   hrCalc := fbCalc.ip.Addition(a,b,nSum);
16   hrCalc := fbCalc.ip.Subtraction(a,b,nDiff);
17 END_IF
18
```

⇒ Das Beispiel ist bereit zum Test.

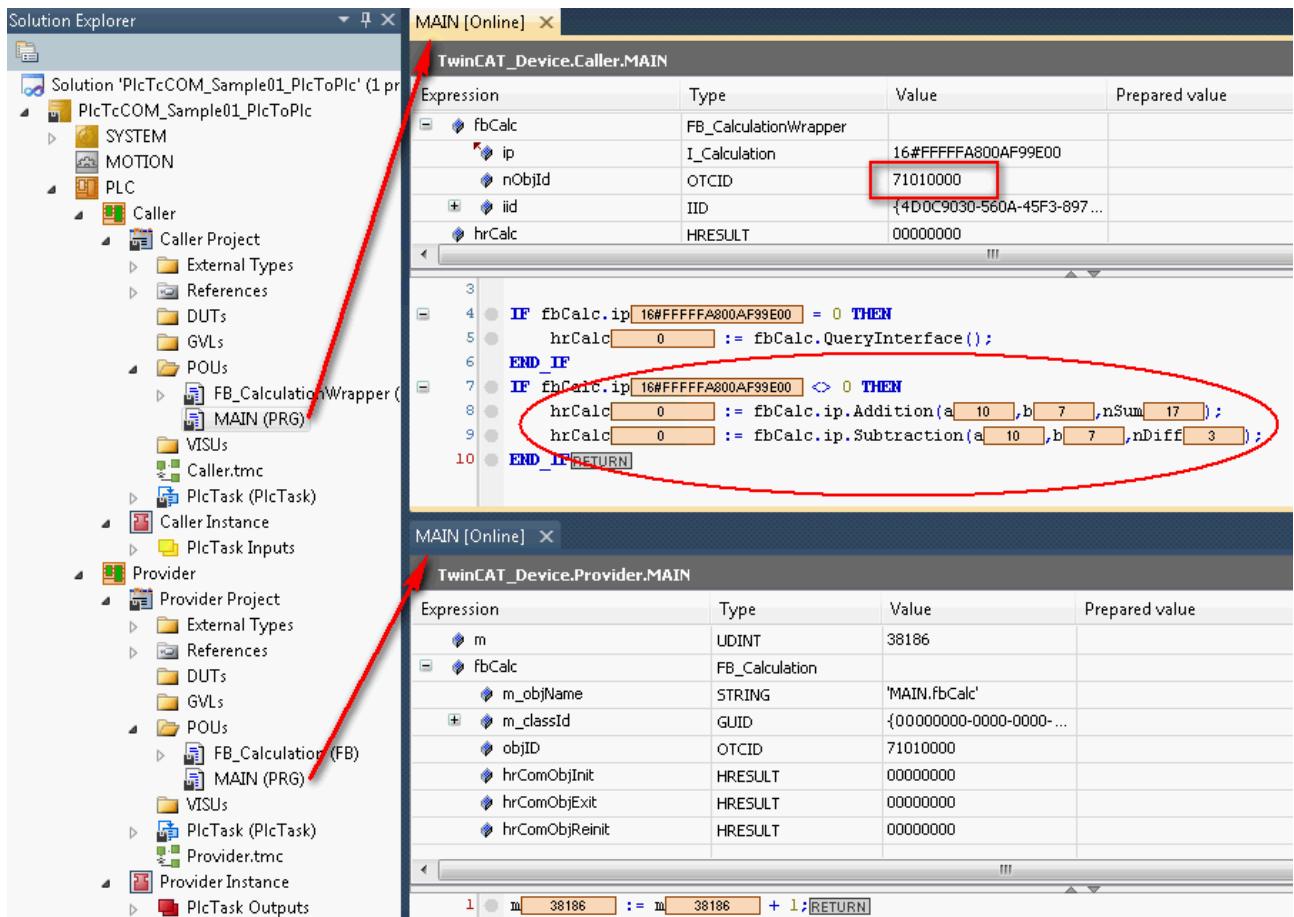
**Hinweis:** In welcher Reihenfolge die zwei SPSn später starten, ist bei dieser Implementierung irrelevant.

### 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++ Treibers 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++ 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. Instanziieren einer TwinCAT C++ Klasse als TwinCAT TcCOM Objekt [▶ 296]
2. Erstellen eines FBs in der SPS, welcher als einfacher Wrapper die Funktionalität des C++ Objektes anbietet [▶ 296]
3. Ausführung des Beispielprojektes [▶ 299]

Download des Beispiels: TcCOM\_Sample02\_PlctoCpp.zip ([https://infosys.beckhoff.com/content/1031/TC3\\_C/Resources/zip/2343048971.zip](https://infosys.beckhoff.com/content/1031/TC3_C/Resources/zip/2343048971.zip))

### Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64	Tc3_Module

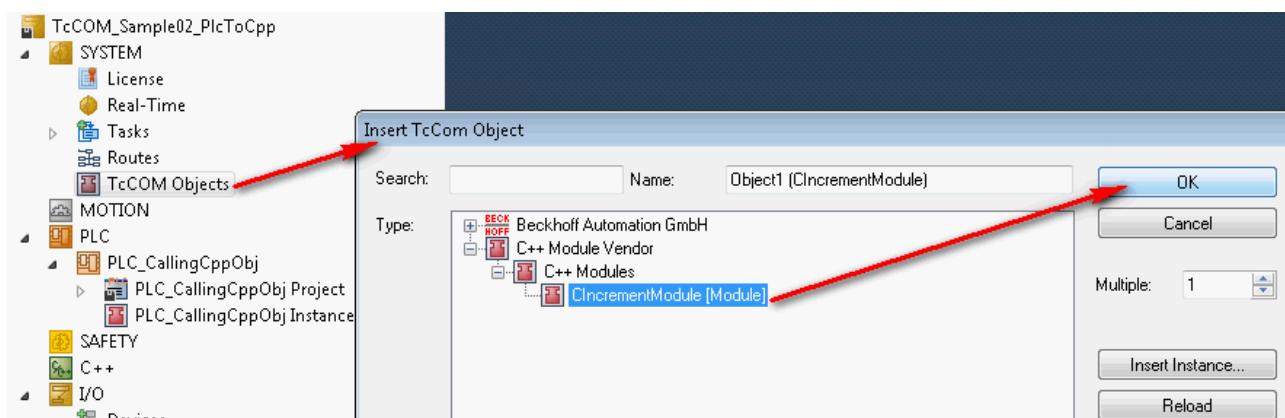
### 15.25.2.1 Instanziieren einer TwinCAT C++ Klasse als TwinCAT TcCOM Objekt

Der TwinCAT C++ Treiber muss auf dem Zielsystem zur Verfügung stehen. TwinCAT bietet hierfür ein Deployment, sodass die Komponenten nur passend auf dem Entwicklungsrechner abgelegt sein müssen.

Der existierende TwinCAT C++ Treiber sowie dessen TMC-Beschreibungsdatei(en) stehen als Treiber Archiv zur Verfügung. Dieses Archiv (IncrementerCpp.zip) wird in folgendem Ordner entpackt:  
C:\TwinCAT\3.1\CustomConfig\Modules\IncrementerCpp\

Das TwinCAT Deployment kopiert die Datei(n) später beim Aktivieren einer Konfiguration im folgenden Ordner auf dem Zielsystem:  
C:\TwinCAT\3.1\Driver\AutoInstall\

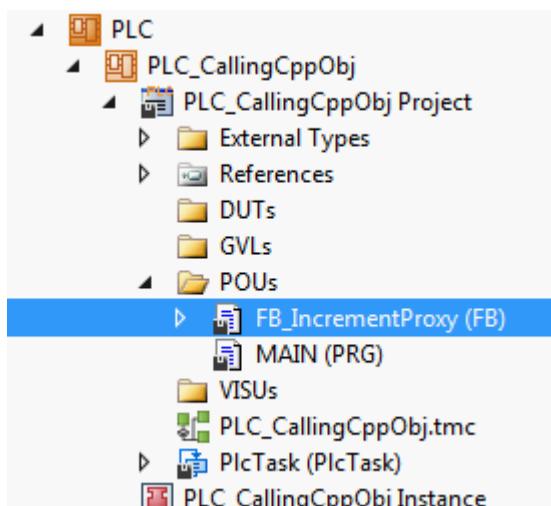
1. Öffnen Sie ein TwinCAT Projekt oder legen Sie ein neues Projekt an.
2. Fügen Sie in der Solution unter dem Knotenpunkt „TcCOM Objekte“ eine Instanz der Klasse ClncrementModule hinzu.



 <b>Hinweis</b>	<b>Erstellung des C++ Treibers</b> In der <a href="#">Dokumentation zu TwinCAT C++ [▶ 9]</a> wird ausführlich erläutert, wie C++ Treiber für TwinCAT erstellt werden. Um das oben erwähnte Treiber-Archiv zu erstellen wird bei der Treibererstellung als letzter Schritt „Publish TwinCAT Modules“ aus dem C++ Projektkontext gewählt.
--	---

### 15.25.2.2 Erstellen eines FBs in der SPS, welcher als einfacher Proxy die Funktionalität des C++ Objektes anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.



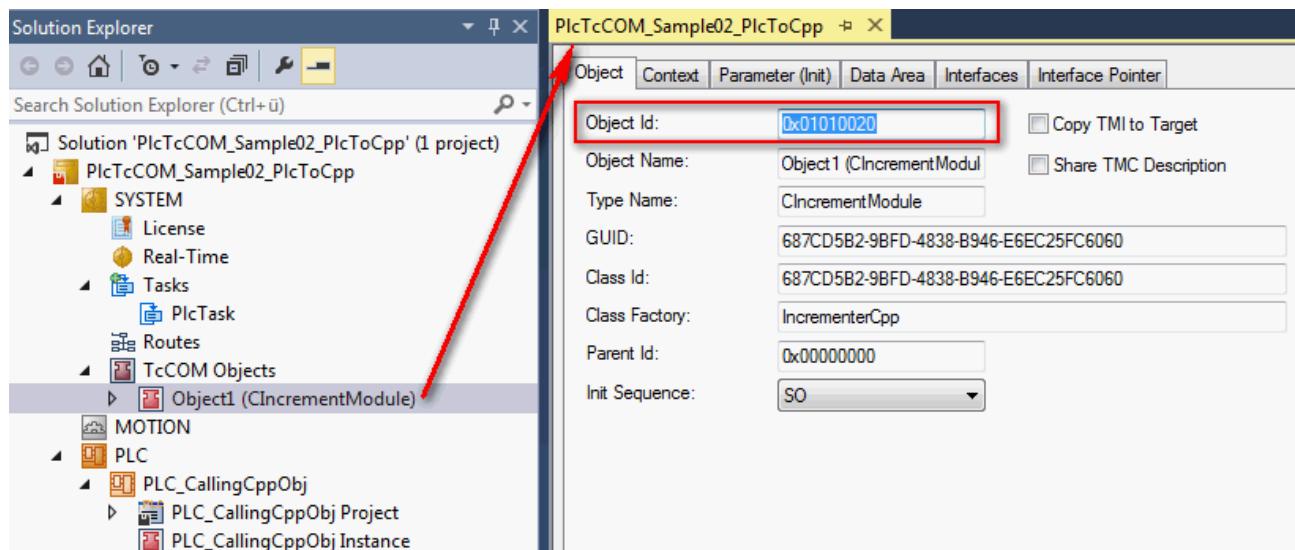
⇒ Dieser Proxy-Baustein soll die Funktionalität bereitstellen, welche in C++ programmiert wurde. Dies kann er über einen Schnittstellenzeiger, der von der C++ Klasse definiert wurde und aufgrund der TMC-Beschreibungsdatei in der SPS bekannt ist.

2. Deklarieren Sie im Deklarationsteil des Funktionsbausteins als Ausgang einen Schnittstellenzeiger auf die Schnittstelle, welche 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.

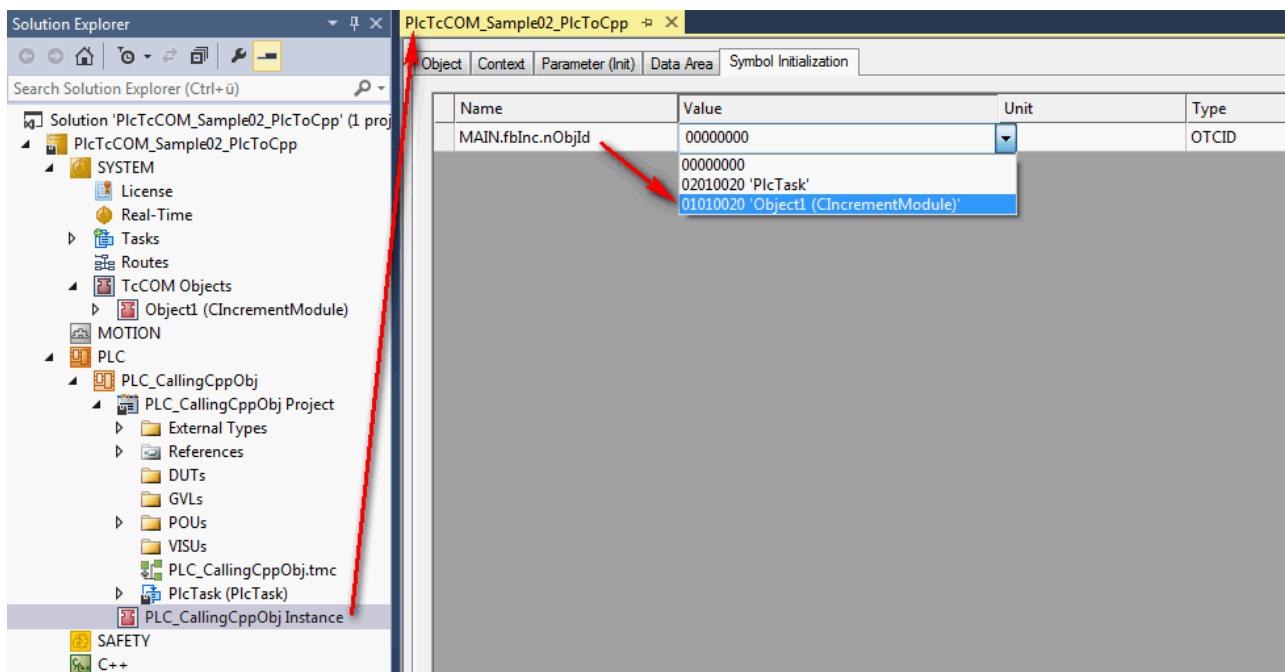
```
FUNCTION_BLOCK FB_IncrementProxy
VAR_OUTPUT
    ip : IIncrement;
END_VAR

VAR
    {attribute 'TcInitSymbol'}
    {attribute 'displaymode':='hex'}
    nObjId : OTCID;      // Instance configured to be retrieved
    iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
    hrInit : HRESULT;
END_VAR
```

⇒ Die Objekt-ID wird bei Anwahl des Objektes unter dem Knoten TcCOM-Objekte angezeigt.



- ⇒ Die Liste der Symbol-Initialisierungen befindet sich, sofern das Attribut TcInitSymbol verwendet wurde, im Knotenpunkt der SPS-Instanz im Reiter „Symbol Initialisierung“. Weisen Sie hier dem Symbolnamen der Variablen mittels DropDownList 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



**Hinweis:** Die Übergabe der Objekt-ID könnte alternativ auch mittels Prozessabbildverknüpfung wie im ersten Beispiel implementiert werden ([TcCOM\\_Sample01\\_PlctoPlc \[► 285\]](#)).

#### 4. 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 # -> X
1 {attribute 'hide'}
2 METHOD FB_init : BOOL
3 VAR_INPUT
4     bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
5     bInCopyCode : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online
6 END_VAR
7
8 IF NOT bInCopyCode THEN // if not online change
9     IF nObjID <> 0 THEN
10         hrInit := FW_ObjMgr_GetObjectInstance(oid:=nObjID, iid:=iid, pipUnk:=ADR(ip));
11     ELSE
12         hrInit := E_HRESULTAdsErr.INVALIDOBJID;
13     END_IF
14 END_IF
```

#### 5. 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 # -> X FB_IncrementProxy.FB_init # -> X
1 {attribute 'hide'}
2 METHOD FB_exit : BOOL
3 VAR_INPUT
4     bInCopyCode : BOOL; // if TRUE, the exit method is called for
5 END_VAR
6
7 IF NOT bInCopyCode THEN // if not online change
8     FW_SafeRelease(ADR(ip));
9 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* x
1 PROGRAM MAIN
2
3 VAR
4     fbInc : FB_IncrementProxy;
5     nValue : UDINT;
6 END_VAR
7
8
9 IF fbInc.ip <> 0 THEN
10     fbInc.ip.doIncrement(4, ADR(nValue));
11 END_IF
12
13

```

⇒ 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.

⇒ 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.

Expression	Type	Value	Prepared value
i	INT	247	
fbInc	FB_IncrementWrapper		
ip	IIncrement	16#FFFFFA800A95C0F8	
nObjId	OTCID	01010020	
iid	IID	{25ACB7D7-0596-4ADS-...}	
hrInit	HRESULT	00000000	
nValue	UDINT	988	

```

1 i 247 := i 247 + 1;
2
3 IF fbInc.ip 16#FFFFFA800A95C0F8 <> 0 THEN
4     fbInc.ip.doIncrement(4, ADR(nValue 988));
5 END_IF RETURN

```

### 15.25.3 TcCOM\_Sample03\_PlCreatesCpp

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 Entwicklungsgrechner 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 [► 300]
2. Erstellen eines FBs in der SPS, welcher das C++ Objekt anlegt und dessen Funktionalität anbietet [► 301]
3. Ausführung des Beispielprojektes [► 303]

Download des Beispiels: TcCOM\_Sample03\_PlCreatesCpp.zip ([https://infosys.beckhoff.com/content/1031/TC3\\_C/Resources/zip/2343051531.zip](https://infosys.beckhoff.com/content/1031/TC3_C/Resources/zip/2343051531.zip))

#### Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64	Tc3_Module

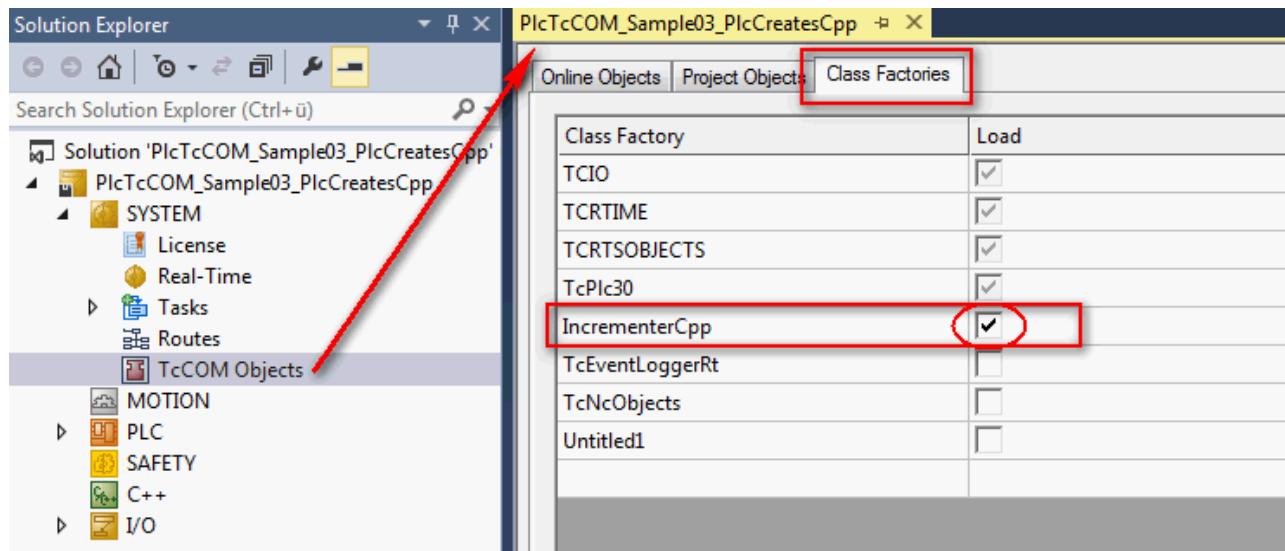
#### 15.25.3.1 Bereitstellen eines TwinCAT C++ Treibers und seiner Klassen

Der TwinCAT C++ Treiber muss auf dem Zielsystem zur Verfügung stehen. TwinCAT bietet hierfür ein Deployment, so dass die Komponenten nur passend auf dem Entwicklungsrechner abgelegt sein müssen.

Der existierende TwinCAT C++ Treiber sowie dessen TMC-Beschreibungsdatei(en) stehen als Treiber Archiv zur Verfügung. Dieses Archiv (IncrementerCpp.zip) wird in folgendem Ordner entpackt:  
C:\TwinCAT\3.1\CustomConfig\Modules\IncrementerCpp\

Das TwinCAT Deployment kopiert die Datei(n) später beim Aktivieren einer Konfiguration in folgenden Ordner auf dem Zielsystem:  
C:\TwinCAT\3.1\Driver\AutoInstall\

1. Öffnen Sie ein TwinCAT-Projekt oder legen Sie ein neues Projekt an.
  2. Wählen Sie in der Solution unter dem Knotenpunkt TcCom-Objekte im Reiter „Class Factories“ den benötigten C++ Treiber aus.
- ⇒ So wird sichergestellt, dass der Treiber beim Starten von TwinCAT auf dem Zielsystem geladen wird. Zudem sorgt diese Auswahl für das beschriebene Deployment.

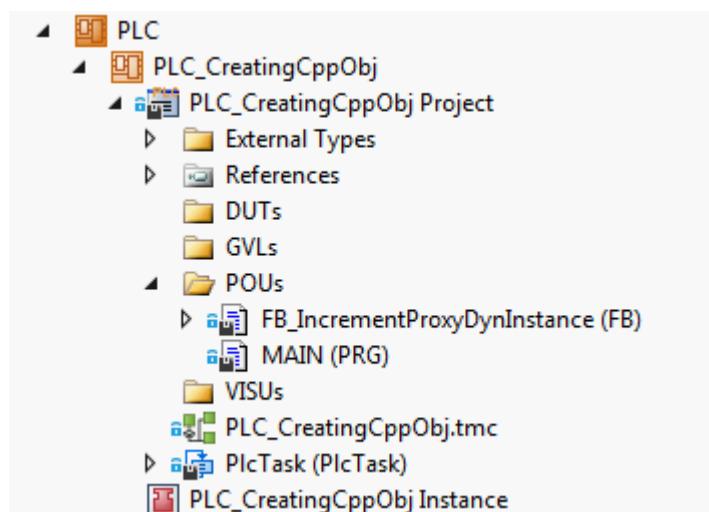


 <b>Hinweis</b>	<p><b>Erstellung des C++ Treibers</b></p> <p>In der <a href="#">Dokumentation zu TwinCAT C++ [▶ 9]</a> wird ausführlich erläutert, wie C++ Treiber für TwinCAT erstellt werden.</p> <p>Für Sample03 ist zu beachten, dass TwinCAT C++ Treiber, deren Klassen dynamisch instanziert werden sollen, als „TwinCAT Module Class for RT Context“ definiert sein müssen. Der C++ Wizard bietet hierfür ein spezielles Template an.</p> <p>Des Weiteren verwendet dieses Beispiel eine TwinCAT C++ Klasse, welche ohne TcCOM-Initialisierungsdaten und ohne TcCOM-Parameter auskommt.</p>
--	--

### 15.25.3.2 Erstellen eines FBs in der SPS, welcher 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, welche 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), welche später die Funktionalität nach außen bereitstellt.

```
FB_IncrementProxyDynInstance* □ X
1  FUNCTION_BLOCK FB_IncrementProxyDynInstance
2  VAR_OUTPUT
3      ip : IIncrement;
4  END_VAR
5
6  VAR
7      classId : CLSID := STRING_TO_GUID('687cd5b2-9bfd-4838-b946-e6ec25fc6060');
8      iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
9      hrInit : HRESULT;
10 END_VAR
11
```

3. Legen Sie die Klassen-ID und die Schnittstellen-ID als Membervariablen an.  
 Während die Schnittstellen-ID über eine globale Liste bereits verfügbar ist, wird die Klassen-ID, sofern sie noch nicht bekannt sein sollte, über einen anderen Weg ermittelt. Wenn Sie die TMC-Beschreibungsdatei des zugehörigen C++ Treibers öffnen, finden Sie die entsprechende GUID dort vor.

```
13 <Modules>
14     <Module GUID="{687cd5b2-9bfd-4838-b946-e6ec25fc6060}" Group="C++">
15         <Name>CIncrementModule</Name>
16         <CLSID ClassFactory="IncrementerCpp">{687cd5b2-9bfd-4838-b946-e6ec25fc6060}</CLSID>
17         <Licenses>
18             <License>
```

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. Dabei 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, wodurch sie bei Instanziierung des Proxy-Bausteins anzugeben sind. Die zu instanzierende 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 □ X
1  METHOD FB_init : BOOL
2  VAR_INPUT
3      bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
4      bInCopyCode : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online change)
5
6      sObjName : STRING; // object name to be set for this instance (optional)
7      eObjState : TCOM_STATE; // target object state (usually TCOM_STATE.TCOM_STATE_OP)
8  END_VAR
9
10 IF NOT bInCopyCode THEN // if not online change
11     objName := sObjName;
12     hrInit := FW_ObjMgr_CreateAndInitInstance( clsId      := classId,
13                                                 iid        := iid,
14                                                 pipUnk    := ADR(ip),
15                                                 objId     := OTCID_CreateNewId,
16                                                 parentId  := TwinCAT_SystemInfoVarList._AppInfo.ObjId, /
17                                                 name      := sObjName,
18                                                 state     := eObjState,
19                                                 pInitData := 0 );
20 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 -> X FB_IncrementProxyDynInstance.FB_init
1 {attribute 'hide'}
2 METHOD FB_exit : BOOL
3 VAR_INPUT
4     bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance
5 END_VAR

1 IF NOT bInCopyCode THEN // if not online change
2     FW_ObjMgr_DeleteInstance(ADR(ip));
3 END_IF

```

⇒ Damit ist die Implementierung des Proxy-Funktionsbausteins bereits abgeschlossen.

- 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* -> X
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

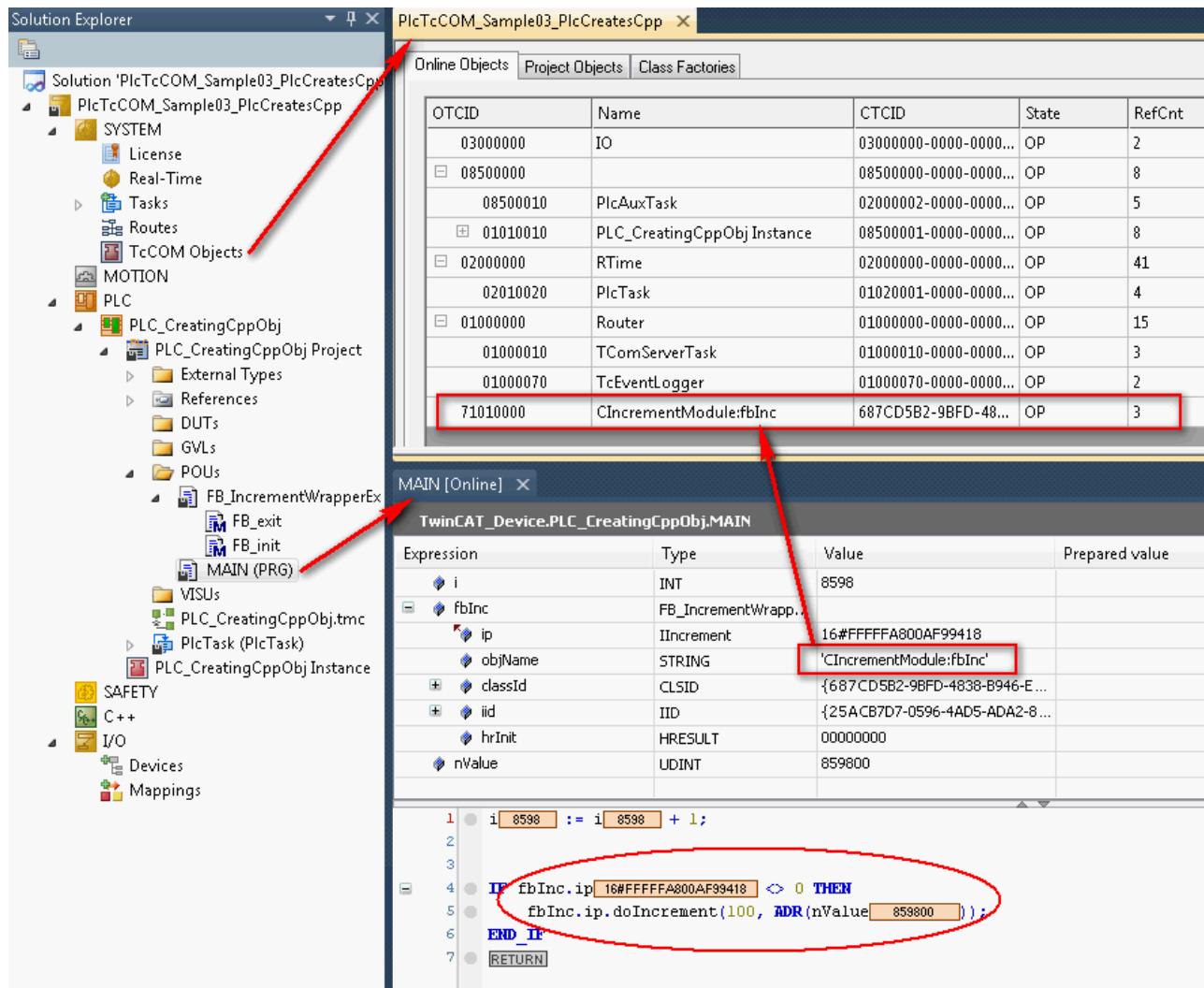
1 IF fbInc.ip <> 0 THEN
2     fbInc.ip.doIncrement(100, ADR(nValue));
3 END_IF
4

```

⇒ Das Beispiel ist bereit zum Test.

### 15.25.3.3 Ausführung des Beispielprojektes

- Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
  - Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten der SPS aus.
- ⇒ In der Onlineansicht der SPS-Applikation ist der gewünschte TcCOM-Objektname im SPS-Proxy-Baustein ersichtlich. Der Projektnamen 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 \[▶ 305\]](#) sind im gesamten TwinCAT 3 Bereich wichtig, hier insbesondere, wenn [ADS-Kommunikation \[▶ 175\]](#) selber implementiert wird.
- Die [Retain Daten \[▶ 308\]](#) (auf NOVRAM Speicher) sind auf ähnliche Art aus der PLC und auch C++ nutzbar.
- Neben dem Konzept der [TcCOM Module \[▶ 29\]](#) ist das TwinCAT 3 [Typsystem \[▶ 311\]](#) 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 \[▶ 317\]](#)
  - [Erstellung von und Umgang mit TcCOM Modulen \[▶ 321\]](#)

### 16.1 ADS Return Codes

Fehlercode: [0x000 \[▶ 305\]](#)..., [0x500 \[▶ 305\]](#)..., [0x700 \[▶ 306\]](#)..., [0x1000 \[▶ 308\]](#).....

#### Globale Fehlercodes

Hex	Dec	Beschreibung
0x0	0	Kein Fehler
0x1	1	Interner Fehler
0x2	2	Keine Echtzeit
0x3	3	Zuweisung gesperrt-Speicherfehler
0x4	4	Postfach voll
0x5	5	Falsches HMSG
0x6	6	Ziel-Port nicht gefunden
0x7	7	Zielrechner nicht gefunden
0x8	8	Unbekannte Befehl-ID
0x9	9	Ungültige Task-ID
0xA	10	Kein IO
0xB	11	Unbekannter ADS-Befehl
0xC	12	Win32 Fehler
0xD	13	Port nicht angeschlossen
0xE	14	Ungültige ADS-Länge
0xF	15	Ungültige AMS Net ID
0x10	16	niedrige Installationsebene
0x11	17	Kein Debugging verfügbar
0x12	18	Port deaktiviert
0x13	19	Port bereits verbunden
0x14	20	ADS Sync Win32 Fehler
0x15	21	ADS Sync Timeout
0x16	22	ADS Sync AMS Fehler
0x17	23	Keine Index-Map für ADS Sync vorhanden
0x18	24	Ungültiger ADS-Port
0x19	25	Kein Speicher
0x1A	26	TCP Sendefehler
0x1B	27	Host nicht erreichbar
0x1C	28	Ungültiges AMS Fragment

#### Router Fehlercodes

<b>Hex</b>	<b>Dec</b>	<b>Name</b>	<b>Beschreibung</b>
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht. Die aktuell gesendete Nachricht wurde abgewiesen.
0x503	1283	ROUTERERR_DEBUGBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht. Die gesendete Nachricht wird nicht im ADS Monitor angezeigt.
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	ROUTERERR_PORTALREADYINUSE	Die gewünschte Portnummer ist bereits vergeben.
0x507	1287	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	ROUTERERR_NOMOREQUEUES	Die maximale Anzahl von Ports ist erreicht.
0x509	1289	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.

### Allgemeine ADS Fehlercodes

Hex	Dec	Name	Beschreibung
0x700	1792	ADSERR_DEVICE_ERROR	Gerätefehler
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt
0x702	1794	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	Lesen und schreiben nicht gestattet.
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt
0x706	1798	ADSERR_DEVICE_INVALIDDATA	Ungültige Parameter-Werte
0x707	1799	ADSERR_DEVICE_NOTREADY	Gerät ist nicht betriebsbereit
0x708	1800	ADSERR_DEVICE_BUSY	Gerät ist beschäftigt
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext (muss in Windows sein)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte
0x70C	1804	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein
0x70F	1807	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	Gerät im ungültigen Zustand
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	Keine weitere Notification Handle
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß
0x718	1816	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert
0x719	1817	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig
0x71E	1822	ADSERR_DEVICE_PENDING	Anforderung steht aus
0x71F	1823	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen
0x720	1824	ADSERR_DEVICE_WARNING	Signal-Warnung
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	Lizenz der System-ID ungültig
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOOLONG	Lizenz-Zeitraum zu lang
0x72c	1836	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	öffentliches Zertifikat
0x740	1856	ADSERR_CLIENT_ERROR	Clientfehler
0x741	1857	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter
0x742	1858	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer
0x743	1859	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten
0x746	1862	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	ADS-Port nicht geöffnet
0x750	1872	ADSERR_CLIENT_NOAMSADDR	Interner Fehler in Ads-Sync

Hex	Dec	Name	Beschreibung
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	Hash-Tabelle-Überlauf
0x752	1874	ADSERR_CLIENT_ADDHASH	Schlüssel nicht gefunden im Hash
0x753	1875	ADSERR_CLIENT_REMOVEHASH	Keine weitere Symbole im Cache
0x754	1876	ADSERR_CLIENT_NOMORESYM	Ungültige Antwort empfangen
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	Sync Port ist gesperrt

### RTIME Fehlercodes

Hex	Dec	Name	Beschreibung
0x1000	4096	RTERR_INTERNAL	Interner Fehler im TwinCAT Echtzeit-System.
0x1001	4097	RTERR_BADTIMERPERIODS	Timer-Wert ist nicht gültig.
0x1002	4098	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	RTERR_INVALIDSTACKPTR	Task Stackpointer hat den ungültigen Wert 0.
0x1004	4100	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	RTERR_NOMORETCB	Kein freies TCB (Task Control Block) zur Verfügung. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	RTERR_EXTIRQNOTDEF	Kein externer Synchronisations-Interrupt angewandt.
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs- Interrupts ist fehlgeschlagen
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

### 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

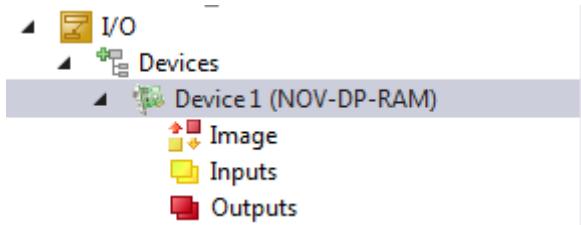
Dieser Bereich beschreibt die Möglichkeit Daten über einen geordneten oder spontanen Neustart einer Anlage bereitzuhalten. Hierfür wird das NOV-RAM eines Gerätes verwendet.

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.

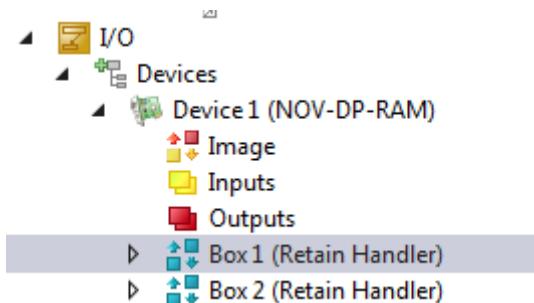
### Konfiguration eines Retain Gerätes

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.

Hierfür wird im IO Bereich der Solution ein NOVRAM DP Device angelegt.

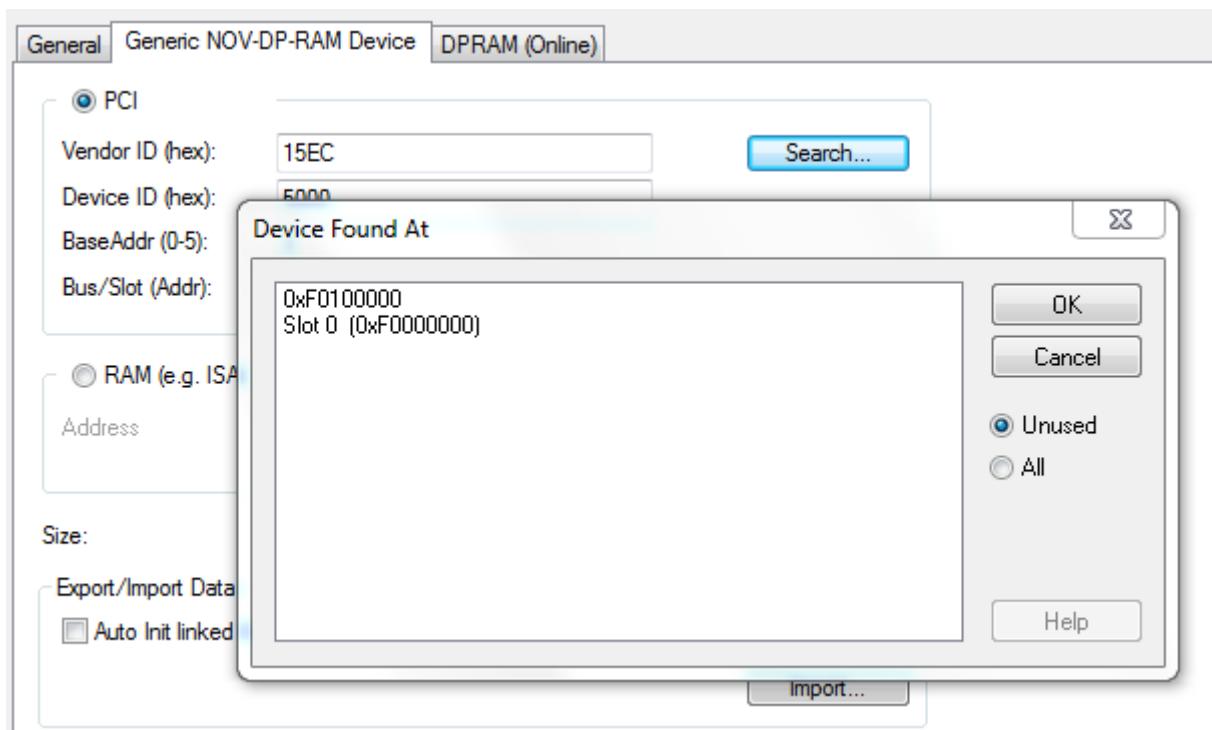


Unterhalb dieses Gerätes können ein oder mehrere Retain Handler angelegt werden



### Speicherort: NOVRAM

Das NOV-DP-RAM-Gerät muss konfiguriert werden. Im Tab „Generic NOV-DP-RAM Device“ kann über „Search...“ der zu verwendende Bereich definiert werden.



Für die Symbolik wird zusätzlich im Boot-/Verzeichnis von TwinCAT ein Retain-/Verzeichnis angelegt.

### Nutzung des Retain Handlers mit einem PLC Projekt

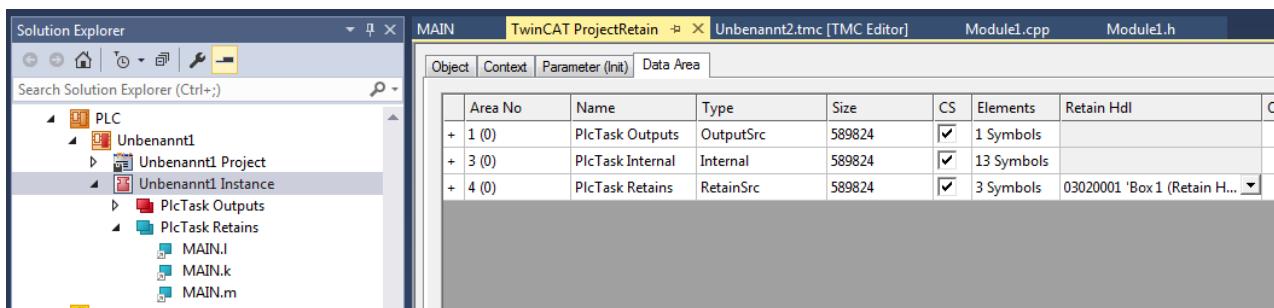
In einem PLC Projekt werden die Variablen entweder in einem VAR RETAIN Bereich angelegt, oder mit dem Attribut TcRetain versehen.

```

PROGRAM MAIN
VAR RETAIN
    l: UINT;
    k AT %Q* : UINT;
END_VAR
VAR
    {attribute 'TcRetain':='1'}
    m: UINT;
    x: UINT;
END_VAR

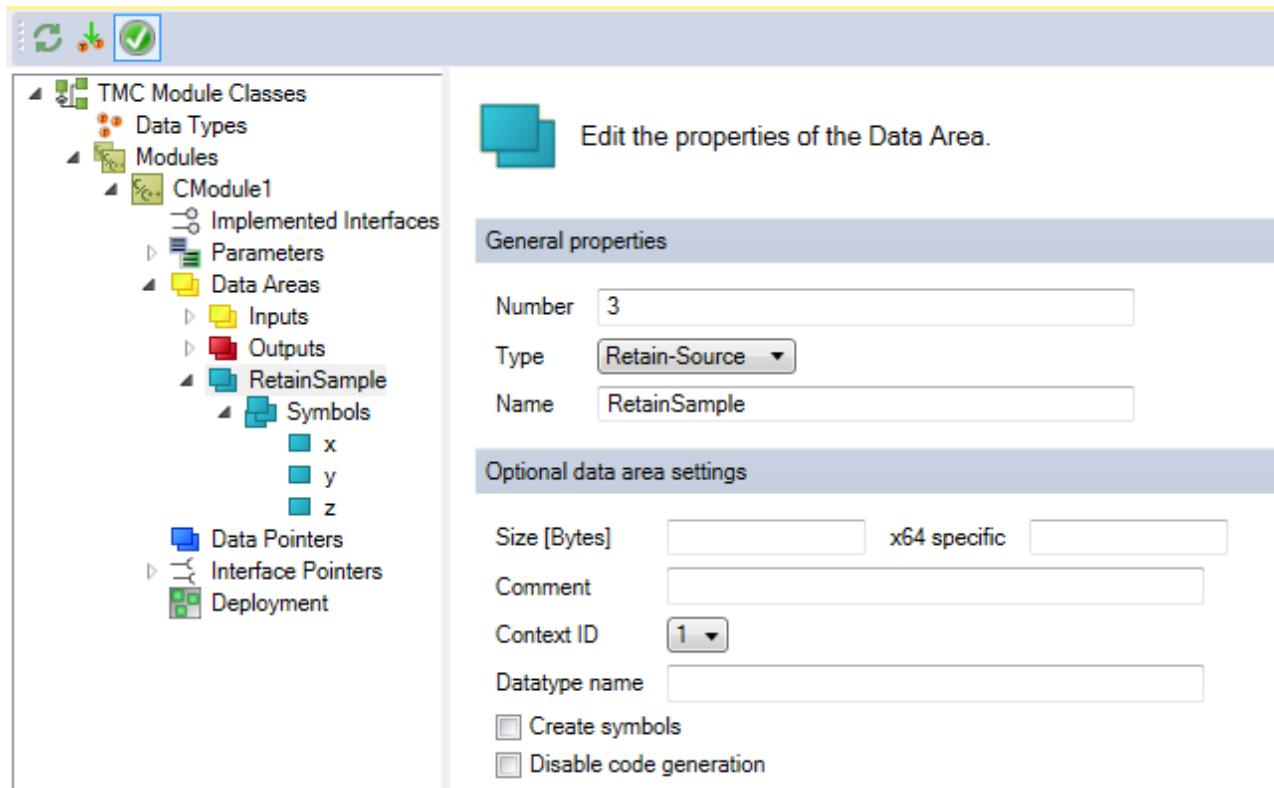
```

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.

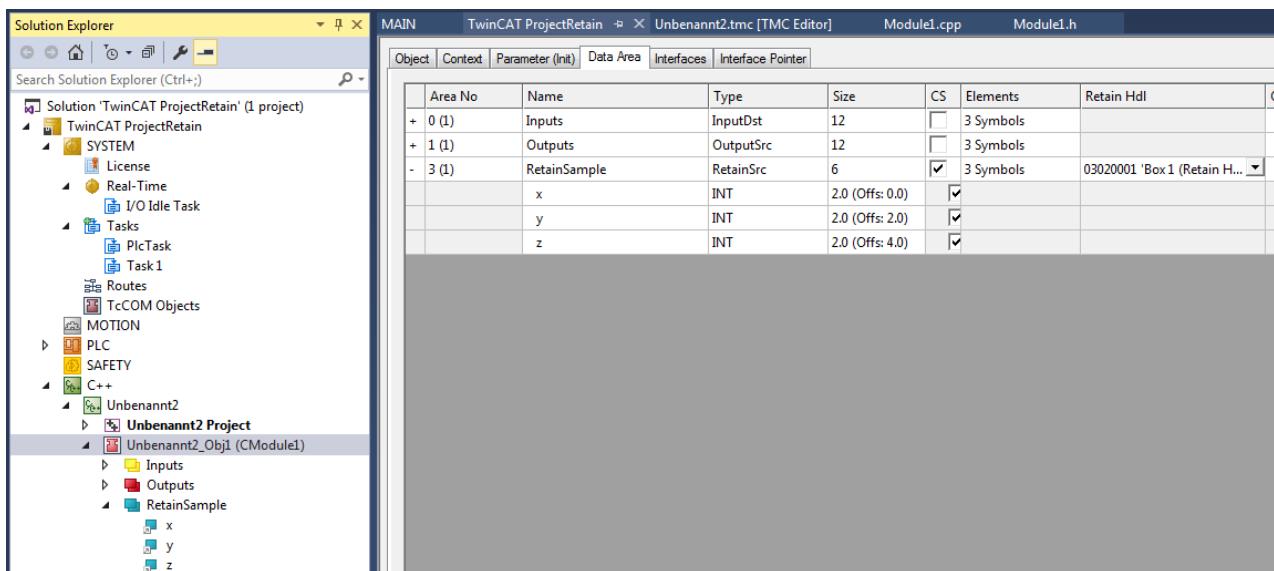


### Nutzung des Retain Handlers mit einem C++ Modul

In einem C++ Modul wird eine Data Area vom Typ Retain-Source angelegt, die die entsprechenden Symbole enthält.

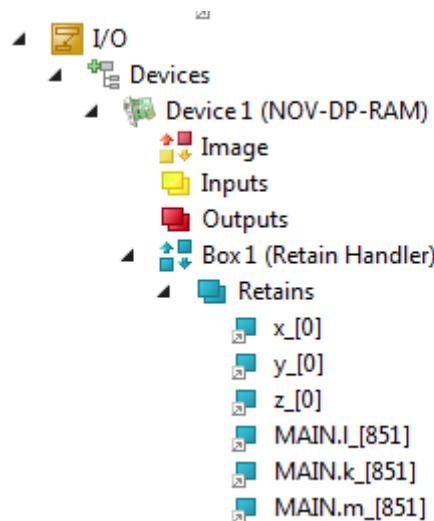


An den Instanzen des C++ Moduls wird ein zu verwendender Retain Handler des NOV-DP-RAM-Gerätes für diese Data Area in der Spalte „Retain Hdl“ vorgenommen.



## 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 Typsystem

TwinCAT 3 stellt ein Typsystem für die Verwaltung von Datentypen bereit. Das Typsystem besteht aus System-Basistypen und kann durch das Kundenprojekt um eigene Datentypen erweitert werden.

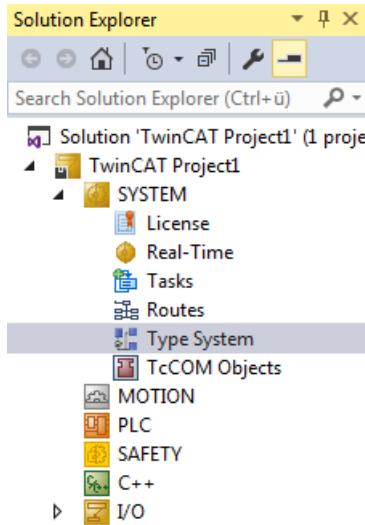
Diese Dokumentation beschreibt das TwinCAT 3 Typsystem und die Verwaltung von Datentypen. Der TMC-Editor, mit dem Datentypen erstellt und beschrieben werden, wird in der Dokumentation "C++" im Abschnitt "["TwinCAT Module Class Editor \(TMC\)" \[▶ 79\]](#)" beschrieben.

### 16.3.1 Projektbasiertes Typsystem

Das TwinCAT 3 Typsystem ist projektspezifisch, d.h. es ist fester Bestandteil eines TwinCAT 3 Projekts in einer Visual Studio Solution.

Datentypen können an unterschiedlichen Stellen definiert und bei Bedarf in das TwinCAT 3 Typsystem transferiert werden. So können auch lokale Datentypen existieren, die nicht im TwinCAT 3 Typsystem vorhanden sind.

Sie finden das Typsystem im TwinCAT 3 Projektbaum als Objekt im Bereich "SYSTEM".



### 16.3.2 Arten von Datentypen

Das TwinCAT 3 Typsystem stellt die Datentypen in einem Editor in vier unterschiedlichen Registerkarten dar. An dieser Stelle werden nur die "Data Types" und "Interfaces" beschrieben. Sie öffnen den Editor mit einem Doppelklick auf das Objekt "Type System" im TwinCAT 3 Projektbaum.

Name	NS	GUID	Size
ADMSYNC_COPYINFO		18071995-00...	32
AdsAddInitCommand		F6F369BF-57...	40
AMSADDR		18071995-00...	8
AMSHED		18071995-00...	32
AMSNETID		18071995-00...	6
EcNcTrafoParameter		D400B256-8F...	4
ETclotMqttClientState		DF915CC7-40...	4
ETHERNET_ADDRESS		CC07E0A0-F...	6
ETYPE_VLAN_HEADER		478C4436-6F...	4
INTERFACE_TYPE		ACAD4AA7-...	4
IOT_FORMAT		F0F5BE0A-A...	4
IP_HEADER		5D507FF7-FB	20

In der Registerkarte "Data Types" werden die folgenden Arten von Datentypen (TMC-Editor: "Specifications") dargestellt:

- Alias: Diese Datentypen sind einfach Synonyme für andere Datentypen. So kann beispielsweise ein Zeitbereich (Duration) als UINT projektspezifisch festgelegt werden.
- Struct: Diese Datentypen sind Strukturen von anderen Datentypen, die auch wieder Strukturen sein können.
- Enum: Diese Datentypen beschreiben Enumerations, also Aufzählungen.
- Array: Diese Datentypen sind Arrays mit definierter Dimensionen-Anzahl sowie jeweiliger Länge.

In der Registerkarte "Interfaces" werden die Interfaces dargestellt. Dieser Datentyp beschreibt eine Schnittstelle, die von unterschiedlichen Komponenten wie Funktionsbausteinen oder TcCOM-Modulen bereitgestellt oder genutzt werden kann. Ein Interface besteht aus Methoden, die eine jeweilige Signatur haben.

### 16.3.3 Handhabung von Datentypen

Um einen Datentyp über das TwinCAT 3 Typsystem anzulegen oder zu verändern, wählen Sie in der entsprechenden Registerkarte des Typsystem-Editors im Kontextmenü der ersten Tabellenspalte den Befehl "New" bzw. "Edit". Beide Befehle öffnen den TMC-Editor, in dem Sie den Datentyp bearbeiten können.

#### Datentypen aus SPS-Projekten

In einem SPS-Projekt können Datentypen (DUTs) angelegt und gespeichert werden. Diese Datentypen sind erst einmal lokal in dem SPS-Projekt vorhanden und aus Sicht des TwinCAT 3 Typsystems nicht nutzbar. Wenn die Datentypen im Ein-/Ausgangspeicherabbild (%I\* / %Q\*) verwendet werden, werden sie im TwinCAT 3 Typsystem eingebracht, sodass sie auch durch das Mapping verknüpfbar sind.

Mit dem Befehl "Convert to Global Type" im Kontextmenü eines DUTs im SPS-Projektbaum können Sie den DUT in das Typsystem des übergeordneten TwinCAT Projekts übertragen. Danach ist der Datentyp in der SPS über die externen Typen nutzbar und wird im TwinCAT 3 Typsystem verwaltet.

Um einen Datentyp vom TwinCAT 3 Typsystem in ein SPS-Projekt zu übertragen, können Sie den Quellcode in dem "Data Types"-Dialog verwenden.

#### Datentypen aus C++-Projekten

In C++-Projekten werden die Datentypen im TMC-Editor parallel zu den Modulen definiert. Diese Datentypen sind analog zu den SPS-Projekt-internen DUTs lokal und damit im TwinCAT 3 Typsystem nicht sichtbar.

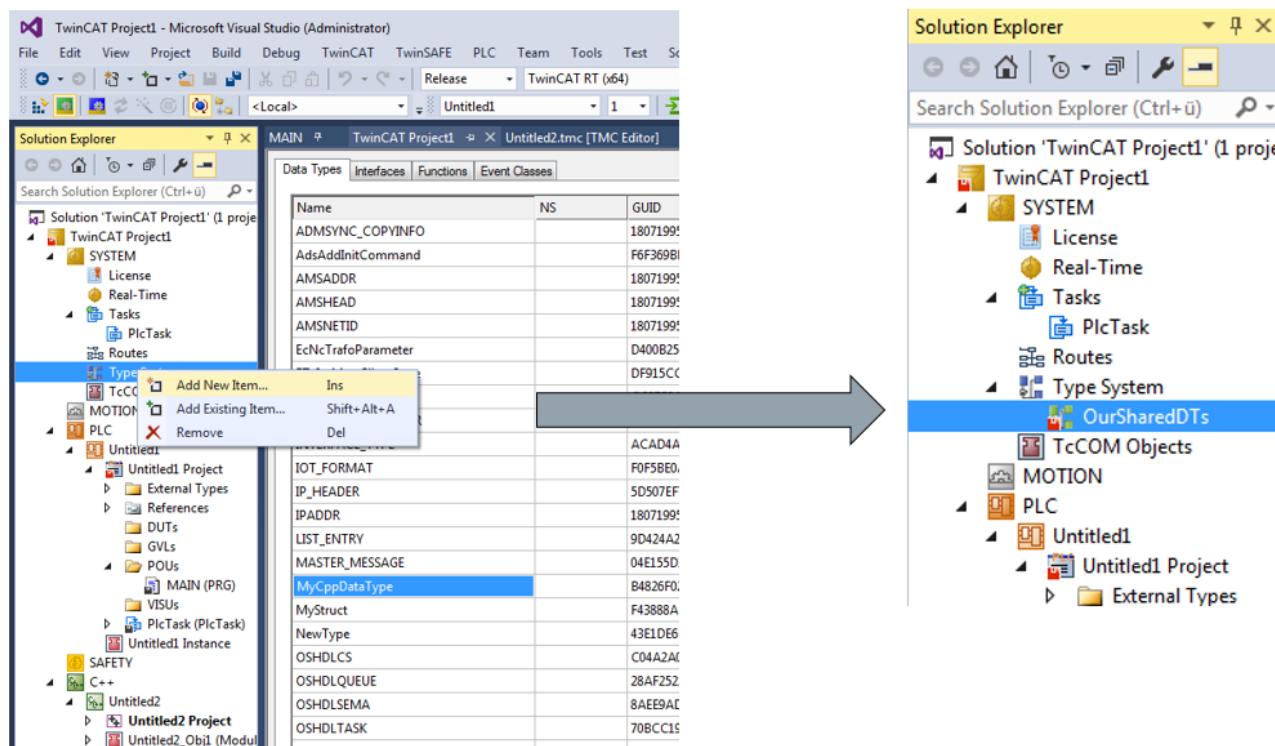
Durch Verwendung der Datentypen in einem C++/Matlab-Modul, welches auch instanziert wurde, werden die Datentypen in das TwinCAT 3 Typsystem eingefügt.

Sie können einen Datentyp auch durch das Aktivieren des Auswahlkästchens "Persistent (even if unused)" in das TwinCAT 3 Typsystem einfügen, ohne dass der Datentyp in einem instanziierten C++-Modul verwendet wird.

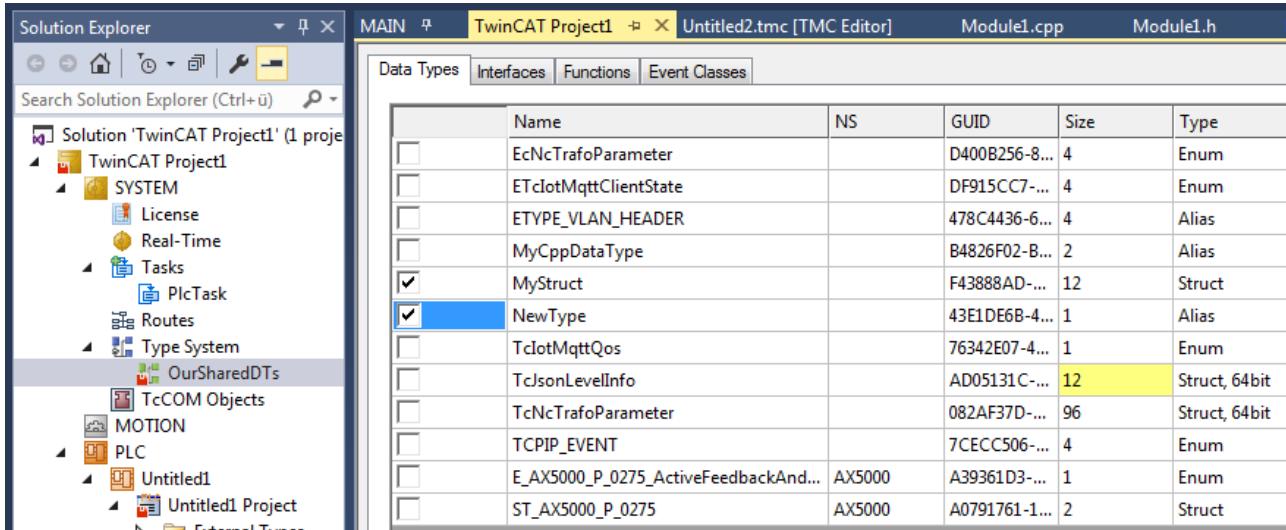
#### Nutzung von Datentypen in mehreren Projekten

In einigen Fällen kann es sinnvoll sein, Datentypen in mehreren Projekten zu verwenden. Insbesondere für EAP-/Netzwerkvariablen kann es sinnvoll sein, auf Publisher- und Subscriber-Seite den gleichen Datentyp zu nutzen.

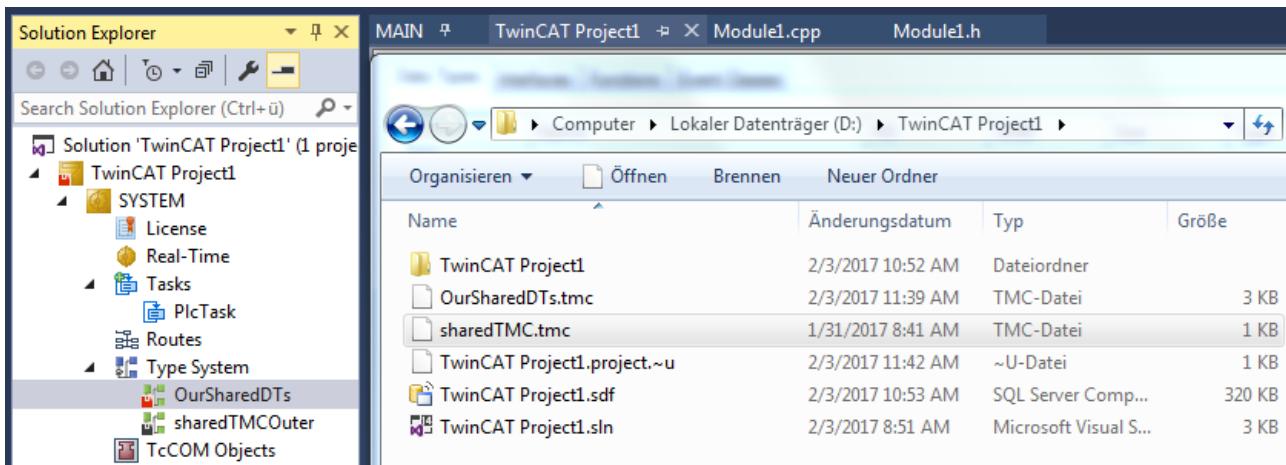
Unter dem Knoten "Type System" können Sie hierfür einzelne TMC-Dateien anlegen.



Im Editorfenster der TMC-Dateien erscheint vor jedem Datentyp ein Auswahlkästchen. Über das Auswahlkästchen können Sie angeben, welcher Datentyp in der jeweiligen TMC-Datei abgelegt werden soll.



Die Datentypen werden dabei zusätzlich in den TMC-Dateien abgelegt. So können diese Dateien z. B. per Dateiaustausch oder Versionskontrolle auf unterschiedlichen Rechnern und in unterschiedlichen Projekten verwendet werden.



Da die GUID zur Identifizierung von Datentypen genutzt wird, erkennt das Typsystem diese doppelte Ablage automatisch.

Beachten Sie bei der Verwendung von Datentypen, nachdem sie in mehreren Projekten eingebunden wurden, dass Änderungen an den Datentypen möglichst nur an einer Stelle vorgenommen werden.

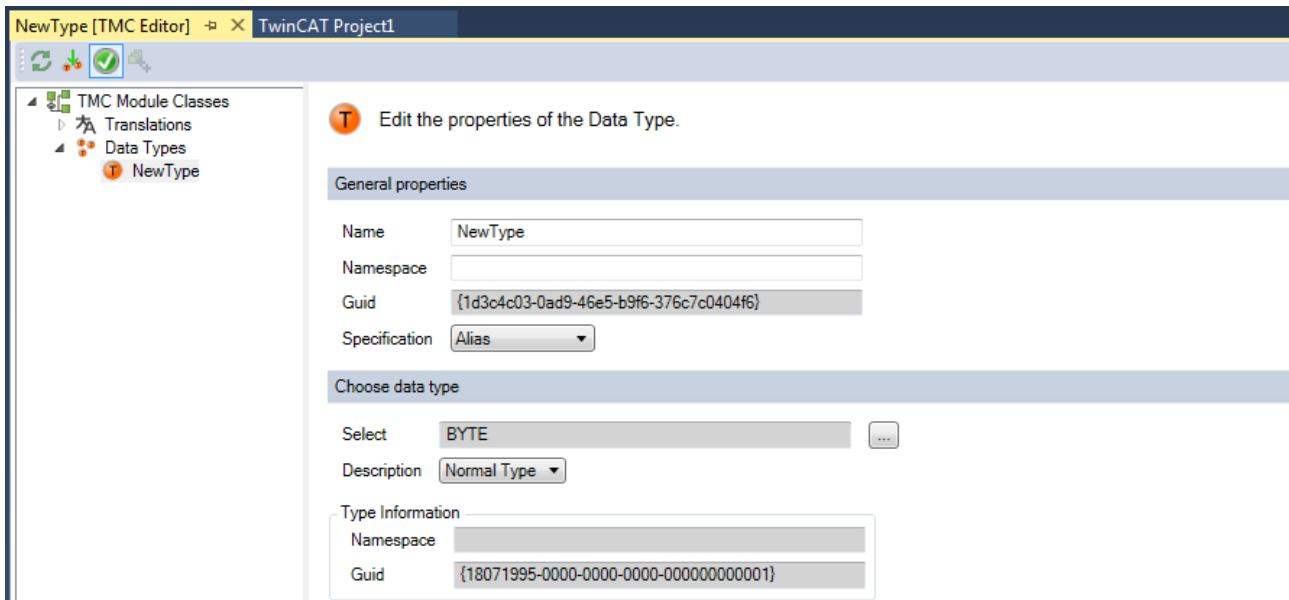
Ansonsten können die unterschiedlichen Varianten nicht mehr auf einen gemeinsamen Stand zusammengeführt werden können.

#### Siehe auch:

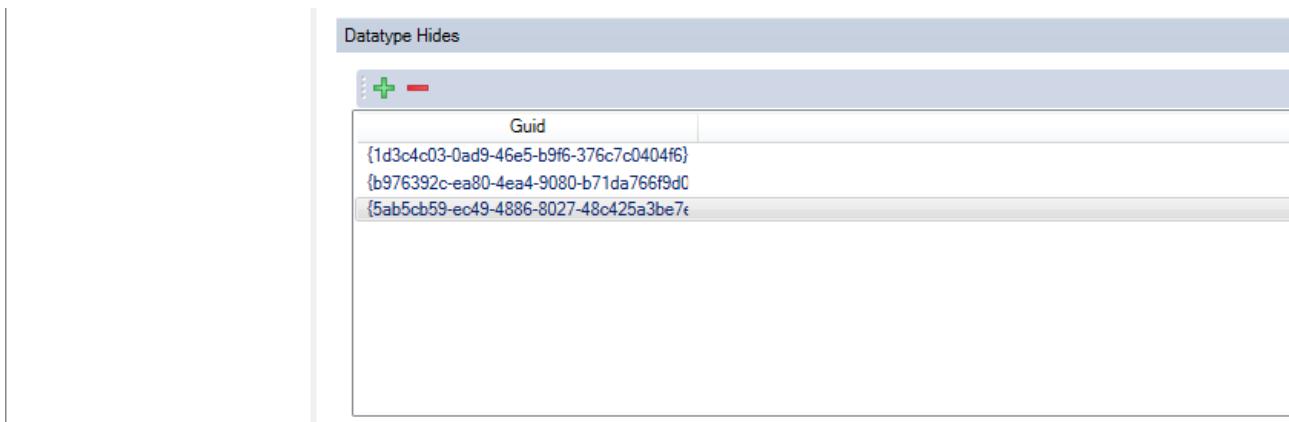
[Verwaltung und Identifizierung von Datentypen \[► 314\]](#)

### 16.3.4 Verwaltung und Identifizierung von Datentypen

Datentypen im TwinCAT 3 Typsystem werden grundsätzlich anhand ihrer GUID identifiziert. Somit können mehrere Datentypen mit gleichen Namen existieren. Dies gilt auch für unterschiedliche Versionen eines Datentyps. Jede Version eines Datentyps bekommt eine neue GUID zugewiesen.



Gleichzeitig besitzt jeder Datentyp eine Liste von Datentypen, die er versteckt ("Datatype Hides").



Hieraus ergibt sich die Möglichkeit unterschiedliche Versionen eines Datentyps gleichzeitig im Projekt zu nutzen.

Der Befehl "Update Instances..." im Kontextmenü eines Datentyps im Editor des Typsystems setzt für ausgewählte Verwendungen eines Datentyps die jeweils neueste Version ein.

TwinCAT besitzt für jeden Datentyp einen sogenannten Reference Counter. Dieser Zähler ist im Editor des Typsystems in der Spalte "RefCount" zu sehen. Jede Verwendung des Datentyps in einem Projekt, aber auch in einem Editor usw. erhöht den Zähler. Ist ein Zähler bei 0, wird der Datentyp nicht mehr genutzt und verworfen.

Mit den Befehlen "Auto Delete (if unused)" und "Persistent (even if unused)" im Kontextmenü kann das Verwerfen unterbunden werden.

The screenshot shows the 'Data Types' tab in the TwinCAT 3 Editor. A context menu is open over a row in the table, specifically for the entry 'NewType'. The menu items are: Edit, New, Auto Delete (if unused) (highlighted in yellow), Persistent (even if unused) (with a checked checkbox), Hide types with same name, Search References..., and Update Instances... Below the menu, there is a button labeled 'Delete unused Types...'.

### 16.3.5 Alignment von Datentypen

Das Speicher-Layout eines Datentyps wird durch das Alignment bestimmt. Weitere Informationen zum Alignment finden Sie in der Dokumentation "PLC" im Abschnitt "Alignment".

Mit dem Default-Alignment von 8-Bytes kann gewährleistet werden, dass der Zugriff auf Datentypen auf unterschiedlichen Plattformen optimal im Sinne von Laufzeit und Zugriff funktioniert. Nur in Ausnahmenfällen sollte hiervon abgewichen werden.

Das TwinCAT 3 Typsystem markiert Datentypen farbig.

- Gelb, wenn die Länge des Datentyps nicht ein Vielfaches des größten, internen Feldes (max. 8 Byte) ist. Dadurch entspricht bei einem Array eines solchen Datentyps das Alignment nicht den Regeln.

Name	Specification	Type	Value	Status
AlignmentMismatch		Struct	035500DD-FE07-4D6D-B195-...	10

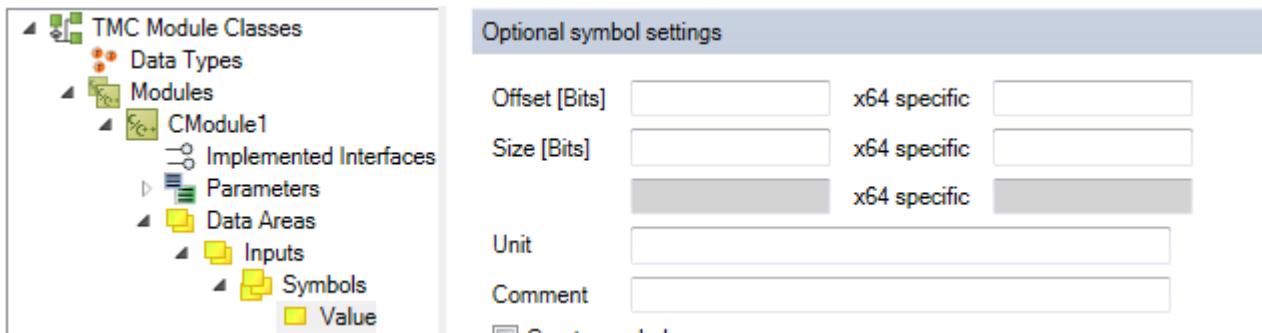
- Rot, wenn innerhalb des Datentyps das Alignment nicht den Regeln entspricht.

Name	Specification	Type	Value	Status
OuterType		Struct	566B0D7D-3403-4C85-8BEC-...	6

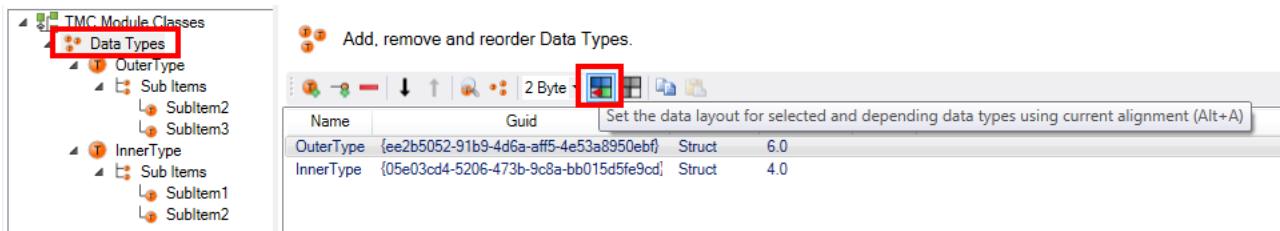
Der TMC-Editor bietet die Möglichkeit für ein ausgewähltes Alignment das Speicher-Layout eines Datentyps festzulegen.

The screenshot shows the 'Symbol Editor' interface. A dropdown menu is open under the alignment settings, listing '8 Byte' (which is selected), '1 Byte', '2 Byte', '4 Byte', '16 Byte', and '32 Byte'. A tooltip at the bottom right of the menu area says 'Set data layout for selected symbols using current alignment (Alt+A)'.

Alternativ kann das Layout über Offsets manuell festgelegt werden.



Wird die Größe eines Datentyps verändert, der in einem anderen Datentyp verwendet wird, muss auch dieser Datentyp angepasst werden. Hierfür bietet der TMC-Editor auf Ebene der Datentypen-Übersicht eine entsprechende rekursive Funktion.



### 16.3.6 Dateien im Zusammenhang mit dem Typsystem

Das TwinCAT 3 Typsystem ist vollständig in XML formuliert.

Je nach Anwendungsbereich gibt es unterschiedliche Dateien, die die Datentypen enthalten:

- .tsproj-Datei – TwinCAT Projekt  
Diese Datei umfasst das gesamte TwinCAT Projekt einschließlich des gesamten TwinCAT 3 Typsystems.
- .tmc-Dateien – TwinCAT Module Class Dateien  
Diese Dateien werden verwendet, um die TcCOM-Module selbst zu beschreiben. Sie umfassen Modulklassenbeschreibungen und verwendete Datentypen. Gleichzeitig werden diese Dateien genutzt um, wie oben beschrieben, einen Austausch von Datentypen zwischen Projekten zu realisieren.
- .tmi-Dateien – TwinCAT Module Instance Dateien  
Diese Dateien beschreiben die Instanz einer Klasse. Sie werden vom TwinCAT 3 Engineering auf dem Ziel abgelegt, um eine Instanz einer Klasse zu beschreiben. Zusätzlich können per .tmi-Datei auch Instanz-Informationen von einem Projekt zum anderen übertragen werden.

## 16.4 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 \[▶ 75\]](#) dokumentiert.

Der Kunde kann eigene Vorlagen definieren, was in dem [entsprechenden Unterabschnitt C++ Abschnitt / Assistenten \[▶ 127\]](#) 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 "TcDriverWizard" 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.

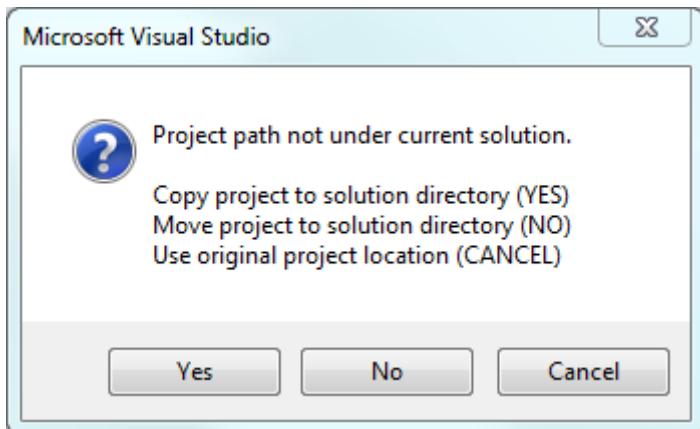
### Bestehende C++ Projekte öffnen

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 ▶ 2021](#))

#### **Modulinstanzen erstellen**

TcCOM Module können am System -> TcCOM Modulknoten erstellt werden. Siehe [dort diesbezügliche Dokumentation ▶ 321](#).

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>"
```

### 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++ \[▶ 43\]](#) 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>"
```

### 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>"
```

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, die hier dokumentiert sind.

## 16.5 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")
```

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

Beide Möglichkeiten reichen aus, um die TcCOM-Module für TwinCAT erkennbar zu machen.

Ein TcCOM-Modul wird durch seinen GUID identifiziert. Dieser GUID kann verwendet werden, um ein TcCOM-Modul zu einer TwinCAT-Konfiguration über die ITcSmTreeItem::CreateChild() Methode hinzuzufügen. Der GUID kann in TwinCAT XAE über die Eigenschaftsseite eines TcCOM-Moduls oder programmgesteuert bestimmt werden, was später in diesem Kapitel erläutert wird.

Object	Context	Parameter (Init)	Parameter (Online)	Data Area	Interfaces	Block Diagram	
Object Id:	0x01010020	<input type="checkbox"/> Copy TMI to Target					
Object Name:	Object1 (TempContr)						
Type Name:	TempContr						
GUID:	8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45						
Class Id:	8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45						
Class Factory:	TempContr						
Parent Id:	0x00000000						
Init Sequence:	PSO						

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"])
```

Denken Sie daran, dass der vInfo Parameter der Methode `ITcSmTreeItem::CreateChild()` den GUID des TcCOM-Moduls enthält, der zur Identifizierung des Moduls in der Liste aller registrierter TcCOM-Module in dem System verwendet wird.

#### **Durch hinzugefügte TcCOM-Module iterieren**

Zur Iteration durch alle hinzugefügten TcCOM-Modulinstanzen können Sie die `ITcModuleManager2` Schnittstelle verwenden. Der folgende Code-Ausschnitt zeigt, wie Sie diese Schnittstelle verwenden.

#### **Code-Ausschnitt (C#):**

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

#### **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 = destModule.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 = dataAreaNo.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)

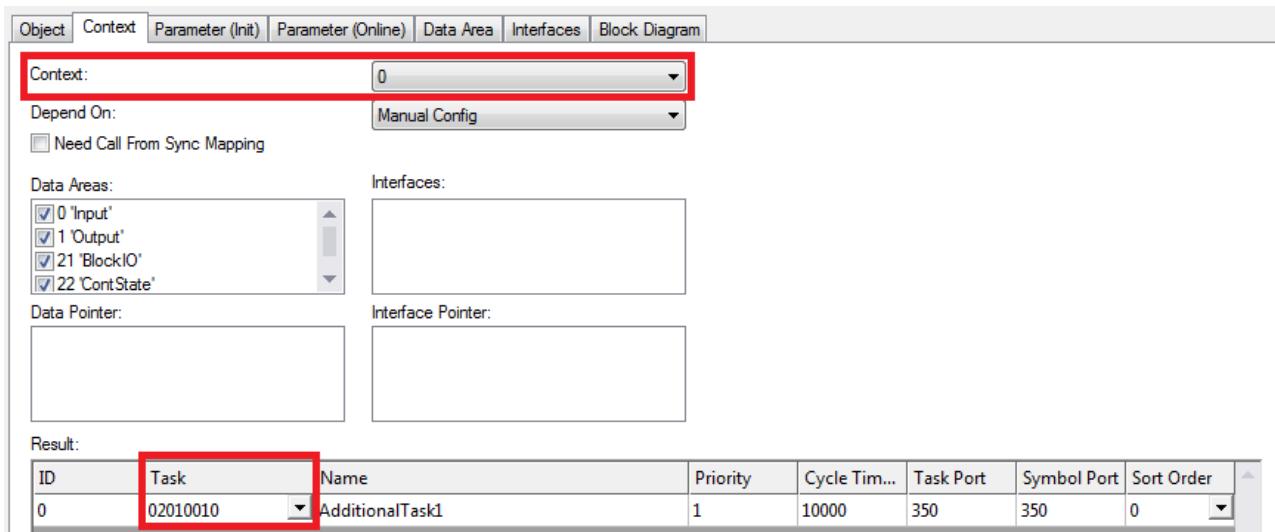
$destModule = $targetDoc.TreeItem.TcModuleInstance.Module
[System.XML.XmlElement] $dataArea = $destModule.SelectSingleNode("DataAreas/DataArea[ContextId='0' and Name='Input']")
$dataAreaNo = $dataArea.SelectSingleNode("AreaNo")
$dataAreaNo.CreateSymbols = "true"

// 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().