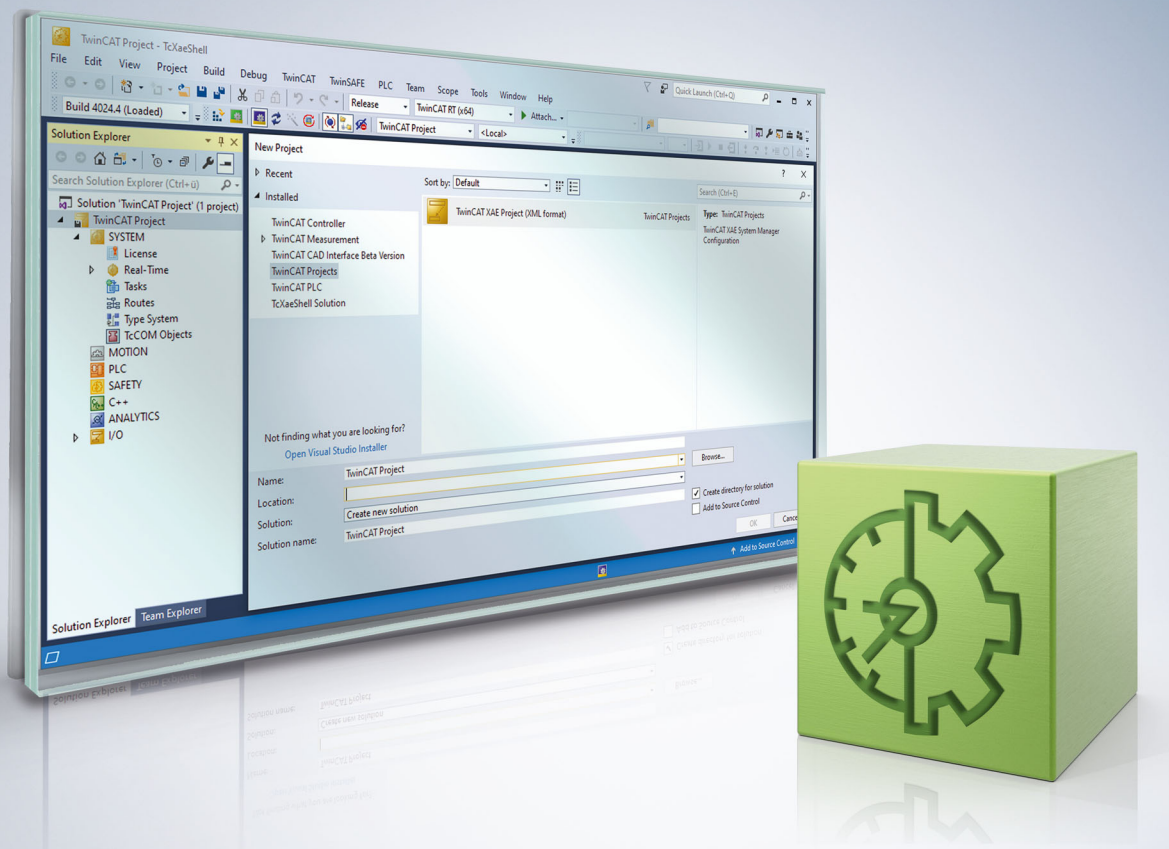
BECKHOFF New Automation Technology

Handbuch | DE

TE1400

TwinCAT 3 | Target for Simulink®



04.01.2024 | Version: 2.1.1

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

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

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

Disclaimer

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

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® 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, EP1456722, EP2137893, DE102015105702
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

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

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

Haftungsausschluss

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

Qualifikation des Personals

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

Signalwörter

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

Warnungen vor Personenschäden

⚠ GEFAHR

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

⚠ WARNUNG

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

⚠ VORSICHT

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

Warnung vor Umwelt- oder Sachschäden

HINWEIS

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

Information zum Umgang mit dem Produkt

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

1.3 Hinweise zur Informationssicherheit

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

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

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

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

2 Übersicht

TE1400 TwinCAT Target for Simulink®

Mit dem TwinCAT 3 Target for Simulink® ist es möglich, in Simulink® entwickelte Modelle in TwinCAT 3 nutzbar zu machen. Dabei können in Simulink® diverse Toolboxen, z. B. SimScape™ oder Stateflow™ oder DSP System Toolbox™ eingebunden werden. Auch eingebettete MATLAB®-Funktionsbausteine werden unterstützt. Die Modelle werden automatisch mithilfe des Simulink Coder™ in C/C++-Code übersetzt und mit dem TwinCAT 3 Target for Simulink® in TwinCAT-Objekte überführt. Diese TwinCAT-Objekte können dann in der TwinCAT-Runtime in Echtzeit ausgeführt werden. Diese TwinCAT-Objekte können sowohl TcCOM-Objekte zum direkten Instanziieren und Verknüpfen mit Echtzeit-Tasks als auch Funktionsbausteine zur Instanziierung und Verarbeitung in einem SPS-Projekt sein.

Einsatzbereiche und Anwendungsbeispiele

Die Einsatzbereiche des TwinCAT Target for Simulink® lassen sich durch folgende Schlagworte zusammenfassen:

- Rapid Control Prototyping
- Echtzeitsimulation
- SiL (**S**oftware **i**n the **L**oop)-Simulation
- HiL (**H**ardware **i**n the **L**oop)-Simulation
- Modellbasiertes Design
- Modellbasiertes Überwachen

Die folgenden Anwendungsbeispiele sollen mögliche Einsatzbereiche veranschaulichen:

- **Beispiel 1: Rapid Control Prototyping**
  Im Entwicklungsstadium der reinen Simulation in Simulink® wird ein Regler als Simulink®-Modell implementiert, welches per *Model Referencing* in das Simulationsmodell des Regelkreises eingebunden wird. Der geschlossene Regelkreis kann so zunächst in Simulation ausgelegt und getestet werden (**M**odel **i**n the **L**oop-Simulation (MiL)). Danach wird das Reglermodell unverändert per Mausklick in ein TwinCAT-Modul kompiliert, welches als Echtzeit-Regler für ein reales System arbeitet. Da als Ein- und Ausgänge Standard-Simulink®-Blöcke verwendet werden, können diese sowohl im übergeordneten Simulink®-Modell als auch im später generierten Modul in TwinCAT verwendet werden.
- **Beispiel 1a: Echtzeitsimulation einer Regelstrecke**
  Die Regelstrecke wird ebenfalls als Simulink®-Modell implementiert, das durch *Model Referencing* in das Modell des geschlossenen Regelkreises eingebunden wird. Mit dem daraus generierten TcCOM-Modul wird eine Echtzeitsimulation durchgeführt, in der ein in IEC61131-3, C++ oder Simulink® implementierter Regler getestet werden kann.
- **Beispiel 2: Echtzeitsimulation einer Maschine/Virtuelle Inbetriebnahme**
  Aus einem in Simulink® erstellten Maschinenmodell wird ein TcCOM-Modul generiert. Dieses kann verwendet werden, um ein SPS-Programm in Echtzeit testen zu können, bevor die reale Maschine angeschlossen ist (Virtuelle Inbetriebnahme). Je nach Konfiguration können so SiL- oder HiL-Simulationen durchgeführt werden. Siehe dazu auch TE1111 EtherCAT Simulation.
- **Beispiel 2a: SiL-Simulation von Anlagenteilen**
  Nach VDI/VDE 3693 Blatt 1 ist Software in the Loop (SiL) definiert als eine auf MiL- Simulation folgende Stufe, in der der Steuerungscode als Seriencode vorliegt. Der Seriencode kann in einer emulierten Steuerung ausgeführt werden und wird gegen ein Anlagensimulationsmodell getestet. Dieser Definition folgend gibt es für eine SiL-Simulation von Anlagen(-teilen) mit TwinCAT zwei Möglichkeiten:
  - Das Anlagenmodell verbleibt in Simulink® und kommuniziert über ADS mit dem Seriencode, welcher in der TwinCAT-Runtime ausgeführt wird. Siehe auch TE1410 Interface für MATLAB Simulink.
  - Das Anlagenmodell wird ebenfalls in ein TcCOM-Modul übersetzt und wird in Echtzeit ausgeführt (siehe Beispiel 1a).

- **Beispiel 2b: HiL-Simulation von Anlagenteilen**
  Nach VDI/VDE 3693 Blatt 1 ist Hardware in the Loop (HiL) definiert als eine weitergehende Test-Stufe, bei welcher der reale Ziel-Steuerungscode auf einer realen Steuerung gegen ein Anlagenmodell getestet wird. Letzteres wird in einem Simulationswerkzeug ausgeführt, welches als Busteilnehmer funktioniert und somit die realen Kommunikationsnetzwerke des Automatisierungssystems verwendet, um mit der realen Steuerung zu kommunizieren.
  Dieser Definition folgend werden das Modell der Anlage bzw. der Anlagenteile in TcCOM-Module überführt und auf einem zweiten Industrie-PC unter Berücksichtigung der Echtzeitanforderungen zur Ausführung gebracht. Unter Verwendung der Function TE1111 EtherCAT Simulation wird dieser IPC so konfiguriert, dass dieser das gespiegelte Prozessabbild der realen Steuerung bereitstellt. Somit ist es möglich, mit der realen Steuerung unter Verwendung der realen Konfiguration mit dem „Simulations-IPC“ in harter Echtzeit zu kommunizieren.
- **Beispiel 3: Modellbasierte Überwachung von Anlagenteilen/Komponenten**
  Oft sind Messgrößen interessant, welche nicht direkt zugänglich sind, oder deren Messung hohen Aufwand/Kosten verursachen. Durch Nutzung eines physikalisch repräsentativen Modells mit messbaren Eingangsgrößen, können nicht-messbare Größen dennoch bestimmt werden. Ein Beispiel ist die Temperaturerfassung an baulich nicht zugänglichen Stellen, wie z. B. der Permanentmagnettemperatur eines Elektromotors. Auf Basis eines thermischen Modells des Motors kann diese anhand von sekundären Größen, wie elektrischem Strom, Drehgeschwindigkeit und Kühltemperatur, geschätzt werden.

Weitere Informationen

Technische Kurzvideos

- TwinCAT Target for Simulink

Produktbeschreibungen

- https://www.beckhoff.com/TE1400

Kundenapplikationsvideos

- Kundenanwendungen im Überblick
- Success Story Vintecc bv
- Success Story Magway

Website zu MATLAB® und Simulink® mit TwinCAT 3: http://www.beckhoff.com/matlab

3 Bis Version 1.2.xxxx.x

TE1400 Target for Simulink® Versionen geringer als 1.2.xxxx.x unterstützen MATLAB R2010b bis MATLAB R2019a.

TE1400 Target for Simulink® Versionen höher als 2.x.xxxx.x unterstützen MATLAB R2019a und höher.

3.1 Installation

Systemvoraussetzung

Es gelten für das Target für MATLAB®/Simulink® zunächst dieselben Anforderungen wie für TwinCAT 3 C/C++. Für eine detaillierte Beschreibung der TwinCAT 3 C/C++-Anforderungen sei auf das Handbuch TwinCAT 3 C++, Kapitel 4 „Anforderungen“, verwiesen.

Im Folgenden werden diese nur stichpunktartig und nicht in aller Ausführlichkeit aufgegriffen.

Auf dem Engineering-PC

- Microsoft Visual Studio 2010 (mit Service Pack 1), 2012, 2013, 2015 oder 2017 Professional, Premium, Ultimate oder Community Edition
  - Installation unter Windows immer mit Rechtsklick **run as admin…**
  - Für Visual Studio 2015 bei der Installation die Checkbox Visual C++ selektieren
  - Für Visual Studio 2017 “Desktop development with C++” manuell auswählen
- Microsoft “Windows Driver Kit” Version 7.1.0 (nur notwendig für TwinCAT Versionen kleiner als TwinCAT 3 build 4024.0)
  - Es genügt die „Build Environments“ zu installieren.
  - Die Umgebungsvariable setzen (Variablenname WINDDK7, Variablenwert <Installationsverzeichnis> z.B. C:\WinDDK\7600.16385.1)
- TwinCAT 3 XAE

Auf dem Laufzeit-PC

- IPC oder Embedded CX PC mit Microsoft Betriebssystem basierend auf „Windows NT Kernel“ (Win XP, Win 7 und entsprechende embedded Versionen, Win 10)
- TwinCAT 3 XAR
  - TwinCAT 3.0 unterstützt auf dem Target nur 32-Bit-Betriebssysteme
  - TwinCAT 3.1 unterstützt 32 Bit und 64 Bit Betriebssysteme. Ist das Target ein x64-System, müssen die erstellten Treiber signiert werden. Das TE1400 unterstützt OS Treibersignierung. Sehen Sie dazu „x64: Treibersignierung“ im Handbuch TwinCAT 3 C++

Zusätzlich zu den obigen Anforderungen, die aus den Anforderungen von TwinCAT 3 C/C++ stammen, werden auf dem Engineering PC benötigt:

- MATLAB®/Simulink® R2010b bis einschließlich R2019a. Ab einschließlich R2019a wird die Nutzung von TE1400 Version 2.x.xxxx.x empfohlen.
- Simulink Coder™ (in MATLAB®-Versionen vor R2011a: Real-Time Workshop®)
- MATLAB Coder™ (in MATLAB®-Versionen vor R2011a: Teil des Real-Time Workshop®)
- Installation des TE1400 Target for MATLAB®/Simulink®

Installationsanleitung

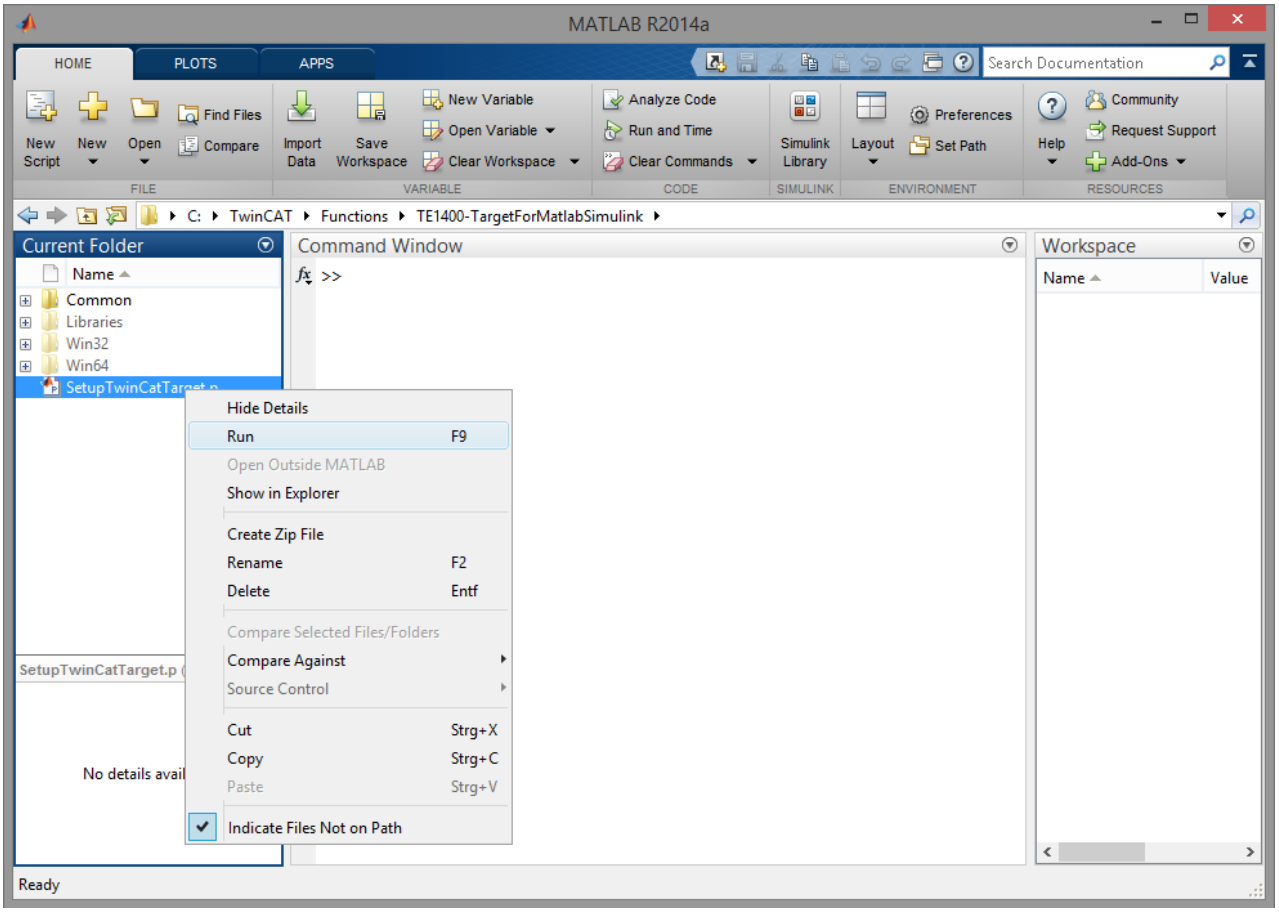
✓ Installieren Sie eine der unterstützten Visual Studio-Versionen, falls nicht bereits vorhanden. Beachten Sie die Installation der C++-Komponenten.

1. Starten Sie das TwinCAT 3 Setup, falls nicht bereits vorhanden.

⇨ Sollte eine Visual Studio- sowie TwinCAT-Installation bereits vorhanden sein, die Visual Studio Version jedoch nicht den oben genannten Anforderungen entsprechen (z. B. Visual Studio Shell oder Visual Studio ohne Visual C++), müssen Sie zunächst eine geeignete Visual Studio Version installieren (ggf. Visual C++ nachinstallieren). Danach müssen Sie das TwinCAT 3-Setup ausführen, um TwinCAT 3 in die neue (oder veränderte) Visual Studio-Version zu integrieren.

2. Installieren Sie, wenn erforderlich, das Microsoft Windows Driver Kit (siehe Installation "Microsoft Windows Driver Kit (WDK)" im Handbuch TwinCAT 3 C/C++).
   Die Reihenfolge, wann das Windows Driver Kit installiert worden ist, ist unerheblich.
3. Falls noch keine **MATLAB**®-Installation auf Ihrem System vorhanden ist, installieren Sie diese. Die Reihenfolge, wann MATLAB® installiert worden ist, ist unerheblich.
4. Starten Sie das Setup *TE1400-TargetForMatlabSimulink* zur Installation des TE1400.
   Die Installation des TE1400 erfolgt in den TwinCAT-Ordner, d. h. sie ist losgelöst von der MATLAB®-Installation. Das Verknüpfen einer auf dem System vorhandenen MATLAB®-Version mit dem TE1400 erfolgt durch Ausführung von Punkt 6.
5. Starten Sie MATLAB® als Administrator und führen Sie *%TwinCAT3Dir%..\Functions\TE1400-TargetForMatlabSimulink\SetupTwinCatTarget.p* in MATLAB® aus.

⇨ Es öffnet sich ein Fenster zur Einrichtung. Siehe dazu den folgenden Abschnitt.



- Das p-file verknüpft die verwendete MATLAB®-Version mit dem TE1400. Wenn eine neue MATLAB®-Version auf dem System installiert wird, muss das p-file in der neuen Version ausgeführt werden.
- Wenn eine neue TE1400 Version über eine vorhandene TE1400 Version installiert wird, sollte das p-file ebenfalls nochmal ausgeführt werden.

User Account Control

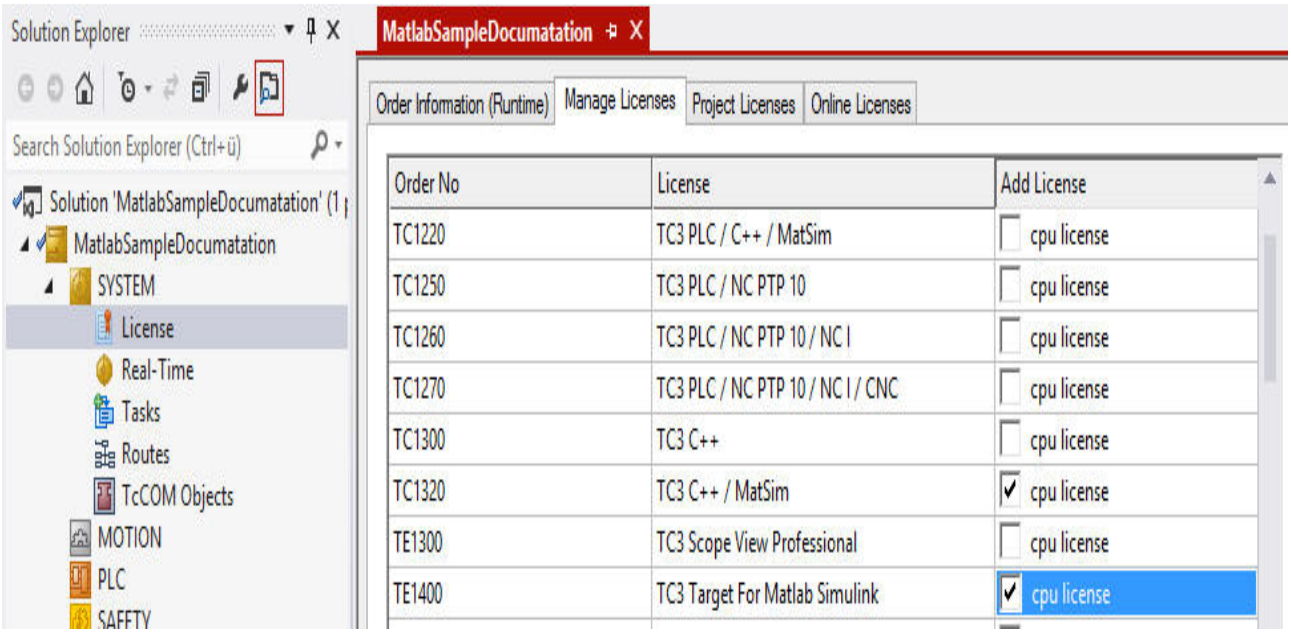
Wenn MATLAB® in einem System mit aktiviertem User Account Control (UAC) ohne Administratorbefugnis ausgeführt wird, kann der MATLAB®-Pfad nicht dauerhaft gespeichert werden. In diesem Fall muss nach jedem Start von MATLAB® SetupTwinCatTarget.p ausgeführt werden, da sonst einige Dateien für die Generierung von TwinCAT-Modulen nicht gefunden werden können.

Treiber-Signierung für Targets mit x64-Betriebssystem

Falls Sie als Laufzeit-PC ein x64-Betriebssystem nutzen möchten, ist eine Signierung der Treiber notwendig. Details finden Sie dazu im Handbuch TC3 C++ unter Treibersignierung.

3.2 Lizenzen

Um die gesamte Funktionalität des TE1400 Target für MATLAB®/Simulink® nutzen zu können, sind zwei Lizenzen erforderlich (siehe Bestellung und Aktivierung von TwinCAT-3-Standardlizenzen).



Erforderliche Lizenzen für TE1400

TE1400: TC3 Target-For-Matlab-Simulink (Modulgenerator-Lizenz)

Diese Lizenz wird für das **Engineeringsystem** für die Modulgenerierung aus MATLAB®/Simulink® benötigt. Zu Testzwecken kann der Modulgenerator des TE1400 im Demomodus auch ohne Lizenz genutzt werden.

Für dieses Produkt ist keine 7-Tage-Testlizenz mit allen Funktionen verfügbar.

Einschränkungen in der Demoversion

Der Modulgenerator hat ohne Lizenz folgende Einschränkungen.

Erlaubt sind Modelle mit maximal :

- 100 Blöcken
- 5 Eingangssignalen
- 5 Ausgangssignalen

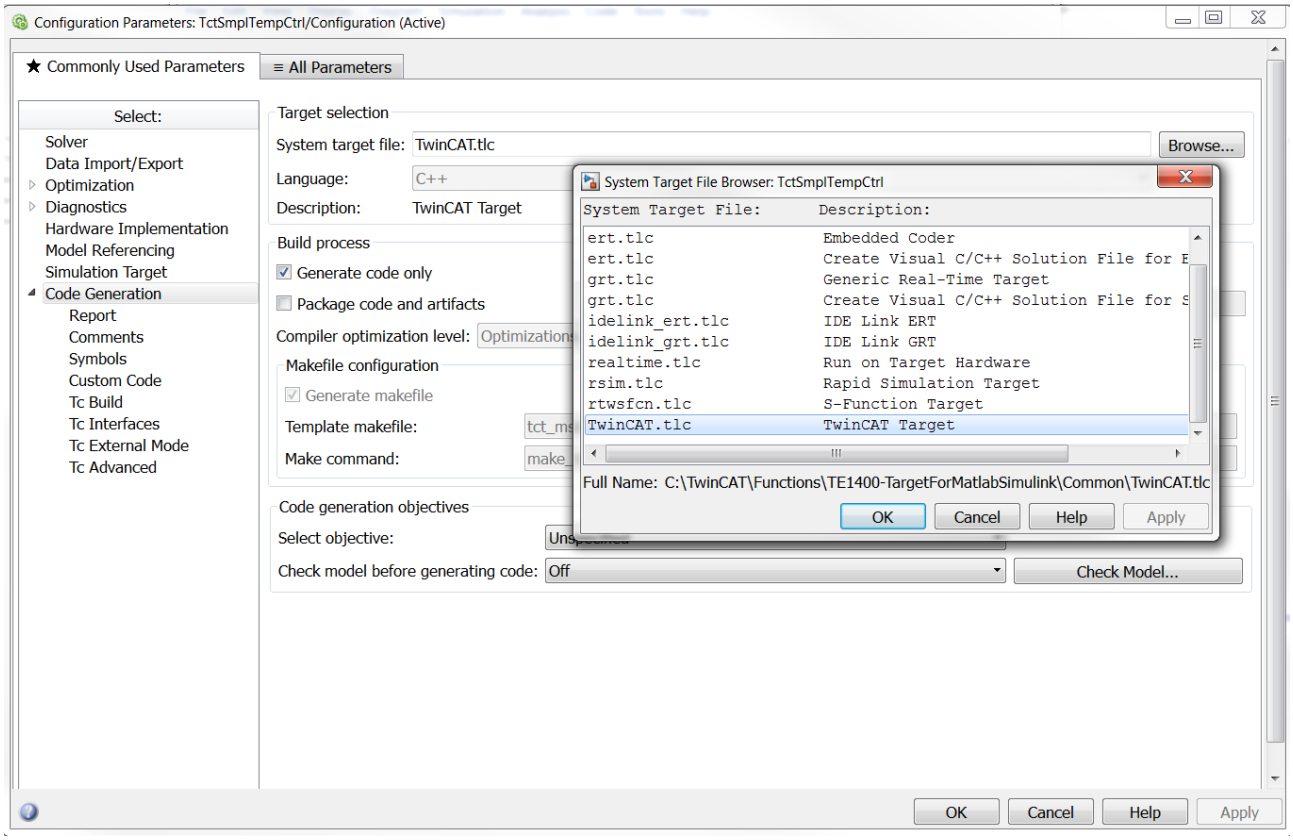
Mit einer Demolizenz erzeugte Module dürfen nur für nichtkommerzielle Zwecke genutzt werden!

TC1320/TC1220: TC3 [PLC /] C++ / MatSim (Laufzeitlizenz)

Die Lizenz TC1320 (bzw. TC1220 mit SPS-Lizenz) wird benötigt, um eine TwinCAT-Konfiguration mit einem aus Simulink® generierten Modul zu starten. Ohne aktivierte Lizenz kann das Modul und damit auch das TwinCAT-System nicht gestartet werden. In dem Fall erhält man Fehlermeldungen bezüglich der Lizenzverletzung. Man kann eine 7-Tage-Testlizenz erzeugen, die erste Tests ohne den Kauf der Lizenz ermöglicht.

3.3 Quickstart

Konfiguration des Simulink® - Modells



Der Zugriff auf die Codereinstellungen kann über den Model Explorer im Menü **View** der Simulink-Umgebung, über **Code Generation** (früher **Real-Time Workshop**) **> Options** im Menü **Tools** oder über den **Configuration Parameters** -Dialog erfolgen. Wählen Sie in der Baumansicht zunächst **Configuration > Code Generation**. Öffnen Sie darunter die Registerkarte **General** und wählen Sie *TwinCAT.tlc* als „System target file“. Alternativ kann mit der Schaltfläche **Browse** ein Auswahlfenster geöffnet und darin das **TwinCAT Target** als Zielsystem ausgewählt werden.

Für die Echtzeitfähigkeit des Simulink-Modells muss außerdem in den Solver-Einstellungen ein Fixed-Step-Solver konfiguriert sein.

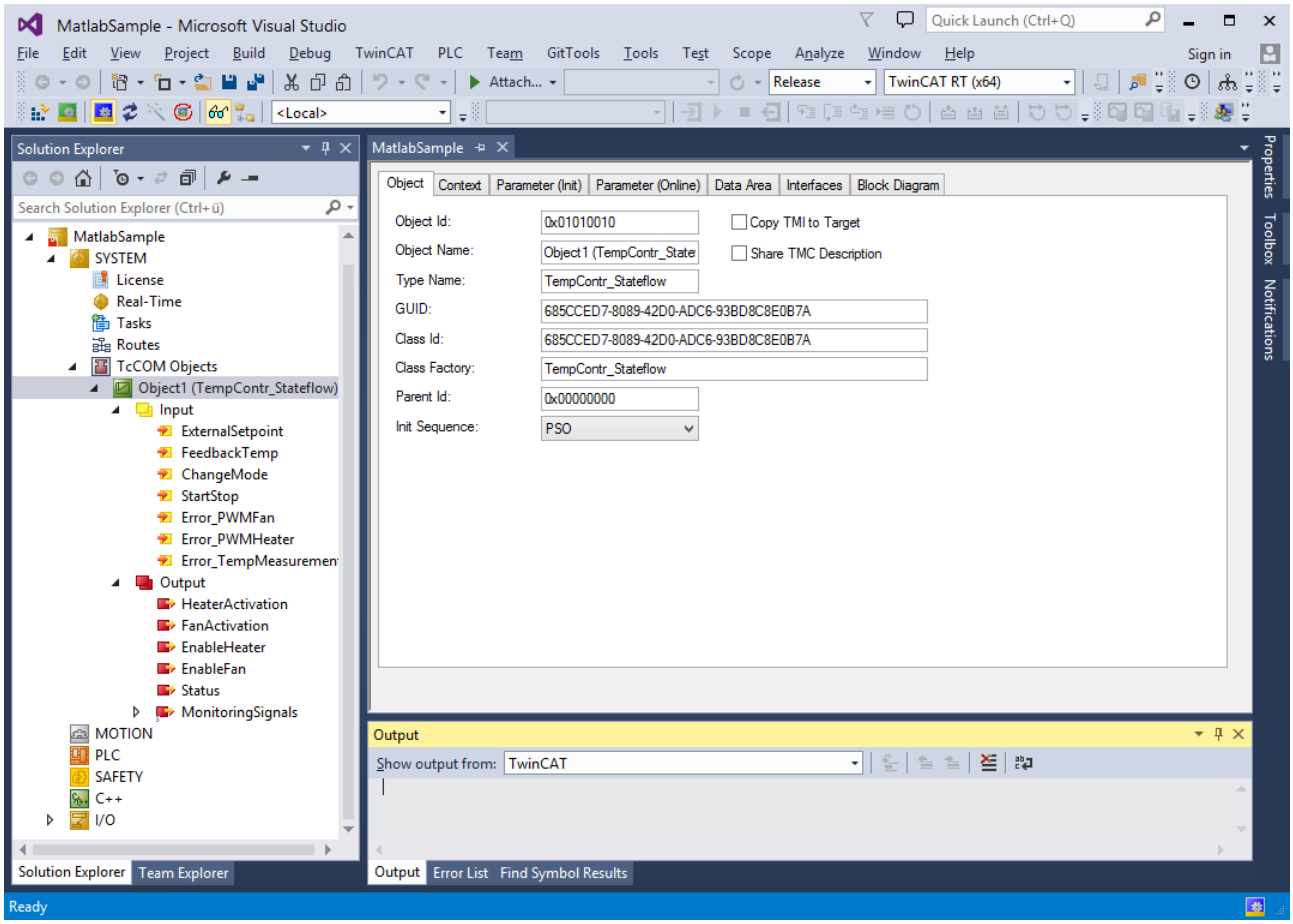
Generieren eines TcCOM Moduls aus Simulink

Das Generieren des C++ Codes bzw. des TcCOM Moduls kann mit der Schaltfläche **Build** (bzw. **Generate code**) im unteren Teil des Fensters für die Codegenerator-Optionen gestartet werden. Ist die Option **Publish module** unter **TC Build** (Defaulteinstellung) aktiviert, wird sofort nach dem Generieren des C++ Codes der Build-Prozess zur Erzeugung ausführbarer Dateien gestartet und ein TcCOM Modul erstellt. Ansonsten stoppt der Modulgenerator nach dem Generieren des C++ Codes und der Projektdatei für Visual Studio™. Weitere Informationen zu diesem Punkt finden Sie unter Publish Module [▶ 23].

Integration des Moduls in TwinCAT 3

Nach dem Exportieren des Moduls mit "Publish"

Wurde vor der Modulgenerierung die Option **Publish Module** aktiviert, ist das Modul bereits in kompilierter Form verfügbar. Eine TwinCAT Module Class (TMC file) wurde dabei erzeugt und kann im Projekt direkt instanziiert werden. Eine TwinCAT Module Instanz (TMI) wird im Folgenden als TcCOM-Objekt oder Modulinstanz bezeichnet.

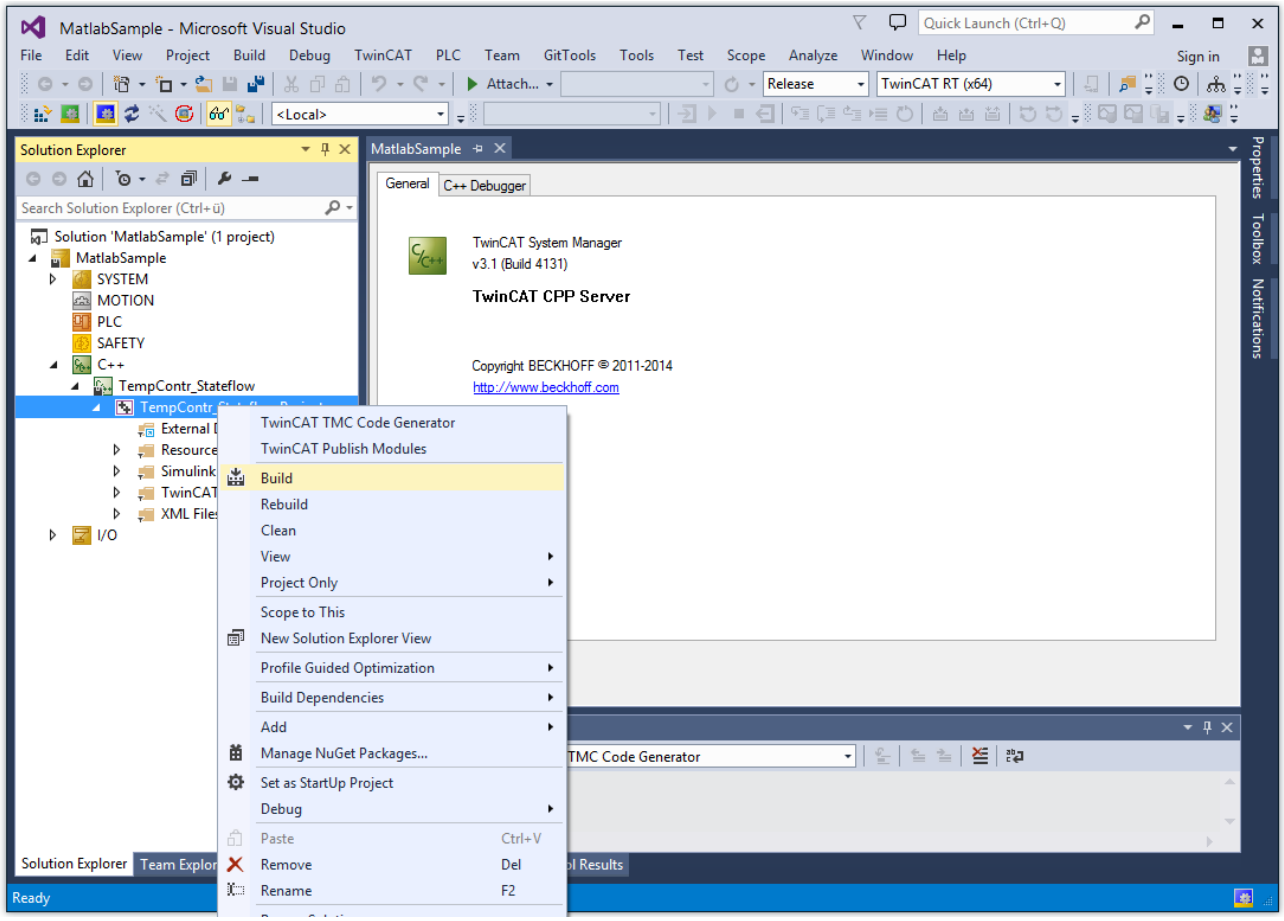


Instanzen des generierten Moduls können beliebig häufig in ein TwinCAT3-Projekt eingebunden werden. Üblicherweise werden TcCOM-Objekte über das Kontextmenü **Add New Item** an den Knoten **TcCOM-Objects** angehängt. Durch Anwahl dieses Kontextmenüs erhalten Sie eine Auswahlliste der auf dem System verfügbaren Module. Die von Simulink generierten Module finden sich unter **TE1400 Module Vendor > Generated Modules**.

Übersetzen des Codes ohne „Publish“

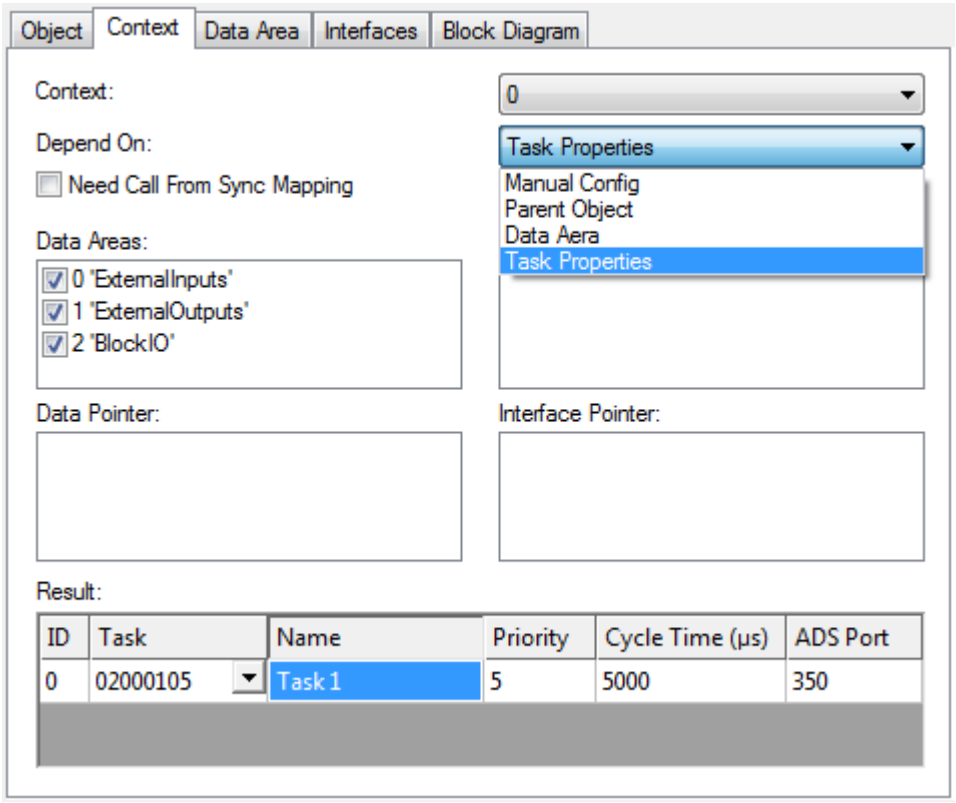
Wurde vor der Modulgenerierung die Option **Publish Module** deaktiviert, muss der zum Modul gehörende, generierte C/C++-Code noch übersetzt werden, bevor er ausgeführt werden kann.

Das C++ Projekt kann über das Kontextmenü des C++-Knotens mit **Add Existing Item** in das TwinCAT-Projekt eingefügt werden. Die C++ Projektdatei befindet sich im Build-Verzeichnis "<MODELNAME>_tct" und trägt den Namen des Moduls mit der Dateiendung *.vcxproj*. Danach kann das Modul in der TwinCAT-Entwicklungsumgebung (XAE) erstellt werden:



Über das Kontextmenü des übergeordneten Knotens des C++-Projektes können auch hier mehrere Instanzen des Moduls erstellt werden, die unterhalb des Projekt-Knotens aufgelistet werden. Weitere Informationen über den Build-Prozess von C++-Projekten in der TwinCAT-Entwicklungsumgebung (XAE) und über die Instanziierung von so erstellten Modulen finden sich im Abschnitt „Ein TwinCAT3 C++ Projekt erstellen“.

Zyklischer Aufruf durch eine Echtzeit-Task



Unter der Registerkarte **Context** der Modulinstanz findet man alle Kontexte des Moduls, welche jeweils einer Echtzeittask zugewiesen werden müssen. Bei der Einstellung **Depend on: Task Properties** werden automatisch Tasks zugewiesen, bei denen Zykluszeit und Priorität den angezeigten Werten entsprechen. Wenn es keine passenden Tasks gibt oder die Einstellung **Depend on: Manual Config** gewählt wurde, können unter **System Configuration > Task Management** Tasks angelegt werden. Weitere Informationen zum zyklischen Aufruf der Modulinstanzen finden Sie im Abschnitt „Cyclic Call [▶ 42]“.

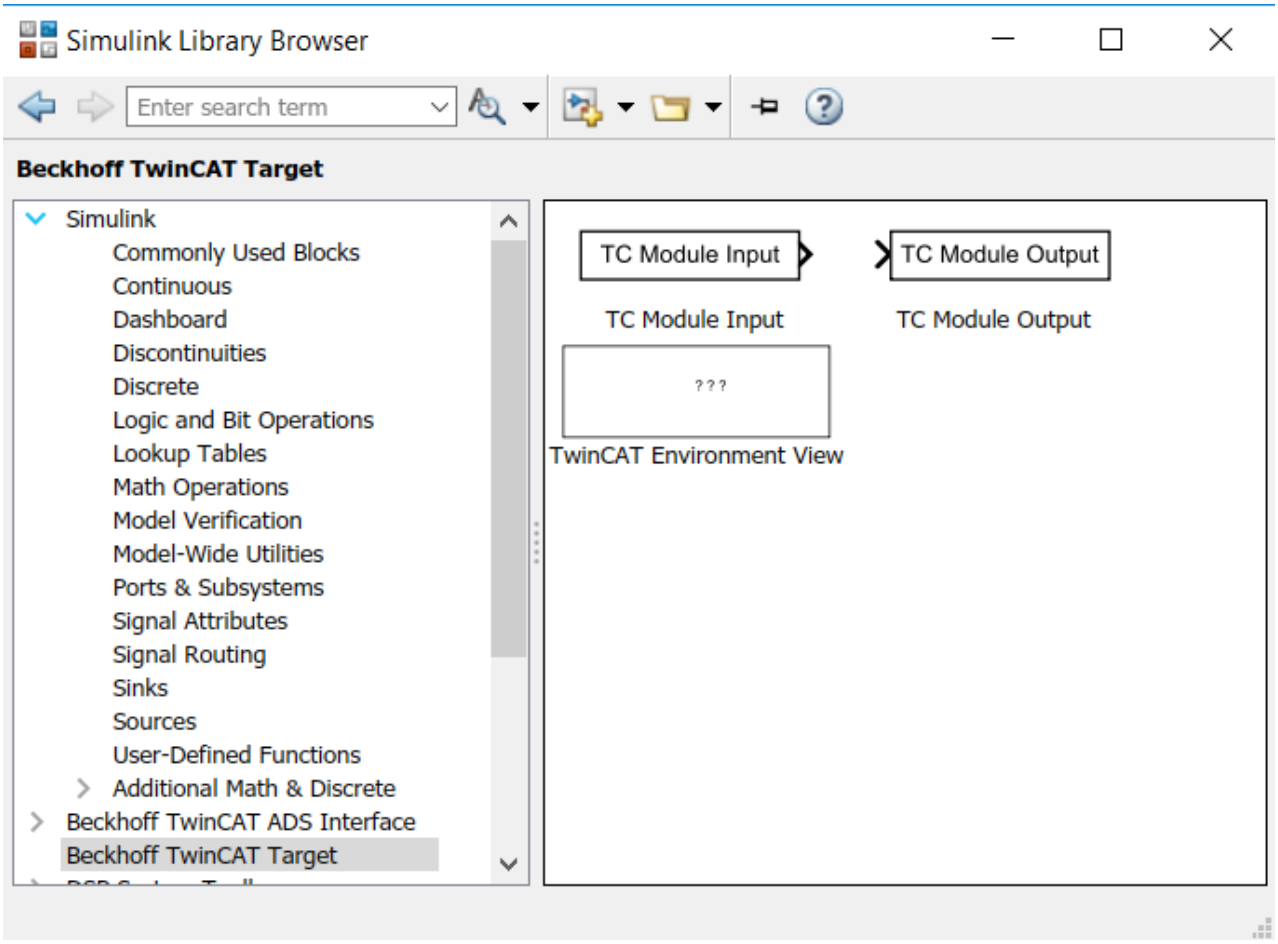
Datenaustausch mit anderen Modulen oder Feldbusgeräten

Unterhalb des Modulinstanz-Knotens in der TwinCAT-Entwicklungsumgebung können die Prozessabbilder der Ein- und Ausgänge des Moduls aufgeklappt werden. Hier finden Sie alle Ports, die im Simulink-Modell mit Hilfe der Blöcke **In1** und **Out1** (Bestandteile der Standard-Simulink-Bibliothek) definiert wurden. Alle Signale innerhalb dieser Prozessabbilder können über das Kontextmenü **Change Link** mit Signalen anderer Prozessabbilder verknüpft werden.

3.4 TwinCAT Library in Simulink®

In Simulink® können (nicht zwingend!) *TwinCAT-spezifische* Ein- und Ausgangsblöcke verwendet werden, um die mit diesen Blöcken verbundenen Signale/Busse als Inputs bzw. als Outputs im späteren TcCOM in TwinCAT festzulegen. Ein allgemeingültiger Weg ist es ebenfalls, die Standard Input Ports (In) und Output Ports (out) von Simulink® zu verwenden. Dies ist in der Regel auch der *best practice* Weg, es sei denn, es werden die unten beschriebenen Zusatzfunktionen der TwinCAT-spezifischen Ein- und Ausgangsblöcke benötigt.

Die TwinCAT-spezifischen Ein- und Ausgangsblöcke befinden sich im
Library Browser > **Beckhoff TwinCAT Target**.



Wenn Sie die von Beckhoff bereitgestellten Input- und Output-Blöcke verwenden, erhalten Sie folgende zusätzlichen Funktionalitäten gegenüber den Standard Simulink®-Input und -Output Ports:

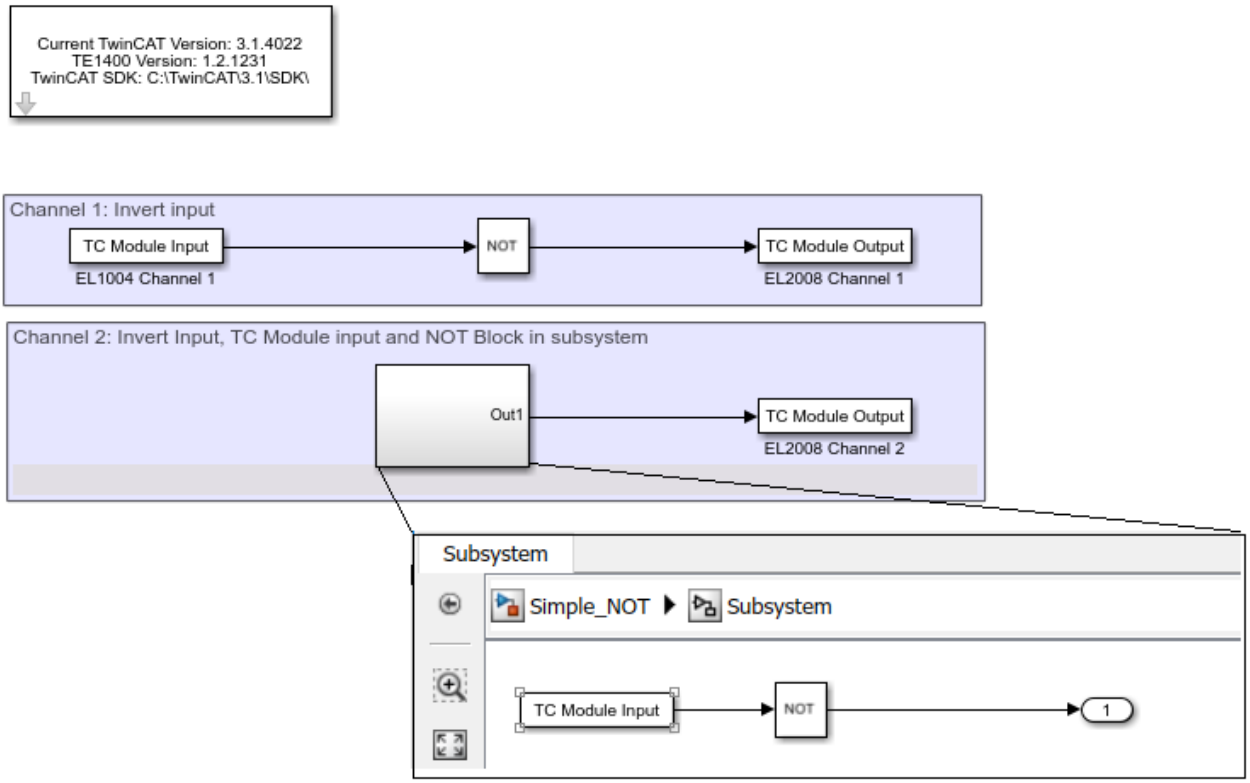
- Sie können Signale und Busse auch aus Subsystemen direkt als Input oder Output für das TcCOM definieren, ohne die Signale/Busse zuerst aus dem Subsystem in das oberste System zu führen.
- Sie können (nicht zwingend) in den Blockparametern ein automatisches Mapping zu anderen TcCOM oder I/Os hinterlegen, sodass direkt bei der Instanziierung des TcCOM das Mapping automatisiert ausgeführt wird.
- Sie können Initialwerte für Eingänge verwenden. Setzen Sie dazu den Wert Value der Tc Module Inputs auf einen beliebigen Wert.

Bei der Nutzung des automatischen Mappings ist zu beachten, dass Sie bei mehrfacher Instanziierung des TcCOM in TwinCAT einen Mapping-Konflikt erhalten, den Sie durch händisches Mappen wieder auflösen müssen. Entsprechend ist bei Mehrfachinstanziierungen diese Option nicht zu empfehlen.

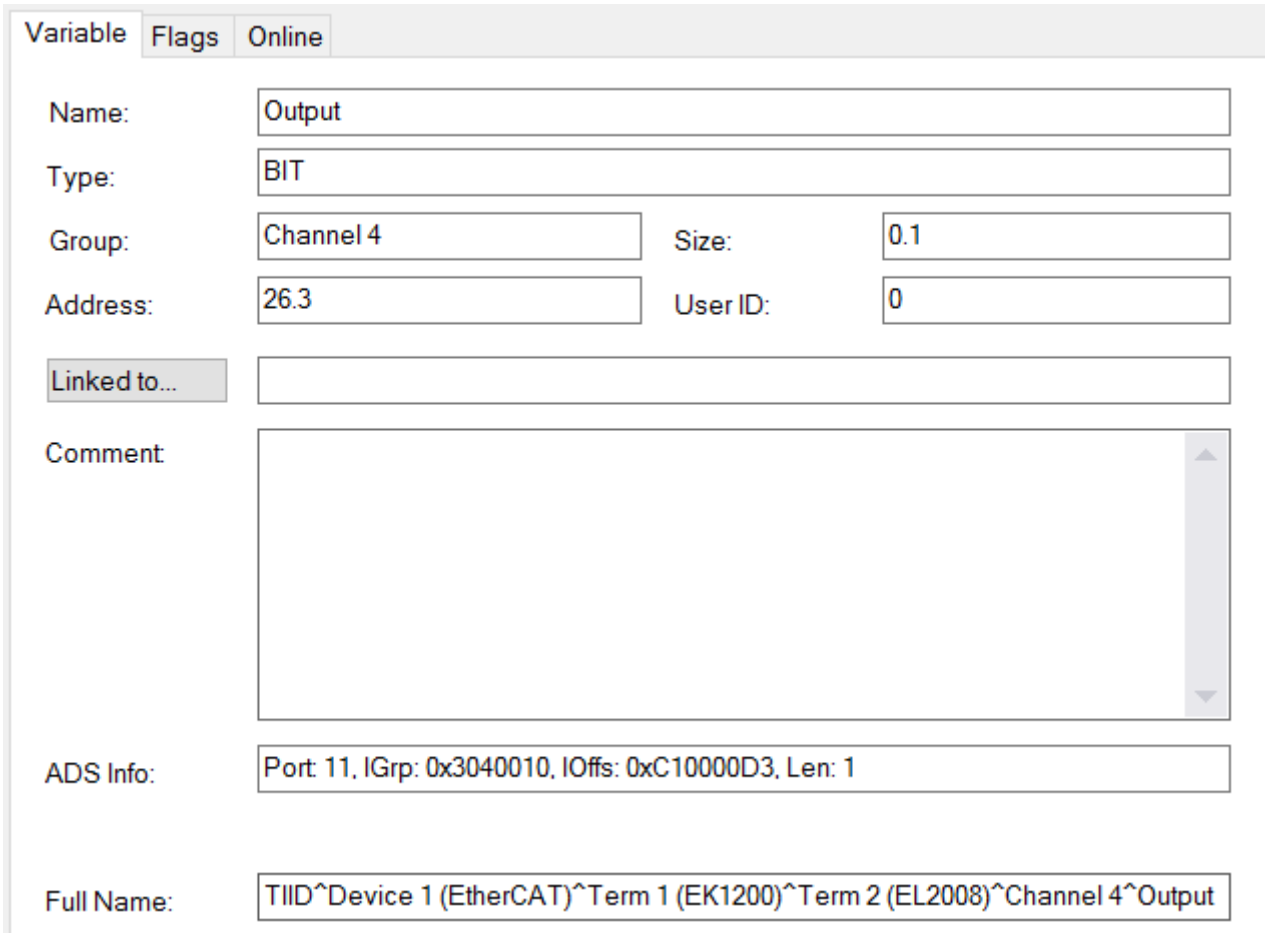
Neben den TwinCAT-spezifischen Input- und Output-Blöcken wird auch ein TwinCAT Environment View Block bereitgestellt. Dieser kann in der Simulink®-Umgebung genutzt werden, um einfach eine Anzeige über die TwinCAT und TE1400 Versionen auf dem System zu erhalten.

Beispiel

Ein Simulink®-Modell wird erstellt, welches zwei Eingänge negiert wieder ausgibt. Ein Eingang wird dabei in einem Subsystem platziert, siehe Abbildung unten.

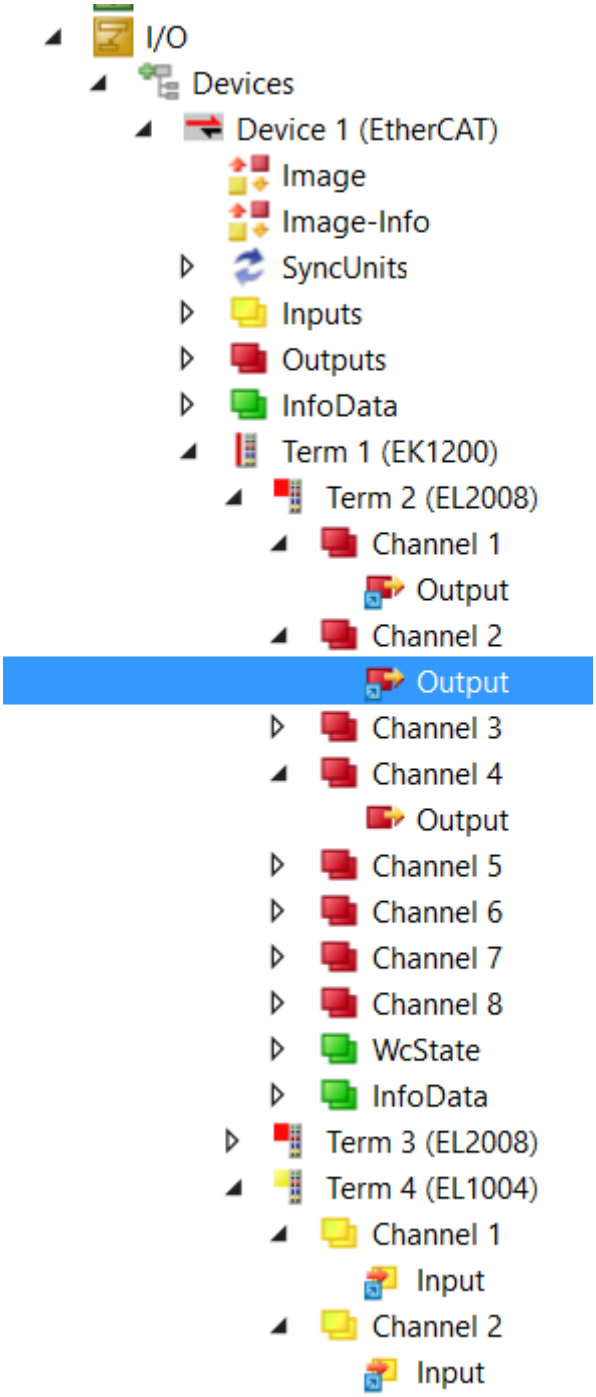


Die Ein- und Ausgänge des Modells werden über die Eigenschaften der TC Module Input und Output automatisch auf digitale Ein- und Ausgänge gemappt. Die dazu notwendigen Tree Items finden Sie in TwinCAT 3, indem Sie den gewünschten Input oder Output anwählen und dann im Tab **Variable** unter **Full Name** den String kopieren.



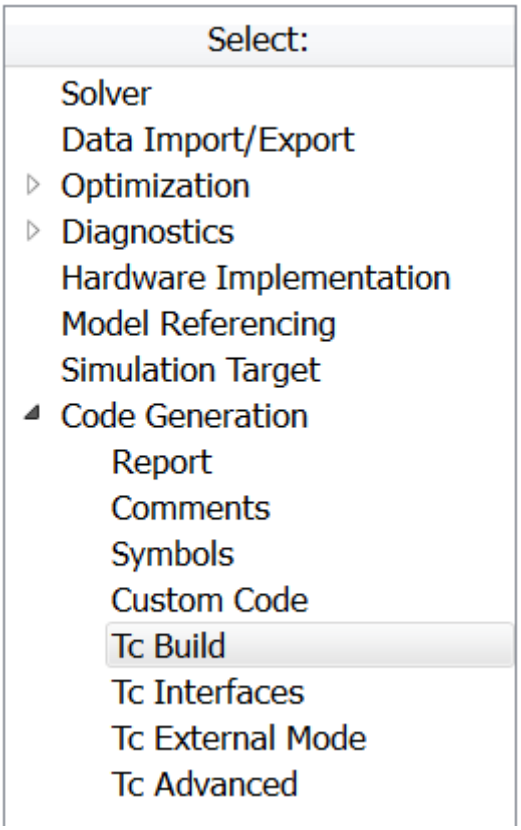
Eine Liste der Kurzformen für den schnellen Zugriff finden Sie in der Dokumentation des **Automation Interface > API > ITcSysManager > ITcSysmanager::LookupTreeItem**.

Wenn das oben beschriebene Simulink®-Modell übersetzt und in TwinCAT 3 eingebunden wird, wird automatisch ein Mapping zu den entsprechenden Inputs und Outputs hergestellt. Die automatisch generierten Mappings werden zur Unterscheidung zum händischen Mapping mit einem blauen Symbol versehen, während händische Mapping-Symbole weiß erscheinen.



3.5 Parametrierung der Codegenerierung in Simulink

Innerhalb von MATLAB®-Simulink® kann eine Vielzahl von Einstellungen zur Konfiguration des zu generierenden TcCOM Moduls vorgenommen werden. Dazu wird die Baumstruktur unter Code Generation um die Einträge Tc Build, Tc Interfaces, Tc External Mode und Tc Advanced erweitert. Viele Parameter können in TwinCAT 3 auf Ebene der Modul-Instanzen wieder verändert werden, siehe dazu Anwendung von Modulen in TwinCAT [▶ 40].



Eine Erläuterung der sich darunter befindenden Einstellungsmöglichkeiten erfolgt im Folgenden.

Tooltips

Verweilt der Mauszeiger kurze Zeit über den Textfeldern der Dialogfenster erscheint eine ausführlichere Beschreibung der Option als Tooltip (Pop-up-Fenster).

3.5.1 Modulgenerierung (Tc Build)

Mit Hilfe des Publish-Mechanismus können TwinCAT-C++-Projekte für mehrere TwinCAT-Plattformen übersetzt und in ein zentrales Publish-Verzeichnis exportiert werden. Im ersten Schritt werden die Module für alle ausgewählten Plattformen gebaut. Danach werden alle zur Instanziierung und Ausführung des Moduls unter TwinCAT 3 benötigten Dateien in das Publish-Verzeichnis kopiert.
"TC3 Module exportieren“ unter **TC3 Engineering > C/C++ > Module-Handhabung** beschreibt, wie der Publish-Mechanismus auf TC3-C++-Module angewendet wird. Im Folgenden wird beschrieben, wie Simulink konfiguriert werden muss, um TwinCAT Module direkt nach Generierung des Codes mit Hilfe des Publish-Mechanismus zu exportieren.

Publish-Verzeichnis

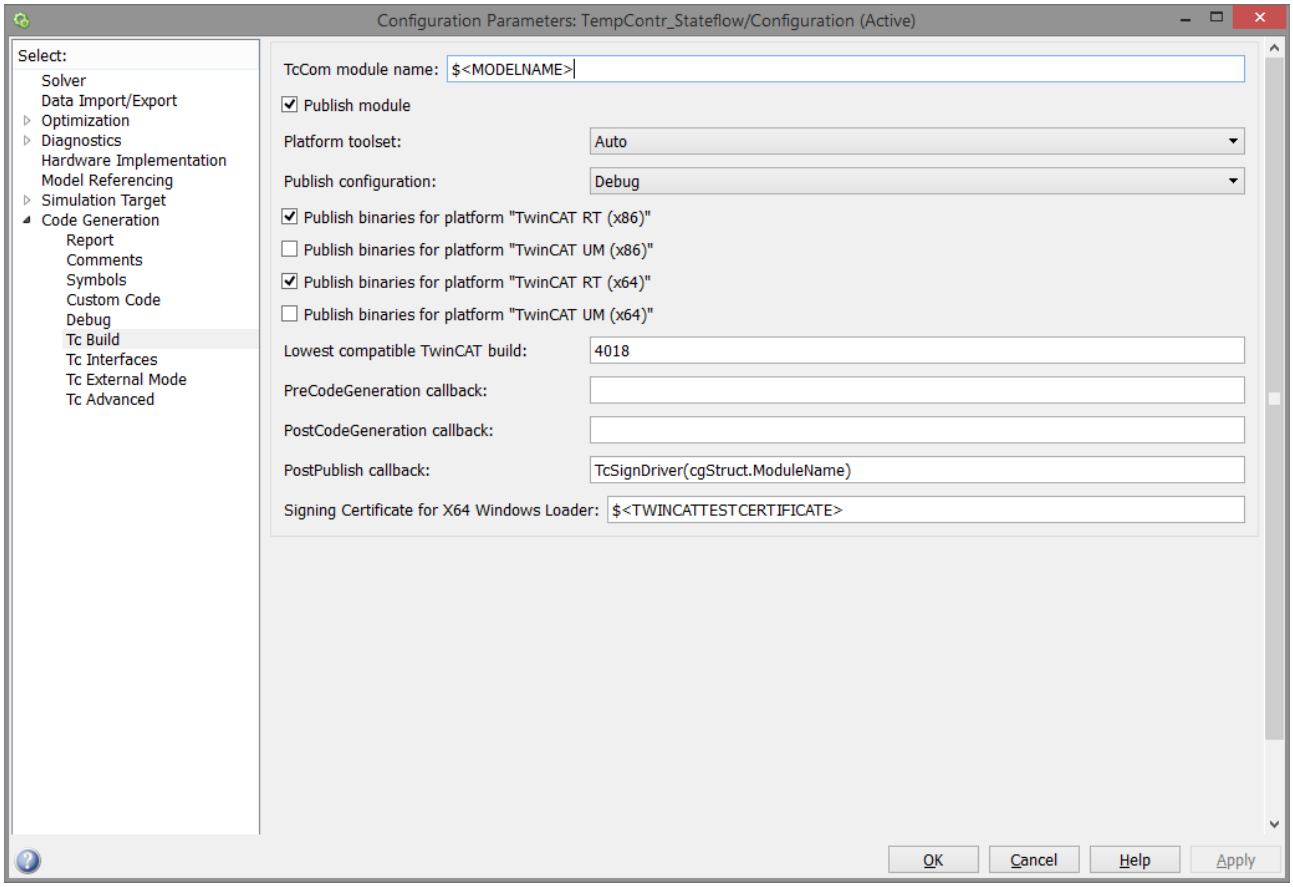
Die Dateien exportierter Module werden in das Verzeichnis *%TwinCat3Dir%CustomConfig\Modules\<MODULENAME>* kopiert. Zur Instanziierung des Moduls auf einem anderen Entwicklungsrechner, kann dieser Ordner in das entsprechende Verzeichnis dieses Rechners kopiert werden.

Anwendung

Sinnvollerweise werden Module dann publiziert, wenn sie nur noch selten geändert werden und sie in mehreren TwinCAT-Projekten verwendet werden. Sonst ist es möglicherweise effizienter, das gesamte C++-Projekt in das TwinCAT-Projekt zu integrieren, z. B. wenn sich das Simulink-Modell noch in der Entwicklung befindet, wodurch regelmäßige Änderungen zu erwarten sind, oder das Modul nur in einem speziellen TwinCAT-Projekt verwendet wird.

Konfiguration in Simulink

Der Publish-Mechanismus kann unter **Tc Build** konfiguriert werden: (Export Optionen für TwinCAT Module)



Publish module:

- **deaktiviert**: Der Modulgenerator wird nach Generierung des C++ Projekts angehalten. Das generierte C++-Projekt muss manuell übersetzt werden, um das Modul in TwinCAT 3 ausführen zu können. Das kann nach dem Einbinden des generierten C++-Projektes in das TwinCAT Projekt direkt aus der TwinCAT-Entwicklungsumgebung erfolgen.
- **aktiviert**: Nach dem Generieren des C++-Projektes wird automatisch das „Publish“ ausgeführt. Danach ist das Modul auf dem Entwicklungsrechner in kompilierter Form für alle TwinCAT-Projekte verfügbar und kann in der TwinCAT-Entwicklungsumgebung (XAE) direkt instanziiert werden. Die weiteren Publish-Einstellungen betreffen die Zielplattformen, auf denen das Modul lauffähig sein soll. Diese Einstellungen können wegen der sequentiellen Erstellung (Build) für die verschiedenen Plattformen die Dauer der Modulgenerierung signifikant beeinflussen.

> **„Generate Code only“ Option**
>
> Die Option **Generate code only** (im Bereich **Build process** des Fensters für die Codereinstellungen **Code Generation**) hat wegen der Verwendung des TwinCAT-Publish-Mechanismus anstelle des Make-Mechanismus von MATLAB keine Funktion.

Platform toolset:

Erlaubt die Auswahl eines bestimmten „Platform toolset“ (Compiler) zum Bauen der Modul-Treiber. Die auswählbaren Optionen hängen von den auf dem System installierten VisualStudio-Versionen ab. Ist die Option **Auto** ausgewählt, wird automatisch ein Compiler ausgewählt.

Publish configuration:

Um ein Debuggen in TwinCAT 3 im exportierten Block Diagramm zu ermöglichen, muss hier **Debug** gewählt werden. Ist ein Debuggen nicht notwendig, z. B. in einer Release Version, kann hier **Release** ausgewählt werden.

- **Publish binaries for platform „<PLATFORMNAME>“:**

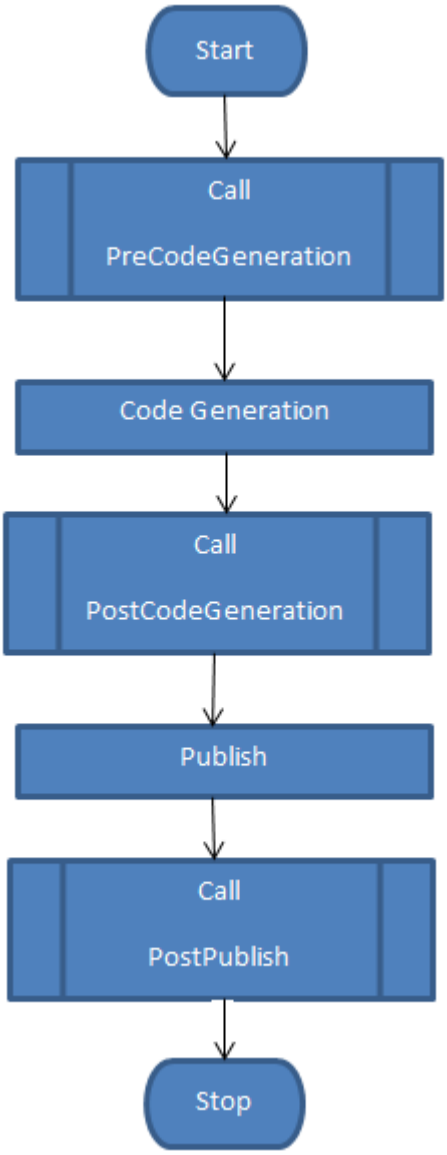
- Hier müssen alle TwinCAT-Plattformen ausgewählt werden, auf denen das Modul lauffähig sein soll. Die Treiber werden dann für alle ausgewählten Plattformen nacheinander gebaut.

- **Lowest compatible TwinCAT build:**
  - Hier muss die Build-Nummer der ältesten TwinCAT-Version eingestellt werden, mit der das Modul noch kompatibel sein soll. Wenn das Modul später mit einer älteren TwinCAT-Version verwendet wird, kann es möglicherweise nicht gestartet werden. Außerdem kann der generierte Code möglicherweise nicht übersetzt werden, wenn das SDK einer älteren TwinCAT-Version verwendet wird.
    Die folgende Tabelle gibt einen Überblick über wesentliche, von der TwinCAT-Version abhängige Eigenschaften des generierten Moduls:

Eigenschaft	TC3 Build	Beschreibung
Große DataAreas	< 4018	DataAreas > 16 MB werden nicht unterstützt
	>= 4018	DataAreas > 16 MB nutzen die Datenbereiche mehrerer DataArea-IDs unter Verwendung der „OBJDATAAREA_SPAN_xxx“-Makros.
Projekt-Unterverzeichnis „_ModuleInstall“	< 4018	Bei der Instanziierung eines zuvor per „Publish“ exportierten Moduls, wird nur die TMC-Beschreibung in das TwinCAT-Projekt importiert. Die Modulinstanz verweist weiterhin auf Dateien innerhalb des Publish-Verzeichnis [▶ 23]. Soll das TwinCAT-Projekt auf weiteren Entwicklungsrechnern geladen werden, müssen die Publish-Verzeichnisse der verwendeten Module manuell in die entsprechenden Verzeichnisse dieser Rechner kopiert werden. Andernfalls kann das Projekt nicht aktiviert werden und das Blockdiagramm wird nicht angezeigt.
	>= 4018	Bei der Instanziierung eines exportierten Moduls werden alle zugehörigen Dateien in das Unterverzeichnis „_ModuleInstall“ des Projektverzeichnisses kopiert. Das Projekt kann jetzt (auch als Archiv verpackt) auf einem anderen Entwicklungsrechner geöffnet werden, ohne zusätzliche Dateien manuell kopieren zu müssen. Weiterer Vorteil ist, dass die Dateien im Publish-Verzeichnis [▶ 23] nun vollständig vom TwinCAT-Projekt entkoppelt sind. Die Modul-Beschreibung, welche nach der Instanziierung Bestandteil des TwinCAT-Projektes ist, und die zugehörigen Dateien (z. B. die Treiber) werden konsistent gehalten. Dateien im Publish-Verzeichnis können überschrieben werden, während das Projekt bis zum „Reload TMC“ mit einer anderen Version des Moduls verwendet und auch weiterhin auf einem Zielsystem neu aktiviert werden kann.

PreCodeGeneration / PostCodeGeneration / PostPublish callback:

Hier können MATLAB-Funktionen eingetragen werden, die vor und nach der Codegenerierung bzw. nach dem Publish aufgerufen werden: (Callback Aufrufreihenfolge)



Um modell- bzw. modulspezifische Aktionen ausführen zu können, kann hier auf die Struktur cgStruct zurückgegriffen werden, die folgende Unterelemente enthält:

Bezeichnung	Wert	Bemerkung
ModelName	Name des Simulink-Modells	
StartTime	Rückgabewert der MATLAB-Funktion „now()“ zu Beginn der Codegenerierung	
BuildDirectory	Aktuelles Build-Verzeichnis	Ab „PostCodeGeneration“
ModuleName	Name des generierten TwinCAT-Moduls	Ab „PostCodeGeneration“
ModuleClassId	ClassId des generierten TwinCAT-Moduls	Ab „PostCodeGeneration“
<UserDefined>	Der Struktur können weitere benutzerspezifische Felder hinzugefügt werden, um nachfolgenden Callbacks zusätzliche Informationen zu übergeben.	

Beispielsweise könnten so im einfachsten Fall zwischen den einzelnen Phasen der Modul-Generierung zusätzliche Informationen ausgegeben werden:

```
PostCodeGeneration callback:    disp(['Code was generated from ' cgStruct.ModelName ' for TcCom-Module ' cgStruct.ModuleName])
```

Siehe auch: Callback-Beispiele [▶ 65]

Signing Certificate for x64 Windows Loader:

Definiert das Zertifikat, das zur Signierung des Treibers für die „TwinCAT RT (x64)“-Plattform verwendet wird. Der Standardwert $(TWINCATTESTCERTIFICATE) verweist auf die Umgebungsvariable TWINCATTESTCERTIFICATE, die unter „Treibersignierung“ (**TC3 Engineering > C/C++ >Vorbereitung**) beschrieben ist. Alternativ kann hier der Zertifikatname direkt eingetragen werden oder es können - je nach gewünschtem Verhalten bei der Signierung - unterschiedliche Platzhalter verwendet werden:

Wert	Verhalten
$<UMGEBUNGSVARIABLE>	Dieser Platzhalter wird schon bei der Codegenerierung aufgelöst und ihr Wert fest in das generierte C++-Projekt geschrieben. Wenn die angegebene Umgebungsvariable nicht gefunden wird, bricht der Codegenerierungsprozess mit einer entsprechenden Fehlermeldung ab.
$(UMGEBUNGSVARIABLE)	Dieser Platzhalter wird erst beim Bauen des generierten C++-Projektes aufgelöst. Wird die Umgebungsvariable nicht gefunden, erscheint lediglich eine Warnung. Der x64-Treiber kann dann trotzdem gebaut aber auf einem Zielsystem nicht vom Windows-Loader geladen werden.
ZertifikatName	Der Name des Zertifikates wird fest in das generierte C++-Projekt geschrieben.
	Bleibt das Feld leer, erscheint lediglich eine Warnung. Der x64-Treiber kann dann trotzdem gebaut, aber nicht auf einem Zielsystem vom Windows-Loader geladen werden.

3.5.2 Datenaustausch (Tc Interfaces)

Konfiguration in Simulink

Abhängig vom Simulink-Modell, gibt es neben den Ein- und Ausgangsvariablen mehrere Gruppen interner Variablen. Je nach Anforderung kann der ADS-Zugriff und der Typ des Prozessabbildes konfiguriert werden. Diese Einstellung beeinflusst in welcher Weise die Variablen mit anderen Prozessabbildern in der TwinCAT-Entwicklungsumgebung verknüpft werden und Daten austauschen können. Folgende Gruppen können konfiguriert werden:

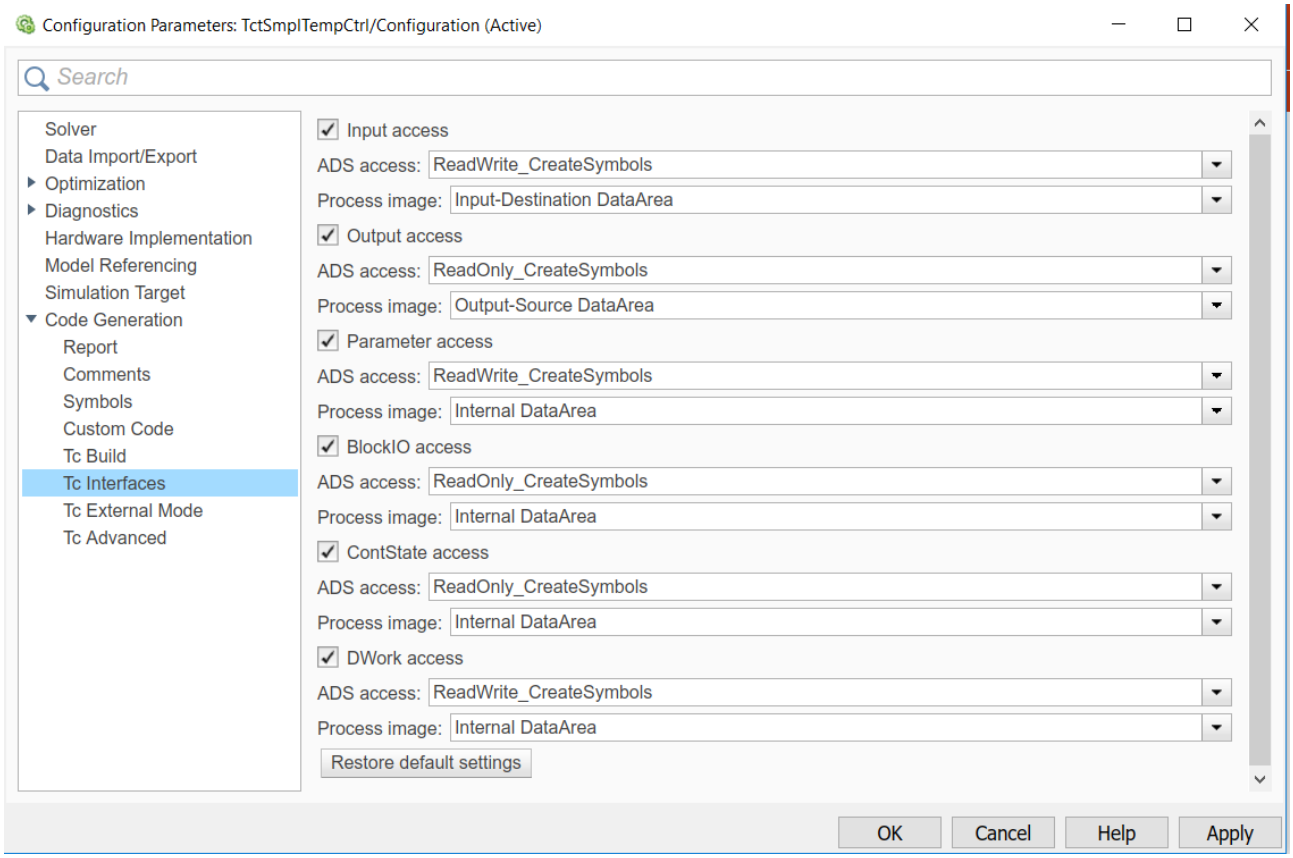
Gruppe	Beschreibung
Input	Modell-Eingänge
Output	Modell-Ausgänge
Parameter	Modellspezifische Parameter: Parameter von Simulink-Blöcken, die "einstellbar" sind
BlockIO	Globale Ausgangssignale von Simulink-Blöcken: Interne Signale, für die ein "Testpunkt" festgelegt wurde oder die wegen der Codeoptimierungen des Codegenerators als global deklariert wurden.
ContState	Kontinuierliche Zustandsvariablen
DWork	Zeitdiskrete Zustandsvariablen

Auf der Konfigurationsseite **TC Interface** in den Coder-Einstellungen gibt es für jede dieser Variablengruppen mehrere Einstellmöglichkeiten. Die auswählbaren Optionen hängen wiederum von der Gruppe ab, d. h. es sind nicht überall alle beschriebenen Optionen verfügbar:

Parameter	Optionen	
GROUP access (checkbox)	TRUE	Das Modul erlaubt Zugriff auf Variablen dieser Gruppe.
	FALSE	Das Modul verweigert Zugriff auf Variablen dieser Gruppe.
ADS access	Nur relevant wenn "*GROUP* access"=TRUE	
	No ADS access	Kein ADS-Zugriff
	ReadOnly_NoSymbols	Kein ADS-Schreibzugriff, ADS-Kommunikation ist nur über die Index-Group und die Index-Offset Informationen möglich
	ReadWrite_NoSymbols	Voller ADS-Zugriff, ADS-Kommunikation ist nur über die Index-Group und die Index-Offset Informationen möglich

Parameter	Optionen	
	ReadOnly_CreateSymbols	Kein ADS-Schreibzugriff, ADS- Symbolinformationen werden erzeugt
	ReadWrite_CreateSymbols	Voller ADS-Zugriff, ADS- Symbolinformationen werden erzeugt
Process image	Nur relevant wenn "*GROUP* access"=TRUE	
	No DataArea	Verknüpfung mit DataArea oder I/O: nein Verknüpfung mit DataPointer: nein
	Standard DataArea	Verknüpfung mit DataArea oder I/O: nein Verknüpfung mit DataPointer: ja
	Input-Destination DataArea	Verknüpfung mit DataArea oder I/O: ja Verknüpfung mit DataPointer: ja
	Output-Source DataArea	Verknüpfung mit DataArea oder I/O: ja Verknüpfung mit DataPointer: ja
	Internal DataArea	Verknüpfung mit DataArea oder I/O: nein Verknüpfung mit DataPointer: nein
	Retain DataArea	Erlaubt die Verknüpfung mit einem „Retain Handler“ (siehe Retain Daten [▶ 29]) zur remanenten Datenhaltung.

Obige Einstellungsmöglichkeiten können Sie in folgender Maske über die entsprechenden „drop-down-lists“ realisieren.



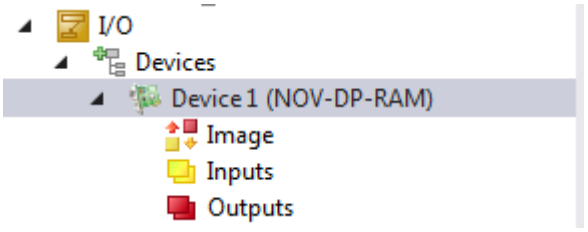
Über **restore default settings** können Sie alle Änderungen wieder rückgängig machen und auf die default-Einstellung zurücksetzen. Die default-Einstellungen sehen Sie in obenstehender Abbildung.

3.5.2.1 Retain Daten

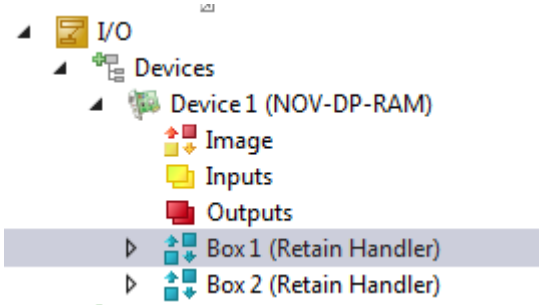
Dieser Bereich beschreibt die Möglichkeit, Daten über einen geordneten oder spontanen Neustart einer Anlage bereitzuhalten. Hierfür wird das NOV-RAM eines Gerätes verwendet. Die EL6080 kann für diese Retain Daten nicht verwendet werden, da die entsprechenden Daten erst übertragen werden müssen, welches zu entsprechenden Laufzeiten führt.
Im Folgenden wird der Umgang mit dem Retain Handler, welcher die Daten speichert und wieder bereitstellt, sowie die Verwendung aus den unterschiedlichen TwinCAT 3 Programmiersprachen beschrieben.

Konfiguration eines Retain Gerätes

1. Die Retain Daten werden dabei von einem Retain Handler, der Teil des NOV-DP-RAM-Geräts im IO Bereich der TwinCAT Solution ist, gespeichert und bereitgestellt. Legen Sie hierfür im IO Bereich der Solution ein NOV-RAM DP Device an.

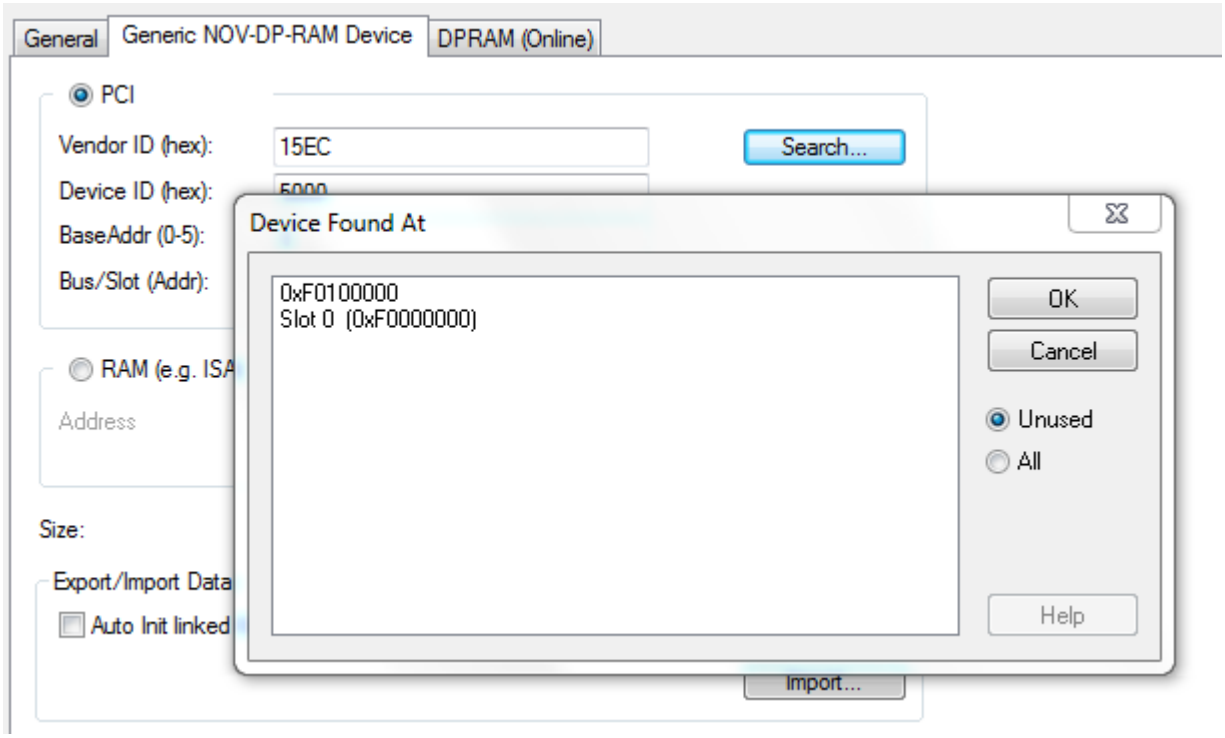


2. Legen Sie unterhalb dieses Gerätes ein oder mehrere Retain Handler an.



Speicherort: NOVRAM

3. Konfigurieren Sie das NOV-DP-RAM-Gerät. Definieren Sie im Tab **Generic NOV-DP-RAM Device** über **Search…** den zu verwendenden Bereich.



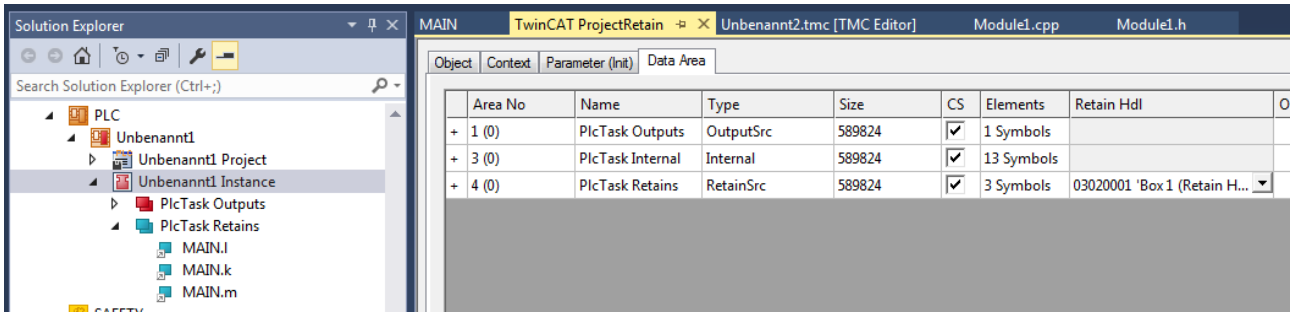
4. Für die Symbolik wird zusätzlich im Boot/-Verzeichnis von TwinCAT ein Retain/-Verzeichnis angelegt.

Nutzung des Retain Handlers mit einem PLC Projekt

In einem 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: UINT;
END_VAR
VAR
  {attribute 'TcRetain':='1'}
  m: UINT;
  x: UINT;
END_VAR
```

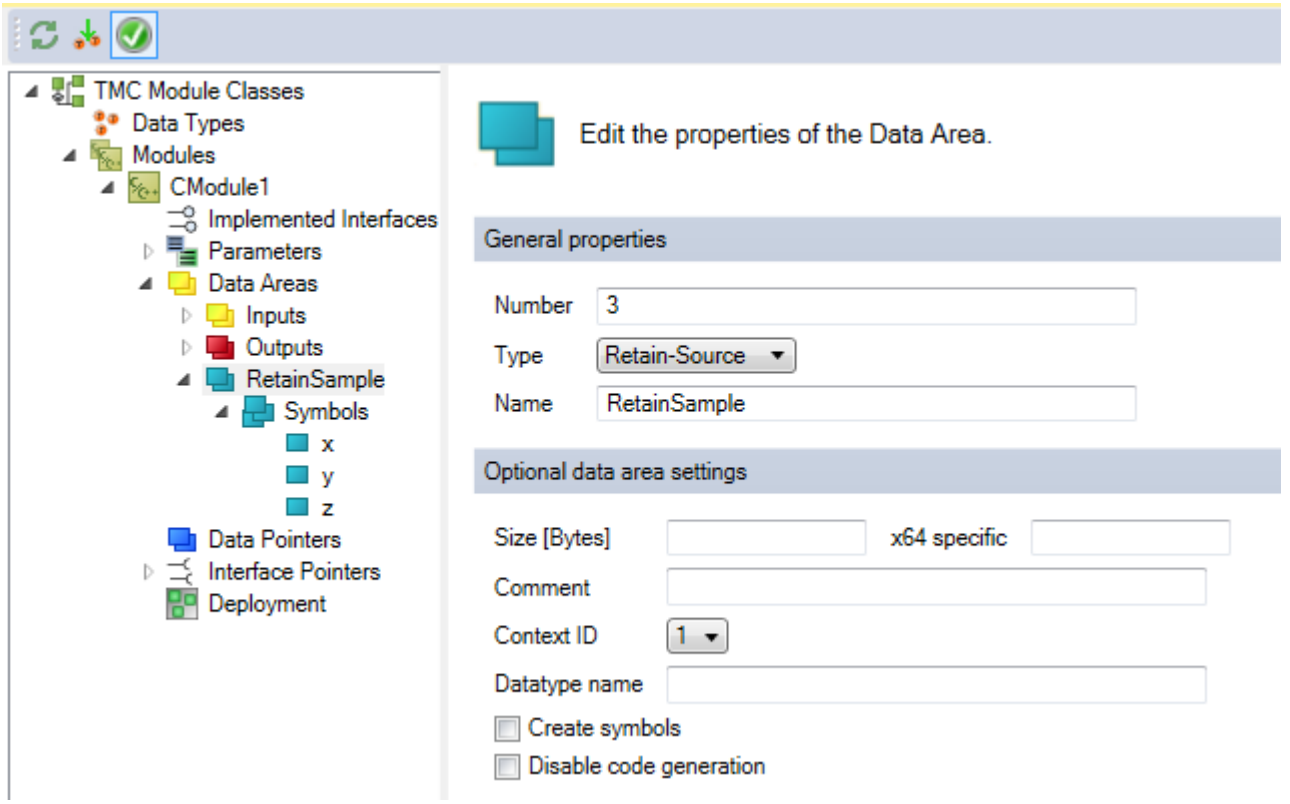
Nach einem „Build“ werden entsprechende Symbole angelegt.
Die Zuordnung zu dem Retain Handler des NOV-DP-RAM Gerätes wird in der Spalte **Retain Hdl** vorgenommen.



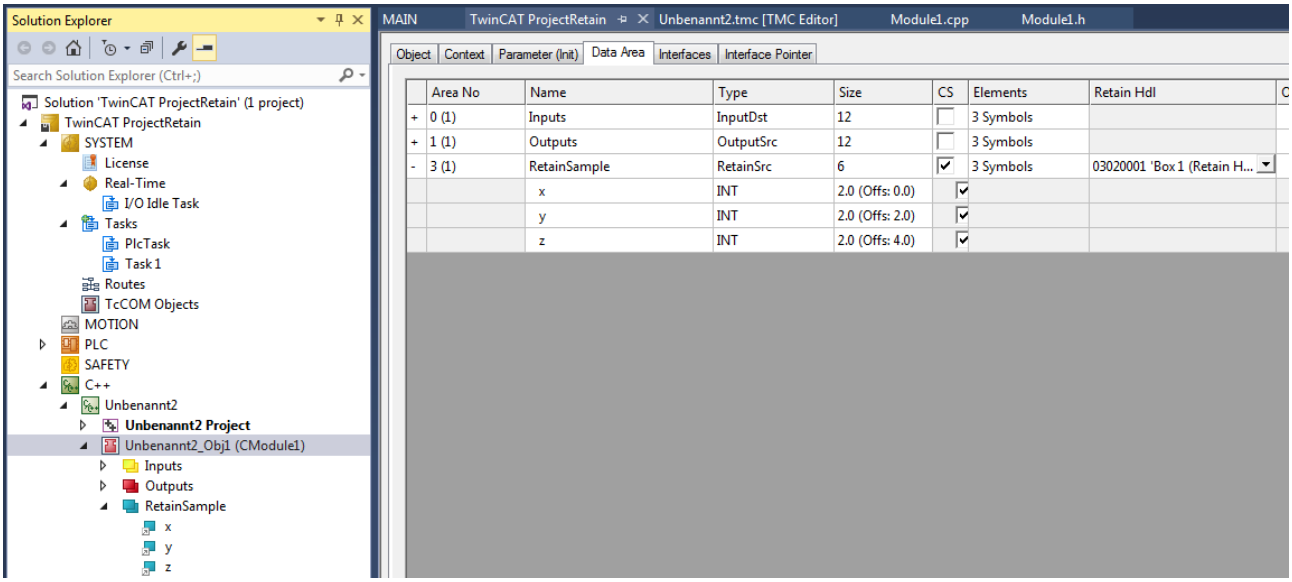
Wenn selbstangelegte Datentypen (DUTs) als Retain verwendet werden, müssen die Datentypen im TwinCAT Typsystem vorhanden sein. Hierfür kann entweder die Option **Convert to Global Type** verwendet werden oder Strukturen können direkt als `STRUCT RETAIN` angelegt werden, womit dann jedoch alle Vorkommen der Struktur über den Retain Handler behandelt werden.
Für POUs (Funktionsbausteine) als Ganzes ist die Verwendung von Retain-Daten nicht möglich. Einzelne Elemente eines POUs können hingegen verwendet werden.

Nutzung des Retain Handlers mit einem C++ Modul

In einem C++ Modul wird eine Data Area vom Typ Retain-Source angelegt, die die entsprechenden Symbole enthält.

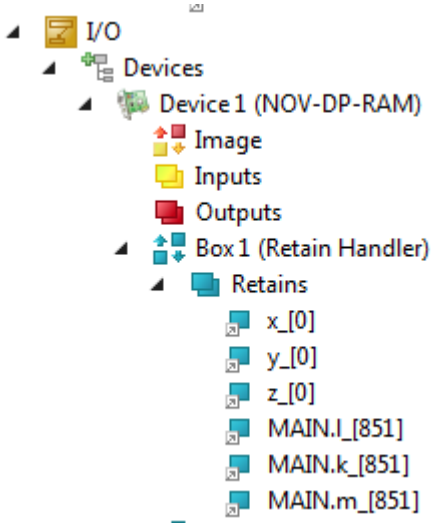


An den Instanzen des C++ Moduls wird ein zu verwendender Retain Handler des NOV-DP-RAM-Gerätes für diese Data Area in der Spalte **Retain Hdl** vorgenommen.



Fazit

Sobald ein Retain Handler in dem jeweiligen Projekt als Ziel ausgewählt wurde, wird nach einem ‚Build' automatisiert sowohl die Symbole unterhalb des Retain Handlers angelegt, wie auch ein Mapping erzeugt.

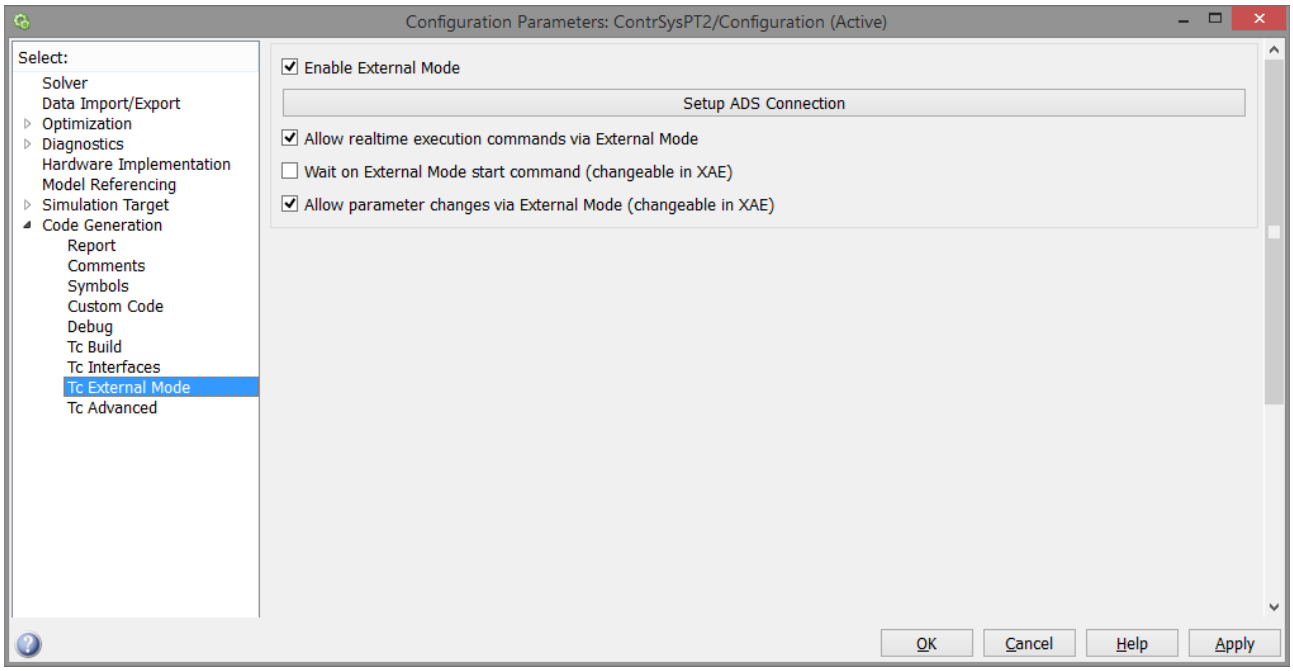


3.5.3 External Mode (Tc External Mode)

In Simulink gibt es verschiedene Ausführungsmodi. Neben dem "Normal Mode", bei dem das Simulink-Modell direkt in der Simulink-Umgebung berechnet wird, gibt es z. B. den "External Mode". In diesem Modus arbeitet Simulink nur als grafische Oberfläche ohne Berechnungen im Hintergrund. Wenn das Modell mit den entsprechenden Einstellungen in ein TcCOM-Modul umgesetzt wurde, kann sich Simulink mit dem instanziierten TcCOM-Objekt verbinden, das gerade in der TwinCAT-Echtzeitumgebung läuft. In diesem Fall werden die internen Signale des Moduls über ADS an Simulink übermittelt, wo sie mit den entsprechenden Simulink-Blöcken aufgezeichnet oder dargestellt werden können. In Simulink veränderte Parameter können online in das TcCOM-Objekt geschrieben werden. Eine solche Online-Parameterveränderung ist allerdings nur bei Parametern möglich, die als "tunable" definiert sind.

Konfiguration des Modulgenerators

Eine „External Mode"- Verbindung ist nur möglich, wenn das generierte Modul sie unterstützt. Dafür muss der **External Mode** vor der Modulgenerierung in den Einstellungen des Simulink Coders unter **TC External Mode** aktiviert werden:



Zusätzlich gibt es hier eine Schaltfläche zur Vorkonfiguration der „External Mode"-Verbindung. Informationen zur Konfiguration der „External Mode"-Verbindung siehe Abschnitt "Verbindung herstellen". Weitere Parameter unter diesem Karteireiter sind:

Parameter	Beschreibung	Defaultwert
Allow real-time execution commands via External Mode	Definiert den Standardwert des Modulparameters [▶ 33] „AllowExecutionCommands“, der festlegt, ob das Modul Start- und Stopp-Befehle aus Simulink verarbeiten soll. **Besonderes Verhalten dieses Parameters:** Der Modulparameter "AllowExecutionCommands" ist ReadOnly, wenn der Wert FALSE ist. Der Code wird in diesem Fall hinsichtlich der Ausführungszeit optimiert und enthält daher nicht die Codeabschnitte zur Verarbeitung von Start- und Stopp-Befehlen.	FALSE
By default wait on External Mode start command	Defaultwert des Modulparameter [▶ 33]s“ WaitForStartCommand“	FALSE

Modulparameter

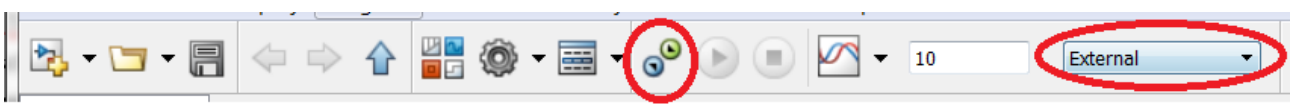
Zur Konfiguration des Verhaltens im **External Mode** (auf der Seite des XAE) ist in generierten Modulen der Parameter ExternalMode als Struktur definiert, der die folgenden Elemente enthält:

Parameter	Beschreibung		Defaultwert
Activated	**ReadOnly**. Legt fest, ob der externe Modus im generierten Modul unterstützt werden soll.		Einstellung des Modulgenerators
AllowExecutionCommands	Nur relevant, wenn "Activated"=TRUE. **ReadOnly**, wenn der Defaultwert FALSE ist, da die Codeabschnitte in diesem Fall im generierten Code nicht enthalten sind. Das heißt, dieser Parameter kann die Verarbeitung von Start- und Stopp-Befehlen deaktivieren, sie jedoch nicht aktivieren, wenn sie bei der Codegenerierung nicht angelegt wurde.		Einstellung des Modulgenerators
	TRUE	Erlaubt Simulink, die Modulabarbeitung über die „External Mode“-Verbindung zu starten und zu stoppen.	
	FALSE	Start- und Stopp-Befehle werden vom Modul ignoriert.	
WaitForStartCommand	Nur relevant, wenn "Activated"=TRUE und "AllowExecutionCommands"=TRUE		Einstellung des Modulgenerators
	TRUE	Das Modul wird nach dem Start von TwinCAT nicht automatisch ausgeführt sondern wartet auf den Startbefehl von Simulink.	
	FALSE	Das Modul wird sofort mit der zugewiesenen TwinCAT-Task gestartet. (Default)	

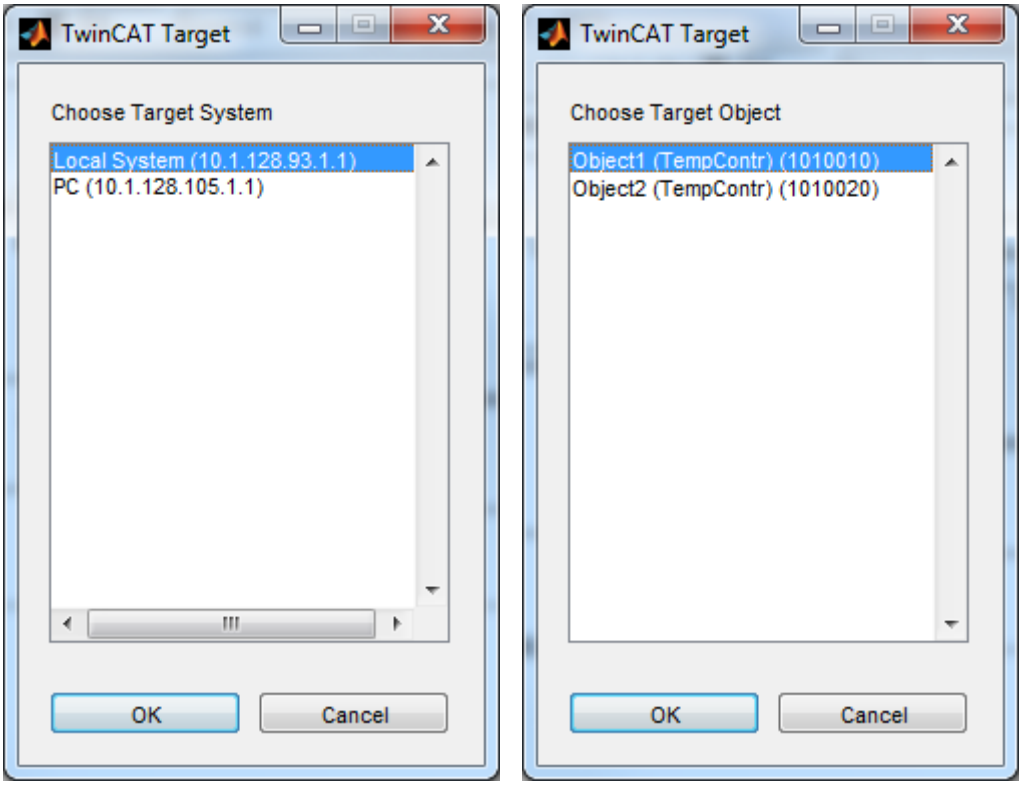
Für weitere Informationen zur Anpassung der Modulparameter in XAE siehe Abschnitt "Parametrierung des generierten Moduls".

Verbindung aus Simulink® herstellen

Die „External Mode“-Verbindung kann aus Simulink über das Icon **Connect to Target** gestartet werden, welches nach Anwahl des Modus **External** in der Simulink-Toolbar zu sehen ist:



Bei fehlenden oder falschen Verbindungsdaten werden nacheinander die folgenden Dialoge angezeigt, damit der Benutzer die Verbindung neu konfigurieren kann:



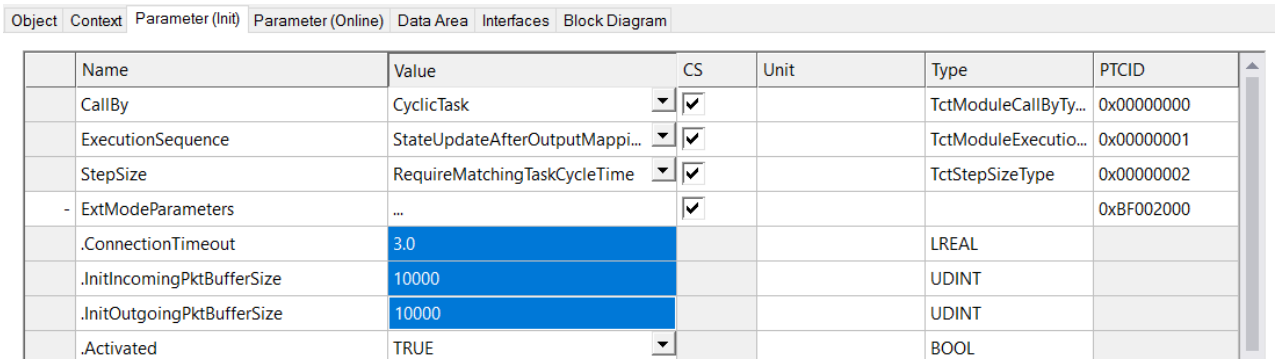
Das erste Dialogfenster zeigt eine Liste von Zielsystemen, das zweite eine Liste der verfügbaren Modul-Instanzen auf dem gewählten Zielsystem. Im folgenden Dialog kann der Benutzer festlegen, ob die neuen Verbindungsdaten gespeichert werden sollen. Nach dem Speichern der Verbindungsdaten wird die Verbindung automatisch hergestellt, wenn die Verbindungsdaten auf ein gültiges und passendes Modul verweisen.
Die gespeicherten Verbindungsdaten können in den Coder-Einstellungen über der Schaltfläche **Setup ADS Connection** unter **TC External Mode** jederzeit geändert werden.

Übermittlung der Berechnungsergebnisse von „minor time steps“

Über ADS übermittelte Signalwerte können unter bestimmten Umständen von den Werten abweichen, die via „Output mapping“ auf andere Prozessabbilder kopiert wurden. Siehe Übermittlung der Berechnungsergebnisse von „minor time steps“ [▶ 61].

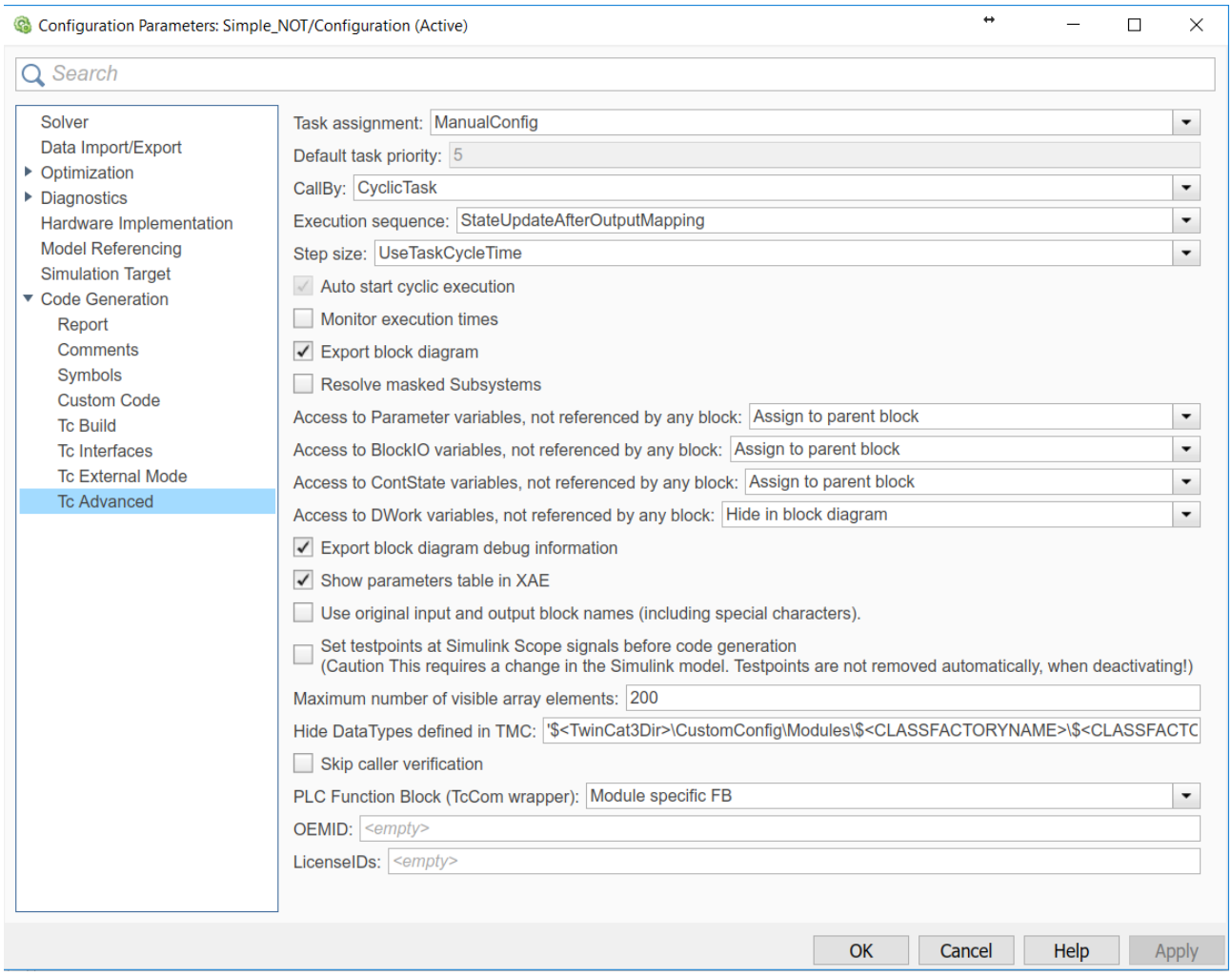
Parametrierung in TwinCAT

Bei großen Modellen müssen viele Daten zwischen TwinCAT und Simulink kommuniziert werden. Dies geschieht über ADS, wobei TwinCAT-seitig Puffer angelegt werden. Sie haben die Möglichkeit die Puffer für eingehende und ausgehende Daten zu adaptieren (Defaultwerte 10000 Byte) sowie die Timeout-Schwelle (Default 3.0 Sekunden) anzupassen.



3.5.4 Erweiterte Einstellungen (Tc Advanced)

In den erweiterten Einstellungen lassen sich Parameter einstellen, die zum einen das Ausführungs- und Aufrufverhalten des Moduls und zum anderen die Darstellungen und Eigenschaften des exportierten Blockdiagramms beeinflussen:

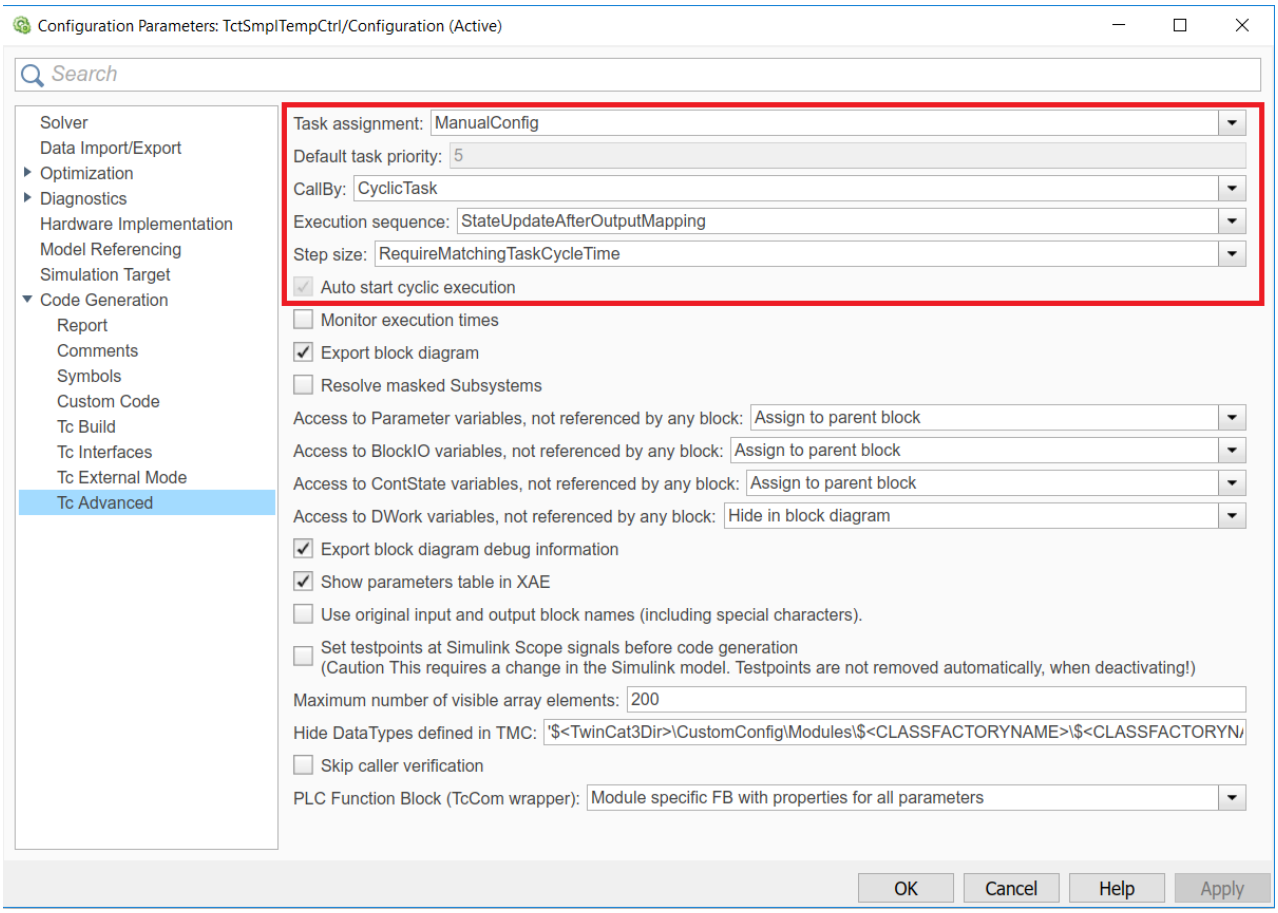


Ausführungsverhalten des generierten Moduls

In TwinCAT 3 kann ein Simulink-Modul direkt von einer zyklischen Echtzeit-Task oder von einem anderen TwinCAT-Modul, z. B. einer SPS, aufgerufen werden. Das Verhalten der generierten Modul-Klasse kann in Simulink unter Tc Advanced parametriert werden. Um das Verhalten der einzelnen Modulinstanzen ggf. abweichend vom Verhalten der Klasse festzulegen, kann die Art der Ausführung in der TwinCAT 3-Entwicklungsumgebung über die TcCOM-Parameterliste im Karteireiter **Parameter (Init)** oder über den Parameterbereich im Blockdiagramm angepasst werden.

Konfiguration der Standardeinstellungen in Simulink

Die Standardwerte der Aufruf-Parameter können in den Simulink-Codereinstellungen konfiguriert werden, um den Aufwand für die Parametrierung der einzelnen Objekte (Modulinstanzen) zu verringern:



Task Zuweisung

Unter **Task assignment** kann die Zuweisungsart einer Task in TwinCAT definiert werden.

Einstellung "Depend On"	Beschreibung
Manual Config	Die Tasks können in der Kontexttabelle manuell zugewiesen werden, indem in der Spalte "Task" die Objekt-IDs der Tasks ausgewählt oder eingetragen werden. Die ausgewählten Tasks müssen alle Kriterien erfüllen, die über die "Aufruf-Parameter" konfiguriert wurden.
Parent Object	Kann nur verwendet werden, wenn der Eltern-Knoten der Modulinstanz im Projektbaum eine Task ist. In diesem Fall dient das Parent-Object als zyklischer Aufrufer des Moduls.
Task Properties	Die Tasks werden dem Modul automatisch zugewiesen, wenn Zykluszeit und Priorität den in Simulink festgelegten Werten entsprechen. Gibt es keine entsprechende Task, können unter dem Knoten "System Configuration -> Task Management" neue Tasks erstellt und entsprechend parametriert werden.

Ist die Option Task properties aktiv, ist die Priorität der entsprechenden Task anzugeben.

Zyklischer Aufruf

Wurde für den Aufruf-Parameter "CallBy" der Wert "CyclicTask" gesetzt, erfolgt beim Start des Moduls eine Verifizierung aller Task-Zykluszeiten. Über den Aufruf-Parameter "Step size" können die Bedingungen der Zykluszeiten der zugeordneten Tasks festgelegt werden. Erfüllen alle Zykluszeiten ihre Bedingungen, kann das Modul starten. Anderenfalls bricht der Start des Moduls und der TwinCAT-Laufzeit mit entsprechenden Fehlermeldungen ab.

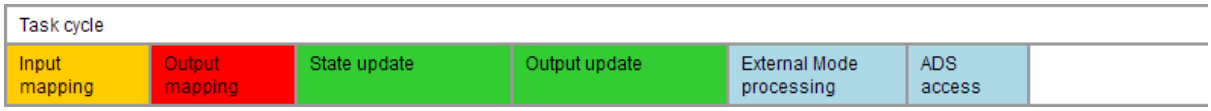
Aufruf aus einem anderen TwinCAT-Modul

Wurde der Aufruf-Parameter "CallBy" auf den Wert "Module" gesetzt, rufen die zugewiesenen Tasks das Modul nicht automatisch auf. Um das generierte TcCOM-Modul stattdessen über ein anderes Modul aufzurufen, kann auf dessen Schnittstellen zugegriffen werden. Das kann aus einem C++-Modul geschehen oder aus einer TwinCAT SPS, wie in „Aufruf des generierten Moduls aus einem SPS-Projekt [▶ 45]“ gezeigt.

Ausführungsreihenfolge

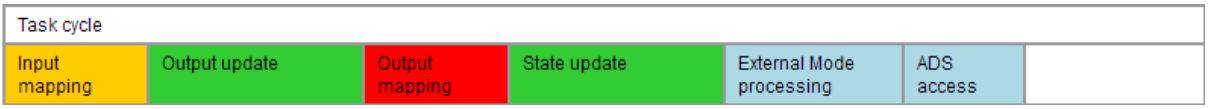
In Modulen, die mit TE1400 ab Version 1.1 erstellt wurden, kann eine Ausführungsreihenfolge festgelegt werden, um Jitter und Reaktionszeiten für die jeweilige Anwendung zu optimieren. Ältere Versionen verwenden immer die Reihenfolge "StateUpdateAfterOutputMapping". Die folgende Tabelle veranschaulicht die Vor- und Nachteile der unterstützten Aufrufreihenfolgen:

- IoAtTaskBegin



  - Längste Reaktionszeit aller Optionen
  + Kleinstmöglicher Jitter in der Reaktionszeit

- StateUpdateAfterOutputMapping

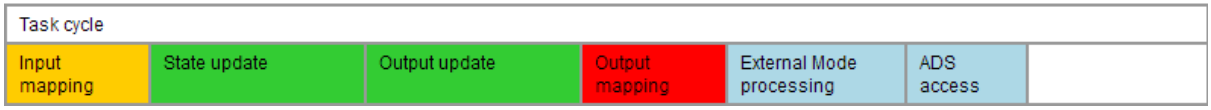


  + Kürzeste Reaktionszeit
  o Mittlerer Jitter in der Reaktionszeit

Ergebnisse aus „Minor Time Steps“ werden per ADS übermittelt

Per ADS übermittelte Signalwerte können von den Werten abweichen, die via „Output mapping“ auf andere Prozessabbilder kopiert wurden. Grund hierfür ist, dass bei „State update“ je nach gewähltem Solver Werte überschrieben werden können. Siehe Übermittlung der Ergebnisse aus „Minor Time Steps“ [▶ 61] für weitere Informationen.

- **StateUpdateBeforeOutputUpdate**



  o Mittlere Reaktionszeit
  - größter Jitter in der Reaktionszeit, da abhängig von Ausführungszeiten für „State update“ und „Output update“

Schrittweiteneinstellung (Step size)

Hier wird das Verhalten des generierten TcCOM bezüglich der Schrittweite (in TwinCAT entsprechend die Cycle Time) definiert.

- RequireMatchingTaskCycleTime
  Das Modul erwartet als Zykluszeit der zugeordneten Task die in Simulink festgelegte "Fixed Step Size". Wird eine andere Zykluszeit eingestellt, wird die Startsequenz des TcCOM mit einer Fehlermeldung abgebrochen. Multitasking-Module erwarten, dass alle zugewiesenen Tasks mit den zugehörigen Simulink-Sampletimes konfiguriert wurden.
- UseTaskCycleTime
  Das Modul erlaubt Zykluszeiten, die von der in Simulink festgelegten "Fixed Step Size" abweichen. Bei Multitasking-Modulen müssen alle Task-Zykluszeiten das gleiche Verhältnis zur zugehörigen Simulink-Sampletime aufweisen. Bei abweichender Zykluszeit von der Simulink-Sampletime wird eine Warnung in TwinCAT ausgegeben, welche auf die Abweichung hinweist.
- UseModelStepSize
  Das Modul verwendet für alle internen Berechnungen die in Simulink eingestellte Sampletime. Diese Einstellung ist vornehmlich für die Verwendung in Simulationen innerhalb der TwinCAT-Umgebung gedacht und kann zur beschleunigten oder auch verlangsamten Simulation eingesetzt werden.

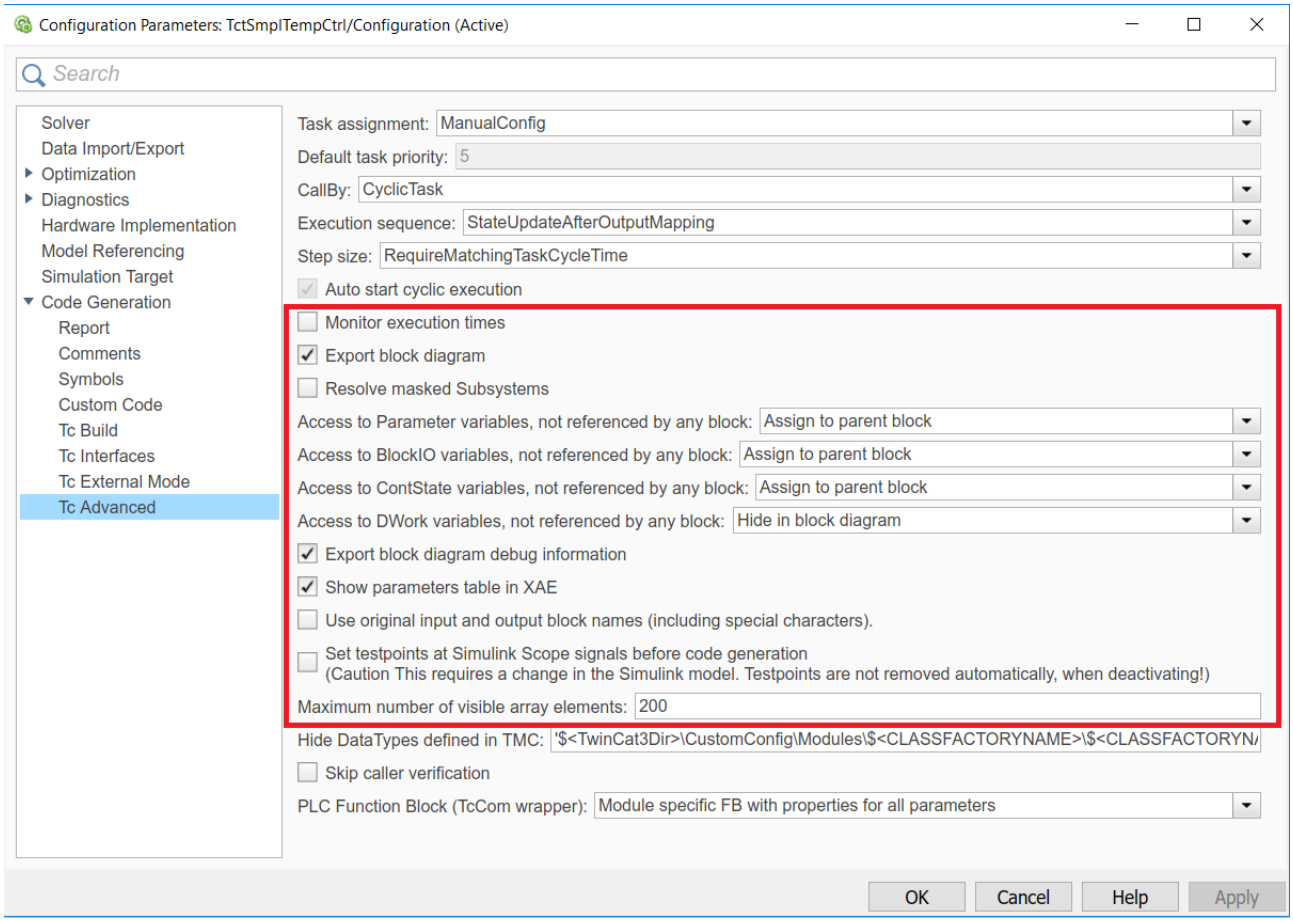
Auto start cyclic execution

Ist diese Option aktiviert (default), wird das Tc-COM Modul beim Aufstarten in den OP-State gesetzt und der generierte Modellcode direkt ausgeführt. Ist diese Option deaktiviert, wird das Modul ebenfalls in den OP-State versetzt, jedoch wird der Modellcode nicht abgearbeitet. Diese Option ist im instanziierten Modul im Karteireiter Parameter (init) und im Parameterbereich des Blockdiagrams unter Module parameters als Variable „ExecuteModelCode“ zu finden und kann hier ebenfalls angepasst werden.

Anpassung der Darstellung, Debugging und Parametrierbarkeit

Aus Simulink erzeugte Module erlauben auch nach der Codegenerierung und Instanziierung noch weitreichende Möglichkeiten zur Parametrierung der Modul- und Modellparameter. Die Parametrierungsmöglichkeiten können vor der Codegenerierung angepasst werden, so dass z. B. in der Entwicklungsphase Debug-Möglichkeiten erlaubt und auch Parameter maskierte Subsysteme aufgelöst werden, welche in einer Release Version versteckt werden sollen. Die Nutzung von Modulen in TwinCAT 3 setzt eine entsprechend der Anforderungen konfigurierbare Darstellung voraus. So können z. B. Debug-Informationen im Blockdiagramm mit exportiert werden.

Die folgenden Coder-Parameter erlauben die Anpassung des Blockdiagramm-Exports, der Parameter- und Signaldarstellung sowie erweiterter Funktionen:



Parameter	Beschreibung		Defaultwert
Monitoring execution times	TRUE	Die Ausführungszeiten des TC-Moduls werden berechnet und zur Überwachung als ADS-Variable bereitgestellt.	FALSE
	FALSE	Berechnung der Ausführungszeiten deaktiviert.	
Export block diagram	TRUE	Das Simulink-Blockdiagramm wird exportiert und nach der Instanziierung des Moduls in XAE unter dessen Karteireiter "Block Diagram" angezeigt.	TRUE
	FALSE	Das Simulink-Blockdiagramm wird nicht exportiert und der Karteireiter "Block Diagram" wird in XAE nicht angezeigt.	
Resolve masked Subsystems	Nur relevant, wenn "Export block diagram"=TRUE.		FALSE

Parameter	Beschreibung		Defaultwert
	TRUE	Maskierte Subsysteme werden aufgelöst. Alle Inhalte maskierter Subsysteme sind für Benutzer des generierten Moduls in XAE sichtbar.	
	FALSE	Maskierte Subsysteme werden nicht aufgelöst. Die Inhalte der maskierten Subsysteme sind für Benutzer des generierten Moduls unsichtbar.	
Access to *VariableGroup*, not referenced by any block	Assign to parent block	Variablen dieser Gruppe, die zu einem Block innerhalb eines nicht aufgelösten Subsystems gehören, werden dem nächsthöheren, sichtbaren Subsystem zugeordnet.	
	Hide in block diagram	Variablen dieser Gruppe, die keinem Simulink-Block zugeordnet werden können, werden im Blockdiagramm nicht angezeigt.	
	Hide, No access	Variablen dieser Gruppe, die keinem Simulink-Block zugeordnet werden können, werden versteckt und unzugänglich gemacht. Dies ist nur möglich, wenn für das Prozessabbild dieser Variablen-Gruppe „No DataArea“ ausgewählt wurde (Konfiguration in Simulink [▶ 27]).	
Export block diagram debug information	TRUE	Debug-Informationen zum Blockdiagramm werden generiert, die eine Zuordnung von Zeilennummern im genierten Code zu dargestellten Blöcken erlauben. Wird für das Debuggen [▶ 202] benötigt.	TRUE
	FALSE	Keine Debug-Informationen werden generiert	
Show parameters table in XAE	TRUE	Der Karteireiter "Parameter (Init)" wird in XAE angezeigt und erlaubt die Parametrierung des Moduls über die Parameterliste.	TRUE
	FALSE	Der Karteireiter "Parameter (Init)“ wird in XAE nicht angezeigt.	
Use original input and output block names	FALSE	Ein- und Ausgänge des Moduls tragen den Namen, der vom Simulink Coder als Variablenname angelegt wurde. Leer- und Sonderzeichen sind dort nicht erlaubt.	FALSE
	TRUE	Ein- und Ausgänge des Moduls tragen genau den Namen, der im Simulink Modell verwendet wurde. Die Namen können Leer- und Sonderzeichen enthalten.	
Set testpoints at Simulink Scope signals before code generation		Scope-Blöcke werden vom Simulink-Coder ignoriert, d. h. die Signale stehen im Allgemeinen im generierten TwinCAT-Modul nicht zur Verfügung und können nicht angezeigt werden. Um die Generierung von Variablen für diese Signale zu erzwingen, können im Simulink-Modell „Testpoints“ definiert werden. Dieser Parameter kann genutzt werden, um automatisiert „Testpoints“ für alle Scope-Eingangssignale zu erstellen.	
Maximum number of visible array elements		Gibt die maximale Anzahl der in der TwinCAT-Entwicklungsumgebung darzustellenden Array-Elemente an. Größere Arrays können nicht aufgeklappt und die Elemente z. B. nicht einzeln verknüpft werden.	

Hide Datatypes defined in TMC

In jeder TMC-Datei werden die benötigten Datentypen spezifiziert und durch Import in TwinCAT 3 im System bekannt gemacht. Den Datentypen wird dabei eine eindeutige GUID zugeordnet. Entsprechend bleibt die GUID unverändert, wenn eine TMC-Datei neu importiert wird, bei der sich ein Datentyp nicht verändert hat. Bei der Verwendung von z. B. Enums oder Strukturen, kann durch Veränderungen (z. B. zusätzlicher Modellparameter in einer Struktur) der Fall auftreten, dass der Datentypname des veränderten Datentyps und der des vorherigen Datentyps identisch sind, jedoch unterschiedliche GUIDs haben. Diese eindeutige

Zuordnung über GUIDs existiert in der PLC nicht, sondern hier wird über den Datentypnamen identifiziert. Bei Aufruf einer TcCOM Instanz aus der PLC heraus ist entsprechend ein Mechanismus vorzuhalten, der Mehrdeutigkeiten vorbeugt.

Über den Punkt **Hide Data Types defined in TMC** wird die zuletzt importierte TMC bzw. deren Datentypnamen und Datentypen für die PLC verwendet. Bereits existierende Datentypnamen mit anderen Datentypen werden für die PLC versteckt. Siehe dazu auch Wie löse ich Datentyp-Konflikte im SPS-Projekt? [▶ 63].

Skip caller verification

Diese Option deaktiviert Abfragen bei dem Aufruf eines TcCOM aus der SPS, siehe Aufruf des generierten Moduls aus einem SPS-Projekt [▶ 45]. Dies führt zu einer schnelleren Abarbeitung des Modulaufrufs, auf der anderen Seite muss dann vom Anwender zwingend sichergestellt sein, dass der Aufruf korrekt und aus dem richtigen Kontext heraus erfolgt.

Diese Option sollte erst dann aktiviert werden, wenn es aus Performanz-Gründen notwendig ist und unter zuvor das Projekt mit aktivierten Abfragen entsprechend getestet wurde.

PLC Function Block

Der POU-Typus zum Aufruf eines Simulink-Objektes aus der SPS kann hier näher definiert werden. Eine ausführliche Beschreibung ist unter „Aufruf des generierten Moduls aus einem SPS-Projekt [▶ 45]“ zu finden.

OEM Lizenzcheck

Optional kann ein generiertes TcCOM an eine OEM-Lizenz gebunden werden. Diese OEM-Lizenz wird beim Starten des TcCOM (neben der Beckhoff Runtime-Lizenz TC1220 oder TC1320) in TwinCAT 3 überprüft. Ist keine gültige Lizenz vorhanden startet das Modul nicht auf und es erscheint eine entsprechende Fehlermeldung.

Wie Sie OEM Zertifikate erstellen und diese dann verwalten finden Sie beschrieben unter TwinCAT3 > TE1000 XAE > Technologien > Security Management.

In Simulink können Sie den OEM Lizenzcheck einfügen über Benennung Ihrer OEM ID und Ihrer abzufragenden Lizenz ID bzw. mehrerer Lizenz IDs. Ihre OEM ID finden Sie in der Security Management Konsole (Extended Info aktiviert), die Lizenz ID können Sie einsehen durch Doppelklick auf den entsprechenden Lizenzeintrag in TwinCAT unter System > License. Beide IDs sind ebenfalls im generierten License Request File enthalten, wenn ein Request File mit Ihrer OEM Lizenz generiert wird.

GUID Form beachten

Die einzutragenden IDs sind als String in GUID-Form zu übergeben, d.h. in Simulink sind die Angaben in Hochkommata zu setzen. Ebenfalls sind keine Leerzeichen in den Angaben erlaubt.

Beispieleintrag:

OEM ID: '{B0D1D1B7-99AB-681G-F452-F4B3F1A993C0}'

License ID: 'Name1,{6B4BD993-B7C3-4B72-B3D1-681FE7DDF3D1}'

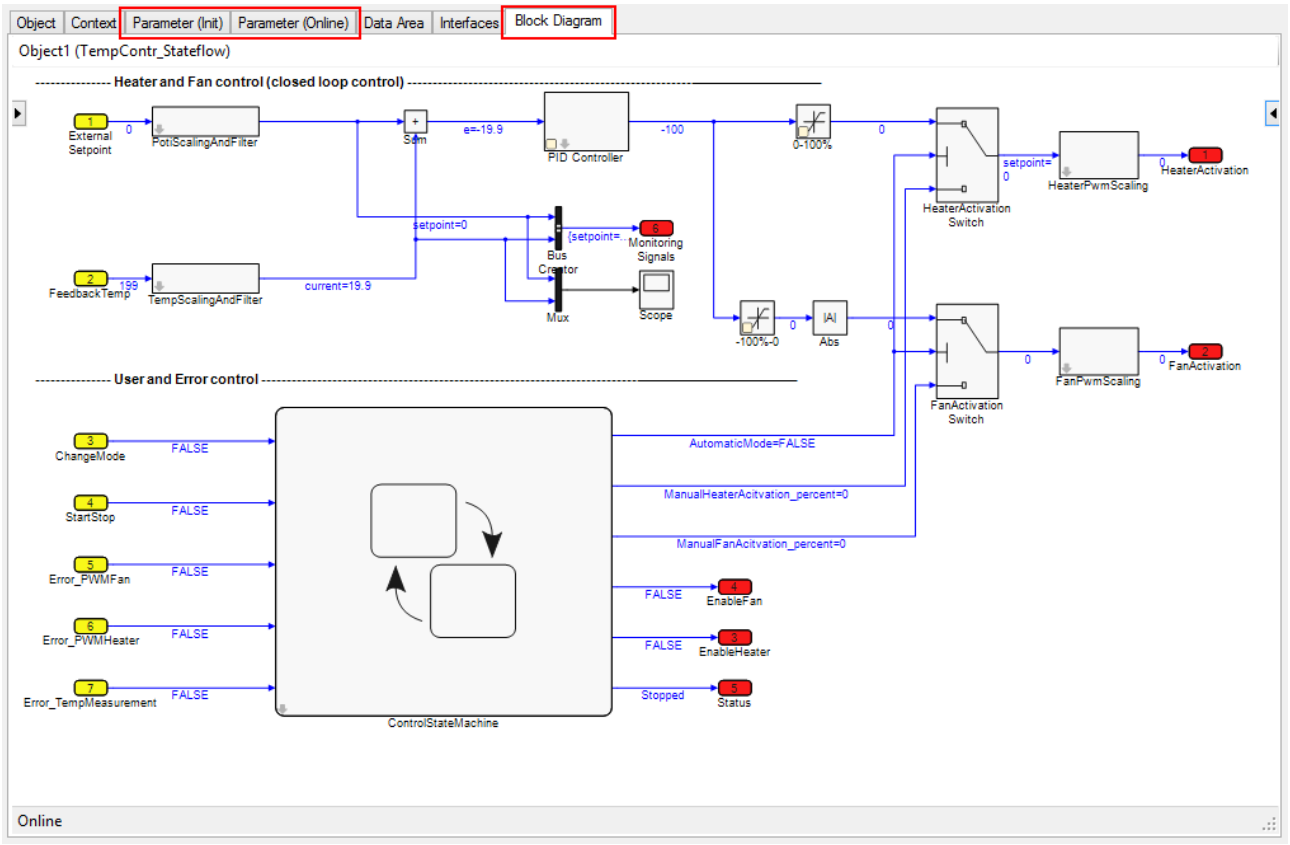
3.6 Anwendung von Modulen in TwinCAT

Die Daten exportierter Module aus Simulink liegen im Verzeichnis *%TwinCat3Dir\CustomConfig\Modules\<MODULENAME>\%* und können von dort auf beliebig viele Entwicklungsrechner mit einem TwinCAT XAE kopiert werden. Eine Simulink Lizenz ist auf diesen Systemen nicht mehr notwendig. Dennoch bietet TwinCAT weiterhin weitreichende Parametrierungsmöglichkeiten der generierten Module. Des Weiteren wird im Folgenden auf die zyklische Ausführung von TcCOM Modulen durch Aufruf über eine Task sowie den Aufruf von Modulen aus einem SPS-Projekt eingegangen.

3.6.1 Parametrierung einer Modul-Instanz

Parameterdarstellung in XAE

Das Block Diagramm im Karteireiter Browser Parameter:

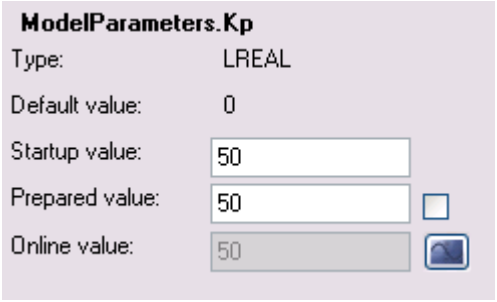


Allgemein kann die Parametrierung von TcCOM-Modulen in der TwinCAT3-Entwicklungsumgebung (XAE) über die Parameterliste im Karteireiter **Parameter (Init)** erfolgen. Simulink-Module können außerdem über das Blockdiagramm parametriert werden, sofern der Export des Blockdiagramms in den Simulink Coder-Einstellungen unter **Tc Advanced** aktiviert ist.

Modul- und modellspezifische Parameter

Die Parameterliste enthält modul- und modellspezifische Parameter. Modulspezifische Parameter sind z. B. "Aufruf-Parameter [▶ 42]" oder "External Mode Parameter [▶ 32]". Im Blockdiagramm werden diese Parameter im Parameterbereich (rechte Seite des Fensters) nur angezeigt, wenn die oberste Ebene des Blockdiagramms ("<root>“) ausgewählt ist.
Modellspezifische Parameter sind in den Simulink-Blöcken als "tunable" definierte Parameter. Die Parameterliste zeigt sie als Struktur an.

Im Blockdiagramm werden die Modell-Parameter einem Block oder auch mehreren Blöcken zugeordnet, ihre Werte können angepasst werden, wenn der zugehörige Block ausgewählt wurde. Im Dropdownmenü der Eigenschaftstabelle oder dem Parameter-Window direkt im Blockdiagramm können dann die Werte der Parameter (Startup, Prepared oder Online) angepasst werden:



Beim Überfahren des Titels des Dropdownmenü (hier ModelParameters.Kp) mit der Maus wird als Tooltip dessen ADS-Information angezeigt. Durch Rechts-Klick auf den Titel kann die ADS-Symbolinformation in die Zwischenablage kopiert werden.

Der Zugang zu den modellspezifischen Parametern ist nur möglich, wenn

- die Simulink-Optimierungsoption **Inline parameters** deaktiviert ist oder in den erweiterten Optionen unter **Inline parameters** Workspace-Variablen als Modellparameter ausgewählt wurden
- unter **Tc Interfaces** der ADS-Zugriff auf Parameter aktiviert ist (Datenaustausch (Tc Interfaces) [▶ 27]).

„Default“, „Startup“, „Online“ und „Prepared“

Im Drop-down-Menü der Eigenschaftstabelle des Blockdiagramms findet man folgende Wertetypen:

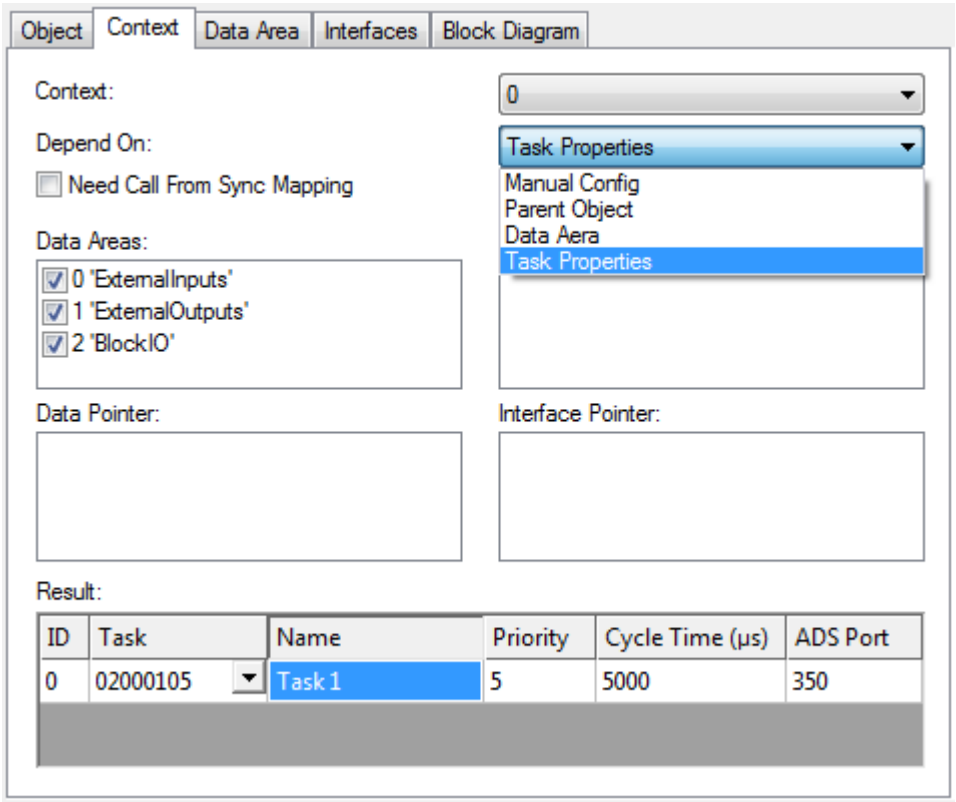
- **Default-Werte** sind die Parameterwerte beim Generieren des Codes. Sie sind unveränderlich in der Modulbeschreibungsdatei gespeichert und ermöglichen es, nach Parameteränderungen die "manufacturing settings" wiederherzustellen.
- **Startup-Werte** werden in der TwinCAT-Projektdatei gespeichert und in die Modulinstanz heruntergeladen, sobald TwinCAT die Modulinstanz startet.
  In Simulink®-Modulen können auch Startwerte für das Eingangs-Prozessabbild festgelegt werden. Dadurch kann das Modul mit Eingangswerten ungleich Null gestartet werden, ohne dass die Eingänge mit anderen Prozessabbildern verknüpft werden müssen. Interne Signale und Ausgangssignale haben keine Startwerte, da sie in jedem Fall im ersten Zyklus überschrieben würden.
- **Onlinewerte** sind nur verfügbar, wenn das Modul auf dem Zielsystem gestartet wurde. Sie zeigen den aktuellen Wert des Parameters im laufenden Modul. Dieser kann auch während der Laufzeit geändert werden. Das entsprechende Eingabefeld muss dazu allerdings erst über das Kontextmenü freigeschaltet werden, um versehentliche Eingaben zu vermeiden.
- **Prepared-Werte** können immer dann festgelegt werden, wenn auch Onlinewerte verfügbar sind. Mit ihrer Hilfe können verschiedene Werte gespeichert werden, um sie konsistent in das Modul zu schreiben. Wenn vorbereitete Werte festgelegt wurden, sind diese in einer Tabelle unterhalb des Blockdiagramms zu sehen. Mit den Schaltflächen rechts neben der Liste können die vorbereiteten Werte als Onlinewert heruntergeladen und/oder als Startwert gespeichert oder auch gelöscht werden.

3.6.2 Ausführung des generierten Moduls unter TwinCAT

In TwinCAT 3 kann ein TcCOM-Modul direkt von einer zyklischen Echtzeit-Task oder von einem anderen Modul, z.B. einer SPS, aufgerufen werden. Um das Verhalten der einzelnen Modulinstanzen festzulegen, kann die Art der Ausführung in der TwinCAT 3-Entwicklungsumgebung definiert werden.

Kontexteinstellungen

Unter dem Karteireiter **Context** der Modulinstanz finden Sie eine Liste aller Simulink-Sampletimes des Moduls. Ist in den Solvereinstellungen des Simulink-Modells "SingleTasking" ausgewählt, ist die Anzahl der Tasks auf 1 beschränkt:



Für jeden der in der Tabelle aufgelisteten Kontexte ist eine Task festzulegen, über die der Aufruf des Moduls erfolgen soll. Abhängig von den Einstellungen unter "Depend On" erfolgt die Zuordnung der Tasks unterschiedlich:

Einstellung "Depend On"	Beschreibung
Manual Config	Die Tasks können in der Kontexttabelle manuell zugewiesen werden, indem in der Spalte "Task" die Objekt-IDs der Tasks ausgewählt oder eingetragen werden. Die ausgewählten Tasks müssen alle Kriterien erfüllen, die über die "Aufruf-Parameter" konfiguriert wurden.
Parent Object	Kann nur verwendet werden, wenn der Eltern-Knoten der Modulinstanz im Projektbaum eine Task ist. In diesem Fall dient das Parent-Object als zyklischer Aufrufer des Moduls.
Task Properties	Die Tasks werden dem Modul automatisch zugewiesen, wenn Zykluszeit und Priorität den in Simulink festgelegten Werten entsprechen. Gibt es keine entsprechende Task, können unter dem Knoten "System Configuration -> Task Management" neue Tasks erstellt und entsprechend parametriert werden.

Konfiguration in XAE

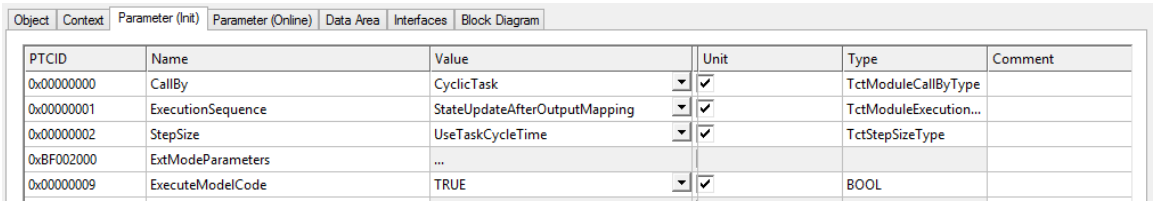
Parameter, die das Verhalten des Simulink-Moduls in der Ausführung beeinflussen, sind:

Parameter	Optionen / Beschreibung	
CallBy	Task	Das Modul hängt sich automatisch an die in den Kontexteinstellungen [▶ 42] festgelegten Tasks an, wenn TwinCAT in den Run-Modus geschaltet wird. Die Tasks rufen das Modul zyklisch auf, bis TwinCAT gestoppt wird.
	Module	Das Modul wird nicht direkt von den zugeordneten Tasks aufgerufen, sondern kann aus der SPS oder einem anderen Modul heraus aufgerufen werden. Wichtig: Dem aufrufenden Modul muss die gleiche Task zugewiesen sein, wie dem aufzurufenden TcCOM-Objects selbst.

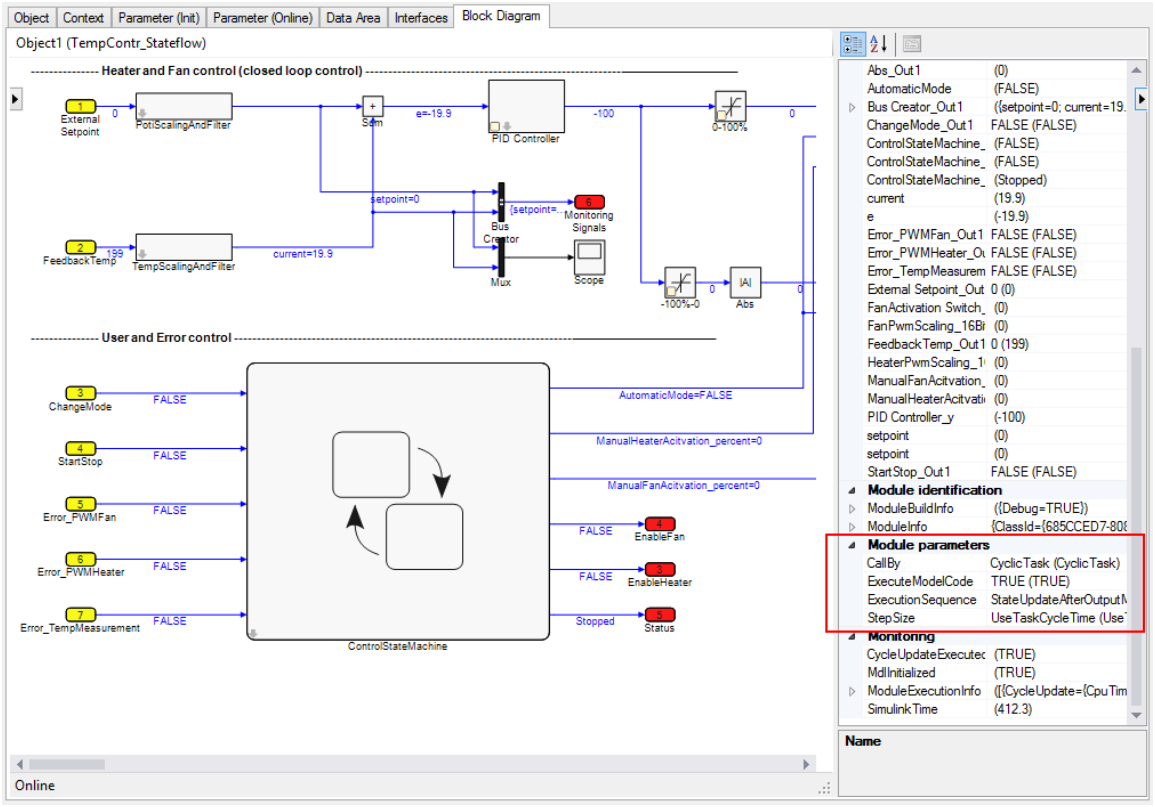
Parameter	Optionen / Beschreibung	
Step size	RequireMatchingTaskCycleTime	Das Modul erwartet als Zykluszeit der zugeordneten Task die in Simulink festgelegte "Fixed Step Size". Multitasking-Module erwarten, dass alle zugewiesenen Tasks mit der zugehörigen Simulink-Sampletime konfiguriert wurden. Anderenfalls kann das Modul (und auch TwinCAT) nicht gestartet werden. Die Startsequenz wird dann mit entsprechenden Fehlermeldungen abgebrochen.
	UseTaskCycleTime	Das Modul erlaubt Zykluszeiten, die von der in Simulink festgelegten "Fixed Step Size" abweichen. Bei Multitasking-Modulen müssen alle Task-Zykluszeiten das gleiche Verhältnis zur zugehörigen Simulink-Sampletime aufweisen.
	UseModelStepSize	Das Modul verwendet für alle internen Berechnungen die in Simulink eingestellte SampleTime. Diese Einstellung ist vornehmlich für die Verwendung in Simulationen innerhalb der TwinCAT-Umgebung gedacht.
ExecutionSequence	Dieser Parameter ist nur in Modulen verfügbar, die ab TE1400 Version 1.1 generiert wurden. Mit ihm kann die Reihenfolge der Berechnungs- und Kommunikationsprozesse angepasst werden, um Jitter und Reaktionszeit für den jeweiligen Anwendungsfall zu optimieren. Mit TE1400 Version 1.0 generierte Module verwenden immer die Reihenfolge "**StateUpdateAfterOutputMapping** ". Die Unterschiede zwischen den verschiedenen Optionen sind unter „ Ausführungsreihenfolge [▶ 37]“ beschrieben.	
	IOAtTaskbeginn	Ausführungsreihenfolge: Eingangs-Mapping -> Ausgangs-Mapping -> Zustandsupdate -> Ausgangs-Update -> External mode Abarbeitung -> ADS Zugriff
	StateupdateAfterOutputMapping	Ausführungsreihenfolge: Eingangs-Mapping -> Ausgangs-Update -> Ausgangs-Mapping -> Zustandsupdate -> External mode Abarbeitung -> ADS Zugriff
	StateupdateBeforeOutpunkUpdate	Ausführungsreihenfolge: Eingangs-Mapping -> Zustandsupdate -> Ausgangs-Update -> Ausgangs-Mapping -> External mode Abarbeitung -> ADS Zugriff

Zugriff auf diese Parameter erhält man in der TwinCAT-Entwicklungsumgebung (XAE) über den Objektknoten unter folgenden Karteireitern:

- **Parameter (Init)** :



- **Blockdiagramm** :



Wird keiner dieser Karteireiter angezeigt, müssen die Simulink-Coder-Einstellungen für Parameterdarstellung in XAE angepasst werden.

3.6.3 Aufruf des generierten Moduls aus einem SPS-Projekt

Wurde der Aufruf-Parameter „CallBy" auf den Wert „Module“ gesetzt, rufen die zugewiesenen Tasks das Modul nicht automatisch auf. Um das generierte TcCom-Modul stattdessen über ein anderes Modul aufzurufen, kann auf dessen Schnittstellen zugegriffen werden. Das kann aus einem C++-Modul geschehen oder, wie im Folgenden gezeigt, aus der TwinCAT SPS.
Bei der Codegenerierung wird eine PLCopen-XML-Datei erzeugt. Sie finden diese Datei im Build-Verzeichnis *<MODEL_DIRECTORY>\<MODEL_NAME>_tct* und - wenn das Modul über den Publish-Step exportiert wurde - auch im Publish-Verzeichnis [▶ 23] des Moduls. Die Datei enthält POUs, die den Aufruf eines Simulink-Objektes aus der SPS vereinfachen, indem sie das Handling der Interface-Pointer kapseln. Die POUs können über **Import PLCopenXML** im Kontextmenü eines SPS-Projektknotens importiert werden.

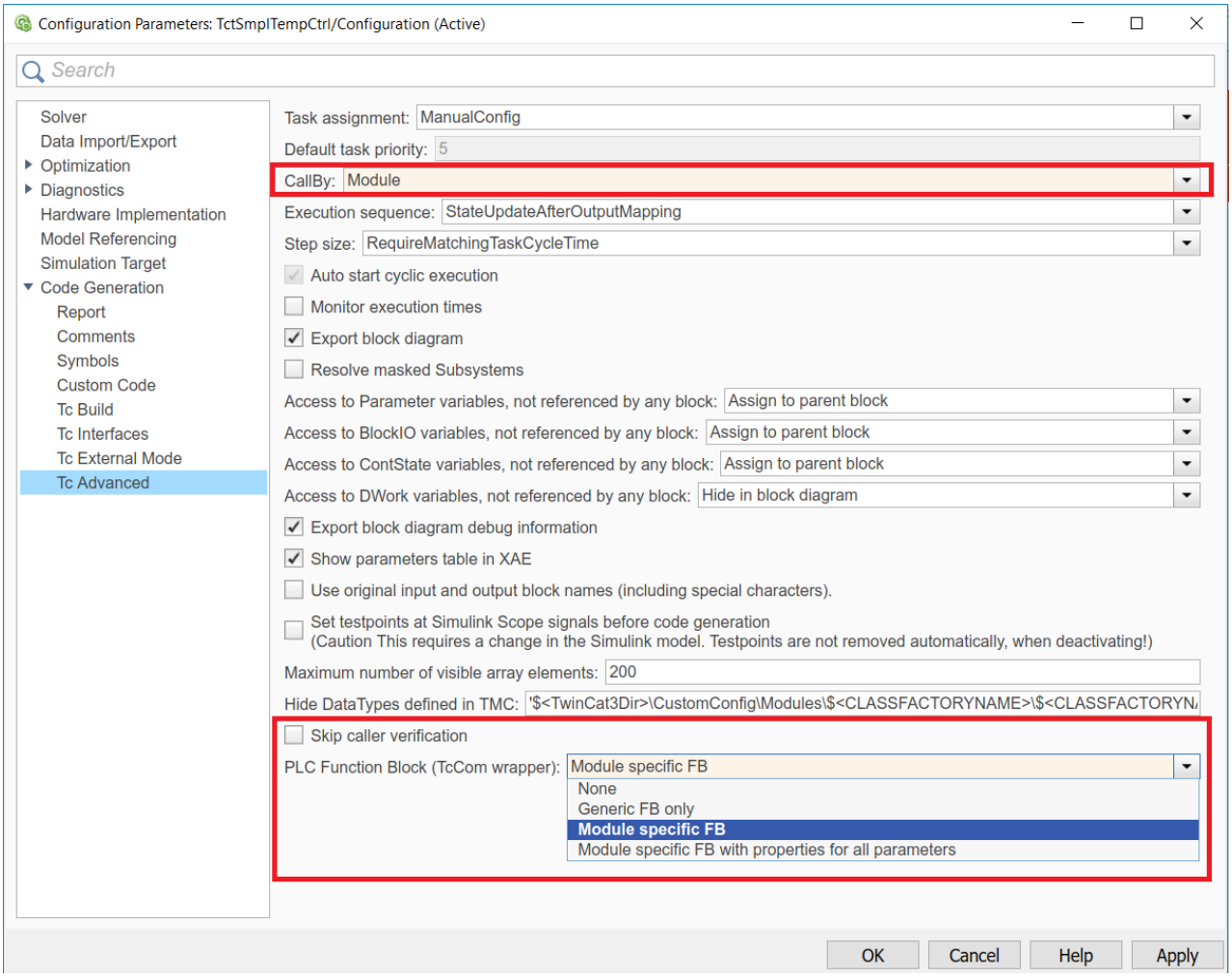
Die folgenden Beschreibungen gelten ab der Version 1.2.1216.0 des TE1400!

Konfiguration in Simulink

In den Einstellungen unter **Tc Advanced** wird zunächst **CallBy** auf **Module** gestellt (kann auch im TwinCAT Engineering nachträglich geändert werden).

Ab TE1400 Version 1.2.1230 ist der Parameter **Skip caller verification** sichtbar.

Ab TE1400 Version 1.2.1216.0 finden Sie den Parameter **PLC Function Block (TcCOM wrapper)**:



Hier kann zwischen folgenden Optionen gewählt werden:

Option	Beschreibung
None	Es wird keine PlcOpen-XML-Datei generiert
Generic FB only	Es wird nur der für alle mit dem TE1400 generierten Module gültige Funktions-Baustein **FB_TcMatSimObject** (siehe auch hier) sowie davon verwendete Datentypen generiert. Der Datenaustausch erfolgt über generische Methoden. Der Anwender muss hierzu modulspezifische Daten, wie beispielsweise Byte-Größen, Parameter-Index-Offsets oder DataArea-IDs kennen. **Versionshinweis:** Bis einschließlich TE1400 1.2.1216.0 ist dieser generische FB nur sinnvoll nutzbar, wenn ein davon abgeleiteter FB implementiert wird, welcher die Initialisierung interner Variablen übernimmt. **Ab Version 1.2.1217.0** stehen zusätzliche Methoden zur Verfügung, die eine direkte Initialisierung, auch ohne abgeleiteten FB ermöglichen.
Module specific FB	Zusätzlich zum generischen **FB_TcMatSimObject,** werden der modulspezifische Baustein **FB_<ModuleName>** und zugehörige Datentypen erzeugt. Die Struktur der Ein- und Ausgangsvariablen entspricht genau der Struktur der zugehörigen DataAreas des Moduls. Zum Datenaustausch können daher die Ein- und Ausgangsvariablen direkt zugewiesen werden, ohne beispielsweise die Größe der DataAreas oder die DataArea-IDs explizit angeben zu müssen.
Module specific FB with properties for all parameters	Der modulspezifische Baustein **FB_<ModuleName>** erhält zusätzliche Properties. Über diese Properties können Parameter des Moduls ausgelesen und ggf. geschrieben werden. Für jeden Modul-Parameter erhält der Baustein jeweils zwei Properties: „*ParameterName*_Startup“ und „*ParameterName*_Online“.

Der modulspezifische Funktions-Baustein

FB_<ModuleName> ist abgeleitet von FB_TcMatSimObject, stellt also auch die oben beschriebenen Methoden und Properties zur Verfügung. Zusätzlich sind hier folgende Properties implementiert:

Public Properties:

Methode	Datentyp	Beschreibung
InitData	ST_<ModuleName>_InitData	Speichert die Startup-Werte der Modulparameter zur Initialisierung einer Modul-Instanz. Die Werte werden während der Zustandsübergänge des Moduls von INIT nach PREOP per SetObjState() an das Modul übertragen. Erforderlich bei dynamischer Instanziierung.
<ParameterName>_Startup	<ParameterType>	Bei entsprechender Konfiguration des Coders für alle Parameter verfügbar. Erlaubt einen übersichtlicheren Zugriff auf das entsprechende Element der InitData-Struktur (lesend und schreibend).
<ParameterName>_Online	HRESULT	Bei entsprechender Konfiguration des Coders für alle Parameter verfügbar. Liest oder schreibt Online-Werte des entsprechenden Modulparameters.

Hinweise bezüglich FB with properties for all parameters

Wird unter Tc Interfaces im Bereich **Parameter access** der Punkt **Process image** auf **Internal DataArea** gesetzt, wird eine Property für die Gesamtheit aller Parameter angelegt. Diese müssen dann als Gesamtheit gelesen und geschrieben werden:

```
PROGRAM MAIN
VAR
// declare function block (details see below)
fbControl : FB_TctSmplTempCtrl(oid := 16#01010010);
// local PLC variable
ModelParameters : P_TctSmplTempCtrl_T;
END_VAR

// read all model parameters
ModelParameters := fbControl.ModelParameters_Online;
// change value
ModelParameters.Kp := 20;
// write all model parameters
fbControl.ModelParameters_Online := ModelParameters;
```

Wird unter Tc Interfaces im Bereich **Parameter access** der Punkt **Process image** auf **No DataArea** gesetzt, wird für jeden Modelparameter eine eigene Property erzeugt. Diese können dann direkt ohne lokale PLC Variable gelesen und geschrieben werden.

```
Fb<ModelName>.ModelParameters_<ParameterName>_Online
```

Referenzieren einer statischen Modul-Instanz:

Der FB kann genutzt werden, um auf vorher im XAE z. B. unter **System > TcCOM Objects** angelegte Modulinstanzen zuzugreifen. Für diesen statischen Fall muss die Objekt-ID der entsprechenden Modulinstanz bei der Deklaration der FB-Instanz übergeben werden:

```
fbStatic : FB_<ModuleName>(oid:=<ObjectId>);
```

Die Objekt-ID finden Sie in der Instanz des TcCOM unter dem Reiter **Object**.

Bespielcode

Das folgende Code-Beispiel zeigt die Verwendung eines modulspezifischen Funktionsbausteins in einem einfachen SPS-Programm in ST-Code am Beispiel eines Objektes der Modulklasse „TempContr“ mit der ObjectID `0x01010010`:

```
PROGRAM MAIN
VAR
// declare function block with ID of referenced Object
fbTempContr : FB_TempContr(oid:= 16#01010010 );
// input process image variable
nInputTemperature AT%I* : INT;
// output process image variable
```

```
bHeaterOn AT%Q* : BOOL;
END_VAR
IF (fbTempContr.State = TCOM_STATE.TCOM_STATE_OP) THEN
// set input values
fbTempContr.stInput.FeedbackTemp := nInputTemperature;
// execute module code
fbTempContr.Execute();
// get output values
bHeaterOn := fbTempContr.stOutput.HeaterOn;
END_IF
```

Erzeugen und Referenzieren einer dynamischen Modul-Instanz:

Für den Fall <ObjectId> = 0, versucht der FB dynamisch eine Instanz des TcCom-Moduls zu erzeugen:

```
fbDynamic : FB_<ModuleName>(oid:=0);
```

In diesem Fall taucht die Modulinstanz nicht im Konfigurations-Baum des XAE auf, sondern erscheint erst zur Laufzeit (also nach der Initialisierung der SPS-Instanz) in der „Project Objects“-Tabelle des Knotens **System > TcCOM Objects**.

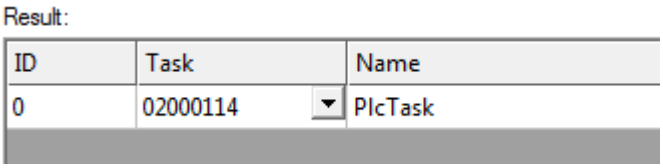
Voraussetzung für die dynamische Instanziierbarkeit eines TcCOM-Moduls ist, dass die zugehörige „Class Factory“ geladen ist. Dazu muss für die „Class Factory“ des Moduls unter dem Karteireiter **Class Factories** des Knotens **System > TcCOM Objects** die Checkbox **Load** (oder bei Verwendung des „TwinCAT Loaders“ die Checkbox **TC Loader**) gesetzt sein. Der Name der „Class Factory“ eines aus Simulink generierten TcCOM-Moduls ist in der Regel gleich dem Modulnamen, wobei der ClassFactory-Name allerdings auf weniger Zeichen begrenzt ist.

Weitere Voraussetzung für das dynamische Instanziieren eines Moduls ist die ausreichende Verfügbarkeit dynamischen Speichers. Dazu muss der ADS-Router-Speicher ausreichend groß gewählt sein.
Beispielcode:

```
PROGRAM MAIN
VAR
// declare function block
fbTempContr_Dyn : FB_TempContr(oid:= 0 );
// input process image variable
nInputTemperature AT%I* : INT;
// output process image variable
bHeaterOn AT%Q* : BOOL;
// reset error code and reinitialize Object
bReset: BOOL;
// initialization helpers
stInitData : ST_TempContr_InitData;
bInitialized : BOOL;
END_VAR
IF (NOT bInitialized) THEN
stInitData := fbTempContr_Dyn.InitData; // read default parameters
// adapt required parameters:
stInitData.ContextInfoArr_0_TaskOid := 16#02010020; // oid of the plc task
stInitData.ContextInfoArr_0_TaskCycleTimeNs := 10 * 1000000; // plc task cycle time in ns
stInitData.ContextInfoArr_0_TaskPriority := 20; // plc task priority
stInitData.CallBy := TctModuleCallByType.Module;
stInitData.StepSize := TctStepSizeType.UseTaskCycleTime;
// set init data, copied to module the next time it switches from INIT to PREOP:
fbTempContr_Dyn.InitData := stInitData;
bInitialized := TRUE;
ELSIF (fbTempContr_Dyn.State<TCOM_STATE.TCOM_STATE_OP) THEN
// try to change to OP mode
fbTempContr_Dyn.State := TCOM_STATE.TCOM_STATE_OP;
ELSIF (NOT fbTempContr_Dyn.Error) THEN
// set input values
fbTempContr_Dyn.stInput.FeedbackTemp := nInputTemperature;
// execute module code
fbTempContr_Dyn.Execute();
// get output values
bHeaterOn := fbTempContr_Dyn.stOutput.HeaterOn;
END_IF

IF (bReset) THEN
IF (fbTempContr_Dyn.Error) THEN
fbTempContr_Dyn.ResetHresult();
END_IF
fbTempContr_Dyn.State := TCOM_STATE.TCOM_STATE_INIT;
END_IF
```

Task-Kontext-Einstellung

In den Kontexteinstellungen [▶ 42] einer **statischen** Modulinstanz muss die SPS-Task konfiguriert sein, aus welcher der Aufruf erfolgt.



Einer **dynamischen** Modulinstanz muss die Objekt-ID der SPS-Task über die InitData-Struktur übermittelt werden. Falls vorhanden, kann das entsprechende InitData-Element über das Property „ContextInfoArr_<contextId>_TaskOid_Startup“ gesetzt werden.

Beim Aufruf des TcCOM Moduls wird eine Kontext-Verifikation durchgeführt und ein entsprechender Fehler ausgegeben, wenn der Kontext nicht korrekt ist. Diese Verifikation benötigt Zeit und wird bei jedem Call durchgeführt. Aus diesem Grund kann diese Verifikation über die Checkbox **Skip caller verification** im **Tc Advanced** Dialog, siehe Skip caller verification [▶ 40], deaktiviert werden.

Import mehrerer PLCopen-XML: FB_TcMatSimObject

Der generische Baustein **FB_TcMatSimObject** ist für **alle** mit dem TE1400 (ab V1.2.12016.0) generierten Module gleich. **Auch bei der Verwendung für unterschiedliche Module muss er nur einmal in das SPS-Project importiert werden**.

Beschreibung des Funktionsbausteins:

Public Methods

Methode	Rückgabe-Datentyp	Beschreibung
Execute	HRESULT	Kopiert die Daten der InputDataArea-Strukturen vom FB zur Modulinstanz (des Objektes), ruft die zyklischen Methoden des Objektes auf und kopiert die Daten aus den Output-DataAreas zurück in die zugehörigen Datenstrukturen des FB
GetObjPara	HRESULT	Liest Parameter-Werte per PID (Parameter ID = ADS Index Offset) aus dem Objekt aus
SetObjPara	HRESULT	Schreibt Parameter-Werte per PID (Parameter ID = ADS Index Offset) in das Objekt
ResetHresult		Quittiert Fehlercodes, die bei der Initialisierung des FB oder beim Aufruf von „Execute()“ aufgetreten sind.
SaveOnlineParametersForInit	HRESULT	Liest die aktuellen Online-Werte der Parameter aus dem Objekt und speichert sie in der Parameter-Struktur hinter der FB-Variablen pInitData, falls diese existiert
SetObjState	HRESULT	Versucht den TCOM_STATE des Objektes schrittweise zum Sollzustand zu führen
AssignClassId		Ab TE1400 1.2.1217.0: Setzt für den Fall der **Referenzierung eines statischen Objektes** die erwartete Class-ID. Diese wird mit der Class-ID des referenzierten Moduls verglichen, um Probleme durch Inkompatibilitäten zu vermeiden. Wird keine Class-ID zugewiesen, entfällt diese Kompatibilitätsprüfung. Zur **Erzeugung eines dynamischen Objektes** muss die Class-ID über diese Methode definiert werden.
SetInitDataInfo		Ab TE1400 1.2.1217.0: Übergibt den Zeiger auf die zu verwendende InitData-Struktur. Diese Struktur muss bei Verwendung dynamischer Objekte angelegt und mit notwendigen

Methode	Rückgabe-Daten-typ	Beschreibung
		Parameterwerten initialisiert werden. Für statische Objekte ist diese Struktur optional. Sie ermöglicht dann eine spätere Neuinitialisierung des Objektes. In diesem Fall kann die Struktur auch durch Aufruf von „SaveOnlineParametersForInit“ initialisiert werden.
SetDataAreaInfo		Ab TE1400 1.2.1217.0: Übergibt den Zeiger auf ein Array von DataArea-Informationen vom Typ ST_TcMatSimObjectDataAreaInfo, sowie die Anzahl der Elemente dieses Arrays. Diese Informationen werden für den zyklischen Datenaustausch innerhalb der Methode „Execute“ benötigt.

Public Properties

Methode	Datentyp	Beschreibung
ClassId	CLSID	Gibt die ClassId des Moduls zurück
Error	BOOL	Gibt TRUE zurück, wenn aktuell ein Fehler ansteht, der Quittiert werden muss
ErrorCode	HRESULT	Gibt den aktuellen Fehlercode zurück
State	TCOM_STATE	Gibt den aktuellen TCOM_STATE des Objektes zurück oder versucht ihn schrittweise zum Sollzustand zu führen

Referenzierung einer Modul-Instanz

Auch FB_TcMatSimObject kann, genau wie der von ihm abgeleitete modulspezifische FB, mit der Objekt-ID einer statischen Modulinstanz oder mit 0 instanziiert werden:

```
fbStatic : FB_TcMatSimObject(oid:=<ObjectId>); // Referenz auf statisches TcCom-Objekt
fbDynamic : FB_TcMatSimObject(oid:=0); // Erzeugen eines dynamisches TcCom-Objektes
```

3.6.4 Verwendung des ToFile Blocks

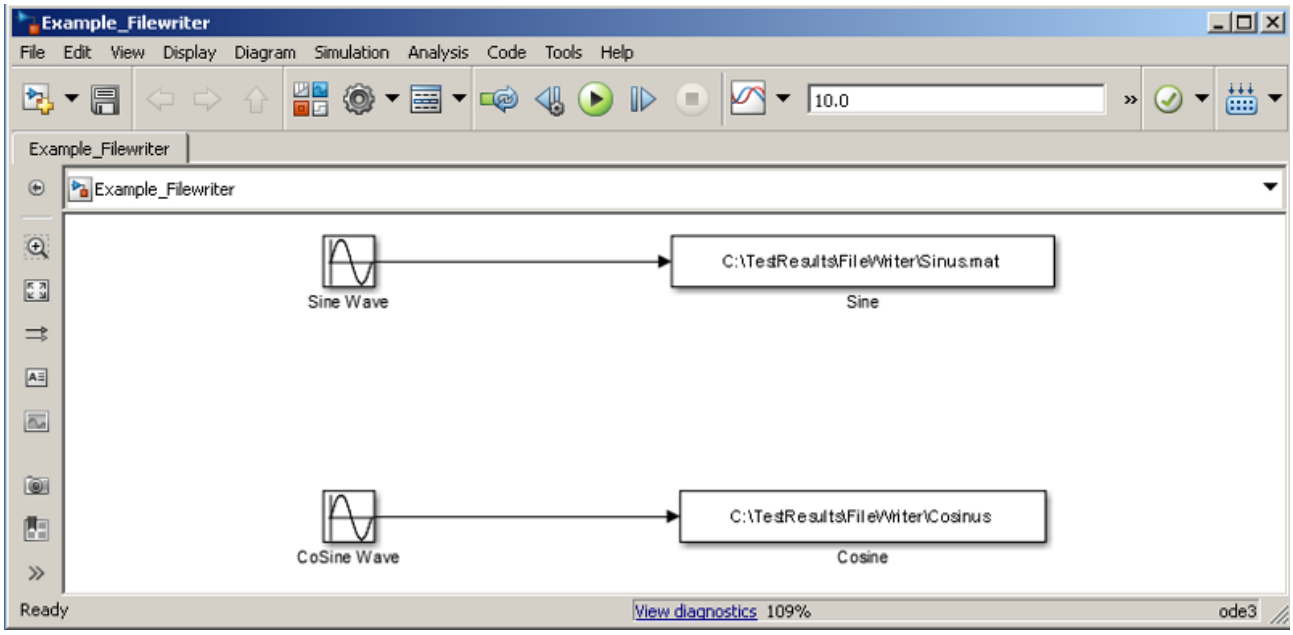
Mit Hilfe des ToFile Blocks aus der Simulink Standardbibliothek ist es möglich Signale in einem MAT-file zu loggen. Auch aus einem erstellten TcCOM heraus, lässt sich dieser Block weiterhin aus der TwinCAT-Runtime heraus nutzen.

Für den Dateisystemzugriff aus der Echtzeit wird ein zusätzliches TcCOM „TcExtendedFilewriter“ erzeugt und mit dem TcCOM mit dem ToFile Block (im folgenden Simulink-TcCOM genannt) verknüpft. Der TcExtendedFilewriter empfängt dann die Daten aus dem zugeordneten TcCOM und schreibt diese in ein MAT-File (mat4).

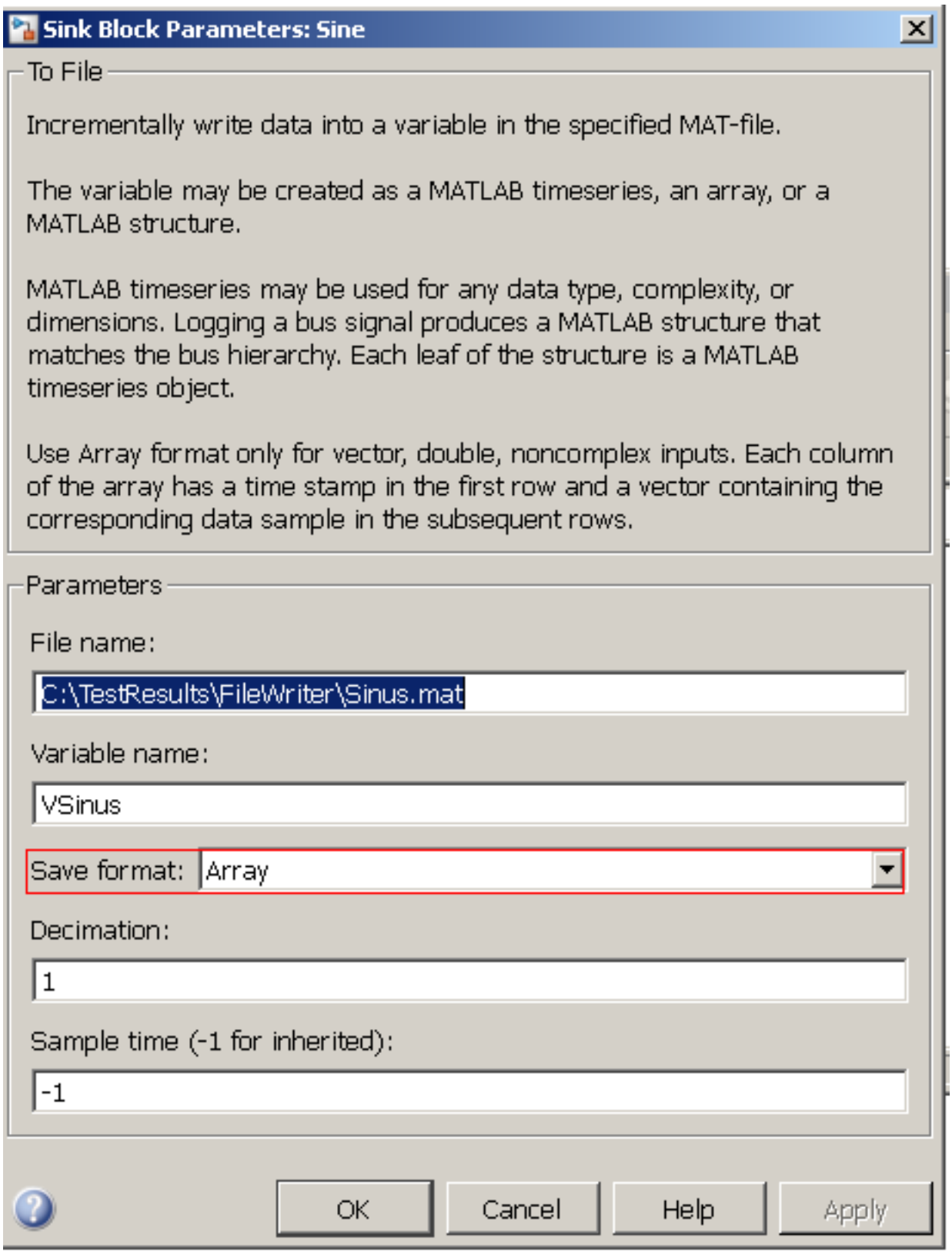
Im Folgenden sind Schritt für Schritt anhand eines Beispiels die Einstellungen in Simulink und in TwinCAT beschrieben.

Konfiguration im Simulink-Modell

Als einfaches Beispiel diene ein Modell mit einer Sinus- und ein Cosinus-Quelle. Beide Signale sollen mit je einem ToFile Block geloggt werden.



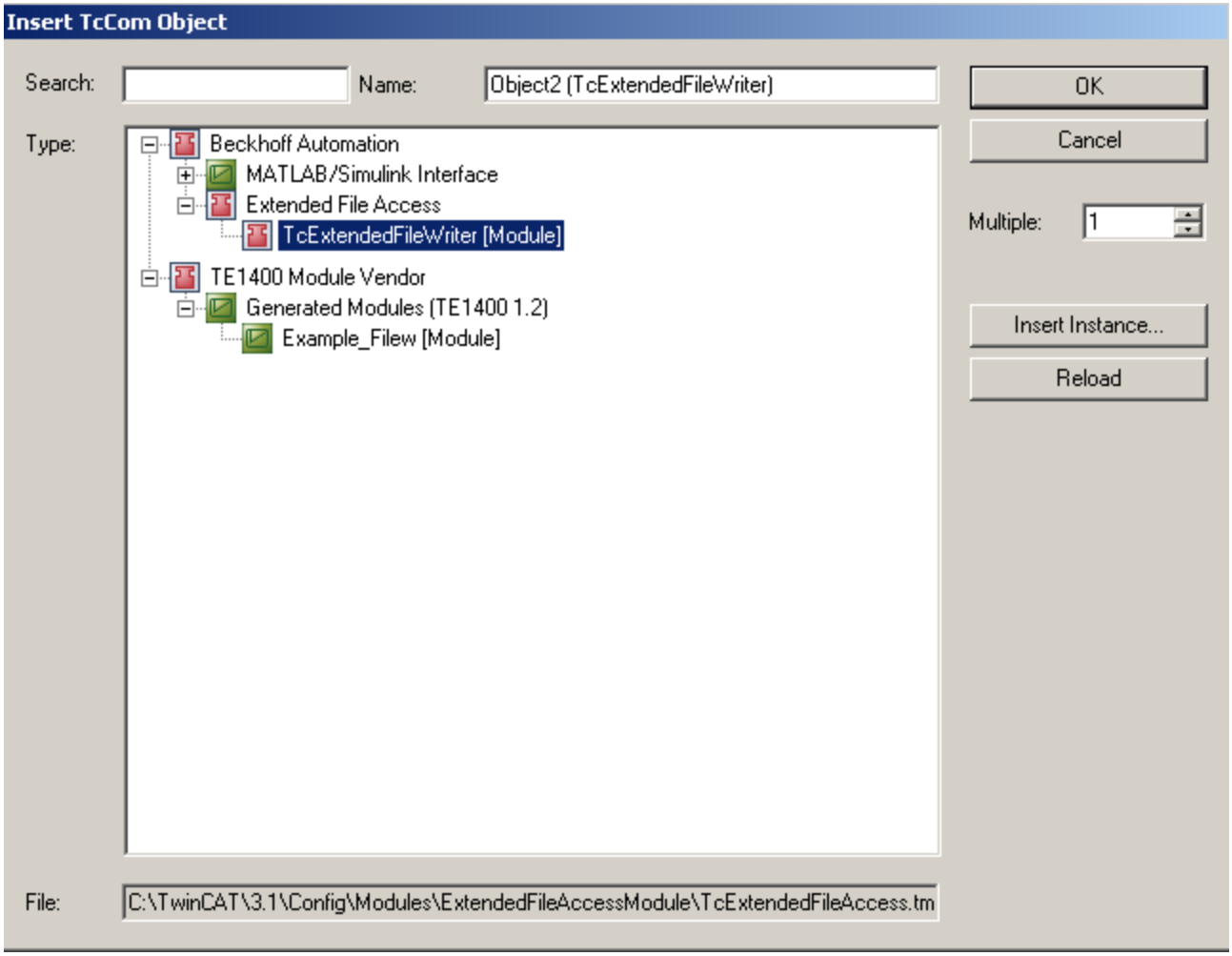
Um die Codegenerierung für die ToFile-Blöcke zu ermöglichen, muss als Format **Array** eingestellt werden:



Das Modell ist nun bereit für die Codegenerierung.

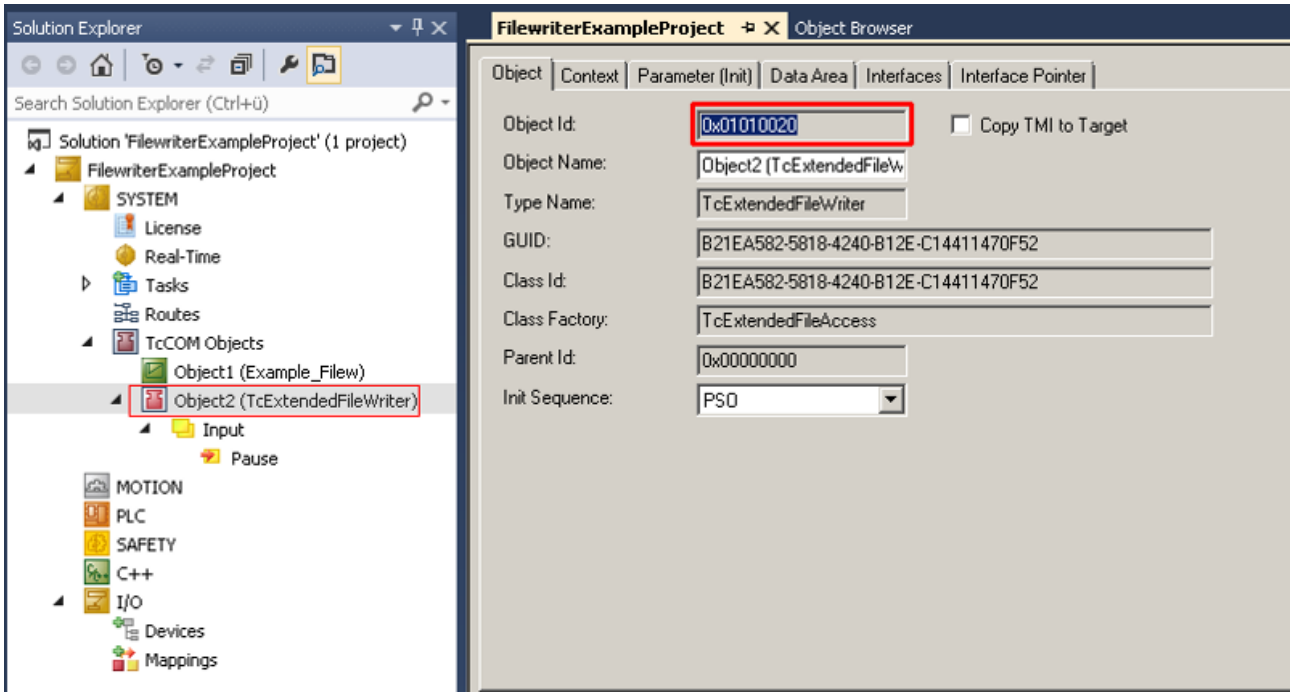
Konfiguration in TwinCAT

Um aus dem generierten Simulink-TcCOM schreiben zu können, wird das mit dem TE1400 installierte TcCOM *TcExtendedFilewriter* benötigt. Dieses nimmt die Daten aus dem Simulink Objekt an und legt die Daten im Dateisystem ab. Das Modul ist im TcCOM-Browser unter *Beckhoff Automation* > *Extended File Access* > *TcExtendedFileWriter* zu finden:

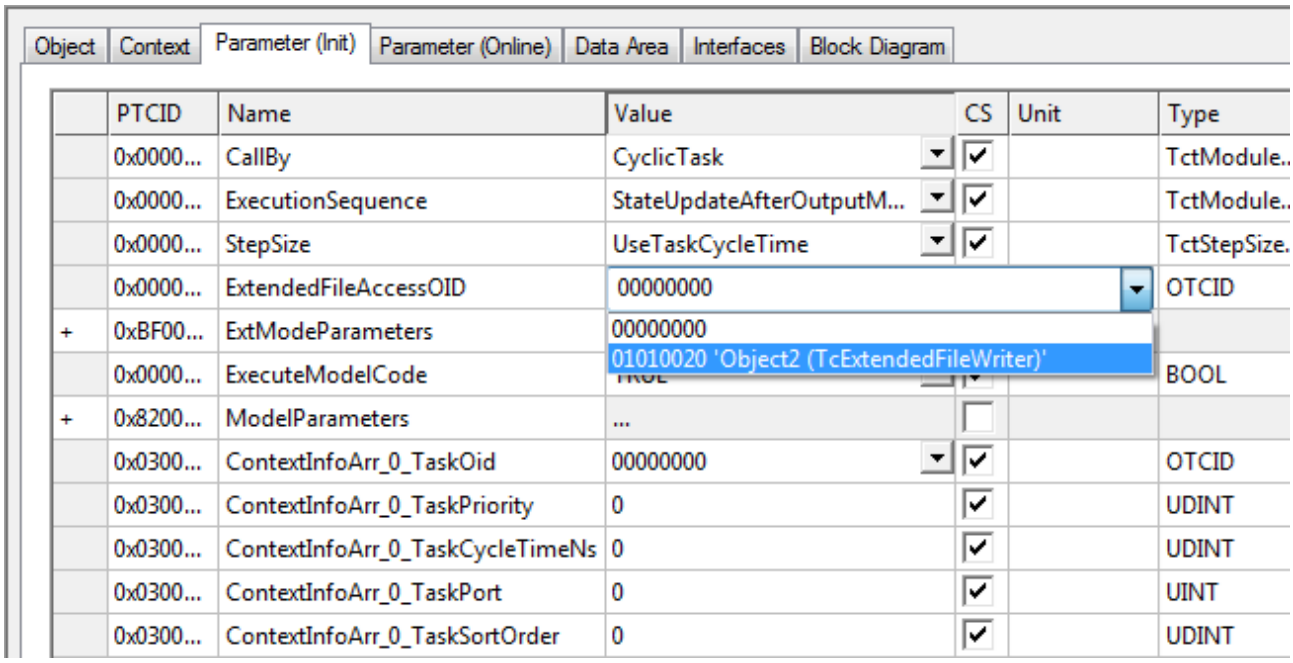


Zunächst werden beide TcCOM instanziiert. Beide Objekte können an eine gemeinsame oder an getrennte Tasks gehängt werden. Um eine Verbindung zwischen beiden Objekten herzustellen wird dem Simulink-TcCOM die ObjektID der *TcExtendedFilewriter*-Instanz mitgeteilt.

Die ObjectID ist unter dem Karteireiter **Object** zu finden.



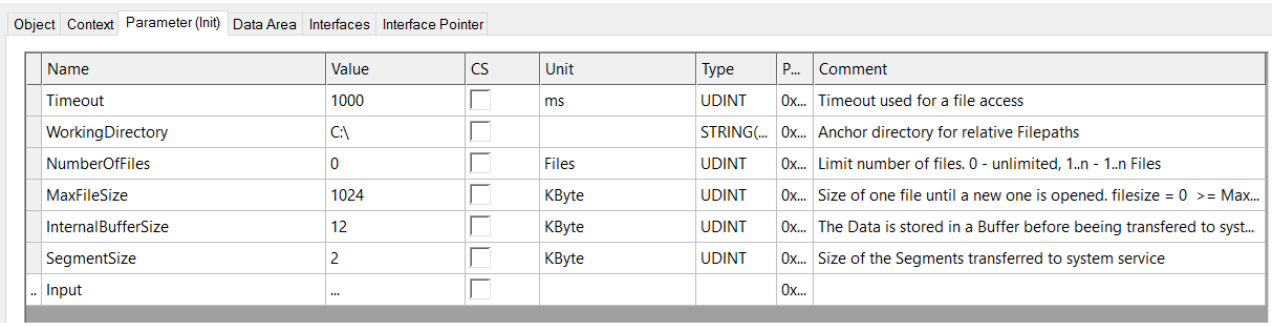
Die ObjectID wird dann unter dem Karteireiter „Parameter (Init)“ des Simulink-TcCOM für den Parameter **ExtendedFileAccessOID** eingefügt:



Es lassen sich auch mehrere Simulink-TcCOM mit einer *TcExtendedFileWriter*- Instanz verknüpfen. Hierbei ist zu beachten, dass keine Konflikte zwischen den verwendeten Dateinamen auftreten. Es lassen sich auch mehrere *TcExtendedFileWriter*-Instanzen parallel benutzen. Beispielsweise kann jedes Simulink-TcCOM mit einem ToFile-Block ein eine eigene *TcExtendedFileWriter*-Instanz verwenden.

Parametrierung der TcExtendedFileWriter-Instanz

Unter dem Karteireiter Parameter (init) der Instanz des *TcExtendedFileWriter* kann das Verhalten des Objekts angepasst werden.



Timeout:

Es kann ein Timeout gesetzt werden

Working Directory:

Wird im ToFile Block ein relativer Pfad genutzt, z.B. /logData, wird der volle Pfad über den Parameter Working Directory aufgelöst.

Number of Files:

Es ist möglich die Dateianzahl zu begrenzen. Steht der Parameter auf 0, wird keine Begrenzung vorgenommen.

Max File Size:

Um auch während des laufenden Moduls auf die aufgenommenen Daten zugreifen zu können, wird nach dem Erreichen einer festgelegten Dateigröße (standardmäßig 1MB) die Datei abgeschlossen und eine neue geöffnet.

Internal Buffer Size:

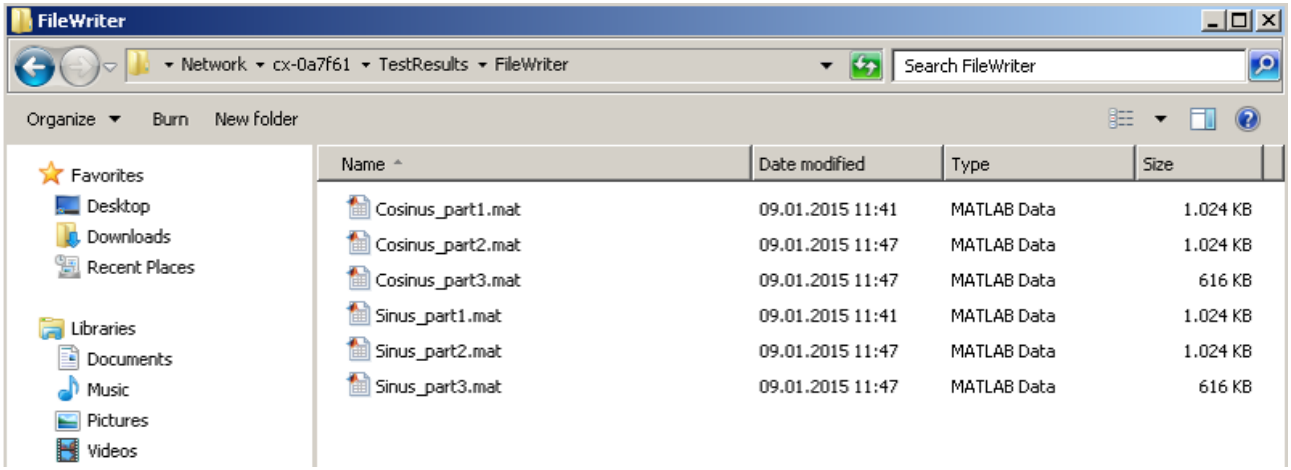
TwinCAT-seitig wird ein Puffer der Größe InternalBufferSize angelegt, aus welchem dann die Daten geschrieben werden.

Segment Size:

Es wird mit jedem Schreibbefehl der *TcExtendedFileWriter*-Instanz ein Segment der Größe SegmentSize aus dem Internal Buffer in das spezifizierte File geschrieben. Die theoretisch maximal mögliche Datenrate zum Schreiben setzt sich entsprechend aus der SegmentSize und der Zykluszeit des TcExtendetFileWriter zusammen (die *TcExtendedFileWriter*-Instanz muss nicht die gleiche Zykluszeit haben wie das zugeordnete Simulink-TcCOM Modul). Jedoch ist zu beachten, dass ein Schreibbefehl möglicherweise noch nicht abgeschlossen ist, wenn der nächste Zyklus beginnt. Dann wird in diesem Zyklus der Schreibbefehl ausgesetzt. Es handelt sich somit um eine Abschätzung für den *best-case*.

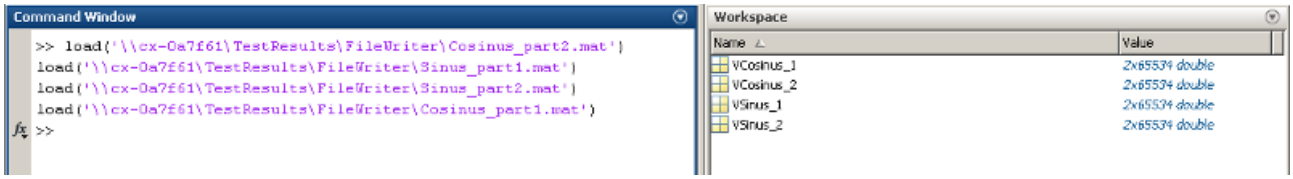
Das TwinCAT-Projekt kann nun aktiviert werden.

Um auch während des laufenden Moduls auf die aufgenommenen Daten zugreifen zu können, wird nach dem Erreichen einer festgelegten Dateigröße (standardmäßig 1MB) die Datei abgeschlossen und eine neue geöffnet, (im Bild: *_part1.mat und *_part2.mat sind abgeschlossen, an *_part3.mat wird noch geschrieben):

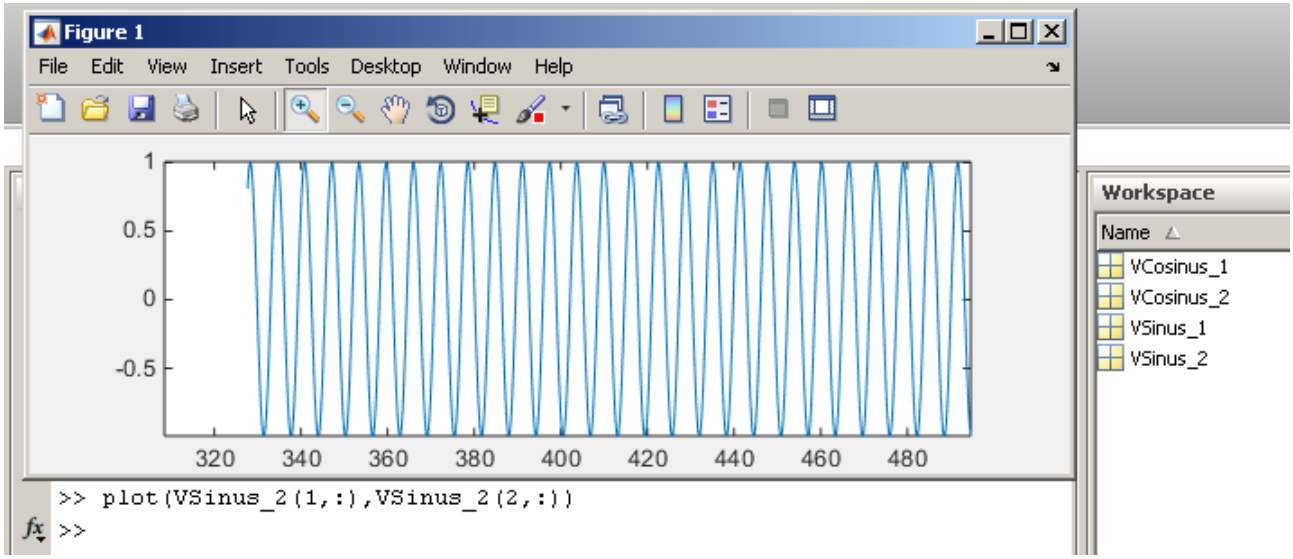


Um ein dauerhaftes Schreiben zu unterbinden, besitzt das TcExtendedFilewriter-Objekt den Eingang Pause. Wenn der Eingang auf TRUE gesetzt wird, wird die aktuell geschriebene Datei geschlossen und alle nun eingehenden Daten verworfen. Wird der Eingang wieder auf FALSE gesetzt, so wird eine neue Datei geöffnet um die eingehenden Daten zu schreiben.

In MATLAB lassen sich die geschlossenen Dateien wie gewohnt öffnen:

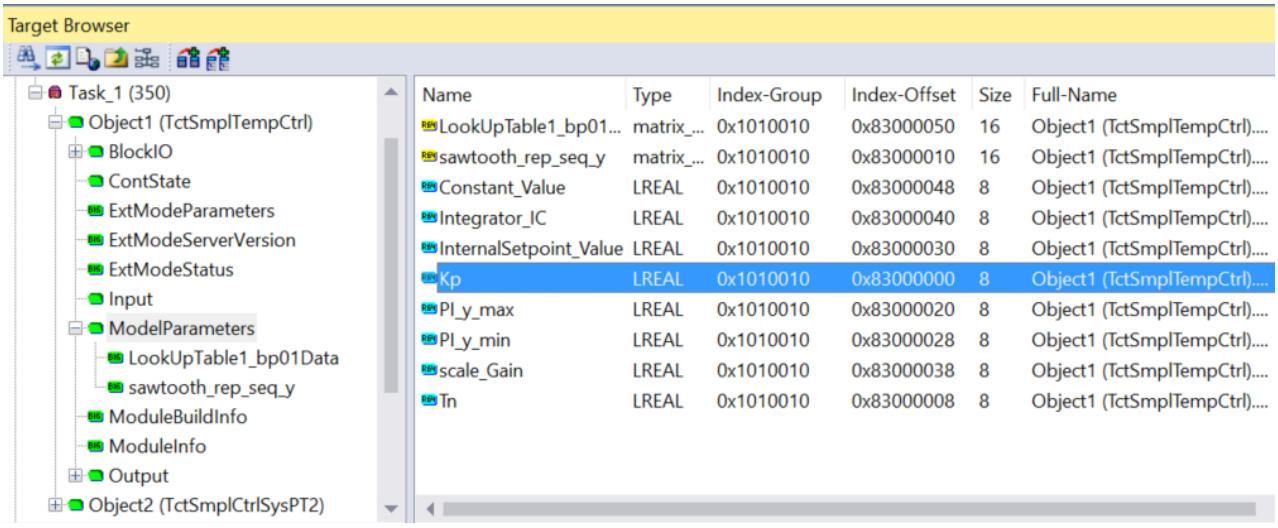


Der Plot zeigt die erwartete Sinusschwingung:

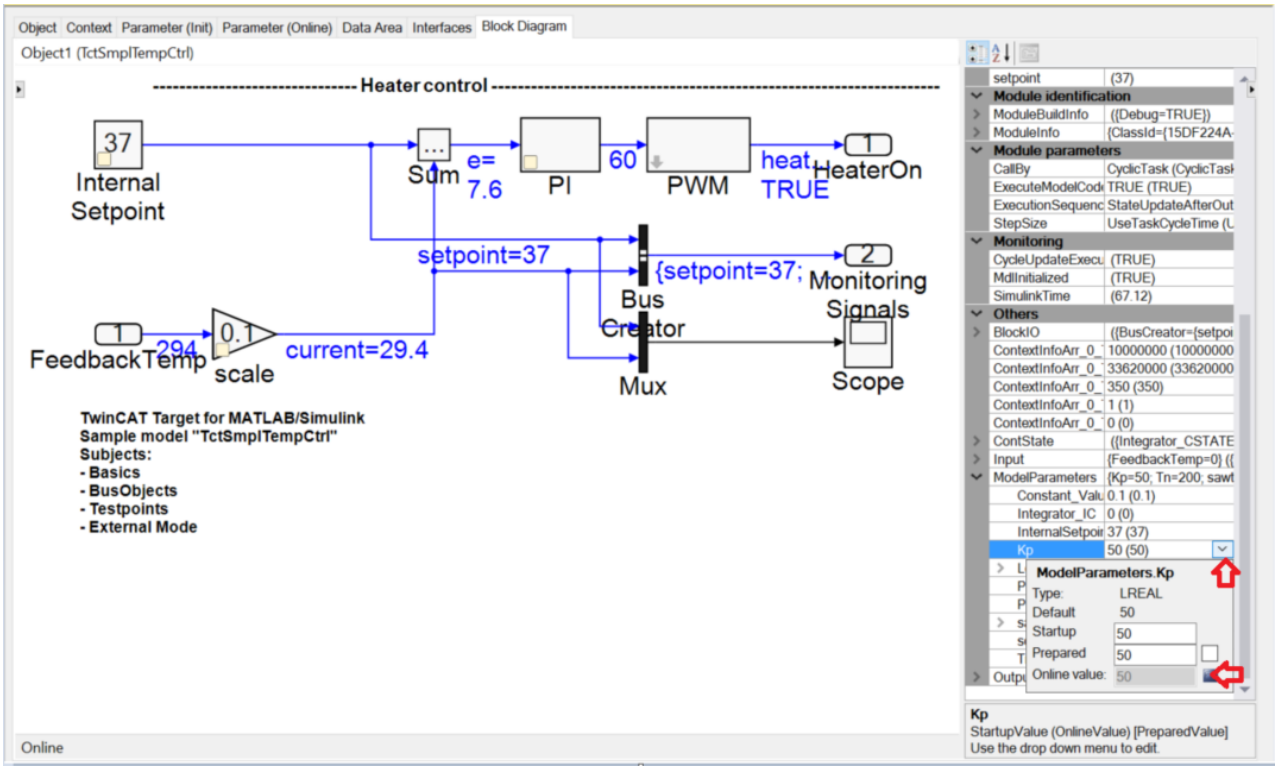


3.6.5 Signalzugriff per TwinCAT 3 Scope

Mit dem TwinCAT 3 Scope ist der Zugang zu allen Variablengruppen möglich, für die zumindest lesender ADS-Zugriff aktiviert wurde, siehe Tc Interfaces [▶ 27] in Simulink. Wenn der **Target Browser** zur Konfiguration des Scope verwendet werden soll, muss in Simulink unter Tc Interfaces die Option **… _CreateSymbols** ausgewählt sein. Ohne die entsprechende Symbolinformation können die aufzuzeichnenden Signale im Scope nur manuell per Index-Group und Index-Offset konfiguriert werden.



Alternativ kann das Scope mit dem entsprechenden Symbol direkt aus der TwinCAT-Entwicklungsumgebung (XAE) gestartet werden. Dafür wird im der Dropdown-Fenster des **Blockdiagramm-Browsers** für jedes verfügbare Signal die Schaltfläche **Show in Scope** angezeigt, wenn die Modulinstanz auf dem Zielsystem läuft.



Die Signale können auch bequem mit der rechten Maustaste aus dem Dropdownmenü (rechte Leiste in obiger Abbildung) oder von den blauen Signallinien im Blockdiagramm (Hauptfenster in obiger Abbildung) in die Scope-Konfiguration gezogen werden (drag & drop). Auch ein Block kann mit der rechten Maustaste in die Scope-Konfiguration gezogen werden, um alle Ein- und Ausgänge dieses Blocks aufzuzeichnen.

3.7 FAQ

3.7.1 Funktioniert die Code-Generierung auch wenn ich S-Functions in mein Modell einbinde?

S-Functions können in Simulink®-Modelle eingebunden und diese dann auch für die Verwendung in der TwinCAT Runtime gebaut werden.

Es gibt verschiedenen Workflows, die auf unterschiedlichen Gegebenheiten beruhen. Die gängigen 4 Fälle werden hier kurz erläutert und die entsprechende Lösung für die Integration in den Code-Generierungs-Prozess aufgezeigt.

Fall 1: Ich habe Zugriff auf den Quellcode, der in der S-Function verwendet wird.

In diesem Fall kann in der S-Function der Ort des Quellcodes angegeben werden. Der Code-Generierungs-Prozess kann ohne weitere Schritte direkt gestartet werden. Der Quellcode wird gefunden und entsprechend für die Verwendung in TwinCAT kompiliert.

Fall 2: Ich habe eine inlined S-Function (TLC Datei bei)

In diesem Fall kann der Code-Generierungs-Prozess kann ohne weitere Schritte direkt gestartet werden, da der einzufügende Code der S-Function in der TLC Datei vorliegt. Wie eine TLC Datei für eine S-Function erstellt wird, ist der Dokumentation von The MathWorks® zu entnehmen: https://de.mathworks.com/help/simulink/sfg/how-to-implement-s-functions.html

Fall 3: Ich habe ein kompiliertes MEX-File ohne Zugriff auf den Quellcode

In diesem Fall wurde von Dritten eine Funktion erstellt und als MEX-File kompiliert. Der Quellcode oder die TLC Datei wurde z.B. aus Know-How-Schutz-Gründen nicht mitgeliefert. In diesem Fall muss die Dritte Partei die das MEX-File liefert, ihren Quellcode als TwinCAT-fähige Bibliothek kompilieren, damit in der Echtzeit auf diese Bibliothek gelinked werden kann. Eine Anleitung ist unter Beispiele zu finden: SFunStaticLib [▶ 80].

Fall 4: Ich binde in meine S-Function (dessen Quellcode vorliegt) wiederum eine MEX-Bibliothek ein, dessen Quellcode mir nicht vorliegt.

Auch in diesem Fall muss die Dritte Partei die das MEX-File liefert, ihren Quellcode als TwinCAT-fähige Bibliothek kompilieren. Eine Anleitung ist unter Beispiele zu finden: SFunWrappedStaticLib [▶ 86].

3.7.2 Warum treten im generierten TwinCAT-Modul zur Laufzeit FPU/SSE exceptions auf, aber nicht im Simulink-Modell?

Simulink behandelt in den Standardeinstellungen floating point exceptions ggf. anders als TwinCAT 3.

Um das Verhalten bei floating point exceptions zwischen Simulink und TwinCAT anzugleichen, können Sie unter Model Configuration Parameters in Simulink im Abschnitt **Diagnostics >Data Validity** in der Box Signals wählen:

- Division by singular matrix: error
- Inf or NaN block output: error

Um eine SSE exception in TwinCAT zu **debuggen** nutzen Sie bitte den C++ debugger, siehe dazu den Bereich Debuggen [▶ 202] in der TE1400 Dokumentation. Unter der Voraussetzung, dass Sie ihr Modell als „debug“ Modul gebaut haben und bei aktivierterm C++ debugger, genügt es auch sich nach dem Start von TwinCAT zum Prozess zu attachen, falls bereits in den ersten Zyklen die Exception auftritt. Häufig tritt die SSE exception direkt im ersten Zyklus auf. Hier kommt es schnell zu einer Division durch Null, wenn bestimmte Signale mit Null initialisiert werden.
Eine weitere Möglichkeit den SSE exceptions zu begegnen ist das **Deaktivieren** von floating point exceptions. Diese können im System Manager unter Tasks ausgeschaltet werden (Deaktivierung der Checkbox *floating point exceptions*). Entsprechend gilt diese Einstellung dann für alle Module, die durch diese Task angesprochen werden. Bei Auftreten einer exception wird dann ein NaN erzeugt und kein Fehler ausgegeben.

HINWEIS

Deaktivierung von floating point excpetions

NaN-Werte dürfen in anderen SPS-Bibliotheken, insbesondere als Stellwerte in Funktionen für Motion Control und zur Antriebssteuerung, nur verwendet werden, wenn sie ausdrücklich zugelassen sind! Anderenfalls können NaN-Werte zu potenziell gefährlichen Fehlfunktionen führen!

3.7.3 Nach Update von TwinCAT und/oder TE1400 bekomme ich bei einem bestehenden Modell eine Fehlermeldung.

Beschreibung der Situation:

Sie haben ein Simulink-Modell bereits erfolgreich in ein TcCOM gewandelt. Nachfolgend haben Sie ein Update des TwinCAT XAE und/oder des TE1400 vorgenommen. Sie möchten das Simulink-Modell nun nochmals übersetzen (z.B. haben Sie ein neues Feature des TE1400 genutzt, oder etwas am Modell verändert, oder auch nichts verändert). Nun erhalten Sie beim Publish Fehlermeldungen.

Mögliche Ursache sowie Lösung:

Im Build-Verzeichnis existiert bereits ein Order mit der Benennung <modelname>_tct, vgl. Welche Dateien werden automatisch bei der Codegenerierung und dem Publish erstellt? [▶ 62]. Dieser Order wurde mit den Sourcen der früheren Software-Version(en) erstellt. Unter Umständen kann es an dieser Stelle zu Konflikten kommen, wenn nun mit einem neuen Software-Stand ein neuer Publish-Vorgang angestoßen wird, welcher in selbigen Order die Sourcen ablegen möchte.

Einfache Lösung ist, den entsprechenden Order zu löschen, sodass alle Sourcen mit dem aktuellen Versionsstand aller Komponenten neu aufgebaut werden, wenn Sie das Modul bauen.

3.7.4 Warum ändern sich nach einem „Reload TMC/TMI“ die Parameter der TcCOM-Instanz nicht immer?

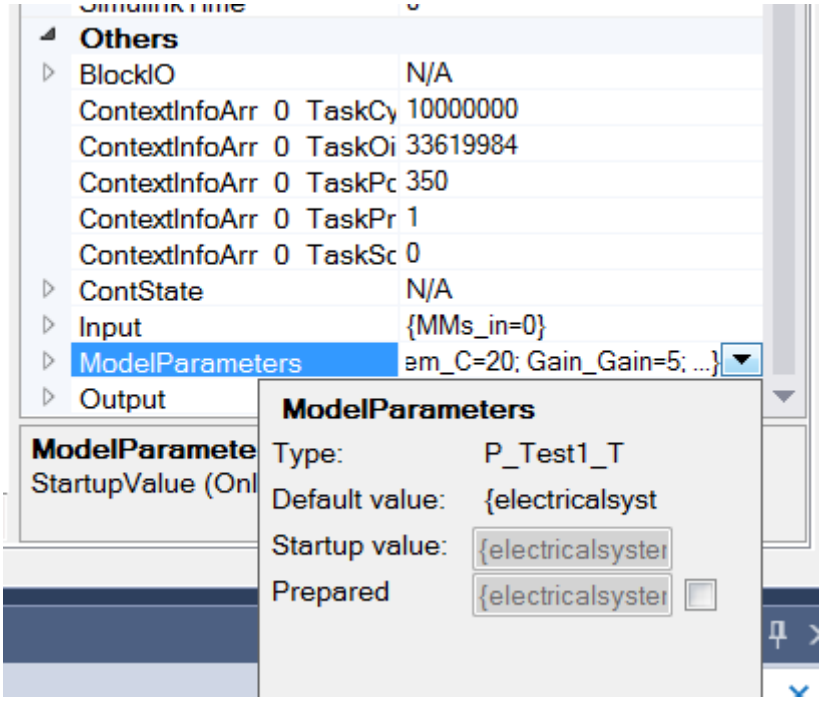
Die Beobachtung:

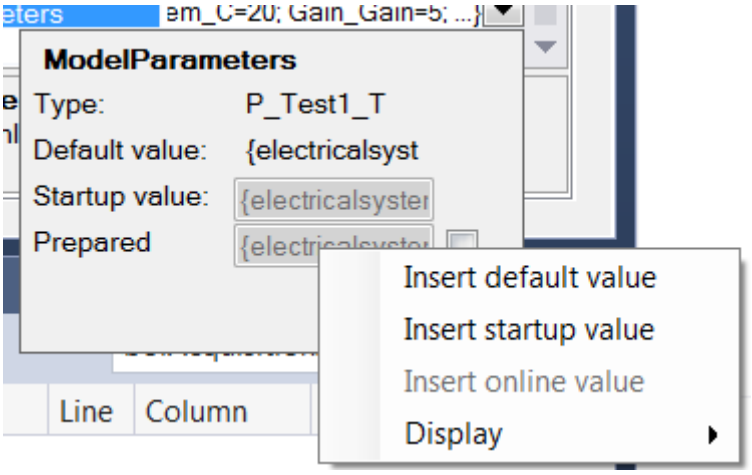
Sie haben eine existierende Instanz eines TcCOM Objects in TwinCAT 3.
Wie bereits beschrieben, haben Sie die Möglichkeit die Modellparameter, z.B. die Parameter eines PID-Reglers, in TwinCAT über das exportierte Blockdiagram oder auch über den Karteireiter **Parameter (init)** des TcCOM Objekts außerhalb von Simulink zu verändern. Verändern Sie ihr Simulink-Modell in Simulink und erstellen ein neues TcCOM Objekt, können Sie dieses natürlich über den Aufruf **reload TMC/TMI** durch rechts-Klick auf das entsprechende TcCOM Objekt in TwinCAT aktualisieren – wobei Ihnen alle Verlinkungen erhalten bleiben, solange das Prozessabbild nicht verändert wird.

Es gilt zwei unterschiedliche Fälle zu betrachten

- Sie haben nur Modellparameter in Simulink verändert, z.B. PID Regelparameter
- Sie haben sowohl Modellparameter als auch weitere strukturelle Veränderungen am Modell vorgenommen

Im ersten Fall werden Sie feststellen, dass sich nach dem Aufruf **Reload TMC/TMI** die Parameter ihres TcCOM Objekt **nicht** verändert haben. Die Startup values werden von der vorherigen TcCOM-Instanz übernommen, so dass Ihre Einstellungen aus TwinCAT bezüglich dieser Modul-Instanz nicht verloren gehen. Möchten Sie die Modellparameter aus Simulink laden, können Sie diese durch Navigieren auf das Dropdown Menü der **ModelParameters** im Fenster des Blockdiagramms - rechte Seite, rechts-Klick auf **Startup value** oder **Prepared** und Auswahl von **Insert default value** laden. Die default values werden aus dem TMC-file geladen, so dass Sie die Parametereinstellungen aus Simulink übernehmen.





Alternativ können Sie auch das alte TcCOM Objekt löschen und das neue TcCOM Objekt einfügen. Dann gehen auch alle vorherigen Modellparameter verloren und das neu eingefügte Objekt besitzt dieselben Modellparameter wie das korrespondierende Simulink Modell.

Haben Sie zusätzlich zu Modellparametern weitere Veränderungen durchgeführt, ändert sich der Modellcode, wodurch die Beibehaltung von vorherigen Modellparameter-Einstellungen nur eingeschränkt durchführbar ist. In diesem Fall werden die TwinCAT-Modulparameter aus der vorherigen Instanz beibehalten, die der System Manager noch eindeutig zuordnen kann.

3.7.5 Nach einem "Reload TMC/TMI" Fehler "Source File <path> to deploy to target not found

Wenn Sie einen Reload TMC/TMI durchführen, achten Sie bitte darauf, dass Sie das TMC-File aus dem Publish-Verzeichnis verwenden: %TwinCAT3Dir%\CustomConfig\Modules\<MODULENAME> und **nicht** aus dem Build-Verzeichnis aus dem Ordner <MODULENAME>_tct.

Wenn Sie die TMC-Datei aus dem Build-Verzeichnis verwenden, kann TwinCAT den entsprechenden Treiber nicht finden und Sie erhalten die in der Überschrift genannte Fehlermeldung beim Start von TwinCAT.

3.7.6 Warum habe ich beim Start von TwinCAT einen ClassID Konflikt?

Die Class ID stellt eine unique Beziehung zwischen der tmc-Datei und dem zugehörigen Echtzeittreiber her.

Wenn Sie ein TcCOM Modul aus Simulink® heraus mit dem TE1400 erzeugt und in einem TwinCAT Projekt instanziiert haben, ist in der Instanz der TcCOM die Class ID verankert und die Instanz erwartet einen entsprechenden Treiber mit ebenfalls dieser Class ID. Gehen Sie nun zurück zu Simulink® und erstellen erneut ein TcCOM mit gleichem Namen wie beim bereits instanziierten Modul wird eine neue tmc-Datei und auch neue Treiber im Publish-Verzeichnis abgelegt, welche eine neue Class ID tragen. Aktivieren Sie nun die TwinCAT-Konfiguration, ohne TwinCAT mitzuteilen, dass sich die Class ID verändert hat, erhalten Sie folgendes Verhalten:

Verhalten für TwinCAT Version < 4018:

Sie bekommen eine entsprechende Fehlermeldung mit dem Hinweis, dass die Class ID nicht zueinander passen.

Verhalten für TwinCAT Version ≥ 4018

Es wird der zur bestehenden Instanz im TwinCAT Projekt passende Treiber aus dem Projektordner _ModulInstall genutzt. Das Verhalten der Modul-Instanz bleibt also für das TwinCAT Projekt unverändert.

Wichtig: Es muss auch entsprechend unter Tc Build das lowest compatible TwinCAT build ≥ 4018 eingetragen sein, damit letzteres Verhalten eintritt. Siehe auch Modulgenerierung (Tc Build) [▶ 23].

Lösung:

Um das Verhalten des neu generierten TcCOM Modules in Ihrem TwinCAT Projekt nutzen zu können, können Sie auf der entsprechenden Instanz des TcCOM einen rechts-klick machen und wählen TMI/TMC-File -> **Reload TMI/TMC File.** Wählen Sie nun die tmc Datei in Ihrem Publish-Verzeichnis aus und bestätigen mit **OK**. Wenn Sie das Modul aus der SPS heraus aufrufen und die PLCopen.xml dafür importiert haben, müssen Sie diese ebenfalls neu importieren und im Dialog **replace the existing object** anwählen.

3.7.7 Warum sind per ADS übermittelte Werte unter Umständen abweichend von Werten die per output mapping übertragen werden?

Übermittlung der Ergebnisse von „minor time steps“

Abhängig von der konfigurierten Abarbeitungsreihenfolge [▶ 42] der Modulinstanz können die übermittelten ADS-Werte von den erwarteten Werten abweichen. Unterschiede können auftreten, wenn die zeitkontinuierlichen Zustandsgrößen nach dem „Output mapping“ aktualisiert werden, um die kürzeste Reaktionszeit zu erhalten:

Task - Zykluszeit						
Input mapping	Ausgangsaktualisierung	Output mapping	State update	Abarbeitung External Mode	ADS access	

Über ADS übermittelte Signalwerte können dann von den Werten abweichen, die via „Output mapping“ auf andere Prozessabbilder kopiert wurden. Der Grund hierfür ist, dass beim „State update“ einige Werte überschrieben werden. Mit anderen Worten: Die übermittelten Werte sind das Ergebnis der Berechnungen innerhalb untergeordneter Zeitschritte des verwendeten Solvers („minor time steps“), während beim „Output mapping“ die Ergebnisse übergeordneter Zeitschritte kopiert werden.
Das gilt auch für Daten, die via External Mode [▶ 32] übermittelt werden.

3.7.8 Gibt es Limitierungen hinsichtlich der Ausführung von Modulen in Echtzeit?

Nicht alle Zugriffe, die in Simulink® unter nicht-Echtzeit-Bedingungen möglich sind, können in der TwinCAT-Echtzeit-Umgebung durchgeführt werden. Im Folgenden werden bekannte Limitierungen beschrieben.

- **Direkter Dateizugriff**: Aus der TwinCAT-Runtime ist kein direkter Zugriff auf das Dateisystem des IPC realisierbar. Eine Ausnahme bildet hier die Verwendung des Simulink® Sink-Bausteins „To File“. Wie unter Verwendung des ToFile Blocks [▶ 50] beschrieben, kann in TwinCAT das TcExtendedFileWriter Module instanziiert werden, welches den Dateizugriff realisiert.
- **Direkter Hardware-Zugriff**: Ein direkter Zugriff auf Geräte/Schnittstellen setzt einen entsprechenden Treiber voraus, z.B. RS232, USB, Netzwerkkarte, … Aus dem Echtzeitkontext kann nicht auf die Gerätetreiber des Betriebssystems zurückgegriffen werden. Z. B. ist es daher nicht einfach möglich, mit der Instrument Control Toolbox™ eine RS232 Kommunikation für den nicht-Echtzeit-Betrieb herzustellen, und diese dann direkt in der TwinCAT-Runtime zu nutzen. Zur Anbindung von externen Geräten kann aber seitens TwinCAT auf eine Vielzahl von Kommunikationsmöglichkeiten zurückgegriffen werden, siehe TwinCAT 3 Connectivity TF6xxx.
- **Zugriff auf die Betriebssystem API**: Es ist nicht möglich aus der TwinCAT-Runtime die API des Betriebssystems direkt zu nutzen, ein Beispiel ist die Einbindung der *windows.h* in C/C++ Code. Diese wird bspw. durch den Simulink Coder® bei Verwendung der FFTW-Implementierung (aber nicht bei der Radix-2-Implementierung) des FFT-Blocks aus der DSP Systems Toolbox™ eingebunden.

3.7.9 Welche Dateien werden automatisch bei der Codegenerierung und dem Publish erstellt?

Es werden in zwei unterschiedlichen Ordnern Dateien angelegt, sobald Sie den build-Vorgang aus Simulink heraus starten. Welche Dateien genau angelegt werden, hängt dabei von der gewählten Konfiguration ab.

Publish-Verzeichnis: %TwinCATDir%\CostumConfig\Modules

In diesem Verzeichnis werden alle Dateien abgelegt, die Instanziierung des TcCOM in TwinCAT benötigt werden.

Datei	Verwendungszweck
<ModelName>.tmc	TwinCAT Modul Class Datei
<ModelName>_ModuleInfo.xml	Block Diagramm Informationen sowie Zusammenfassung der Versionen des Engineering Systems (Matlab Version, TC Version, …)
<ModelName>_PlcOpenPOUs.xml	Optionale Datei. Kann für den Aufruf des TcCOM aus der SPS eingebunden werden, siehe Aufruf des generierten Moduls aus einem SPS-Projekt [▶ 45].
<ModelName>.sys	In den Unterverzeichnissen TwinCAT RT (x64) und TwinCAT RT (x86). Echtzeittreiber des erstellen Moduls.
<ModelName>.pdb	In allen Unterverzeichnissen. Debug-Informationsdatei.
<ModelName>.dll	In den Unterverzeichnissen TwinCAT UM (x64) und TwinCAT UM (x86). Treiber für die User-Mode runtime.

Zur Verwendung der in diesem Verzeichnis beschriebenen TcCOM auf weiteren Engineering Systemen kann der gesamte Ordner auf das Engineering System in den entsprechenden Ordner kopiert werden.

Build-Verzeichnis

Das build-Verzeichnis ist in der Regel der aktuelle matlab path, der beim Start des build-Vorgangs aktiv ist. Im build-Verzeichnis werden zwei Unterverzeichnisse angelegt. Zum einen legt der Simulink Coder das Verzeichnis slprj an an, in welchem Simulink spezifische cache-files abgelegt werden, zum anderen legt das TE1400 ein Verzeichnis <ModelName>_tct an, in welchem alle wichtigen Ressourcen zusammengefasst werden.

Datei	Verwendungszweck
Unterordner html <ModelName>_codegen_rpt.html	Zusammenfassung der Codegenerierung und des Publish-Vorgangs in html Format.

Datei	Verwendungszweck
*.cpp und *.h	Quellcode der automatischen Codegenerierung
<ModelName>.vcxproj	Visual Studio Projekt der automatischen Codegenerierung. Kann im TwinCAT-Knoten C++ als *existing project* eingebunden und von dort ge-published werden.
<ModelName>_PublishLog.txt	Textdatei mit Publish log.
<ModelName>_ModuleInfo.xml	Block Diagramm Informationen sowie Zusammenfassung der Versionen des Engineering Systems (Matlab Version, TC Version, …)
<ModelName>_PlcOpenPOUs.xml	Optionale Datei. Kann für den Aufruf des TcCOM aus der SPS eingebunden werden, siehe Aufruf des generierten Moduls aus einem SPS-Projekt [▶ 45].

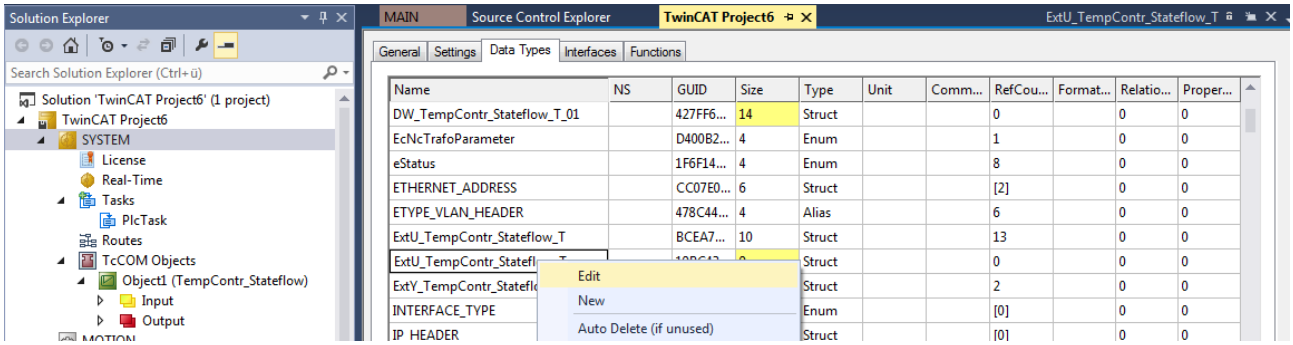
Die im build-Verzeichnis abgelegten Dateien eignen sich, wie die Dateien im Publish-Verzeichnis, zur Weitergabe an andere Engineering Systeme. Auf den entsprechenden Engineering Systemen muss dann der publish manuell über den C++ Bereich in TwinCAT erfolgen. Neben den Ressourcen zum publish liegen hier alle weitere relevanten Daten zum Nachvollziehen der Herkunft des generierten Quellcodes (ohne Matlab- oder Simulink-Quellcode).

3.7.10 Wie löse ich Datentyp-Konflikte im SPS-Projekt?

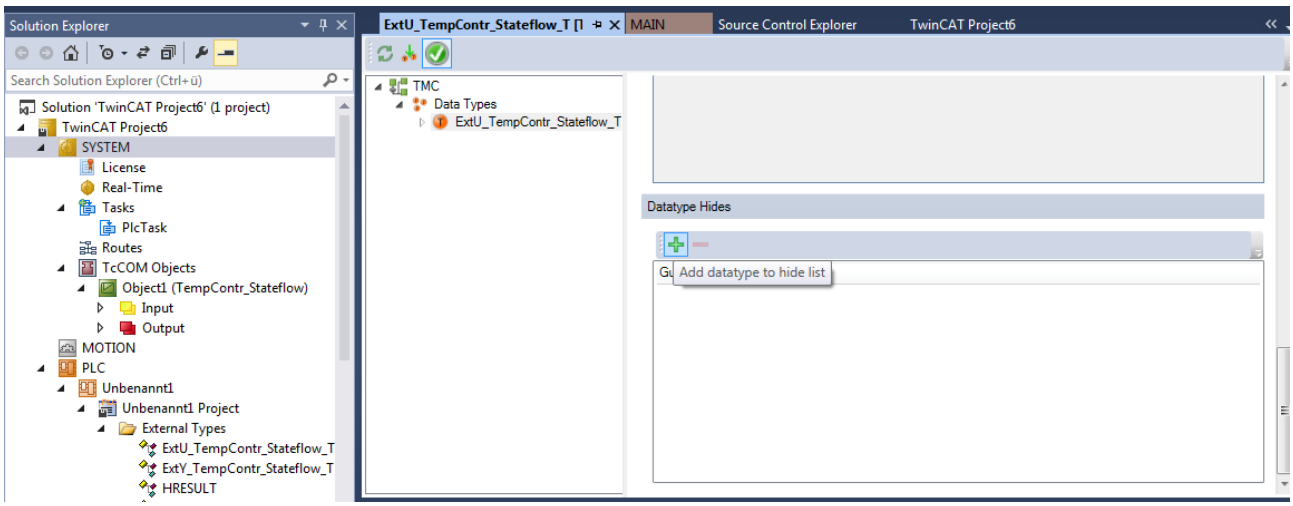
Werden Eingänge, Ausgänge, Parameter oder Zustandsvariablen eines Simulink-Modells verändert, ändern sich die zugehörigen Datentypen im daraus generierten TwinCAT-Modul. Die Datentypen haben nach der Aktualisierung den gleichen Namen aber eine andere GUID. Das Typsystem der TwinCAT-Entwicklungsumgebung (XAE) kann mehrere gleichnamige Datentypen mit unterschiedlicher GUID verwalten. Allerdings sind mehrere gleichnamige Datentypen in einem SPS-Projekt unzulässig.

Speziell nach der Aktualisierung einer Modul-Instanz per „Reload TMC“ können mehrere gleichnamige Datentypen im Typsystem existieren, von denen aber üblicherweise nur der zur aktuell instanziierten Modulklasse gehörende verwendet werden soll. Insbesondere bei der Nutzung der vom TE1400 generierten SPS-Funktions-Bausteine, muss in manchen Fällen manuell festgelegt werden, welcher der Datentypen im SPS-Projekt zur Verfügung stehen soll.

Hierzu kann der Datentyp-Editor über das Kontext-Menü des zu verwendenden Typs in der Tabelle **SYSTEM > Data types** gestartet werden:



Durch Hinzufügen von **Datatype Hides** lassen sich veraltete Datentypen gezielt von der Verwendung in SPS-Projekten ausschließen:



3.7.11 Warum sind in der TwinCAT Darstellung die Parameter des Transfer-Funktion Blocks nicht identisch mit der Darstellung in Simulink?

Der Simulink Coder® generiert echtzeitfähigen Code, wobei er alle Transfer-Funktion-Darstellungen in die Zustandsraumdarstellung transformiert. Entsprechend werden die Matrizen der Zustandsraumdarstellung (A, B, C, D) im von Simulink Coder® generierten Code verwendet, welche wiederum von TwinCAT 3 aus angezeigt und verändert werden können.

Die Transformation von der Transfer-Funktion-Darstellungen in die Zustandsraumdarstellung kann in MATLAB z.B. über die Funktion `[A,B,C,D] = tf2ss(NUM,DEN)` erfolgen.

3.7.12 Warum dauert meine Codegenerierung/mein Publish so lange?

Der Gesamtprozess der Generierung von instanziierbaren TcCOM Modulen durchläuft zwei Phasen. Zum einen die Codegenerierung und zum anderen den Publish-Prozess. Im *diagnostic viewer* von Simulink® wird Ihnen angezeigt:

```
################################################################################

### You can use the C++ project TctSmplTempCtrl.vcxproj to build the TcCOM module manually with
Microsoft VisualStudio.
### Necessary source and project files have been generated successfully.
### Duration of the code generation (HH:MM:SS): 00:00:15
### Publishing TcCOM module ##########################################################
### Configuration: "Debug" ### Platform(s): "TwinCAT RT (x86);TwinCAT RT (x64)"
### TwinCAT SDK: "C:\TwinCAT\3.1\SDK\"
### Platform Toolset: "Microsoft Visual C++ 2015 (V14.0)" (Automatically selected)
### Now you can instantiate the generated module in TwinCAT3 on the target platform(s) "TwinCAT RT
(x86);TwinCAT RT (x64)".
### Publish procedure completed successfully for TwinCAT RT (x86);TwinCAT RT (x64)
### Duration of code generation and build (HH:MM:SS): 00:00:24
### Generating code generation report ################################
```

Hinweise zur Dauer bei der Codegenerierung

Die Dauer der Codegenerierung hängt natürlich im hohen Maß von dem individuellen Modell ab und setzte sich aus der Codegenerierung des Simulink Coders sowie der Codegenerierung für das TcCOM Framework zusammen. Das TE1400 hat entsprechend nur Einfluss auf das TcCOM Framework.

Werden **große Parameterlisten**, z.B. Look-up-table als tunable markiert, wird in die zu erzeugende tmc-Datei der Look-up-table eingetragen, was unter Umständen zu erhöhten Zeiten bei der Codegenerierung führt.

Hinweise zur Dauer beim Publish-Prozess

Der Publish-Prozess setzt sich zusammen aus Kompilieren des C/C++ Codes mit dem MS Visual C++ Compiler, linken sowie kopieren der Moduldateien in das Publish Verzeichnis (<TwinCAT-Folder>\3.1\CustomConfig\Modules). Entsprechend ist für diesen Schritt die Leistungsfähigkeit des Compilers entscheidend, wobei diese von der Compiler Version oder auch von den Einstellungen (z. B. debug oder release) abhängt.

In Simulink® unter **Tc Build** ist es möglich für verschiedene **Zielsysteme** entsprechende binaries zu kompilieren. Diese werden in einem sukzessiven Prozess erstellt. Sollten Sie ein großes Modell bauen wollen, ist es ratsam hier nur für die Plattform(en) zu bauen, die Sie später auch nutzen.

3.8 Beispiele

Beispiel-Modelle zur Generierung von TcCom-Modulen:

Beispiel	Themen	Beschreibung
TemperatureController_minimal [▶ 65]	• Grundlagen	Ein sehr einfacher Temperaturregler, der die Grundlagen beschreibt.
TemperatureController [▶ 71]	• Parameterzugriff • Verwendung von Bus-Objekten • Verwendung von Testpunkten • Verwendung von referenzierten Modellen • Verwendung von External Mode • Generieren von TwinCAT Modulen aus Untersystemen	Ein sehr einfacher Temperaturregler mit PWM Ausgang. Bietet einen schnellen Überblick, wie der Modulgenerator zu verwenden ist. Nutzt darüber hinaus Simulink BusObjects (Strukturen) für eine Ausgabe und beinhaltet einen Testpunkt, der die Zugänglichkeit von internen Signalen vie ADS beeinflusst. Auch ExternalMode wird in dem Beispiel verwendet.
SFunStaticLib [▶ 80]	• SFunction • Statische Bibliothek	Generiert TwinCAT Module aus Simulink Modellen mit SFunctions, die von Drittanbietern ohne Quellcode zur Verfügung gestellt werden.
SFunWrappedStaticLib [▶ 86]	• SFunction • Statische Bibliothek	Generiert TwinCAT Module aus Simulink Modellen mit SFunctions, für die der Quellcode verfügbar ist, aber von statischen Bibliotheken abhängt.

Beispiele zu ModuleGeneration-Callbacks [▶ 25]:

Beispiel	Themen	Beschreibung
Moduldateien als ZIP verpacken [▶ 91]	• PostPublish callback • Archivierung generierter Modul-Dateien	Dieses einfache Beispiel zeigt die automatische Archivierung generierter Modul-Dateien.

3.8.1 TemperatureController_minimal

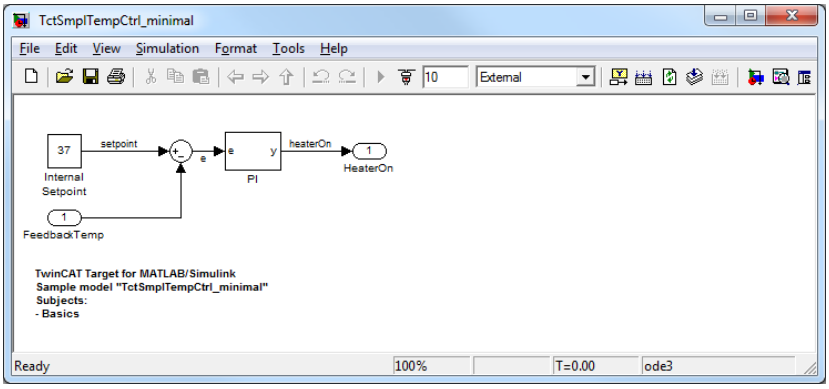
Beschreibung

Im folgenden Beispiel werden die Grundlagen der Generierung eines TwinCAT Moduls aus einem Simulink Modell veranschaulicht.

Überblick Projektverzeichnis

https://infosys.beckhoff.com/content/1031/te1400_tc3_target_Matlab/Resources/1539966475.zip enthält alle für die Reproduktion dieses Beispiels notwendigen Dateien:

TctSmplMinTempCtrl.mdl Simulink Modell eines sehr einfachen PI-Temperaturreglers.



TctSmplTempCtrlParameters.mat enthält alle notwendigen Modellparameter.

TctSmplMinCtrlSysPT2.mdl Simulink Modell einer einfachen PT2-Regelstrecke (wird in der folgenden Beschreibung nicht verwendet)

_PrecompiledTcComModules Dieses Unterverzeichnis enthält bereits fertig kompilierte, aus den beiliegenden Simulink-Modellen erzeugte TwinCAT-Module. Diese erlauben es, die Einbindung eines Moduls in TwinCAT ohne vorherige Modulgenerierung auszuprobieren. Sie können z. B. genutzt werden, wenn beispielsweise noch keine MATLAB-Lizenz vorhanden ist. Eine Kurzanleitung zur Installation der Module auf dem Entwicklungsrechner liegt ebenfalls bei.

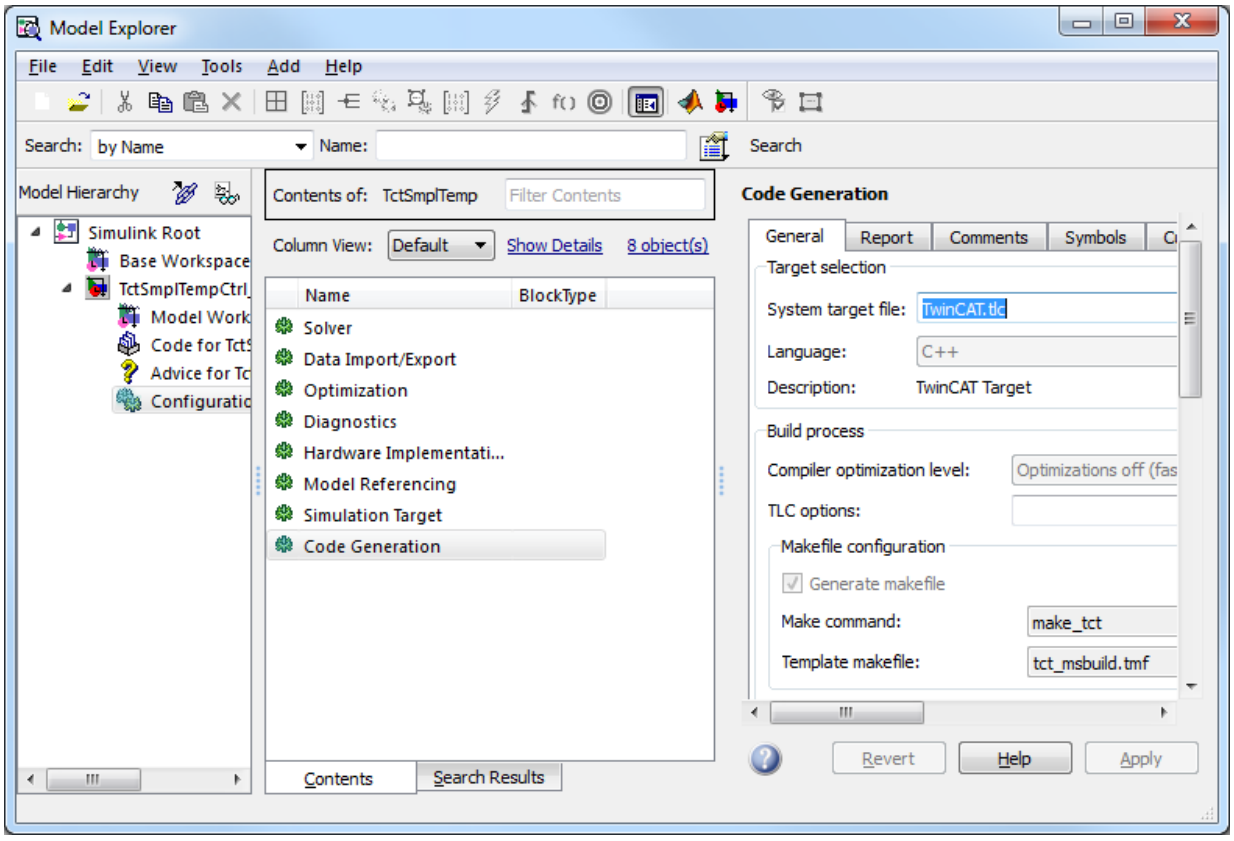
Info: Um das Modul auf einem x64-Zielsystem starten zu können, muss dieses in den Testmodus versetzt werden!

_PreviousSimulinkVersions Die oben beschriebenen MDL-Dateien sind im Datei-Format der aktuellen Simulink-Version gespeichert. Dieses Unterverzeichnis enthält die Modelle im Datei-Format älterer Simulink-Versionen.

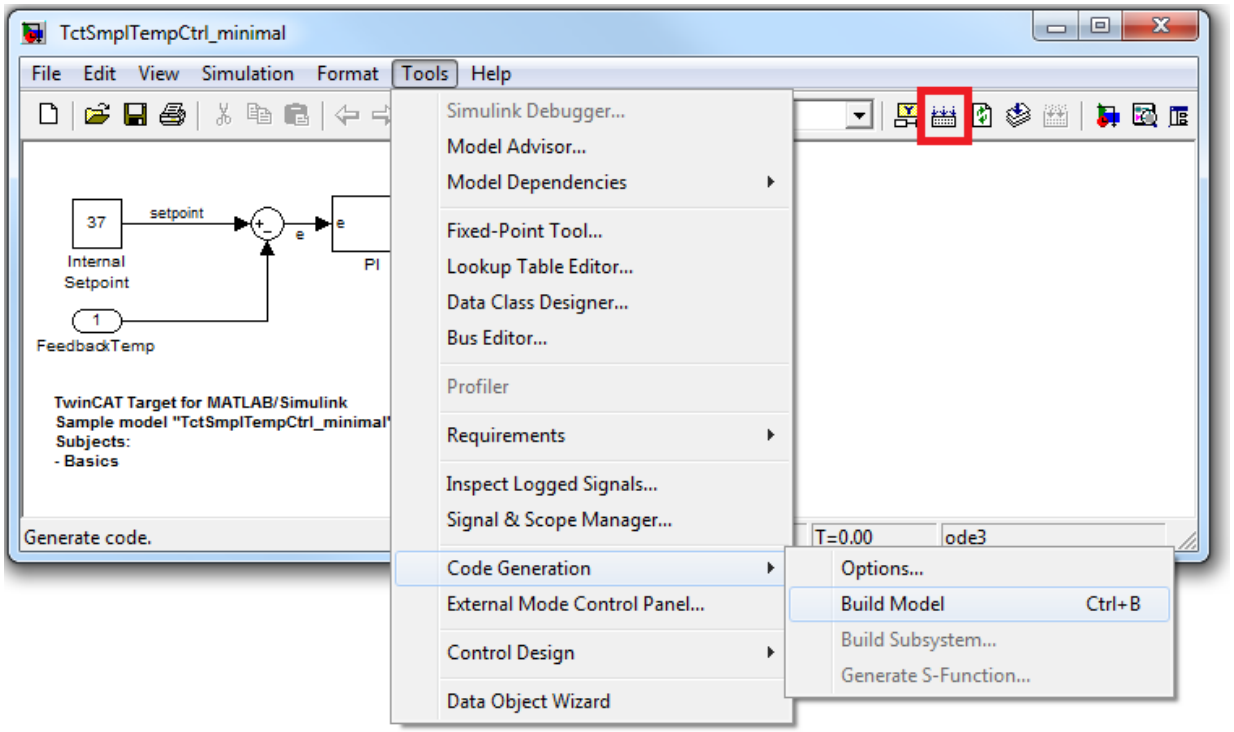
Ein TwinCAT Modul generieren

1. *TctSmplMinTempCtrl.mdl* in Simulink öffnen
2. **Model Explorer** starten

3. Unter **Configuration -> Code Generation** die **System target Datei** *TwinCAT.tlc* auswählen - entweder per Hand eintippen oder die Schaltfläche **Suchen** benutzen:



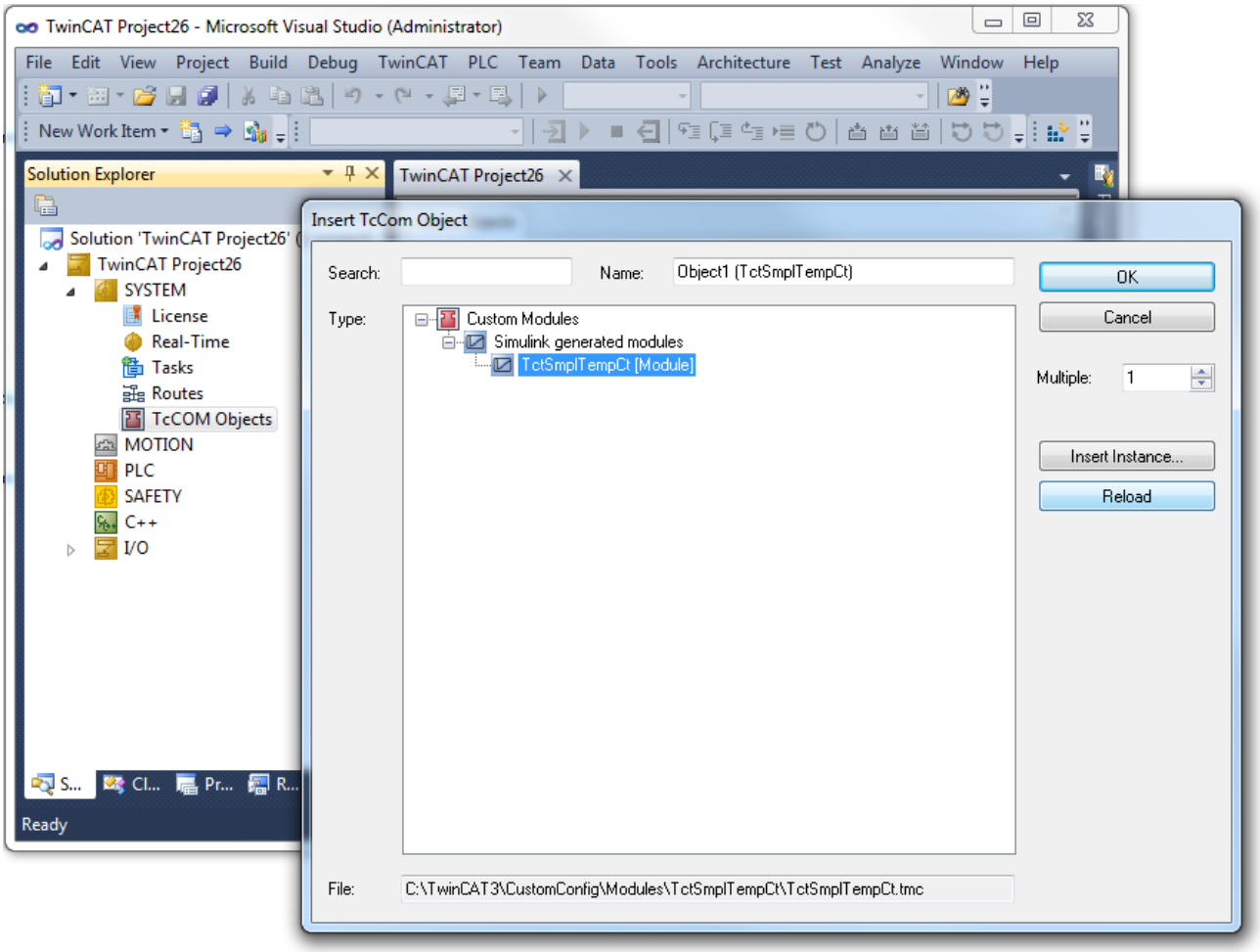
4. **Model Explorer** schließen
5. Codegenerierung über das Simulink-Menü **Tools->Code Generation->Build Model** oder über das Toolbarsymbol **Incremental build** starten



⇨ Der Fortschritt der Codegenerierung wird im MATLAB Befehlsfenster angezeigt.

Das generierte TwinCAT Modul verwenden

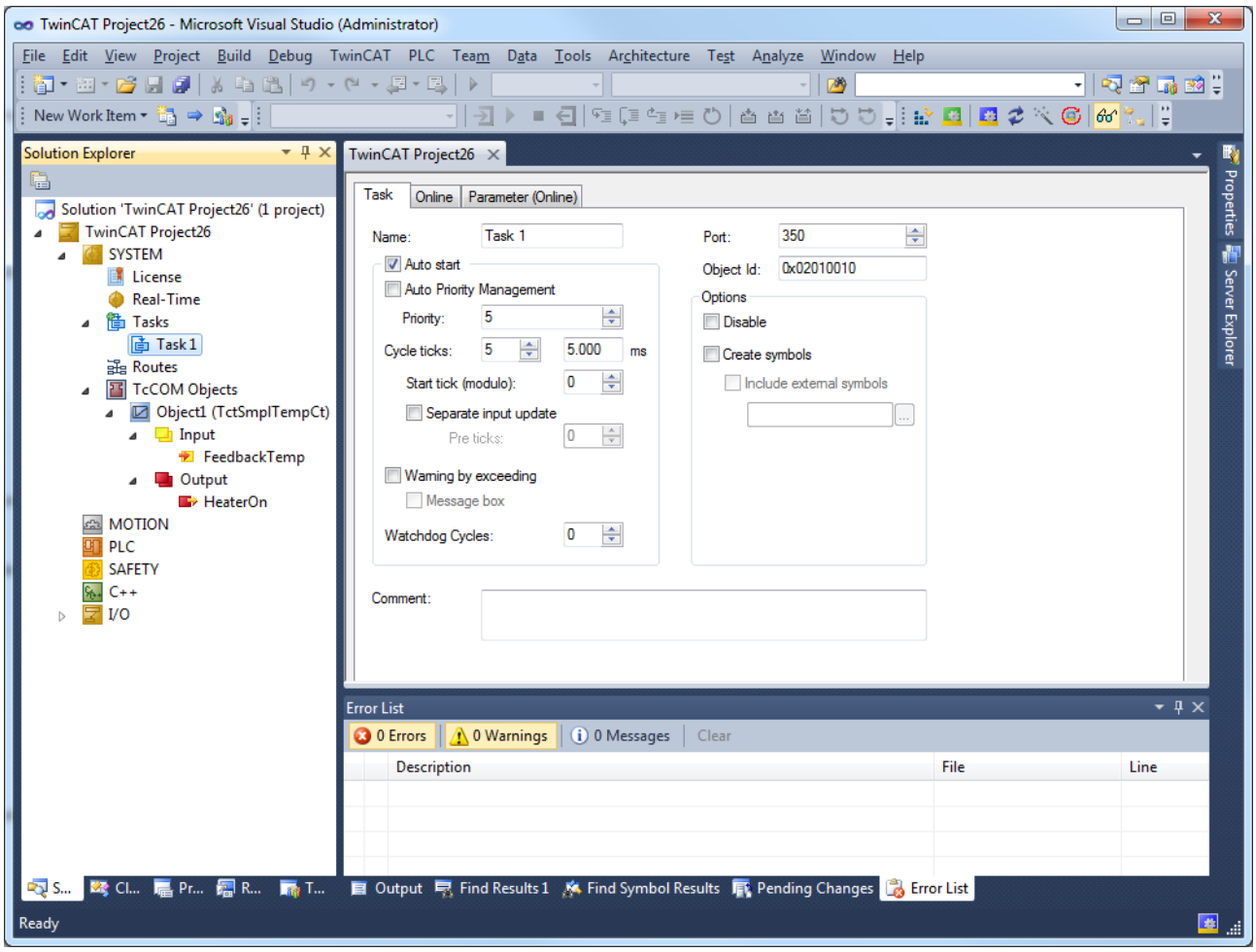
TwinCAT Entwicklungsumgebung öffnen und ein neues TwinCAT Projekt erstellen. Knoten **System** im **Solution Explorer** aufklappen. Im Kontextmenü des Knotens **TcCOM Objects** den Menüpunkt **Add new item** auswählen. Es wird folgender Dialog eingeblendet:



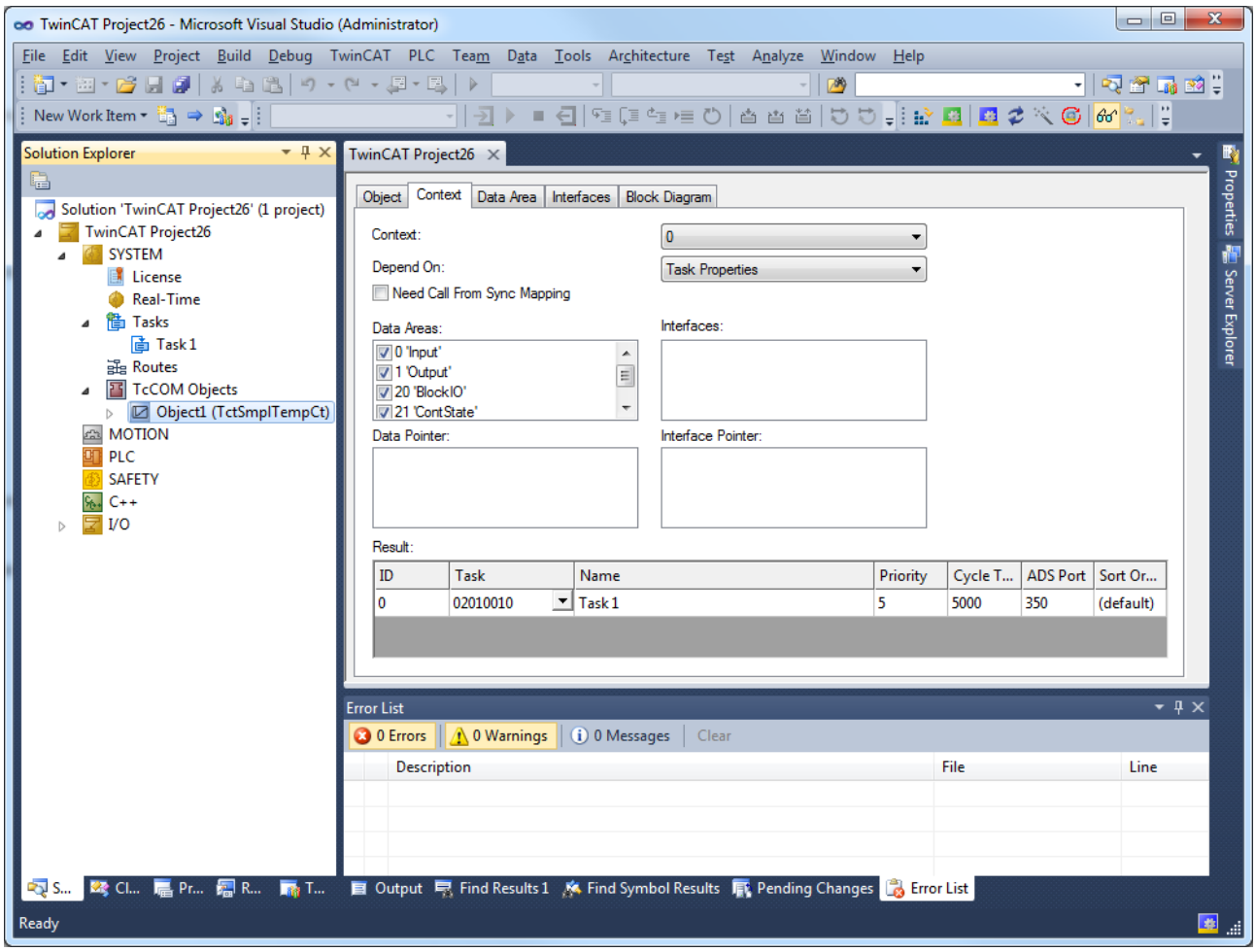
Wählen Sie das generierte Modul aus der Gruppe **Custom Modules** -> **Simulink generated modules**. Wenn XAE vor Abschluss der Codegenerierung gestartet wurde, muss zuerst die Schaltfläche **Reload** betätigt werden.

Fügen Sie einen neuen Task mit Hilfe des Kontextmenüs des Knotens **System** ->**Tasks** hinzu und konfigurieren die neue Task mit den Defaultparametern des generierten Moduls:

- Priorität: 5
- Cycle Time: 5 ms



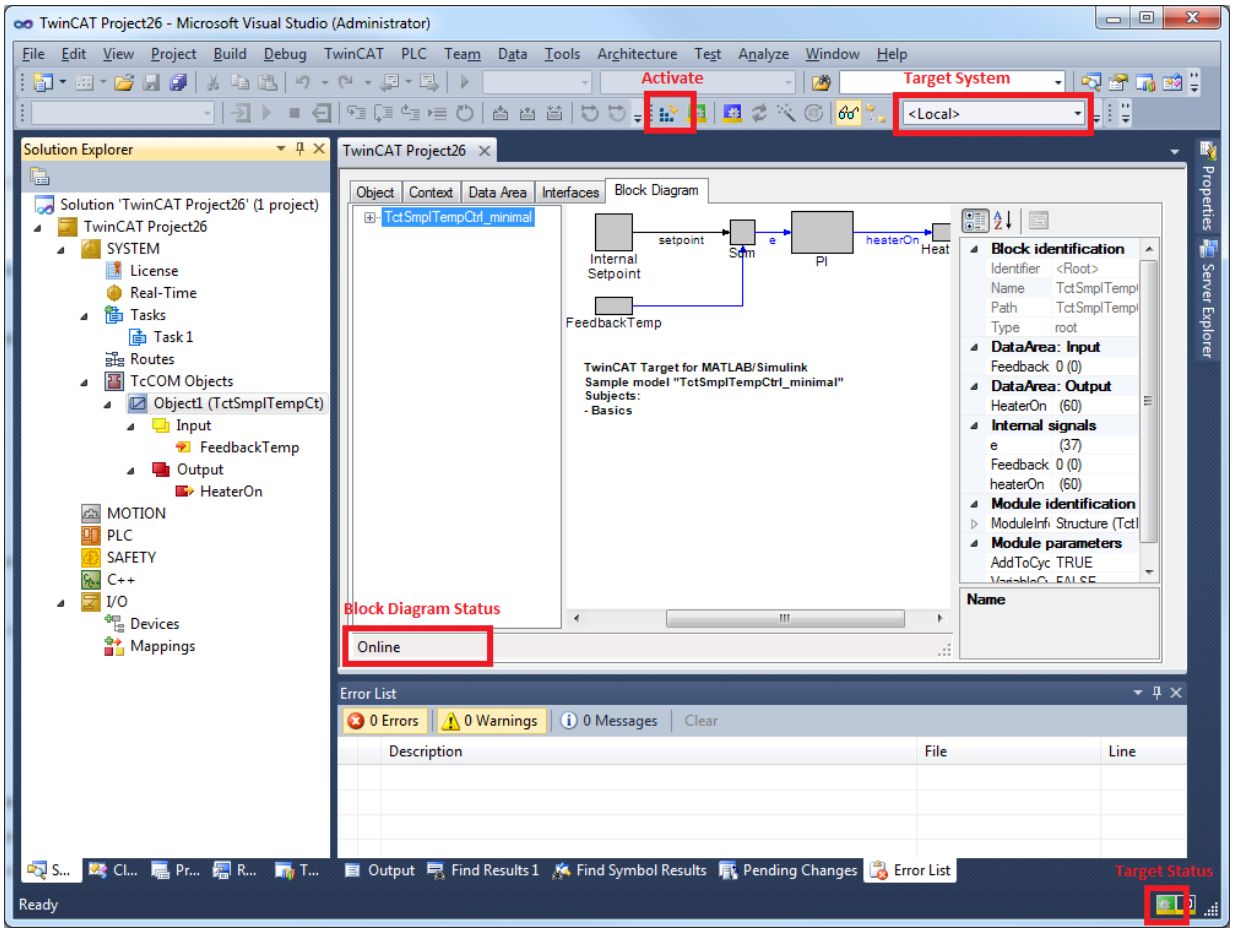
Anschließend müsste das Modul (mit seinen Standardeinstellungen) automatisch für das Anhängen an diese Task konfiguriert worden sein. Um das zu überprüfen, wählen Sie den Objektknoten **Object1 (TctSmplTempCt)** und öffnen den Karteireiter **Context.** Die Tabelle **Result** sollte die Objekt-ID und den Objektnamen der Task wie in nachfolgender Abbildung dargestellt enthalten:



Jetzt ist die Konfiguration abgeschlossen und kann auf dem Zielsystem aktiviert werden.

1. Wählen Sie das Zielsystem, die aktuelle Konfiguration muss aktiviert sein.
2. Falls es keine Lizenz gibt, aktivieren Sie eine kostenlose Probelizenz um die mit Simulink generierten Module (TC1320 oder TC1220) auf dem Zielsystem auszuführen.
3. Aktivieren Sie die Konfiguration auf Ihrem Zielsystem. Bestätigen Sie die Frage nach dem Überschreiben der aktuellen Konfiguration und starten das TwinCAT System.
4. Das Zustandssymbol auf dem Ziel sollte seine Farben auf grün (läuft) ändern.

5. Wenn der Karteireiter **Block Diagramm** ausgewählt wurde, wechselt der Blockdiagrammzustand in „Online“ und die Tabelle Eigenschaften zeigt einige online-Werte an.



3.8.2 Temperature Controller

Beschreibung

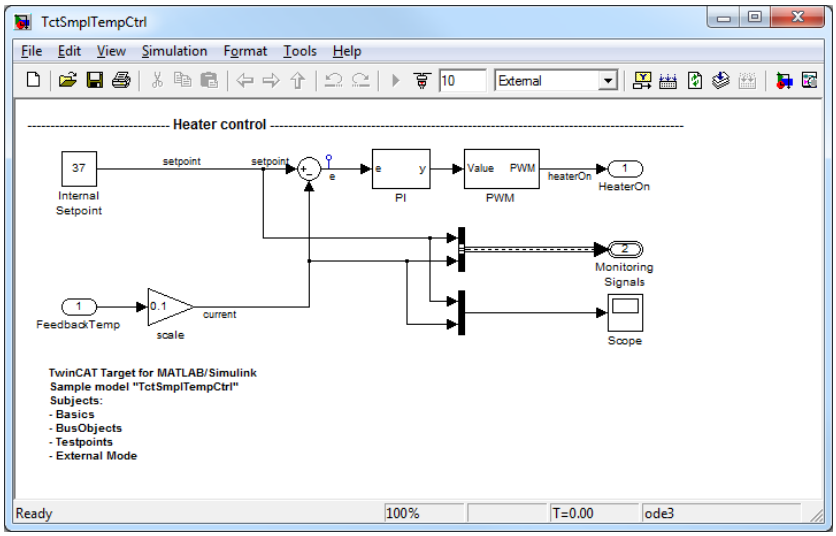
Im folgenden Beispiel werden die Grundlagen aus dem Beispiel „TemperatureController_minimal“ durch folgende Elemente erweitert:

- Parameterzugriff [▶ 72]
- Verwendung von Bus-Objekten [▶ 74]
- Verwendung von Testpunkten [▶ 75]
- Verwendung von referenzierten Modellen [▶ 77]
- Verwendung des Externen Modus [▶ 79]
- Generieren von TwinCAT Modulen aus Untersystemen [▶ 80]

Überblick Projektverzeichnis

https://infosys.beckhoff.com/content/1031/te1400_tc3_target_Matlab/Resources/1539964811.zip enthält alle Dateien für dieses Beispiel:

TctSmplTempCtrl.mdl erweiterter (aber immer noch sehr einfacher) Temperaturregler.



TctSmplCtrlSysPT2.mdl sehr einfaches PT2 Modell für die Regelstrecke.

TctSmplClosedLoopCtrl.mdl Modell eines geschlossenen Regelkreises, das durch Referenzieren der Reglermodelle und der Regelstrecke implementiert wurde.

TctSmplTempCtrlParameters.mat enthält alle notwendigen Modellparameter.

TctSmplTempCtrlBusObjects.mat enthält alle notwendigen Simulink BusObjects (Strukturdefinitionen).

_PrecompiledTcComModules Dieses Unterverzeichnis enthält bereits fertig kompilierte, aus den beiliegenden Simulink-Modellen erzeugte TwinCAT-Module. Diese erlauben es, die Einbindung eines Moduls in TwinCAT ohne vorherige Modulgenerierung auszuprobieren. Sie können z. B. genutzt werden, wenn beispielsweise noch keine MATLAB-Lizenz vorhanden ist. Eine Kurzanleitung zur Installation der Module auf dem Entwicklungsrechner liegt ebenfalls bei.

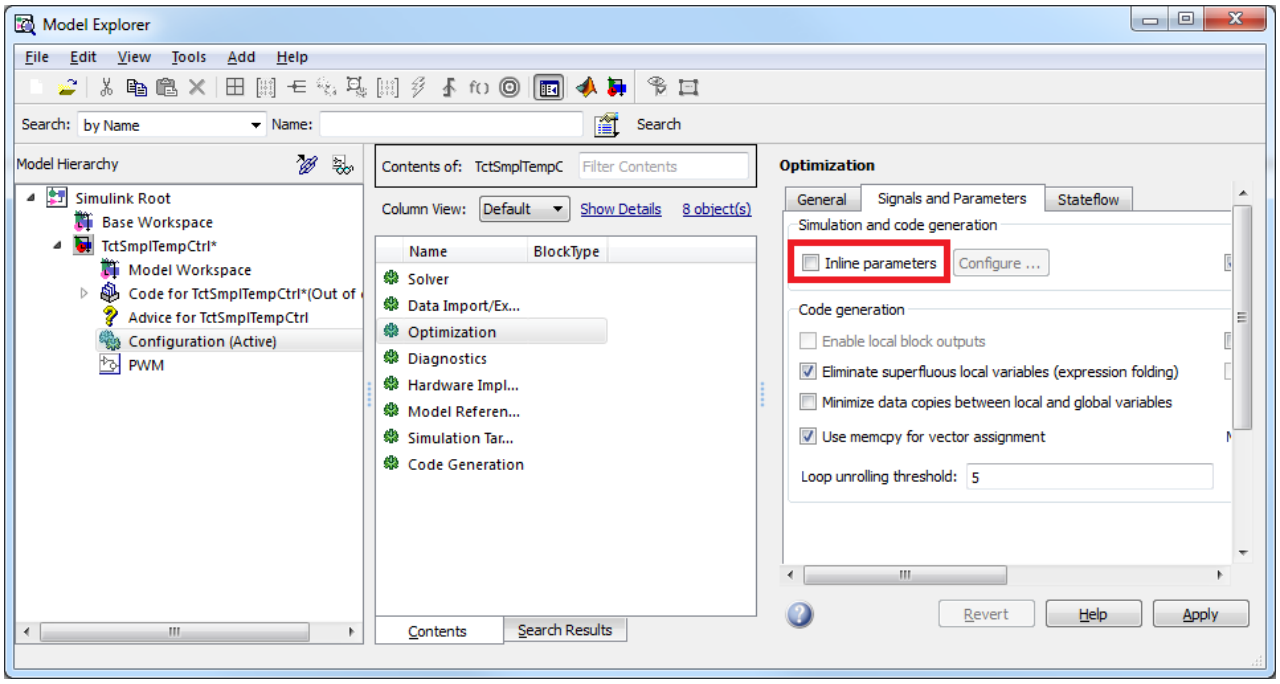
Info: Um das Modul auf einem x64-Zielsystem starten zu können, muss dieses in den Testmodus versetzt werden!

_PreviousSimulinkVersions Die oben beschriebenen MDL-Dateien sind im Datei-Format der aktuellen Simulink-Version gespeichert. Dieses Unterverzeichnis enthält die Modelle im Datei-Format älterer Simulink-Versionen.

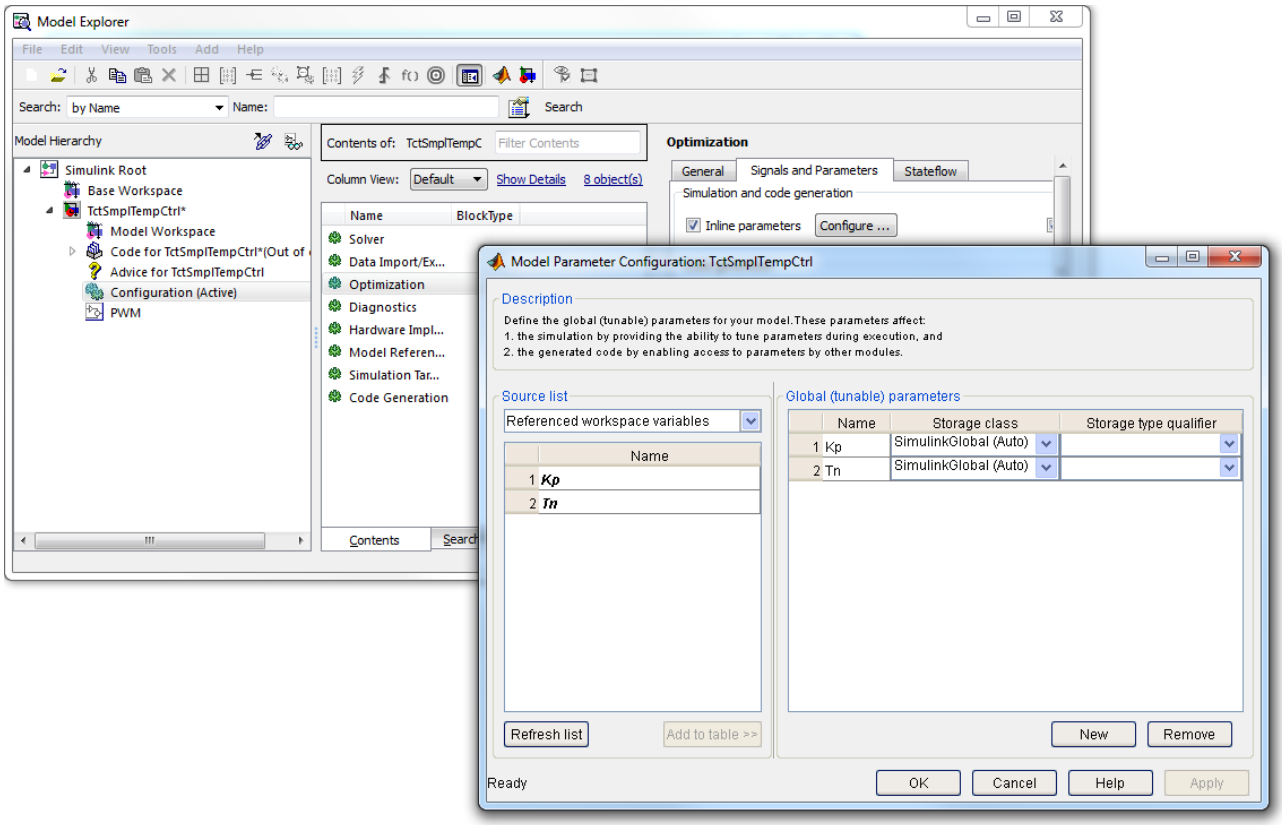
Parameterzugriff

TctSmplTempCtrl.mdl hat keine eingebetteten Parameterwerte (inline parameters), d.h. die Parameterwerte sind in der entsprechenden Modellparameterstruktur gespeichert. Darüber hinaus ist der Modulgenerator unter dem Karteireiter **TCT Advanced** der Codereinstellungen so konfiguriert, dass ADS-Zugriff auf die Parameter und die Erzeugung von ADS-Symbolen erlaubt sind. Damit ist aus TwinCAT Scope View oder

anderen ADS-Clienten ADS-Zugriff möglich. Der Karteireiter **Block diagram** in TwinCAT XAE ist ein ADS-Client und der Zugriff auf seine Parameter hängt an diesen Einstellungen.



Wenn die Option **Inline parameters** ohne weitere Konfigurationen aktiviert ist sind alle Parameterwerte in den generierten Modulcodes fest vorgegeben. Über die Schaltfläche **Configure...** neben **Inline parameters** kann ein Konfigurator geöffnet werden, in dem Sie Variablen des MATLAB Arbeitsbereichs auswählen können, die im generierten Modul einstellbar bleiben sollen:

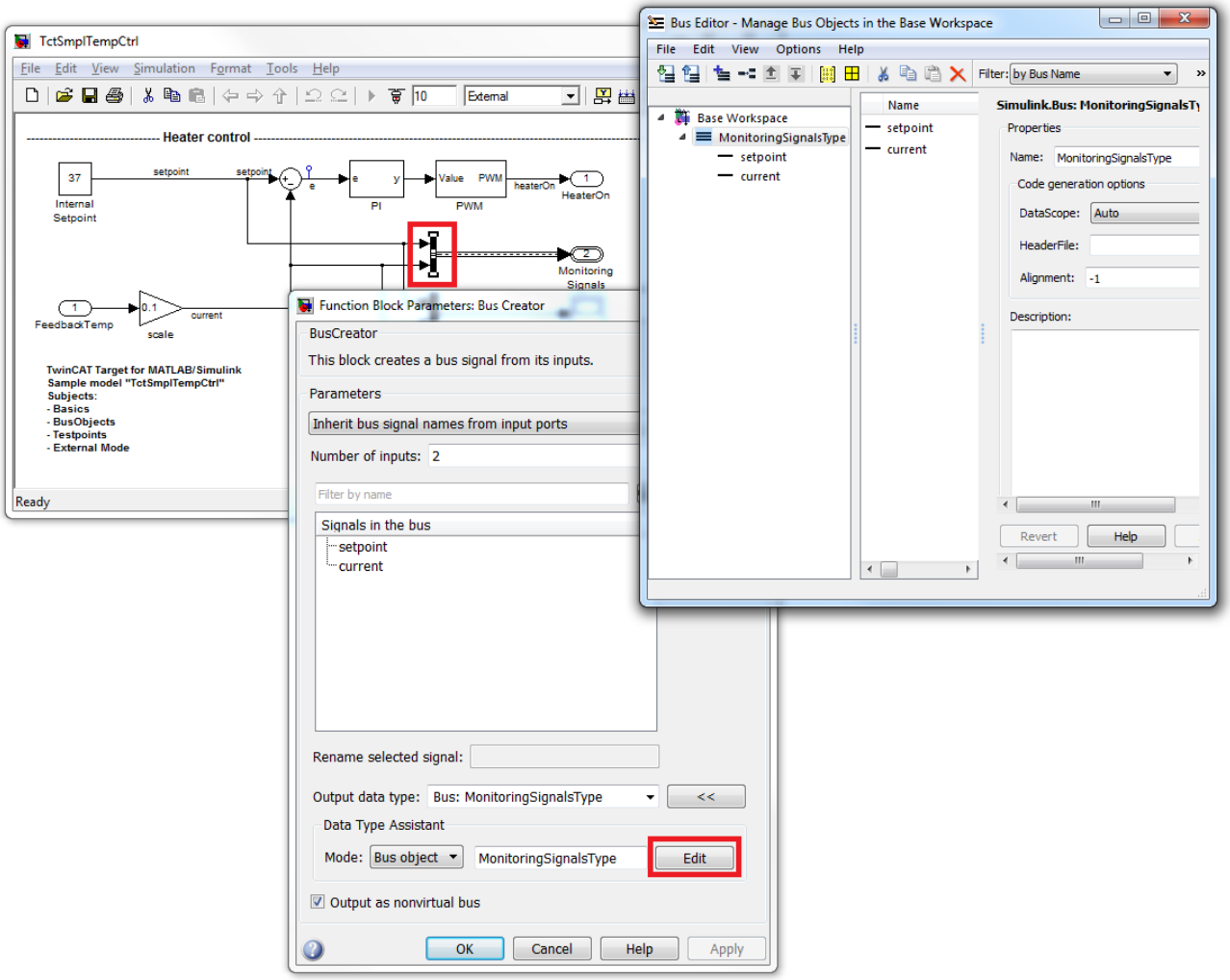


Im gezeigten Beispiel bleiben nur die Arbeitsbereichsvariablen *Kp* und *Tn* einstellbar, somit sind auch nur die von diesen Variablen abhängigen Simulink Blockparameter einstellbar. Die Parameterstruktur ist auf diese beiden Elemente reduziert.

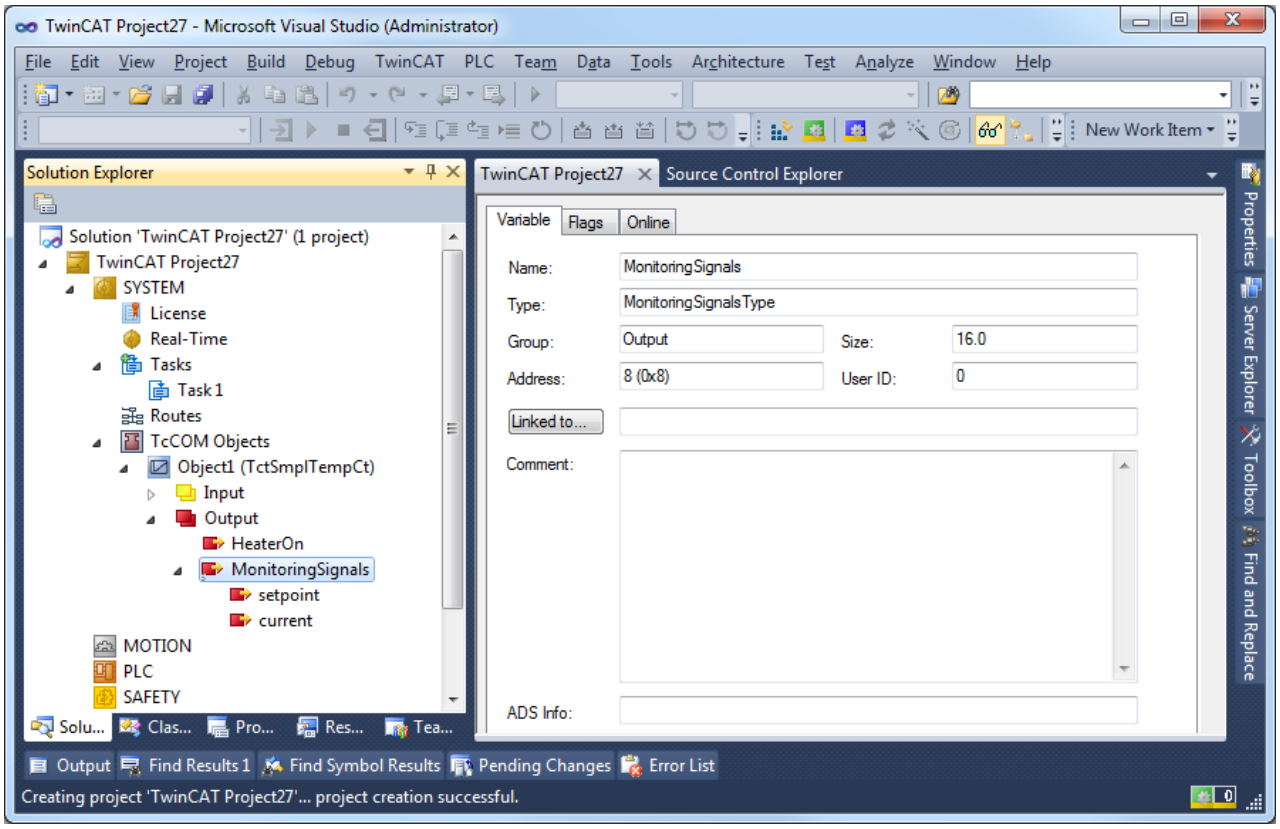
Weitere Informationen zu *parameter inlining*, siehe
Simulink Dokumentation.

Verwendung von Bus-Objekten

Mit Simulink BusObjects ist ein Zugriff auf in Simulink generierte TwinCAT Module über strukturierte Symbole möglich. Das vorliegende Beispiel enthält ein vordefiniertes BusObject namens *MonitoringSignalsType*. Es ist eine Ausgabestruktur, d.h. es weist die enthaltenen Signale einem SPS-Modul zu.
Die Konfiguration eines BusObjects wird durch Doppelklick auf den **BusCreator** Block gestartet. Um den Bus Editor zu starten, klicken Sie im Begrüßungsfenster auf die **Bearbeiten** Schaltfläche, wie in nachfolgender Abbildung gezeigt. Weitere Informationen zur Verwendung von BusObjects finden Sie in der Simulink Dokumentation.

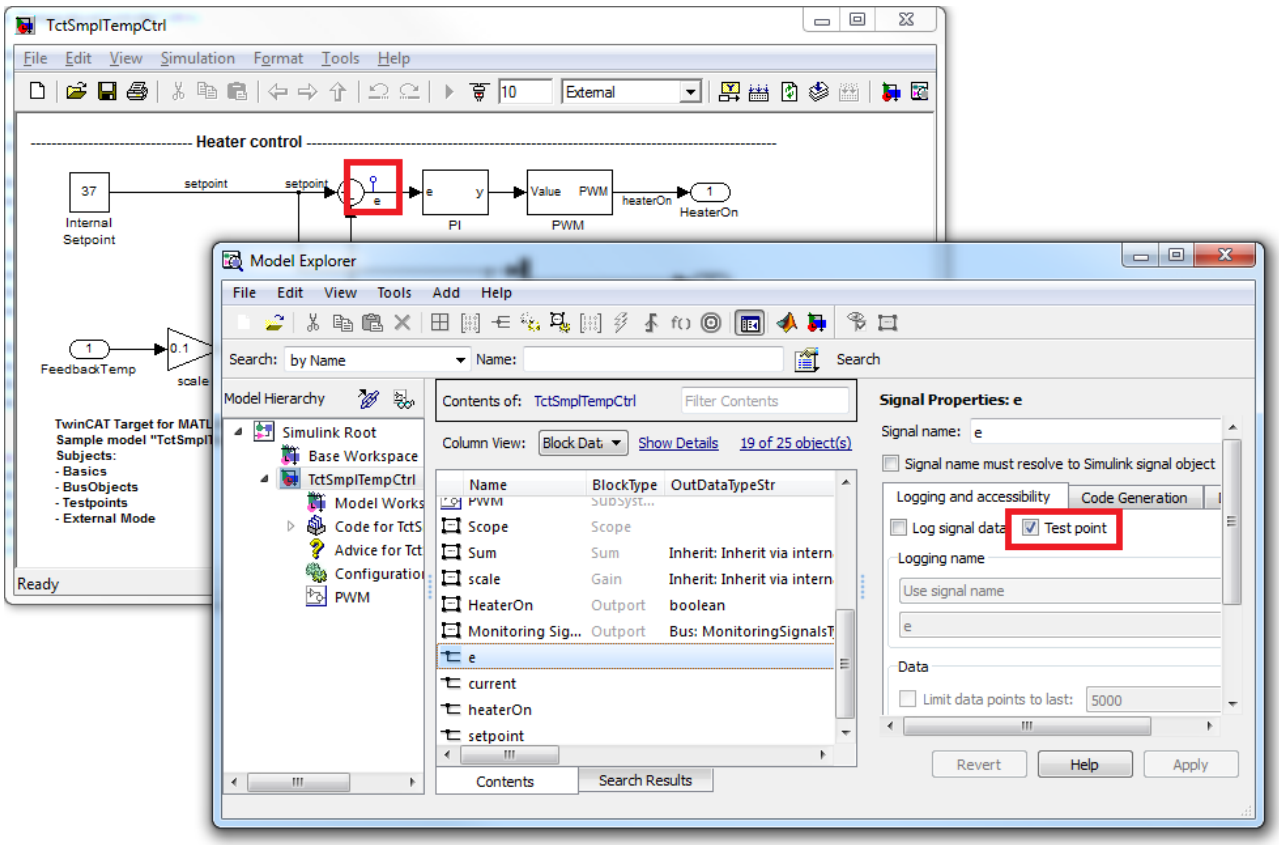


Bei der Instanziierung des generierten Moduls in einem TwinCAT Projekt wird das spezifizierte BusObject als globaler TwinCAT Datentyp in das TwinCAT Projekt importiert. Dieser Datentyp wird vom generierten Modul selber für die Darstellung der Ausgabestruktur verwendet, kann aber auch von anderen Modulen wie eine SPS verwendet werden, die z. B. mit dieser Ausgabestruktur verknüpft wird.

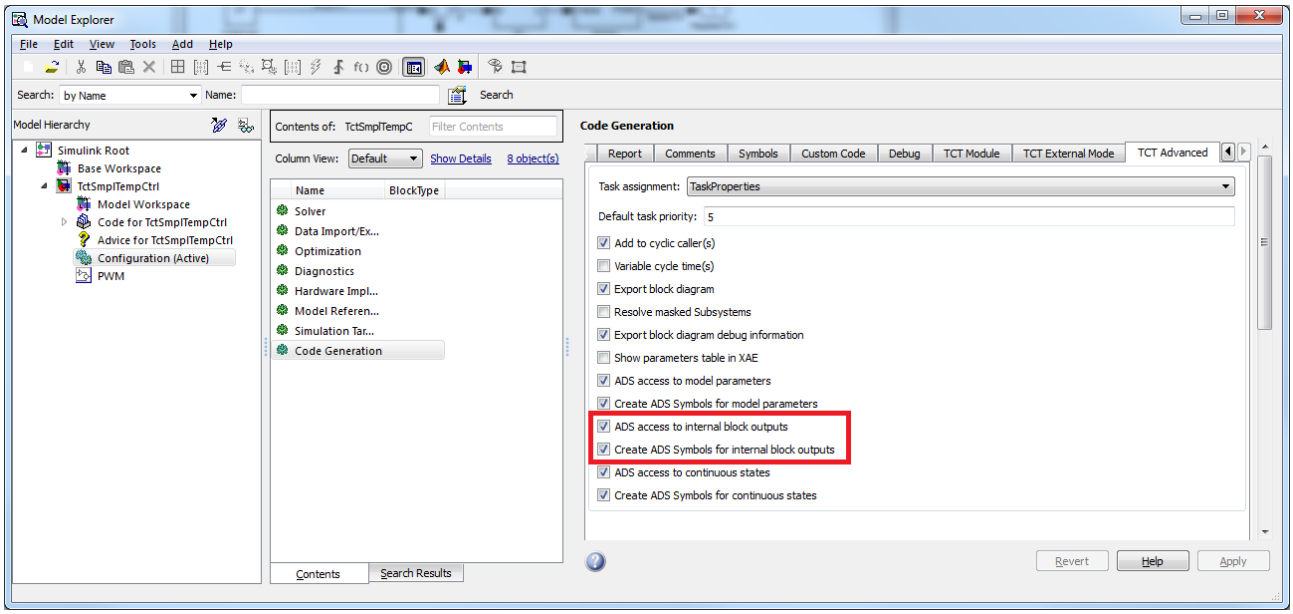


Verwendung von Testpunkten

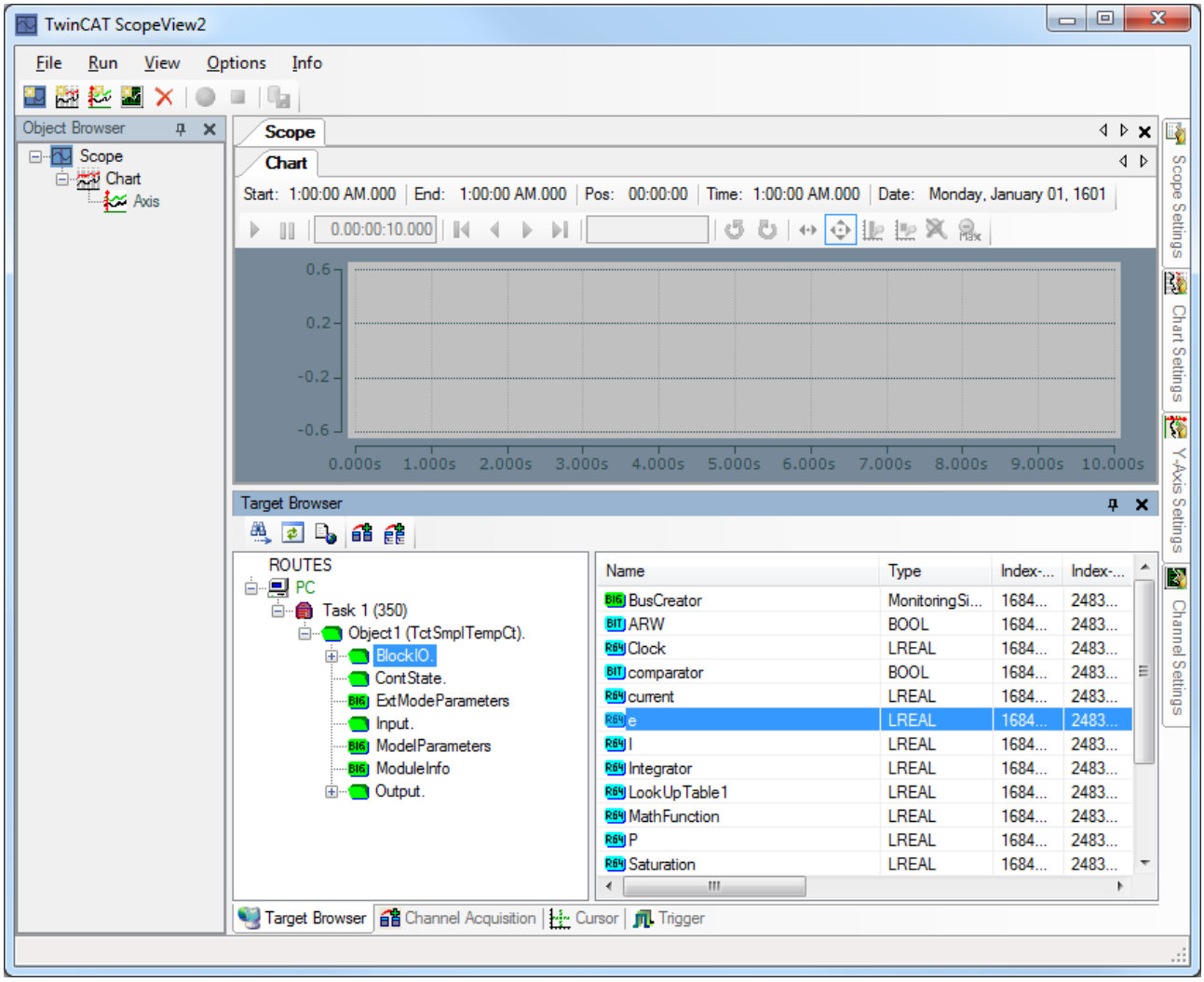
Sie können in Simulink auf Signalen Testpunkte festlegen, die z. B. von Simulink "Floating Scope" überwacht werden. Signale mit solchen Testpunkten werden bei Verwendung des TwinCAT Target Modulgenerators zwingend als Membervariable des generierten TwinCAT Moduls deklariert. Dadurch wird ADS-Zugriff auf das Signal möglich. Weitere Informationen zu Testpunkten siehe Simulink Dokumentation.
In diesem Beispiel wird mit dem **Model Explorer** ein Testpunkt für die Regeldifferenz *e* definiert:



Zur Freigabe von ADS-Zugriff, **internal block output** in den Codereinstellungen unter dem Karteireiter **TCT Advanced** ermöglichen:

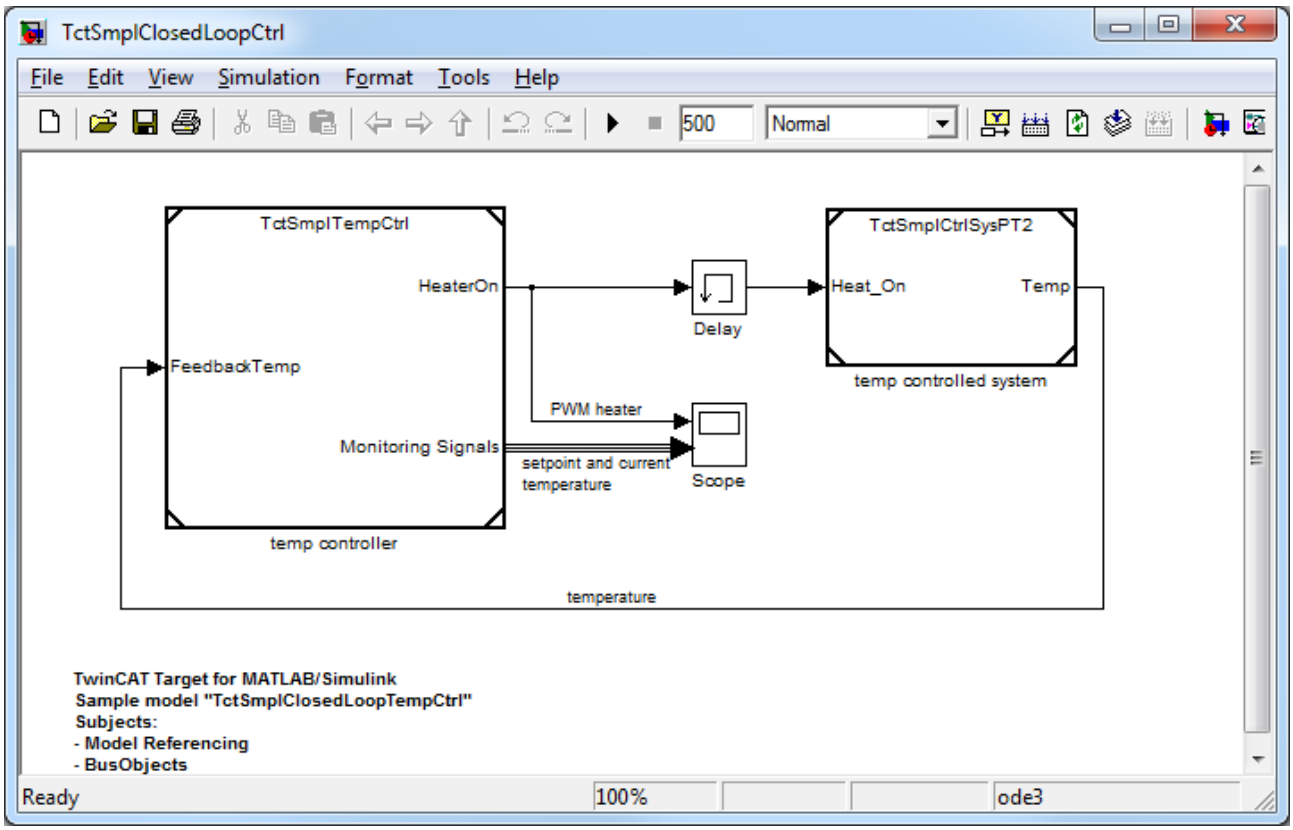


So können Sie z.B. mit TwinCAT Scope View beim Ausführen des generierten TwinCAT Moduls auf das Signal mit Testpunkten und einige weitere Blockausgangsvariable zugreifen.



Verwendung von referenzierten Modellen

Öffnen Sie das Modell *TctSmplClosedLoopCtrl.mdl,* das zwei Modellreferenzen enthält. Referenzierte Modelle sind der bekannte Temperaturregler und ein einfaches P-T2 Modell einer Temperaturregelstrecke.



Ein solche Modellreferenzierung hat einige Vorteile, generell oder auch in Kombination mit TwinCAT Target. Zwei Grundmöglichkeiten für strukturierte Modellierung und, insbesondere in diesem Beispiel, für Reglergestaltung sind:

Simulation zur Optimierung des Reglers:

Optimieren des Reglerdesigns durch Simulation des Regelkreises mit MATLAB/Simulink und anschließend nur den optimierten Regler in die Echtzeitumgebung von TwinCAT 3 übertragen. Durch Verwendung der Standard-Simulink Ein- und Ausgabeblöcke für die Definition der TwinCAT Modulprozessabbildern müssen vor dem Start der Modulgenerierung keine Änderungen am Reglermodell erfolgen.

Modellwiederverwendung und schnelleres Erstellen:

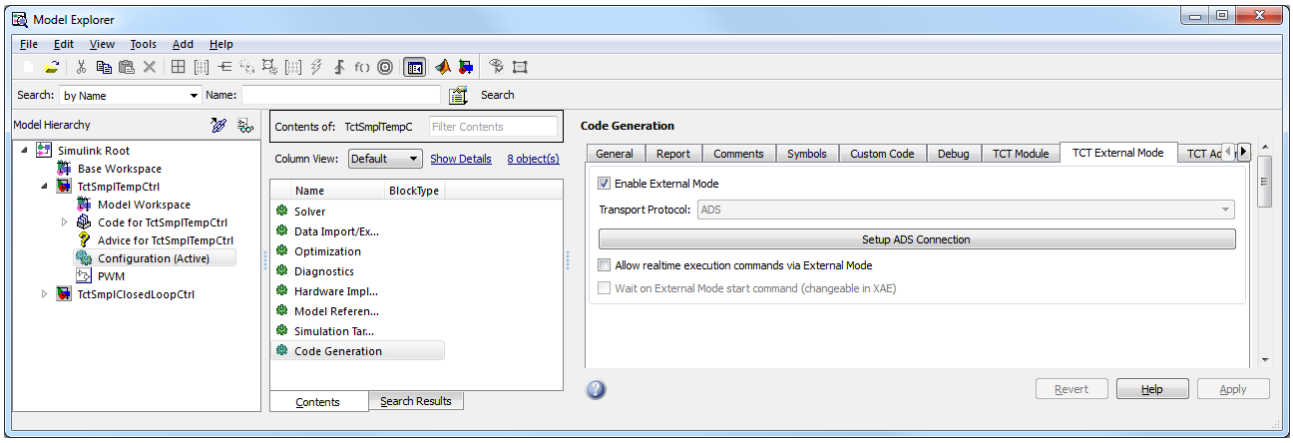
Ein Modell kann mehrere Male in einem oder mehreren übergeordneten Modellen referenziert werden. So können die Modelle in wiederverwendbare funktionale Einheiten aufgeteilt werden, wie in Text-Programmiersprachen, wo der Code in Funktionen oder Methoden gegliedert ist. So wird die Lesbarkeit komplexer Modelle verbessert.
Der generierte Code von referenzierten Modellen wird zu statischen Bibliotheken kompiliert, die nur aktualisiert werden, wenn das referenzierte Modell seit der letzten Codegenerierung verändert wurde. Das kann die Erstellung komplexer Modelle beschleunigen, wenn Sie selten veränderte Teile in referenzierten Modellen ablegen.
Sie können in diesem Beispiel die Modellgenerierung für ein Regelkreismodell starten und eine Echtzeit-Regelkreissimulation in der TwinCAT Laufzeit ausführen.

Hinweis zu Lizenzen:

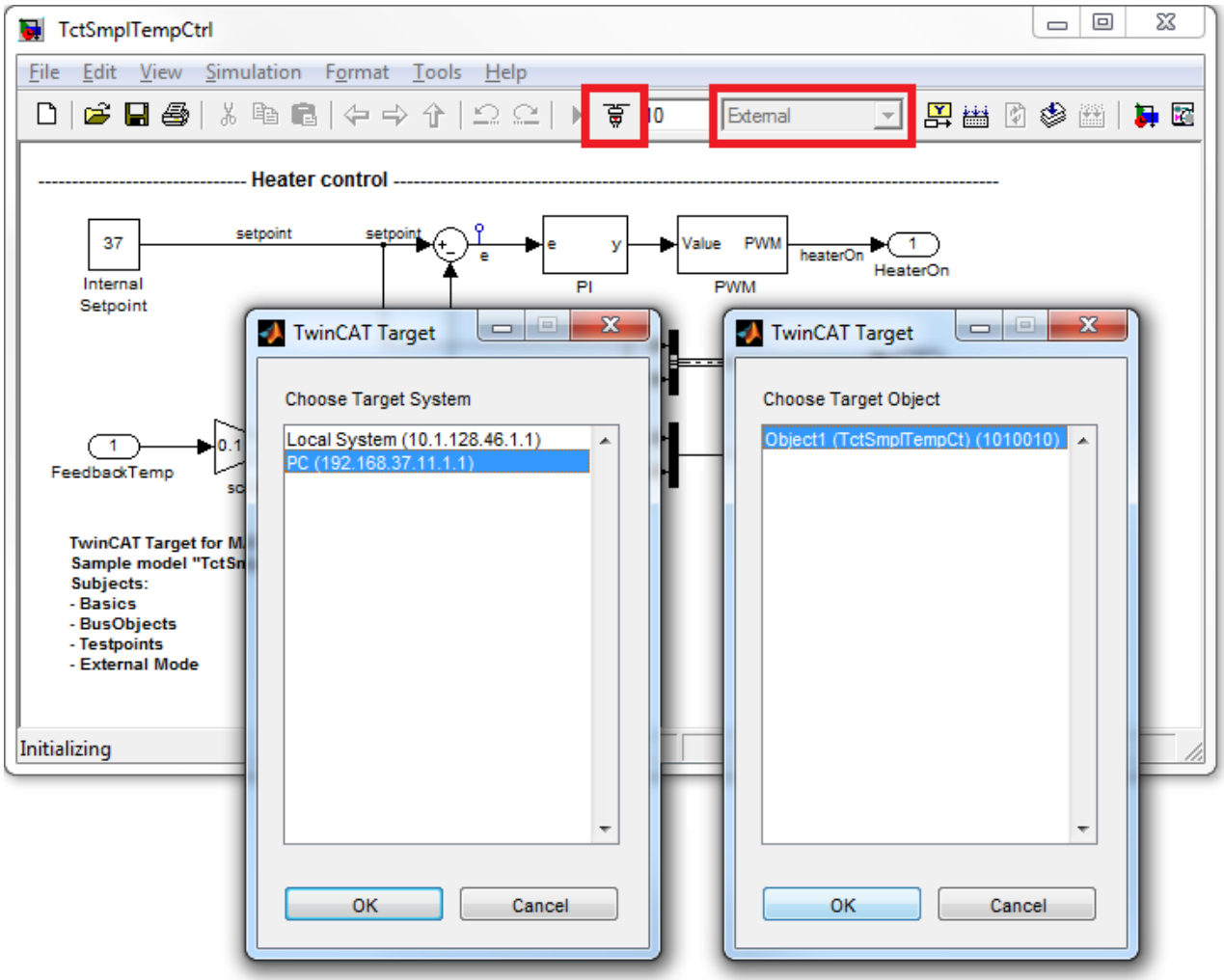
Das Regelkreismodell dieses Beispiels kann nur mit einer gültigen TwinCAT Target Lizenz (TE1400) in ein TwinCAT Modul übersetzt werden. Sonst sprengt dieses Modell die Grenzen für unlizenzierte Modelle.

Verwendung von External Mode

Das Temperaturreglermodell *TctSmplTempCtrl.mdl* wurde so vorkonfiguriert, dass ExternalMode Verbindungen erlaubt sind:



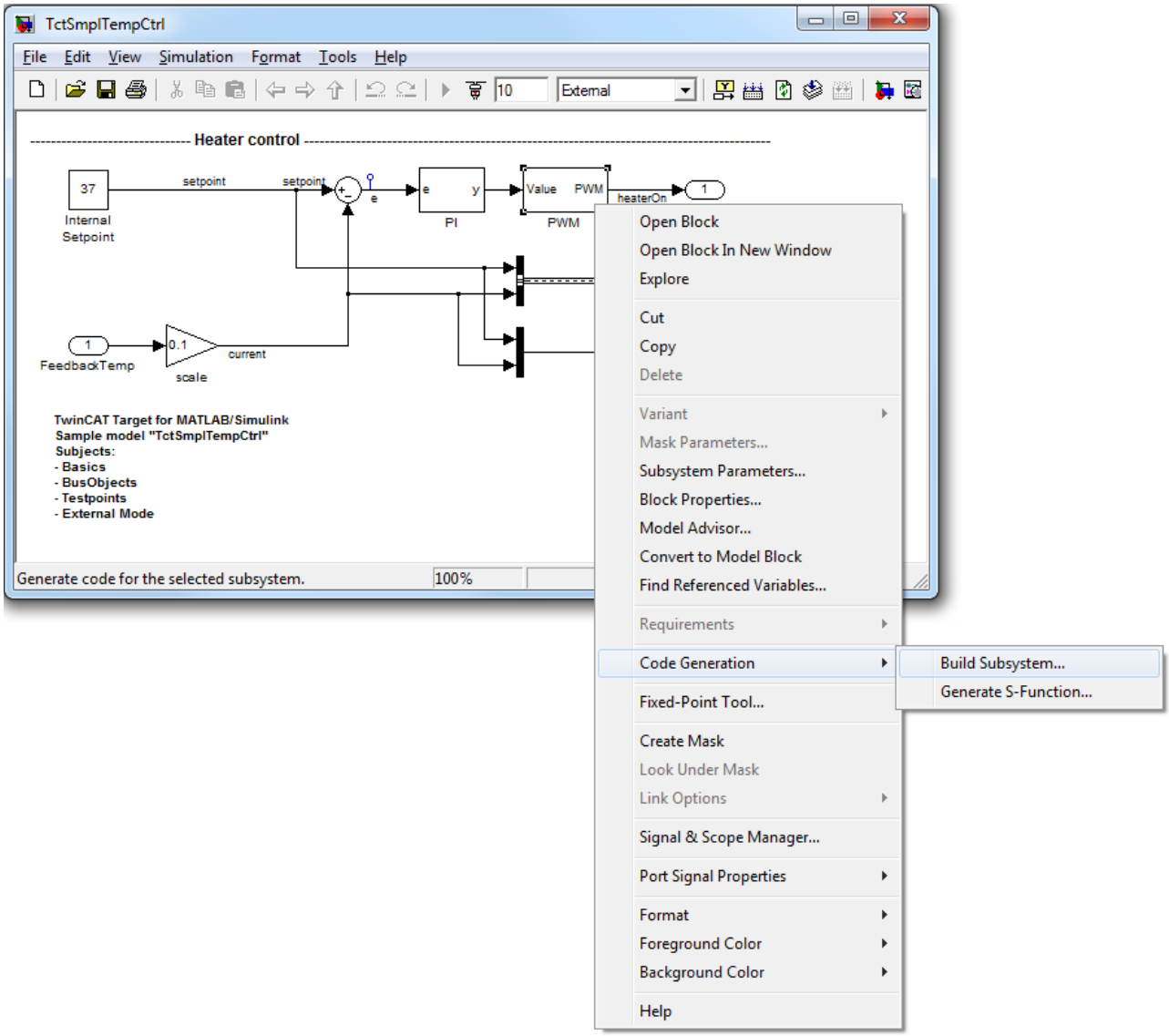
Wegen dieser Konfigurationen können Sie mit dem **Connect to Target** Symbol in der Simulink Toolbar via ExternalMode eine Verbindung mit dem generierten Temperaturregler herstellen. Das Modul muss vorher generiert und auf einem TwinCAT System gestartet worden sein und es muss eine ADS-Route zwischen Ihrem Entwicklungssystem und dem entsprechenden Target System konfiguriert worden sein. Es werden einige Dialoge eingeblendet, die Ihnen bei der Navigation zur gewünschten Modulinstanz helfen.



Jetzt können Sie den **Scope** Block in Simulink zur Überwachung der Echtzeitsignale des generierten und nun verbundenen TwinCAT Moduls verwenden. Sie können auch z.B. den Wert des **Internal Setpoint** Blocks ändern. Sobald die Parameteränderung bestätigt ist, wird sie direkt zum Zielmodul heruntergeladen. Das ist nur möglich für einstellbare Parameter, wenn die Modellparameter *not inlined* sind (siehe "Parameterzugriff [▶ 72]").

Generieren von TwinCAT Modulen aus Untersystemen

Erstellen eines TwinCAT Moduls in einem Simulink Untersystem, statt dem gesamten Modell, über das Untersystem Kontextmenü:



3.8.3 SFunStaticLib

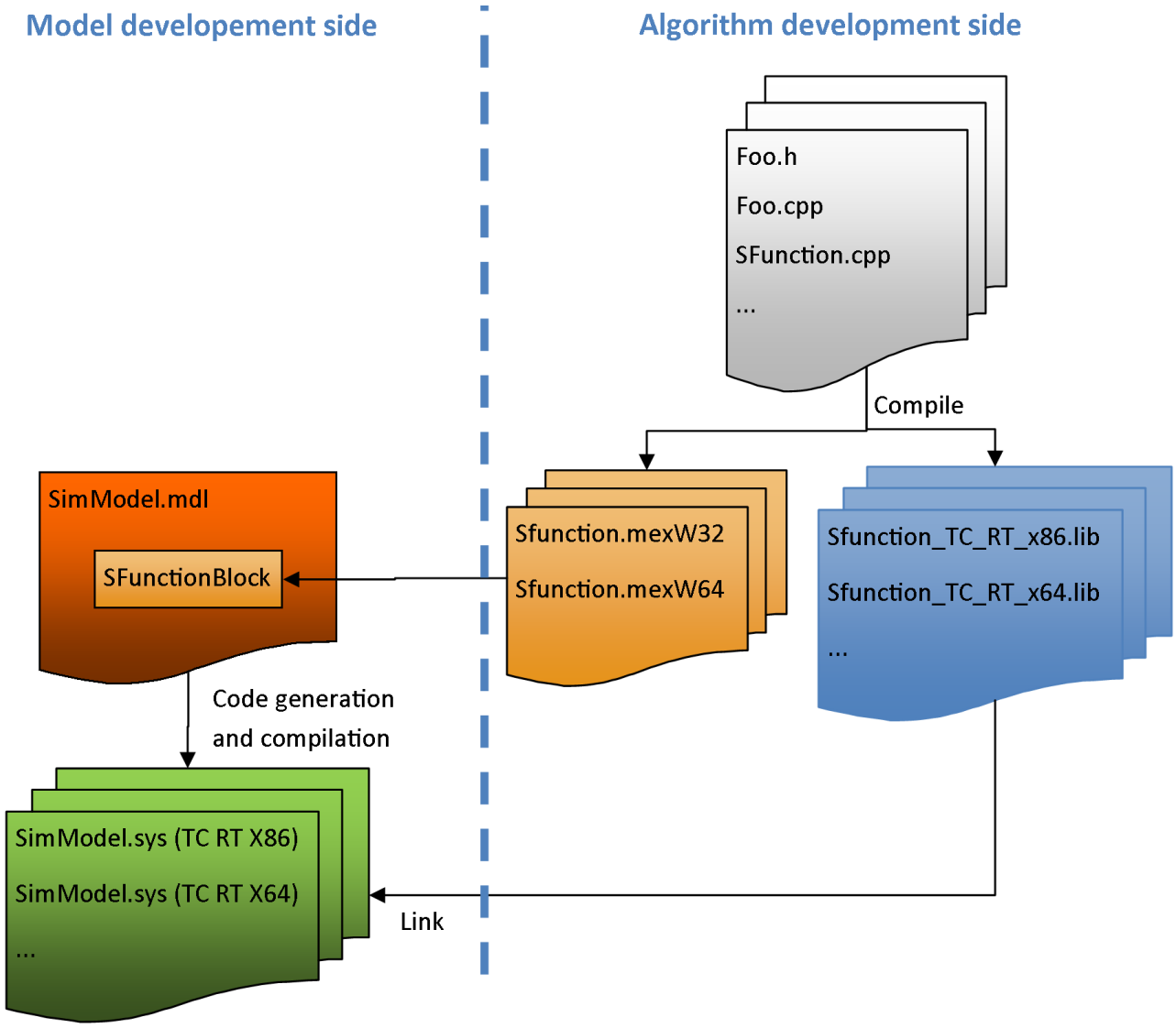
Anwendung

Die Einbindung eigener Codes in statische Bibliotheken kann nützlich sein, um

- die Erstellung von Modulen zu beschleunigen, sofern der Code nahezu unveränderliche Algorithmen enthält
- Kunden TwinCAT Target kompatible SFunction Algorithmen zur Verfügung zu stellen, ohne dass der Quellcode herausgegeben werden muss (es werden nur kompilierte Bibliotheken herausgegeben).

Beschreibung

Das folgende Beispiel zeigt, wie mit SFunctions TwinCAT Module aus Simulink Modellen generiert werden, für die kein Quellcode verfügbar ist. In diesem Fall kann die SFunction Funktionalität über statische Bibliotheken in das generierte TwinCAT Modul eingebunden werden. Das setzt aber voraus, dass für alle TwinCAT Plattformen, für die das Modul erstellt werden soll, geeignete Bibliotheken verfügbar sind.
Die folgende Grafik verdeutlicht den typischen Arbeitsfluss bei der Nutzung von Algorithmen Dritter in einem eigenen Simulink Modell:



In diesem Beispiel ist der Quellcode der „Algorithmus Entwicklungsseite“ verfügbar und kann für alle TwinCAT Plattformen kompiliert werden. Es zeigt wie

- SFunctions mit geeigneten TwinCAT Bibliotheken erzeugt werden
- solche Bibliotheken bereitgestellt werden (z.B. Kunden)
- solche Bibliotheken in eigenen Modellen verwendet werden

Übersicht Projektverzeichnis

https://infosys.beckhoff.com/content/1031/te1400_tc3_target_Matlab/Resources/1539910283.zip enthält alle notwendigen Dateien, um dieses Beispiel zu reproduzieren:

TctSmplSFunStaticLib.mdl	enthält das Modell, das die SFunction referenziert.
BuildLibsAndSFunction.m	enthält ein M-Skript, das die statische Bibliothek für alle derzeit verfügbaren TwinCAT Plattformen und die SFunktion erstellt.

OpenLibProject.m	enthält ein M-Skript, das die MATLAB Build Umgebung für Visual Studio, wie z.B. MATLAB Include- und Bibliotheks-Verzeichnisse definiert. Anschließend wird die statische Bibliothek mit den vordefinierten Umgebungsvariablen in Microsoft Visual Studio 2010 geöffnet.
Unterverzeichnis SFunLibProject	enthält das Sfunction Projekt.
Unterverzeichnis BuildScripts	enthält einige M-Skripts für "BuildLibsAndSFunction.m" und "OpenLibProject.m".
_PrecompiledTcComModules	Dieses Unterverzeichnis enthält bereits fertig kompilierte, aus den beiliegenden Simulink-Modellen erzeugte TwinCAT-Module. Diese erlauben es, die Einbindung eines Moduls in TwinCAT ohne vorherige Modulgenerierung auszuprobieren. Sie können z. B. genutzt werden, wenn beispielsweise noch keine MATLAB-Lizenz vorhanden ist. Eine Kurzanleitung zur Installation der Module auf dem Entwicklungsrechner liegt ebenfalls bei. **Info:** Um das Modul auf einem x64-Zielsystem starten zu können, muss dieses in den Testmodus versetzt werden!
_PreviousSimulinkVersions	Die oben beschriebenen MDL-Dateien sind im Datei-Format der aktuellen Simulink-Version gespeichert. Dieses Unterverzeichnis enthält die Modelle im Datei-Format älterer Simulink-Versionen.

SFunction und entsprechende statisch verknüpfte Bibliotheken erstellen

Erstellungsvoraussetzungen

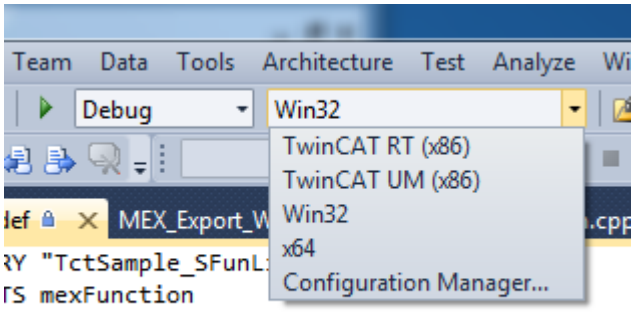
Für die Erstellung der Binärdateien sollte (nicht muss) TwinCAT 3 auf Ihrem Rechner installiert sein. Voraussetzungen:

Windows-Treiberkit	auf dem Rechner installiert, Umgebungsvariable *WinDDK* auf entsprechenden Pfad gesetzt.
TwinCAT SDK	auf dem Rechner installiert, Umgebungsvariable *TwinCatSdk* auf entsprechenden Pfad gesetzt.

Weitere Auskünfte dazu siehe Systemanforderungen in der TwinCAT 3 Dokumentation.

Binärdateien manuell erstellen

Sie können die Binärdateien mit Visual Studio manuell erstellen. Führen Sie dazu *OpenLibProject.m* aus. Dadurch werden die erforderlichen Umgebungsvariablen vorbereitet und das SFunction Projekt in Visual Studio geöffnet. Erzeugen Sie ein Projekt für alle Plattformen, die unterstützt werden sollen.



TwinCAT xx(xxx)	Erzeugt die plattformspezifische statische Bibliothek, die mit dem generierten TwinCAT Modul verknüpft wird.
Win32	Erzeugt die .MEXW32 SFunction, mit der die Simulation des Modells mit Simulink in 32-Bit-MATLAB laufen kann. Sie kann nur erzeugt werden, wenn Visual Studio aus 32-Bit-MATLAB gestartet wurde.

x64	Erzeugt die .MEXW64 SFunction, mit der die Simulation des Modells mit Simulink in 64-Bit-MATLAB laufen kann. Sie kann nur erzeugt werden, wenn Visual Studio aus 64-Bit-MATLAB gestartet wurde. Um die MEX Dateien dieses Beispiels für 32 und 64 Bit MATLAB zu erzeugen, muss Visual Studio aus beiden MATLAB Varianten heraus gestartet werden.

Erstellen der Binärdateien via Build Skript

Um das Erstellen der Dateien für einen schnellen Überblick zu beschleunigen, können Sie alternativ **BuildLibsAndSFunction.m** ausführen. Damit werden die Variablen der Build-Umgebung vorbereitet und MSBUILD wird mehrfach aufgerufen, um die .LIB und .MEXWxx Dateien für die TwinCAT Plattformen und die aktuelle MATLAB Plattform (32 oder 64 Bit) zu erzeugen. Um die MEX Dateien dieses Beispiels für 32 und 64 Bit MATLAB zu erzeugen, muss *BuildLibsAndSFunction.m* aus beiden MATLAB Varianten heraus gestartet werden.
Nach der Erstellung werden alle Build-Ausgabedateien in das Unterverzeichnis *LibProject\TctSample_SFunLib\BuildOutput* kopiert. Alle notwendigen Binärdateien werden darüber hinaus auch in das Verzeichnis *LibProject\TctSample_SFunLib\LibPackage* kopiert.

Binärdateien liefern

Das Verzeichnis *LibProject\TctSample_SFunLib\LibPackage* kann nun an Kunden übergeben werden, die die erzeugte - TwinCAT Target kompatible – SFunction verwenden möchten. Kopieren Sie den Inhalt dieses Verzeichnisses in das Verzeichnis *%TwinCat3Dir%Functions\TE1400-TargetForMatlabSimulink\Libraries* auf dem System des Nutzers. Wurde die Binärdatei mit *BuildLibsAndSFunction.m* erzeugt, wurde das im lokalen System bereits erledigt. Das Verzeichnis sollte folgenden Inhalt haben:

Unterverzeichnis TwinCAT xx (xxx)	enthält die statischen Bibliotheken für verschiedene TwinCAT Plattformen. Sie werden verwendet, um TwinCAT Module aus entsprechenden Simulink Modellen zu generieren.
Unterverzeichnisse Win32 / Win64	enthalten die MEX Dateien (und optional einige statische Bibliotheken) für die MATLAB Plattformen (32 und/oder 64 Bit). Das Unterverzeichnis Win32 oder Win64 wird dem MATLAB-Pfad beim Einrichten von TwinCAT Target über *SetupTwinCatTarget.m* hinzugefügt. So findet MATLAB die SFunction MEX Dateien und kann sie direkt von hier aus verwenden.

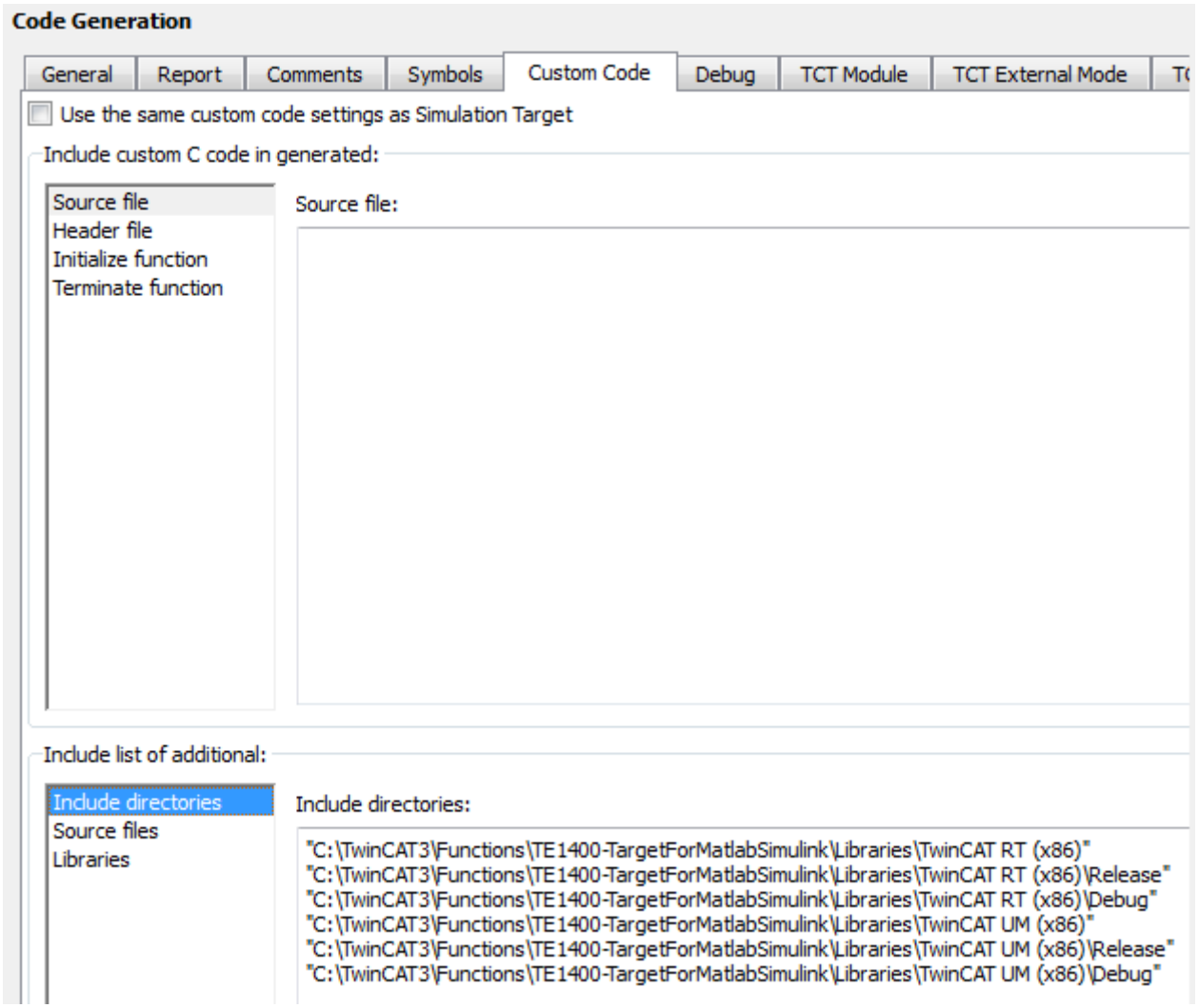
Simulation ausführen

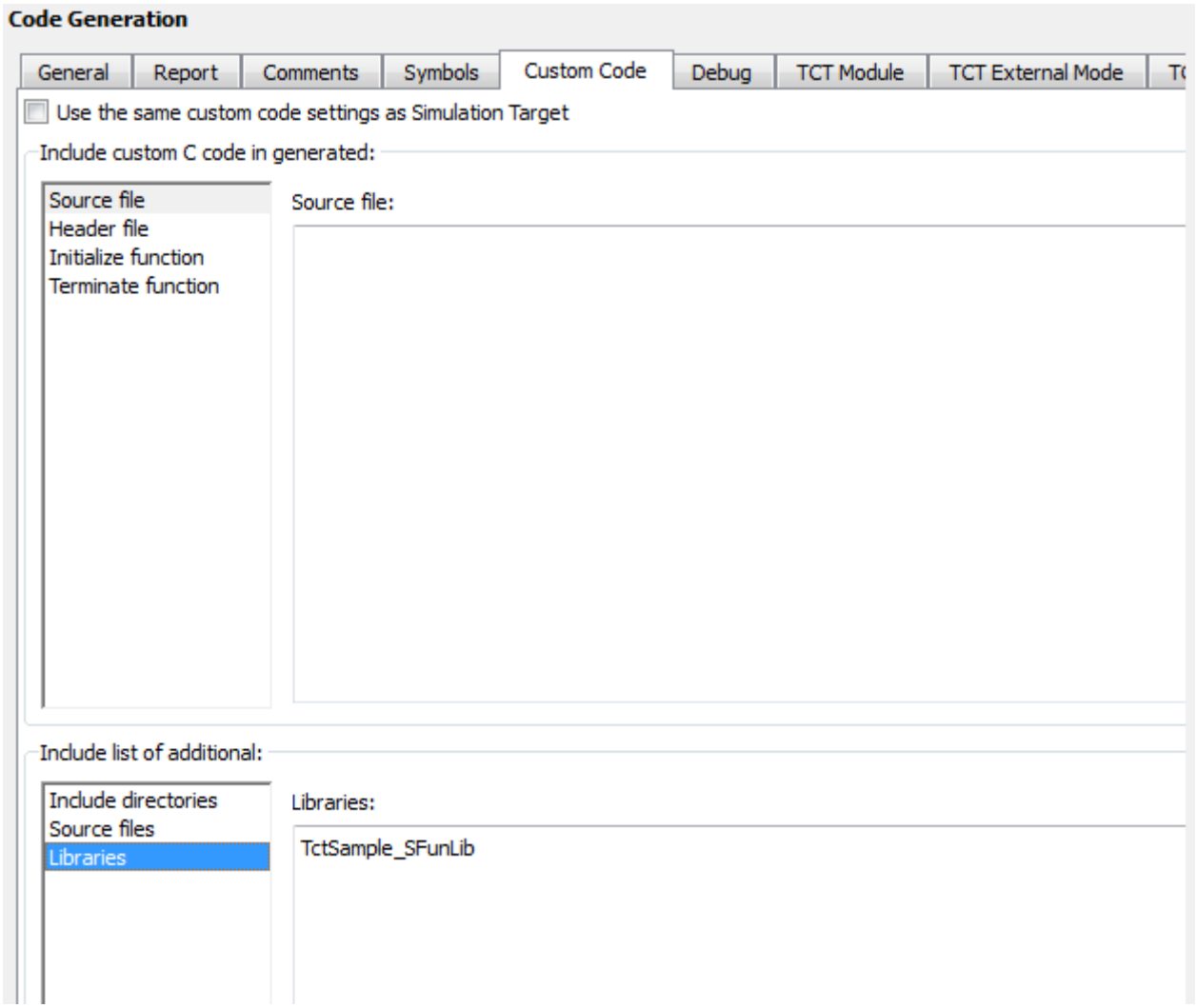
Zum Testen öffnen Sie "TctSmplSFunWrappedStaticLib.mdl" und starten die Simulation. Die Simulation muss ohne Fehlermeldung starten.

TwinCAT Modul generieren

Modell konfigurieren

Die allgemeinen Einstellungen zum Generieren eines TwinCAT Moduls müssen gesetzt sein, so muss ein Fixed-Step-Solver konfiguriert und unter dem Karteireiter „General“ in den Modellcodereinstellungen das SystemTargetFile "TwinCAT.tlc“ ausgewählt sein. Weitere Informationen zur allgemeinen Konfiguration des TwinCAT Modulgenerators siehe Quickstart.
Der Codegenerator muss auch wissen, welche statischen Bibliotheken mit dem generierten Code zu verknüpfen sind und wo diese zu finden sind. Tragen Sie diese Informationen in die entsprechenden Optionsfelder des Simulink-Coders ein, wie in den folgenden Abbildungen gezeigt.


Code Generation
General
Report
Comments
Symbols
Custom Code
Debug
TCT Module
TCT External Mode
Use the same custom code settings as Simulation Target
Include custom C code in generated:
Source file
Header file
Initialize function
Terminate function
Source file:
Include list of additional:
Include directories
Source files
Libraries
Include directories:
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Release"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Debug"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Release"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Debug"



Das "Include-Verzeichnisse" sollte bereits automatisch von TwinCAT Target erstellt worden sein. Die "Libraries" Einstellung muss die Namen der zu verknüpfenden statischen Bibliotheken enthalten.

Hintergrundinformation zu diesen Einstellungen:

In diesem Beispiel (und den meisten anderen Fällen) gibt es diese Bibliotheken in den angegebenen Verzeichnissen mehrfach. Welche Version mit dem Modul verknüpft wird, entscheidet MSBUILD bei der Verknüpfung der generierten TwinCAT Modul-Binärdateien.

Dieses Beispiel als Vorlage verwenden

Nachfolgend ist der einfachste Weg zu einer eigenen TwinCAT Target kompatible SFunction kurz beschrieben:

1. Beispielverzeichnis kopieren
2. MDL Datei durch eigene ersetzen
3. VCXPROJ Datei in den Namen Ihrer SFunction umbenennen
4. Eigene Quelldateien in das Verzeichnis der VCXPROJ Datei kopieren
5. Die Skripts *BuildLibsAndSFunction.m* und *OpenLibProject.m* an den neuen Projektnamen anpassen
6. VCXPROJ Datei mit *OpenLibProject.m* öffnen
7. Bestehende CPP-Dateien aus dem Projekt entfernen
8. Eigene CPP Dateien zum Projekt hinzufügen
9. Inhalt der DEF Datei an den neuen Projektnamen anpassen
10. Gegebenenfalls Include-und Dependency-Verzeichnisse sowie Bibliotheken zu Compiler- und Linkereinstellungen hinzufügen

11. Projekt erzeugen (für verschiedene Plattformen und/oder Konfigurationen)
12. VCXPROJ Datei schließen
13. BuildLibsAndSFunction.m starten

3.8.4 SFunWrappedStaticLib

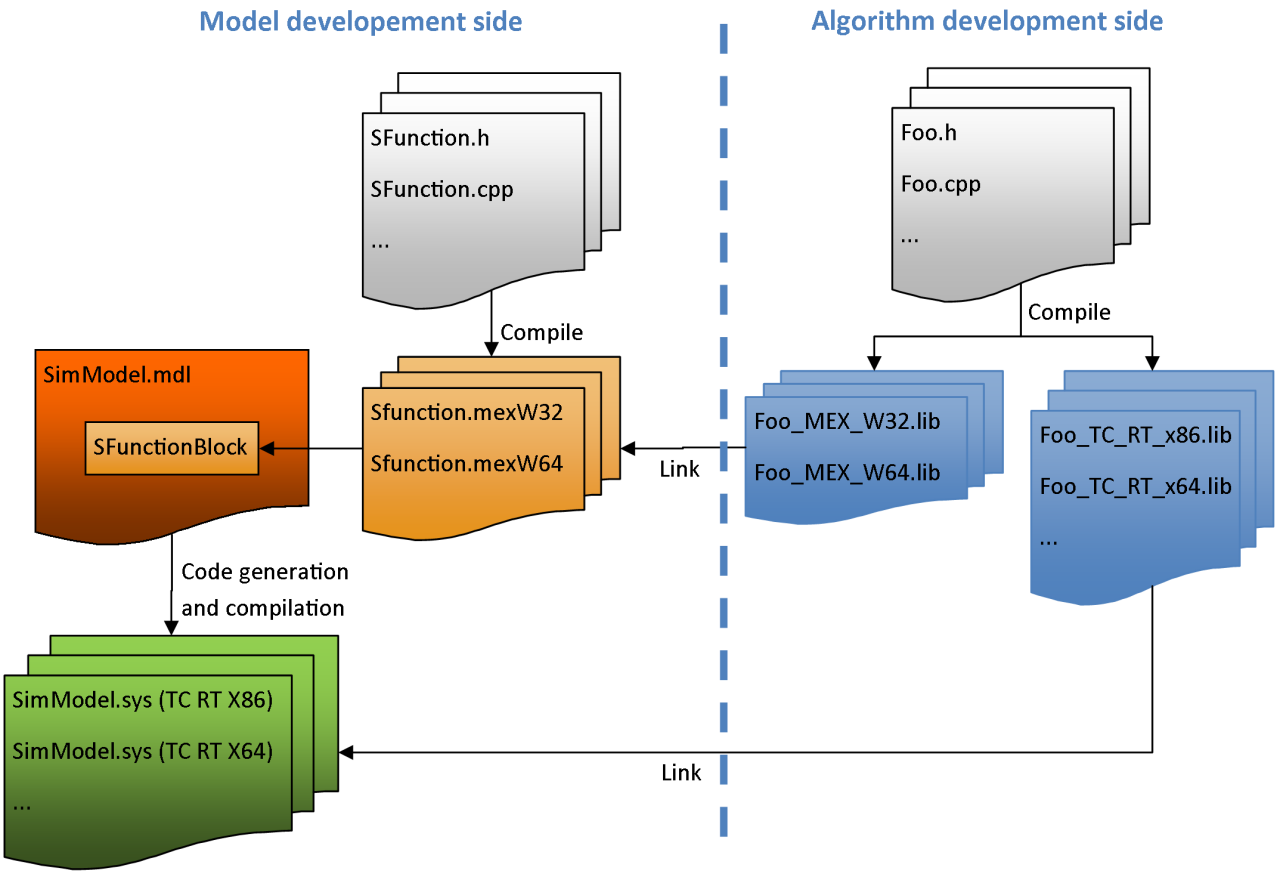
Anwendung

Die Einbindung eigener Codes in statische Bibliotheken kann nützlich sein, um

- die Erstellung von Modulen zu beschleunigen, sofern der Code nahezu unveränderliche Algorithmen enthält
- Kunden TwinCAT Target kompatible SFunction Algorithmen zur Verfügung zu stellen, ohne dass der Quellcode herausgegeben werden muss (es werden nur kompilierte Bibliotheken herausgegeben).

Beschreibung

Im folgenden Beispiel zeigt die Konfiguration des Modulgenerators, wenn von statisch verknüpften Bibliotheken abhänge SFunctions verwendet werden. Für diese Art der Codeintegration muss für alle TwinCAT Plattformen, für die das Modul erstellt werden soll, eine geeignete Bibliothek verfügbar sein.
Die folgende Grafik verdeutlicht den typischen Arbeitsfluss bei der Nutzung von Algorithmen Dritter in einem eigenen Simulink Modell:



In diesem Beispiel ist der Quellcode der „Algorithmus Entwicklungsseite“ verfügbar und kann für alle TwinCAT Plattformen kompiliert werden. Es zeigt wie

- abhängige Bibliotheken erzeugt werden
- solche Bibliotheken bereitgestellt werden (z.B. Kunden)
- solche Bibliotheken in eigenen Modellen verwendet werden

Überblick Projektverzeichnis

https://infosys.beckhoff.com/content/1031/te1400_tc3_target_Matlab/Resources/1539911947.zip enthält alle notwendigen Dateien, um dieses Beispiel zu reproduzieren:

TctSmplSFunWrappedStaticLib.mdl	enthält das Modell, das die SFunction referenziert.
TctSample_SFunLibWrapper.cpp	muss auf Zielsystem vorhanden sein. Enthält den Quellcode der SFunction.
StaticLib.cpp	einfacher Beispiel-Quellcode einer statischen Bibliothek.
BuildLibsAndSFunction.m	enthält ein M-Skript, das die statische Bibliothek für alle derzeit verfügbaren TwinCAT Plattformen und die SFunktion erstellt.
OpenLibProject.m	enthält ein M-Skript, das die MATLAB Build Umgebung für Visual Studio, wie z.B. MATLAB Include- und Bibliotheks-Verzeichnisse definiert. Anschließend wird die statische Bibliothek mit den vordefinierten Umgebungsvariablen in Microsoft Visual Studio 2010 geöffnet.
Unterverzeichnis LibProject	enthält die statische Bibliothek.
Unterverzeichnis BuildScripts	enthält einige M-Skripts für "BuildLibsAndSFunction.m" und "OpenLibProject.m".
_PrecompiledTcComModules	Dieses Unterverzeichnis enthält bereits fertig kompilierte, aus den beiliegenden Simulink-Modellen erzeugte TwinCAT-Module. Diese erlauben es, die Einbindung eines Moduls in TwinCAT ohne vorherige Modulgenerierung auszuprobieren. Sie können z. B. genutzt werden, wenn beispielsweise noch keine MATLAB-Lizenz vorhanden ist. Eine Kurzanleitung zur Installation der Module auf dem Entwicklungsrechner liegt ebenfalls bei. **Achtung:** Um das Modul auf einem x64-Zielsystem starten zu können, muss dieses in den Testmodus versetzt werden!
_PreviousSimulinkVersions	Die oben beschriebenen MDL-Dateien sind im Datei-Format der aktuellen Simulink-Version gespeichert. Dieses Unterverzeichnis enthält die Modelle im Datei-Format älterer Simulink-Versionen.

SFunction und entsprechende statisch verknüpfte Bibliotheken erstellen

Erstellungsvoraussetzungen

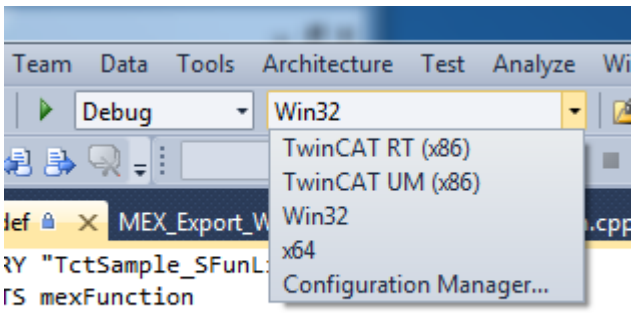
Für die Erstellung der Binärdateien sollte (nicht muss) TwinCAT 3 auf Ihrem Rechner installiert sein. Voraussetzungen:

Windows-Treiberkit	auf dem Rechner installiert, Umgebungsvariable *WinDDK* auf entsprechenden Pfad gesetzt.
TwinCAT SDK	auf dem Rechner installiert, Umgebungsvariable *TwinCatSdk* auf entsprechenden Pfad gesetzt.

Weitere Auskünfte dazu siehe Systemanforderungen in der TwinCAT 3 Dokumentation.

Statische Bibliotheken manuell erstellen

Sie können die statischen Bibliotheken mit Visual Studio manuell erstellen. Führen Sie dazu *OpenLibProject.m* aus. Dadurch werden die erforderlichen Umgebungsvariablen vorbereitet und das SFunction Projekt in Visual Studio geöffnet. Erzeugen Sie ein Projekt für alle Plattformen, die unterstützt werden sollen. Die Ausgabedatei für alle Plattformen ist eine statische Bibliothek:



Erstellen der statischen Bibliothek via Build Skript

Starten Sie *BuildLibsAndSFunction.m.* Dadurch wird die Build-Umgebung vorbereitet und MSBUILD mehrfach aufgerufen, um die lib Dateien für die Plattformen zu erstellen. Anschließend werden alle Build-Ausgabedateien in das Unterverzeichnis *LibProject\TctSample_WrappedStaticLib\BuildOutput* kopiert.
Die .LIB Dateien zum Erzeugen der SFunction und die generierten TwinCAT Module werden ebenfalls in das Verzeichnis *LibProject\TctSample_WrappedStaticLib\LibPackage* kopiert.

Weitergabe statischer Bibliotheken

Das Verzeichnis *LibProject\TctSample_WrappedStaticLib\LibPackage* kann nun an Kunden weitergegeben werden, die diese Bibliothek in ihren eigenen - TwinCAT Target kompatiblen – SFunctions verwenden möchten. Kopieren Sie den Inhalt dieses Verzeichnisses in das Verzeichnis *%TwinCat3Dir%Functions\TE1400-TargetForMatlabSimulink\Libraries* auf dem System des Nutzers. *BuildLibsAndSFunction.m* erledig das auf einem lokalen System. Das Verzeichnis sollte folgenden Inhalt haben:

Unterverzeichnis TwinCAT xx (xxx)	enthält die statischen Bibliotheken für verschiedene TwinCAT Plattformen. Sie werden verwendet, um TwinCAT Module aus dem entsprechenden Simulink Modell zu generieren.
Unterverzeichnisse Win32 / Win64	enthalten die statischen Bibliotheken für die MATLAB Plattformen (32 und/oder 64 Bit). Sie werden verwendet, um TwinCAT Module aus dem entsprechenden Simulink Modell zu generieren. Um die Bibliotheken in diesem Beispiel für 32 und 64 Bit MATLAB zu erzeugen, muss *BuildLibsAndSFunction.m* aus beiden MATLAB Varianten heraus gestartet werden.

MEX-Dateicode kompilieren

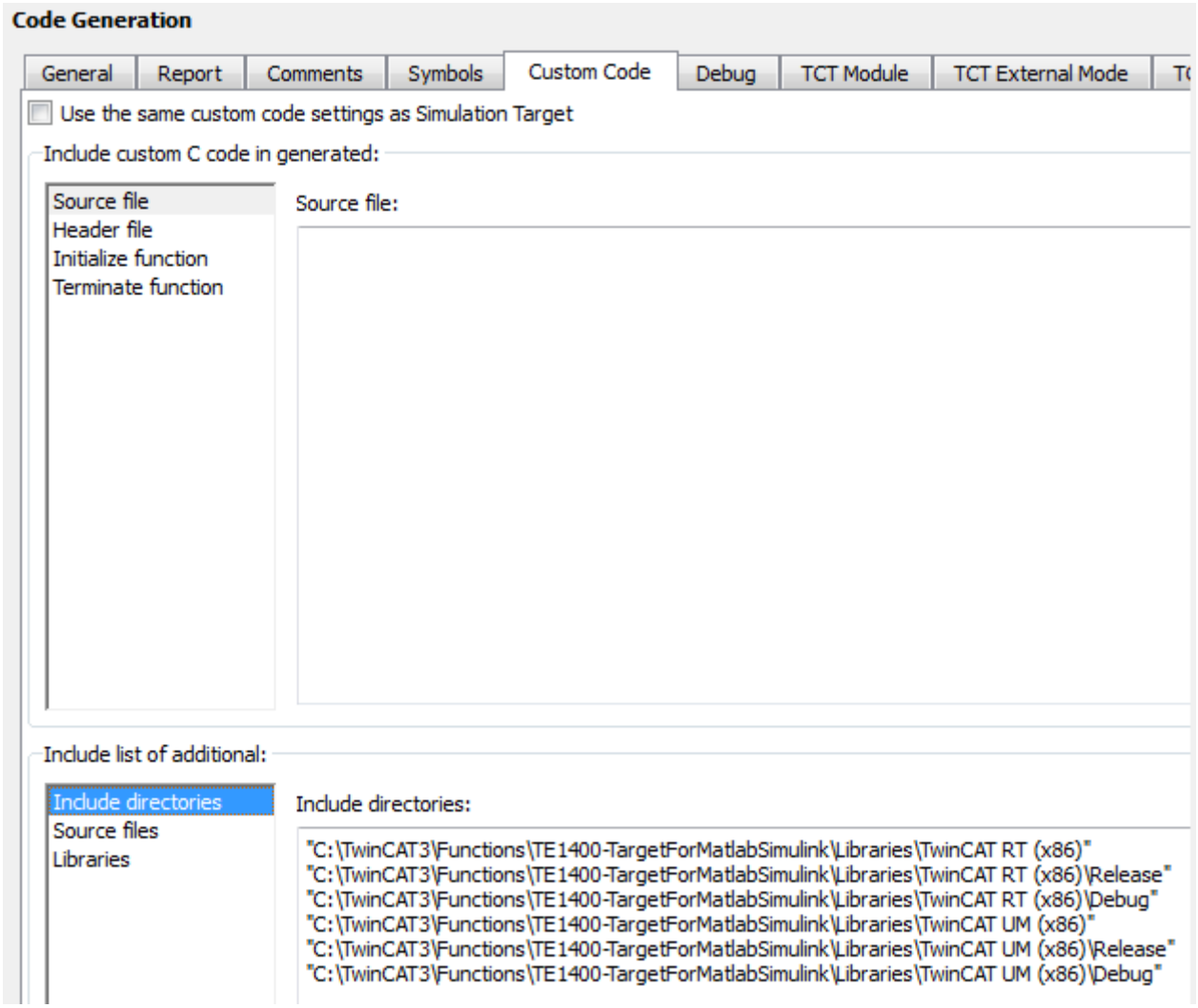
Bevor die SFunction im Simulink Modell verwendet werden kann, muss sie erzeugt werden. Das kann manuell geschehen, wie für andere SFunktionen auch. Der MEX Compiler muss die Anweisung erhalten, die statische Bibliothek mit der SFunktion zu verknüpfen.
BuildLibsAndSFunction.m erledigt das automatisch. Danach gibt es in Ihrem Arbeitsverzeichnis eine Datei "SFunStaticLib.mexw32".
Zum Testen öffnen Sie "TctSmplSFunWrappedStaticLib.mdl" und starten die Simulation. Die Simulation muss ohne Fehlermeldung starten.

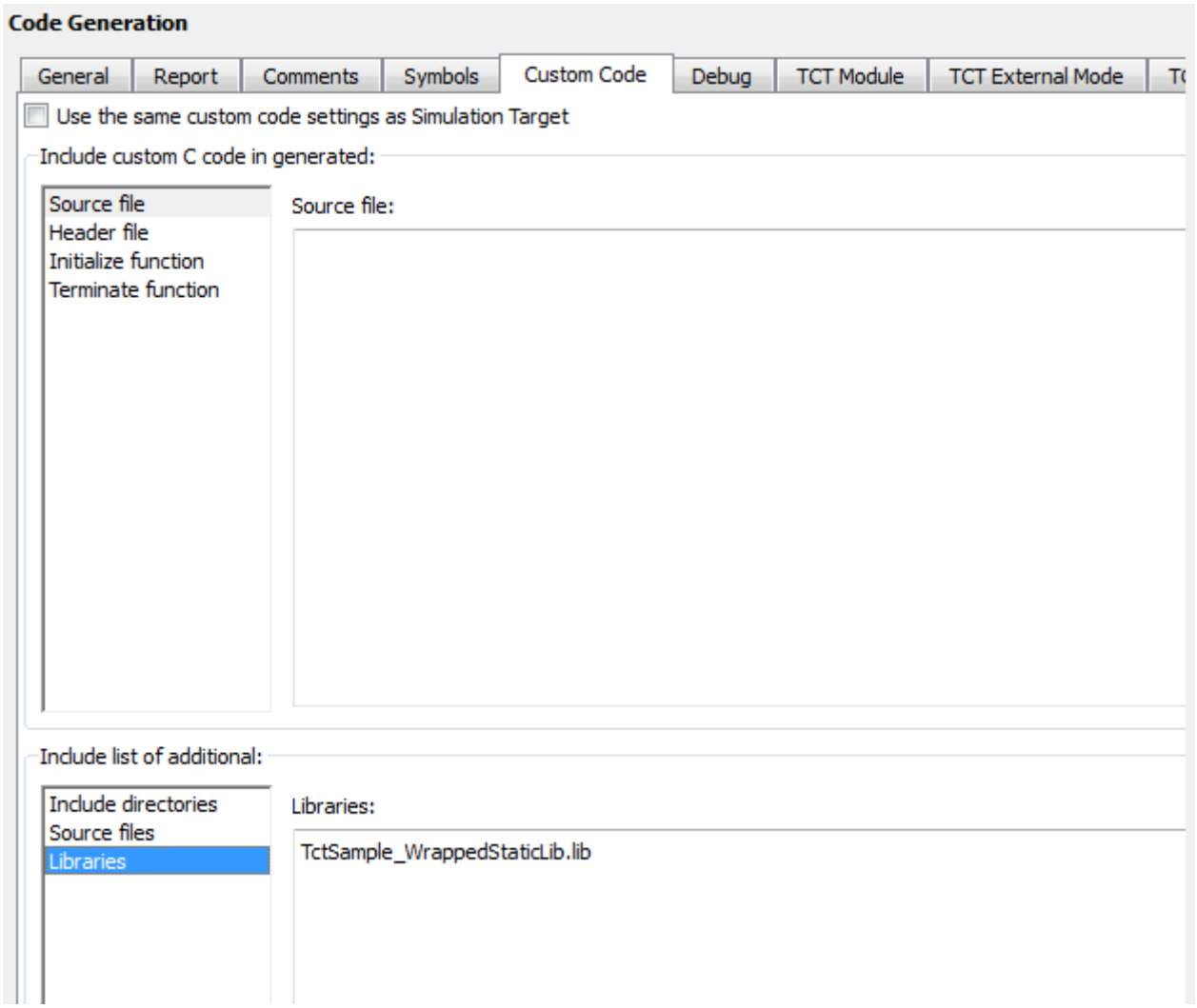
TwinCAT Modul generieren

Modell konfigurieren

Die allgemeinen Einstellungen zum Generieren eines TwinCAT Moduls müssen gesetzt sein, so muss ein Fixed-Step-Solver konfiguriert und unter dem Karteireiter „General“ in den Modellcodereinstellungen das SystemTargetFile "TwinCAT.tlc“ ausgewählt sein. Weitere Informationen zur allgemeinen Konfiguration des

TwinCAT Modulgenerators siehe Quickstart [▶ 15].
Der Codegenerator muss auch wissen, welche statischen Bibliotheken mit dem generierten Code zu verknüpfen sind und wo diese zu finden sind. Tragen Sie diese Informationen in die entsprechenden Optionsfelder des Simulink-Coders ein, wie in den folgenden Abbildungen gezeigt.





Das "Include-Verzeichnisse" sollte bereits automatisch von TwinCAT Target erstellt worden sein. Die "Libraries" Einstellung muss die Namen der zu verknüpfenden statischen Bibliotheken enthalten.

Hintergrundinformation zu diesen Einstellungen:

In diesem Beispiel (und den meisten anderen Fällen) gibt es diese Bibliotheken in den angegebenen Verzeichnissen mehrfach. Welche Version mit dem Modul verknüpft wird, entscheided MSBUILD bei der Verknüpfung der generierten TwinCAT Modul-Binärdateien.

Dieses Beispiel als Vorlage verwenden

Nachfolgend ist der einfachste Weg zu einer eigenen TwinCAT Target kompatible SFunction Dependency kurz beschrieben:

1. Beispielverzeichnis kopieren
2. MDL Datei durch eigene ersetzen
3. VCXPROJ Datei in den Namen Ihrer SFunction umbenennen
4. Eigene Quelldateien in das Verzeichnis der VCXPROJ Datei kopieren
5. Die Skripts *BuildLibsAndSFunction.m* und *OpenLibProject.m* an den neuen Projektnamen anpassen
6. VCXPROJ Datei mit *OpenLibProject.m* öffnen
7. Bestehende CPP-Dateien aus dem Projekt entfernen
8. Eigene CPP Dateien zum Projekt hinzufügen
9. Gegebenenfalls Include-und Dependency-Verzeichnisse sowie Bibliotheken zu Compiler- und Linkereinstellungen hinzufügen
10. Projekt erzeugen (für verschiedene Plattformen und/oder Konfigurationen)

11. VCXPROJ Datei schließen
12. BuildLibsAndSFunction.m starten

3.8.5 ModuleGeneration-Callbacks

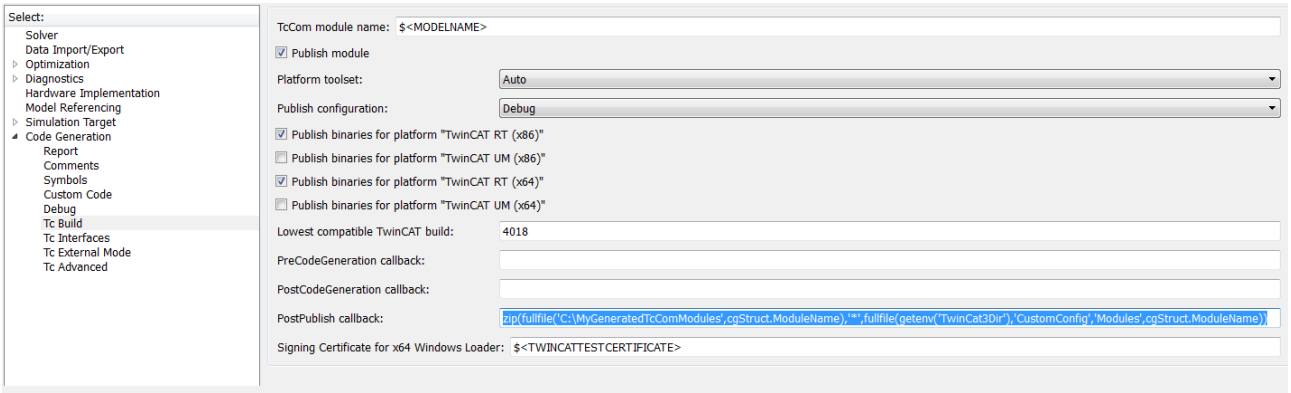
Beispiele zu ModuleGeneration-Callbacks [▶ 25]:

Beispiel	Themen	Beschreibung
Moduldateien als ZIP verpacken [▶ 91]	• PostPublish callback • Archivierung generierter Modul-Dateien	Dieses einfache Beispiel zeigt die automatische Archivierung generierter Modul-Dateien.

3.8.5.1 Moduldateien als ZIP verpacken

Callbacks können beispielsweise verwendet werden, um generierte Modul-Dateien als ZIP-Archiv abzulegen. Erstellen Sie dazu zunächst das Verzeichnis *C:\MyGeneratedTcComModules* und kopieren dann den folgenden Befehl in das **PostPublish callback**-Feld der Codegenerator-Einstellungen des Simulink-Modells unter **Tc Build**:

```
zip(fullfile('C:
\MyGeneratedTcComModules',cgStruct.ModuleName),'*',fullfile(getenv('TwinCat3Dir'),'CustomConfig','Mo
dules',cgStruct.ModuleName))
```



4 Ab Version 2.x.xxxx.x

- TE1400 Target for Simulink® Versionen geringer als 1.2.xxxx.x unterstützen MATLAB R2010b bis MATLAB R2019a.
- TE1400 Target for Simulink® Versionen höher als 2.x.xxxx.x unterstützen MATLAB R2019a und höher.
- Die Installationen für beide Versionen können parallel auf einem Engineering System verwendet werden.
- Kompatibilität der erstellten Module: Siehe Mapping geht verloren bei Reload TMI/TMC [▶ 237].

4.1 Installation

Systemvoraussetzung

Im Folgenden wird zwischen dem Engineering PC und dem Laufzeit-PC unterschieden. Dazu wird folgende Definition getroffen: Auf dem Engineering PC werden Simulink®-Modelle oder MATLAB®-Funktionen in TwinCAT-Objekte durch Einsatz des TwinCAT Target for Simulink® bzw. Target for MATLAB® konvertiert. Ebenso kann, muss aber nicht, auf diesem PC eine TwinCAT-Solution erstellt werden, welche die erstellten Objekte verwendet. Die erstellte TwinCAT-Solution wird dann vom Engineering PC auf einen Laufzeit-PC in die TwinCAT-Laufzeitumgebung zur Ausführung des Projekts geladen.

Auf dem Engineering PC

- MATLAB R2019a oder höher
  - Simulink® und Simulink Coder™ Toolbox für die Nutzung des Target for Simulink®
  - MATLAB® und MATLAB Coder™ Toolbox für die Nutzung des Target for MATLAB®
- Visual Studio 2017 oder höher (Professional, Ultimate oder äquivalente Edition)
  - Bei der Installation muss manuell die Option *Desktop development with C++* ausgewählt werden. Die Option kann auch nachträglich installiert werden.
  - Die Visual Studio Version muss vom TwinCAT XAE Setup unterstützt werden, siehe hier.
- TwinCAT 3.1.4024.7 oder höher
  - Installieren Sie TwinCAT 3 XAE oder Full Setup erst, nachdem Visual Studio mit *Desktop development with C++* installiert wurde.
- TwinCAT Tools for MATLAB® and Simulink® Setup

Auf dem Laufzeit-PC

- Unterstützte Betriebssysteme
  - Windows 7, Windows 10, Windows Server (32bit und 64bit)
  - TwinCAT/BSD®
- TwinCAT XAR Version 3.1.4024.7 oder höher

Gebaute Objekte können einfach weitergegeben werden

Auf einem Engineering PC (oder Build Server) gebaute TwinCAT-Objekte können einfach an weitere Personen weitergereicht werden. Diese benötigen lediglich die TwinCAT-XAE-Entwicklungsumgebung, um die erstellten Objekte (TcCOM- oder PLC-Funktionsbausteine) in einer TwinCAT Solution zu nutzen.

Installation

✓ Installieren Sie eine der unterstützten **Visual Studio**-Versionen, falls nicht bereits vorhanden. Beachten Sie die Installation der Option *Desktop development with C++*.

1. Starten Sie das **TwinCAT 3 XAE oder Full Setup**, falls nicht bereits vorhanden.
   Sollte eine Visual Studio- sowie TwinCAT-Installation bereits vorhanden sein, die Visual Studio Version jedoch nicht den oben genannten Anforderungen entsprechen (z. B. TwinCAT XAE Shell oder Visual

Studio ohne C++-Option), müssen Sie zunächst eine geeignete Visual Studio-Version installieren (ggf. die C++-Option nachinstallieren). Danach müssen Sie das TwinCAT 3-Setup ausführen, um TwinCAT 3 in die neue (oder veränderte) Visual Studio-Version zu integrieren.

2. Falls noch keine **MATLAB®**-Installation auf Ihrem System vorhanden ist, installieren Sie diese. Die Reihenfolge, wann MATLAB® installiert worden ist, ist unerheblich.
3. Starten Sie **TwinCAT Tools for MATLAB® and Simulink®** Setup zur Installation des TE1400.

   ⇨ Die Installation des TE1400 erfolgt innerhalb der TwinCAT-Ordnerstruktur, d. h. sie ist losgelöst von der MATLAB®-Installation.
4. Starten Sie MATLAB® als Administrator und führen Sie *%TwinCAT3Dir%.. \Functions\TE14xx-ToolsForMatlabAndSimulink\SetupTE14xx.p* in MATLAB® aus.

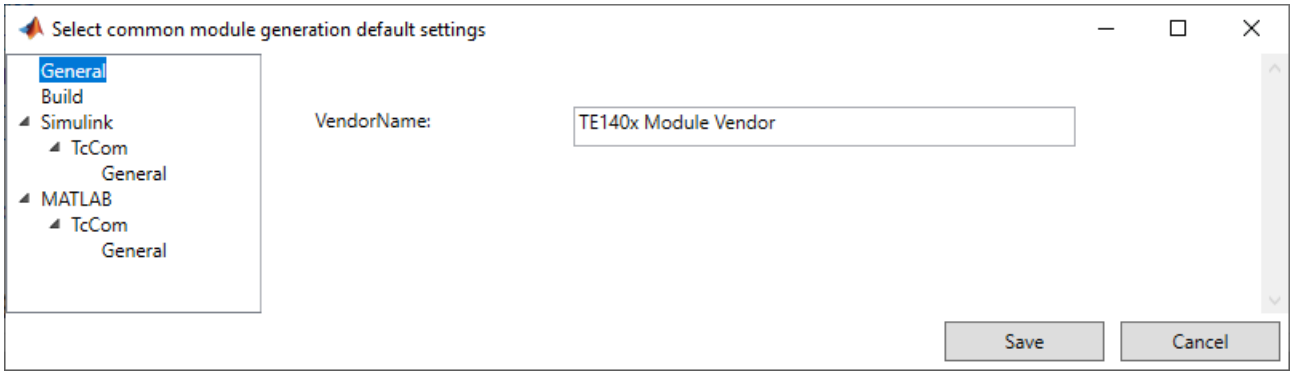
⇨ Es öffnet sich ein Fenster zur Einrichtung. Siehe dazu den folgenden Abschnitt.

Initiale Einrichtung der Software (Common Settings-Dialog)

SetupTE14xx.p ausführen

Nach der Ausführung des p-Files (siehe Schritt 4 unter Installation) öffnet sich ein Dialog, in dem Sie generelle Default-Einstellungen speichern können. Diese Einstellungen gelten dann systemweit, also für *alle* installierten MATLAB®-Versionen.

Sie können die Einstellungen zu diesem Zeitpunkt setzen oder zu einem späteren Zeitpunkt einstellen/ verändern.



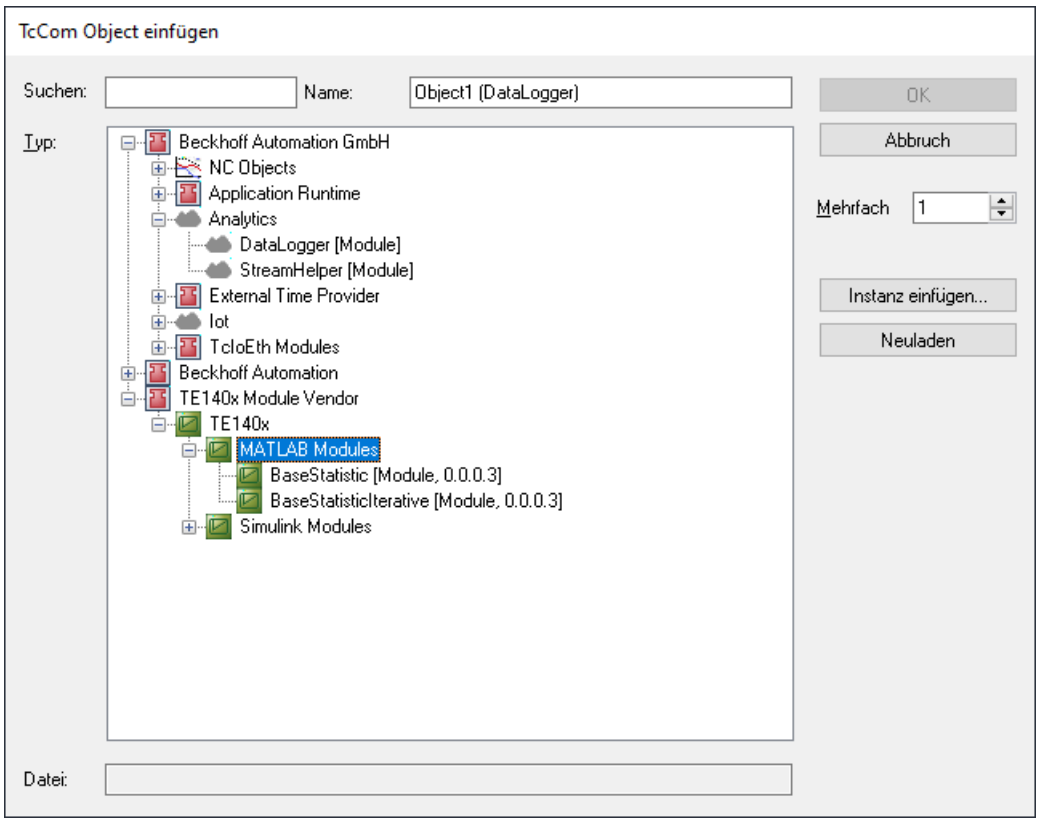
Es gibt folgende Einstellungsmöglichkeiten im Dialog:

- *VendorName*
- *GroupName* (MATLAB®) und
- *GroupName* (Simulink®)

Diese Einstellungen beeinflussen die Hierarchie, in der die generierten TwinCAT-Objekte einsortiert werden. Siehe untenstehende Grafik.

Dort sehen Sie die Einträge:

- VendorName „TE140x Module Vendor“
- GroupName „TE140x“
  - „MATLAB Modules“ für MATLAB® und
  - “Simulink Modules“ für Simulink®



Im beschriebenen Dialog unter dem Reiter **Build** können Sie ein Default-Zertifikat zur Treibersignierung hinterlegen. Die Einrichtungsmöglichkeiten der Treibersignierung werden vollumfänglich im Kapitel Einrichten der Treibersignierung [▶ 96] erläutert.

Das Eintragen eines Zertifikats ist an dieser Stelle optional und nicht zwingend erforderlich.

SetupTE14xx.p „silent“ ausführen

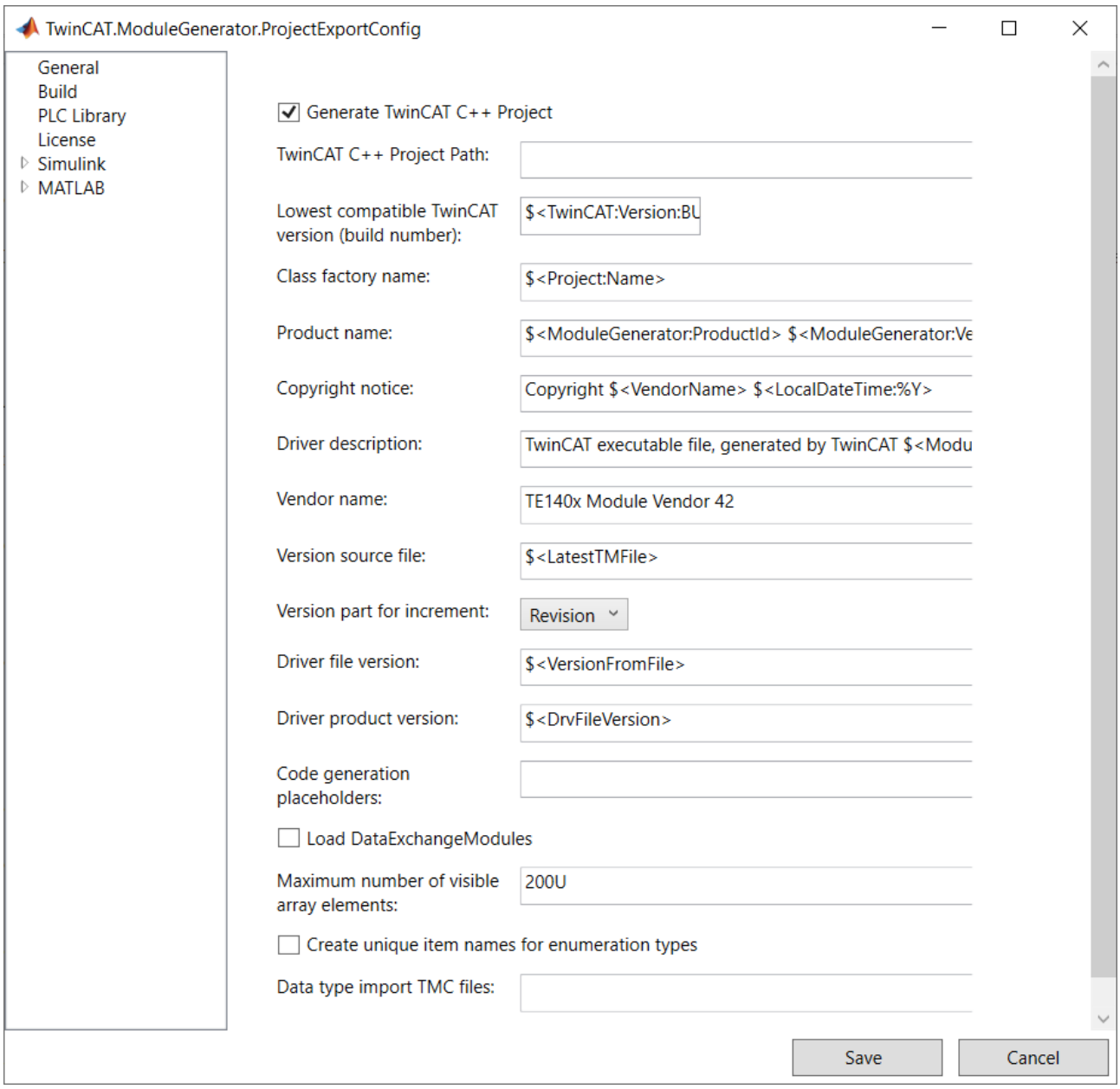
Wenn Sie ohne diesen Dialog das p-File ausführen möchten, können Sie folgenden Befehl nutzen:
`SetupTE140x('Silent', true);`
Dabei werden die Standardwerte des Dialogs gesetzt.

Einrichtung der Software verändern

Um die Default-Einstellungen des TwinCAT-Targets zu verändern, können Sie wie folgt in der MATLAB® Console auf einen Dialog zugreifen:

```
TwinCAT.ModuleGenerator.Settings.Edit
```

Hier werden Ihnen diverse Einträge angeboten, die Sie als Default-Wert hinterlegen können.



Änderungen übernehmen

1. Tragen Sie die neuen Default-Einstellungen im Dialogfenster ein.
2. Bestätigen Sie mit dem Button **Save**.
3. Starten Sie MATLAB® neu.

⇨ Die Änderungen sind übernommen worden.

4.2 Lizenzen

Um die gesamte Funktionalität des TE1400 TwinCAT Target for Simulink® nutzen zu können, sind zwei Lizenzen erforderlich. Zum einen die Engineering-Lizenz TE1400 zur Erstellung von TwinCAT-Objekten aus Simulink® und zum anderen eine Laufzeitlizenz zur Ausführung dieser Objekte in der TwinCAT-Laufzeit.

Engineering-Lizenz

Die Lizenz *TE1400 Target for Simulink®* wird für das **Engineering System** für das Erstellen von TcCOM- und SPS-Bausteinen aus Simulink® benötigt. Zu Testzwecken kann das Produkt im Demomodus auch ohne Lizenz als Demoversion genutzt werden.

Für dieses Produkt ist keine 7-Tage-Testlizenz mit vollständigem Funktionsumfang verfügbar.

Einschränkungen in der Demoversion

Ohne gültige Lizenz TE1400 sind Modelle erlaubt mit maximal:

- Alle cpp und header files vom Simulink Coder™ (inkl. dependent libraries) dürfen insgesamt nicht größer als 100 kB sein.
- 5 Eingangssignalen
- 5 Ausgangssignalen

Mit einer Demolizenz erzeugte Module dürfen nur für nichtkommerzielle Zwecke genutzt werden!

Laufzeitlizenz

Um eine TwinCAT-Konfiguration zu starten, welche eine oder mehrere aus Simulink® generierte TwinCAT-Objekte enthält, ist eine Laufzeitlizenz erforderlich.

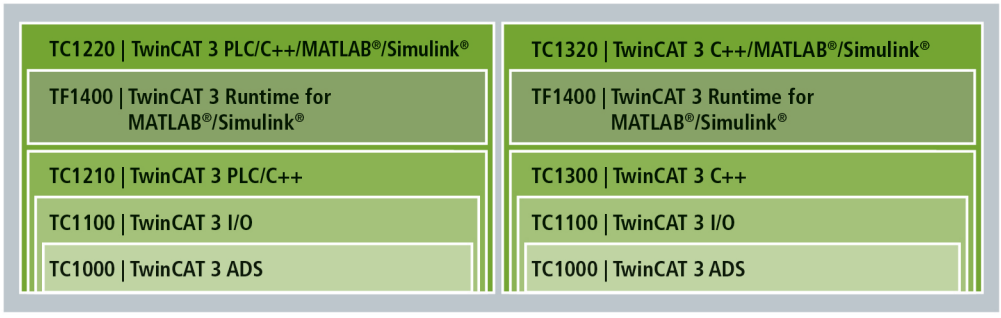
Erforderliche Lizenzen für die Ausführung sind:
TF1400 TwinCAT 3 Runtime for MATLAB®/Simulink® **und** TC1300 TwinCAT 3 C++. Beide Lizenzen sind in den **Lizenz-Bundles** TC1320 und TC1220 enthalten. TC1320 bündelt die Lizenzen TC1300 und TF1400 (C++ und MATLAB®/Simulink® Runtime). TC1220 bündelt die Lizenzen TC1200, TC1300 und TF1400 (PLC, C++ und MATLAB®/Simulink® Runtime).

Lizenz TF1400 Lizenz erst ab TwinCAT 3.1.4026.0

Die Lizenz TF1400 wird erst ab TwinCAT 3.1.4026.0 unterstützt. Für frühere Versionsstände sind nur die Lizenz-Bundles TC1220 oder TC1320 nutzbar. Die Lizenz-Bundles werden auch mit dem Build 4026 und höher weiterhin unterstützt. Die Lizenz TF1400 ist nur dann notwendig, wenn bereits über eine Lizenz die TwinCAT 3 C++-Lizenz auf dem Target System vorhanden ist.

Ohne aktivierte Lizenz kann das Modul und damit auch das TwinCAT-System nicht gestartet werden.

Sie können für die benannten Laufzeitlizenzen eine 7-Tage-Testlizenz erzeugen, die erste Tests ohne den Kauf der Lizenz ermöglicht.



4.3 Einrichten der Treibersignierung

Ein OEM-Zertifikat Level 2 erstellen

Aus MATLAB® oder Simulink® generierte TwinCAT-Objekte basieren, wie auch TwinCAT C++-Objekte, auf einem tmx-Treiber (TwinCAT Module Executable). Diese Treiber müssen mit einem OEM-Zertifikat Level 2 signiert werden, damit dieser in der TwinCAT-Laufzeit auf dem Laufzeit-PC geladen werden kann.

Unter folgenden Links finden Sie eine ausführliche Dokumentation zur Erstellung eines OEM-Zertifikats zur Treibersignierung.

- Allgemeine Dokumentation zu OEM-Zertifikaten
- Anwendungsbezogene Dokumentation zu tmx-Treibersignierung

Das Wichtigste in Kürze:

- Sie können sich selbst ein Zertifikat erstellen. Gehen Sie dazu in Visual Studio auf: **Menu bar > TwinCAT > Software Protection…**
- Sie benötigen ein OEM-Zertifikat Crypto Version 2 (Option: *Sign TwinCAT C++ executables (*.tmx)*).
- Alle Treiber (für 32bit und für 64bit Betriebssysteme) müssen signiert werden.
- Treiber können auch ohne Signierung erstellt und nachträglich signiert werden.
- Für Testzwecke in der Entwicklungsphase genügt ein nicht-gegensignierter Zertifikat.
- Gegensignierte Zertifikate können bei Beckhoff kostenfrei bestellt werden (TC0008).

Nutzen eines OEM-Level-2-Zertifikats zur Treibersignierung

Zum Signieren von tmx-Treibern benötigen Sie ein Zertifikat und ein dem Zertifikat zugehöriges Passwort.

Handling des Zertifikats

Es gibt vier Varianten zur Signierung von tmx-Treibern.

Variante 1: Systemweites Default-Zertifikat für TwinCAT C++ und TE14xx

Sie können auf einem Engineering PC ein Default-Zertifikat setzen, welches immer für TwinCAT C++, Target for MATLAB® und Target for Simulink® genutzt wird, sofern Sie kein anderes Zertifikat explizit angeben.

Für diese Variante nutzen Sie eine Windows-Umgebungsvariable. Legen Sie eine neue **Umgebungsvariable** unter **User > Variables** an mit:

Variable: *TcSignTwinCatCertName*

Value: Name des gewünschten Zertifikats

Verfügbare Zertifikate liegen unter *TwinCAT\3.1\CustomConfig\Certificates.*

Variante 2: Systemweites Default-Zertifikat für TE14xx

Sie können in ihrer MATLAB®-Umgebung ein Default-Zertifikat setzen, welches immer für Target for MATLAB® und Target for Simulink® (**nicht** TwinCAT C++) genutzt wird, sofern Sie kein anderes Zertifikat explizit angeben.

Öffnen Sie den oben genannten Common Settings-Dialog [▶ 93] mit *TwinCAT.ModuleGenerator.Settings.Edit* und tragen Sie das gewünschte Default-Zertifikat ein unter **Build > Certificate name for TwinCAT signing**. Dieses Zertifikat wird in Ihrem User-Verzeichnis als Default gespeichert und von allen MATLAB®-Versionen auf Ihrem System als Standard genutzt.

Variante 3: Zertifikat in der Konfiguration des Simulink®-Modells

Sie können für jeden Build-Vorgang ein Zertifikat explizit benennen. Für Variante 3 müssen Sie keine weiteren Einstellungen vorab treffen. Sie können vor jedem Build-Vorgang ein Zertifikat Ihrer Wahl für genau diesen Build-Vorgang definieren.

Target for Simulink®: **TC Build > Certificate for TwinCAT signing**

Target for MATLAB®: Property `SignTwinCatCertName`

Variante 4: Ohne Zertifikat bauen und später mit TcSignTool signieren

Sie können ohne Zertifikat bauen und nachträglich mit dem **TcSignTool** signieren. Für **Variante 4** können Sie das TcSignTool verwenden.

Das TcSignTool ist ein Kommandozeilen-Programm, welches sich im Pfad `..\TwinCAT\3.1\sdk\Bin\` befindet. Öffnen Sie bspw. das Command Prompt und führen Sie `tcsigntool sign /?` aus, um die Hilfe angezeigt zu bekommen.

TcSignTool aus MATLAB® bedienen

Aus MATLAB® kann das Tool mit dem Befehl `system()` oder mit `!` aufgerufen werden.

```
Command Window
>> system('C:\TwinCAT\3.1\sdk\Bin\tcsigntool sign /?')
TcSignTool Version 3.1.4024.22

Sign files with certificate:

TcSignTool.exe sign /f certificatefile {/p password|/g} [/q] file1 [file2 ...]

    /g  Password read from stdin, interactive mode is not supported
        Usage examples with powershell:
        Get-Content pwd.txt | .\TcSignTool.exe command /f certificatefile /g ...
        Read-Host "Enter Password" -AsSecureString | ConvertFrom-SecureString -AsPlainText | .\TcSignTool.exe command /f certificatefile /g ...

    /q  quiet mode

    returns 0 = succeeded, 1 = failed

ans =

     1

fx >>
```

Beispielaufruf zur Signierung eines tmx-Treibers für TwinCAT:

```
TcSignTool sign /f "C:\TwinCAT\3.1\CustomConfig\Certificates\ MyCertificate.tccert" /p MyPassword
"C:\TwinCAT\3.1\Repository\TE140x Module Vendor\ModulName\0.0.0.1\TwinCAT RT (x64)\MyDriver.tmx"
```

Handling des Zertifikat-Passworts

Nur bei *Variante 4* des Zertifikat-handlings wird das Passwort direkt eingegeben. Für *Variante 1 bis 3* ist zusätzlich zur Angabe des Zertifikats das zugehörige Passwort zu hinterlegen. Das Passwort soll aus Sicherheitsgründen nicht im Quellcode im Simulink®-Modell oder im MATLAB®-Code eingetragen werden. Mit dem TcSignTool können Sie Ihren Zertifikaten zugehörige Passwörter obfuskiert in der Registry des Windows-Betriebssystems ablegen. Dadurch ist das Passwort zu einem bestimmten Zertifikat im Betriebssystem (für den entsprechenden User) bekannt und wird automatisch verwendet.

Die Ablage des Passworts wird mit folgendem Aufruf durchgeführt:

```
tcsigntool grant /f "C:\TwinCAT\3.1\CustomConfig\Certificates\MyCertificate.tccert" /p MyPassword
```

Die Ablage des obfuskierten Passworts erfolgt in der Registry unter:
`HKEY_CURRENT_USER\SOFTWARE\Beckhoff\TcSignTool\`

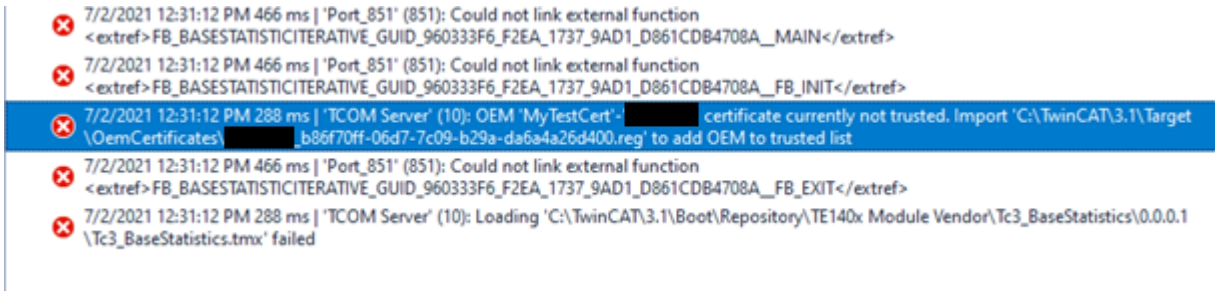
Mit folgendem Aufruf wird das Passwort gelöscht:

```
tcsigntool grant /f "C:\TwinCAT\3.1\CustomConfig\Certificates\MyCertificate.tccert" /r
```

Verhalten der TwinCAT-Laufzeit

Wird ein aus MATLAB® oder Simulink® erstelltes TwinCAT-Objekt mit signiertem Treiber in einer TwinCAT-Solution genutzt und mit **Activate Configuration** auf ein Zielsystem geladen, ist Folgendes zu beachten:

Jede TwinCAT-Laufzeit (XAR) hat eine eigene eine White-List an vertrauenswürdigen Zertifikaten. Ist das Zertifikat, welches zur Signierung genutzt wurde, nicht in dieser White-List enthalten, wird der Treiber nicht geladen. Im TwinCAT Engineering (XAE) wird eine entsprechende Fehlermeldung ausgegeben.



Die Fehlermeldung enthält die Anweisung, ein Registry File, welches auf dem Zielsystem automatisch erstellt wurde, auf dem Zielsystem als Administrator auszuführen. Dieser Vorgang fügt das genutzte Zertifikat der White-List hinzu.

Registry File ist nur vom OEM-Zertifikat abhängig

Das Registry File kann ebenso auf weiteren Zielsystemen genutzt werden. Es enthält nur Informationen über das genutzte OEM-Zertifikat und ist nicht zielsystemabhängig.

Wenn Sie ein nicht-gegensigniertes OEM-Zertifikat zur Signierung nutzen, müssen Sie zudem ihr Zielsystem in den Test-Modus versetzen. Führen Sie dazu den folgenden Befehl als Administrator auf dem Zielsystem aus:

```
bcdedit /set testsigning yes
```

Wenn Sie ein gegensigniertes OEM-Zertifikat nutzen, ist dieser Schritt nicht notwendig.

4.3.1 Nutzerzertifikate zur Auslieferung ohne Testmode

Systemvoraussetzungen

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

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

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

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

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

Auf welchen Rechnern ist das TwinCAT 3 Nutzerzertifikat TC0008 erforderlich?

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

Gültigkeit des TwinCAT 3 Nutzerzertifikats

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

Was passiert, wenn die Gültigkeit des Zertifikats abgelaufen ist?

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

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

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

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

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

Das neu signierte Zertifikat ist damit vollständig kompatibel zur Ursprungsversion.

4.3.1.1 TwinCAT 3 Nutzerzertifikat beantragen

Übersicht über den Bestell- und Validierungsprozess

Die Beantragung eines TwinCAT Nutzerzertifikates erfordert eine offizielle Bestellung.

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

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

1. Kontaktieren Sie Ihren Beckhoff Vertriebskontakt und kündigen Sie die Beantragung eines TwinCAT 3 OEM-Zertifikats an. Bestellen Sie “TC0008”.
2. Wichtig: Geben Sie, als Anfrager, Ihre Kontaktdaten als Lieferadresse (= Kontaktname und Email-Adresse) und den Einsatzbereich des Zertifikats an (Firmenname, Adresse).
3. Die im Auftrag angegebenen Kontaktdaten werden verifiziert und Sie (der in der Lieferadresse genannte Anfrager) werden vom Beckhoff Vertrieb kontaktiert.
4. Bei der Beantragung eines neuen OEM-Zertifikats erstellen Sie im TwinCAT 3 Engineering eine „Certificate Request Datei“ [▶ 100].
5. Ermitteln Sie mit Hilfe des TwinCAT Engineerings den „File Fingerprint“ der OEM-Zertifikatsdatei (siehe File Fingerprint der Zertifikatsdatei ermitteln [▶ 104]). Teilen Sie diesen File Fingerprint im Rahmen Ihrer Kontaktdatenverifizierung dem Beckhoff Vertriebskontakt mit. Die Übermittlung des File Fingerprints muss auf einem anderen Kommunikationsweg erfolgen als für die Zusendung der OEM-Zertifikatsanfragedatei.
6. Schicken Sie die „OEM-Zertifikatsdatei“ nun an den Beckhoff Vertriebskontakt.
7. Nach der Signierung der Zertifikatsdatei in der Beckhoff Zentrale erhalten Sie diese über Ihren Ansprechpartner per Email zugesandt.

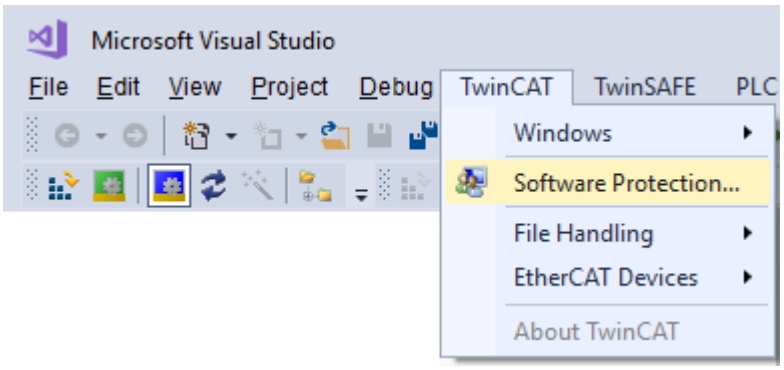
Beachten Sie, dass die Validierung der Kontaktdaten und die Ausstellung des Zertifikats einige Tage Zeit in Anspruch nehmen können.

4.3.1.2 Erstellung der Certificate Request Datei für TC0008

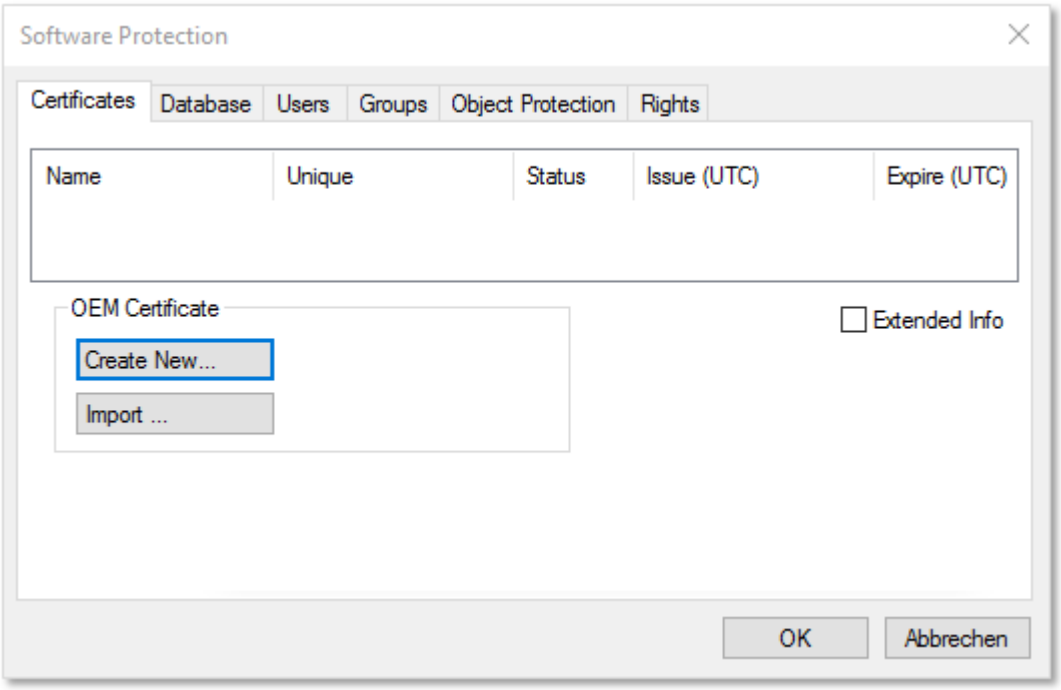
Systemvoraussetzungen

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

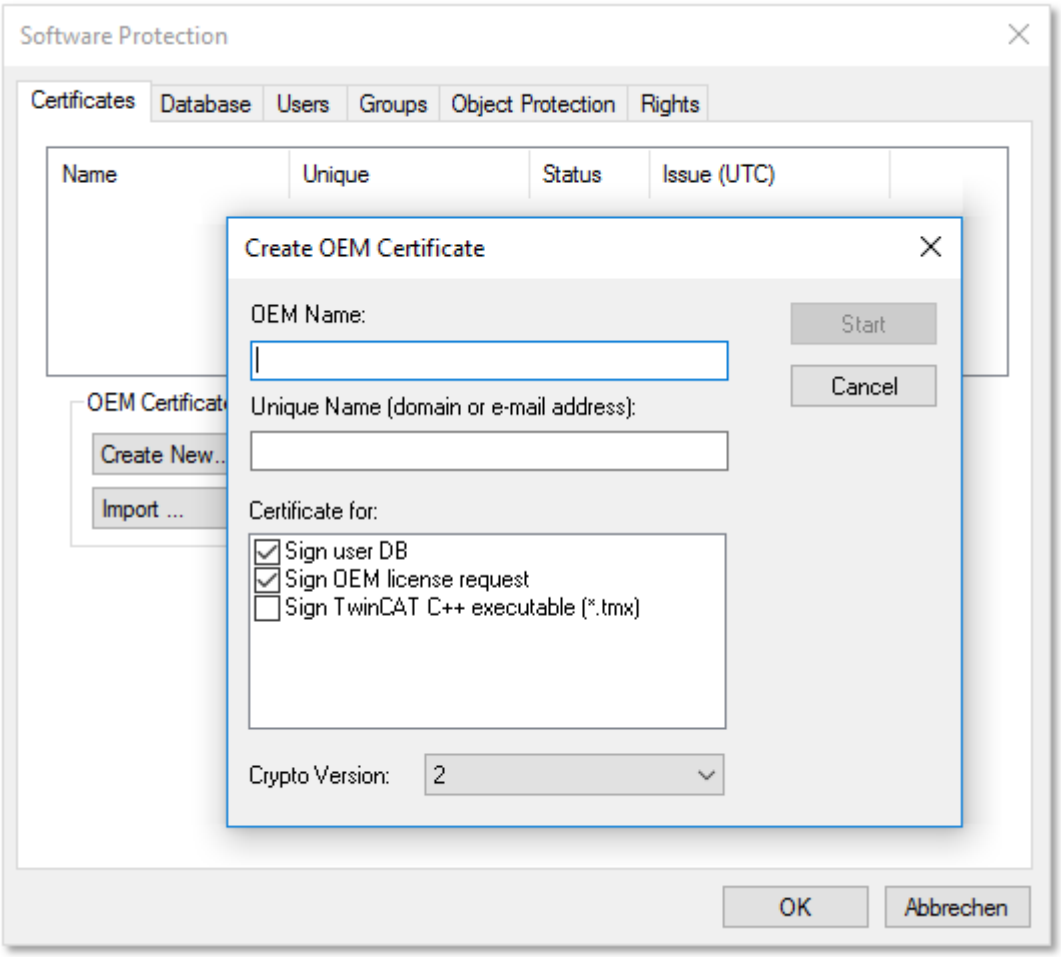
1. Rufen Sie den Konfigurator der Software Protection auf. Wählen Sie dazu im Hauptmenü unterhalb des Punktes **TwinCAT** den Menüpunkt **Software Protection** aus:



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

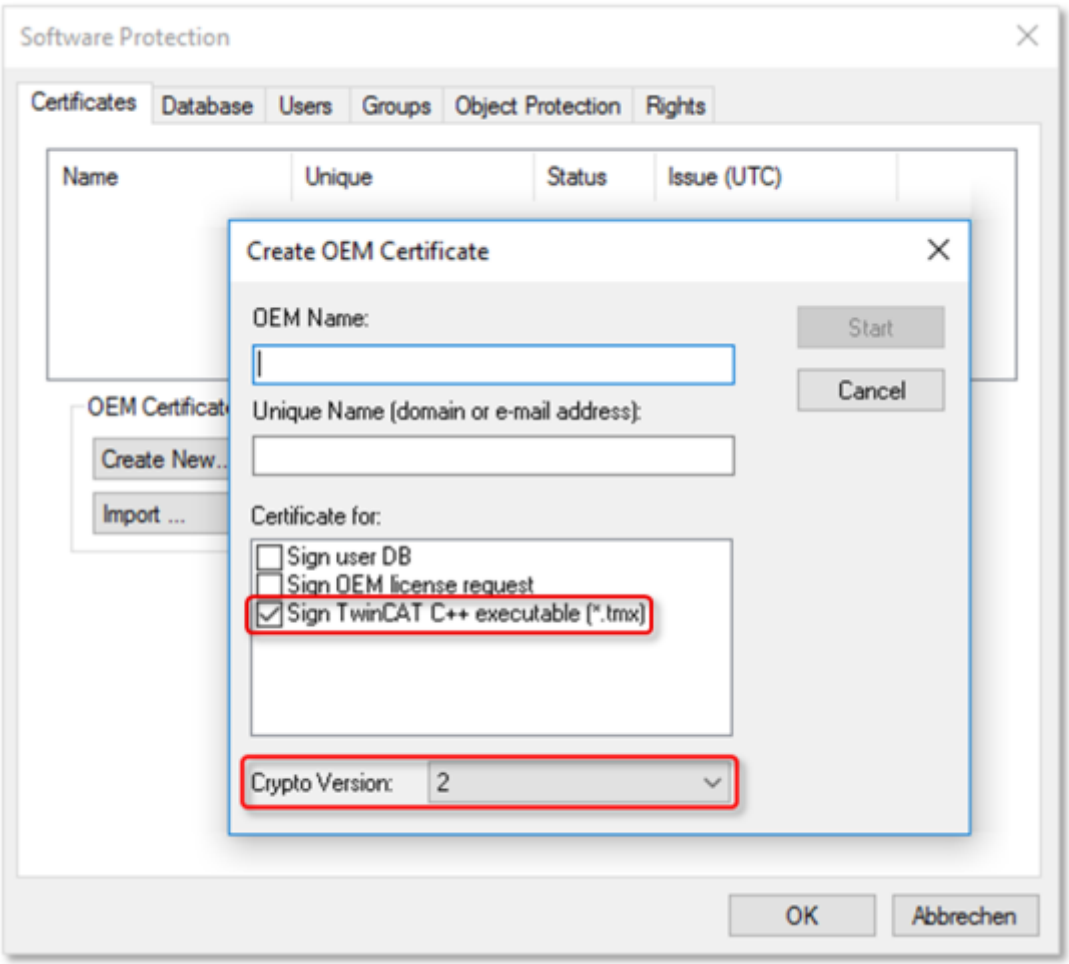


3. Das Eingabefenster **Create OEM Certificate** öffnet sich:



4. Geben Sie die erforderlichen Daten ein:
   - Geben Sie im Textfeld **OEM Name** Ihren Firmennamen ein. Der Name muss einen klaren Bezug zu Ihrer Firma oder Ihrem Unternehmensteil haben.
   - Geben Sie einen **Unique Name** ein. Der „OEM Unique Name“ muss ein einmaliger Name sein, anhand dessen der Eigentümer des Zertifikats weltweit eindeutig identifiziert werden kann, vorzugsweise die URL der Webseite Ihrer Firma oder Ihre E-Mail-Adresse. Die E-Mail-Adresse muss eine Firmen-E-Mail-Adresse sein, also eindeutig Ihrer Firma zugeordnet werden können.

- Markieren Sie die Checkbox **Sign TwinCAT C++ executables**:



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

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

5. Wenn Sie die Daten eingegeben haben, klicken Sie auf **Start** und Sie können ein Verzeichnis auswählen, um die Datei zu speichern.
   Übernehmen Sie einfach das vorgeschlagene Verzeichnis „c:\twincat\3.1\customconfig\certificates“. Sie benötigen die neu erzeugte Datei in diesem Verzeichnis, um in einem späteren Schritt den „File Fingerprint“ für diese Datei auslesen [▶ 104] **zu können.**

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

6. Vergeben Sie ein Passwort für den OEM Private Key**.**

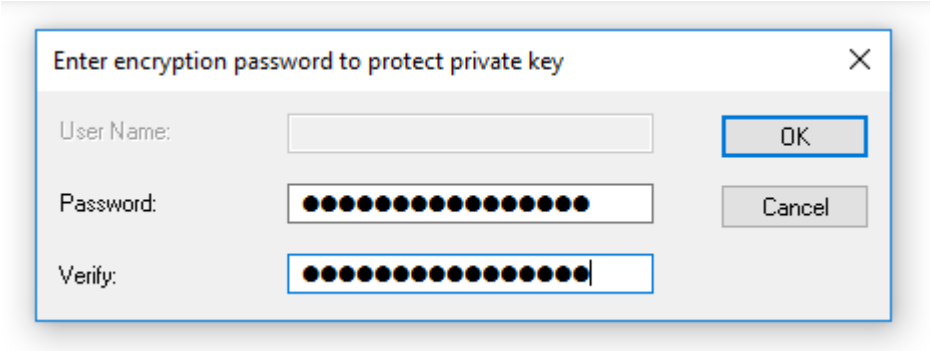
Wichtig: Passwort-Sicherheit!

Verwenden Sie unbedingt ein starkes Passwort für Ihr Zertifikat!
Schützen Sie Ihr Passwort durch geeignete Maßnahmen, damit es nicht in fremde Hände fallen kann!

Passwort bei Verlust nicht wiederherstellbar

Beckhoff kann Ihr Passwort nicht wiederherstellen oder zurücksetzen. Wenn Sie das Passwort für Ihr Zertifikat vergessen oder verlieren, können Sie das Zertifikat nicht mehr verwenden und müssen ein neues Zertifikat ausstellen lassen.

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



⇨ Die Datei wird gespeichert.

Die so erzeugte „Certificate Request Datei“ muss nun noch von der Beckhoff Zertifikatsstelle signiert werden, um gültig zu sein. Das Verfahren dazu wird im Kapitel Zertifikat beantragen [▶ 100] beschrieben.

4.3.1.3 File Fingerprint der OEM-Zertifikatsdatei ermitteln

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

Systemvoraussetzung

Diese Funktionalität erfordert mindestens die Version TwinCAT 3.1 Build 4024.

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

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

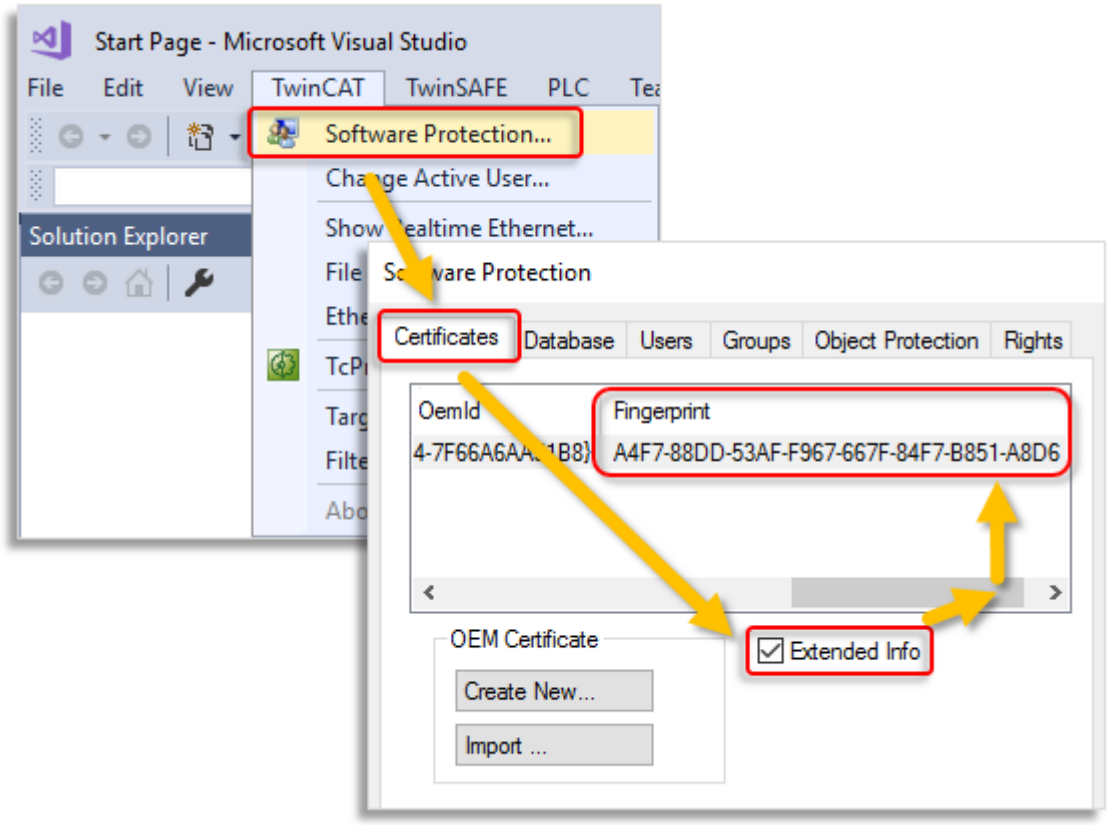
Für diese Funktion ist es erforderlich, dass die OEM-Zertifikatsdatei in diesem Verzeichnis liegt: „c:\twincat\3.1\customconfig\certificates“.

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

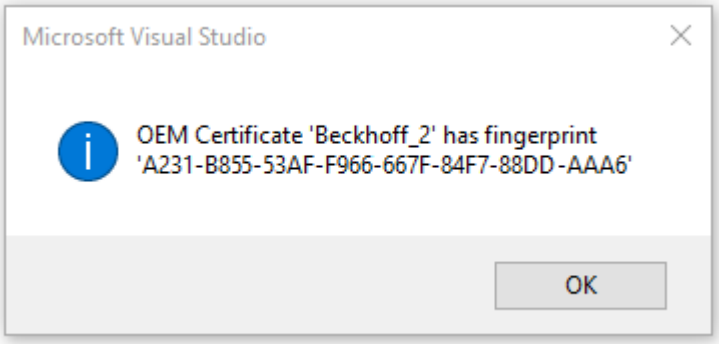
Sofern Sie beim Erstellen der „OEM Certificate Request Datei“ das vorgeschlagene Verzeichnis nicht geändert haben, liegt die Datei bereits in diesem Verzeichnis.

Vorgehensweise:

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



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



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

4.3.1.4 Speichern des fertig signierten TwinCAT Nutzerzertifikates

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

Systemvoraussetzungen

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

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

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

Auf welchen Rechnern ist das TwinCAT 3 Nutzerzertifikat TC0008 erforderlich?

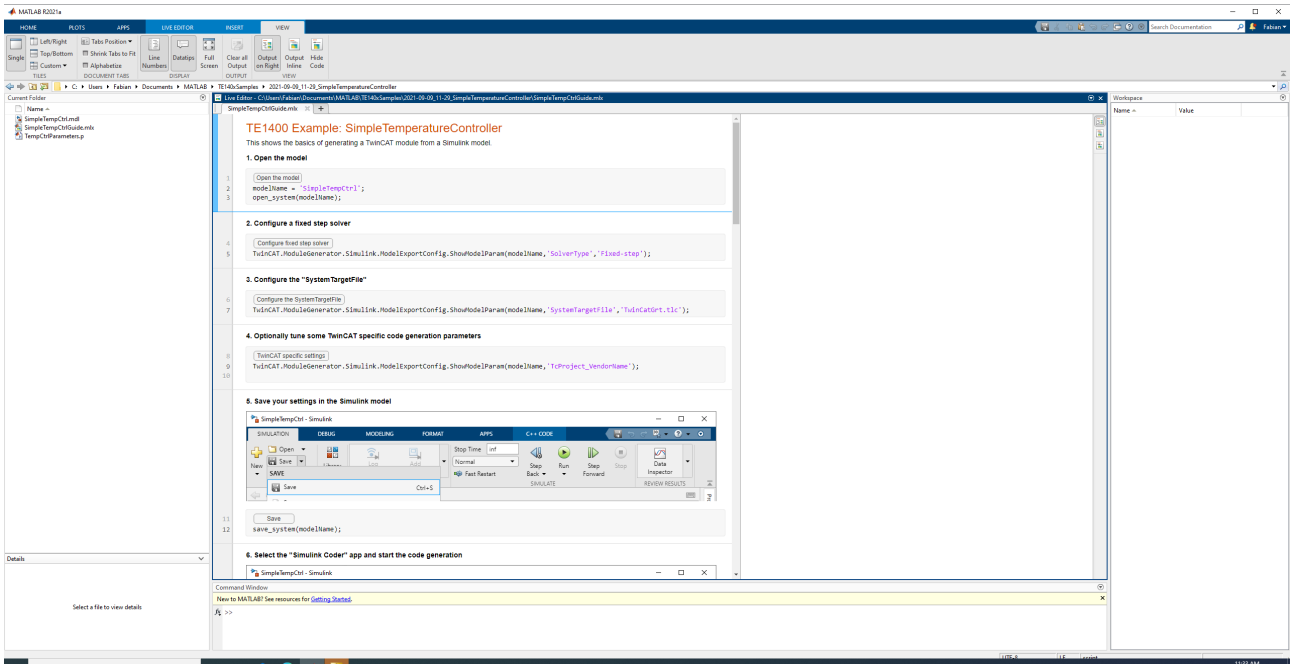
Das TwinCAT 3 Nutzerzertifikat sollte sich ausschließlich auf dem Engineering-Rechner befinden, auf dem die Signierung der TMX-Dateien erfolgt – also **NICHT** auf jedem Zielsystem.

4.4 Quickstart

Starten mit einem einfachen Simulink®-Modell

✓ Nutzen Sie gern die Möglichkeit, unsere eingebauten Samples für erste Schritte mit dem TwinCAT Target for Simulink® zu gehen. Im MATLAB® Command Window erhalten Sie eine Liste der verfügbaren Samples über: `TwinCAT.ModuleGenerator.Samples.List`

1. Wählen Sie beispielsweise den `SimpleTemperatureController` aus und starten Sie das Beispiel über den Start-Link im Command Window.



⇨ Im Folgenden wird der Quickstart entlang dieses Samples ausgeführt.

2. Starten Sie zu Beginn, indem Sie den Button **Open the model** im Live Script anwählen.

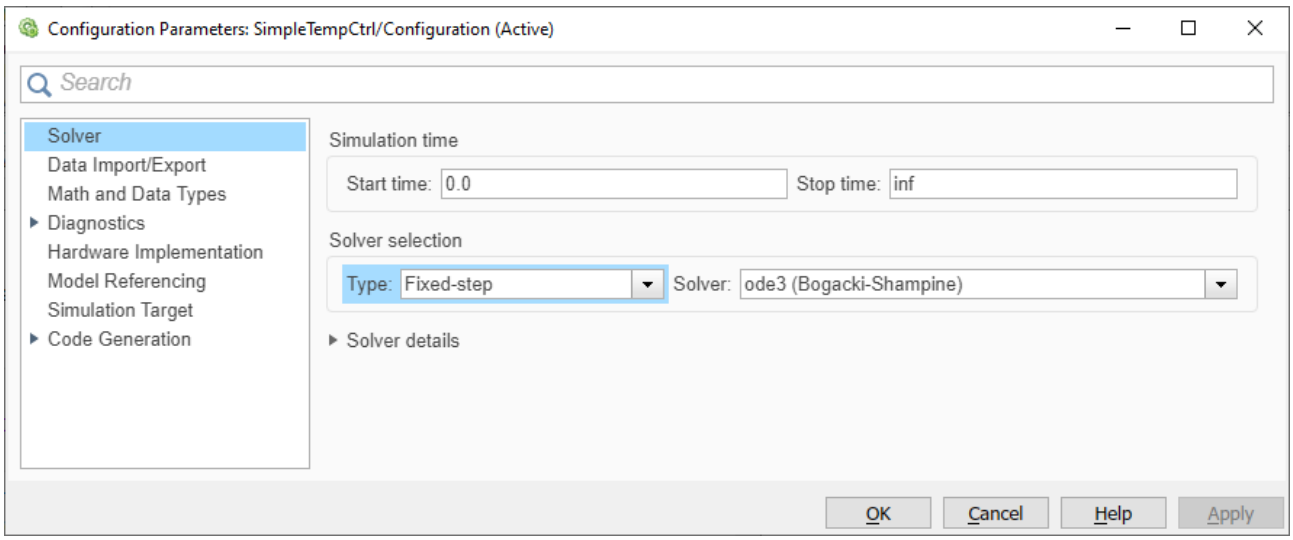
⇨ Für die weiteren Konfigurationsschritte in Simulink® können Sie im Live Script einfach den jeweils nächsten Button betätigen.

Einsteigervideo

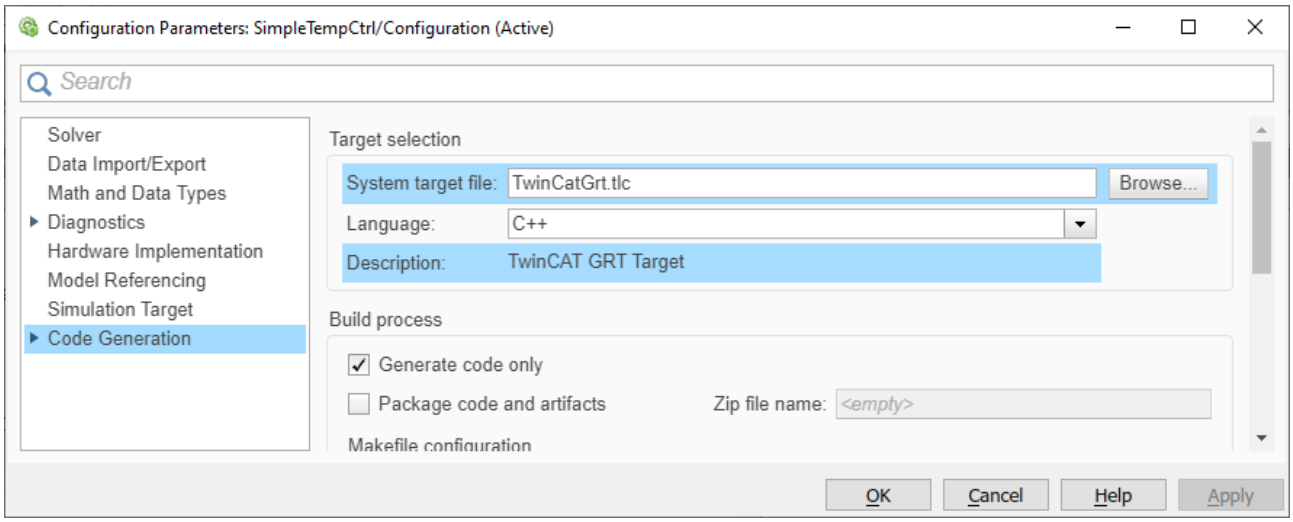
Als Einstieg kann ebenso folgendes Video (nur auf Englisch verfügbar) genutzt werden: TwinCAT Target for Simulink®

Die Konfigurationsschritte in Simulink®

1. Wählen Sie einen Fixed-Step solver aus. Gehen Sie dazu in die **Configuration Parameters** des Modells.

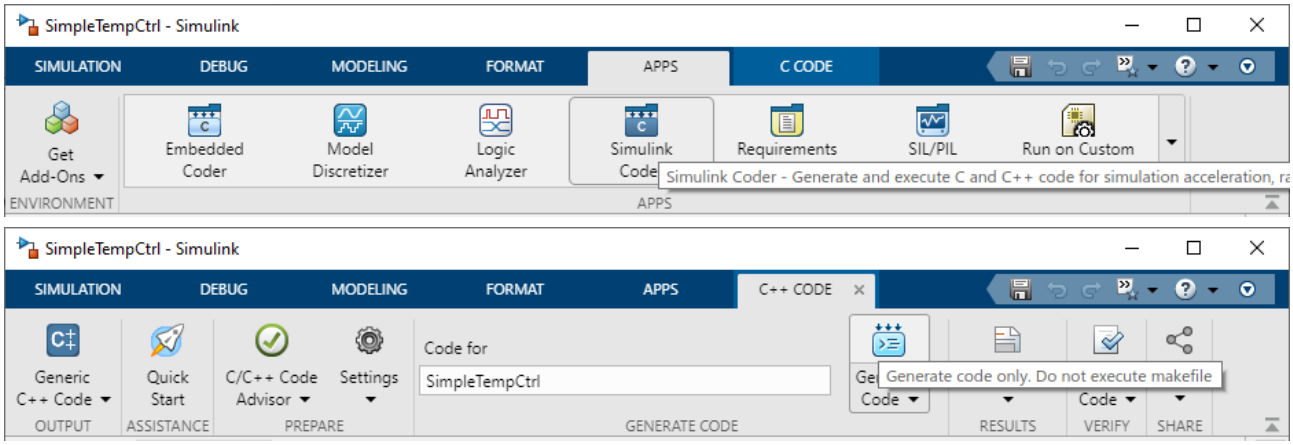


2. Wählen Sie das System Target file zu „TwinCatGrt.tlc“.



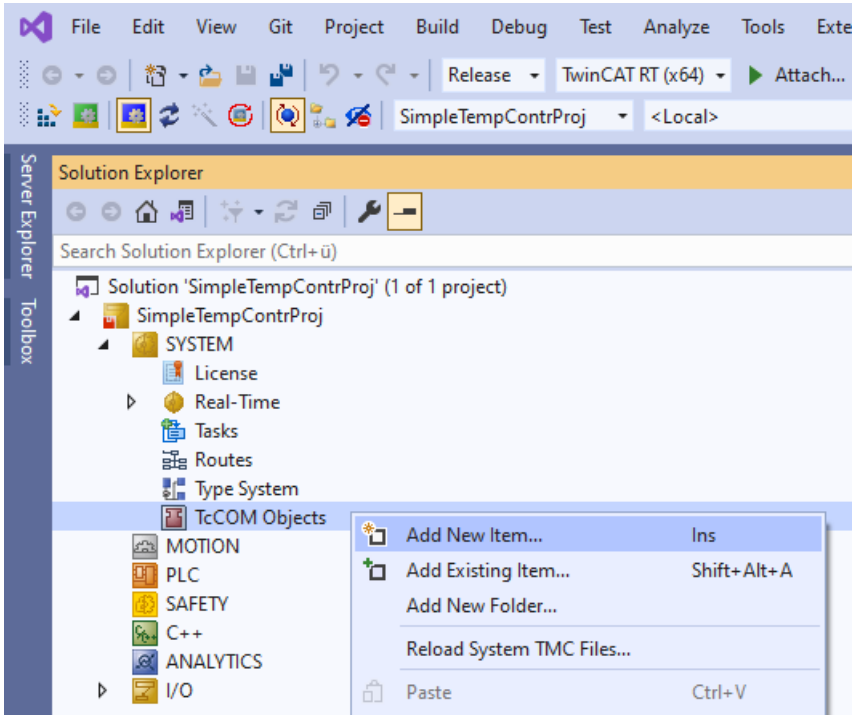
Optional: Stellen Sie unter *Optimization* den Parameter *Default parameter behavior* auf Tunable, damit Sie in TwinCAT weiterhin Modellparameter verändern können. Siehe auch Parametrierung einer Modulinstanz [▶ 190].

3. Speichern Sie ihre Änderungen im Simulink®-Modell.
4. Starten Sie die Code-Generierung über die Simulink Coder™ App.

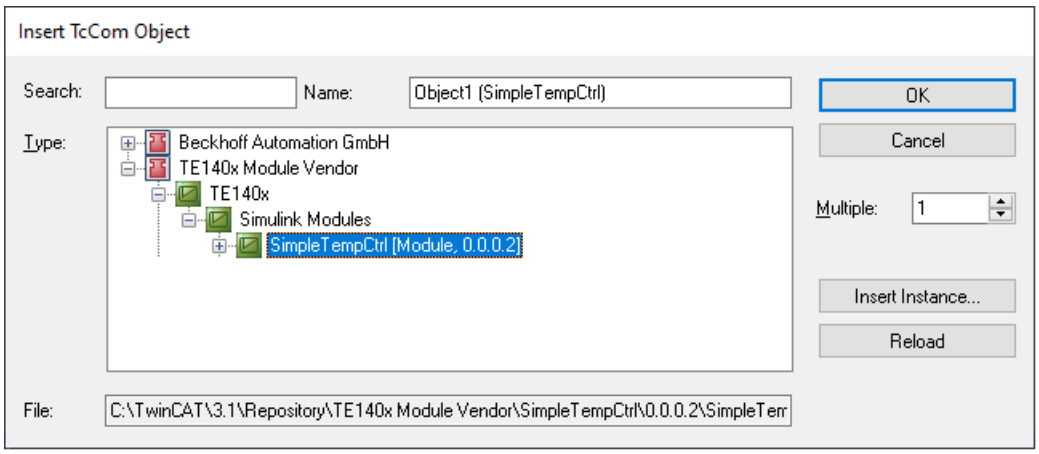


TcCOM in TwinCAT einfügen

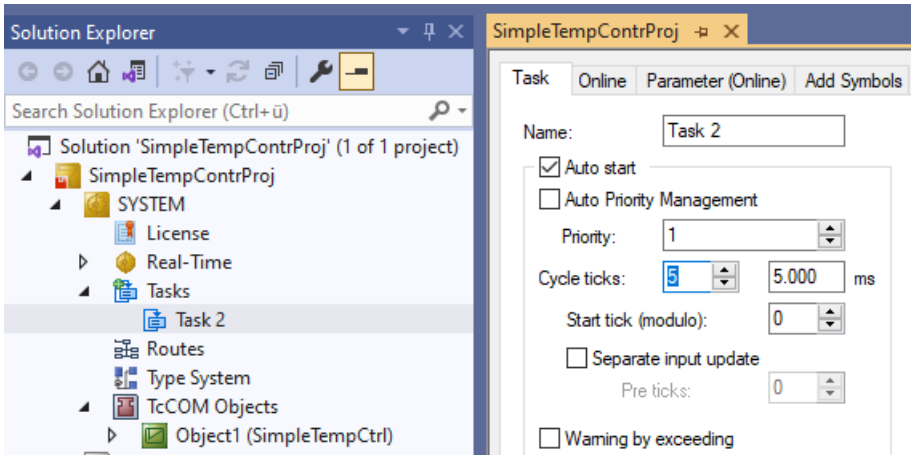
1. Öffnen Sie TwinCAT (TwinCAT XAE oder TwinCAT in einer Visual-Studio-Umgebung).
2. Instanziieren Sie ein neues TcCOM-Objekt.



3. Wählen Sie das gewünschte Objekt.

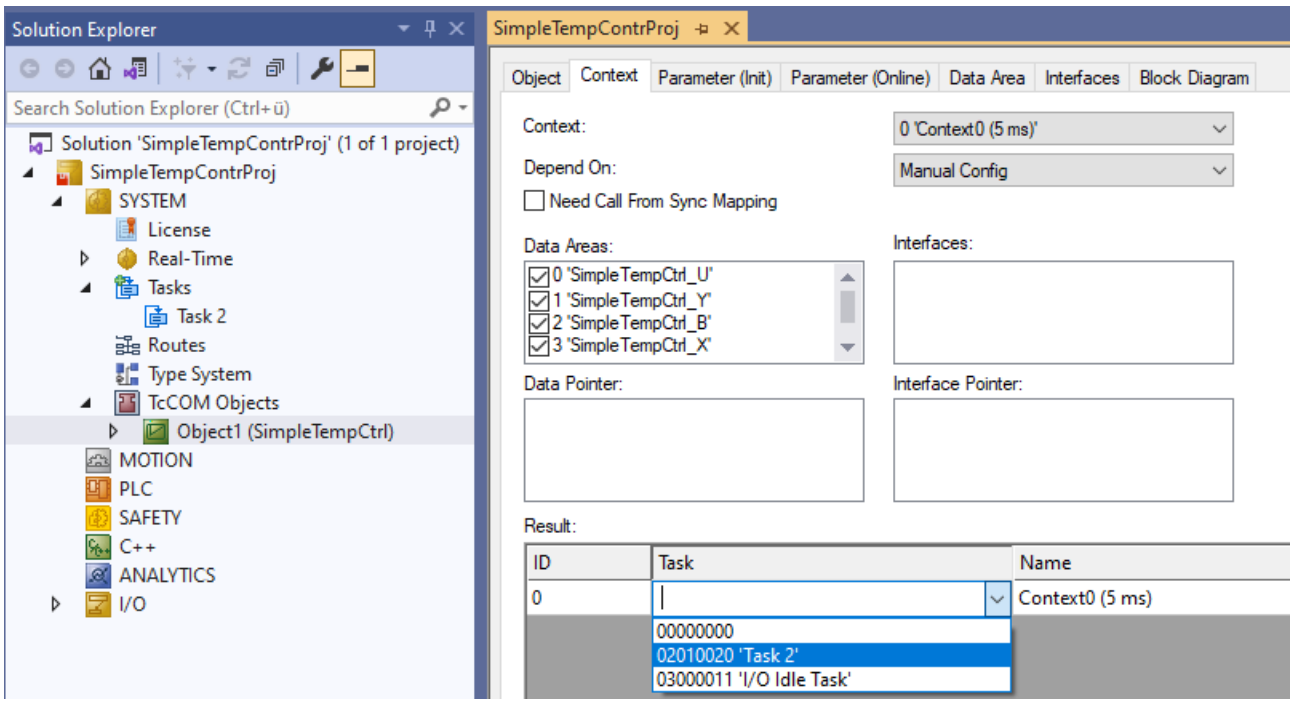


4. Erstellen Sie eine zyklische Task.

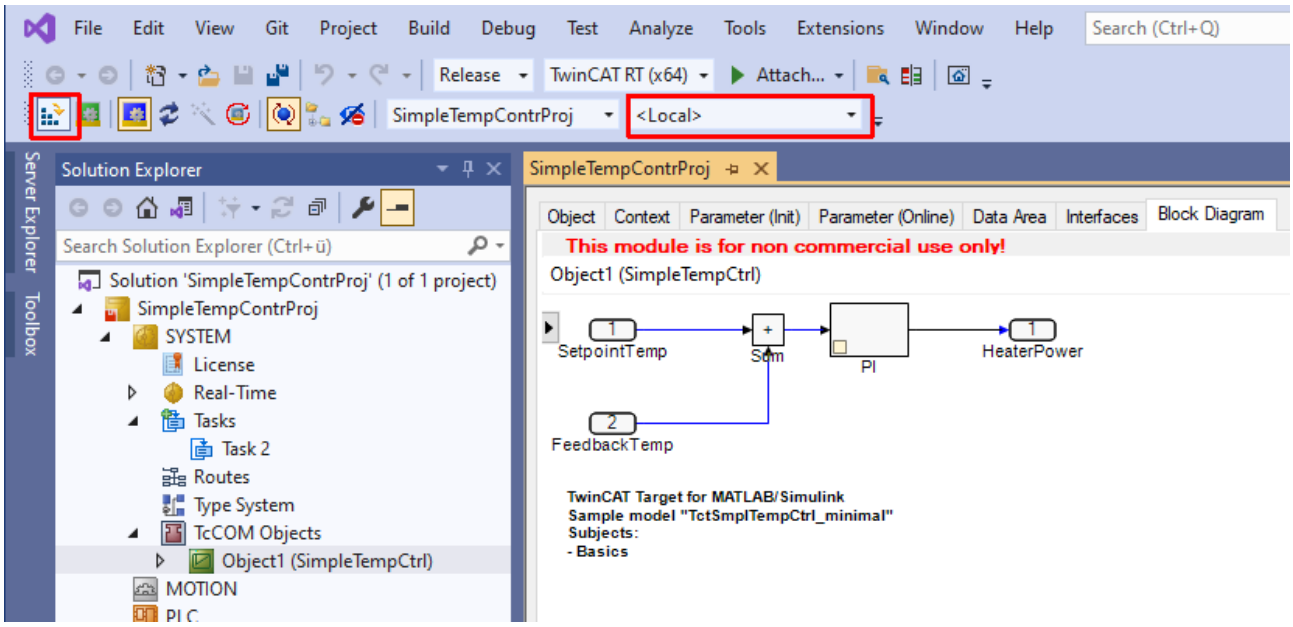


5. Weisen Sie die erstelle Task ihrer TcCOM-Instanz zu.

Beachten Sie, dass die Zykluszeit der Task und die SampleTime in Simulink® (hier 5 ms) passen.



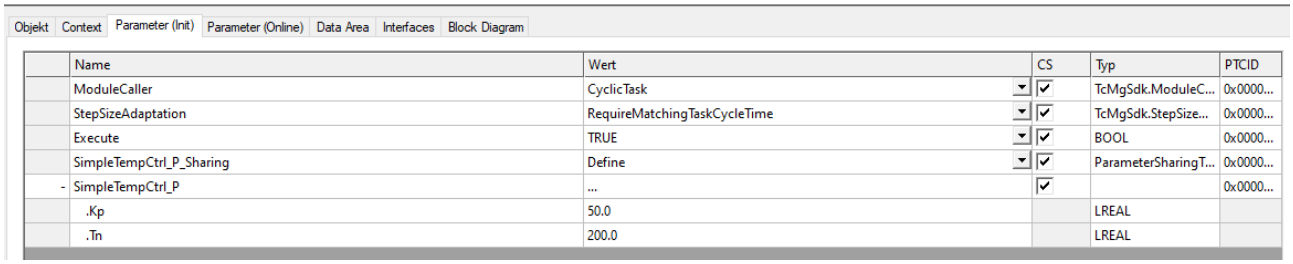
6. Aktivieren Sie die Konfiguration.



TcCOM-Instanz konfigurieren und verknüpfen

Der Datenaustausch der TcCOM-Instanz erfolgt über Mappings des Prozessabbilds. Simulink®-Inputs und Simulink®-Outputs werden automatisch als Inputs bzw. Outputs im Prozessabbild abgebildet und können mit I/O oder anderen Objekten verknüpft werden.

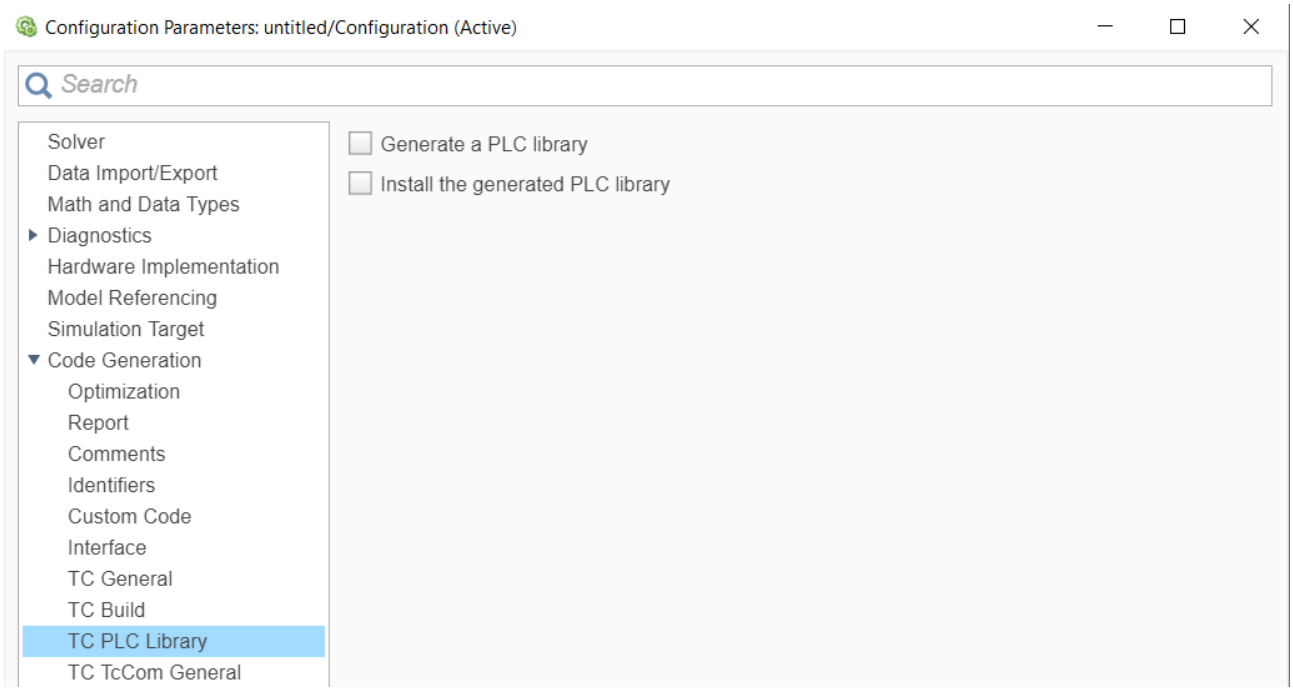
Im Bereich Parameter (Init) der TcCOM-Instanz können Sie die Instanz optional anders konfigurieren als bei der Erstellung aus Simulink® angegeben.



Beispielsweise können Sie hier den Parameter Kp auf „52“ setzen. Das TcCOM-Modul würde dann diesen Wert als Startup-Wert für diese Instanz nutzen.

Einfügen als SPS-Funktionsbaustein

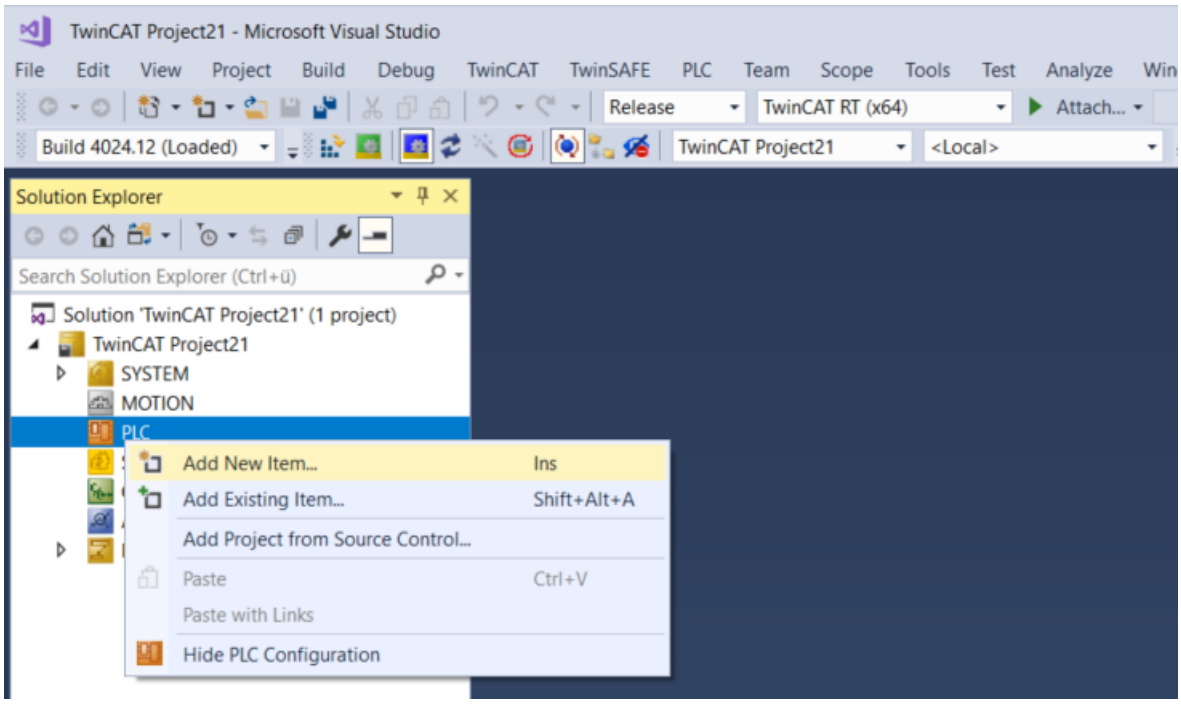
Die im Folgenden genutzte SPS-Bibliothek ist nur dann vorhanden, wenn folgende Parameter in Simulink® gesetzt sind:



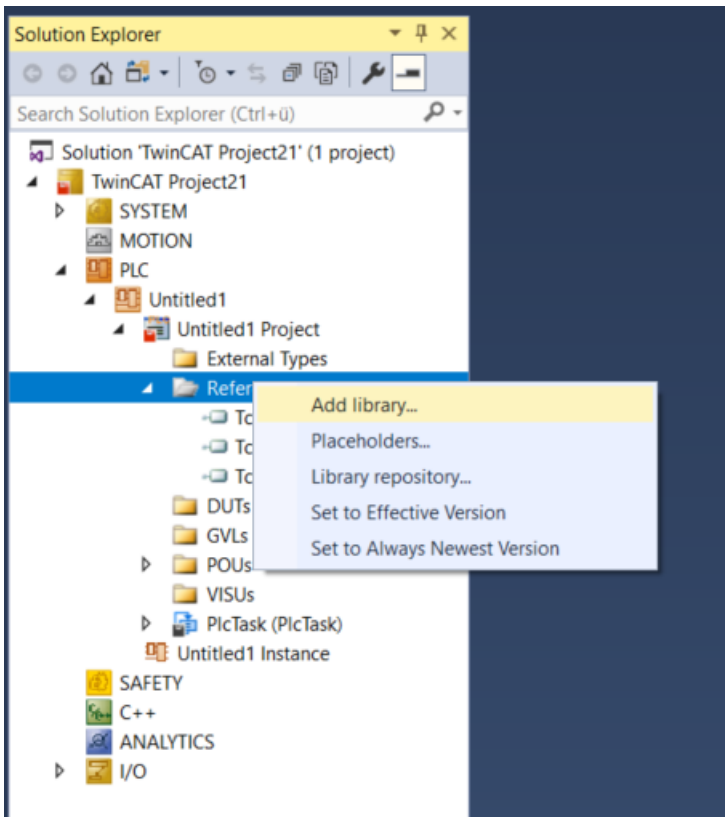
Sollten diese Optionen nicht gesetzt worden sein, kann dies nachträglich ohne Einsatz von Simulink® und dem TwinCAT Target for Simulink® erfolgen. Siehe dazu SPS-Bibliothek erzeugen und installieren [▶ 211].

Kurzübersicht Handlungsschritte SPS-Bibliothek/Funktionsbaustein

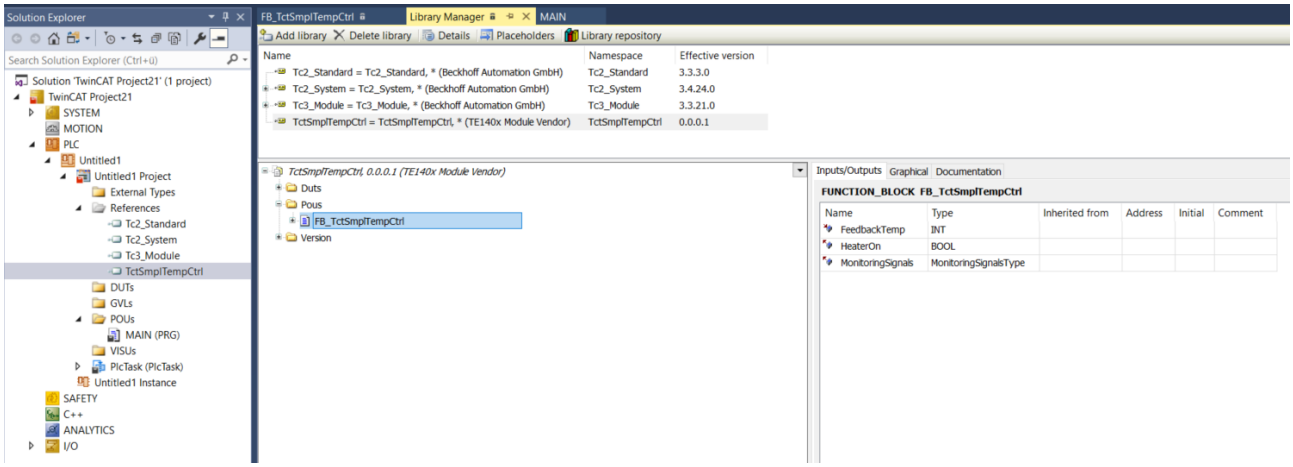
- SPS in TwinCAT erstellen:



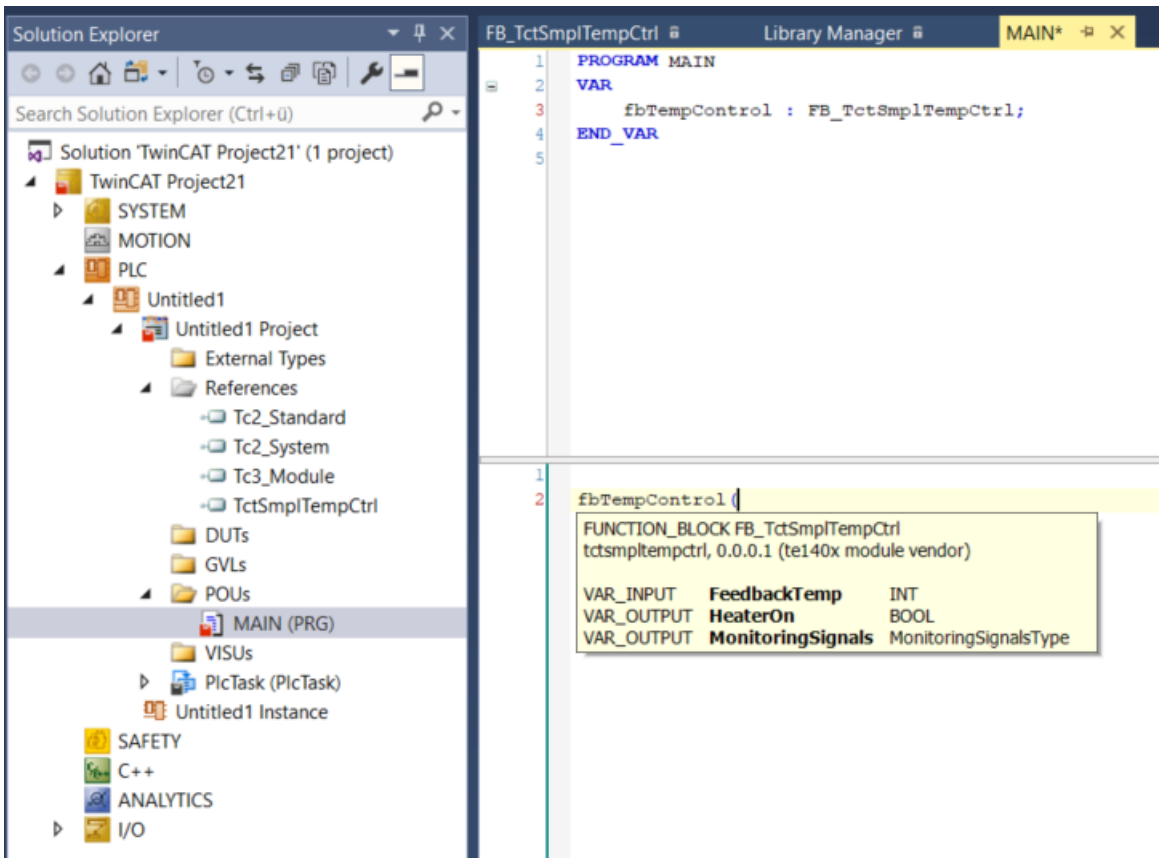
- SPS Bibliothek hinzufügen:



- SPS Bibliothek selektieren und Inhalt betrachten:



- Funktionsbaustein aus der Bibliothek in der SPS verwenden:

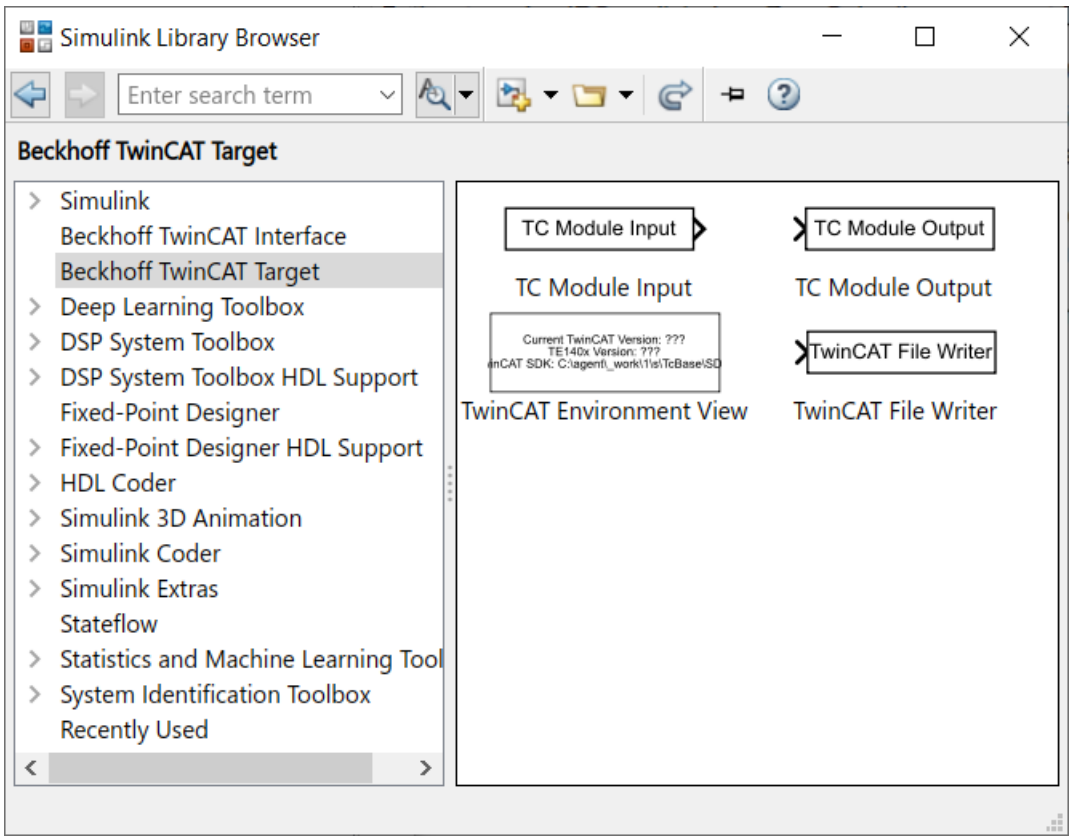


Aufruf eines TcCOM-Objekts auch aus der SPS möglich

Zusätzlich zu der hier beschriebenen Variante können Sie eine Instanz eines TcCOM auch aus der SPS heraus aufrufen. Siehe Anwenden des TcCOM-Wrapper-FB [▶ 214].

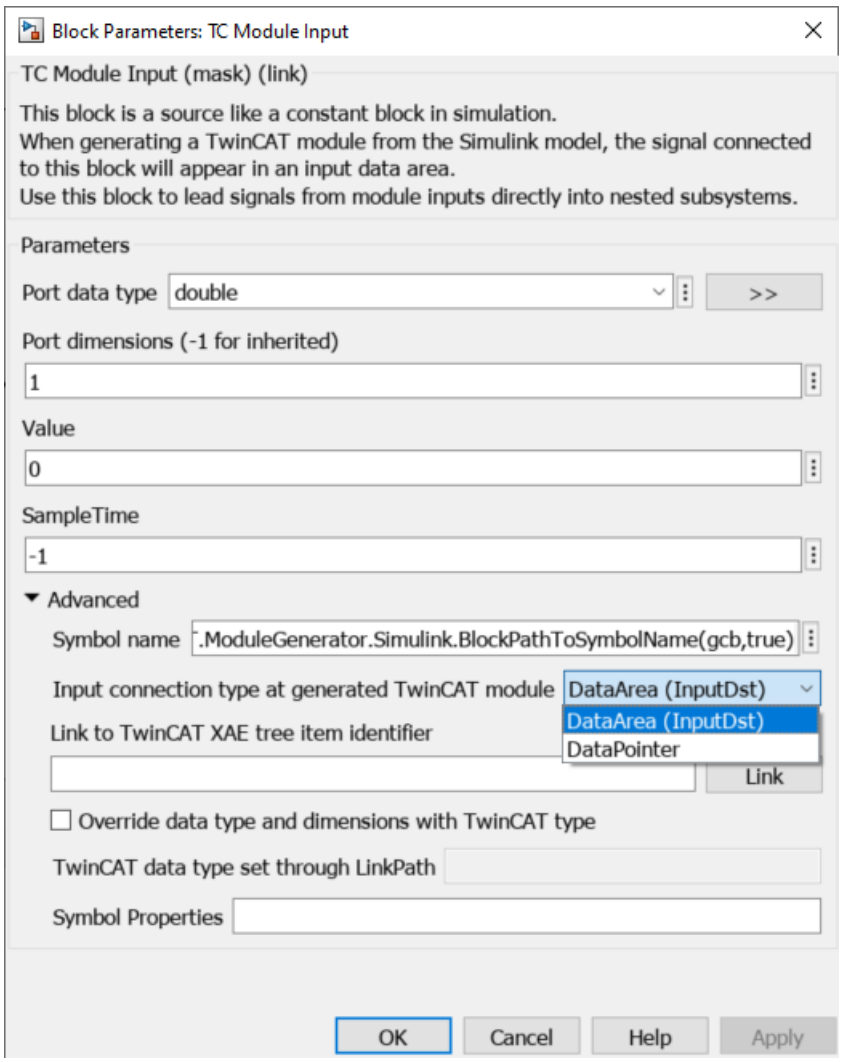
4.5 TwinCAT Library in Simulink®

Im Bereich **Library Browser** > **Beckhoff TwinCAT Target** befinden sich spezifische Blöcke für das TwinCAT Target for Simulink®.



4.5.1 TwinCAT Module Input und Output

In Simulink® können optional *TwinCAT-spezifische* Ein- und Ausgangsblöcke verwendet werden. Ein ebenfalls gültiger Weg ist es, die Standard Input Ports (In) und Output Ports (Out) von Simulink® zu verwenden. Dies ist in der Regel auch der *best practice* Weg, es sei denn, es werden die unten beschriebenen Zusatzfunktionen der TwinCAT-In- und -Output-Module benötigt.



Zusatzfunktionen der TC Module Blöcke

Wenn Sie die von Beckhoff bereitgestellten Input- (TC Module Input) und Output-Blöcke (TC Module Output) verwenden, erhalten Sie folgende zusätzliche Funktionalitäten gegenüber den Standard Simulink®-Input und -Output Ports:

- Sie können Signale und Busse auch aus **Subsystemen direkt als Input oder Output für das TcCOM definieren**, ohne die Signale/Busse zuerst aus dem Subsystem in das oberste System zu führen.
  Beispiel: Subsystem-Eingänge und -Ausgänge [▶ 115].
- Sie können optional in den Blockparametern ein automatisches Mapping zu anderen TcCOM oder I/Os hinterlegen, sodass direkt bei der Instanziierung des TcCOM das **Mapping automatisiert** ausgeführt wird, siehe Automatisches Mapping [▶ 115].
- Sie können den *connection type* jedes TC Module Inputs/Outputs individuell -zwischen Mapping und **DataPointer-** wählen. Sie können z. B. alle Standard Input Ports per Mapping zugänglich machen und andere über die TC Module Inputs per DataPointer erreichbar machen, siehe DataArea oder DataPointer [▶ 117].
  Beispiel: Geteilter Speicher zwischen TcCOM-Instanzen [▶ 154].
- Sie können den **Symbol Name** beeinflussen, d.h., zum Beispiel den Namen und die Hierarchie des Eingangs oder Ausgangs im Prozessabbild verändern, siehe Symbol Name [▶ 118].
- Sie können Signale und Busse mit spezifischen **Symbol Properties** ausstatten, siehe Symbol properties [▶ 118].
- Sie können **Initialwerte** für Eingänge verwenden. Setzen Sie dazu den Wert Value der TC Module Inputs auf einen beliebigen Wert, siehe Initialwerte [▶ 119].

Initialwerte können auch bei Standard Input Ports realisiert werden, siehe Option Input: Initial values unter TC TcCom Interfaces [▶ 147].

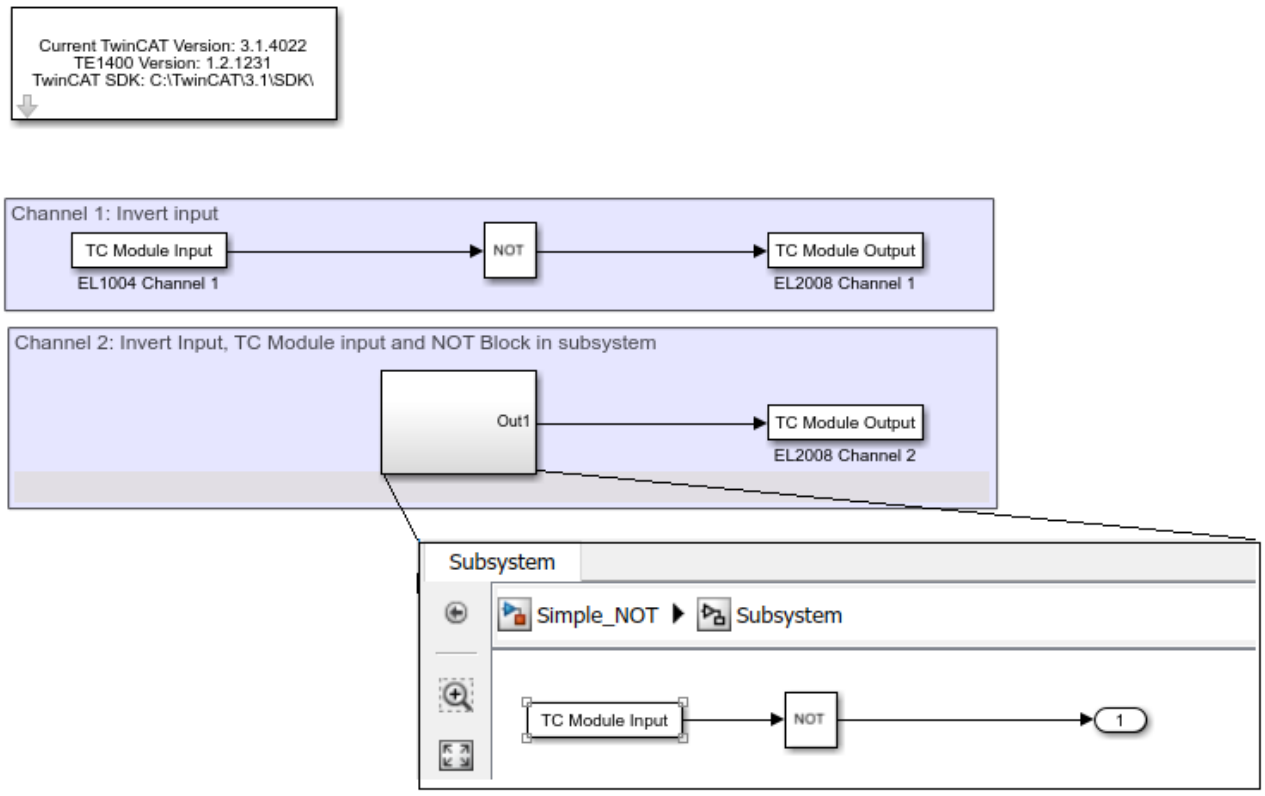
Bei der Nutzung des automatischen Mappings ist zu beachten, dass Sie bei mehrfacher Instanziierung des TcCOM in TwinCAT einen Mapping-Konflikt erhalten, den Sie durch händisches Mappen wieder auflösen müssen. Entsprechend ist bei Mehrfachinstanziierungen diese Option nicht zu empfehlen.

4.5.1.1 Subsystem-Eingänge und -Ausgänge

Erstellen eines TcCOM Eingangs/Ausgangs aus einem Simulink®-Subsystem

Ein Simulink®-Modell wird erstellt, welches zwei Eingänge negiert wieder ausgibt. Ein Eingang wird dabei in einem Subsystem platziert, siehe Abbildung unten.

Gemäß der beschriebenen Eigenschaft der TC Module-Input-/Output-Blöcke, wird auch der TC Module Input im Simulink®-Subsystem in TwinCAT in das Prozessabbild des TcCOM aufgenommen. Es muss also nicht, wie bei Standard-In- oder Out-Ports auf die oberste Ebene des Simulink®-Modells geroutet werden.



4.5.1.2 Automatisches Mapping

Die Ein- und Ausgänge des Simulink®-Modells sollen im Folgenden automatisch auf digitale Ein- und Ausgänge gemappt werden. Das bedeutet, dass nach Instanziierung des TcCOM in einem TwinCAT-Projekt die Mappings automatisiert erstellt werden. Ein händisches Verknüpfen ist nicht mehr notwendig.

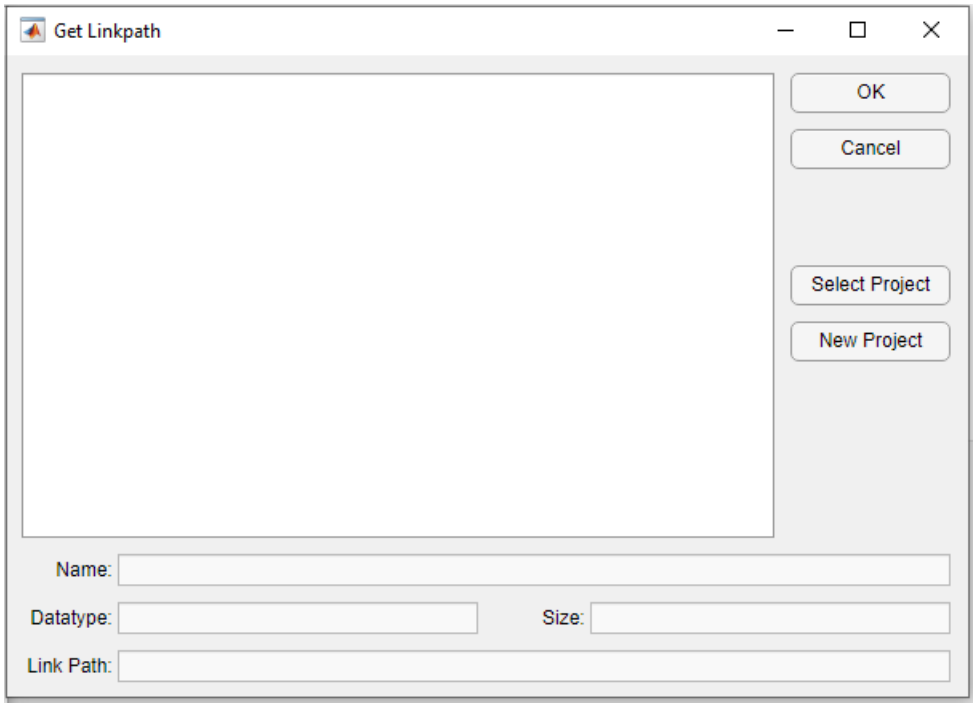
Navigieren Sie in den Block Parameter des Tc Module In oder Tc Module Out unter **Advanced** auf den Parameter **Link to TwinCAT XAE tree item identifier**. Sie können den Tree Item Identifier und den Datentyp entweder von Hand eintragen, oder aus Simulink® heraus über einen Browser selektieren.

Durch Anwählen des Buttons **Link** öffnet sich ein neuer Dialog. Sie können nun ein bestehendes TwinCAT-Projekt laden (**Select Project**) und die darin bestehenden Inputs bzw. Outputs browsen oder Sie erstellen ein neues Projekt (**New Project**) und können in dem neuen Projekt automatisch den EtherCAT Feldbus scannen, um dann mit den erkannten I/Os zu verknüpfen.

✓ Neues Projekt erstellen und Inputs bzw. Outputs des Targets anzeigen lassen und selektieren:

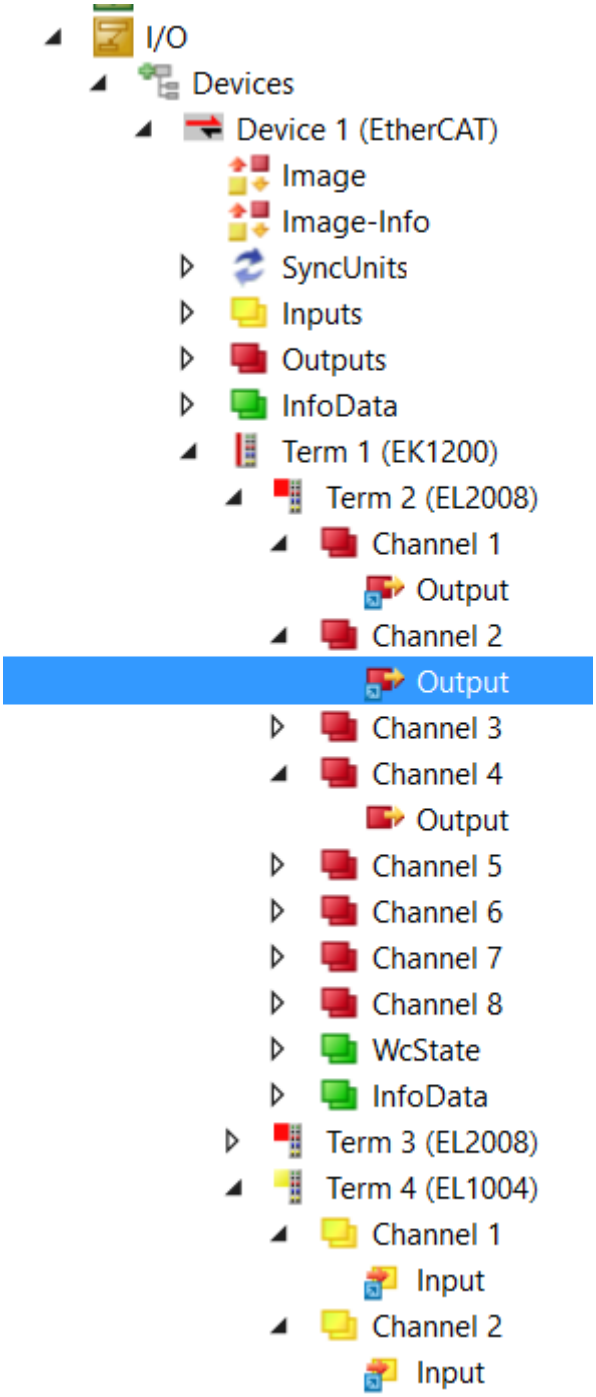
1. Erstellen Sie ein neues Projekt.
2. Wählen Sie dafür einen Speicherpfad des neu zu erstellenden TwinCAT-Projekts aus.
3. Wählen Sie das Target aus, auf welches Sie das Projekt herunterladen möchten.
4. Scannen Sie automatisch den I/O Tree des Targets.

⇨ Dadurch werden Ihnen alle Inputs bzw. Outputs des Targets angezeigt und können selektiert werden.



Durch Anwahl des Inputs oder Outputs, den sie verlinken möchten, wird der Tree Item Identifier automatisch gesetzt und der passende Datentyp automatisch in Simulink® gesetzt.

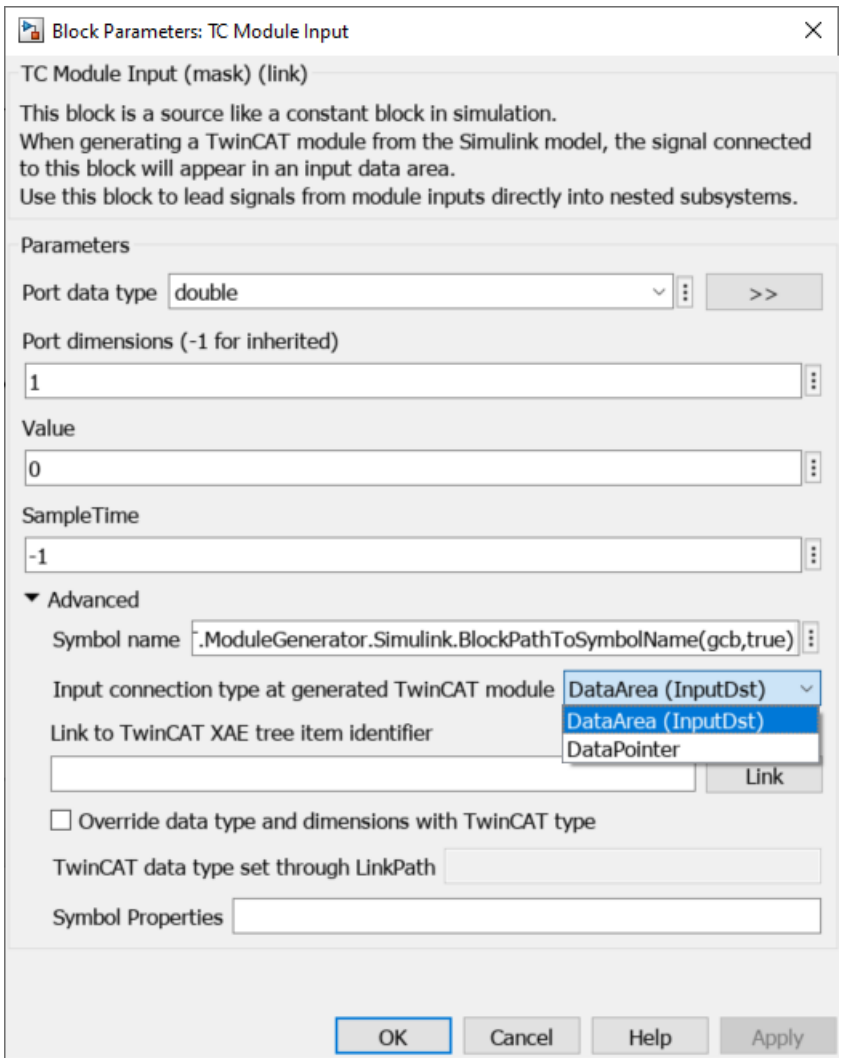
Wenn das oben beschriebene Simulink®-Modell in ein TcCOM übersetzt und in einer TwinCAT 3 Solution eingebunden wird, wird automatisch ein Mapping zu den in Simulink® gewählten Inputs und Outputs hergestellt.
Farbgebung der Symbole zur Unterscheidung:
Die automatisch generierten Mappings werden mit einem *blauen* Symbol versehen, während händische Mapping-Symbole *weiß* dargestellt werden.



4.5.1.3 DataArea oder DataPointer

Standardmäßig ist als **Input connection type at generated TwinCAT module** „DataArea (InputDst)“ ausgewählt. Das bedeutet, dass ein TC Module Input als *Input Destination DataArea* und ein TC Module Output entsprechend als *Output Source DataArea* angelegt werden.

Alternativ kann hier auch „DataPointer“ ausgewählt werden, siehe dazu Geteilter Speicher zwischen TcCOM-Instanzen [▶ 154].



Wenn das oben aufgezeigte Simulink®-Modell mit dem **Target for Simulink®** erstellt wird, so erscheinen im Prozessabbild auf der TcCOM Instanz 2 Eingänge und 2 Ausgänge, wenn als **Input connection type at generated TcCOM module** „mapping“ ausgewählt wird.

4.5.1.4 Symbol properties

Sie können für die TC Module In-/Out-Blöcke spezifische Symbol properties definieren, vgl. Symbol Properties und Attribut-Pragmas [▶ 171].

Beispiel: `OPC.UA.DA.=1`

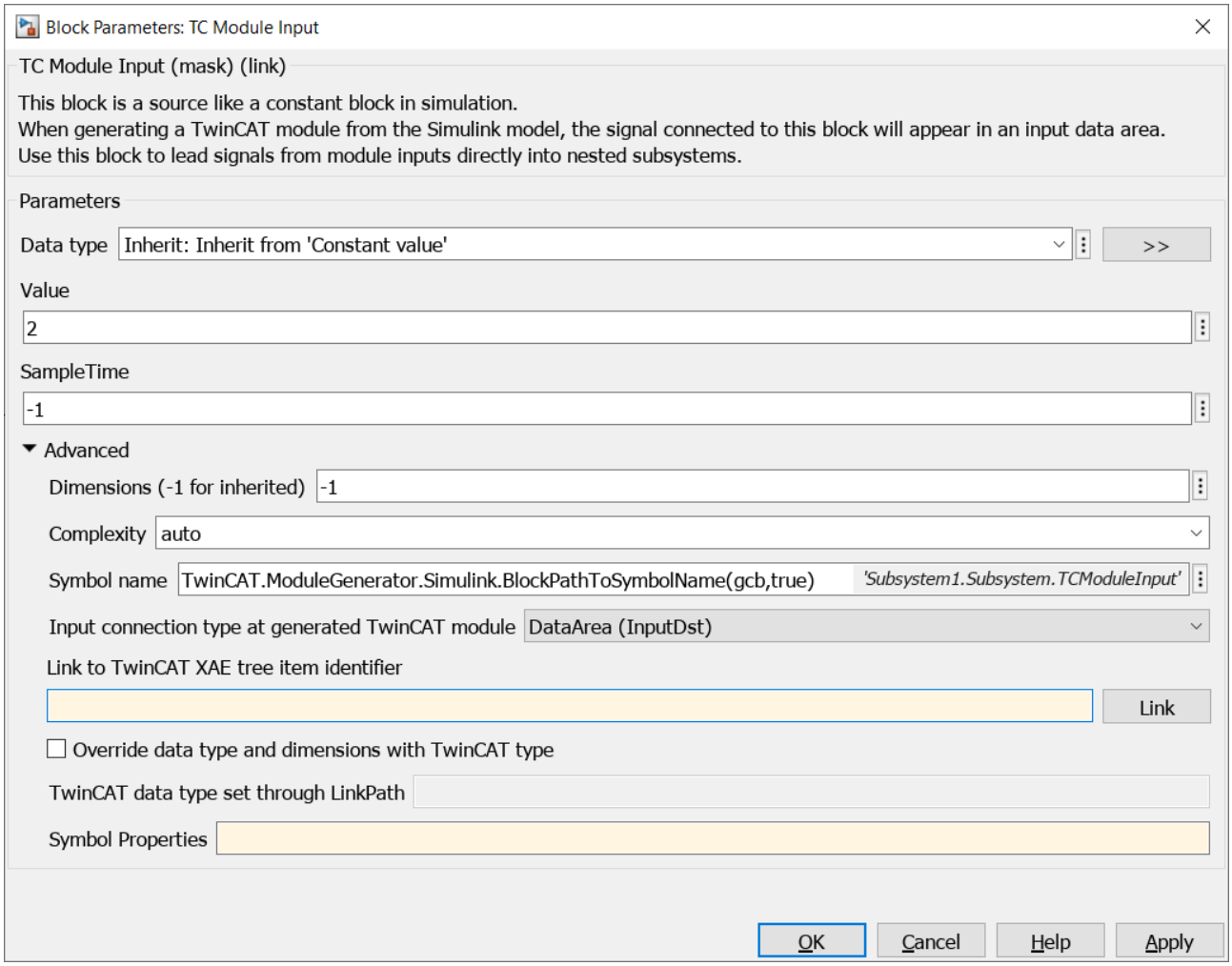
Einschränkung

Der Symbol name darf bei Nutzung von Symbol properties nicht verschachtelt sein.

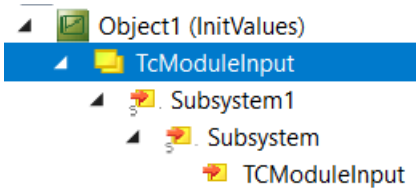
4.5.1.5 Symbol Name

Sie können unter *Symbol name* den Namen und damit die Darstellung des Symbols im Prozessabbild beeinflussen.

Im Folgenden Beispiel wird ein TC Module Input in zwei Subsystemen verschachtelt. Die Default-Namensauflösung ist (wie rechts neben dem Symbol name zu sehen) <Subsystem>.<Subsystem>.<TcModuleName>.



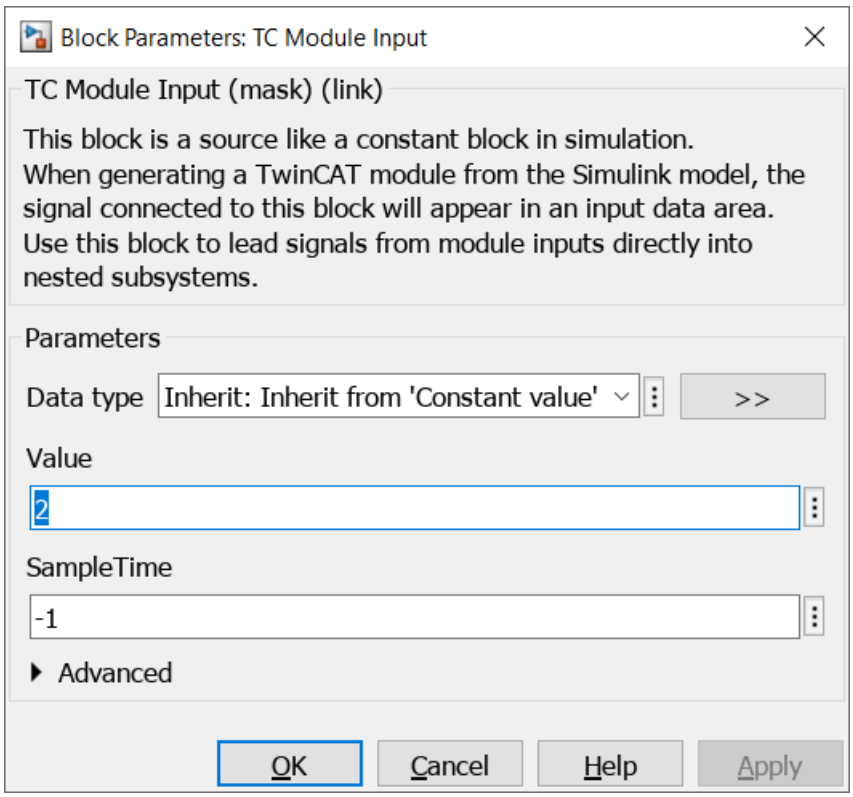
Das Prozessabbild sieht entsprechend wie folgt aus:



Insbesondere in Fällen, bei denen es zu sehr tiefen Verschachtelungen kommt, ist ggf. eine Kürzung des Symbol-Namens wünschenswert. Sie haben daher die Möglichkeit, den Symbolnamen händisch als Text einzutragen, bspw. `'Subsystem.InputValue'`.

4.5.1.6 Initialwerte

Neben Datentyp, Dimension und SampleTime ist zusätzlich zum Standard-Simulink®-In- und -Out-Port ein optionales **Value** Feld editierbar. Über dieses Feld können Sie einen Initialwert für den Eingang (nur für TC Module Input verfügbar) angeben. Wird der Eingang in TwinCAT nicht mit einem entsprechenden Output verknüpft, wird der hier eingetragene Wert als Eingangswert genutzt.



4.5.2 TwinCAT Environment View

Ziehen Sie den TwinCAT Environment View in Ihre Simulink®-Umgebung, um, zum Beispiel für Supportfälle, direkt die verfügbare TwinCAT-XAE-Version und die aktuell installierte TE1400-Version angezeigt zu bekommen.

4.5.3 TwinCAT File Writer

Der TwinCAT-File-Writer-Block schreibt .mat-Dateien aus der TwinCAT-Umgebung heraus. Der Block erfüllt diese Funktion nur, wenn das Simulink®-Modell bereits in ein TcCOM oder FB überführt wurde und in einer TwinCAT Runtime ausgeführt wird. Wenn das Simulink®-Modell in Simulink® ausgeführt wird, hat dieser Block keine Funktion.

Parameter	Beschreibung	Anmerkung
Port data type	Datentyp des eingehenden Signals	Unterstützt werden: • Integer Types • float • double • boolean • enums • bus objects
Port dimension	Dimensionalität des eingehenden Signals	-1 -> Inherit Ansonsten bspw. [1,2], [1,5], …
SampleTime	Block Sample Time in Sekunden	-1 -> Inherit

Parameter	Beschreibung	Anmerkung
file name	Dateiname der .mat-Datei	Fullpath oder relativer Pfad möglich. Relativer Pfad relativ zu *TwinCAT\3.1\Boot*.
Maximum file byte size	Maximale Größe (in Bytes) der .mat-Datei. Wird diese Größe erreicht, wird die aktuelle Datei abgeschlossen und eine neue begonnen.	0 -> Dateiformatbedingtes Maximum
Maximum file count	Maximale Anzahl der zu schreibenden .mat-Dateien.	0 -> Unendlich viele Dateien. Wird das Maximum erreicht, werden alte Dateien überschrieben, beginnend mit `_part0.mat`.
Pause writing files	Pausiert das Schreiben	Parameter am TcCOM
Write simulation time with data	Schreibt pro Datum eine Struktur mit 2 Feldern, „time“ und „data“.	
Expose Pause as block input	Erzeugt einen Inport, über den der TwinCAT File Writer pausiert werden kann.	

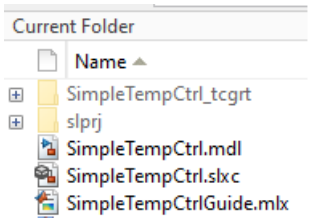
4.6 Übersicht zu automatisch generierten Dateien

Wenn ein Build-Prozess angestoßen wird, werden einige Dateien und Ordner automatisch erzeugt. „Wo liegen die Dateien?“, „Was kann damit gemacht werden?“ und „Was bedeuten die Dateien?“ Diese Fragen werden im Folgenden beantwortet.

Welche Kategorien an automatisch generierten Dateien gibt es?

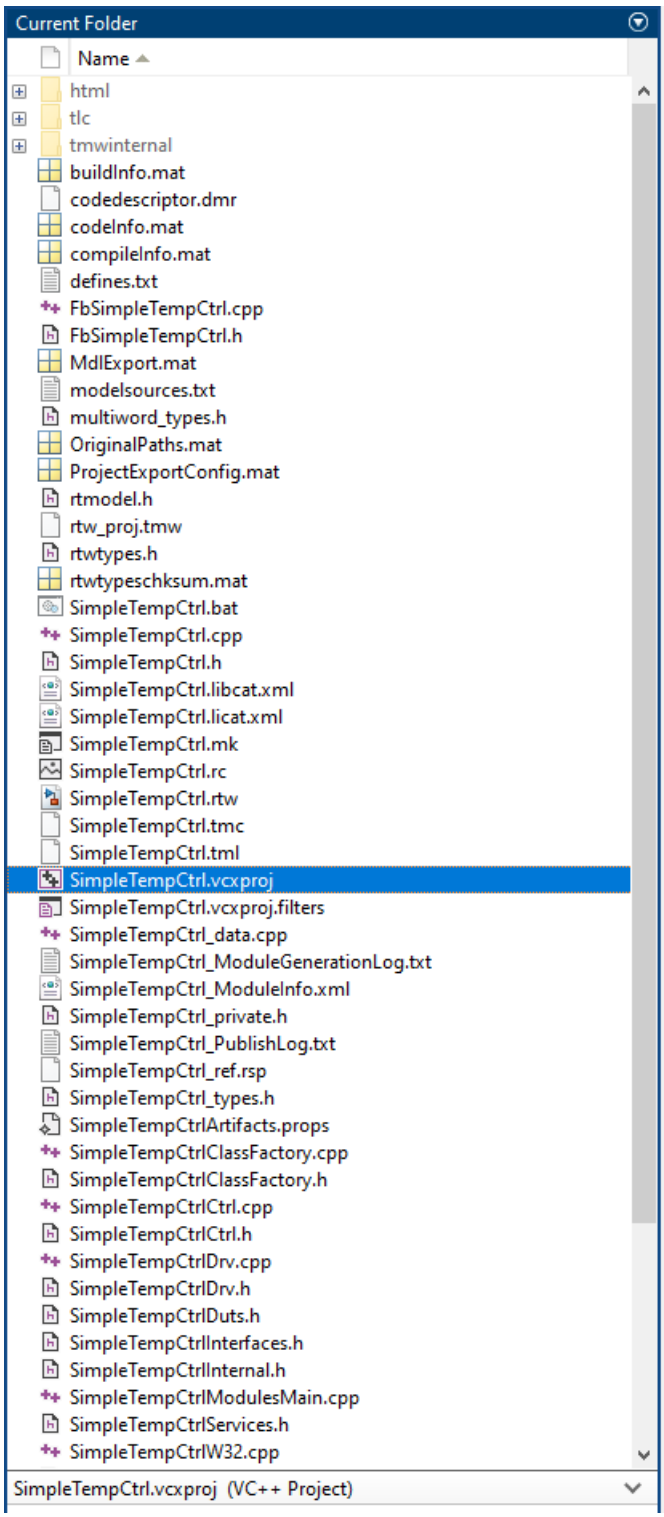
- Es wird Source Code generiert [▶ 121].
- Es werden Log-Files generiert [▶ 123].
- Es werden die TwinCAT-Objekte, Treiber (*.tmx) und Beschreibungsdateien (*.tmc, *.library, ...), erzeugt [▶ 124].

Alle vom TwinCAT-Target erstellten Dateien werden im aktuellen MATLAB®-path im Ordner *<SimulinkModelName>_tcgrt* zusammengefasst. Der Ordner liegt neben dem von MathWorks® generiert slprj-Ordner.



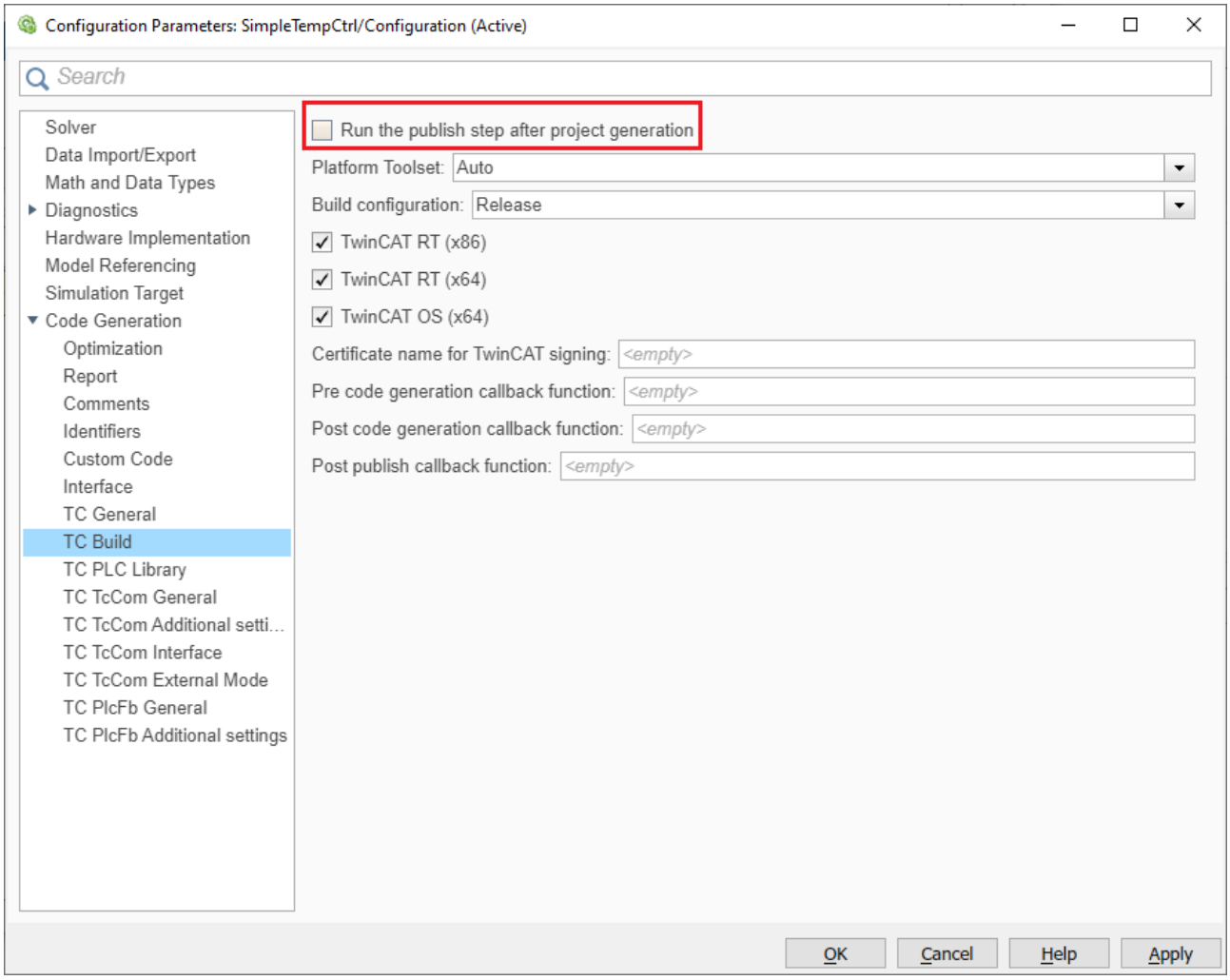
Generierter Source Code

Zentrale Datei für den Source Code ist *<SimulinkModelName>.vcxproj* und liegt im <SimulinkModelName>_tcgrt Ordner. Die Datei öffnet ein TwinCAT C++ Projekt, welches zum Inspizieren des generierten Quellcodes, zum nachträglichen Bauen von TwinCAT-Objekten oder auch zum Debuggen in TwinCAT [▶ 220] genutzt werden kann.



Hinsichtlich der Build-Möglichkeit aus dem generierten TwinCAT C++ Projekt heraus, ist wissenswert, dass Sie den Publish-Step, d.h. das Bauen für die konfigurierten Plattformen, in Simulink® ausschalten können.

Sie können in Simulink® die Code-Generierung ohne Build durch Abwählen von *Run the publish step after project generation* erreichen. Der *publish step* beinhaltet den *Build* der TwinCAT-Objekte für die ausgewählten Plattformen (TwinCAT RT x86, x64 …).

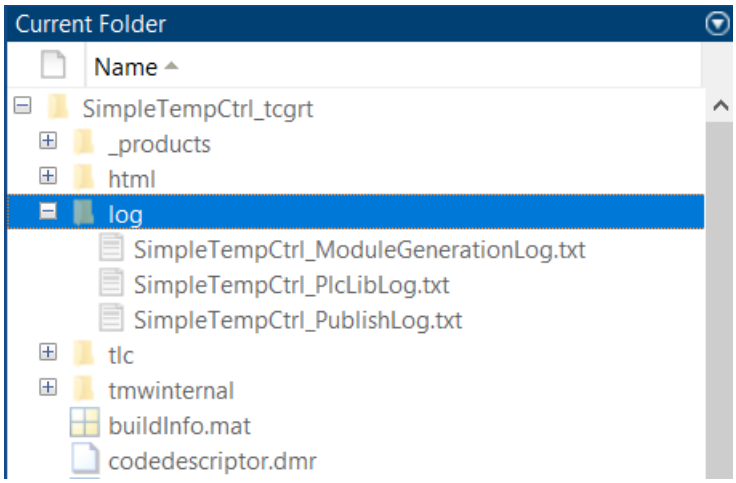


Weitere nützliche Informationen hierzu:

- Pack-and-go mit dem Target for Simulink®
- Continuous Integration

Generierte Logfiles

Die generierten Logfiles werden im Unterordner *log* zusammengefasst. Die erstellen Logfiles sind die erste Anlaufstelle beim Debugging.



In diesem Ordner befinden sich bis zu drei Logfiles. Hauptanlaufstelle ist die Zusammenfassung alles Logs in der Datei *<ModelName>_ModuleGenerationLog.txt*.

Beckhoff Support benötigt Logfile

Wenn Sie Hilfe bei unserem Support in Anspruch nehmen, senden Sie bitte mindestens folgende Datei aus dem log-Ordner mit: *<ModelName>_ModuleGenerationLog.txt*

Der Aufbau des ModuleGenerationLog ist in mehrere Segmente unterteilt, die unterschiedliche Schritte des TwinCAT-Targets repräsentieren. Die Schritte werden jeweils mit ### im Log angezeigt.

Beispiel:

```
2023-10-18 14:48:37: ### Starting build procedure for: SimpleTempCtrl
2023-10-18 14:48:43: ### Save TLC export
2023-10-18 14:48:47: ### Export block diagram
2023-10-18 14:48:47: Block diagram export succeeded
2023-10-18 14:48:48: ### Export TwinCAT C++ project
2023-10-18 14:48:53: ### Save project
2023-10-18 14:48:53: The TwinCAT C++ project "C:
\Users\xyz\Documents\MATLAB\TE14xxSamples\2023-10-18_13-58_SimpleTemperatureController\SimpleTempCtr
l_tcgrt\SimpleTempCtrl.vcxproj" was generated successfully
```

Treten keine Warnungen oder Fehler in den Schritten auf, werden in der Regel keine Einträge für die durchgeführten Schritte vorgenommen. In manchen fällen wird auf erstellte Dateien explizit hingewiesen, wie im obigen Beispiel die erstellte vcxproj-Datei. Es wird für detailliertere Informationen auch auf weitere Logfiles verwiesen, siehe folgendes Beispiel:

```
2023-10-18 14:48:53: ### Publish TMX
2023-10-18 14:48:53: Configuration: "Release"
2023-10-18 14:48:53: Platform(s): "TwinCAT RT (x86);TwinCAT RT (x64);TwinCAT OS (x64)"
2023-10-18 14:48:53: TwinCAT SDK: "C:\TwinCAT\3.1\SDK\" (Version 3.1.4024.50)
2023-10-18 14:48:53: Platform Toolset: V142 (Automatically selected)
2023-10-18 14:48:53: Microsoft (R) Build Engine version 16.11.2+f32259642 for .NET Framework
2023-10-18 14:48:53: Copyright (C) Microsoft Corporation. All rights reserved.
2023-10-18 14:49:06: Publish procedure completed successfully
2023-10-18 14:49:06: See log file "C:
\Users\xyz\Documents\MATLAB\TE14xxSamples\2023-10-18_13-58_SimpleTemperatureController\SimpleTempCtr
l_tcgrt\log\SimpleTempCtrl_PublishLog.txt" for details
```

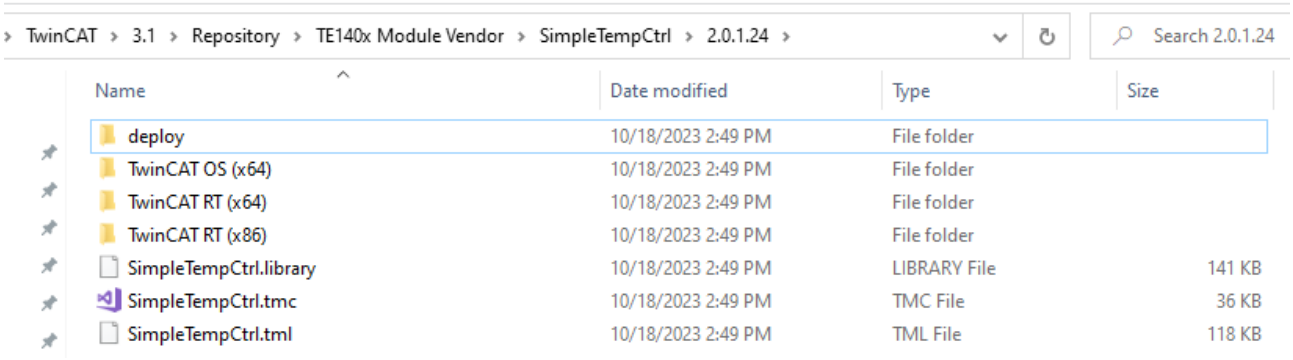
Treten Warnungen oder Fehler auf, werden diese im ModuleGenerationLog angezeigt. Die detaillierten Logs beinhalten hingegen alle Ausgaben des durchgeführten Schritts. Beispielsweise sind hier Warnungen im Bereich der Signatur-Prüfung der erstellen tmx-Dateien zu sehen:

```
2023-10-18 14:49:06: ### Publish summary
2023-10-18 14:49:06: Configuration: "Release"
2023-10-18 14:49:06: Platform(s): "TwinCAT RT (x86);TwinCAT RT (x64);TwinCAT OS (x64)"
2023-10-18 14:49:06: TwinCAT SDK: "C:\TwinCAT\3.1\SDK\" (Version 3.1.4024.50)
2023-10-18 14:49:06: Platform Toolset: V142 (Automatically selected)
2023-10-18 14:49:06: Vendor name: TE140x Module Vendor
2023-10-18 14:49:06: Library name: SimpleTempCtrl
2023-10-18 14:49:06: Library version: 2.0.1.24
2023-10-18 14:49:06: Local installation folder: "C:\TwinCAT\3.1\Repository\TE140x Module
Vendor\SimpleTempCtrl\2.0.1.24"
2023-10-18 14:49:06: TMX archive: -
2023-10-18 14:49:06: Signatures:
2023-10-18 14:49:06: File 'C:\TwinCAT\3.1\Repository\TE140x Module
Vendor\SimpleTempCtrl\2.0.1.24\TwinCAT RT (x86)\SimpleTempCtrl.tmx' has signature.
2023-10-18 14:49:06: issuer TestSign123 (x.yz@beckhoff.com), certificate expires on 08/15/2025
2023-10-18 14:49:06: Warning: Signature found, but OEM certificate was not signed by Beckhoff.
Driver can only be used in test mode.
2023-10-18 14:49:06: File 'C:\TwinCAT\3.1\Repository\TE140x Module
Vendor\SimpleTempCtrl\2.0.1.24\TwinCAT RT (x64)\SimpleTempCtrl.tmx' has signature.
2023-10-18 14:49:06: issuer TestSign123 (x.yz@beckhoff.com), certificate expires on 08/15/2025
2023-10-18 14:49:06: Warning: Signature found, but OEM certificate was not signed by Beckhoff.
Driver can only be used in test mode.
2023-10-18 14:49:06: File 'C:\TwinCAT\3.1\Repository\TE140x Module
Vendor\SimpleTempCtrl\2.0.1.24\TwinCAT OS (x64)\SimpleTempCtrl.tmx' has signature.
2023-10-18 14:49:06: issuer TestSign123 (x.yz@beckhoff.com), certificate expires on 08/15/2025
```

Erstellte TwinCAT-Objekte

Die erstellten Binär-Dateien und Beschreibungsdateien, welche zur weiteren Nutzung im TwinCAT XAE verwendet werden können, liegen nach erfolgreichem *Build* im sogenannten *Engineering Repository*, d. h. auf dem Engineering PC unter:

%TwinCATInstallDir% \3.1\Repository\<Vendor>\<ModelName>\<Version>



- <ModelName>.tmc: Beschreibungsdatei des TcCOM-Objekts (Objektklasse)
- <ModelName>.tml: Library Datei: Kann wie .library file genutzt werden.
- <ModelName>.library: TwinCAT PLC library file
- Deploy\<ModelName>_ModuleInfo.xml: Modulinformationen, z. B. Blockdiagram
- <Plattform>\<ModelName>.tmx: Treiber-Datei

Verteilung von TwinCAT-Objekten auf andere XAE-Systeme: Wenn der Ordner auf Ebene <ModelName> auf andere PCs mit TwinCAT XAE in die lokalen *Engineering Repositories* kopiert wird, können deren Nutzer die erstellten TwinCAT-Objekte in ihren TwinCAT Solutions verwenden.

Vergleiche dazu auch Teilen von erstellten TwinCAT-Objekten [▶ 139].

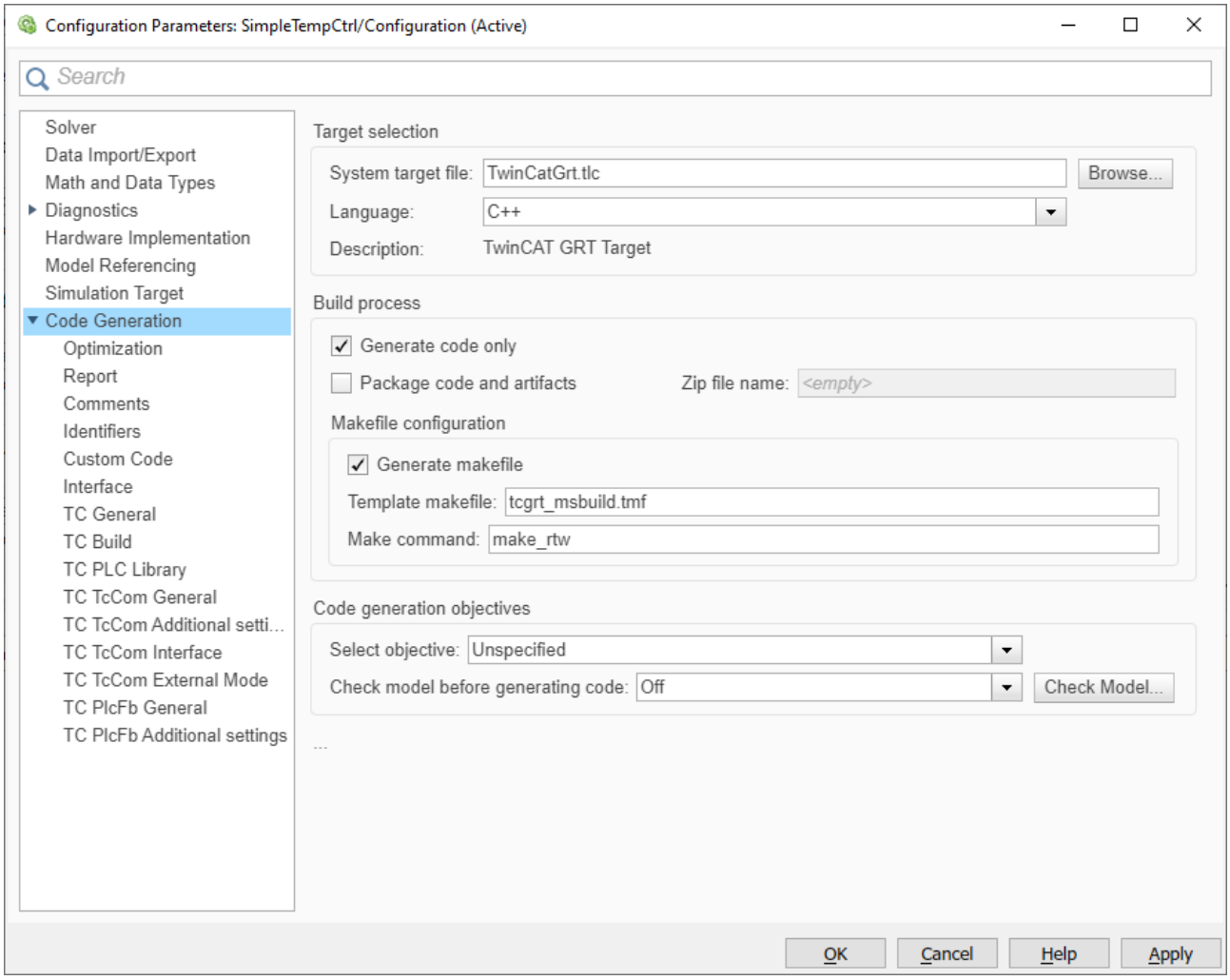
Weitere Hinweise

Beschreibung der generieten C++ Dateien und Binärdateien

Versionierte C++ Projekte

4.7 Parametrierung der Code-Generierung in Simulink®

Innerhalb von Simulink® kann eine Vielzahl von Einstellungen zur Konfiguration der zu generierenden TwinCAT-Objekte (TcCOM und Funktionsbausteine) vorgenommen werden. Dazu wird die Baumstruktur unter Code Generation um mehrere Einträge erweitert (siehe Einträge beginnend mit TC) sobald Sie als System **target file** das **TwinCatGrt.tlc** ausgewählt haben.



Ansicht konfigurieren

Aufgrund der Vielzahl von Konfigurationsmöglichkeiten ist es möglich, die Ansicht umzuschalten. Nach Installation ist das Konfigurations-Level auf *Standard* eingestellt, welches nur die am häufigsten genutzten Parameter anzeigt. Sie können auch das Konfigurations-Level auf *Advanced* erhöhen, um deutlich mehr Einstellungen vornehmen zu können.

Nutzen Sie zur Einstellung der Ansicht das MATLAB® Command Window:

```
TwinCAT.ModuleGenerator.Settings.Change('ConfigurationLevel', 'Advanced')
TwinCAT.ModuleGenerator.Settings.Change('ConfigurationLevel', 'Standard')
```

Die getätigte Einstellung ist zunächst nur temporär. Um diese zu speichern, nutzen Sie den Save Befehl:

```
TwinCAT.ModuleGenerator.Settings.Save;
```

Sollten Sie die Code-Generierung und den Build-Prozess auf unterschiedlichen Systemen ausführen, stellen Sie sicher, dass das Konfigurations-Level auf beiden Systemen identisch ist.

Übersicht der Konfigurationsparameter

Einen Überblick über alle Konfigurationsparameter finden Sie in der Dokumentation im Abschnitt Übersichtstabelle über alle Konfigurationsparameter [▶ 128].

Lesen Sie die Tooltips

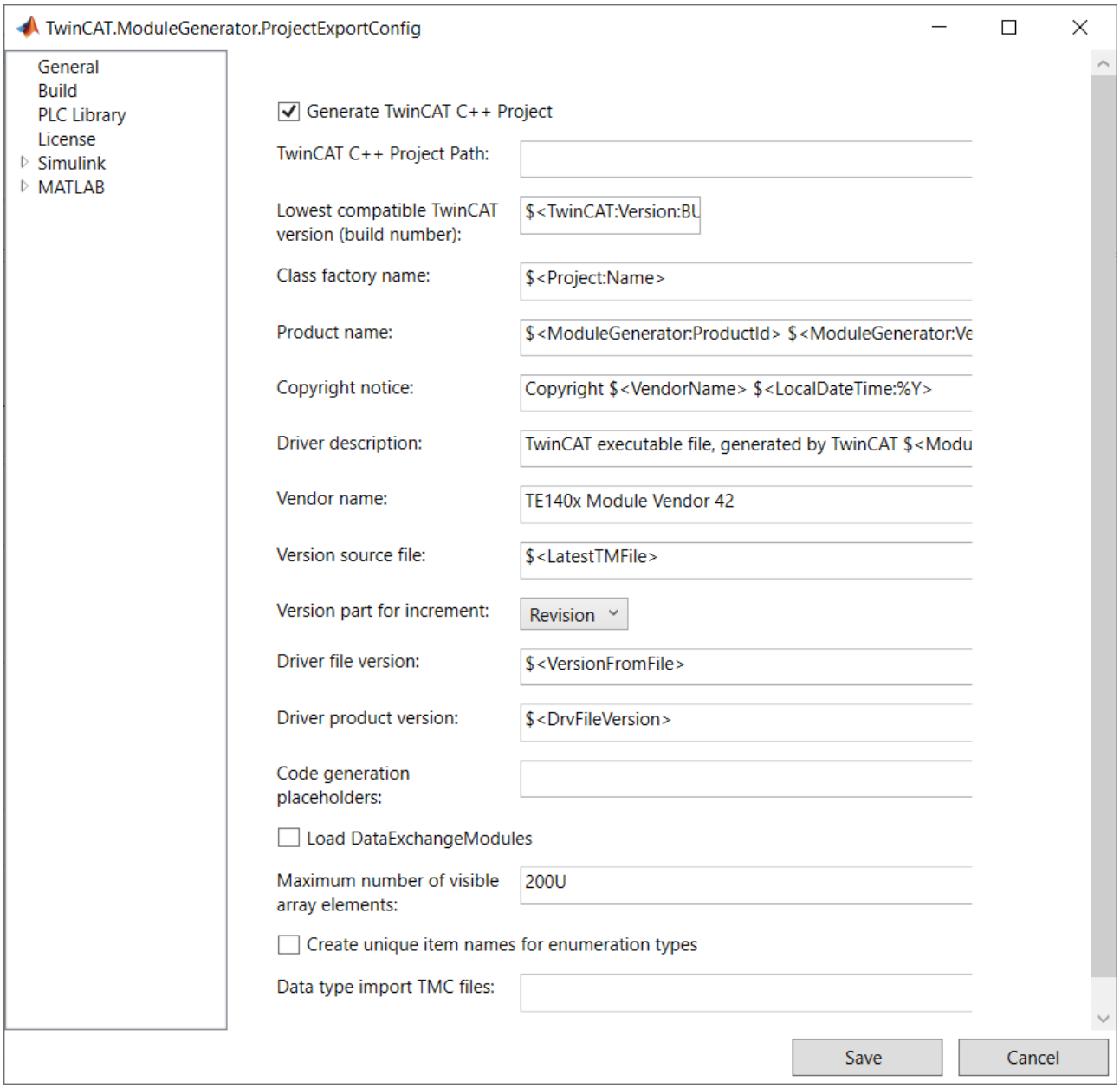
Wenn der Mauszeiger kurze Zeit über den Textfeldern der Dialogfenster verweilt, erscheint eine ausführliche Beschreibung der Option als Tooltip.

Einrichtung der Software verändern

Um die Default-Einstellungen des TwinCAT-Targets zu verändern, können Sie wie folgt in der MATLAB® Console auf einen Dialog zugreifen:

```
TwinCAT.ModuleGenerator.Settings.Edit
```

Hier werden Ihnen diverse Einträge angeboten, die Sie als Default-Wert hinterlegen können.



Änderungen übernehmen

1. Tragen Sie die neuen Default-Einstellungen im Dialogfenster ein.
2. Bestätigen Sie mit dem Button **Save**.
3. Starten Sie MATLAB® neu.

⇨ Die Änderungen sind übernommen worden.

Konfigurationen verändern nach dem Build

Viele der in den *Configuration Parameter* gewählten Einstellungen können in TwinCAT 3 auf Ebene der TcCOM-Instanzen wieder verändert werden, sodass beispielsweise für die Klasse eines Modells definiert ist, dass es über eine zyklische Task aufgerufen werden soll, die individuelle Instanz jedoch auch nachträglich zum Aufruf aus der SPS konfiguriert werden kann.

4.7.1 Übersichtstabelle über alle Konfigurationsparameter

Im Folgenden wird eine Übersicht aller Konfigurationsparameter gegeben. Die Konfigurationsparameter sind je nach gesetztem Konfigurationslevel im UI von Simulink® sichtbar oder unsichtbar. Sollten Sie den Modulgenerator über ein m-File konfigurieren [▶ 135], hat das Konfigurationslevel keinen Einfluss.

Zu jedem Parameter wird eine Kurzbeschreibung gegeben und in einigen Fällen für ausführlichere Informationen in weitere Abschnitte der Dokumentation verwiesen. Die Kategorisierung orientiert sich an der Darstellung im UI. Im UI wird der Displayname verwendet. Wenn Sie den Modulgenerator über ein m-File nutzen, ist die Spalte „Name“ entscheidend.

Category	Name	Displayname	Default	Description
TC General	Generate	Generate TwinCAT C++ Project	TRUE	Generate a TwinCAT C++ project. If unset, only code artifacts will be generated which can get used to generate C++ projects later [▶ 135].
	FullPath	TwinCAT C++ Project Path		Full path to the generated VCXPROJ file (e. g. "C:\Temp\MyGeneratedProject.vcxproj")
	LowestCompatibleTcBuild	Lowest compatible TwinCAT version (build number)	$<TwinCAT:Version:BUILD>	The lowest TwinCAT build number the generated C++ project and its modules and POUs are to be compatible with.
	ClassFactoryName	Class factory name	$<Project:Name>	Name of the generated C++-Project, Name of the TcCOM classfactory and tmx-file name
	ProductName	Product name	$<ModuleGenerator:ProductId> $<ModuleGenerator:Version:MAJOR.MINOR>	Product name, used e.g. for the module driver description and the module TMC description [▶ 164].
	Copyright	Copyright notice	Copyright $<VendorName> $<LocalDateTime:%Y>	Copyright notice of the generated module driver file [▶ 164]
	Description	Driver description	TwinCAT executable file, generated by TwinCAT $<ModuleGenerator:ProductId>	Driver description [▶ 164]
	VendorName	Vendor name	TE140x Module Vendor	Module vendor name, used as the company name of the generated executables in the repository [▶ 124] and the major module group as shown in the TwinCAT XAE module dialog [▶ 93].
	VersionSrc	Version source file	$<LatestTMFile>	Path to an existing TMC, TML or XML file containing the previous version value [▶ 142].
	IncrementVersion	Version part for increment	Revision	The part of the version number that is to be incremented [▶ 142].
	DrvFileVersion	Driver file version	$<VersionFromFile>	Executable file version and library version. [▶ 142]

Category	Name	Displayname	Default	Description
	DrvProductVersion	Driver product version	$<DrvFileVersion>	Product version [▶ 142]
	CodeGenPlaceholders	Code generation placeholders		Define custom placeholders
	UseDataExchangeModules	Load DataExchangeModules	0	Manually set DataExchangeModule dependency (currently no need to set manually)
	MaxVisibleArrayElements	Maximum number of visible array elements	200U	Specifies the maximum number of array elements to be displayed in the TwinCAT XAE. In the TwinCAT XAE, larger arrays cannot get expanded and linked to by its individual items
	CreateUniqueEnumItemNames	Create unique item names for enumeration types	1	Create unique item names for enumeration types.
	DataTypeTmcFiles	Data type import TMC files		TMC file(s) containing additional type definitions for code generation
TC Build	PreferToolArchitectureX64	Prefer X64 build tools	TRUE	Prefer X64 compiler and linker. Useful for complex source files, where X86 tools may run out of heap space.
	Verbosity	Codegeneration and build verbosity	Normal	Verbosity level of code generation and build output messages. Silent and Detailed are other possible values.
	Publish	Run the publish step after project generation	TRUE	Start the build procedure after code generation for all selected platforms. The generated module binaries and module descpription files will get copied to the "publish folder". Published modules are automatically located by the XAE and can get instantiated in all TwinCAT 3 projects. If unset, the module generation process will be stopped after code generation. To instantiate in a TwinCAT3 project, the generated C++ project needs to be inserted and built from.
	PublishPlatformtoolset	Platform Toolset	Auto	Choose Platform Toolset to build binaries.
	PublishConfiguration	Build configuration	Release	Build configuration to build binaries.
	PublishTcRTx86	TwinCAT RT (x86)	TRUE	Publish binaries for platform 'TwinCAT RT (x86).'
	PublishTcRTx64	TwinCAT RT (x64)	TRUE	Publish binaries for platform 'TwinCAT RT (x64).'
	PublishTcOSx64	TwinCAT OS (x64)	TRUE	Publish binaries for platform 'TwinCAT OS (x64)' (e.g. TwinCAT/BSD)
	ForceRebuildForPublish	Always rebuild all source files on publish	FALSE	Always rebuild all source files on publish

Category	Name	Displayname	Default	Description
	SignTwinCatCertName	Certificate name for TwinCAT signing		Certificate name for TwinCAT signing with OEM Certificate level 2. [▶ 96]
	TmxInstall	Install TMX	TRUE	Install all generated TwinCAT Objects on local XAE (fill local Engineering Repository [▶ 124]).
	TmxArchive	TMX Archive		Name of an optional archive containing all files required to use the generated TwinCAT Objects on another TwinCAT development system. [▶ 139]
	MsBuildPublishProperties	MsBuild publish properties		Set additional MsBuild publish properties.
	MsBuildProjProperties	MsBuild project properties		Set additional MsBuild project properties.
	PreCodeGenerationCallbackFcn	Pre code generation callback function		The defined MATLAB® function is called before code generation [▶ 187].
	PostCodeGenerationCallbackFcn	Post code generation callback function		The defined MATLAB® function is called after code generation [▶ 187].
	PostPublishCallbackFcn	Post publish callback function		The defined MATLAB® function is called after publish [▶ 187].
TC PLC library	LibCatPath	PLC library category description file	$<ProjectDir>\$<Name>.libcat.xml	Path to the PLC library category description file
	LibraryCategories	PLC library categories	$<VendorName>	Define PLC library category hierarchy. Default only one hierarchy level = vendor. List separated with | possible: <MainCategory>|<SubCategory1>|…
	GeneratePlcLibrary	Generate a PLC library	FALSE	Generate a PLC library with POUs. [▶ 211] Define containing POUs with parameter TcComWrapperFb and PlcFb>General>Generate.
	InstallPlcLibrary	Install the generated PLC library	FALSE	Install the generated PLC library for use in the local TwinCAT XAE/PLC [▶ 211].
	PlcTypePrefixes	Type Prefixes		Define custom type prefixes
	PlcVarPrefixes	Variable Prefixes	`PVOID=p | BOOL=b | BOOL32=b | DATE=d | TIME_OF_DATE=td | TIME=t | LTIME=t | GUID=n`	Define custom variable prefixes.
TC License	OemId	ID of OEM		ID of OEM. Required for OEM Licence checks [▶ 161]

Category	Name	Displayname	Default	Description
	OemLicenses	IDs of OEM Licenses		IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 161]
TC TcCom General	Generate	Generate TcCOM Module (TwinCAT Module Class)	TRUE	Generate a TcCOM module class for the model.
	OnlineChange	Online change support	FALSE	Allow to switch between different TcCOM module versions without switching TwinCAT runtime to config mode [▶ 145].
	ModuleProperties	TMC Properties		Additional properties added to the module description in the TMC file: Name1=Value1| Name2=Value2|...
	GroupName	GroupName	TE140x| Simulink Modules	Minor module group name in the TwinCAT XAE module dialog
	GroupDisplayName	GroupDisplayName	$<GroupName>	Minor module group description in the TwinCAT XAE module dialog
	GroupIcon	GroupIcon	$<TE140x:Icon>	Optional module group icon in the TwinCAT XAE module dialog
	ModuleIcon	ModuleIcon	$<TE140x:Icon>	Optional module icon in the TwinCAT XAE module dialog
	InitExceptionHandling	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress or handle floating point exceptions during initialization [▶ 225].
	UpdateExceptionHandling	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress or handle floating point exceptions during cyclic execution [▶ 225].
	AdditionalIncludeFiles	Additional include files		Additional files required to be included after rtwtypes.h
TC TcCom License	OemLicenses	IDs of OEM License	$<Project:OemLicenses>	IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 161]
TC TcCom Wrapper	TcComWrapperFb	TcCom Wrapper FB	FALSE	Generate a PLC Functionblock simplifying the interaction between a PLC and an instance of the generated TcCOM module [▶ 214]
	TcComWrapperFbProperties	TcCom Wrapper FB properties	FALSE	Generate properties for accessible data in the referenced TcCOM object [▶ 214]
	TcComWrapperFbPropertyMonitoring	TcCom Wrapper FB property monitoring	NoMonitoring	NoMonitoring: Online values of properties are not monitored in the PLC online view, CyclicUpdate: Update property values in the PLC online view cyclically, ExecutionUpdate: Update

Category	Name	Displayname	Default	Description
				property values in the PLC online view when the property getter or setter is called [▶ 214]
TC TcCom Additional settings	ModuleCaller	Default module caller	CyclicTask	CyclicTask: Call module via TwinCAT Task. Module: Cal module from another TwinCAT module (see e.g. TcCOM-Wrapper-FB).
	CallerVerification	Verify caller	Default	Verify the caller context to prevent concurrent execution of the model code and corresponding DataArea mappings. Skip verification to reduce the execution time.
	StepSizeAdaptation	Default StepSize adaptation mode	RequireMatchingTaskCycleTime	Configure how to handle differences between the default model step size(s) and the cycle time of the assigned task(s).
	ExecutionSequence	Default execution sequence	UpdateBeforeOutputMapping	Configure the execution order of input mapping, model code execution and output mapping.
	ExecuteModelCode	Execute model code after startup	TRUE	Start cyclic execution of the model code after startup by default. If FALSE, Module Parameter Execute needs to be set to TRUE to start execution of code.
	BlockDiagramExport	Export BlockDiagram	TRUE	Export graphical block diagram information for monitoring and optional debugging on the generated TwinCAT module in TwinCAT XAE [▶ 198]
	ResolveMaskedSubsystems	Resolve Masked Subsystems	FALSE	Resolve masked subsystems in the block diagram
	ExtendSignalResolution	Extended resolution of signals in block diagram	FALSE	Intensified search for assignments of variables and block diagram signals (blue signals). This option increases the build time. [▶ 240]
	BlockDiagramVariableAccess	Access to VariableGroup not referenced by any block	AssignToParent	Variables from a block within an unresolved subsystem are either assigned to the next higher visible block or hidden in the block diagram.
	BlockDiagramDebugInfoExport	Export BlockDiagram debug info	TRUE	Export additional information required to debug the module using the block diagram [▶ 202].
TC TcCom Interfaces	ExecutionInfoOutput	Create ExecutionInfo output	FALSE	Create additional output DataAreas containing execution and exception information [▶ 225].
	MonitorExecutionTime	Monitor execution time	FALSE	Calculate and expose the execution time of the module as an ADS variable for monitoring purposes.

Category	Name	Displayname	Default	Description
	InputDataAccess	Input: Data Access	Input Destination DataArea	Defines how the input variables are exposed in TwinCAT [▶ 147].
	InputCreateSymbols	Input: Create ADS Symbols	TRUE	Create ADS symbol information for the input variables [▶ 147]
	InputInitValues	Input: Initial values	FALSE	Create module parameters for the input variables to allow definition of initial values [▶ 147]
	InputProperties	Input: TMC Properties		Additional properties added to the Input symbol description in the TMC file. [▶ 171]
	OutputDataAccess	Output: Data Access	Output Source DataArea	Defines how the output variables are exposed in TwinCAT [▶ 147].
	OutputCreateSymbols	Output: Create ADS Symbols	TRUE	Create ADS symbol information for the output variables [▶ 147].
	OutputProperties	Output: TMC Properties		Additional properties added to the Output symbol description in the TMC file. [▶ 171]
	ParametersDataAccess	Parameters: Data Access	Internal DataArea	Defines how the model parameter variables are exposed in TwinCAT [▶ 147]
	ParametersCreateSymbols	Parameters: Create ADS Symbols	TRUE	Create ADS symbol information for the model parameter variables [▶ 147].
	ParametersInitValues	Parameters: Initial values	TRUE	Create module parameters for the model parameter variables to allow definition of initial values [▶ 147].
	ParametersProperties	Parameters: TMC Properties		Additional properties added to the Parameters symbol description in the TMC file. [▶ 171]
	BlockIoDataAccess	BlockIO: Data Access	Internal DataArea	Defines how the BlockIO variables are exposed in TwinCAT [▶ 147]
	BlockIoCreateSymbols	BlockIO: Create ADS Symbols	TRUE	Create ADS symbol information for the BlockIO variables [▶ 147].
	BlockIoProperties	BlockIO: TMC Properties		Additional properties added to the BlockIO symbol description in the TMC file. [▶ 171]
	ContStateDataAccess	ContState: Data Access	Internal DataArea	Defines how the continuous state variables are in TwinCAT [▶ 147]
	ContStateCreateSymbols	ContState: Create ADS Symbols	TRUE	Create ADS symbol information for the continuous state variables [▶ 147].
	ContStateProperties	ContState: TMC Properties		Additional properties added to the ContState symbol description in the TMC file. [▶ 171]
	DWorkDataAccess	DWork: Data Access	Internal DataArea	Defines how the DWork variables are exposed in TwinCAT [▶ 147]

Category	Name	Displayname	Default	Description
	DWorkCreateSymbols	DWork: Create ADS Symbols	TRUE	Create ADS symbol information for the DWork variables [▶ 147].
	DWorkProperties	DWork: TMC Properties		Additional properties added to the DWork symbol description in the TMC file. [▶ 171]
	DataStoreDataAccess	DataStore: Data Access	None	Defines how the DataStore variables are exposed in TwinCAT [▶ 147]
	DataStoreCreateSymbols	DataStore: Create ADS Symbols	TRUE	Create ADS symbol information for the DataStore variables [▶ 147].
	DataStoreReadOnly	DataStore: Read Only	FALSE	Restrict ADS access to be read only for the DataStore variables [▶ 147].
	DataStoreProperties	DataStore: TMC Properties		Additional properties added to the DataStore symbol description in the TMC file. [▶ 171]
	SymbolProperties	Additional TMC Symbol Properties		Additional properties added to specific symbol descriptions in the TMC file. [▶ 171]
	VariableSymbolMapping	Mapping between variable names and ADS symbol names	Identical	Defines the TwinCAT symbol names for the generated C/C++ variables. 'Identical': Symbol name equals variable name, 'Classic': Use symbol names known from TE1400 Release 1.2.x.x [▶ 147]
TC TcCom External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [▶ 222].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode [▶ 222] connection before starting model code execution.
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [▶ 222].
TC PlcFb General	Generate	Generate TwinCAT PLC Function Block	TRUE	Generate a PLC-FB for the model [▶ 219].
	InitExceptionHandling	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during initialization [▶ 225].
	UpdateExceptionHandling	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during cyclic execution [▶ 225].
TC PlcFb License	OemLicenses	IDs of OEM License	$<Project:OemLicenses>	IDs of OEM Licenses. Multiple IDs may be inserted as a comma seperated list. "{GUID},{GUID}" [▶ 161]

Category	Name	Displayname	Default	Description
TC PlcFb Additional settings	MonitorExecutionTime	Monitor ExecutionTime	FALSE	Calculate and expose the execution times of TwinCAT modules as an ADS variable for monitoring purposes.
PlcFb->Interface	InputAttributes	Input variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
	OutputAttributes	Output variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
TC PlcFb External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [▶ 222].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode connection before starting model code execution [▶ 222].
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [▶ 222].

4.7.2 Parametrierung der Code-Generierung über ein m-file

Es gibt zwei Möglichkeiten, die Parametrierung der Code-Generierung über ein m-file (oder mlx-file) vorzunehmen:

- über modell-spezifischen Simulink®-Parameter mit `set_param`
- über eine Instanz des Modulgenerators `TwinCAT.ModuleGenerator`

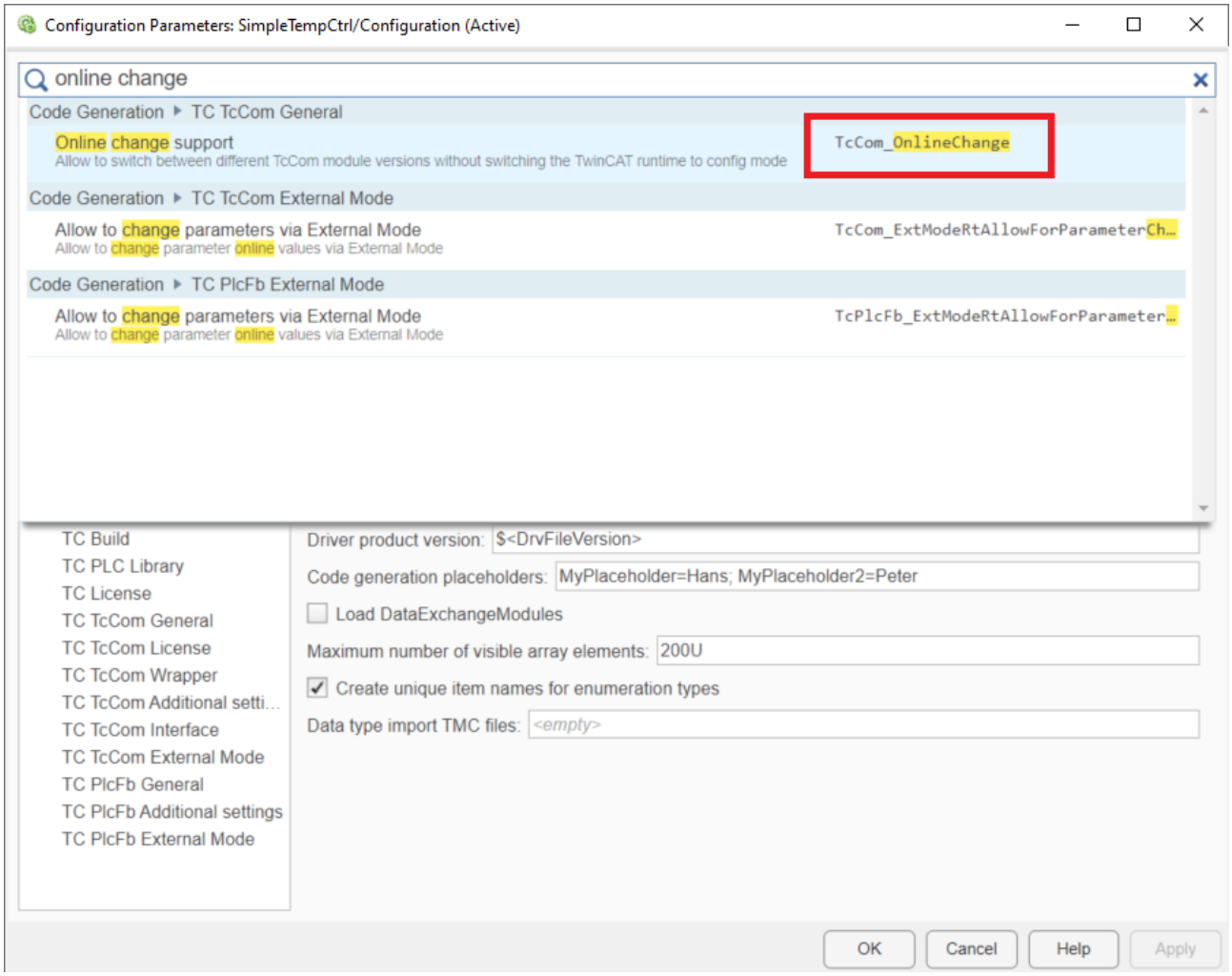
Konfiguration über Simulink®-Parameter

Mit `set_param` können Sie einem Objekt in Simulink® spezifische Parameter zuweisen. Wenn Sie die Konfiguration des Modulgenerators mit `set_param` vornehmen, wird diese entsprechend im Simulink®-Modell hinterlegt.

Zur Strukturierung der Konfigurationsparameter [▶ 128] kann hier kein Namespace genutzt werden, deshalb wird den Konfigurationsparametern ein Präfix vorangestellt. Dem Parameternamen (Spalte „Name“ in der Tabelle der Konfigurationsparameter [▶ 128]) wird je nach Ebene ein `Project_`, `TcCom_` oder `TcPlcFb_` vorgesetzt.

- `Project_` Präfix: Bei allen Parametern, die in der Darstellung der Configuration Parameter in den Tabs TC General, TC Build, TC PLC Library und TC License, gruppiert sind.
- `TcCom_` Präfix: Bei allen Parametern, die in der Darstellung der Configuration Parameter in den Tabs TC TcCom gruppiert sind.
- `TcPlcFb_` Präfix: Bei allen Parametern, die in der Darstellung der Configuration Parameter in den Tabs TC PlcFb gruppiert sind.

Sie können ebenfalls den genauen Parameternamen durch die Suchfunktion im Simulink®-Modell herausfinden.



Beispiel

```
% load Simulink model
controller = load_system('TempCtrl.mdl');

% configure TwinCatGrt
set_param(controller,'SystemTargetFile','TwinCatGrt.tlc');

% set project specific parameters
set_param(controller,'TcProject_VendorName','CompanyName');
set_param(controller,'TcProject_GeneratePlcLibrary','on');
set_param(controller,'TcCom_OnlineChange','on');
set_param(controller,'TcPlcFb_MonitorExecutionTime','on');

% build the model
slbuild(controller);

% save and close the model
close_system(controller,1);
```

Konfiguration über den Modulgenerator

Parameter-Einstellungen wie im obigen Beispiel können auch über den Modulgenerator getätigt werden. Damit verbleiben die Einstellungen nicht im Simulink®-Modell, sondern in der Instanz des Modulgenerators.

Dazu wird lediglich für das Simulink®-Modell das SystemTargetFile definiert und der Parameter TcProject_Generate ausgeschaltet. Dadurch werden nur *Code Artifacts* erzeugt, aber kein TwinCAT C++-Projekt abgeleitet und auch dementsprechend nicht kompiliert. Die Code Artifacts werden im aktuellen MATLAB®-Pfad im Ordner *<ModelName>_tcgrt* abgelegt.

Export des Blockdiagramms

Auf Ebene des Simulink®-Modells sollten Sie ebenfalls entscheiden, ob Sie das Blockdiagramm exportieren möchten. In den nachfolgenden Schritten arbeiten Sie auf den *Code Artifacts* und nicht mehr auf dem Simulink®-Modell.

Sie können im Nachgang den *Code Artifacts* Ordner angeben, um eine ProjectExport-Konfiguration in den `TwinCAT.Modulgenerator` zu laden. Hier können Sie dann Ihre Einstellungen vornehmen und das Projekt kompilieren.

Beispiel

```
% load Simulink model
controller = load_system('TempCtrl.mdl');

% configure TwinCatGrt
set_param(controller,'SystemTargetFile','TwinCatGrt.tlc');

% disable generation of C++ project files for each model (suppresses build)
set_param(controller,'TcProject_Generate','off');

% create code artifacts
slbuild(controller);

% save and close the model
close_system(controller,1);

% find the code artifacts in the existing code generation directories
controllerBuildDir = fullfile(pwd,'TempCtrl_tcgrt');

% load existing export configurations
controllerCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(controllerBuildDir);

% show complete configuration in MATALB Command Window
controllerCfg

% set project specific parameters
controllerCfg.Project.VendorName = "Company";
controllerCfg.Project.GeneratePlcLibrary = true;
controllerCfg.ClassExportCfg{1}.TcCom.OnlineChange = true;
controllerCfg.ClassExportCfg{1}.PlcFb.MonitorExecutionTime = true;

% set generate to true
controllerCfg.Project.Generate = true;

% instantiate and run the project exporter
TwinCAT.ModuleGenerator.ProjectExporter(controllerCfg);
```

Anwendungsbeispiele

Die Trennung des Code-Generierungsvorgangs in die Schritte (1) Erstellung der *Code Artifacts* und (2) Konfiguration des Modulgenerators und Erstellung der TwinCAT-Objekte, kann in unterschiedlichen Szenarien hilfreich sein:

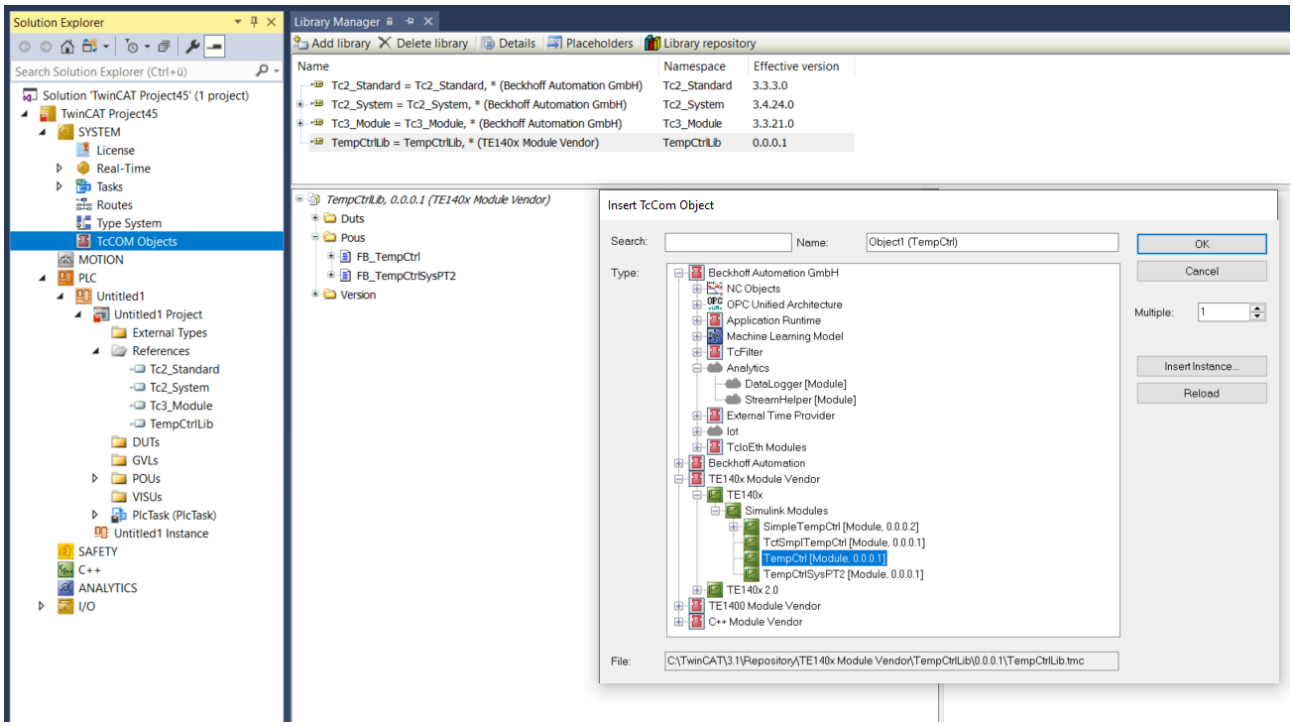
- Sie möchten die Einstellungen des Modulgenerators nicht in jedem Simulink®-Modell speichern.
- Sie möchten mehrere Simulink®-Modelle in einem Projekt zusammenfügen. [▶ 137] Damit werden alle Modelle in einem Treiber und in einer SPS-Bibliothek zusammengefasst. Beispiel: `TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules')`
- Sie arbeiten mit einem Build-Server oder mit einem CI/CD System. Beispiel: `TwinCAT.ModuleGenerator.Samples.Start('Continuous Integration')`

4.7.3 Bündelung mehrerer Modelle in einem TwinCAT-Treiber

Automatisch generierter Code von Simulink®-Modellen und MATLAB®-Funktionen können in einem einzelnen C++-Projekt gebündelt werden. Nach dem Build-Vorgang sind dann alle gebündelten Objekte in einem Treiber verfügbar.

Vorteile der Bündelung

Wenn eine SPS-Bibliothek erzeugt wird, sind dann alle erstellen Objekte als Funktionsblock (FB) in dieser Bibliothek gelistet. Obwohl nur ein Treiber und eine tmc-Datei erstellt wird, sind dennoch alle Module einzeln unter **System > TcCOM** instanziierbar, d. h. aus Sicht des Anwenders in TwinCAT XAE ändert sich hinsichtlich der Nutzung der TcCOM-Objekte nichts. Bei Nutzung der SPS-Bibliothek erhöht sich die Übersichtlichkeit.



Vorteile der Bündelung in einem Treiber:

- Die Anzahl der Dateien im Repository Verzeichnis wird deutlich reduziert. Damit sind auch weniger Dateien auf andere Engineering Systeme zu kopieren, um eine Vielzahl von Modulen auf Engineering Systemen verfügbar zu machen.
- Die Verwaltung von verschiedenen Versionen wird vereinfacht, da miteinander interagierende Module gebündelt ausgetauscht werden können, sodass keine Versionskonflikte entstehen.

HINWEIS

Simulink Coder™ unterstützt keine Namespaces

Werden Datentypen oder Funktionen mit gleicher Benennung in mehreren Modellen definiert, schlägt der Build-Vorgang fehl, da die Definitionen im selben Namespace liegen.

Vorgehensweise

1. Deaktivieren Sie in Simulink® „Run the publish step after project generation”. Dadurch wird nach der Code-Generierung abgebrochen und das erstellte C++-Projekt nicht kompiliert.
2. Nutzen Sie bei der Bündelung mehrerer Module einfach die generierten Ordner *<modelname>_tcgrt*, welche im aktuellen MATLAB®-Pfad abgelegt werden.
3. Laden, Bündeln und Konfigurieren Sie in einem Projekt über den ModuleGenerator die Exportkonfigurationen *(<modelname>_tcgrt*).
4. Erstellen Sie ein Export-Projekt.

Im Folgenden ist dies beispielhaft an der Bündelung von 2 Exportkonfigurationen gezeigt:

```
% find the code artifacts in the existing code generation directories
controllerBuildDir = fullfile(pwd,'TempCtrl_tcgrt');
ctrlsystemBuildDir = fullfile(pwd,'TempCtrlSysPT2_tcgrt');
% load existing export configurations
controllerCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(controllerBuildDir);
ctrlsystemCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(ctrlsystemBuildDir);
% create a new project export configuration
combinedCfg =
```

```
TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',fullfile(pwd,'TempCtrlLib','TempCtrlLib.vcxpr
oj'));
% add the loaded class export configurations to the new project configuration
combinedCfg.AddClassExportConfig(controllerCfg.ClassExportCfg{1});
combinedCfg.AddClassExportConfig(ctrlsystemCfg.ClassExportCfg{1});
% ...
% additinal class export configurations can be added here, loaded from
% - Simulink code generation directories (as described)
% - MATLAB code generation directories (from MATLAB Coder with TE1401) in the same way
% turn on generation and installation of the PLC library
combinedCfg.Project.GeneratePlcLibrary = true; % generate a PLC Lib true/false
combinedCfg.Project.InstallPlcLibrary = true; % install the PLC lib on local system true/false
% instantiate and run the project exporter
TwinCAT.ModuleGenerator.ProjectExporter(combinedCfg);
```

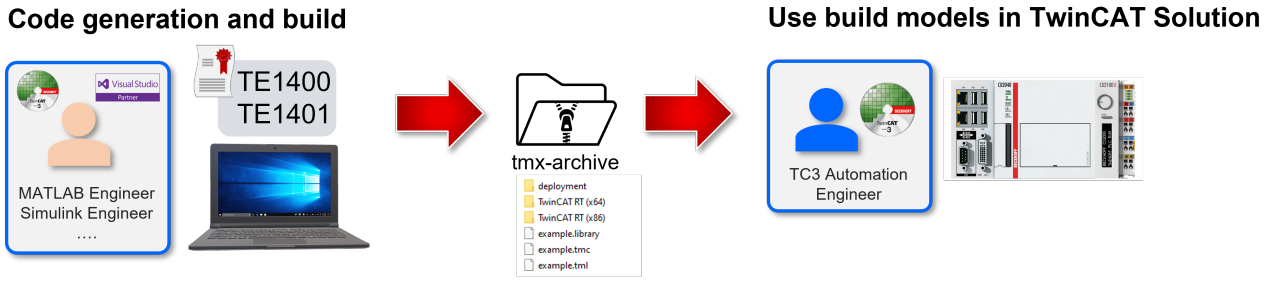
Beispielcode in MATLAB®

Öffnen Sie das passende Beispiel mit:
`TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules')`

4.7.4 Teilen von erstellten TwinCAT-Objekten

Häufig sind die Kolleginnen oder Kollegen, die aus Simulink® oder MATLAB® TwinCAT-Objekte (TcCOM und/oder SPS-Bibliothek) erstellen, nicht diejenigen, die die erstellten Module in eine TwinCAT-Konfiguration einbinden und den Maschinencode erstellen.

Um mit den erstellten TwinCAT-Objekten im TwinCAT XAE arbeiten zu können, müssen diese auf dem lokalen Engineering PC im Repository Ordner vorhanden sowie die SPS-Bibliothek im lokalen PLC Library Repository installiert sein. Händisches Kopieren auf Engineering PCs ist fehleranfällig. Es wird daher dringend empfohlen, ein sogenanntes *TMX-Archiv* zu erstellen und dieses mit Kolleginnen und Kollegen zu teilen, welche die erstellten Module nutzen möchten.

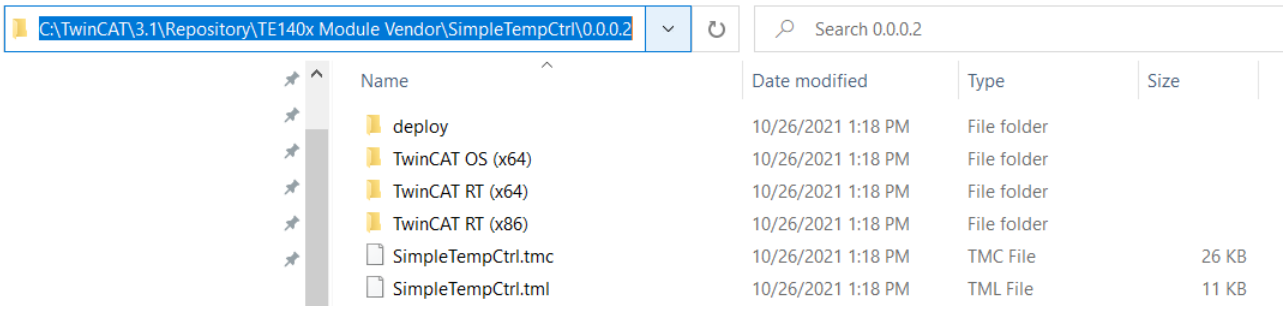


Was ist ein TMX-Archiv?

Das TMX-Archiv ist ein Archiv aller notwendigen Dateien, die für die Nutzung von TcCOM und SPS-Bibliotheken in einem TwinCAT Engineering benötigt werden. Das Archiv beinhaltet alle kompilierten Treiber, Beschreibungsdateien (z. B. für das Blockdiagramm in TwinCAT), die tmc-Datei für TcCOM oder die tml- bzw. library-Datei für die SPS.

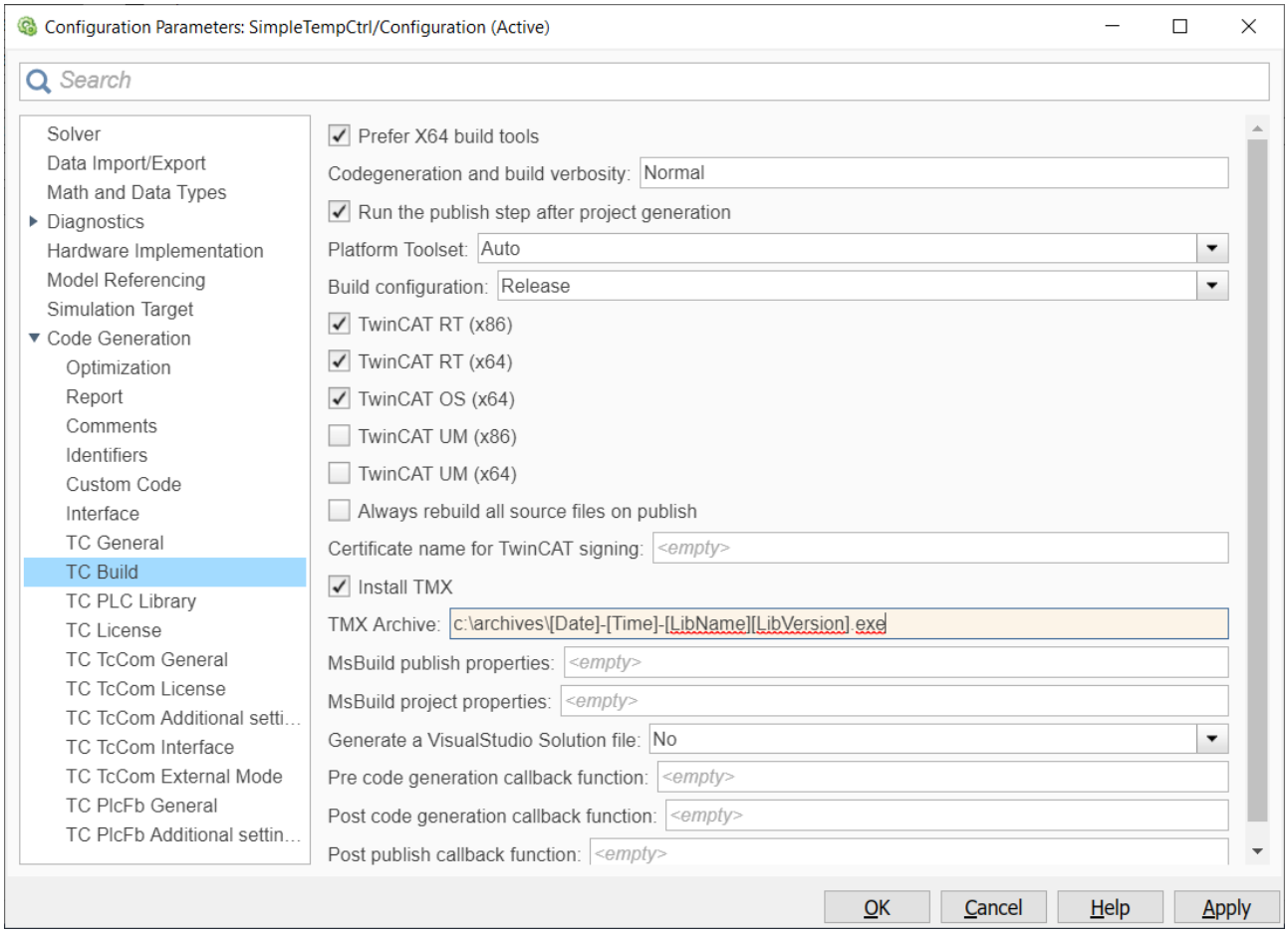
Ein TMX-Archiv muss nur in einen beliebigen Pfad auf einen Engineering PC kopiert und ausgeführt werden. Es handelt sich um ein selbstextrahierendes Archiv, welches alle Dateien automatisch an die korrekte Stelle kopiert.

Beispielsweise müssen die Dateien des Modells *SimpleTempCtrl* in Version 0.0.0.2 vom Vendor TE140x Module Vendor an dieser Stelle liegen:



Wie erstelle ich ein TMX-Archiv?

Sie können unter TC Build den Pfad und den Namen des TMX-Archivs angeben, um es mit dem nächsten Build erstellen zu lassen.



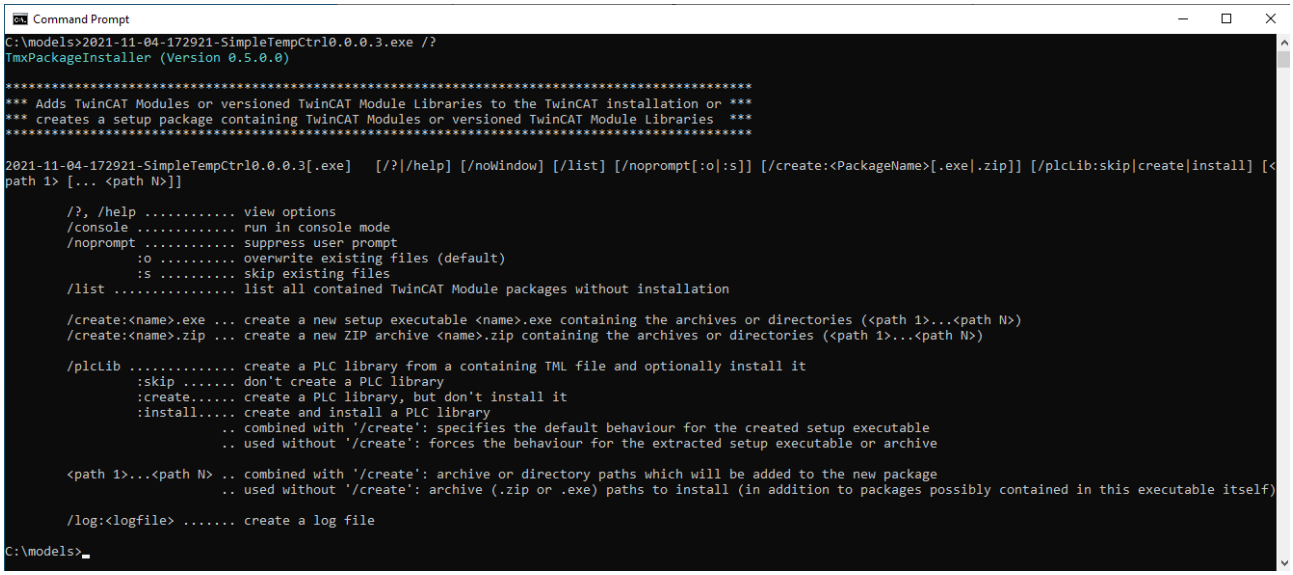
Sie können bei der Pfad- und Namensangabe auch Platzhalter, wie oben im Beispiel angegeben, nutzen. Ergebnis dieser Einstellung ist z. B. ein TMX-Archiv *2021-11-04-172921-SimpleTempCtrl0.0.0.3.exe* (neuer Build, daher Revision hochgezählt).

Wie verwende ich ein TMX-Archiv?

Das TMX-Archiv können Sie in einen beliebigen Pfad auf einen Engineering PC kopieren und ausführen. Dadurch werden die Dateien des Archivs an die richtige Stelle in Ihrem Repository kopiert (selbstextrahierendes zip). Möchten Sie auf dem Engineering PC nicht nur die im Archiv befindlichen TcCOM-Module nutzen, sondern auch die SPS-Bibliothek, muss diese noch über das PLC–Library Repository installiert werden.

Alternativ können Sie auch das **Command Prompt** nutzen. Rufen Sie die Hilfe des **TmxPackageInstaller** auf über:

```
<tmxarchive>.exe /?
```



Um beispielsweise ein TMX-Archiv zu entpacken und die darin befindliche SPS-Bibliothek im lokalen TwinCAT Engineering zu installieren, führen Sie folgenden Befehl in der Kommandozeile aus:

```
<tmxarchive>.exe /plclib:install
```

Was benötige ich an Software auf dem TwinCAT Engineering PC?

Auf dem Engineering PC, auf dem Sie die bereits kompilierten TwinCAT-Objekte nutzen möchten, um Sie in eine TwinCAT-Konfiguration einzubinden, benötigen Sie lediglich eine TwinCAT XAE Installation.

Sie benötigen auf diesem Engineering PC weder ein vollständiges Visual Studio (die XAE Shell des TwinCAT XAE Setups genügt) noch benötigen Sie eine MATLAB®-Installation.

In einigen Fällen ist es notwendig, auf dem Engineering PC die sogenannten *DataExchange-Modules* zu installieren, damit Sie die auf einem anderen System erstellten TwinCAT-Objekte verwenden können.

Diese Fälle sind:

- Das erstellte Modul wurde mit der „External Mode“-Option erstellt.
- Das erstellte Modul nutzt den TwinCAT File Writer [▶ 120] oder MAT-File Logging.
- Das erstellte Modul enthält Blöcke aus dem TE1410 TwinCAT Interface for MATLAB®/Simulink®.

In allen anderen Fällen besitzen die erstellten TwinCAT-Objekte keine Abhängigkeit zu den DataExchange-Modules.

Das **DataExchange-Modules** Setup ist Bestandteil des TE14xx-ToolsForMatlabAndSimulink Setup und wird standardmäßig installiert.

Das DataExchange-Modules Setup wird zusätzlich in den folgenden Ordner kopiert, sodass der/die Mitarbeitende, welche/r das TE14xx-ToolsForMatlabAndSimulink Setup installiert hat, das DataExchange-Modules Setup an die betreffenden Kolleginnen und Kollegen verteilen kann.

```
<TwinCATInstallDir>\TwinCAT\Functions\TE14xx-
ToolsForMatlabAndSimulink\TE140x\SDK
```

Continuous Integration mit dem Target for Simulink® oder Target for MATLAB®

Tipps zur Einrichtung einer CI/CD Pipeline sind zusammengestellt in folgendem Sample:

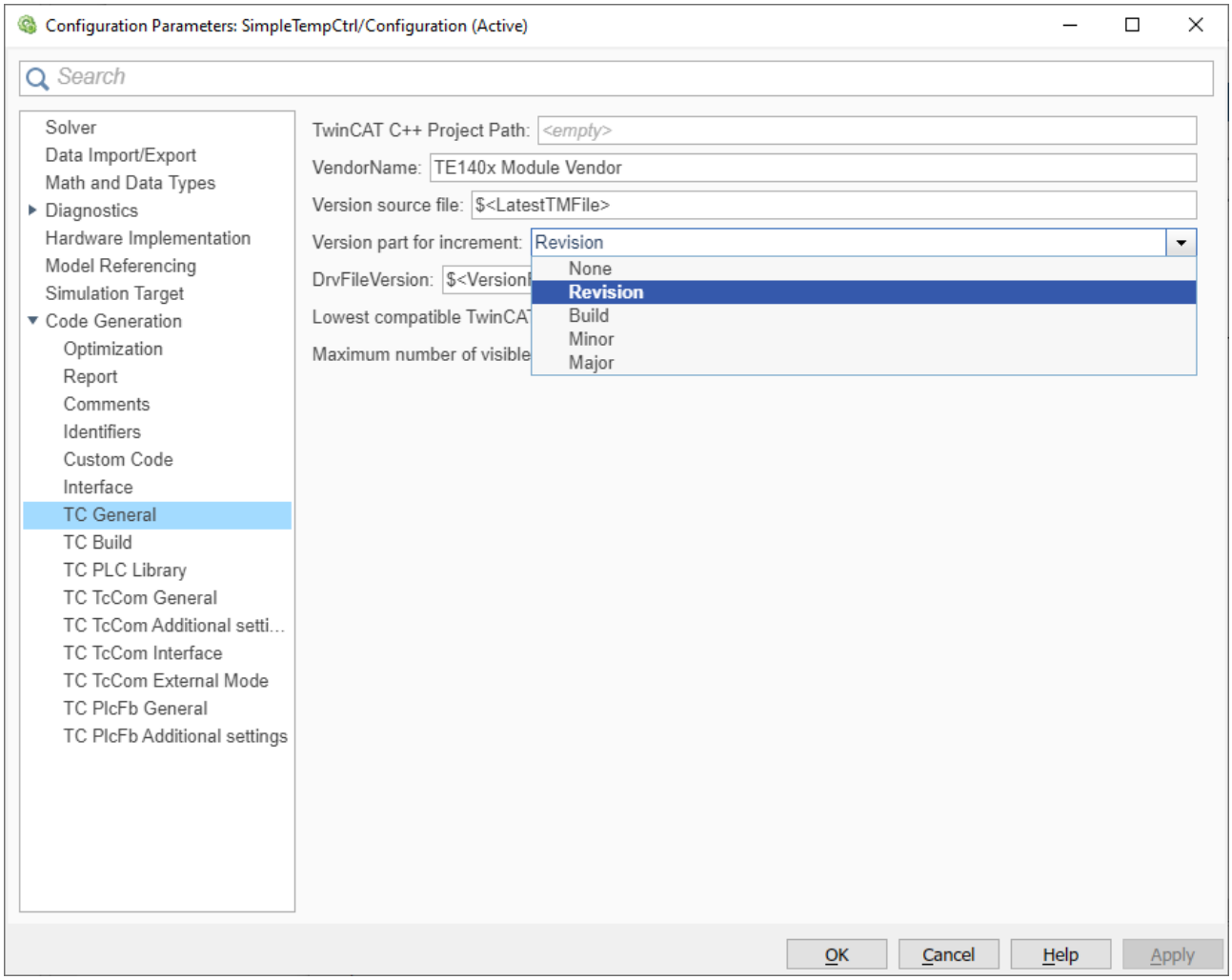
```
TwinCAT.ModuleGenerator.Samples.Start('Continuous Integration')
```

4.7.5 Erstellung versionierter Treiber

Jedes aus Simulink® erstellte Objekt trägt eine Versionsinformation. Sie können entsprechend mehrere Versionen eines Simulink®-Modells bauen und in TwinCAT versions-selektiv die erstellten Module instanziieren.

Versionierung in Simulink® definieren

Vor dem Erstellen eines SPS-Funktionsbausteins oder eines TcCOM-Objekts können Sie unter TC General mit den Einträgen „Version source file“ und „Version part for increment“ die Version des TcCOM und der erstellten SPS-Bibliothek definieren.



Automatische Versions-Inkremente

Die Basis-Version auf Basis derer ein Versionsupdate erstellt werden soll, wird über „Version source file“ angegeben. Im Standardfall ist dort $<LatestTMFile> angegeben. Dadurch wird auf dem lokalen Engineering PC nach der letzten verfügbaren Version des Modells gesucht und diese dann als Basis für den Versionsinkrement verwendet.

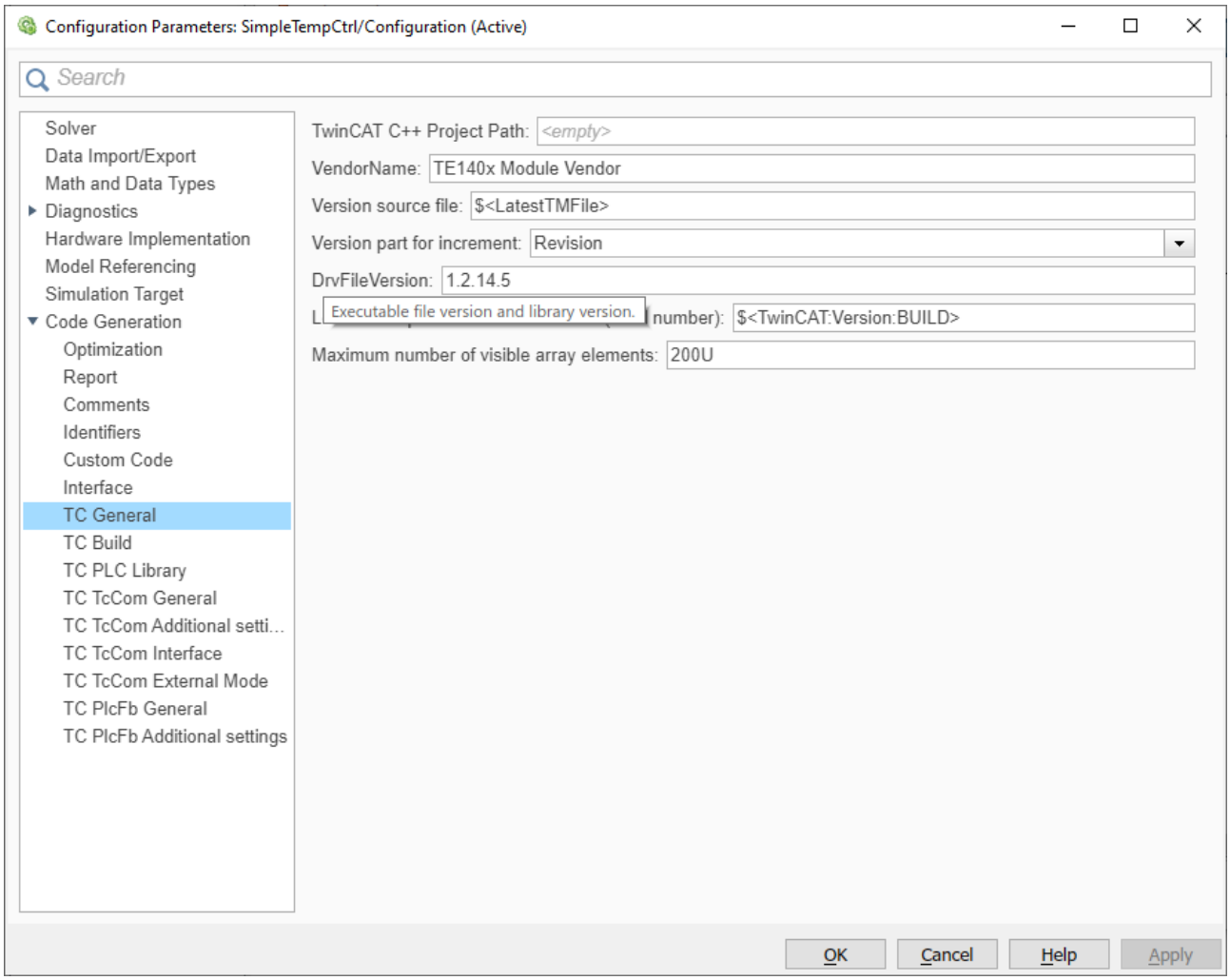
Die Versionsnummer besteht aus vier Digits, z. B. 1.0.3.2 oder 2.12.123.14. Jede Stelle kann separat hochgezählt werden nach dem Schema: <Major>.<Minor>.<Build>.<Revision>

Ist beispielsweise die letzte Version eines Modells mit dem Namen „MyModelXY“ auf dem Engineering PC mit 1.2.12.4 gefunden und das Inkrement auf „Revision“ gesetzt, wird eine Version 1.2.12.5 erstellt.

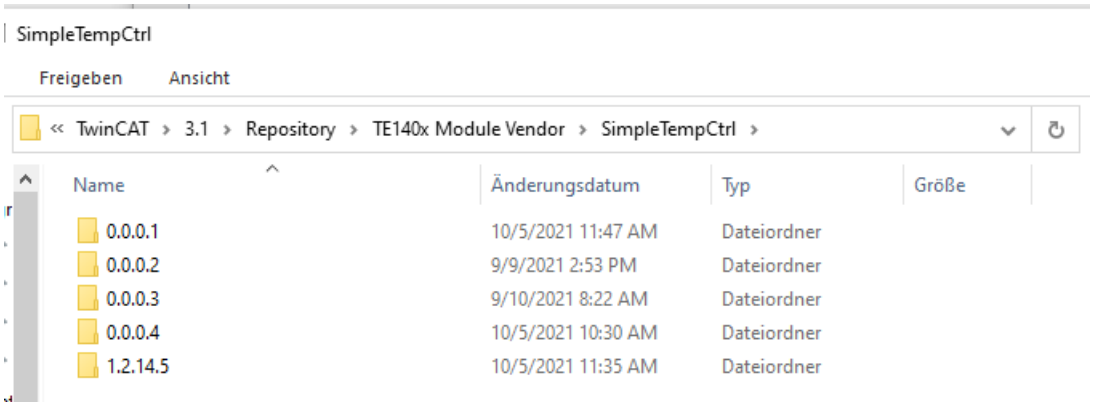
Wird „None“ gewählt, erfolgt kein Versionsupdate und die letzte Version auf dem Engineering PC wird überschrieben.

Vorgabe einer festen Versionsnummer

Soll eine Version in Simulink® vorgegeben werden, kann dies über **DrvFileVersion** erfolgen. Geben Sie einfach die Zielversion im Eingabefeld an.



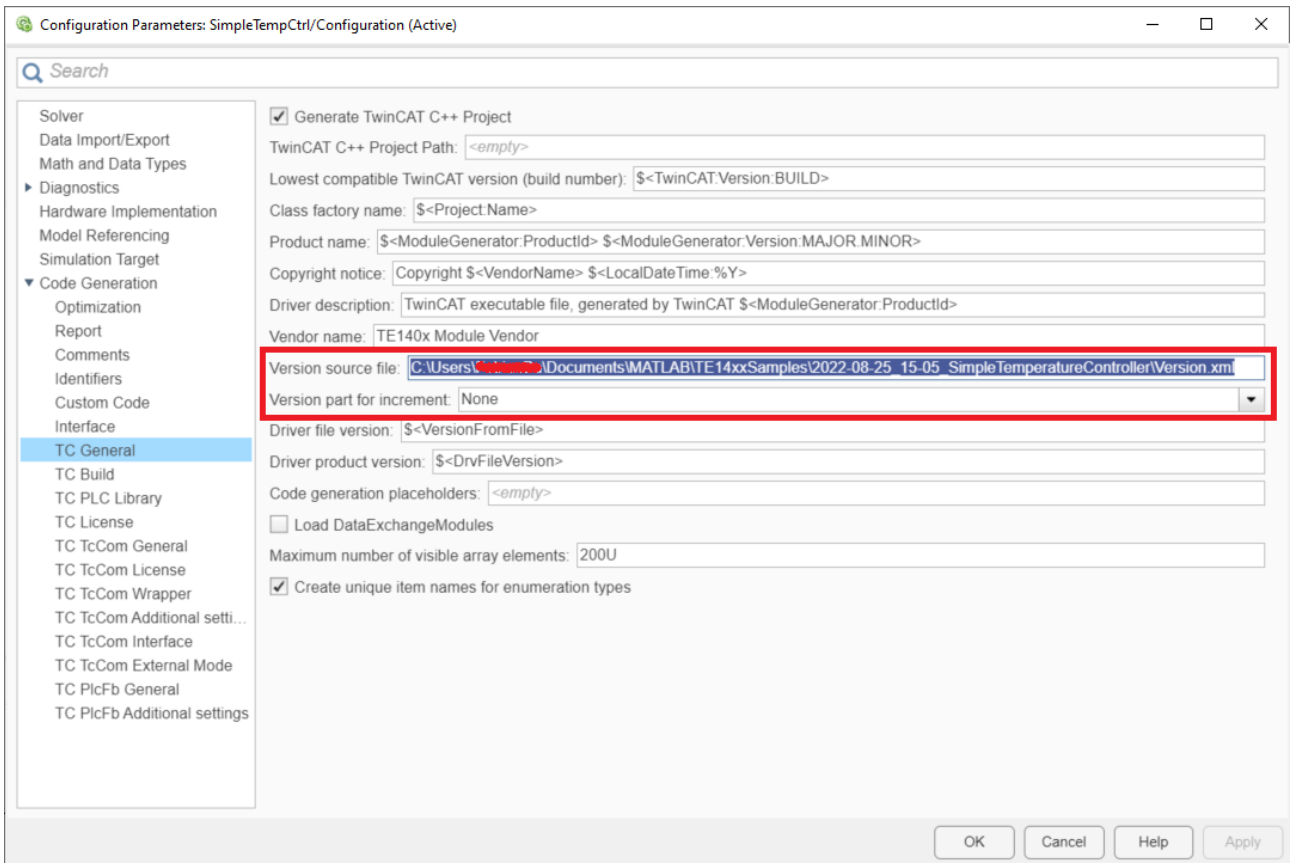
Im Engineering Repository wird für jede erstellte Version ein Ordner unter dem Modellnamen erstellt. Jeder Versionsordner enthält dann die entsprechenden Treiber und TwinCAT-Dateien. Siehe auch TwinCAT Objekte [▶ 124].



Vorgabe einer Versionsnummer über ein externes File

Sie können ebenso eine externe Datei zur Vorgabe einer Version nutzen. Dies ist beispielsweise eine gängige Methode bei der Nutzung von Build Agents bei *Continuous Integration*.

Zur Konfiguration in Simulink® setzen Sie unter TC General **Version part for increment** auf „None“ und geben Sie den Fullpath zu ihrem Version-File unter **Version source file** an.



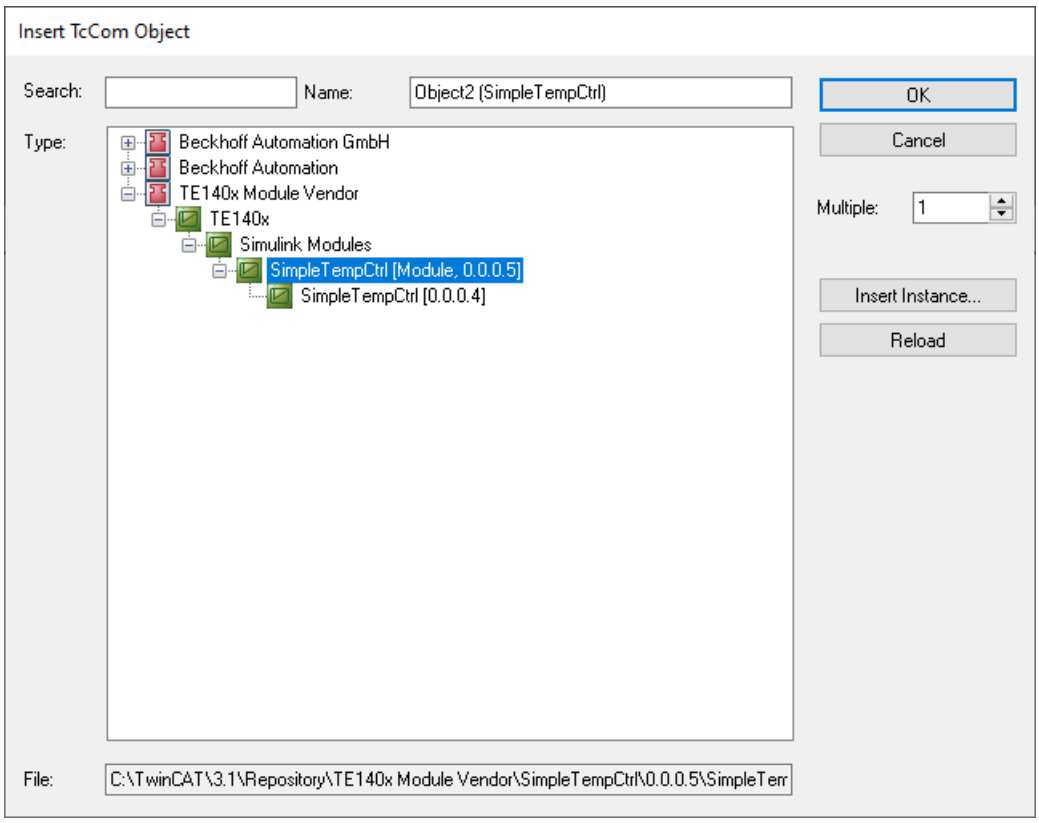
Aufbau der externen Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appSettings>
    <add key="major" value="1" />
    <add key="minor" value="3" />
    <add key="build" value="1301" />
    <add key="revision" value="267" />
  </appSettings>
</configuration>
```

Versionierte Modelle im TwinCAT XAE verwenden

Alle im Engineering Repository verfügbaren Modelle können auch in jeder verfügbaren Version instanziiert werden. Navigieren Sie dazu, wie gewohnt, durch den Baum, um das TcCOM ihrer Wahl zu finden. In der letzten Hierarchieebene können Sie nun auch die Version des TcCOM auswählen.

Beispielhaft sind hier zwei Versionen (0.0.0.4 und 0.0.0.5) des SimpleTempCtrl verfügbar:



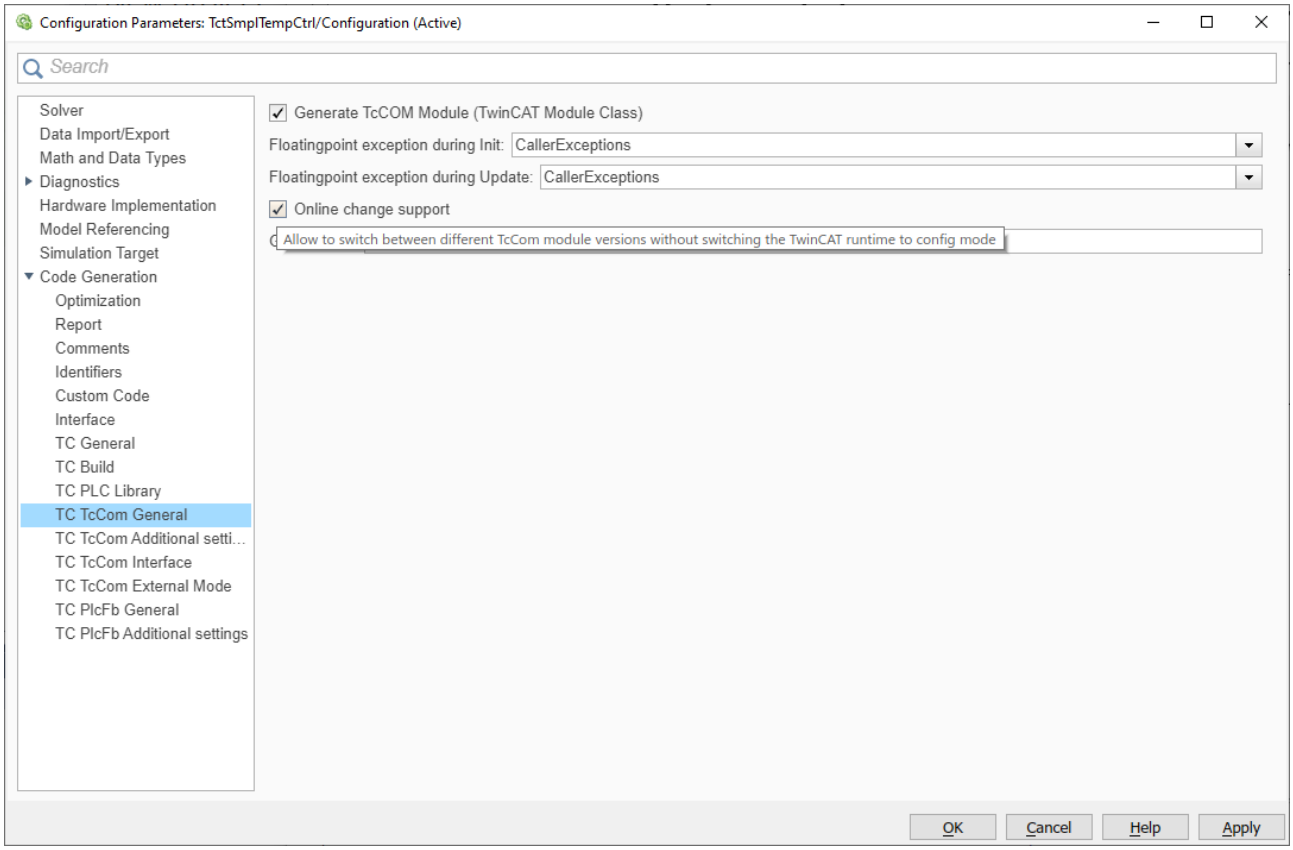
Online Change eines TcCOM während TwinCAT Run

✓ Um im laufenden Betrieb zwischen verschiedenen Versionen eines TcCOM zu schalten, muss das entsprechende Interface implementiert sein.

1. Wählen Sie dazu in Simulink® die Checkbox zu **Online change support** unter dem Tab **TC TcCom General** an.

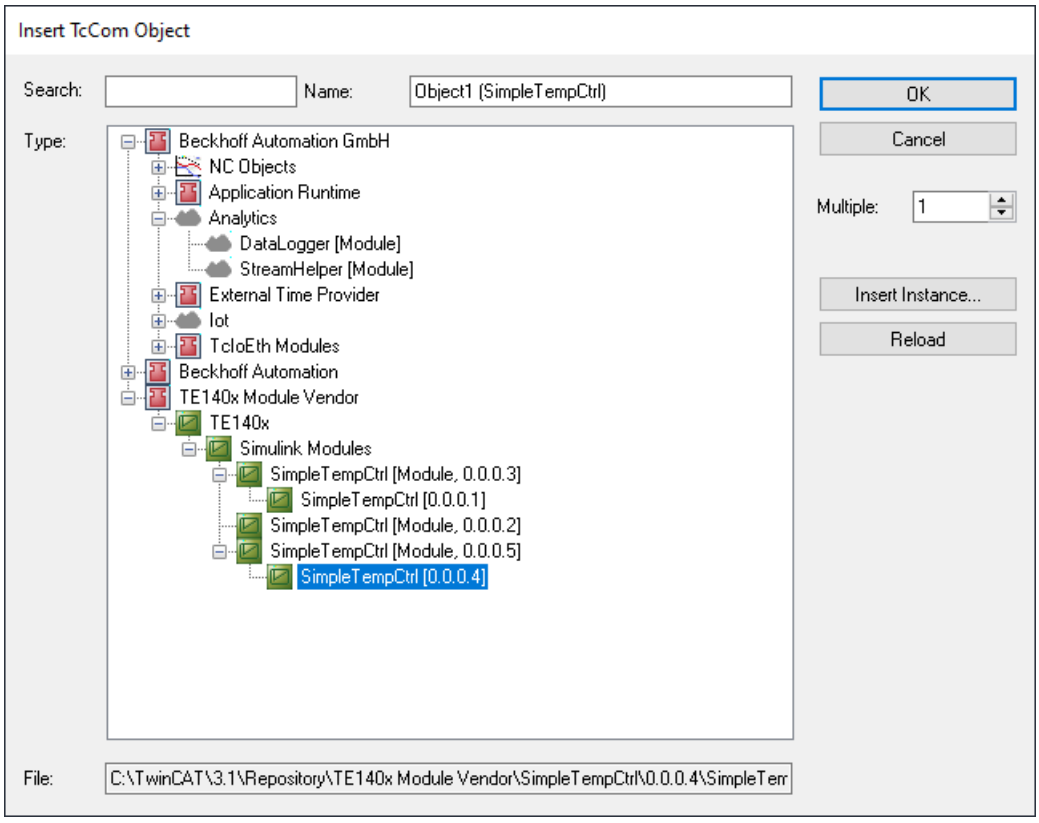
Online Change für SPS-Funktionsbaustein

Wenn Sie den Funktionsbaustein (PLC-FB) in einer versionierten SPS-Bibliothek verwenden, müssen Sie die Checkbox **Online change support** nicht aktivieren. Der Online-Change-Prozess läuft dann über den SPS-spezifischen Mechanismus.

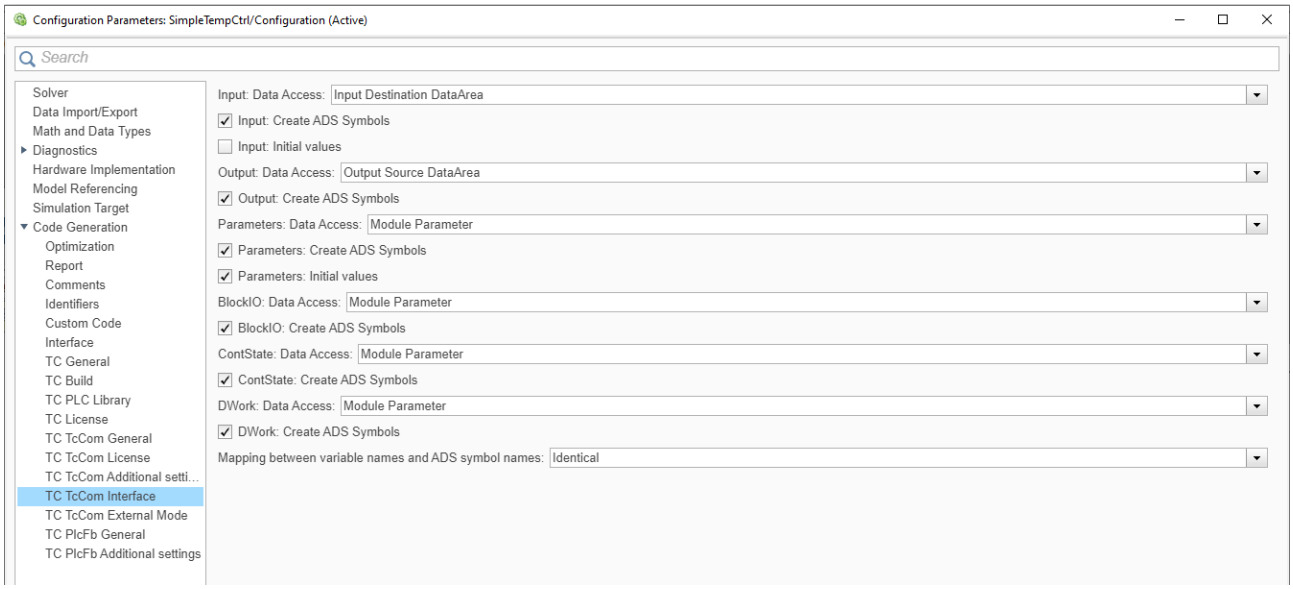


Darüber hinaus müssen angelegte Data Areas des TcCOM kompatibel zueinander sein. Wenn **Online change support** aktiviert ist, ist im Insert TcCom Object-Dialog die letzte Hierarchieebene strenger differenziert. Nur online-change kompatible TcCOM werden zusammengefasst.

Im Folgenden ist zu sehen, dass die Versionen 0.0.0.1 und 0.0.0.3 bzw. 0.0.0.4 und 0.0.0.5 kompatibel für Online-Change sind. Jedoch nicht 0.0.0.3 zu 0.0.0.4 oder 0.0.0.2.



Um die Kompatibilität der Data Areas besser gewährleisten zu können, ist es zum Beispiel möglich, die Parameter, Block I/O, ContState und DWork eines Modells nicht in einer internen Data Area zu halten, sondern als Module Parameter. Damit sind nur die Inputs und Outputs als Data Area für die Kompatibilität der TcCOM-Versionen relevant.



2. Nutzen Sie zur Durchführung des Online-Change im TwinCAT XAE das **tree item TcCOM Modules** und navigieren Sie zum Tab **Online Changeable Objects**.

3. Wählen Sie im Drop-down-Menü unter **Online Version** eine Version Ihrer Wahl aus (es werden nur kompatible Versionen angezeigt).
4. Machen Sie einen Rechtsklick auf die Zeile des Objekts und wählen Sie **Apply changed online object versions**, um die neue Version des TcCOM zu aktivieren.

⇨ Details finden Sie auch in dieser TwinCAT C++ Dokumentation.

4.7.6 Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts

Im Bereich *TC TcCom Interface* konfigurieren Sie, auf welche Art und Weise auf Daten bestimmter Variablen-Gruppen zugegriffen werden kann. Je nach Anforderung kann der ADS-Zugriff und der Typ des Prozessabbildes konfiguriert werden. Diese Einstellung beeinflusst, in welcher Weise eine Variable in einer Gruppe mit anderen Prozessabbildern in der TwinCAT-Entwicklungsumgebung verknüpft werden kann und Daten ausgetauscht werden können.

Variablen-Gruppen

Abhängig vom Simulink®-Modell gibt es neben den Ein- und Ausgangsvariablen mehrere Gruppen interner Variablen.

Folgende Gruppen können konfiguriert werden:

Gruppe	Beschreibung	Benennung der DataArea (default)	Benennung der DataArea (Option „classic“)
Input	Modell-Eingänge	<ModelName>_U	Input

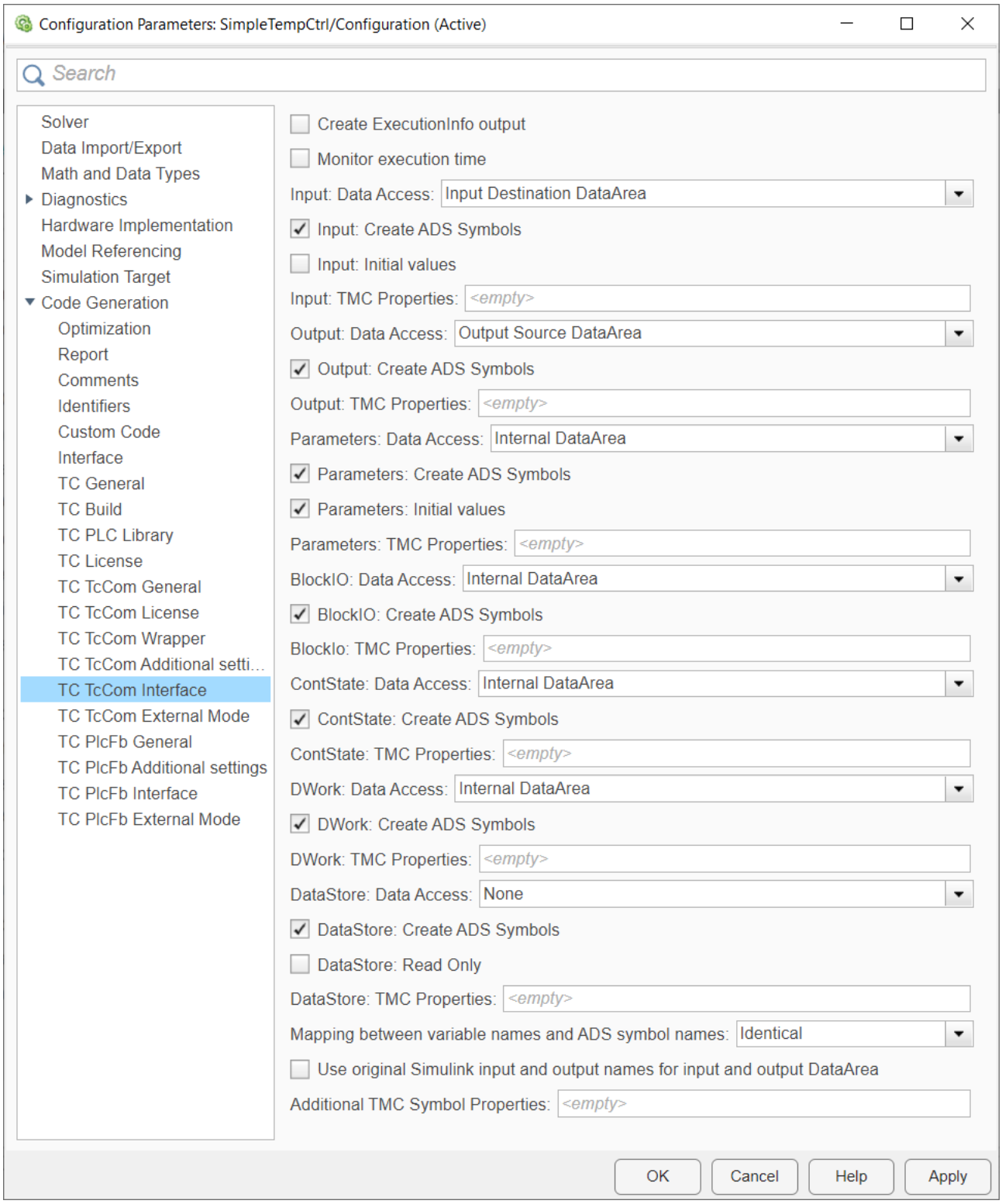
Gruppe	Beschreibung	Benennung der DataArea (default)	Benennung der DataArea (Option „classic“)
Output	Modell-Ausgänge	<ModelName>_Y	Output
BlockIO	Globale Ausgangssignale von Simulink®-Blöcken: Interne Signale, für die ein "Testpunkt" festgelegt wurde.	<ModelName>_B	BlockIO
Parameter	Modellspezifische Parameter: Parameter von Simulink®-Blöcken, die *tunable* sind.	<ModelName>_P	Model-Parameters
ContState	Kontinuierliche Zustandsvariablen	<ModelName>_X	ContStates
DWork	Zeitdiskrete Zustandsvariablen	<ModelName>_DW	DWork
DataStore	Data Store Memory	<ModelName>_DW_<DataStoreName>	<ModelName>_DW_<DataStoreName>

Unter TC TcCom Interface kann mit der Option **Mapping between variable names and ADS symbols** die Benennung der DataAreas beeinflusst werden. Standardmäßig erhalten sie den gleichen Namen, wie die zugehörige C++-Variable. Dieser wird vom Simulink Coder™ vorgegeben. Mit der Einstellung "Classic" werden die Namen abgekürzt auf den der Variablengruppe, wie man es aus früheren TE1400-Versionen 1.2.x kennt. Die Eingänge sind dann bspw. in der DataArea "Input" zusammengefasst und nicht in "<ModelName>_U".

TcCOM mit mehreren Task-Kontexten

Wird ein TcCOM erzeugt, welches mehr als einen Task-Kontext besitzt (siehe Multitask, Concurrent Execution und OpenMP [▶ 164]), werden die DataAreas automatisch aufgetrennt. So existieren bspw. mehrere Inputs oder Dwork DataAreas.

Konfigurationsoberfläche in Simulink®

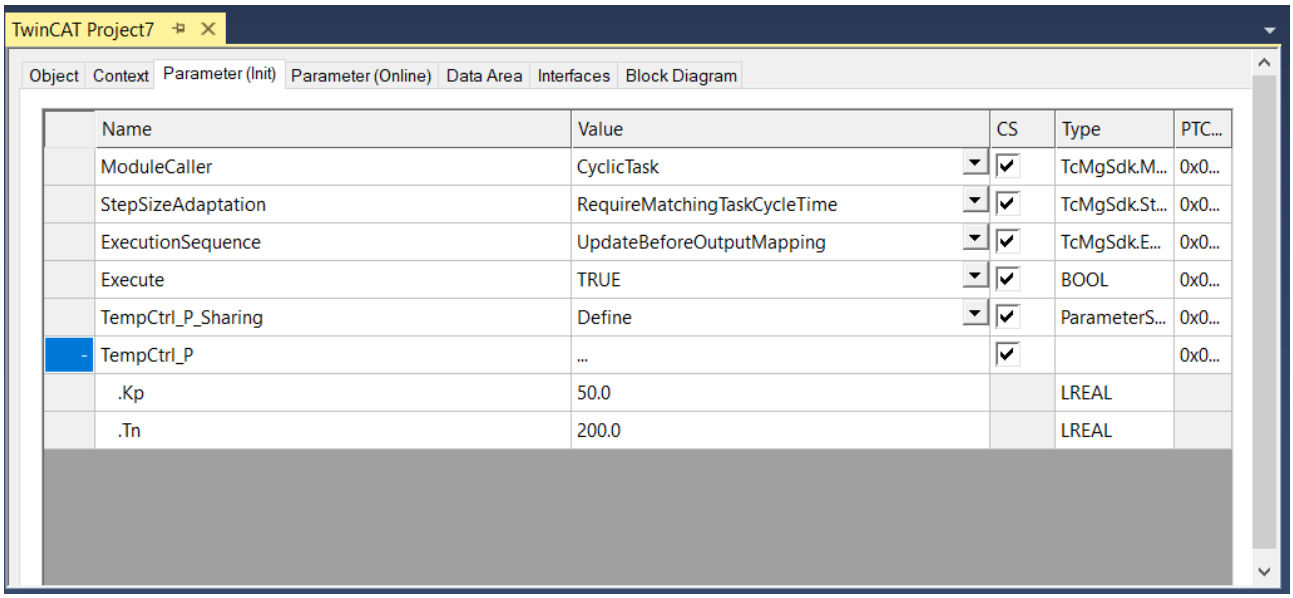


Zu jeder oben benannten Variablen-Gruppe kann individuell der *Data Access* definiert werden. Auswahlmöglichkeiten zu DataAreas und Modul-Parametern sind in folgender Tabelle zusammengefasst:

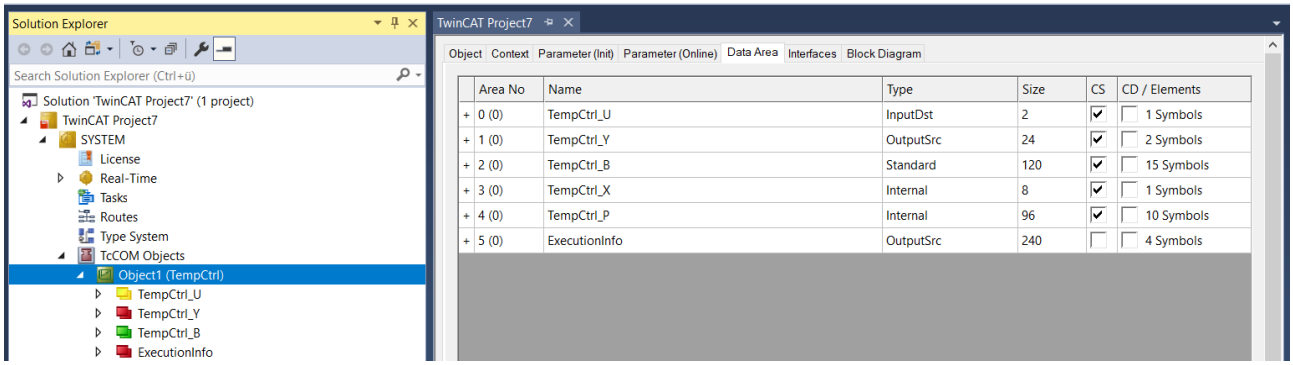
Data Access	Zugriff per ADS	Zugriff per Mapping	Zugriff per Data Pointer
None	Nein	Nein	Nein
Modul Parameter	Ja	Nein	Nein
Input Destination DataArea	Ja	Ja (mit Output Source)	Ja
Output Source DataArea	Ja	Ja (mit Input Destination)	Ja

Data Access	Zugriff per ADS	Zugriff per Mapping	Zugriff per Data Pointer
Internal DataArea	Ja	Nein	Nein
Standard DataArea	Ja	Nein	Ja
Retain Source DataArea	Ja	Ja (mit Retain Handler)	Nein

Modul Parameter sind im Gegensatz zu DataAreas nicht für den zyklischen (Prozess-)Datenaustausch mit anderen Modulen oder I/Os gedacht. Sie werden im Allgemeinen asynchron gelesen oder geschrieben, z. B. per ADS. Die Parameter unter dem Reiter **Parameter (Init)** haben einen im Projekt konfigurierbaren Initialwert, der beim Start der TcCOM-Instanz geschrieben wird. Die Parameter unter **Parameter (Online)** dagegen, haben keinen konfigurierbaren Initialwert und dienen typischerweise der Überwachung interner Zustände.



DataAreas sind auf der TcCOM-Instanz unter dem Tab **DataArea** zu finden. Der genaue Typ der DataArea wird ebenfalls hier angezeigt. Internal DataAreas werden in der Entwicklungsumgebung nicht als Prozessdatenabbild angelegt.



Eigenschaften der Parametrierung

Der **Zugriff per ADS** ist in der Regel **ReadOnly**. Nur die Gruppen Inputs und Parameter können per ADS auch beschrieben werden.

Unabhängig davon kann zu jeder Gruppe eingestellt werden, ob für die jeweilige Gruppe ADS-Symbolinformationen (**Create ADS Symbols**) erstellt werden sollen. Wenn keine ADS-Symbolinformationen vorhanden sind, kann per ADS nur noch über IndexGroup und IndexOffset auf die Daten zugegriffen werden.

Über die Option **„Input: Initial values“** erhalten Sie zusätzlich zur DataArea (für den zyklischen Datenaustausch) einen Eintrag unter "Parameter (Init)". So können Initialwerte für Eingänge realisiert werden. Diese Werte gelten dann, wenn die entsprechenden Eingänge nicht verknüpft wurden. Ohne diese Parameter sind nicht verknüpfte Eingangswerte immer 0.

Über die Option **„Parameter: Initial values“** werden die Paramater des Simulink®-Modells zusätzlich zur DataArea-Einstellung ebenfalls als Modul-Parameter (Struktur <ModelName>_P) angelegt.

Sonderfall: DataArea für Modell-Parameter

Wird das Code Interface auf Reusable function [▶ 196] gestellt (Default), dann wird keine DataArea für Modell-Parameter erstellt, da die Parameter für diesen Fall global gehalten werden und somit zwischen mehreren Instanzen geteilt werden. Wenn "Parameter: Initial values" ausgewählt ist, wird in diesem Fall unter **Parameter (Init)** neben dem Eintrag für die Modell-Parameter ein weiterer Parameter "<ModellParameterName>_Sharing" angelegt. Dieser Parameter kann nur bei einer Instanz dieser TcCOM-Modulklasse auf "Define" eingestellt sein.

Alle weiteren Instanzen müssen "Inherit" verwenden. Da sich alle Instanzen in diesem Fall die Parameter teilen, arbeiten dann alle mit dem beim "Definer" eingestellten Parametersatz. Die Modell-Parameter-Einstellungen der anderen Instanzen werden ignoriert.
Siehe auch Parametrierung mehrerer Modulinstanzen [▶ 196].

Monitor Execution Time: Erstellt einen Modul-Parameter unter **Parameter (Online)** und im Blockdiagramm (rechte Seite). Über diesen Parameter kann die Ausführungszeit des TcCOM ausgelesen werden. Die Messung wird bei jedem Aufruf des TcCOM aktualisiert, sodass bspw. die Execution Time über das TwinCAT Scope präzise für jeden Aufruf verfolgt werden kann.

Create ExecutionInfo output: Erstellt eine zusätzliche Output Source DataArea mit Informationen zum Modulaufruf, siehe Exception Handling [▶ 225].

DataStores: Variablen aus einem Data Store Memory Block werden standardmäßig im DWork abgelegt. Sie können eine zusätzliche Adressierung dieser Variablen anlegen, wenn Sie für DataStores eine DataArea erzeugen. Standardmäßig wird keine DataArea angelegt. Beispiel siehe Geteilter Speicher zwischen TcCOM-Instanzen [▶ 154].

Weiterführende Beschreibungen:

- Aufbau eines TcCOM: Beschreibung der TcCOM Eigenschaften
- Eigenschaften von DataAreas
- Nutzen einer RetainDataArea mit dem NOV-RAM: RetainDataArea
- DataPointer können Sie bspw. über die TC Module Input- und TC Module Output Blöcke verwenden, siehe Geteilter Speicher zwischen TcCOM-Instanzen [▶ 154].

4.7.6.1 Best Practice: Zugriff auf Daten des TcCOM

Einführung

Im Bereich Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts [▶ 147] wurde beschrieben, wie man den Datenzugriff auf ein TcCOM unter TC TcCom Interfaces in den Konfigurationsparametern eines Simulink®-Modells einstellen kann. Im Folgenden werden **Vor- und Nachteile** und auch **Anwendungsszenarien** sowie **nützliche Hinweise** gegeben, wann welcher Typ von Datenzugriff auf ein TcCOM sinnvoll sein kann.

Zugriff per ADS

Eigenschaften

- On-demand Datenaustausch
- Asynchrone Kommunikation
- Threadsafe
- Zeitpunkt des Lesens oder Schreibens nicht determiniert

Use Cases

- Lesender oder schreibender Zugriff von außerhalb der TwinCAT-Laufzeit, z.B. von einer HMI. Lesen und schreiben von Modell-Parametern oder lesen von internen Signalen.

- Zugriff von innerhalb der TwinCAT-Laufzeit auf das TcCOM, wenn das schreibende Software-Modul (anderes TcCOM oder SPS) nicht im selben Task-Kontext aufgerufen wird. Somit erfolgt der Datenaustausch thread-safe. Beispielsweise nutzbar, wenn ein Software-Modul Parametersätze an unterschiedliche Software-Module in diversen Kontexten liefert.

Hinweise

- Sollen mehrere Parameter gleichzeitig verändert oder gelesen werden, sollte ein ADS-Summenkommando verwendet werden. Ansonsten kann nicht garantiert werden, dass separate ADS Lese- oder Schreibbefehle allesamt konsistent in einem Task-Zyklus abgearbeitet werden.
- Soll der Datenzugriff über das Netzwerk über OPC UA und nicht per ADS erfolgen, gelten ebenfalls diese Eigenschaften.

Zugriff per Mapping

Eigenschaften

- Zyklischer Datenaustausch
- Threadsafe
- Zeitpunkt des Datenaustausches determinieret

Use Cases

- Zyklischer Datenaustausch zwischen unterschiedlichen Software-Modulen, insbesondere für Daten die sich jeden Task-Zyklus ändern.

Hinweise

- Auch nicht veränderte Werte werden in jedem Task-Zyklus kopiert. Daher ist es ratsam, die Input- und Output Mapping DataAreas klein zu halten und auf das wesentliche zu beschränken. Große Simulink®-Bus-Strukturen als Input, bei denen sich nur wenige Elemente zyklisch verändern, sollten hier vermieden werden.
- Der Zeitpunkt des Datenaustausches ist determiniert und kann in Simulink® unter Cofiguration Parameters -> TC TcCom Additional settings -> Default execution sequence spezifiziert werden.
- Modell-Parameter als Input Destination DataArea zu definieren ist nur dann sinnvoll, wenn sich die Parameter zyklisch ändern (wobei hier zunächst geprüft werden sollte, ob der Parameter sich nicht besser als Modell-Input abbilden lässt) oder die Anzahl der Parameter so gering ist, dass der Overhead des zyklischen Kopierens nicht stört.

Zugriff per Data Pointer

Eigenschaften

- On-demand Datenaustausch
- Nicht Threadsafe
- Zeitpunkt des Lesens oder Schreibens determiniert (geteilter Speicherbereich)
- Lokale Kopie der Daten in jeder Instanz
- Flexible Verknüpfung von DataArea und Data Pointer im XAE

Use Cases

- Gemeinsamer Speicherbereich von mehreren TcCOM, beispielsweise für Variablen im DataStore. TcCOM-Instanzen können sich über Data Pointer Daten miteinander in einem gemeinsamen Speicherbereich teilen.

Beispiel für Data Pointer

Ein Beispiel finden Sie hier: Geteilter Speicher zwischen TcCOM-Instanzen [▶ 154].

Exported Global/Imported Extern

Eigenschaften

- On-demand Datenaustausch

- Nicht Threadsafe
- Zeitpunkt des Lesens oder Schreibens determiniert (geteilter Speicherbereich)
- Keine lokalen Kopien der Daten in jeder Instanz
- Nur Module, die im selben Treiber gebündelt wurden [▶ 137], können auf die globale Variable zugreifen.

Use Cases

- Gemeinsamer Speicherbereich für mehrere TcCOM. Auch für große Variablen geeignet.

Beispiel für Exported Global/Imported Extern

Ein Beispiel finden Sie hier: Geteilter Speicher zwischen TcCOM-Instanzen [▶ 154].

Spezialfall: Interaktion über den TcCOM-Wrapper-FB

Der TcCOM-Wrapper-FB (siehe Anwenden des TcCOM-Wrapper-FB [▶ 214]) vereinfacht die Interaktion zwischen SPS und TcCOM. Dieser FB ermöglicht mit geringem Programmieraufwand in erster Linie die zyklische Ausführung des verknüpften TcCOM-Objektes aus dem SPS-Code, inklusive des Austauschs der Ein- und Ausgangsdaten. Er ermöglicht aber auch einen einfachen azyklischen Zugriff auf die Parameter des Objektes und bei Bedarf einen flexiblen Zugriff auch auf die DataAreas.

Der Wrapper stellt dazu einerseits das ITcADI-Interface (siehe ADI Interface [▶ 218]) und andererseits (optional) Properties am Funktionsbaustein (siehe FB properties [▶ 217]) bereit. Der TcCOM-Wrapper-FB sollte im selben Kontext wie das TcCOM aufgerufen werden.

Properties des Funktionsbausteins

- Zugriff nur auf Modul Parameter
- Kein Zugriff auf DataAreas. Hinweis: Modell Parameter werden standardmäßig sowohl als DataArea als auch als Modul Parameter angelegt.
- Einfacher Zugriff auf Modul Parameter per Name
- Nicht Threadsafe

ITcADI-Interface

- Effizienter und flexibler Zugriff auf alle DataAreas (auch Teilbereiche einer DataArea)
- Zugriff nur über DataArea Nummer und ByteOffset
- Keine Typ-Informationen (type cast ist durch den Nutzer zu realisieren)
- Jeden Task-Zyklus muss der Pointer neu geholt und auch wieder freigegeben werden
- Kein Zugriff auf Modul Parameter
- Nicht Threadsafe

Änderungen persistieren

Oben wurde beschrieben, wie Daten zur TwinCAT-Laufzeit in einer TcCOM Instanz verändert werden können. Wird TwinCAT neu gestartet, gehen diese Änderungen verloren, wenn nicht weitere Maßnahmen zur Wiederherstellung getroffen wurden. Standardverhalten eines TcCOM ist, dass bei Start einer TcCOM-Instanz die Parametrierung entsprechend seiner **Startup Values** vorgenommen wird.

Darüber hinaus können remanente Variablen definiert werden, welche ihren Wert über die übliche Programmlaufzeit hinaus behalten. Sie können remanente Variablen als **RETAIN**-Variablen oder noch strenger als **PERSISTENT**-Variablen deklarieren.

Nähere Informationen zu remanenten Daten (Persistent und Retain) in der TwinCAT PLC: Link

Im folgenden werden drei Wege beschrieben, wie Werte nach Neustart von TwinCAT wieder hergestellt werden können.

1) Startup values

Siehe Parametrierung einer Modulinstanz [▶ 190] zur Differenzierung zwischen Online, Prepared, Default und Startup Values.

TwinCAT 3 XAE notwendig

Die Veränderung von Startup Values erfolgt in der TwinCAT Konfiguration, also im Engineering. Änderungen können hier eingetragen werden. Die Änderung muss aber kompiliert und auf das Laufzeit-System heruntergeladen werden.

Werden Online Values während der Laufzeit verändert, können Sie die aktuellen Online-Werte z. B. via ADS lesen. Dies kann z.B. mit dem TE1410 TwinCAT Interface for MATLAB®/Simulink® erfolgen. Die gelesenen Werte können dann als neue Startup Values in die TwinCAT Konfiguration eintragen werden.

Nutzen Sie dazu z.B. das Automation Interface Beispiel:

```
TwinCAT.ModuleGenerator.Samples.Show('AutomationInterface')
```

In dem Beispiel wird ein Live Script verwendet, um unterschiedliche Funktionen zu zeigen. Der Abschnitt *Read and write parameter online values via ADS* beschreibt das Lesen und Schreiben von Online-Werten von einem TcCOM. Der Abschnitt *Read and change parameter startup value* beschreibt, wie Sie Startup Values lesen und auch schreiben können.

Der Eintrag der neuen Startup Values erfolgt in der TwinCAT Konfiguration auf dem Engineering System. Um die Änderungen auf dem Laufzeitsystem verfügbar zu machen, müssen Sie die Konfiguration auf dem System aktivieren. Auch dies können Sie mit dem Automation Interface realisieren (`sysManager.ActivateConfiguration`). Sie können entweder das System per Automation Interface direkt neu starten (`sysManager.StartRestartTwinCAT`), oder Sie können ohne Neustart das System laufen lassen. Beim nächsten Start von TwinCAT sind die Startup Values gesetzt und das TcCOM fährt mit den neuen Startup Values hoch.

2) Retain Data

Unter TC TcCom Interfaces [▶ 147] können Sie die Modell Parameter (und auch andere Gruppen) als DataArea vom Typ Retain Source DataArea anlegen. Sie können dann einen Retain Handler in der TwinCAT Konfiguration anlegen und mit der DataArea verbinden. Details siehe NOV-RAM und Retain Handler.

NOV-RAM notwendig

Um mit dem Retain Handler zu arbeiten, benötigen Sie einen IPC/CX mit eingebautem NOV-RAM. Die Nutzung einer EtherCAT-Klemme mit NOV-RAM (z.B. EL6080) wird nicht unterstützt.

3) Persistent Data

Ohne TwinCAT XAE und ohne NOV-RAM können Sie auf persistente Daten in der TwinCAT 3 SPS zurückgreifen.

TwinCAT 3 SPS Laufzeit notwendig

Um persistente Daten nutzen zu können, müssen Sie in der SPS-Variablen als persistent deklarieren.

Persistente Daten können nur in der SPS angelegt werden. Nutzen Sie daher die SPS, um die Daten zu definieren, die sie persistieren möchten. Arbeiten Sie z.B. mit dem TcCOM-Wrapper-FB [▶ 214], um Daten aus dem TcCOM auszulesen und über die persistente SPS Variable zu persistieren. Erstellen Sie eine Zustandsmaschine für das Aufstarten des TcCOM Objekts nach einem TwinCAT Restart, um die persistierten Daten wieder in das TcCOM Objekt hineinzuschreiben, bevor es wieder im Code aufgerufen wird.

4.7.7 Geteilter Speicher zwischen TcCOM-Instanzen

In machen Anwendungen kann es vorteilhaft sein, dass sich TcCOM-Instanzen einen Speicherbereich teilen, sodass bestimmte Strukturen/Variablen einmal in einem Objekt definiert werden und alle anderen Objekte auf diese Speicherstelle referenzieren.

Um dies zu erreichen, können Sie in TwinCAT zwei unterschiedliche Wege verfolgen:

- Verknüpfen von TcCOM-Instanzen in TwinCAT mit Data Pointern [▶ 155]
- Nutzen von globalen Variablen (Exported Global/Imported Extern) [▶ 159]

Im weiteren Verlauf dieses Kapitels werden beide Wege beschrieben.

Während Sie mit Data Pointern flexibler sind in der Verknüpfung der Daten, haben Sie bei der Nutzung von globalen Variablen den Vorteil, dass die Struktur/Variable nur einmal im Speicher vorhanden ist. Bei Data Pointern wird in jeder TcCOM-Instanz eine lokale Kopie der Daten gehalten.

HINWEIS

Datenübertragung nicht Threadsafe

Nutzen Sie Data Pointer und Globale Variablen mit Vorsicht. Der Datenaustausch erfolgt nicht Threadsafe. Deshalb empfehlen wir dringend, alle beteiligten TcCOM-Instanzen im selben Task-Kontext zu betreiben.

Verknüpfen von TcCOM-Instanzen in TwinCAT mit Data Pointern

Im Folgenden wird beschrieben, wie Sie gezielt per Data Pointer einen gemeinsamen Speicherbereich zwischen TcCOM-Instanzen nutzen können.

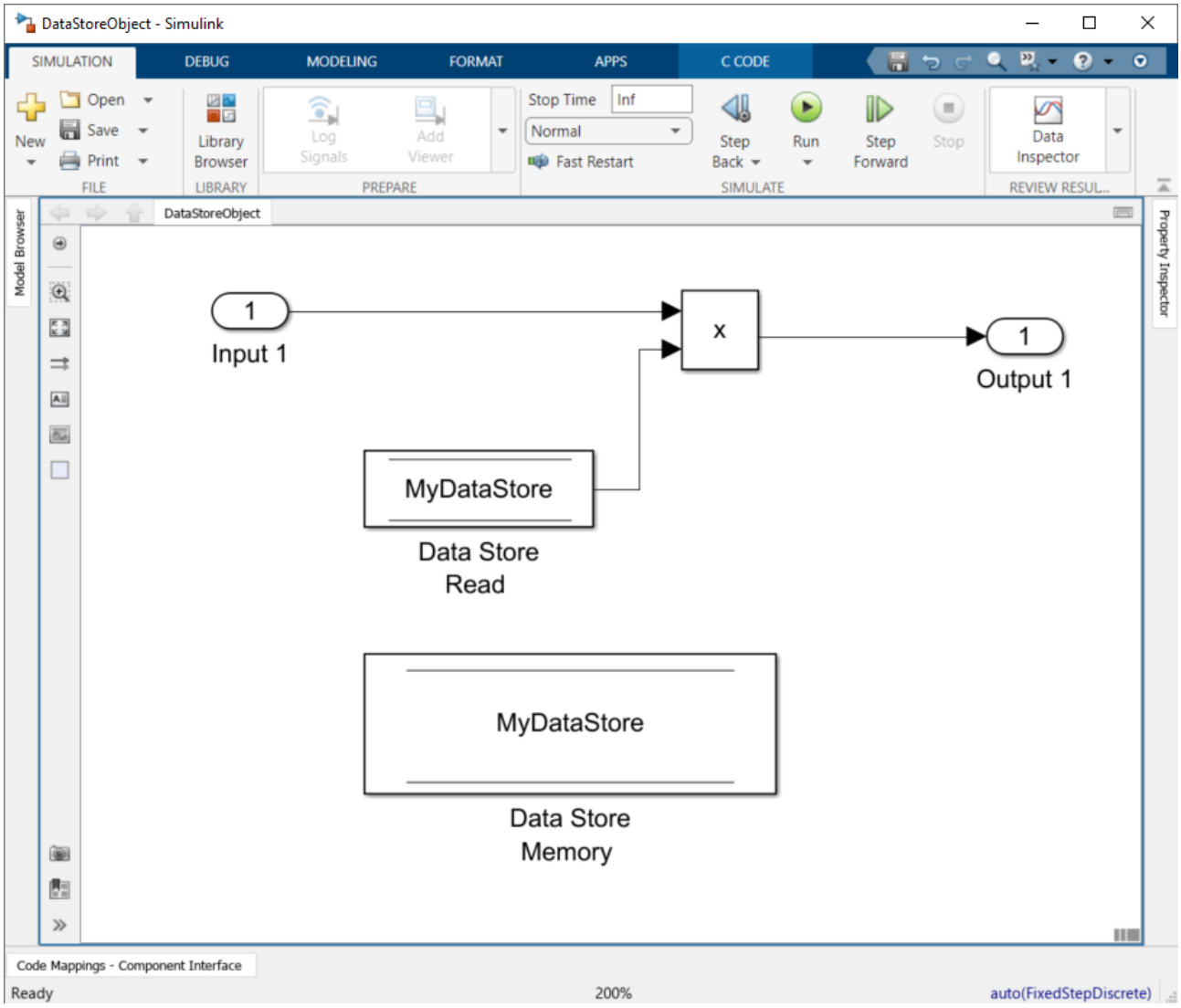
Wie im Bereich Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts [▶ 147] beschrieben, können Sie DataAreas per DataPointer erreichbar machen. Per DataPointer erreichbar sind Input Destination DataArea, Output Source DataArea sowie Standard DataArea. Sie können prinzipiell jede Variablen-Gruppe, Modell-Parameter, DWork, BlockIO usw. als Standard DataArea anlegen und damit als Data Pointer erreichbar machen.

Zielstellung

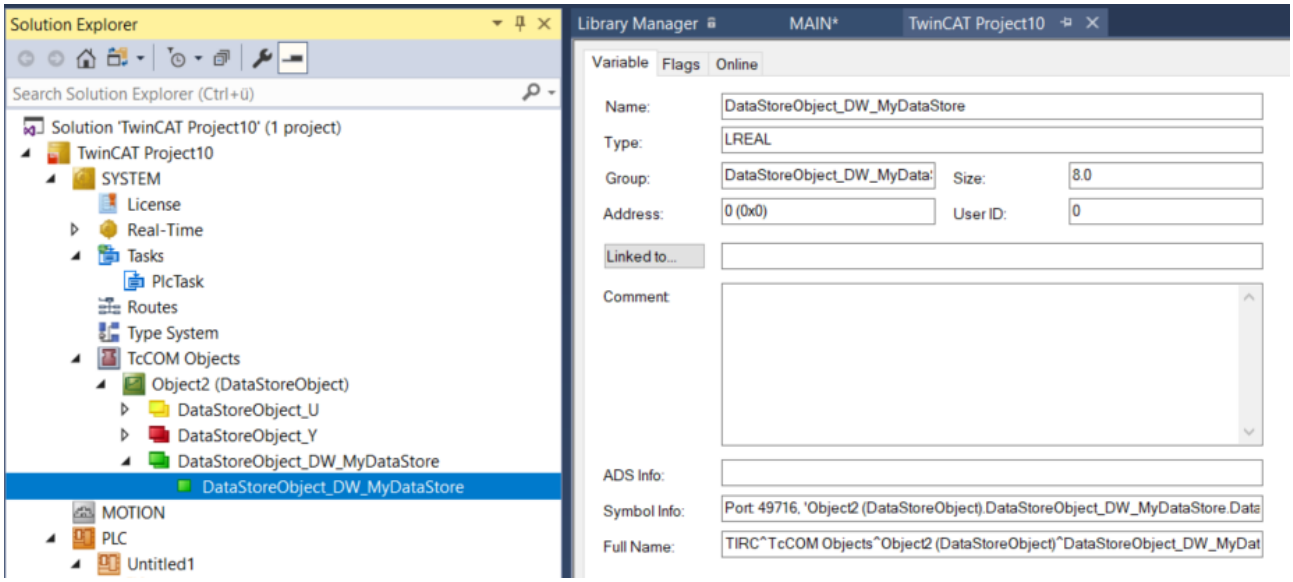
Im Folgenden wird gezeigt, wie Sie gezielt einen Teilbereich von Variablen per DataPointer zugänglich machen und nicht gleich den ganzen Variablen-Gruppen-Bereich. Das Grundkonzept basiert auf der Nutzung von Data Store Memory Blöcken in Simulink®.

Modell mit DataStore

In einem Modell *„DataStoreObject“* wird ein Data Store Memory Block angelegt. In diesem Beispiel mit nur einer einzigen Variable vom Typ double. Diese Variable wird mit einem Data Store Read ausgelesen, mit einem Input multipliziert und auf einen Ausgang gegeben.



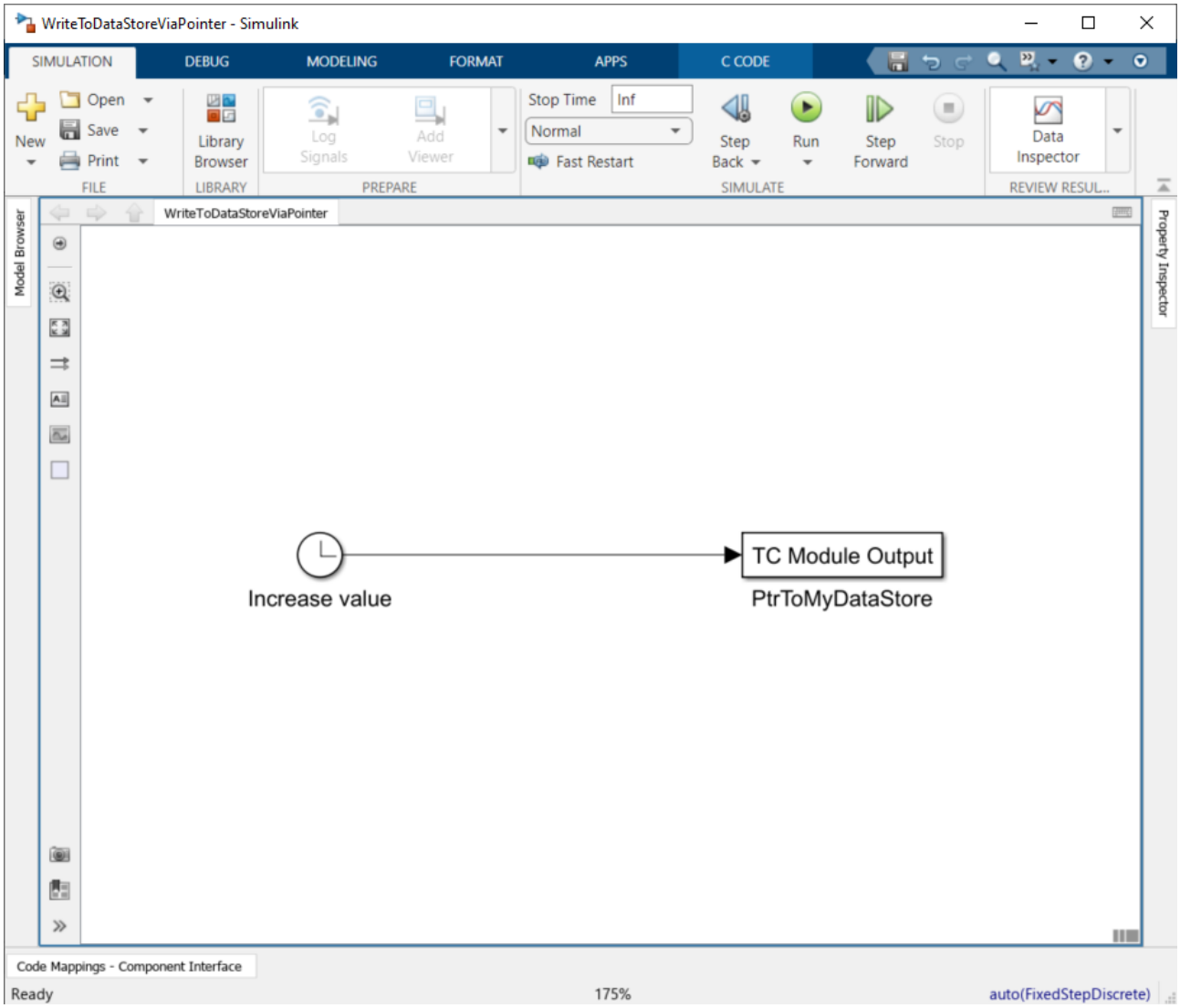
Unter TC TcCom Interface wird die Option *DataStore: Data Access* von None auf *Standard DataArea* geändert und das Simulink®-Modell in ein TcCOM übersetzt. Wenn das Objekt in TwinCAT instanziiert wird, sehen Sie die folgende Darstellung:



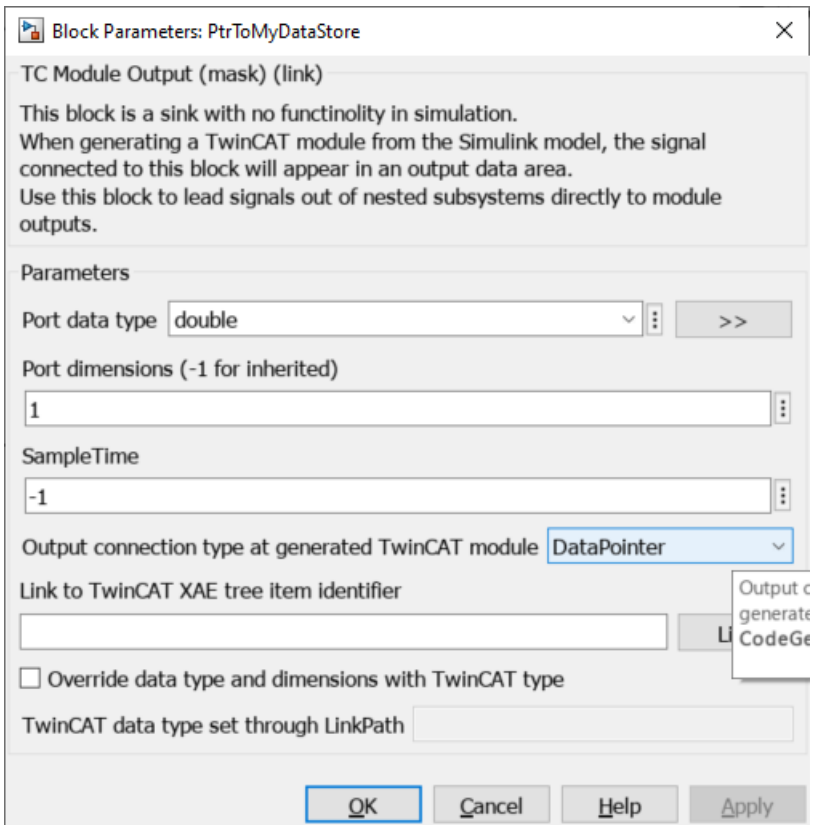
Die DataStore DataArea wird nun im Prozessabbild angezeigt und enthält eine Variable vom Typ LREAL, auf welche per DataPointer zugegriffen werden kann.

Modell mit DataPointer

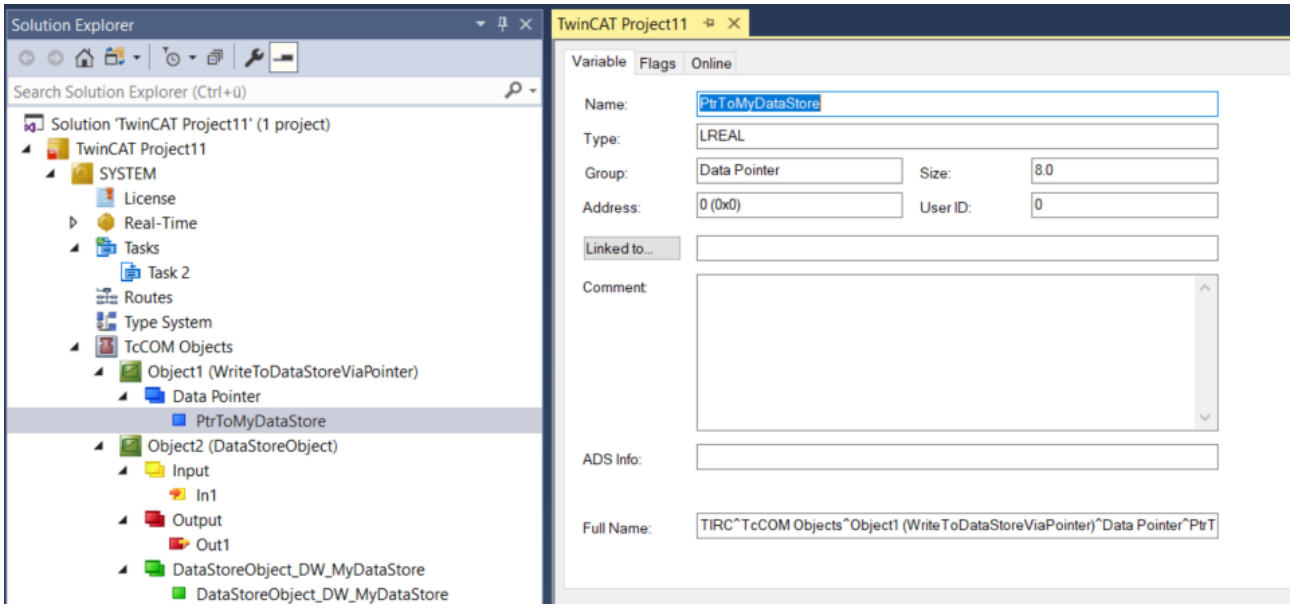
In einem zweiten Simulink®-Modell wird durch eine Clock ein double-Wert erzeugt und dieser auf den TC Module Output [▶ 113] gegeben.



In der Konfiguration des TC Module Output wird der connection type auf DataPointer sowie der Datentyp auf double gestellt.



Wenn dieses Modell in ein TcCOM-Objekt übersetzt wird, sehen Sie in TwinCAT die folgende Darstellung:

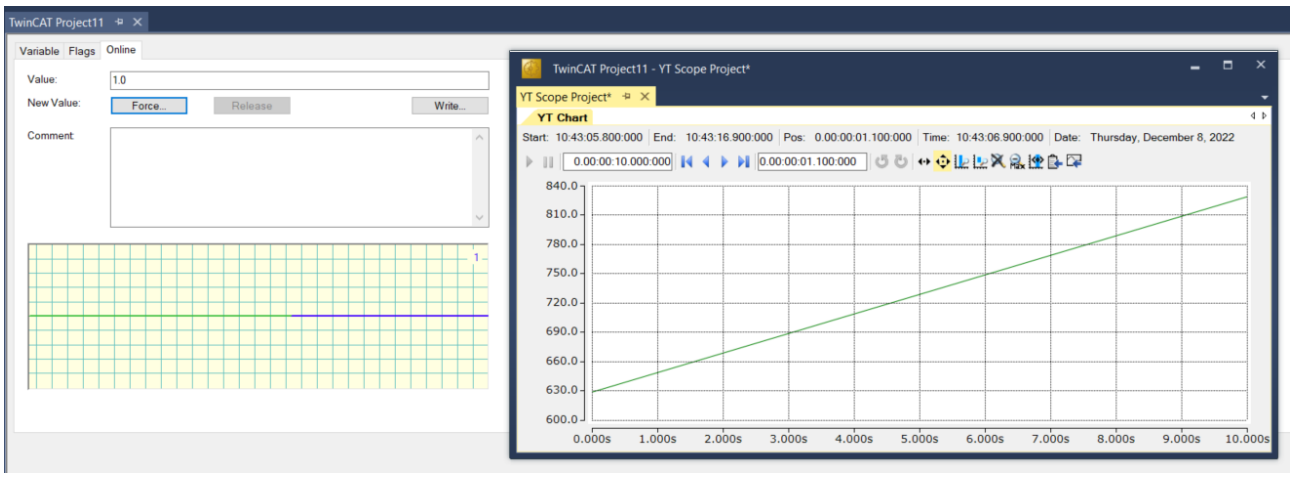


Unter Data Pointer wird nun der Name des TC Module Output „PtrToMyDataStore“ vom Typ LREAL angezeigt. Diese kann nun mit Doppelklick auf die DataStoreObject_DW_MyDataStore verknüpft werden.

Beide TcCOM-Instanzen sollten in derselben Task aufgerufen werden, da die Kommunikation nicht Threadsafe ist.

Wenn unterschiedliche Tasks verwendet werden, müssen Sie als Anwender dafür Sorge tragen, dass Daten konsistent sind und zu passenden Zeitpunkten gelesen oder geschrieben werden.

Das Verhalten des hier aufgeführten Beispiels wird in untenstehender Grafik gezeigt.
Wird der Eingang In1 des DataStore Modells manuell auf 1 gesetzt und der Ausgang Out1 über das TwinCAT Scope beobachtet, sieht man eine ansteigende Gerade. Das bedeutet, dass per Pointer der ansteigende Wert aus der Clock in den DataStore des DataStoreObject-TcCOM geschrieben wird.



Lesender Zugriff per TC Module Input

Lesender Zugriff auf eine DataArea kann per DataPointer über den TC Module Input realisiert werden.

Lokale Datenkopie in jedem Modul

Der Simulink Coder™ erzeugt den C/C++ Code in der Art, dass für die TC Module Input und TC Module Output lokale Variablen angelegt werden und somit jede Modulinstanz eine lokale Kopie der Daten enthält. Entsprechend erhöht sich der Speicherbedarf des Projekts mit jeder Instanz.

Nutzen von globalen Variablen (Exported Global/Imported Extern)

Im Folgenden wird beschrieben, wie Sie in Simulink® über die Storage Classes eine globale Variable im TcCOM anlegen können und diese in anderen TcCOM referenzieren.

Zielstellung

Es wird ein Simulink® Parameter GlobalParam angelegt, welcher eine Struktur von 3 Elementen enthält. Ziel ist es, diese Struktur über ein TcCOM-Objekt zu definieren und weitere TcCOM-Objekte zu instanziieren, die auf diese Struktur über geteilten Speicherbereich zugreifen, sodass Änderungen im definierenden TcCOM direkt in allen verbundenen TcCOM-Instanzen ankommen.

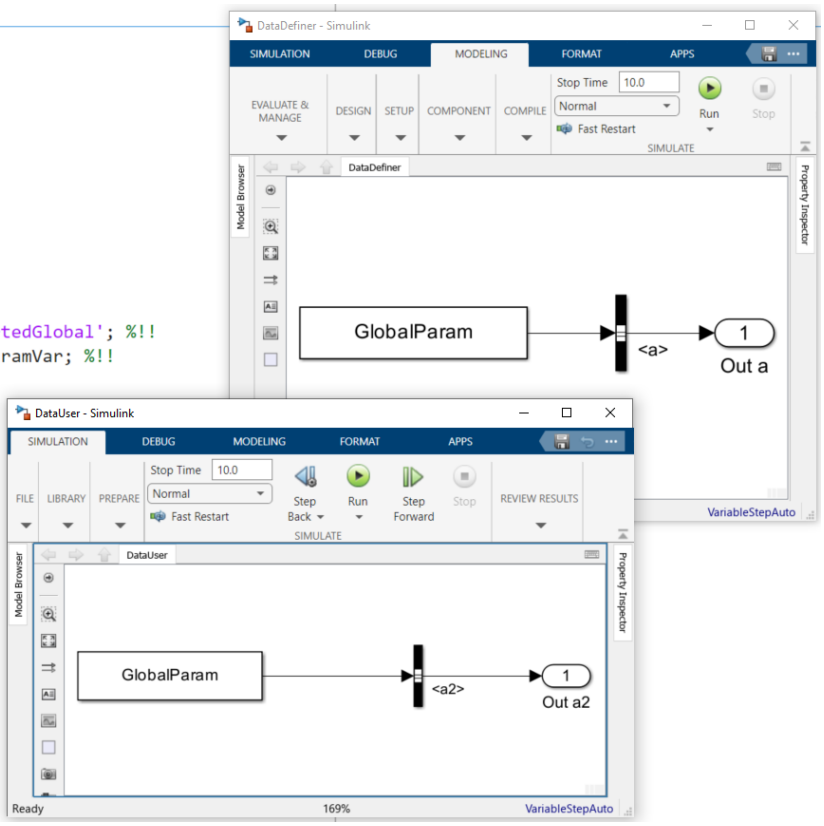
Modellierung in Simulink®

Es werden zwei Simulink®-Modelle erstellt:

1. Modell „DataDefiner“
2. Modell „DataUser“

Des Weiteren wird ein Simulink®-Bus definiert und als Simulink®-Paramater namens `GlobalParam` angelegt. Das Modell DataDefiner bestimmt am Ende den Inhalt von GlobalParam, während die Modelle DataUser den Inhalt nur lesen.

```
%% Global parameter variable
% parameter structure and appropriate bus
ParamStruct = struct('a',5,'a1',5,'a2',6);
struct2bus(ParamStruct,'ParamBus');

% global variable name
globalParamVar = 'GlobalParam';

% global parameter object
GlobalParam = Simulink.Parameter;
GlobalParam.Value = ParamStruct;
GlobalParam.Complexity = 'real';
GlobalParam.CoderInfo.StorageClass = 'ExportedGlobal'; %!!
GlobalParam.CoderInfo.Identifier = globalParamVar; %!!
GlobalParam.CoderInfo.Alignment = -1;
GlobalParam.Description = '';
GlobalParam.DataType = 'Bus: ParamBus';
GlobalParam.Min = [];
GlobalParam.Max = [];
GlobalParam.DocUnits = '';
```



Beachten Sie, dass die Definition der Storage Class von GlobalParam im Folgenden für den DataDefiner und DataUser unterschiedlich gesetzt werden. Für den DataDefiner wird als Storage Class **Exported Global** gesetzt, während für den DataDefiner die Storage Class auf **Imported Extern** gestellt wird.

Das folgende Skript bündelt die beiden beschriebenen Simulink®-Modelle in einem Treiber, vgl. Bündelung mehrerer Modelle in einem TwinCAT-Treiber [▶ 137].

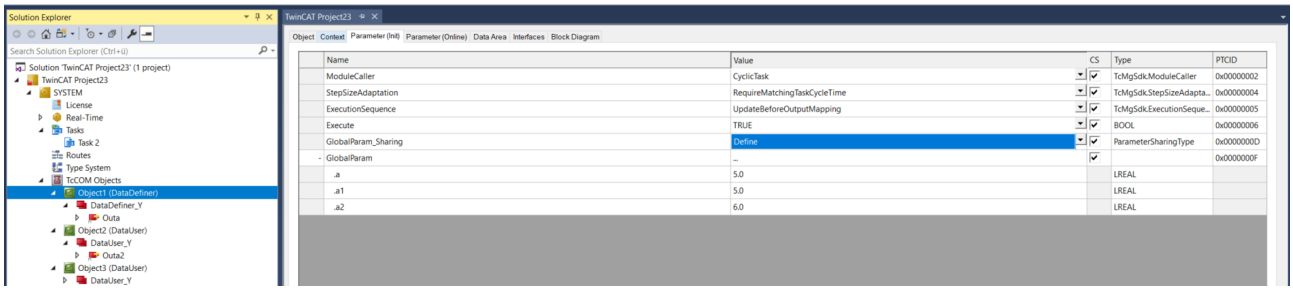
```
thisDir = fileparts(mfilename("fullpath"));

% model names
mdlNames = ["DataDefiner","DataUser"];
codeDirectories = fullfile(thisDir,strcat(mdlNames,'_tcgrt'));
isParamDefiner = [true,false];

% instantiate project configuration
projCfg = TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',fullfile(thisDir,'TcCppProj','DataS
haringModules.vcxproj'));

regenerate = false;
for i=1:length(mdlNames)
   % generate code for the models (only if the code directory doesn't exist) -
> remove the directory to rebuild specific models
   if regenerate || ~isfolder(codeDirectories(i))

      % load the model and apply basic settings
      mdl = load_system(mdlNames(i));
      set_param(mdl,'SystemTargetFile','TwinCatGrt.tlc');
      set_param(mdl,'CodeInterfacePackaging','C++ class');
      set_param(mdl,'TcCom_TcComWrapperFb','off');
      set_param(mdl,'TcProject_Generate','off');
      set_param(mdl,'SolverType','Fixed-step');

      % adapt the storage class of the shared parameter structure
      if isParamDefiner(i)
         GlobalParam.CoderInfo.StorageClass = "ExportedGlobal";
      else
         GlobalParam.CoderInfo.StorageClass = "ImportedExtern";
      end

      % generate code and close the model
      slbuild(mdl);
      close_system(mdl,0);
   end
```

```
   % add the class export configuration to the "global" project export configuration
   mdlProjCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(codeDirectories(i));
   clsCfg = mdlProjCfg.ClassExportCfg{1};
   projCfg.AddClassExportConfig(clsCfg)
end

% generate and build the project
TwinCAT.ModuleGenerator.ProjectExporter(projCfg);
```

Konfiguration in der TwinCAT XAE

✓ Erstellen Sie in der TwinCAT XAE **eine** Instanz des DataDefiners (mehr sind nicht erlaubt!). Hingegen können Sie beliebig viele DataUser-Instanzen erzeugen.

1. Stellen Sie unter **Parameter (Init)** > **Parameter GlobalParam_sharing** für den DataDefiner auf „Define“.
2. Stellen Sie unter **Parameter (Init)** > **Parameter GlobalParam_sharing** für alle DataUser auf „Inherit“.
3. Erstellen Sie eine Task.
4. Weisen Sie diese Task allen erstellten Instanzen zu.



5. Aktivieren Sie das Projekt.

⇨ Verändern Sie, bspw. über das Block Diagram, die Werte von GlobalParam im DataDefiner und beobachten Sie die direkte Auswirkung auf die DataUser-Instanzen.

4.7.8 Erstellen eines Moduls mit OEM-Lizenzabfrage

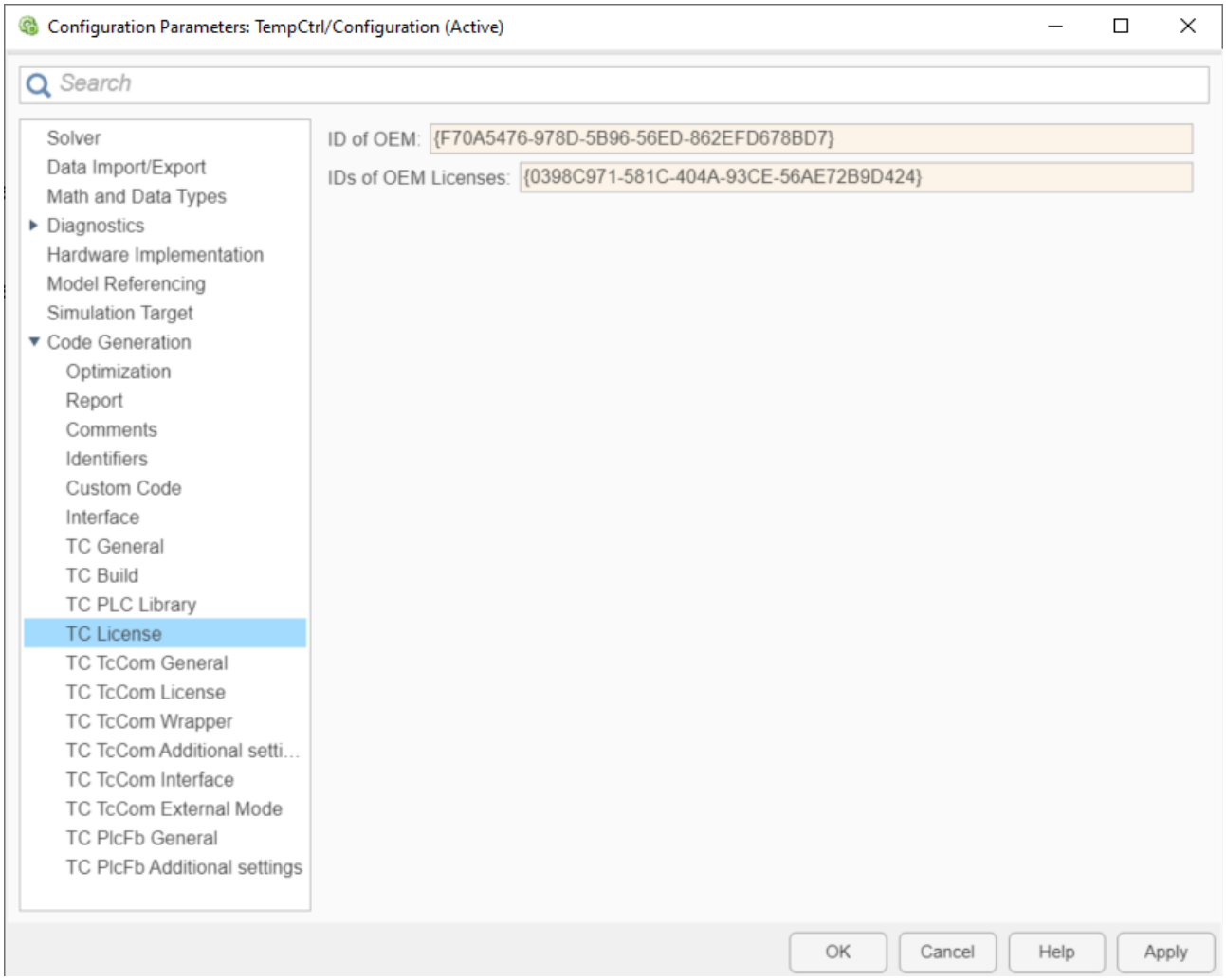
Wofür eine eigene Lizenz an ein Modul koppeln?

Ist ein TwinCAT-Objekt, neben den TwinCAT-Lizenzen, auch an eine OEM-Lizenz gebunden, kann dadurch eine Bindung dieses TwinCAT-Objekts an eine Hardware realisiert werden, sodass die Anwendung vor Klonen geschützt ist. Außerdem können über diesen Weg Funktionalitäten einer Anwendung an Endkunden lizenziert werden.

Weitere Informationen finden Sie im Bereich Software Protection / Eigene OEM-Lizenzen.

Konfiguration in Simulink®

- Wechseln Sie in den Konfigurations-Level Advanced:
  `TwinCAT.ModuleGenerator.Settings.Change('ConfigurationLevel', 'Advanced')`
- Tragen Sie Ihre OEM-ID und die anzufragende(n) OEM-Lizenz(en) ein:



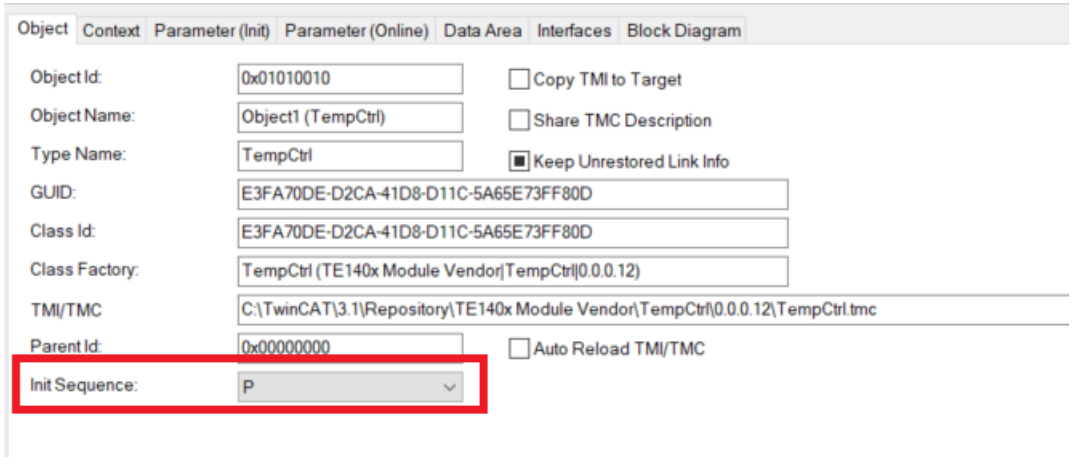
Wird das Modul mit obigen Einstellungen erstellt und in TwinCAT instanziiert, muss neben einer gültigen TwinCAT-Lizenz (TC1220, TC1320) auch eine gültige OEM-Lizenz vorhanden sein, damit TwinCAT aktiviert werden kann.

Zu beachten bei Lizenz-Dongles

Folgendes ist zu beachten, wenn Sie die OEM-Lizenz für das *Zielsystem* auf einem Dongle nutzen:

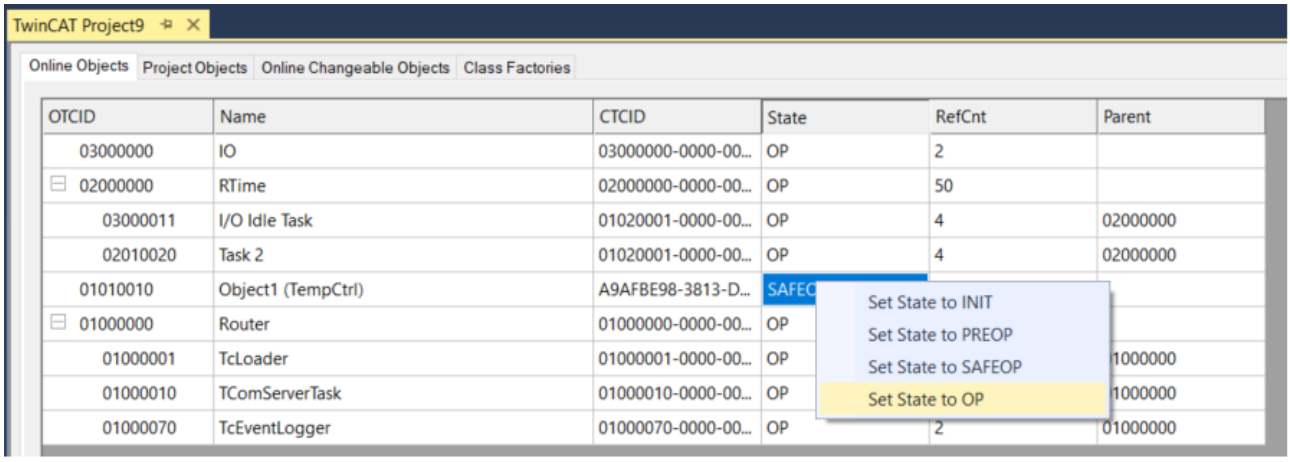
✓ **Nutzen Sie eine Instanz eines TcCOM?**

1. Stellen Sie die Init Sequence auf der Objekt-Instanz auf P.



2. Beachten Sie, dass Sie in diesem Fall keine aktiven Mappings benutzen können. Es wird dazu geraten, den TcCOM-Wrapper-FB zu nutzen, oder das Modul aus TwinCAT C++ aufzurufen.
3. Aktivieren Sie die Konfiguration.

4. Schalten Sie das TcCOM-Objekt -nachdem TwinCAT im Run-Mode ist- in den OP-State, z. B. über das XAE (siehe untenstehende Grafik), über den TcCOM-Wrapper-FB oder per ADS.



⇨ Die Lizenz wird beim Hochfahren in den OP-State geprüft und (wenn gültig) akzeptiert.

✓ **Nutzen Sie den TcCOM-Wrapper-FB aus der erstellten SPS-Bibliothek und referenzieren auf eine statische TcCOM-Instanz?**

1. Stellen Sie die Init Sequence auf der Objekt-Instanz auf P (siehe oben).
2. Nutzen Sie den TcCOM-Wrapper-FB, um das referenzierte TcCOM in OP zu schalten.

⇨ Die Lizenz wird beim Hochfahren in den OP-State geprüft und (wenn gültig) akzeptiert.

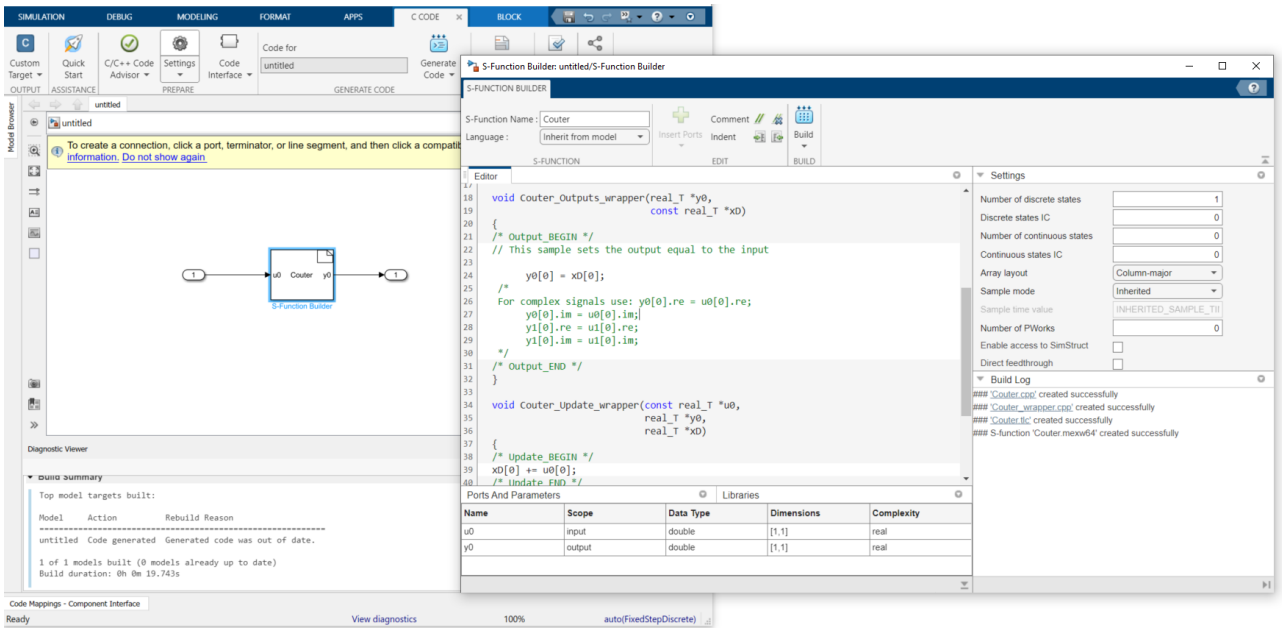
✓ **Wann müssen Sie nichts weiter beachten?**

1. Wenn Sie den PLC-FB aus der erstellten SPS-Bibliothek nutzen.
2. Wenn Sie mit dem TcCOM-Wrapper-FB dynamisch ein TcCOM erstellen.

4.7.9 Einbinden von eigenem C/C++-Code

Das TwinCAT Target for Simulink® unterstützt auch das Einbinden von eigenem C/C++-Code in Simulink®. MathWorks® bietet dazu mehrere Möglichkeiten an, zum Beispiel den Weg über den S Function Builder.

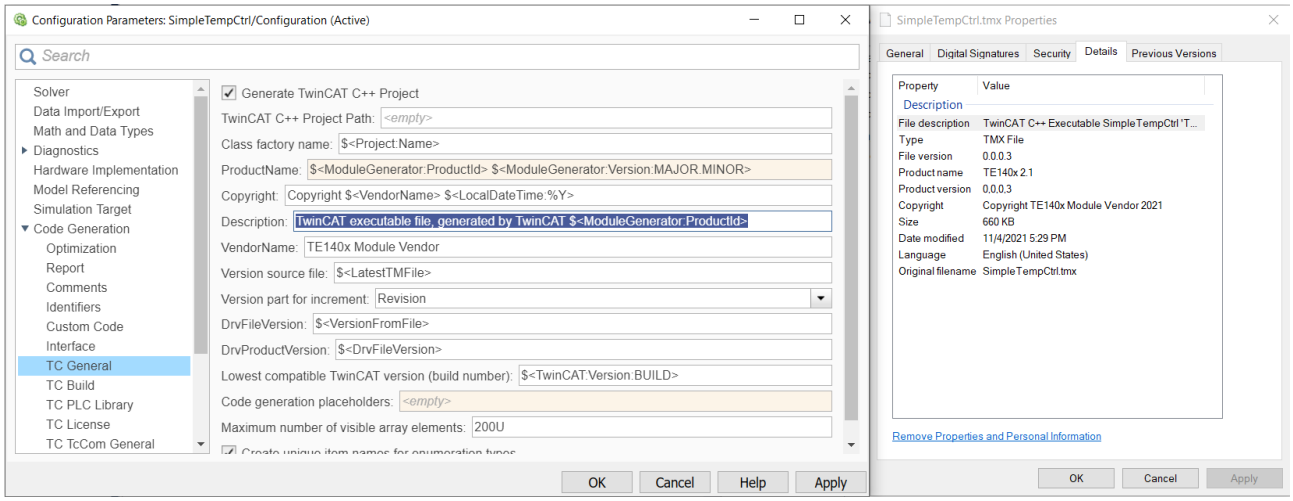
Beachten Sie, dass Sie die Sprache auf „Inherit from model“ setzen. Sie können auch Bibliotheken einbinden, solange diese plattformunabhängig sind und als Quellcode vorhanden sind. Die Einbindung einer vorkompilierten DLL ist beispielsweise nicht möglich.



4.7.10 Konfiguration der TMX-Datei-Properties

Sie können die Einträge in der TMX-Datei (TwinCAT Module Executable) aus Simulink® heraus parametrieren. Wechseln Sie dazu in den Advanced Modus:

```
TwinCAT.ModuleGenerator.Settings.Change('ConfigurationLevel', 'Advanced')
```



Beziehung von TMX-Properties (links) zu Parametern in Simulink® (rechts)

File description -> Description

File Version -> DrvFileVersion

Product name -> ProductName

Product version -> DrvProductVersion

Copyright -> Copyright

Beachten Sie: $< > beschreibt Platzhalter [▶ 177]. Beispielsweise wird DrvProductVersion auf den Wert in DrvFileVersion gesetzt, welches wiederum den Wert aus Version Source File bezieht.

4.7.11 Multitask, Concurrent Execution und OpenMP

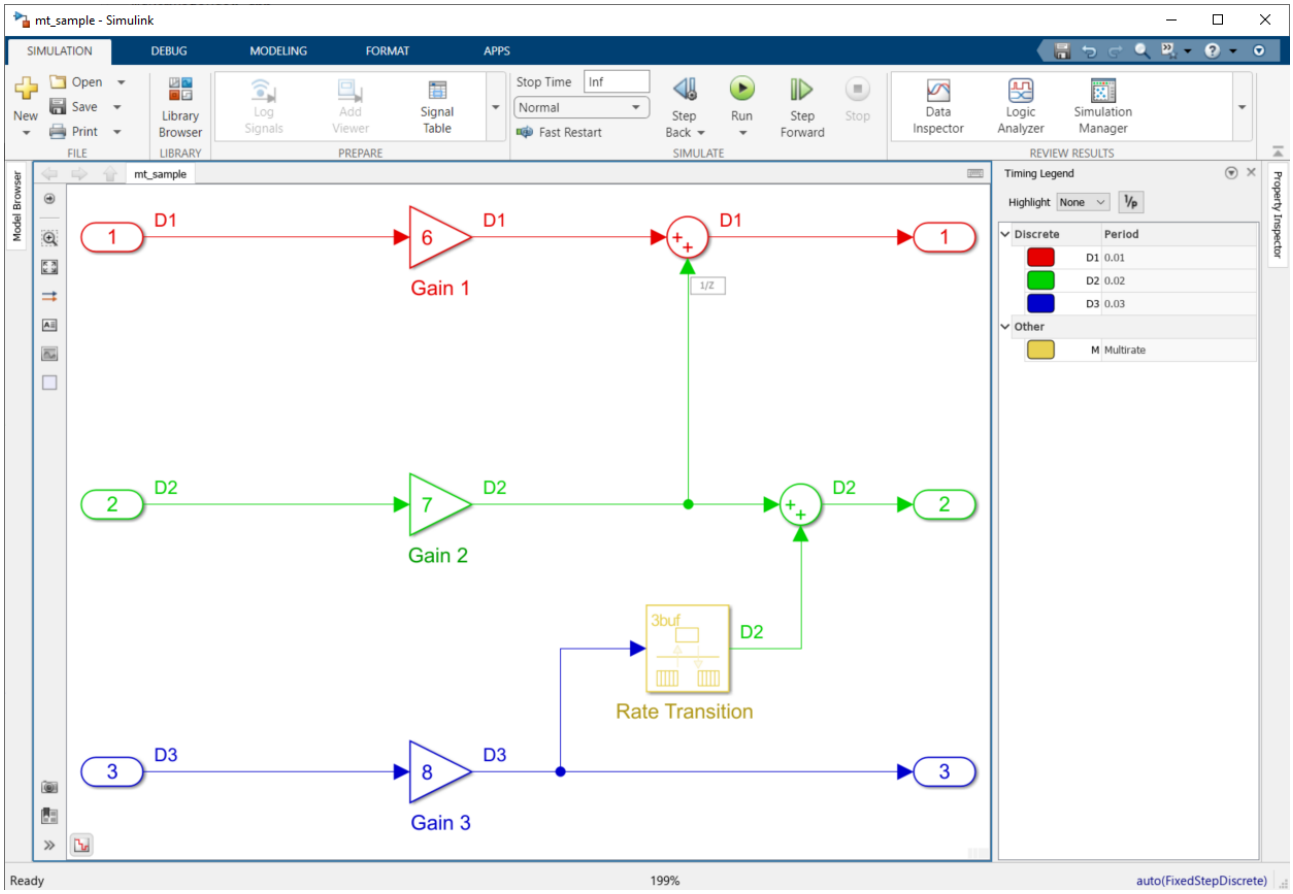
In Simulink® können Sie Ihre Modelle für die Ausführung auf Multicore Target Systemen konfigurieren. Details entnehmen Sie dazu der MathWorks®-Dokumentation. Beckhoff Targets bieten in der Regel eine Multicore-Architektur, welche mit TwinCAT 3 effizient genutzt werden kann. Dies ist auch mit dem TwinCAT Target for Simulink® möglich, wie im Folgenden gezeigt wird.

Es wird in dieser Beschreibung unterschieden zwischen Multitask, Concurrent Execution und OpenMP.

- Bei Multitask [▶ 166] wird ein TcCOM-Objekt erzeugt, welches mehrere Tasks zur Verfügung hat. **Alle** Tasks müssen auf demselben Core ausgeführt werden. Es wird nicht parallelisiert.
- Bei Concurrent Execution [▶ 168] wird ebenfalls ein TcCOM-Objekt mit mehreren Tasks erzeugt, die auf unterschiedlichen Cores verteilt werden können. Rechnungen können tatsächlich parallel ausgeführt werden.
- Bei OpenMP [▶ 169] wird ein TcCOM-Objekt mit einem Task-Kontext erzeugt. Darüber hinaus können mehrere JobTasks, die auf unterschiedlichen Cores verteilt liegen, die Code-Fragmente parallelisiert ausführen, welche als OpenMP-Code generiert wurden.

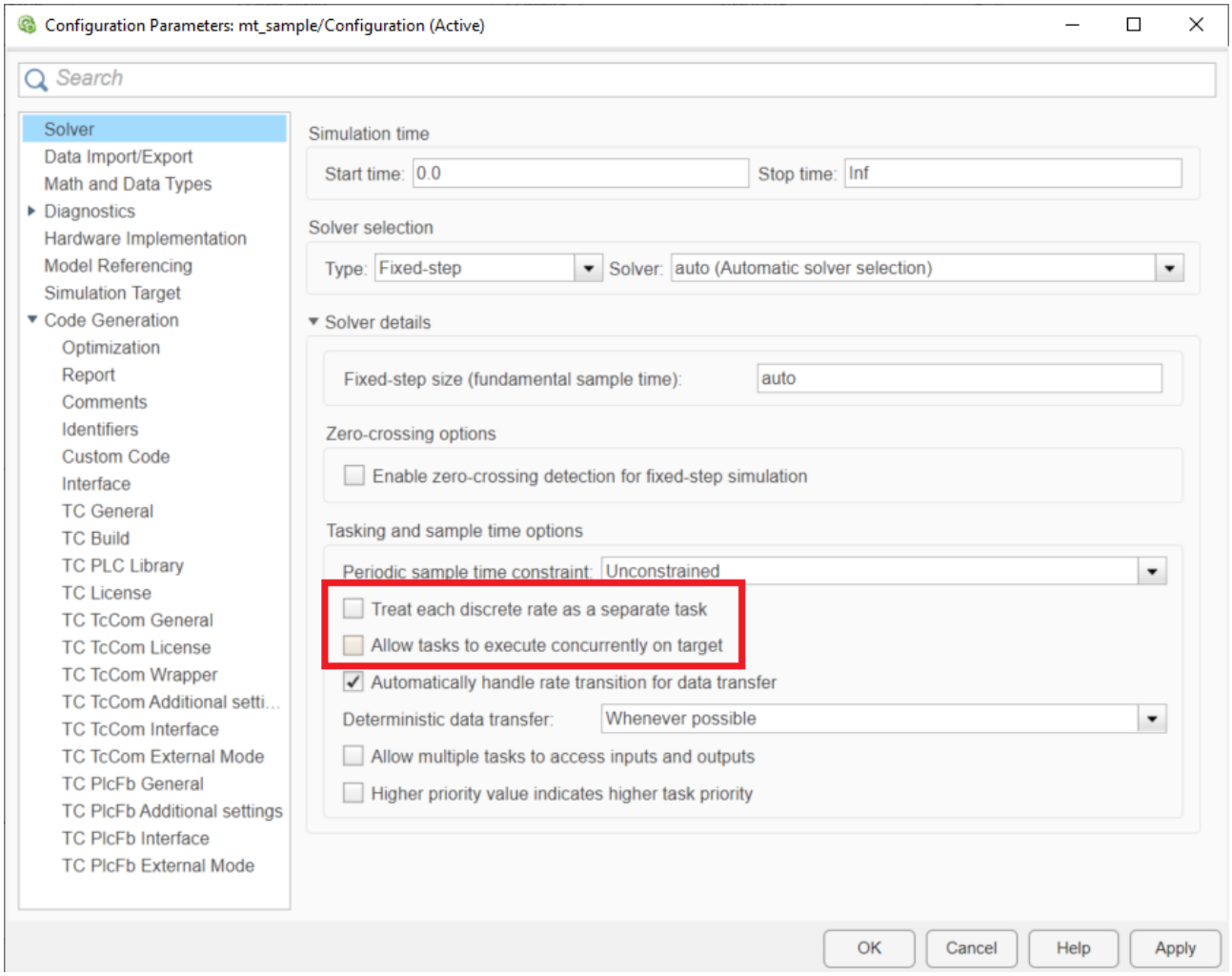
Multitask und Concurrent Execution

Für die Beschreibungen zu den Optionen *Multitask* und *Concurrent Execution* wird das folgende Multirate System in Simulink® betrachtet. Das Modell besitzt eine explizite und eine implizite *rate transition*.



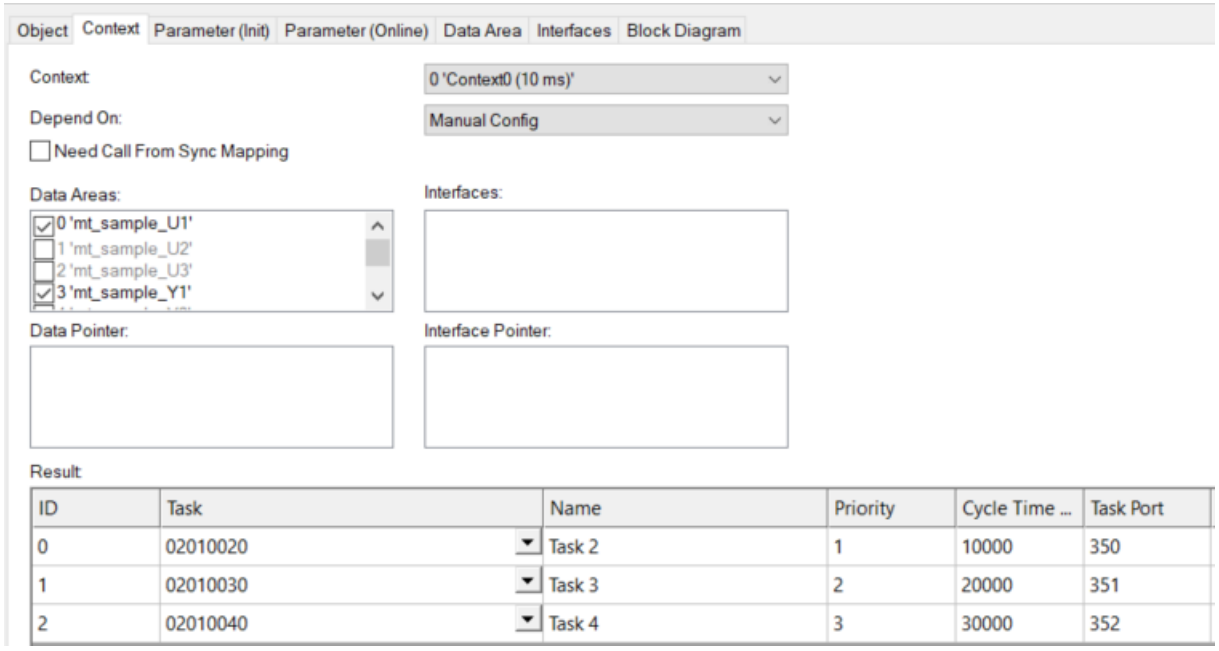
Wählen Sie unter **Configuration Parameters** den Punkt **Solver** aus. Hier können Sie auswählen zwischen:

- Treat each discrete rate as separate task
- Allow tasks to execute concurrently on target

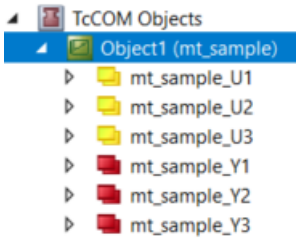


Treat each discrete rate as separate task: Multitask

Wird ein TcCOM-Objekt mit aktivierter Option *Treat each discrete rate as separate task* erstellt, erhalten Sie ein Objekt, welchem Sie mehrere Task-Kontexte zuweisen können. In diesem Fall 3 Tasks.



Die Ein- und Ausgänge sowie alle weiteren DataAreas werden in die unterschiedlichen Kontexte geteilt, sodass in diesem Fall 3 Input DataAreas und 3 Output DataAreas existieren.



Die zyklischen Tasks müssen in diesem Fall alle auf demselben Core gelegt werden. Es findet keine parallele Verarbeitung der Tasks statt.

Der Vorteil gegenüber einem TcCOM mit nur einem Task-Interface liegt darin, dass nun nicht alle Berechnungen innerhalb der schnellsten Task-Zykluszeit abgeschlossen werden müssen (siehe Scheduling). Würde obiges Simulink-Modell mit Standardeinstellung ohne *Treat each discrete rate as separate task* erstellt werden, wäre nur eine Task mit 10 ms (schnellste Task) verknüpfbar. Das bedeutet, dass dann alle Rechnungen innerhalb dieser Zeit abgeschlossen werden müssen. Durch Verteilung auf mehrere Tasks auf demselben Core, wird diese Regel außer Kraft gesetzt, da Tasks sich gegenseitig unterbrechen können (siehe Prioritäten).

Object	RT-Core	Base Time (ms)	Cycle Time (ms)	Cycle Ticks	Priority
Task 2	Default (3)	1 ms	10 ms	10	1
Task 3	Default (3)	1 ms	20 ms	20	2
Task 4	Default (3)	1 ms	30 ms	30	3
I/O Idle Task	Core 2	1 ms	1 ms	1	11

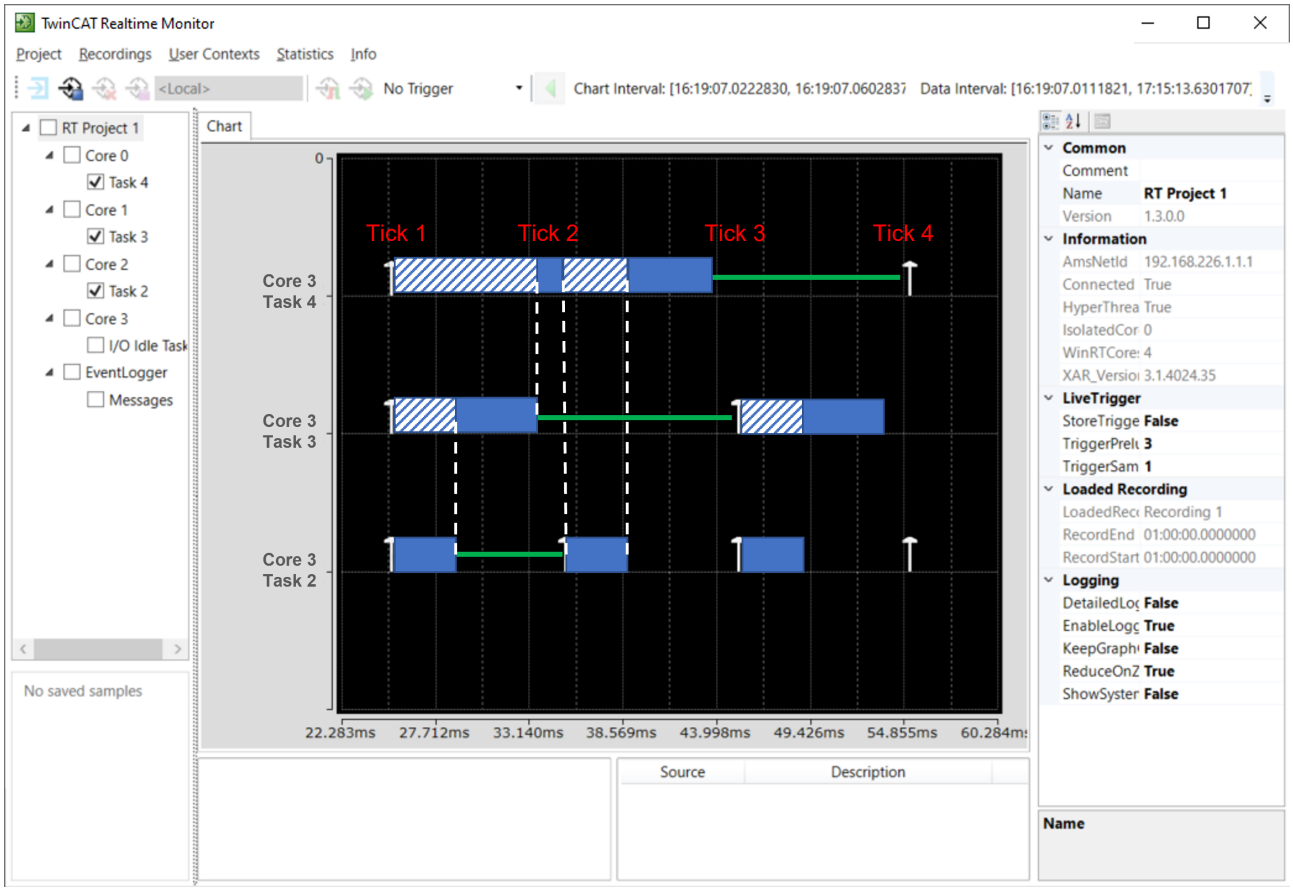
Eigenschaften:

- Es wird kein Funktionsbaustein in der PLC unterstützt.
- Die TwinCAT Usermode Runtime wird nicht unterstützt.
- Alle Tasks sind demselben Core zugewiesen.
- Der schnellsten Task muss die höchste Priorität (kleinster Priority-Wert) zugewiesen werden. Der zweitschnellsten Task die zweithöchste Priorität usw.

Scheduling Details:

Untere Grafik beschreibt ein Beispiel, wie die Rechenzeiten verteilt sein können. Die schraffierten Flächen zeigen an, dass in dieser Zeit eine Task durch eine höher priorisierte Task nicht arbeiten darf. Die volle blaue Fläche zeigt an, dass die Task arbeitet. Beachten Sie, dass die Flächen nur zur Förderung der Verständlichkeit nachträglich über die Realtime Monitor Aufnahme gelegt wurden und keine realen Aufnahmen sind.

- In Tick 1 werden Task 2, Task 3 und Task 4 nacheinander auf demselben Core ausgeführt. Die Ausführung von Task 2 und Task 3 laufen dabei ohne Unterbrechung. Die Ausführung von Task 4 wird von der höher priorisierten Task 2 im Übergang zu Tick 2 unterbrochen.
- In Tick 2 wird zunächst Task 2 ausgeführt. Die Ausführung von Task 4 wird nach Beendigung von Task 2 wieder aufgenommen.
- In Tick 3 beginnt wieder Task 2 und es folgt Task 3.



Falls es zu Zykluszeitüberschreitungen kommt und das Scheduling nicht eingehalten werden kann, wird die Ausführung des jeweiligen Task-Kontexts so lange übersprungen, bis alle relevanten Kontexte im passenden Zustand sind. Im TcCOM-Objekt kann dieses Verhalten über den Online-Parameter `SkippedExecutionCount` beobachtet werden.

Allow tasks to execute concurrently on target: Concurrent Execution

Wird ein TcCOM-Objekt mit aktivierter Option *Allow tasks to execute concurrently on target* erstellt, erhalten Sie ein Objekt, welchem Sie mehrere Task-Kontexte zuweisen können. In diesem Fall, wie im obigen Beispiel, 3 Tasks.

Auch hier werden die DataAreas aufgetrennt in die unterschiedlichen Kontexte. Unterschied zum Multitask-Objekt ist, dass Sie nun die Tasks auf unterschiedliche Cores verteilen können, sodass die Abarbeitung tatsächlich parallelisiert wird.

Object	RT-Core	Base Time (ms)	Cycle Time (ms)	Cycle Ticks	Priority
Task 2	Core 2	1 ms	10 ms	10	1
Task 3	Core 1	1 ms	20 ms	20	2
Task 4	Core 0	1 ms	30 ms	30	3
I/O Idle Task	Default (3)	1 ms	1 ms	1	11

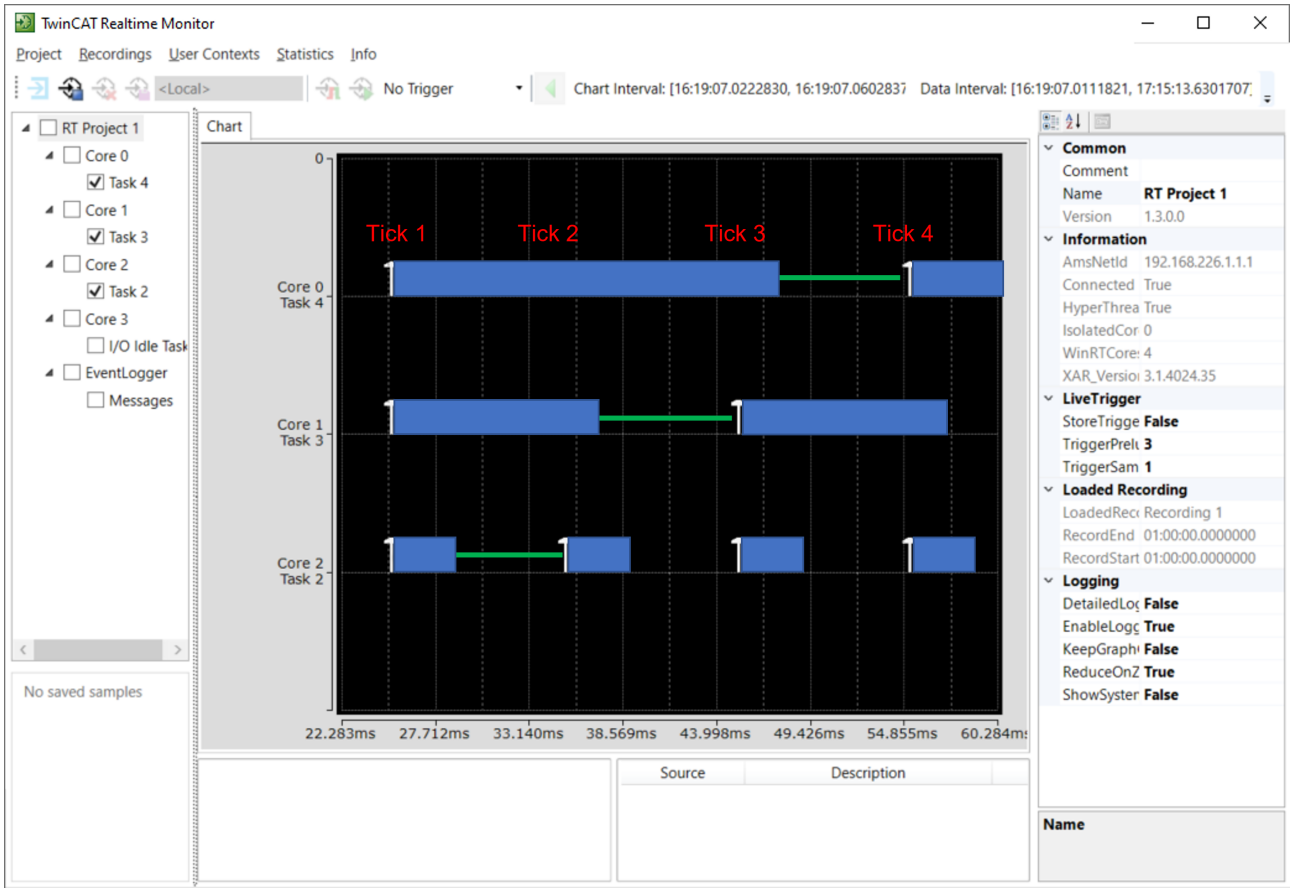
Eigenschaften:

- Es wird kein Funktionsbaustein in der PLC unterstützt.
- Die TwinCAT Usermode Runtime wird unterstützt.
- Tasks können unterschiedlichen Cores zugewiesen werden.
- Der schnellsten Task muss die höchste Priorität (kleinster Priority-Wert) zugewiesen werden. Der zweitschnellsten Task die zweithöchste Priorität usw.

Scheduling Details:

Untere Grafik beschreibt ein Beispiel, wie die Rechenzeiten verteilt sein können. Die volle blaue Fläche zeigt an, dass die Task arbeitet. Beachten Sie, dass die Flächen nur zur Förderung der Verständlichkeit nachträglich über die Realtime Monitor Aufnahme gelegt wurden und keine realen Aufnahmen sind.

- In Tick 1 werden Task 2, Task 3 und Task 4 parallel auf unterschiedlichen Cores ausgeführt. Die Ausführung von Task 1 muss bis zu Beginn von Tick 2 beendet sein.
- In Tick 2 wird Task 2 wieder ausgeführt. Task 3 und Task 4 dürfen weiterhin arbeiten. Die Ausführung von Task 2 und Task 3 muss bis zu Beginn von Tick 3 beendet sein.
- In Tick 3 werden Task 2 und Task 3 wieder ausgeführt. Task 4 darf weiterhin arbeiten. Die Ausführung von Task 2 und Task 4 muss bis zu Beginn von Tick 4 beendet sein.

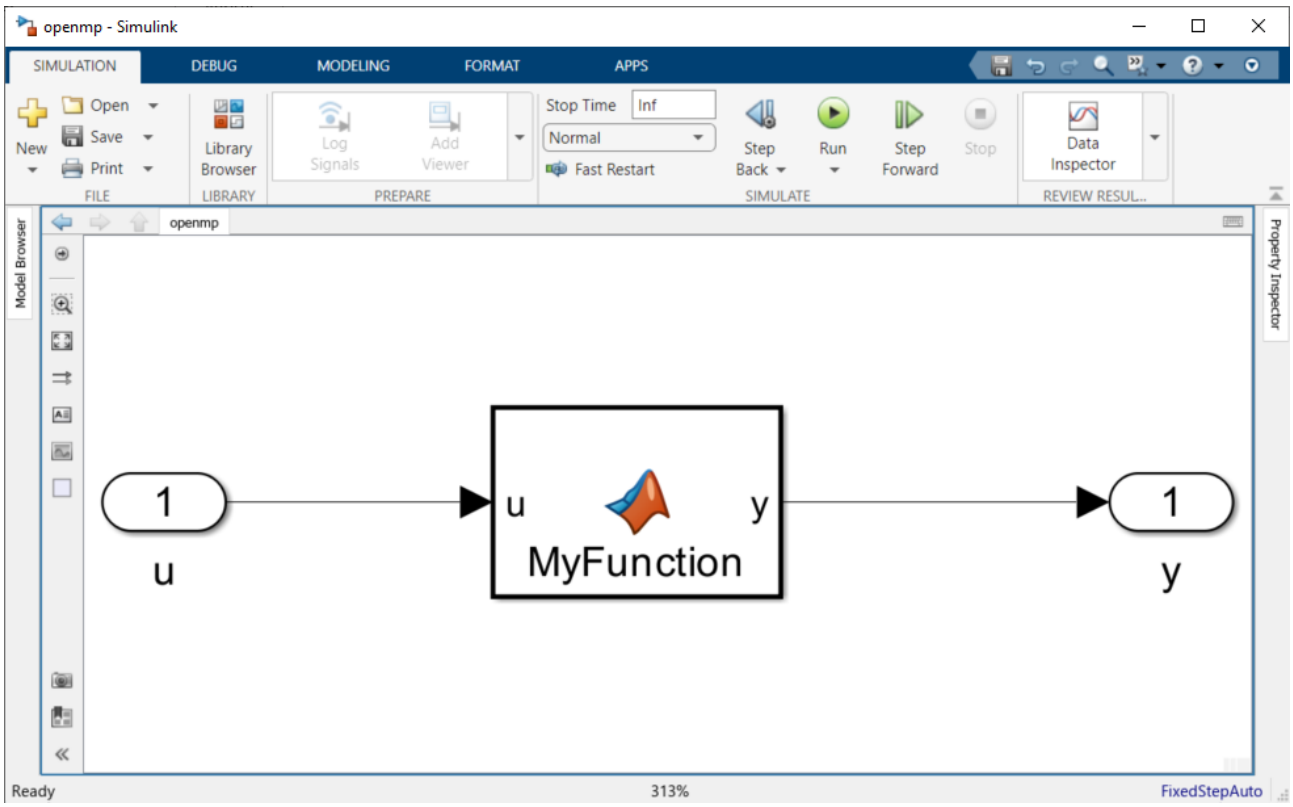


Falls es zu Zykluszeitüberschreitungen kommt und das Scheduling nicht eingehalten werden kann, wird die Ausführung des jeweiligen Task-Kontexts so lange übersprungen, bis alle relevanten Kontexte im passenden Zustand sind. Im TcCOM-Objekt kann dieses Verhalten über den Online-Parameter `SkippedExecutionCount` beobachtet werden.

OpenMP

Der Simulink Coder™ bzw. der MATLAB Coder™ können openMP Code erzeugen. In welchen Fällen das genau passiert, entnehmen Sie bitte der MathWorks®-Dokumentation.

Im Folgenden wird ein Beispiel mit einer MATLAB® Function in Simulink® betrachtet. Ein MATLAB®-Beispiel finden Sie in Verbindung mit dem TE1401 TwinCAT Target for MATLAB® in den Beispielen: `TwinCAT.ModuleGenerator.Samples.Start('Code parallelization with OpenMP')`.



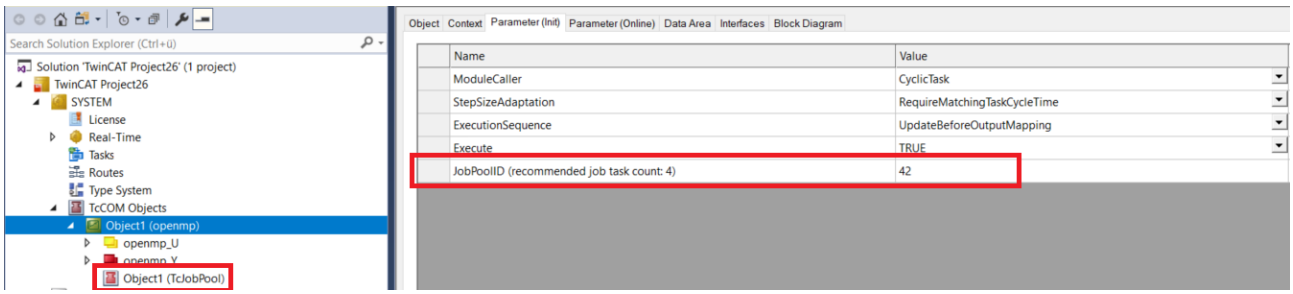
In der MATLAB® Function wird das parfor-Kommando genutzt, um die for-Schleife zu parallelisieren. In diesem Fall wird die Anzahl der parallelen Worker auf 4 begrenzt.

```
function y = MyFunction(u) %#codegen

A = ones(20,50);
t = 42;

parfor (i = 1:10,4)
    A(i,1) = A(i,1) + t;
end

y = A(1,4) + u;
```

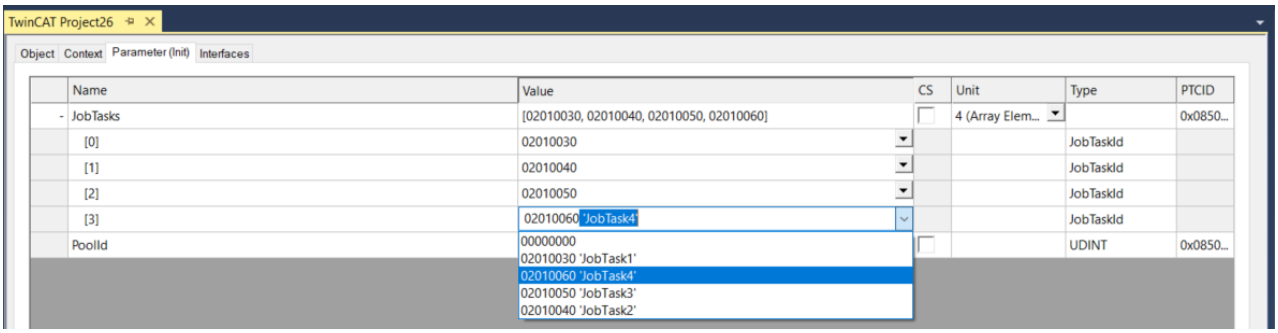
Für das TwinCAT-Target sind keine besonderen Einstellungen hinsichtlich openMP vorzunehmen. Sie generieren Ihre TwinCAT-Objekte wie gewohnt. Der Simulink Coder übersetzt diese Code-Stelle in openMP Code, sodass das der C/C++ Code entsprechend auch parallelisiert wird. Der Embedded Coder wird für dieses Feature nicht benötigt.

In der TwinCAT XAE können Sie nun das erstellte TcCOM oder den PLC-FB instanziieren und entsprechend konfigurieren. Die Objekt-Instanz bietet unter dem Reiter Context wie gewohnt nur ein zyklisches Task-Interface an. In diesem Beispiel wird eine Task 2 mit 200 ms Zykluszeit erstellt und dem Objekt zugewiesen.

Unter Parameter (Init) befindet sich ein Parameter JobPoolID. Hier wird, soweit durch den C/C++ Code bekannt auch angezeigt, wie viele Worker parallel arbeiten können. Ein JobPool ist eine Organisationseinheit für JobTasks, welche im Knoten Tasks angelegt werden können.



Es ist entsprechend unter TcCOM Objects mit „Add new item“ ein Objekt von Typ TcJobPool hinzuzufügen. Unter Parameter (Init) auf dem TcJobPool-Objekt ist die JobPoolId einzutragen und eine Gruppe von JobTasks zu referenzieren. Definieren Sie zunächst wie viele JobTasks der Pool zusammenfassen soll und wählen Sie dann mit dem Drop-Down-Menü die JobTasks aus.



Unter System > Realtime können Sie JobTasks auf unterschiedliche Cores verteilen.

Object	RT-Core	Base Time (ms)	Cycle Time (ms)	Cycle Ticks	Priority
Task 2	Default (4)	1 ms	200 ms	200	1
JobTask1	Core 0	1 ms	(none)	0	2
JobTask2	Core 1	1 ms	(none)	0	3
JobTask3	Core 2	1 ms	(none)	0	4
JobTask4	Core 3	1 ms	(none)	0	5
I/O Idle Task	Default (4)	1 ms	1 ms	1	11

Die Ausführung in der oben abgebildeten Konfiguration findet dann wie folgt statt. Task 2 wird auf Core 4 ausgeführt und triebt zyklisch das openmp-Objekt an. Die Codefragmente die als openMP Code generiert wurden, können dann über den JobPool Aufgaben an die konfigurierten JobTasks auslagern. Haben die JobTasks mit ihren Berechnungen beendet werden alle Teilergebnisse wieder gebündelt und Task 2 auf Core 4 führt den Code zu Ende aus.

4.7.12 Symbol Properties und Attribut-Pragmas

Was sind Properties und Attribute?

Bei TcCOM-Objekten können Sie allen Definitionen, z. B. DataAreas, Datentypen, SubItems usw., **Properties** zuweisen. Ein Property ist dabei als Name-Value-Pair definiert. Es können beliebige Zusatzinformationen mitgegeben werden.

Attribute werden in der SPS in der Regel im Deklarationsteil verwendet und können ebenfalls beliebige Zusatzinformationen z. B. an eine Variable binden. Eine Liste von SPS-Attributen finden Sie in der SPS-Dokumentation.

Viele TwinCAT Functions nutzen Attribute und Properties. Beispiele sind:

- TwinCAT OPC-UA
  - `{attribute 'OPC.UA.DA' := '1'}`
  - `{attribute 'OPC.UA.DA.Access' := '1'}`
- Analytics Logger
  - `{attribute 'TcAnalytics'}`
- JSON Library Tc3_JsonXml
- …

Attribute und Properties können aber auch vom Nutzer selber definiert und für eigene Anwendungen genutzt werden.

Wie nutze ich Symbol Properties und Attribute zusammen mit dem TwinCAT-Target?

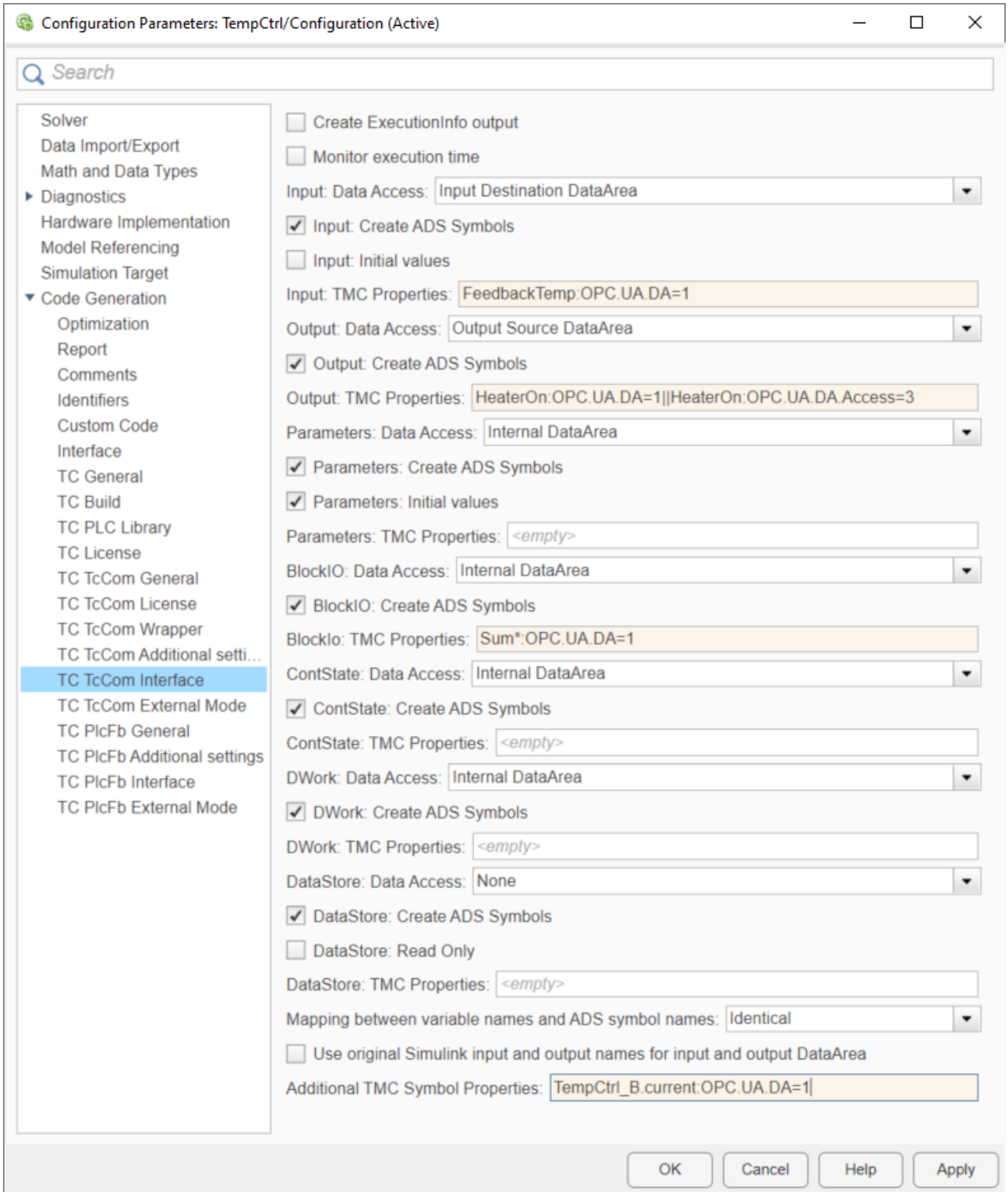
Mit dem TwinCAT-Target ist es möglich, Properties ADS-Symbolen zuzuweisen. ADS-Symbole sind im Sinne von Variablen zu verstehen.

(TcCOM) TMC Properties

In Simulink® können Sie in den Configuration Parameters unter TC TcCom Interface Properties frei als String zuweisen. Properties können Sie jeweils für die DataArea definieren:

- Input: TMC Properties
- Output: TMC Properties
- Parameters: TMC Properties
- BlockIO: TMC Properties
- ContState: TMC Properties
- DWork: TMC Properties
- DataStore: TMC Properties

Zusätzlich können Sie über das Feld *Additional TMC Symbol Properties* unabhängig von der DataArea Properties zuweisen.



Für DataArea-spezifische TMC Properties wird folgende Notation verwendet:

```
SymbolName1:PropertyName1=Value1
```

Beispiel: `FeedbackTemp:OPC.UA.DA=1`

Sollen mehrere Properties zugewiesen werden, sind diese mit | zu trennen:

```
SymbolName1:PropertyName1=Value1|SymbolName2:PropertyName2=Value2
```

Beispiel: `HeaterOn:OPC.UA.DA=1|HeaterOn:OPC.UA.DA.Access=3`

Ab MATLAB R2020b können **wildcards** verwendet werden:

```
*:PropertyName1=Value1
```

Das Zeichen * wird als wildcard genutzt und weist in diesem Beispiel allen Symbolen in der DataArea den PropertyName1 mit Value1 zu. Es können auch Teilwörter mit wildcards kombiniert werden. Mit dem folgenden Beispiel wird allen Symbolen, die mit Sum beginnen, das angegebene Property zugewiesen.

```
Sum*:OPC.UA.DA=1
```

Dieselbe Struktur ist im Feld Additional TMC Symbol Properties einzuhalten. Jedoch ist die DataArea zur Adressierung hinzuzufügen.

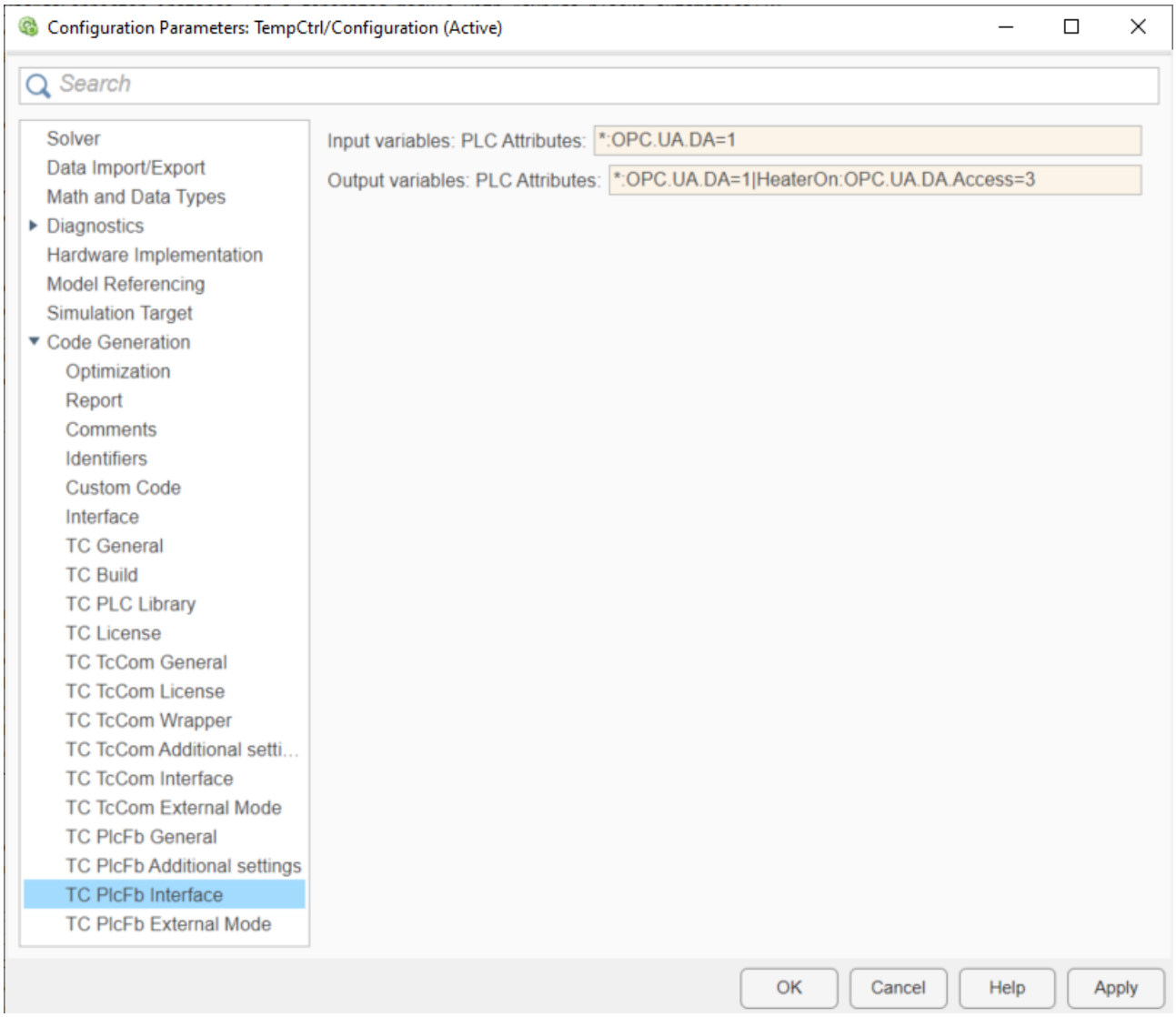
```
DataAreaName1.SymbolName1:PropertyName1=Value1
```

Beispiel: `TempCtrl_B.current:OPC.UA.DA=1`

PLC Attributes

Selbige Vorgehensweise wie für TcCOM gilt für den PLC-FB [▶ 219], dort allerdings nur für die Ein- und Ausgänge möglich. Bei Verwendung des TcCOM-Wrapper-FB [▶ 214] wird ein TcCOM-referenziert, sodass Sie dabei die Konfiguration nach TMC Properties vornehmen.

Die Konfiguration für den PLC-FB nehmen Sie unter TC PlcFb Interface vor.



Beispielcode in MATLAB®

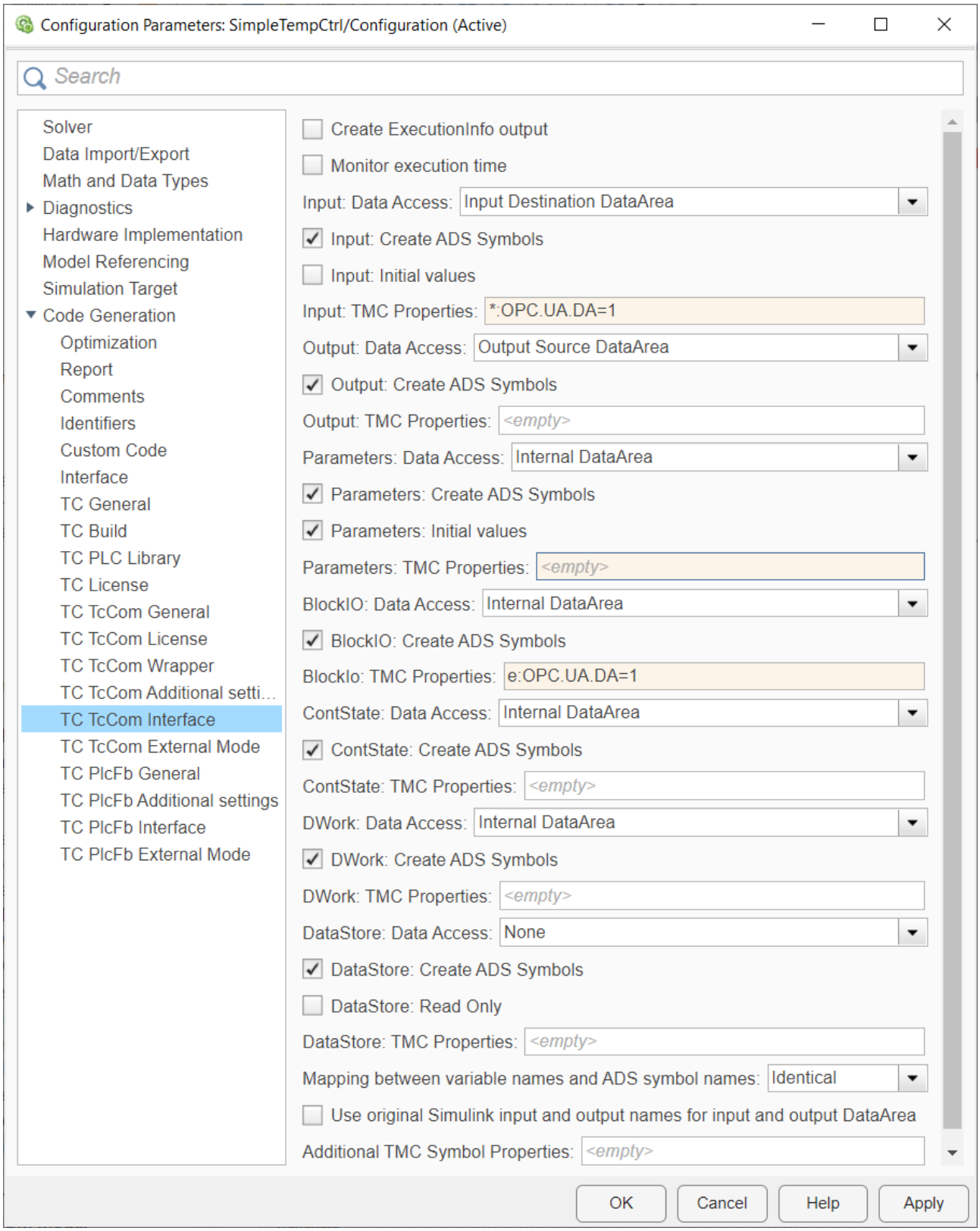
Öffnen Sie das passende Beispiel mit:

```
TwinCAT.ModuleGenerator.Samples.Start('Simulink TMC Symbol Properties')
```

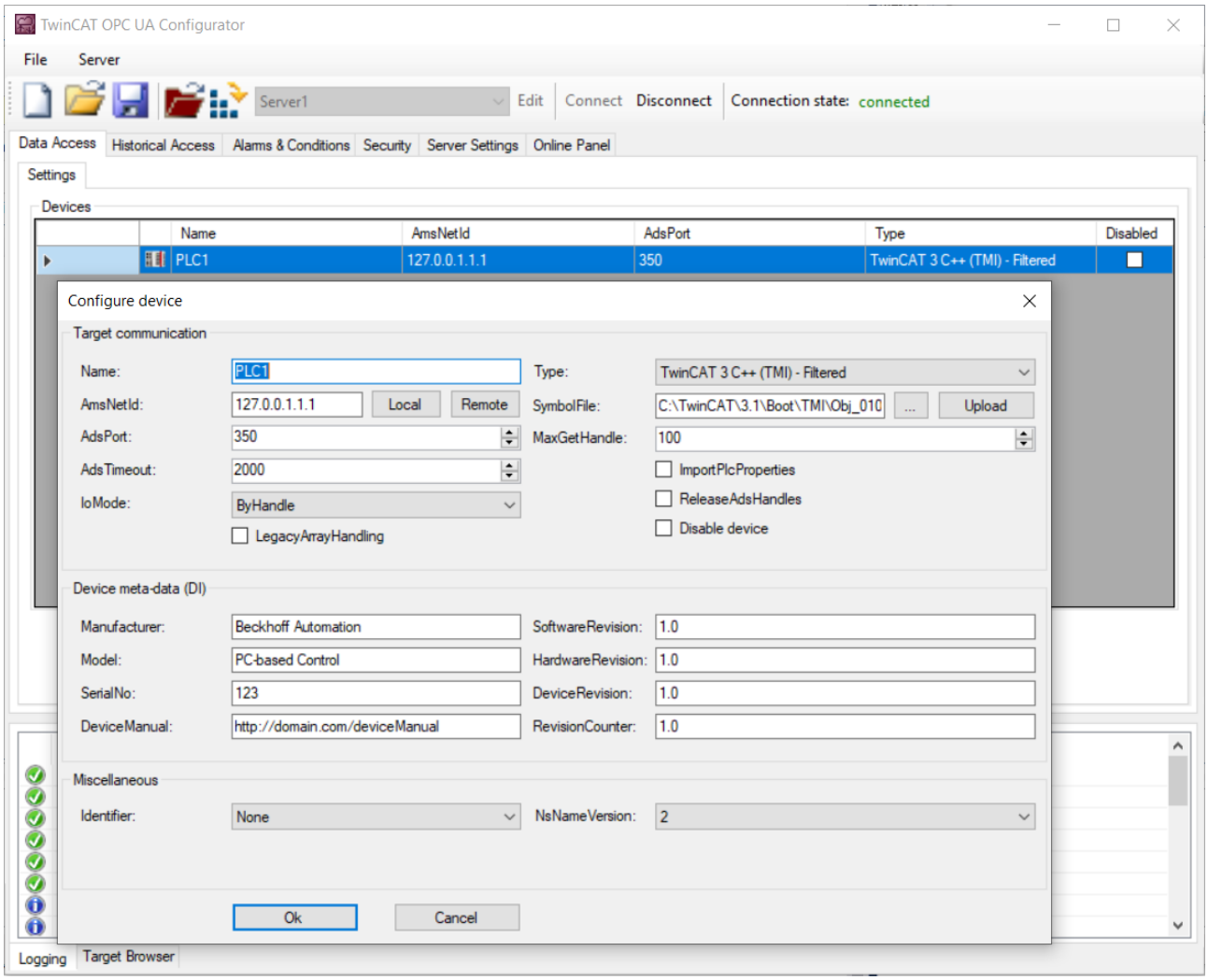
Wofür kann ich Properties und Attribute verwenden?

Ein Beispiel ist die Verwendung von Attributen in der SPS oder Symbol Properties für TcCOM im Zusammenhang mit **TwinCAT 3 OPC-UA**. Siehe dazu die OPC-UA Liste von Attributen.

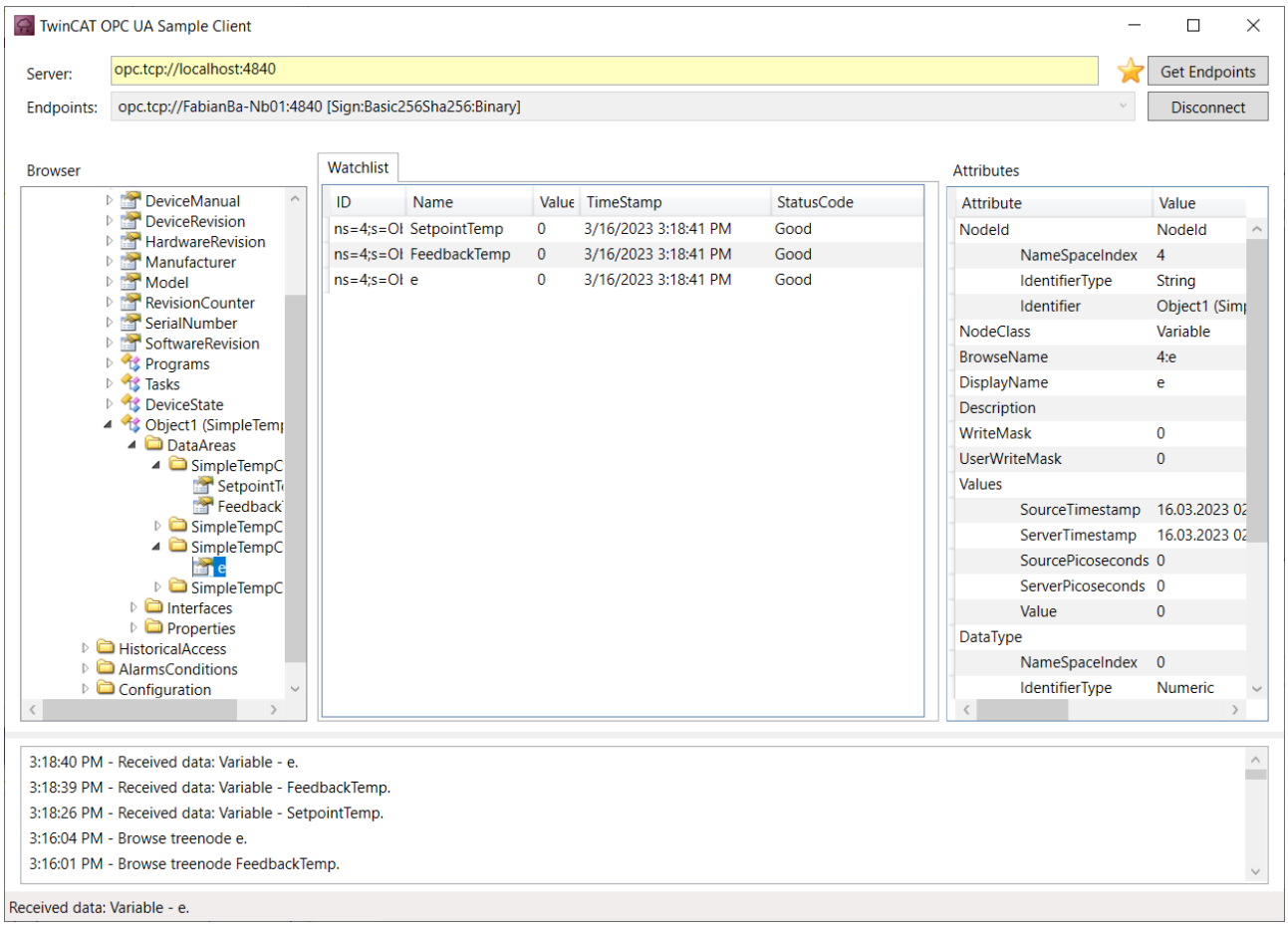
Im Folgenden wird beispielhaft gezeigt, wie alle Input-Variablen der Input DataArea und ausschließlich das Signal e aus der BlockIO DataArea mit der OPC-UA data access Eigenschaft versehen werden. Für die Input DataArea wird dabei die Wildcard * genutzt.



Im OPC-UA Konfigurator muss auf der rechten Seite unter Type TwinCAT 3 C++ (TMI) – Filtered gewählt werden, damit ausschließlich die Symbole mit entsprechendem Property im Server angezeigt werden. Möchten Sie alle Symbole im Server sehen, wählen Sie an dieser Stelle einfach TwinCAT 3 C++ (TMI) – All. Dann werden alle Symbole, ungeachtet der Properties, angezeigt. Wählen Sie zudem als SymbolFile die TMI-Datei des TcCOM auf dem TwinCAT Target Device aus (Boot\TMI Folder).

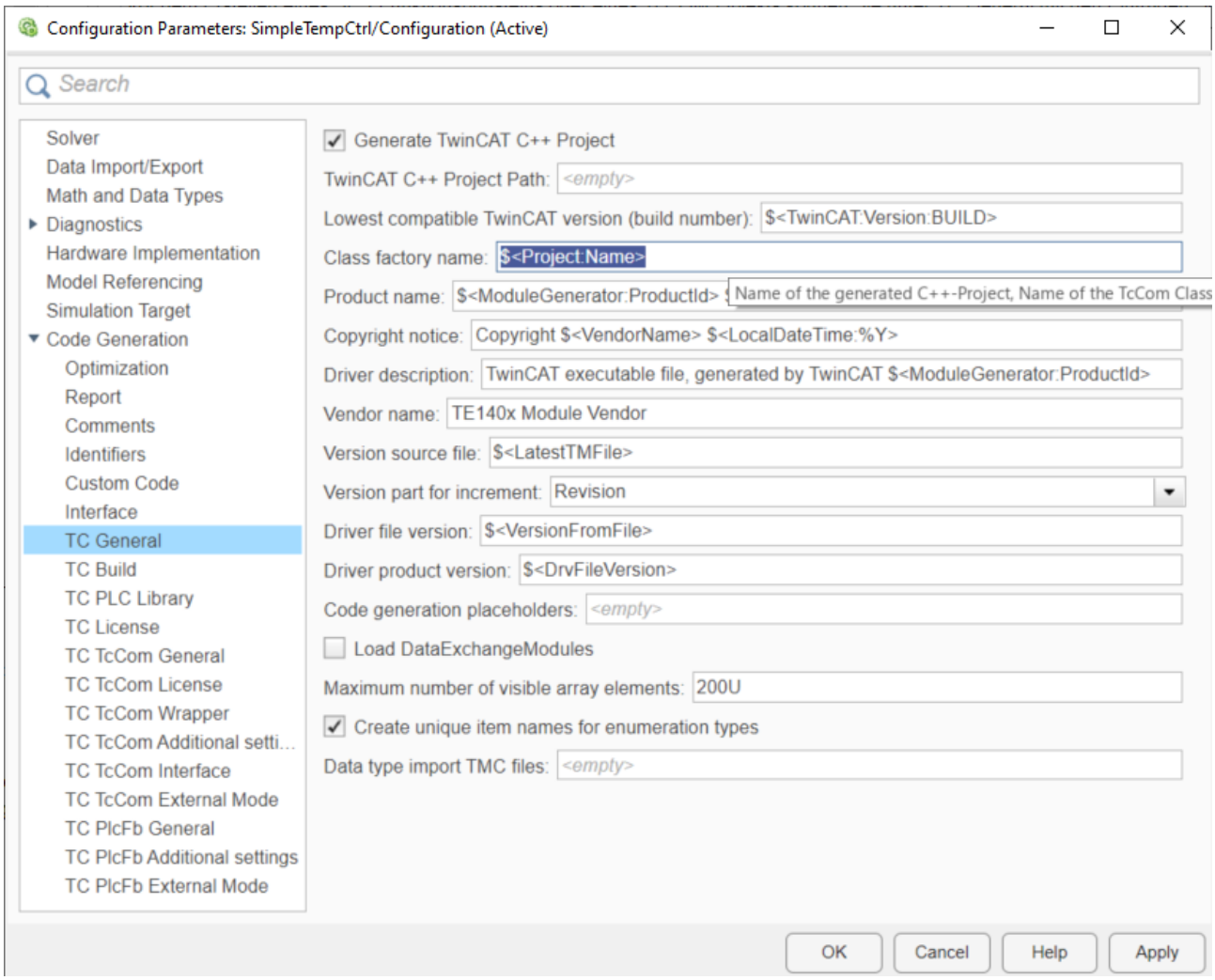


Verbinden Sie sich mit einem Client auf den OPC-UA Server, um den Namespace zu inspizieren. Hier wurde der OPC-UA Sample Client von TwinCAT 3 genutzt. Es ist zu sehen, dass ausschließlich die Symbole im Server angezeigt werden, welche explizit mit dem OPC.UA.DA=1 Property versehen wurden.



4.7.13 Verfügbare Platzhalter (Placeholder)

Platzhalter werden in der Konfiguration des Target for Simulink® verwendet, um den Konfigurationsaufwand zu reduzieren und die Übersichtlichkeit zu erhöhen. Platzhalter werden in der Konfiguration mit *$<PlaceholderName>* angegeben.



Was sind Platzhalter?

Platzhalter können genutzt werden, um den Wert eines Konfigurationsparameters abstrakt als Variable anzugeben. Es existieren spezifische Platzhalter auf Ebene des Targets (Modulgenerator), des Projekts und der Module (TcCOM und PLC FB). Darüber hinaus ist der Konfigurationsparameter selbst ebenso Platzhalter, sodass man ihn wiederverwenden kann.

Beispiel:

In obiger Grafik wird Driver product version mit dem Platzhalter *$<DrvFileVersion>* angegeben. Dieser Platzhalter verweist auf den Eintrag Driver file Version, welcher wiederum mit dem Platzhalter *$<VersionFromFile>* besetzt ist. Der Platzhalter *$<VersionFromFile>* nimmt den Versionswert an, wobei die Quelle des Versionswerts mit dem Parameter Version source file definiert wird.

Wofür benötige ich Platzhalter?

Platzhalter bieten die Möglichkeit, einen Parameterwert zu definieren und an vielen Stellen der Konfiguration wiederzuverwenden, oder sie in einer Kettenabhängigkeit zueinander zu setzen (siehe obiges Beispiel).

Übersicht über verfügbare Platzhalter

Platzhalter aus der Gruppe der Konfigurationsparameter

Category	Name	Displayname	Default	Description
TC General	Generate	Generate TwinCAT C++ Project	TRUE	Generate a TwinCAT C++ project. If unset, only code artifacts will be generated which can get used to generate C++ projects later [▶ 135].
	FullPath	TwinCAT C++ Project Path		Full path to the generated VCXPROJ file (e. g. "C:\Temp\MyGeneratedProject.vcxproj")
	LowestCompatibleTcBuild	Lowest compatible TwinCAT version (build number)	$<TwinCAT:Version:BUILD>	The lowest TwinCAT build number the generated C++ project and its modules and POUs are to be compatible with.
	ClassFactoryName	Class factory name	$<Project:Name>	Name of the generated C++-Project, Name of the TcCOM classfactory and tmx-file name
	ProductName	Product name	$<ModuleGenerator:ProductId> $<ModuleGenerator:Version:MAJOR.MINOR>	Product name, used e.g. for the module driver description and the module TMC description [▶ 164].
	Copyright	Copyright notice	Copyright $<VendorName> $<LocalDateTime:%Y>	Copyright notice of the generated module driver file [▶ 164]
	Description	Driver description	TwinCAT executable file, generated by TwinCAT $<ModuleGenerator:ProductId>	Driver description [▶ 164]
	VendorName	Vendor name	TE140x Module Vendor	Module vendor name, used as the company name of the generated executables in the repository [▶ 124] and the major module group as shown in the TwinCAT XAE module dialog [▶ 93].
	VersionSrc	Version source file	$<LatestTMFile>	Path to an existing TMC, TML or XML file containing the previous version value [▶ 142].
	IncrementVersion	Version part for increment	Revision	The part of the version number that is to be incremented [▶ 142].
	DrvFileVersion	Driver file version	$<VersionFromFile>	Executable file version and library version. [▶ 142]
	DrvProductVersion	Driver product version	$<DrvFileVersion>	Product version [▶ 142]
	CodeGenPlaceholders	Code generation placeholders		Define custom placeholders
	UseDataExchangeModules	Load DataExchangeModules	0	Manually set DataExchangeModule dependency (currently no need to set manually)

Category	Name	Displayname	Default	Description
	MaxVisibleArrayElements	Maximum number of visible array elements	200U	Specifies the maximum number of array elements to be displayed in the TwinCAT XAE. In the TwinCAT XAE, larger arrays cannot get expanded and linked to by its individual items
	CreateUniqueEnumItemNames	Create unique item names for enumeration types	1	Create unique item names for enumeration types.
	DataTypeTmcFiles	Data type import TMC files		TMC file(s) containing additional type definitions for code generation
TC Build	PreferToolArchitectureX64	Prefer X64 build tools	TRUE	Prefer X64 compiler and linker. Useful for complex source files, where X86 tools may run out of heap space.
	Verbosity	Codegeneration and build verbosity	Normal	Verbosity level of code generation and build output messages. Silent and Detailed are other possible values.
	Publish	Run the publish step after project generation	TRUE	Start the build procedure after code generation for all selected platforms. The generated module binaries and module descpription files will get copied to the "publish folder". Published modules are automatically located by the XAE and can get instantiated in all TwinCAT 3 projects. If unset, the module generation process will be stopped after code generation. To instantiate in a TwinCAT3 project, the generated C++ project needs to be inserted and built from.
	PublishPlatformToolset	Platform Toolset	Auto	Choose Platform Toolset to build binaries.
	PublishConfiguration	Build configuration	Release	Build configuration to build binaries.
	PublishTcRTx86	TwinCAT RT (x86)	TRUE	Publish binaries for platform 'TwinCAT RT (x86).'
	PublishTcRTx64	TwinCAT RT (x64)	TRUE	Publish binaries for platform 'TwinCAT RT (x64).'
	PublishTcOSx64	TwinCAT OS (x64)	TRUE	Publish binaries for platform 'TwinCAT OS (x64)' (e.g. TwinCAT/BSD)
	ForceRebuildForPublish	Always rebuild all source files on publish	FALSE	Always rebuild all source files on publish
	SignTwinCatCertName	Certificate name for TwinCAT signing		Certificate name for TwinCAT signing with OEM Certificate level 2. [▶ 96]
	TmxInstall	Install TMX	TRUE	Install all generated TwinCAT Objects on local XAE (fill local Engineering Repository [▶ 124]).

Category	Name	Displayname	Default	Description
	TmxArchive	TMX Archive		Name of an optional archive containing all files required to use the generated TwinCAT Objects on another TwinCAT development system. [▶ 139]
	MsBuildPublishProperties	MsBuild publish properties		Set additional MsBuild publish properties.
	MsBuildProjProperties	MsBuild project properties		Set additional MsBuild project properties.
	PreCodeGenerationCallbackFcn	Pre code generation callback function		The defined MATLAB® function is called before code generation [▶ 187].
	PostCodeGenerationCallbackFcn	Post code generation callback function		The defined MATLAB® function is called after code generation [▶ 187].
	PostPublishCallbackFcn	Post publish callback function		The defined MATLAB® function is called after publish [▶ 187].
TC PLC library	LibCatPath	PLC library category description file	$<ProjectDir>\$<Name>.libcat.xml	Path to the PLC library category description file
	LibraryCategories	PLC library categories	$<VendorName>	Define PLC library category hierarchy. Default only one hierarchy level = vendor. List separated with | possible: <MainCategory>|<SubCategory1>|…
	GeneratePlcLibrary	Generate a PLC library	FALSE	Generate a PLC library with POUs. [▶ 211] Define containing POUs with parameter TcComWrapperFb and PlcFb>General>Generate.
	InstallPlcLibrary	Install the generated PLC library	FALSE	Install the generated PLC library for use in the local TwinCAT XAE/PLC [▶ 211].
	PlcTypePrefixes	Type Prefixes		Define custom type prefixes
	PlcVarPrefixes	Variable Prefixes	`PVOID=p | BOOL=b | BOOL32=b | DATE=d | TIME_OF_DATE=td | TIME=t | LTIME=t | GUID=n`	Define custom variable prefixes.
TC License	OemId	ID of OEM		ID of OEM. Required for OEM Licence checks [▶ 161]
	OemLicenses	IDs of OEM Licenses		IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 161]
TC TcCom General	Generate	Generate TcCOM Module (TwinCAT Module Class)	TRUE	Generate a TcCOM module class for the model.

Category	Name	Displayname	Default	Description
	OnlineChange	Online change support	FALSE	Allow to switch between different TcCOM module versions without switching TwinCAT runtime to config mode [▶ 145].
	ModuleProperties	TMC Properties		Additional properties added to the module description in the TMC file: Name1=Value1| Name2=Value2|...
	GroupName	GroupName	TE140x| Simulink Modules	Minor module group name in the TwinCAT XAE module dialog
	GroupDisplayName	GroupDisplayName	$<GroupName>	Minor module group description in the TwinCAT XAE module dialog
	GroupIcon	GroupIcon	$<TE140x:Icon>	Optional module group icon in the TwinCAT XAE module dialog
	ModuleIcon	ModuleIcon	$<TE140x:Icon>	Optional module icon in the TwinCAT XAE module dialog
	InitExceptionHandling	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress or handle floating point exceptions during initialization [▶ 225].
	UpdateExceptionHandling	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress or handle floating point exceptions during cyclic execution [▶ 225].
	AdditionalIncludeFiles	Additional include files		Additional files required to be included after rtwtypes.h
TC TcCom License	OemLicenses	IDs of OEM License	$<Project:OemLicenses>	IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 161]
TC TcCom Wrapper	TcComWrapperFb	TcCom Wrapper FB	FALSE	Generate a PLC Functionblock simplifying the interaction between a PLC and an instance of the generated TcCOM module [▶ 214]
	TcComWrapperFbProperties	TcCom Wrapper FB properties	FALSE	Generate properties for accessible data in the referenced TcCOM object [▶ 214]
	TcComWrapperFbPropertyMonitoring	TcCom Wrapper FB property monitoring	NoMonitoring	NoMonitoring: Online values of properties are not monitored in the PLC online view, CyclicUpdate: Update property values in the PLC online view cyclically, ExecutionUpdate: Update property values in the PLC online view when the property getter or setter is called [▶ 214]
TC TcCom Additional settings	ModuleCaller	Default module caller	CyclicTask	CyclicTask: Call module via TwinCAT Task. Module: Cal module from another TwinCAT module (see e.g. TcCOM-Wrapper-FB).

Category	Name	Displayname	Default	Description
	CallerVerification	Verify caller	Default	Verify the caller context to prevent concurrent execution of the model code and corresponding DataArea mappings. Skip verification to reduce the execution time.
	StepSizeAdaptation	Default StepSize adaptation mode	RequireMatchingTaskCycleTime	Configure how to handle differences between the default model step size(s) and the cycle time of the assigned task(s).
	ExecutionSequence	Default execution sequence	UpdateBeforeOutputMapping	Configure the execution order of input mapping, model code execution and output mapping.
	ExecuteModelCode	Execute model code after startup	TRUE	Start cyclic execution of the model code after startup by default. If FALSE, Module Parameter Execute needs to be set to TRUE to start execution of code.
	BlockDiagramExport	Export BlockDiagram	TRUE	Export graphical block diagram information for monitoring and optional debugging on the generated TwinCAT module in TwinCAT XAE [▶ 198]
	ResolveMaskedSubsystems	Resolve Masked Subsystems	FALSE	Resolve masked subsystems in the block diagram
	ExtendSignalResolution	Extended resolution of signals in block diagram	FALSE	Intensified search for assignments of variables and block diagram signals (blue signals). This option increases the build time. [▶ 240]
	BlockDiagramVariableAccess	Access to VariableGroup not referenced by any block	AssignToParent	Variables from a block within an unresolved subsystem are either assigned to the next higher visible block or hidden in the block diagram.
	BlockDiagramDebugInfoExport	Export BlockDiagram debug info	TRUE	Export additional information required to debug the module using the block diagram [▶ 202].
TC TcCom Interfaces	ExecutionInfoOutput	Create ExecutionInfo output	FALSE	Create additional output DataAreas containing execution and exception information [▶ 225].
	MonitorExecutionTime	Monitor execution time	FALSE	Calculate and expose the execution time of the module as an ADS variable for monitoring purposes.
	InputDataAccess	Input: Data Access	Input Destination DataArea	Defines how the input variables are exposed in TwinCAT [▶ 147].
	InputCreateSymbols	Input: Create ADS Symbols	TRUE	Create ADS symbol information for the input variables [▶ 147]
	InputInitValues	Input: Initial values	FALSE	Create module parameters for the input variables to allow definition of initial values [▶ 147]

Category	Name	Displayname	Default	Description
	InputProperties	Input: TMC Properties		Additional properties added to the Input symbol description in the TMC file. [▶ 171]
	OutputDataAccess	Output: Data Access	Output Source DataArea	Defines how the output variables are exposed in TwinCAT [▶ 147].
	OutputCreateSymbols	Output: Create ADS Symbols	TRUE	Create ADS symbol information for the output variables [▶ 147].
	OutputProperties	Output: TMC Properties		Additional properties added to the Output symbol description in the TMC file. [▶ 171]
	ParametersDataAccess	Parameters: Data Access	Internal DataArea	Defines how the model parameter variables are exposed in TwinCAT [▶ 147]
	ParametersCreateSymbols	Parameters: Create ADS Symbols	TRUE	Create ADS symbol information for the model parameter variables [▶ 147].
	ParametersInitValues	Parameters: Initial values	TRUE	Create module parameters for the model parameter variables to allow definition of initial values [▶ 147].
	ParametersProperties	Parameters: TMC Properties		Additional properties added to the Parameters symbol description in the TMC file. [▶ 171]
	BlockIoDataAccess	BlockIO: Data Access	Internal DataArea	Defines how the BlockIO variables are exposed in TwinCAT [▶ 147]
	BlockIoCreateSymbols	BlockIO: Create ADS Symbols	TRUE	Create ADS symbol information for the BlockIO variables [▶ 147].
	BlockIoProperties	BlockIO: TMC Properties		Additional properties added to the BlockIO symbol description in the TMC file. [▶ 171]
	ContStateDataAccess	ContState: Data Access	Internal DataArea	Defines how the continuous state variables are in TwinCAT [▶ 147]
	ContStateCreateSymbols	ContState: Create ADS Symbols	TRUE	Create ADS symbol information for the continuous state variables [▶ 147].
	ContStateProperties	ContState: TMC Properties		Additional properties added to the ContState symbol description in the TMC file. [▶ 171]
	DWorkDataAccess	DWork: Data Access	Internal DataArea	Defines how the DWork variables are exposed in TwinCAT [▶ 147]
	DWorkCreateSymbols	DWork: Create ADS Symbols	TRUE	Create ADS symbol information for the DWork variables [▶ 147].
	DWorkProperties	DWork: TMC Properties		Additional properties added to the DWork symbol description in the TMC file. [▶ 171]
	DataStoreDataAccess	DataStore: Data Access	None	Defines how the DataStore variables are exposed in TwinCAT [▶ 147]

Category	Name	Displayname	Default	Description
	DataStoreCreateSymbols	DataStore: Create ADS Symbols	TRUE	Create ADS symbol information for the DataStore variables [▶ 147].
	DataStoreReadOnly	DataStore: Read Only	FALSE	Restrict ADS access to be read only for the DataStore variables [▶ 147].
	DataStoreProperties	DataStore: TMC Properties		Additional properties added to the DataStore symbol description in the TMC file. [▶ 171]
	SymbolProperties	Additional TMC Symbol Properties		Additional properties added to specific symbol descriptions in the TMC file. [▶ 171]
	VariableSymbolMapping	Mapping between variable names and ADS symbol names	Identical	Defines the TwinCAT symbol names for the generated C/C++ variables. 'Identical': Symbol name equals variable name, 'Classic': Use symbol names known from TE1400 Release 1.2.x.x [▶ 147]
TC TcCom External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [▶ 222].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode [▶ 222] connection before starting model code execution.
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [▶ 222].
TC PlcFb General	Generate	Generate TwinCAT PLC Function Block	TRUE	Generate a PLC-FB for the model [▶ 219].
	InitExceptionHandling	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during initialization [▶ 225].
	UpdateExceptionHandling	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during cyclic execution [▶ 225].
TC PlcFb License	OemLicenses	IDs of OEM License	$<Project:OemLicenses>	IDs of OEM Licenses. Multiple IDs may be inserted as a comma seperated list. "{GUID},{GUID}" [▶ 161]
TC PlcFb Additional settings	MonitorExecutionTime	Monitor ExecutionTime	FALSE	Calculate and expose the execution times of TwinCAT modules as an ADS variable for monitoring purposes.
PlcFb->Interface	InputAttributes	Input variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
	OutputAttributes	Output variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.

Category	Name	Displayname	Default	Description
TC PlcFb External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [▶ 222].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode connection before starting model code execution [▶ 222].
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [▶ 222].

Platzhalter auf Ebene des Targets/Projekts (Modulgenerator)

Platzhalter dieser Gruppe können auf Ebene des Projekts und der Module verwendet werden.

Placeholder name	Description
ModuleGenerator:ProductName	Product name of the module generator
ModuleGenerator:Version	Version of the module generator
TwinCAT:Version	Version of local TwinCAT installation
UsablePlatformToolsets	Available and supported platform toolsets
LocalDateTime:Format	Actual local time as string where Format must be defined like the format string for std::put_time (e.g. '%Y-%m-%d')
UtcDateTime:Format	Actual UTC time as string where Format must be defined like the format string for std::put_time (e.g. '%Y-%m-%d')
EnvironmentVarName	Any environment variable defined for the system, the current user or the current process (MATLAB®).

Platzhalter auf Ebene des Projekts

Platzhalter dieser Gruppe können auf Ebene des Projekts und der Module verwendet werden.

Placeholder name	Description
Project:Name	Name of the project file (without directory and extension)
Project:Dir	Project file directory
Project:Ext	Project file extension
Project:Path	Full project file path
Project:Guid	Project GUID
Project:LibraryID	LibraryID of the generated repository driver
Project:VendorName	Company name part of the LibraryID
Project:DriverName	Driver name part of the LibraryID
Project:DrvFileVersion	Version part of the LibraryID
Project:LatestTMFile	Path to an existing corresponding TML or TMC file with the highest library version (searching project directory and repository)
Project:LatestTMFile:Repository	Path to an existing corresponding TML or TMC file with the highest library version (searching only repository)
Project:LatestTMFile:ProjectDir	Path to an existing corresponding TML or TMC file with the highest library version (searching only project directory)
Project:VersionFromFile	Version read from file defined by "VersionSrc"

Platzhalter auf Ebene der Module (TcCOM und PLC FB)

Platzhalter dieser Gruppe können nur auf Ebene der Module genutzt werden.

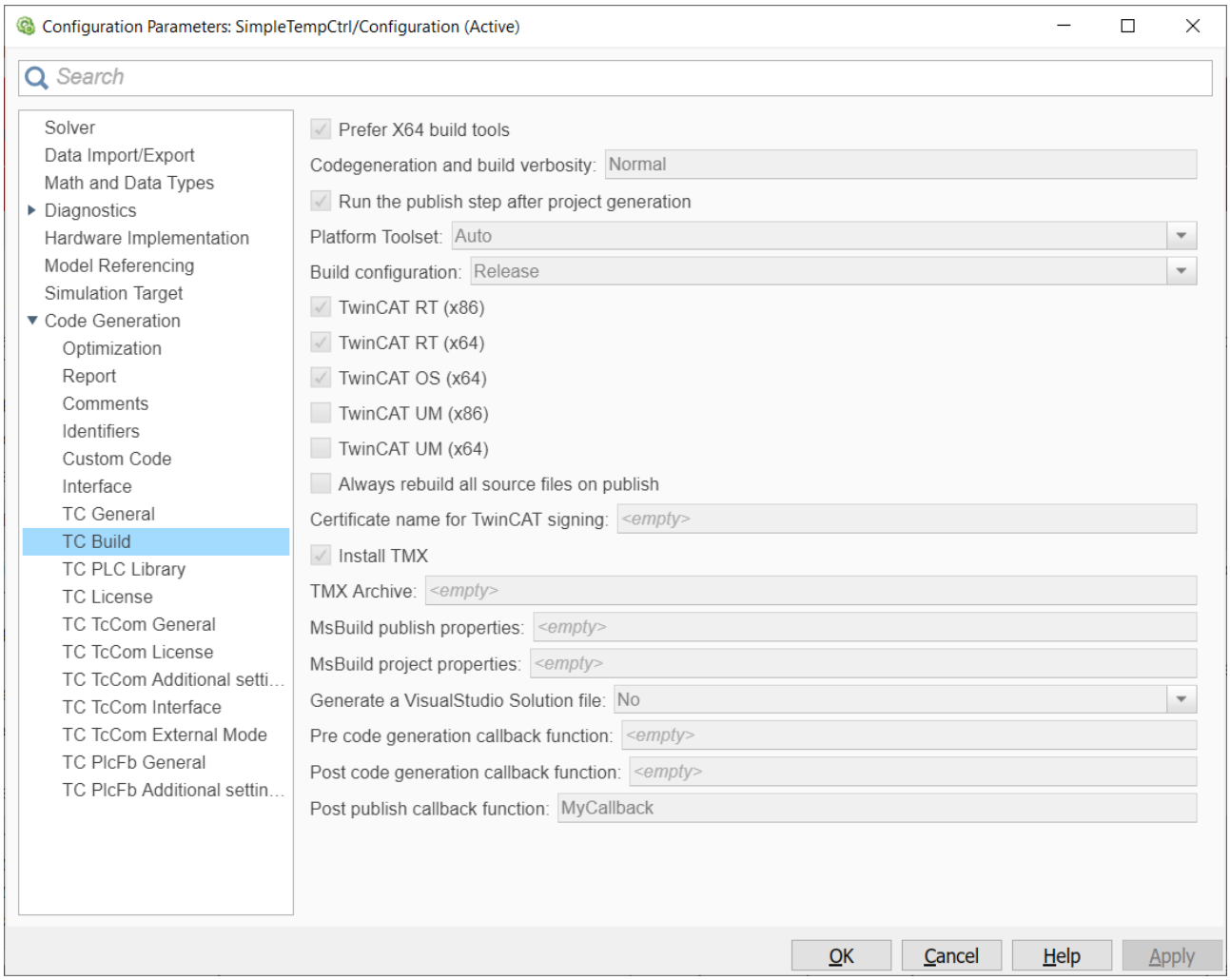
Placeholder name	Description
Module:Name	Name of the TcCom module or PLC FB
Module:ClsId	Class ID of the TcCom module (TcCom only)
Module:ContextCount	Number of task contexts
Module:ClassName	Name of the TcCom module
Module:CppClassFileName	Name of the corresponding .h and .cpp files
Module:ModelName	Name of the corresponding Simulink Model (TE1400 only)
Module:MFileName	Name of the corresponding M-File (TE1401 only)
Module:FileFilterName	Visual Studio project filter name

4.7.14 Arbeiten mit Callbacks

Es existieren drei unterschiedliche Callback Funktionen:

- **Pre code generation callback function:** Callback bevor das Modell in C++-Code umgesetzt wird.
- **Post code generation callback function:** Callback nachdem das Modell in C++-Code umgesetzt wurde.
- **Post publish callback function:** Callback nachdem das erstellte C++-Projekt für die konfigurierten Plattformen gebaut wurde.

Tragen Sie hier den Namen Ihrer erstellten MATLAB®-Funktion ein, um diese aufzurufen.



Ihre MATLAB®-Funktion bekommt als Übergabeparameter das ProjektExporter-Objekt übergeben:

```
function MyCallback(obj)
…
return
```

Das Objekt trägt in dessen Properties die aktuelle Konfiguration des Builds.

```
ProjectExporter with properties:
ProjectGenerator: [1×1 TwinCAT.ModuleGenerator.ProjectGenerator]
Configuration: [1×1 TwinCAT.ModuleGenerator.ProjectExportConfig]
Project: [1×1 TwinCAT.ModuleGenerator.Project]
State: [1×1 struct]
ClassExporters: {[1×1 TwinCAT.ModuleGenerator.Simulink.ModelExporter]}
AdditionalExports: [1×1 containers.Map]
```

Beispielcode in MATLAB®

Öffnen Sie das passende Beispiel mit:
`TwinCAT.ModuleGenerator.Samples.Start('Callbacks')`

4.8 Anwendung von Modulen in TwinCAT

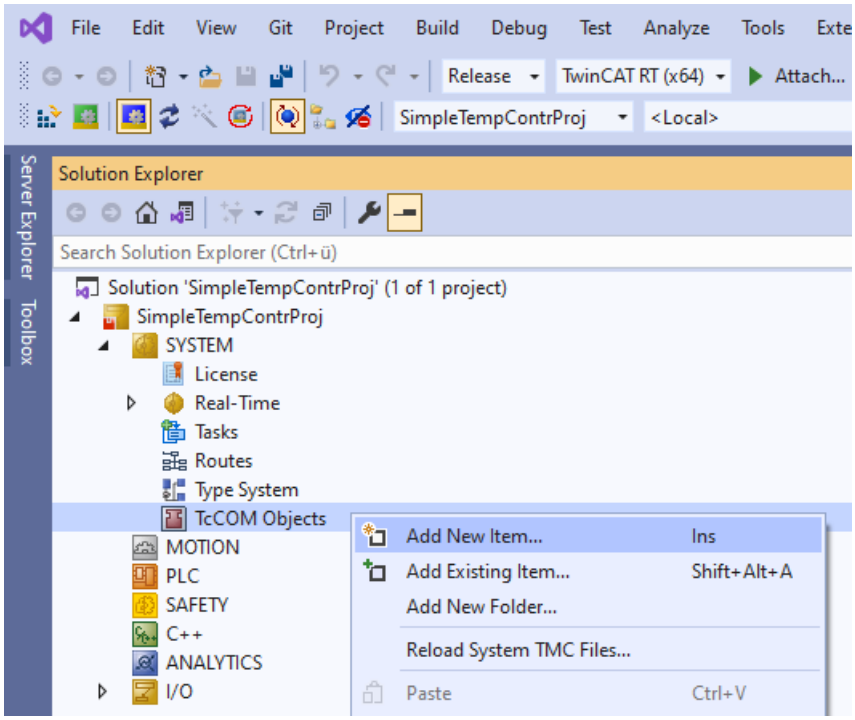
Mit dem Target for Simulink® erstellte TcCOM und Funktionsbausteine lassen sich nahtlos im TwinCAT XAE verwenden.

Einzige Voraussetzung für die Verwendung auf einem beliebigen TwinCAT-XAE-System ist die Verwendung von TwinCAT XAE Version 3.1.4024.7 und höher. MATLAB®, eine volle Visual Studio Installation usw. sind nicht notwendig, da Sie mit bereits für TwinCAT kompilierten Objekten und Beschreibungsdateien arbeiten. Kopieren Sie einfach den Engineering Repository-Ordner auf das Engineering System Ihrer Wahl. Vergleiche TwinCAT Objekte [▶ 124]. Behalten Sie dabei die Ordnerstruktur immer bei: *%TwinCATInstallDir% \3.1\Repository\<TE140x Module Vendor>\<ModelName>\<Version>*.

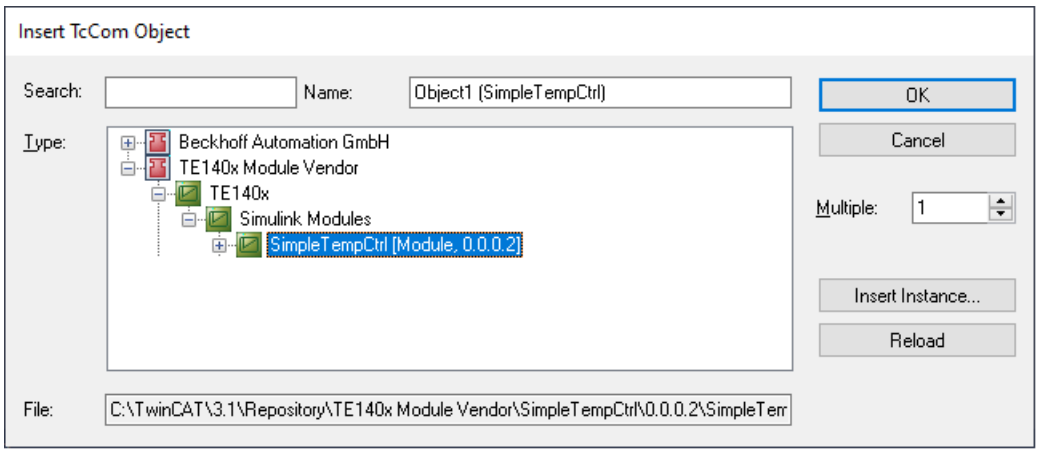
4.8.1 Arbeiten mit dem TcCOM-Modul

TcCOM in TwinCAT einfügen

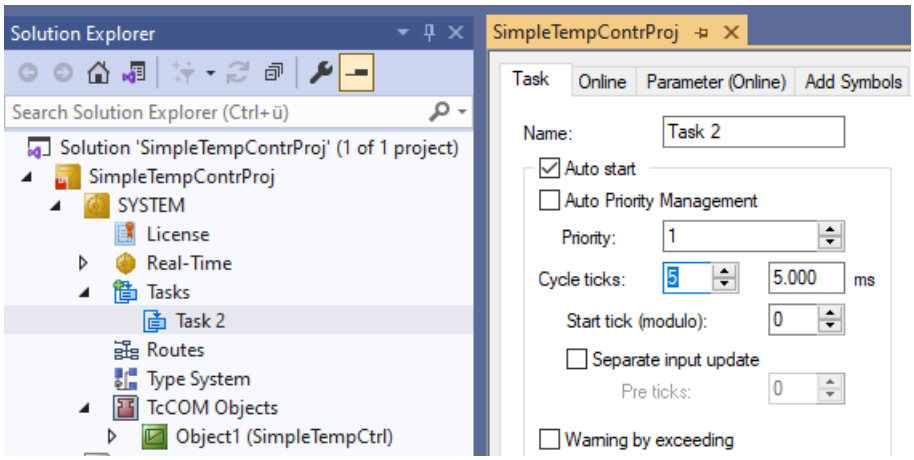
1. Öffnen Sie TwinCAT (TwinCAT XAE oder TwinCAT in einer Visual-Studio-Umgebung).
2. Instanziieren Sie ein neues TcCOM-Objekt.



3. Wählen Sie das gewünschte Objekt.

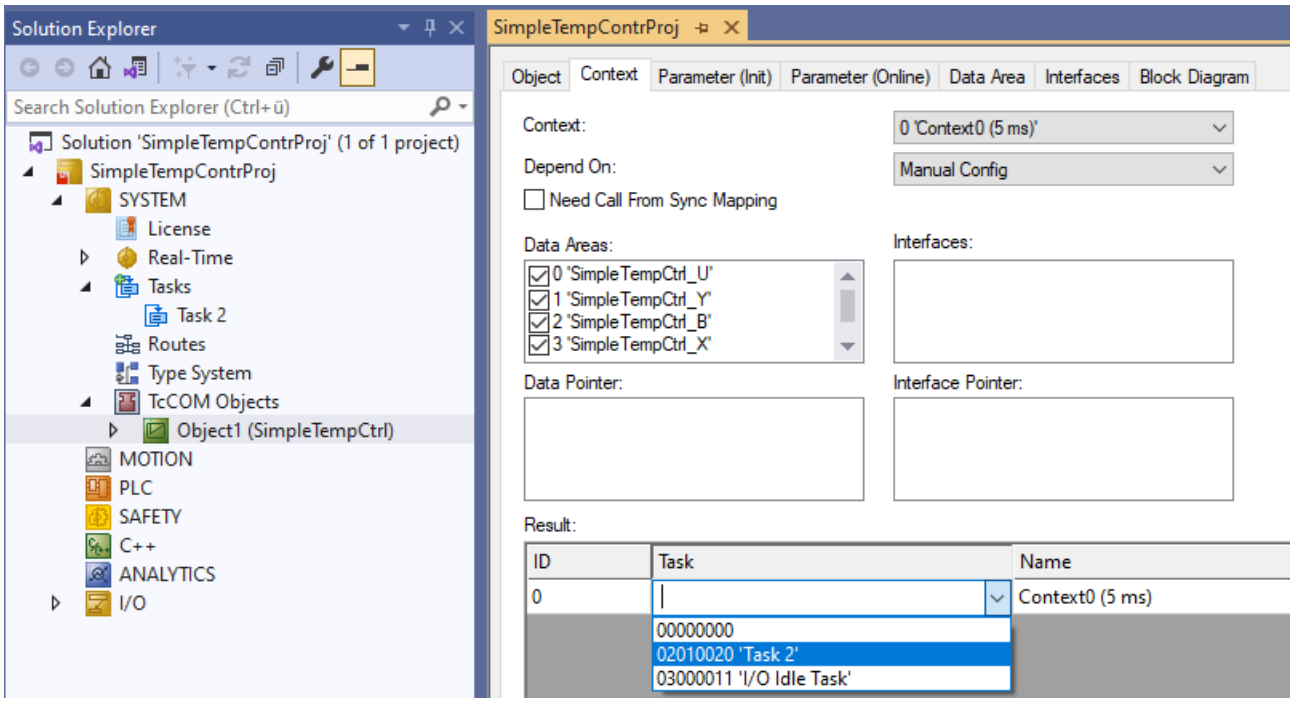


4. Erstellen Sie eine zyklische Task.

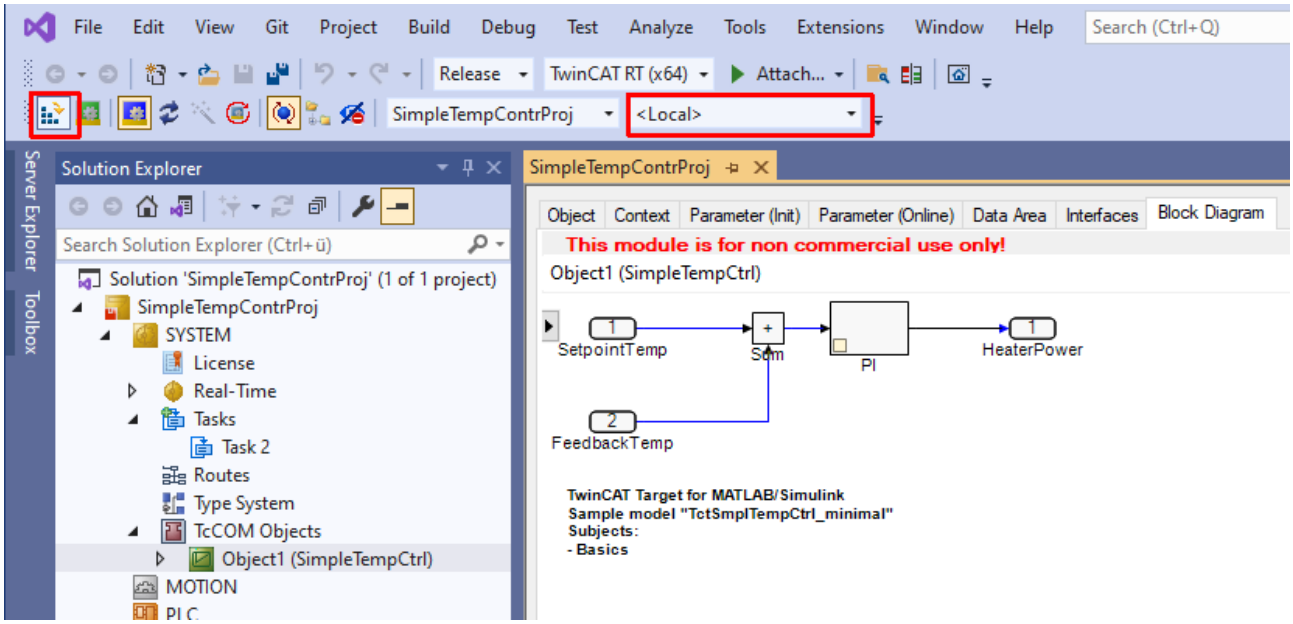


5. Weisen Sie die erstelle Task ihrer TcCOM-Instanz zu.

Beachten Sie, dass die Zykluszeit der Task und die SampleTime in Simulink® (hier 5 ms) passen.

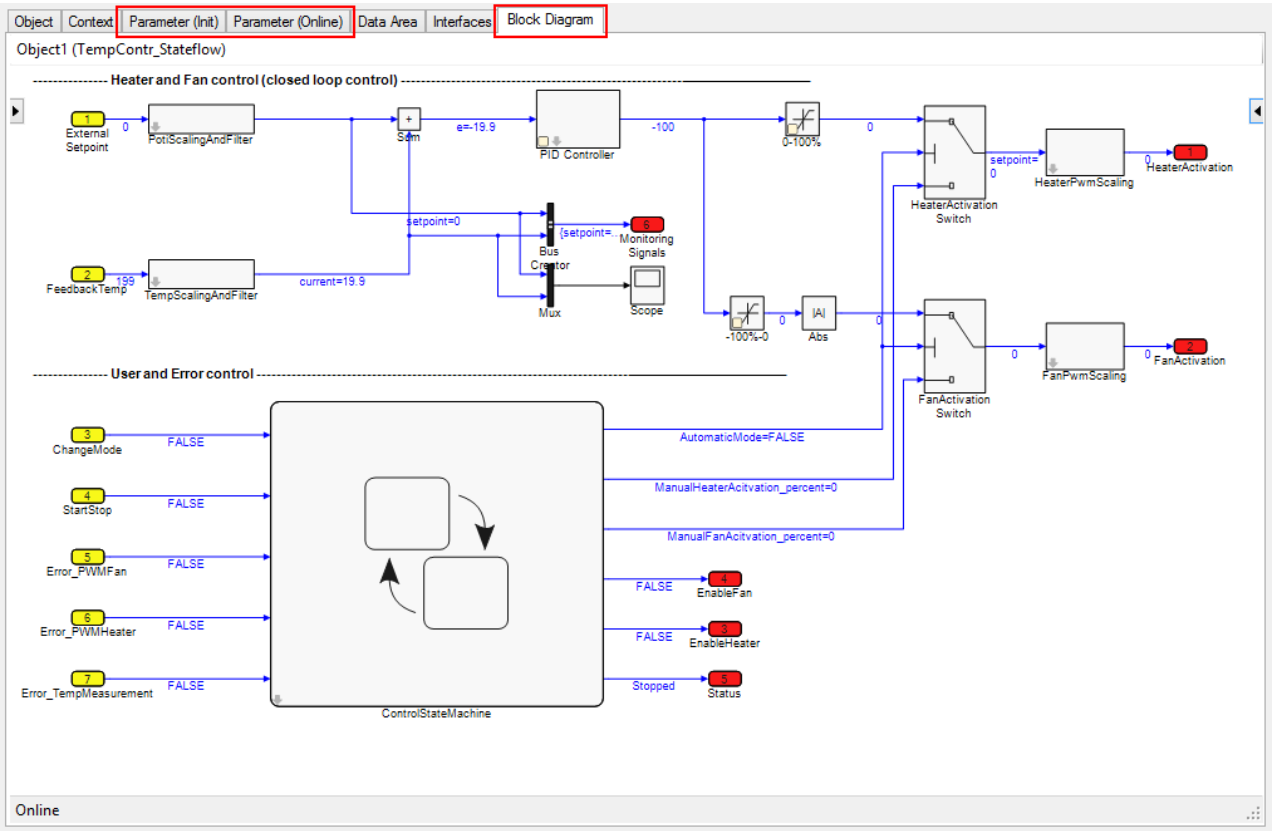


6. Aktivieren Sie die Konfiguration.



4.8.1.1 Parametrierung einer Modulinstanz

Gegeben ist eine Instanz eines TcCOM, unter welchen Tabs sind Parameter zu finden?



Parameter können Sie unter **Parameter (Init)**, **Parameter (Online)** und im **Block Diagramm** einsehen und verändern. Unter **Data Area** können Sie zwar die angelegten DataAreas und deren Inhalt (Parametername und Datentyp) sehen, können hier aber keine Werte über das XAE manuell verändern.

Siehe auch Best Practice: Zugriff auf Daten des TcCOM [▶ 151] für Möglichkeiten des Datenzugriff auf ein TcCOM.

Default-, Startup-, Online- und Prepared-Werte

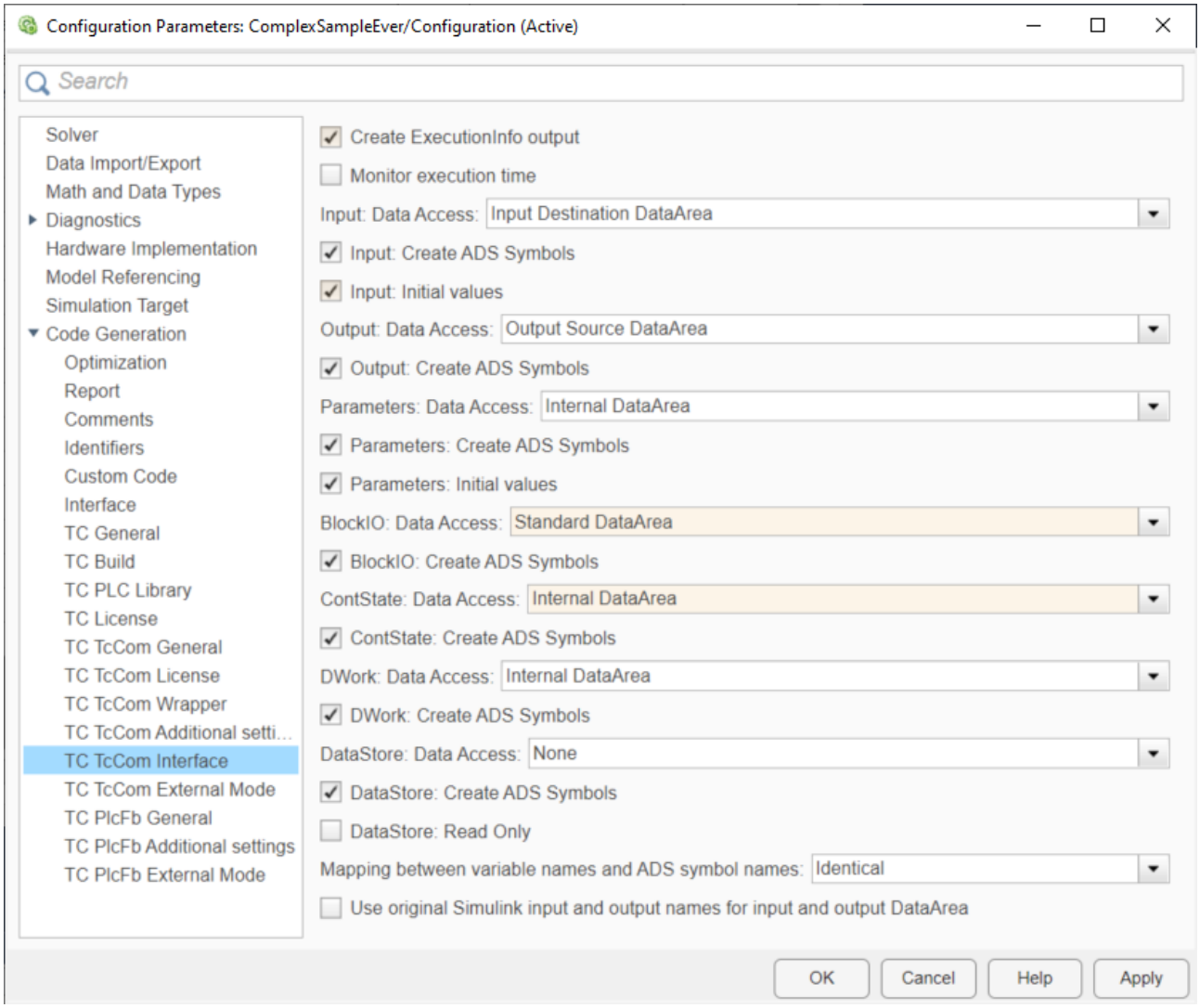
Parameter können unterschiedliche Zustände/Eigenschaften haben. Diese werden im Folgenden definiert:

- **Default-Werte** sind die Parameterwerte beim Generieren des Codes (wie sie in Simulink® gesetzt wurden). Sie sind unveränderlich in der Modulbeschreibungsdatei (*.tmc), also der Beschreibung der Modul-Klasse, gespeichert.
- **Startup-Werte** werden in der TwinCAT-Projektdatei gespeichert und in die Modulinstanz geschrieben. Die Startup-Werte liegen entsprechend auf dem Zielsystem und definieren die Werte beim Start der Modulinstanz.
- **Online-Werte** sind nur verfügbar, wenn die Instanz des TcCOM auf dem Zielsystem gestartet wurde. Sie zeigen den aktuellen Wert des Parameters im laufenden Modul. Dieser kann auch während der Laufzeit geändert werden.
- **Prepared-Werte** können immer dann festgelegt werden, wenn auch Onlinewerte verfügbar sind. Mit ihrer Hilfe können mehrere Parameter-Werte geändert und gleichzeitig in das Modul geschrieben werden.

Parametrierung der Modulinstanz

Beispieleinstellung unter TC TcCom Interface

Um die Parametrierung einer Modulinstanz zu erläutern, wird von einem Modul mit folgenden Eigenschaften ausgegangen.



Eigenschaften des TcCOM aufgrund dieser Einstellungen:

- Die Struktur der Modell-Parameter <ModelName>_P ist als Modul Parameter angelegt, da Parameters: Initial Values aktiviert ist. Als DataArea sind die Modell-Parameter erreichbar, wenn Code Interface packaging nicht auf *Reusable function* steht.

Modell-Parameter als Tunable eintragen

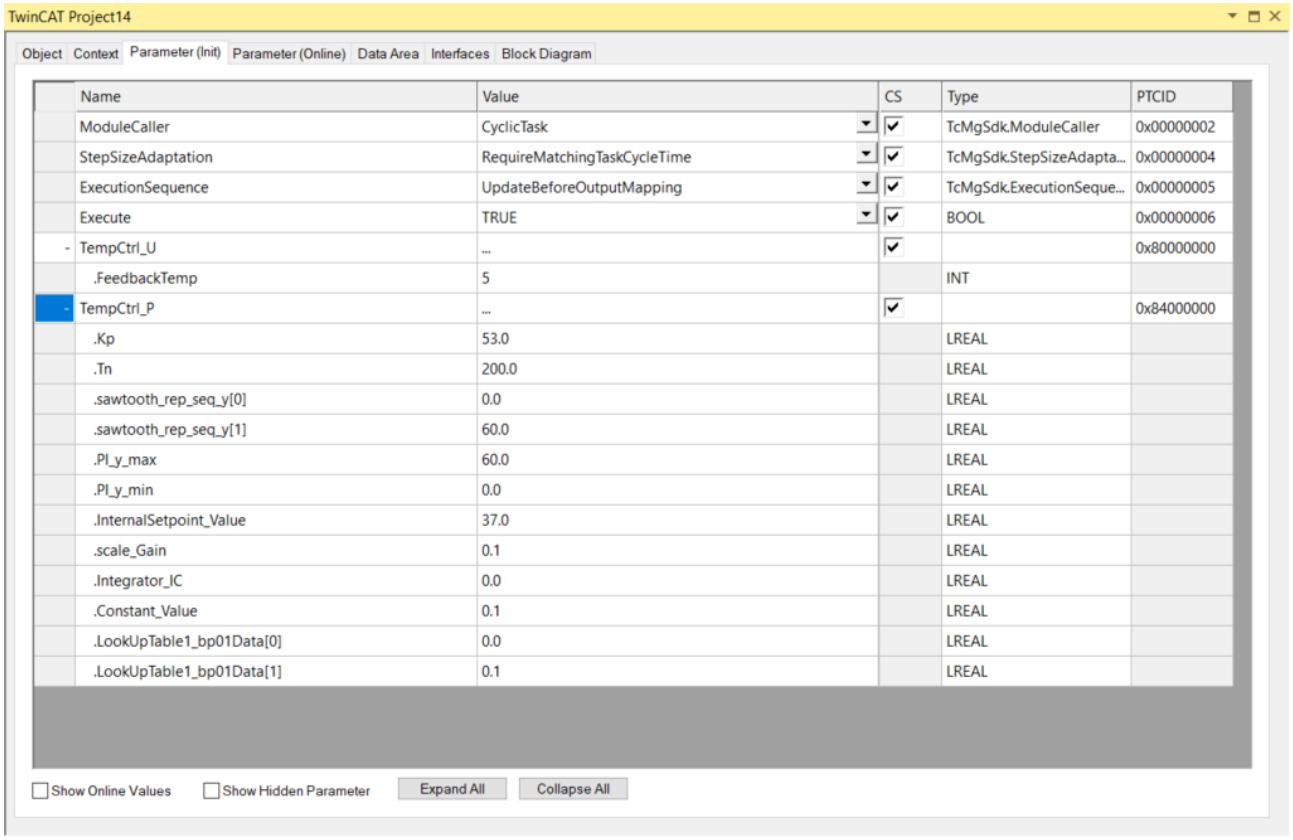
Als Default-Verhalten werden vom Simulink Coder™ Modell-Parameter als „inlined“ gesetzt. Damit ist der verwendete Speicher des Objekts kleiner und der generiete Code hinsichtlich der Laufzeit optimiert, jedoch sind mit dieser Einstellung oft nur wenige Parameter zur Laufzeit veränderbar. Verwenden Sie in den *Configuration Parameter* unter *Optimization* > *Default parameter behavior* „Tunable“, damit Sie zur Laufzeit ihr Modell parametrieren können.

- BlockIO wird als Standard DataArea angelegt und ist somit im Prozessabbild des TcCOM sichtbar und kann per DataPointer verknüpft werden.
- Create ExecutionInfo output ist aktiv, entsprechend existiert eine weitere DataArea vom Typ Output Source mit den Execution Informationen der Instanz.
- Create ADS Symbols ist auf allen DataAreas aktiviert, damit sind alle Parameter in den DataAreas per ADS Symbol Name erreichbar.
- Für die Inputs des Modells werden Modul-Parameter angelegt, da Input: Initial Values aktiviert ist.

Beschreibung der Einstellungsoptionen, siehe Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts [▶ 147].

Mögliche Parameter-Einstellungen unter Parameter (Init)

Unter Parameter (Init) sind Modul-Parameter aufzufinden, die sich nicht zyklisch ändern sondern azyklisch verändert werden können. Im Config-Mode, bzw. wenn das TcCOM nicht gestartet ist, sind keine Online-Werte zu sehen.



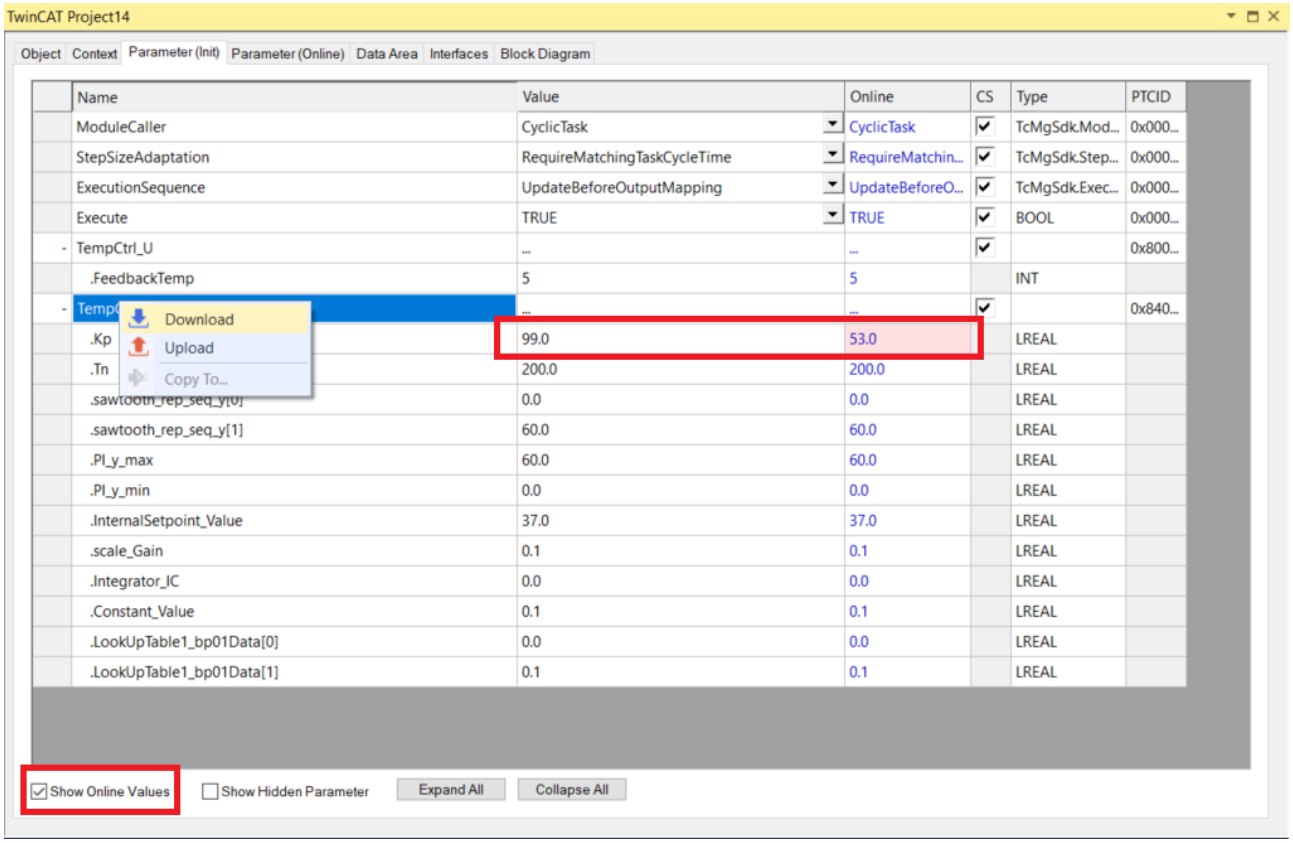
Startup Values definieren

Startup Values können Sie in der **Parameter (Init)**-Darstellung oder in der **Block Diagram**-Darstellung (siehe fortlaufendes Kapitel) setzen. Wählen Sie dazu in der Spalte **Value** den Eintrag aus, den Sie auf der Modulinstanz anpassen möchten und tragen Sie einen Wert ein. Der so eingestellte Wert liegt zu diesem Zeitpunkt nur in der Projektdatei im XAE vor. Mit Activate Configuration wird der eingestellte Wert (mit dem Gesamtprojekt) auf das Zielsystem geladen.

Einstellungen, wie z.B. *ModuleCaller*, *StepSizeAdaption* oder *ExecutionSequence*, können Sie nicht zur Laufzeit des Moduls verändern, sondern sie sind nur als Startup Values definierbar. Andere Modul-Parameter, wie Execute oder die Modell-Parameter, können Sie darüber hinaus auch noch Online verändern.

Online Werte verändern

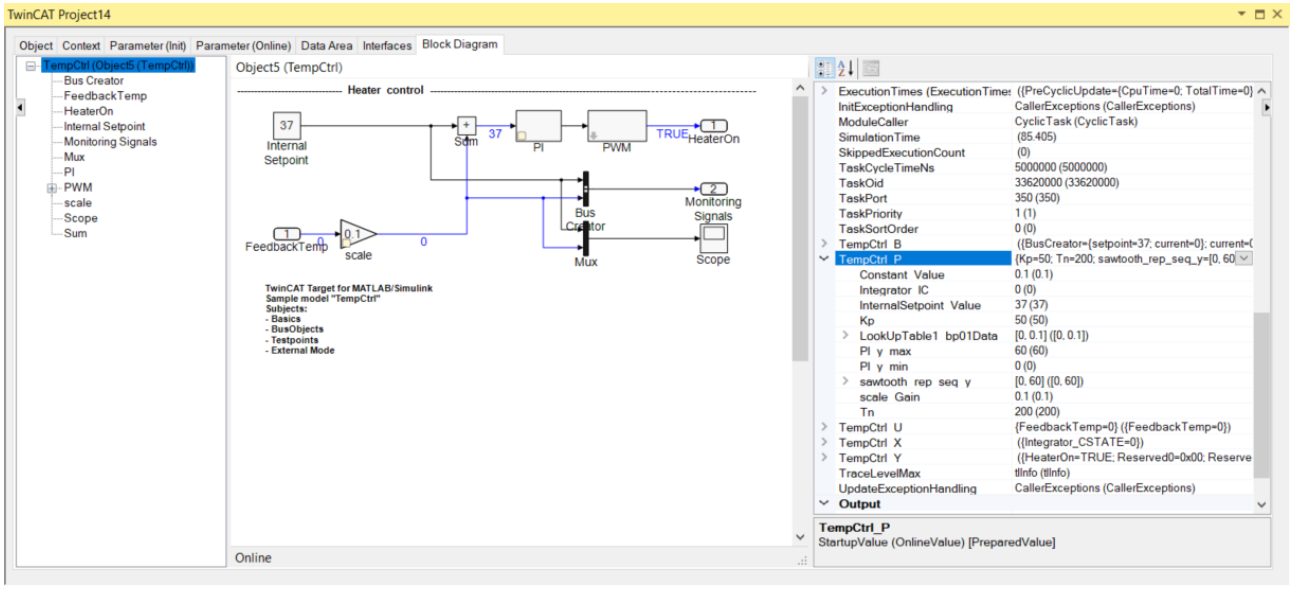
Ist das TcCOM Modul aktiv, können Sie mit **Show Online Values** die aktuellen Werte der Instanz sichtbar machen. Sie können in der Spalte **Value** einen neuen Wert eintragen; somit ist er ein Prepared Value. Ist der Prepared Value noch nicht auf das Zielsystem geladen, erscheint das Feld mit Abweichung zwischen Value und Online mit roter Markierung. Sind alle Änderungen vorgenommen, können Sie auf Ebene der Struktur einen Rechtsklick durchführen und mit „Download“ die Prepared Values im Zielsystem setzen. Beachten Sie, dass Sie dadurch nicht den Startup Value der Instanz auf dem Zielsystem ändern. Dazu müssen Sie erst die aktuelle Konfiguration auf dem Zielsystem aktivieren. Ebenso können Sie mit „Upload“ die aktuellen Online Values als Startup Values im Projekt setzen. Auch hier gilt, dass die Änderung erst auf dem Laufzeitsystem aktiv ist, wenn das Projekt kompiliert und heruntergeladen ist, vgl. Best Practice: Zugriff auf Daten des TcCOM [▶ 151].



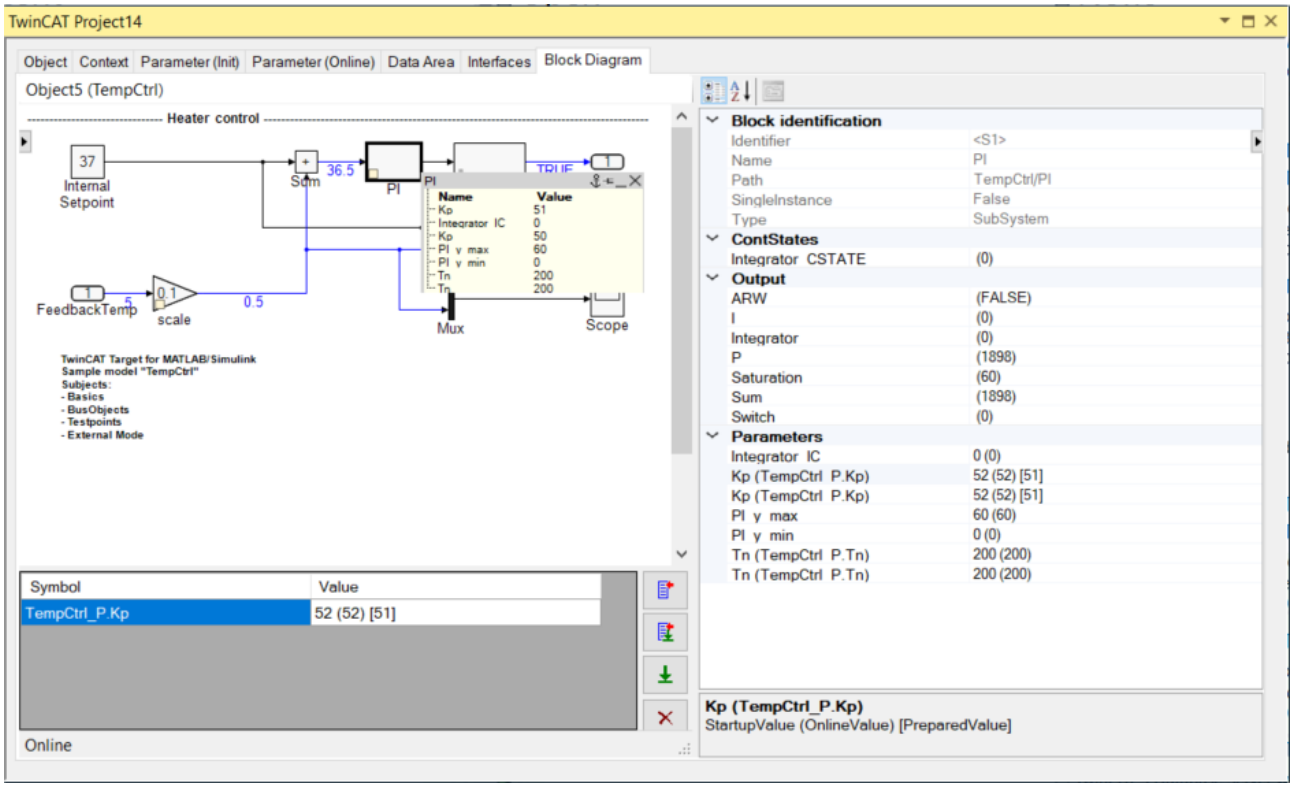
Mögliche Parameter-Einstellungen im Block Diagram

Im TwinCAT 3 Block Diagram werden alle Parameter im Parameterbereich (rechte Seite des Fensters) nur angezeigt, wenn Sie sich in der obersten Ebene des Blockdiagramms ("<root>“) befinden. Sind Sie in einem Subsystem oder haben Sie einen Block selektiert, werden nur die Parameter des aktiven Blocks angezeigt.

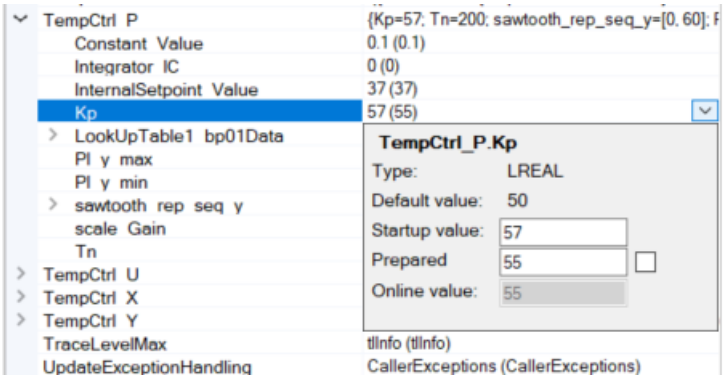
In der Darstellung des Block Diagrams haben Sie die Möglichkeit, Online Werte zu lesen und zu schreiben, sowie Startup Values zu verändern.



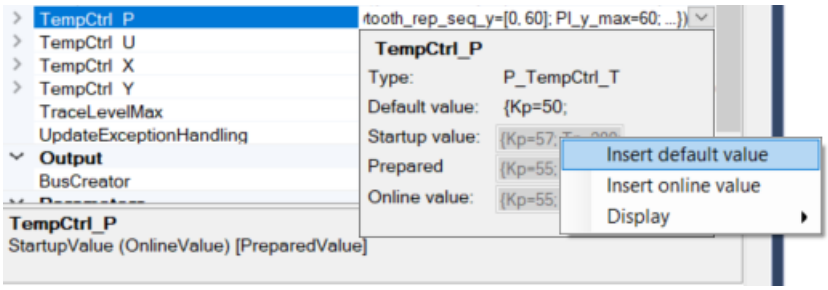
Selektieren Sie einen Block mit einem gelben Punkt in der unteren linken Ecke. Klicken Sie direkt auf den gelben Punkt, so öffnet sich ein Kontextmenü in dem Sie die Online Werte des Blocks sehen und auch verändern können. Verändern Sie einen Wert wird er in der Prepared Liste eingetragen. Wenn Sie alle Änderungen getätigt haben, können Sie die vorbereiteten Werte auf das Zielsystem schreiben. Sie können in der Prepared Liste auswählen, ob Sie die Prepared Werte als Online und/oder Startup Values setzen möchten.



Sie können auch über die Parameter-Liste im rechten Bereich des Blockdiagramms gehen und hier Werte verändern. Suchen Sie nach dem Parameter, den Sie verändern möchten und klicken Sie auf den Pfeil nach unten am rechten Rand der Liste. Hier können Sie in den editierbaren Feldern Änderungen vornehmen. Gehen Sie mit dem Mauszeiger über den Parameternamen (in folgender Grafik TempCtrl_P.Kp), so werden Ihnen die ADS-Adressinformationen angezeigt. Durch einen Rechtsklick auf den Parameternamen können Sie die ADS-Adressinformationen des Parameters in die Zwischenablage kopieren.



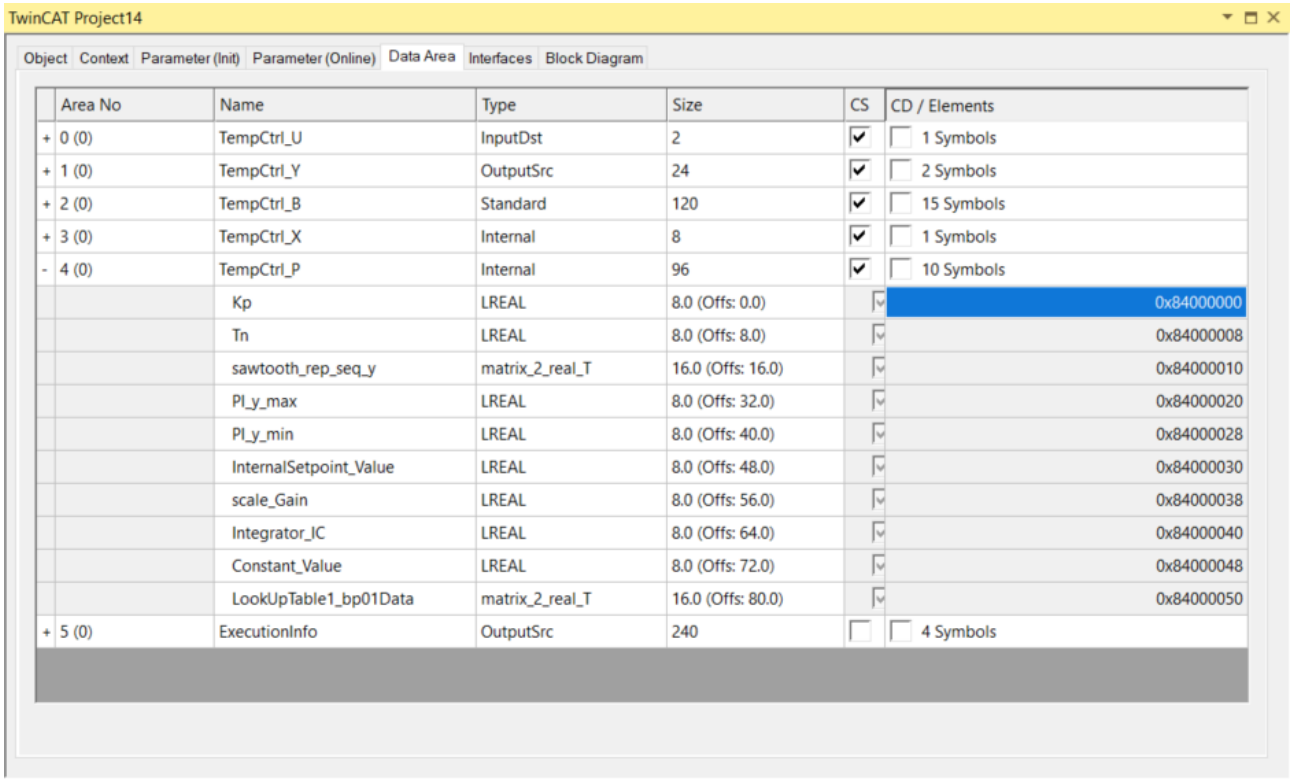
Sie können auch ganze Strukturen verändern. Wählen Sie Beispielsweise die <ModelName>_P Struktur, d. h. die Struktur, die alle Modell-Parameter enthält, und wählen Sie hier den nach unten zeigenden Pfeil aus. Durch Rechtsklick, z. B. auf **Startup Values**, erscheint ein Kontextmenü. Hier können Sie z. B. auswählen, dass Sie alle aktuellen Online Werte der Struktur als Startup Values setzen möchten, oder Sie können die Startup-Liste wieder auf die Default-Werte zurücksetzen.



Interaktion mit DataAreas

Über das XAE können Sie DataAreas nicht verändern. Sie können lediglich einsehen, um was für einen Typ DataArea es sich handelt (siehe Spalte Type) und daraus ableiten, welche Interaktionsmöglichkeiten mit der DataArea möglich sind, siehe Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts [▶ 147].

Über die Spalte CS können Sie sehen, ob für diese DataArea ADS Symbolname generiert werden sollen. CS ist editierbar durch den Nutzer. Ist CS aktiv, können Sie bspw. den Target Browser verwenden, um die ADS-Symbole zu suchen und in eine Scope Konfiguration einbinden. Ist CS deaktiviert, können Sie nur per Index Group und Index Offset auf die Daten der DataArea zugreifen. Index Group ist dabei die Object-ID der Instanz (siehe Object-Tab) und Indexoffset wird in der Spalte CD/Elements angezeigt.



	Area No	Name	Type	Size	CS	CD / Elements
+	0 (0)	TempCtrl_U	InputDst	2	☑	1 Symbols
+	1 (0)	TempCtrl_Y	OutputSrc	24	☑	2 Symbols
+	2 (0)	TempCtrl_B	Standard	120	☑	15 Symbols
+	3 (0)	TempCtrl_X	Internal	8	☑	1 Symbols
-	4 (0)	TempCtrl_P	Internal	96	☑	10 Symbols
		Kp	LREAL	8.0 (Offs: 0.0)		0x84000000
		Tn	LREAL	8.0 (Offs: 8.0)		0x84000008
		sawtooth_rep_seq_y	matrix_2_real_T	16.0 (Offs: 16.0)		0x84000010
		PI_y_max	LREAL	8.0 (Offs: 32.0)		0x84000020
		PI_y_min	LREAL	8.0 (Offs: 40.0)		0x84000028
		InternalSetpoint_Value	LREAL	8.0 (Offs: 48.0)		0x84000030
		scale_Gain	LREAL	8.0 (Offs: 56.0)		0x84000038
		Integrator_IC	LREAL	8.0 (Offs: 64.0)		0x84000040
		Constant_Value	LREAL	8.0 (Offs: 72.0)		0x84000048
		LookUpTable1_bp01Data	matrix_2_real_T	16.0 (Offs: 80.0)		0x84000050
+	5 (0)	ExecutionInfo	OutputSrc	240	☐	4 Symbols

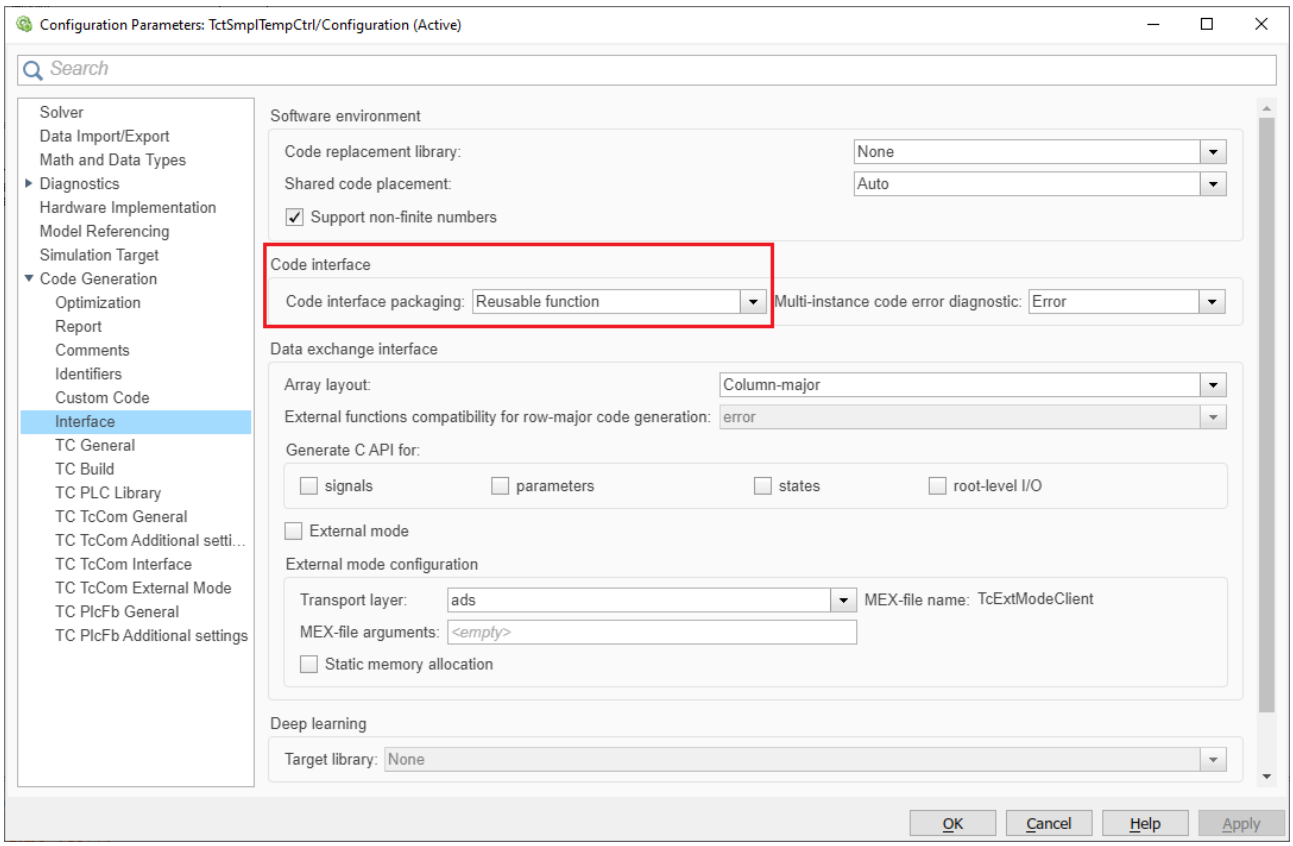
Für weitere Informationen zur Interaktion mit DataAreas, siehe Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts [▶ 147] und Best Practice: Zugriff auf Daten des TcCOM [▶ 151].

4.8.1.2 Parametrierung mehrerer Modulinstanzen

Im obigen Teil wurde beschrieben, dass eine Instanz eines TcCOM, auch abweichend zu den Parametern in Simulink®, in TwinCAT parametriert werden kann.

Wenn mehrere Instanzen eines TcCOM in einer TwinCAT-Solution genutzt werden, existieren unterschiedliche Möglichkeiten bezüglich der individuellen Parametrierung der Instanzen. Um die folgenden drei Möglichkeiten umzusetzen, müssen Sie in Simulink® die Einstellung **Code interface packaging** nutzen.

Achten Sie darauf, dass unter *Optimization* die Einstellung *Default parameter behavior* auf *Tunable* steht.



✓ **Alle Instanzen sollen dieselben Parameter besitzen.**

1. Stellen Sie den Parameter auf „Reusable function“. Dies ist der Standardwert bei Auswahl des Target *TwinCatGrt.tlc*.
2. Erzeugen Sie in TwinCAT mehrere Instanzen ihres TcCOM.
3. Konfigurieren Sie unter Parameter (Init) den Parameter <ModelName>_P_Sharing zu *define* oder *inherit*. Define gibt die Parametrierung aller abhängigen Instanzen, die mit *inherit* konfiguriert sind vor.

⇨ Es darf nur eine Instanz mit *define* konfiguriert sein.

✓ **Jede Instanz soll individuell parametriert werden können.**

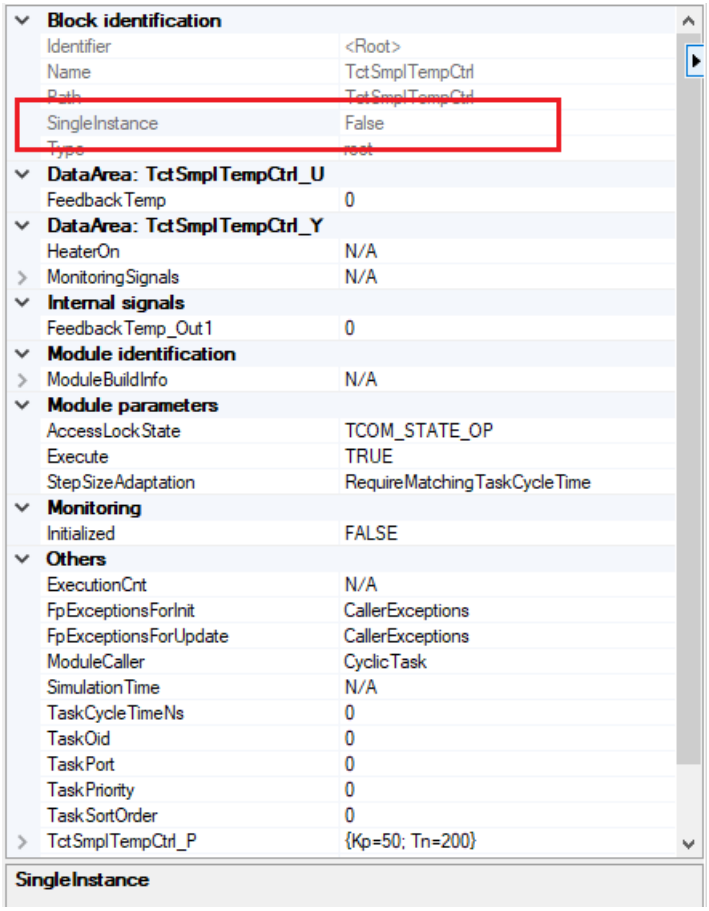
1. Stellen sie den Parameter auf „C++ class“.
2. Erzeugen Sie in TwinCAT mehrere Instanzen ihres TcCOM.

⇨ Es wird kein Parameter <ModelName>_P_Sharing erzeugt. Jede Instanz kann individuell parametriert werden.

✓ **Es soll nur eine einzige Instanz im Projekt zugelassen werden.**

1. Stellen Sie den Parameter auf „Nonreusable function“.
   ⇨ Wenn Sie in TwinCAT mehrere Instanzen ihres TcCOM erzeugen, erhalten Sie eine Fehlermeldung beim Aktivieren der Solution.
   ⇨ Ob eine Instanz eines TcCOM mehrfach instanziierbar ist, kann im TC3 BlockDiagram eingesehen werden.
2. Gehen Sie dazu auf den Parameterbereich auf der rechten Seite. Unter Block Identification ist ein Parameter „SingleInstance“ sichtbar.

⇨ Der Wert False bedeutet mehrfach instanziierbar. True entsprechend nur einmal instanziierbar.



Einstellungen gelten ebenso für die Nutzung des SPS-Funktionsbausteins

Die Einstellung *des Code Interface Packaging* haben sowohl für die Nutzung der TcCOM als auch für die Nutzung der SPS-Funktionsbausteine dieselbe Bedeutung.

Beispiel zur Parametrierung öffnen

Öffnen Sie in MATLAB® das Beispiel Multi Instance: `TwinCAT.ModuleGenerator.Samples.Start('Multi_Instance')`

4.8.1.3 Arbeiten mit dem Blockdiagramm in TwinCAT

4.8.1.3.1 Simulink®-TcCOM

Wenn mit dem TwinCAT Target for Simulink® ein TwinCAT-Objekt erzeugt wurde und der Blockdiagram Export dabei ausgeführt wurde, kann das Block Diagram des Simulink®-Modells als Control in der TwinCAT XAE dargestellt werden.

4.8.1.3.1.1 Bedienung des Blockdiagramms

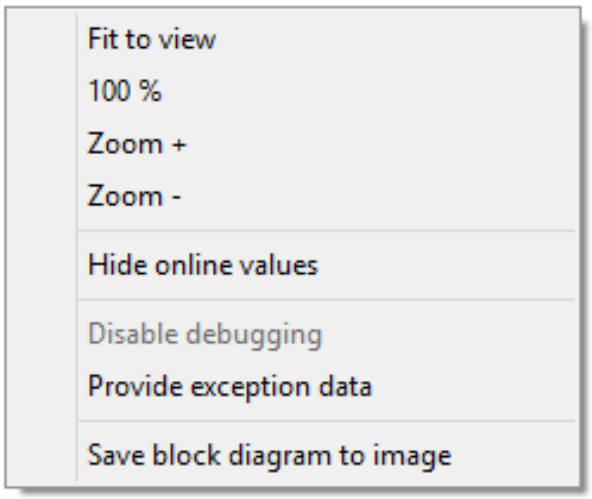
Bei der Generierung eines TcCOM-Moduls aus MATLAB® oder Simulink® kann der Export des Blockdiagramms konfiguriert werden. Wenn der Export aktiviert wurde, findet man das Blockdiagramm in der TwinCAT-Entwicklungsumgebung unter dem Karteireiter „Block Diagram“ der Modul-Instanz.

Unter Verwendung von Shortcuts, Drag&Drop sowie einem Kontextmenü, kann man durch die Hierarchie des TcCOM-Moduls navigieren, Parameterwerte ansehen, Signalwerte darstellen und optional zusätzliche Debug-Informationen erhalten.

Shortcut-Funktionen:

Shortcut	Funktion
Space	Zoom auf die aktuelle Größe des Blockdiagramm-Reiters
Backspace	Wechseln auf die nächst höhere Hierarchiestufe
ESC	Wechseln auf die nächst höhere Hierarchiestufe
STRG + “+“	Herein zoomen
STRG + “-“	Heraus zoomen
F5	Attach Debugger (*System- > Real-Time -> C++ Debugger -> Enable C++ Debugger* muss aktiviert sein)

Kontextmenü-Funktionen:

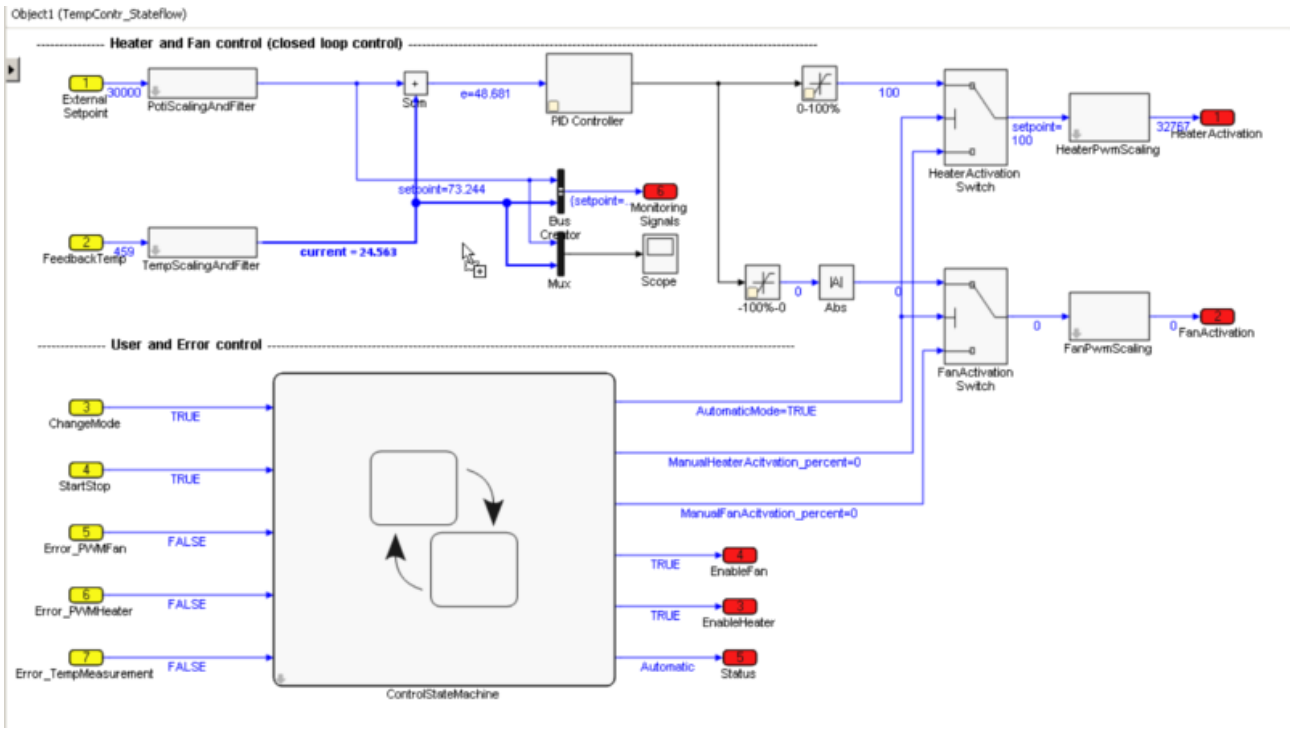


4.8.1.3.1.2 Anzeigen von Signalverläufen

Oft ist es hilfreich, sich zur Verifikation und Fehlersuche Signalverläufe anzeigen zu lassen. Das Blockdiagramm bietet hierzu folgende Möglichkeiten:

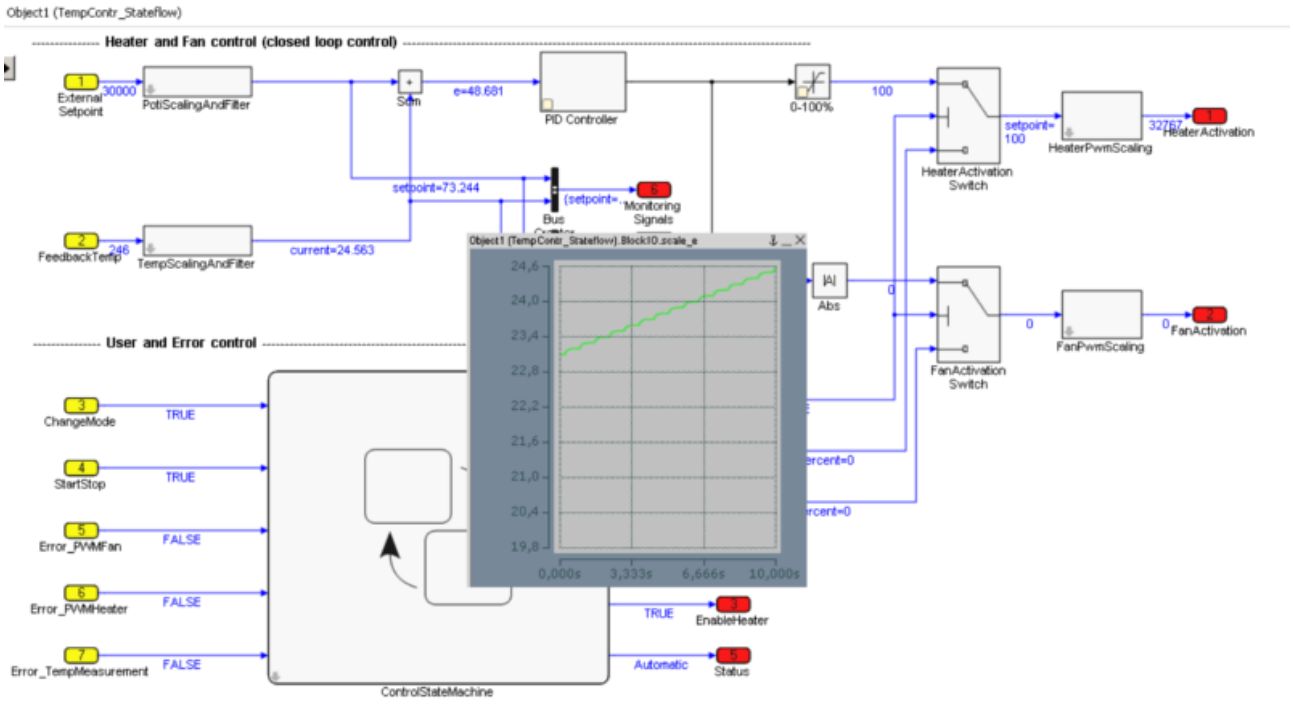
Anzeigen von Signalverläufen im Blockdiagramm

Das Blockdiagramm bietet die Möglichkeit, Signalverläufe in einem Fenster anzuzeigen. Hierzu wird ein Signal oder Block durch Drag-and-Drop auf einen freien Bereich des Blockdiagramms gezogen.



Erstellen eines Scopes im Blockdiagramm

Nach dem Drop öffnet sich ein Scope-Fenster im Blockdiagramm.



Anzeige des Scopes im Blockdiagramm

Die Titelleiste vom Scope-Fenster bietet folgende Optionen:

✖	Fenster schließen
⚓	Fenster über alle Blockdiagrammhierarchien im Vordergrund halten
—	Fenster auf die Titelleiste minimieren

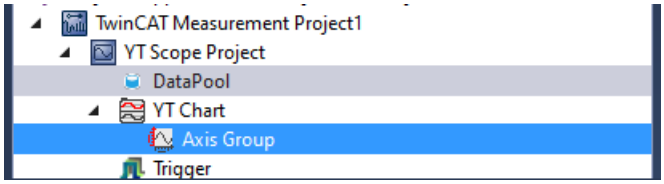
Für die Darstellung des Scopes im Blockdiagramm-Control [▶ 247] wird eine Lizenz des Scope View Professional (TE1300) benötigt. Im TwinCAT XAE ist keine Scope View Professional Lizenz notwendig.

Beim Erstellen eines Scope-Fensters im Blockdiagramm für einen Simulink®-Bus werden direkt alle Signale des Bus im Scope-Fenster dargestellt.

Es bietet sich an, das Scope-Fenster im Blockdiagramm für einen schnellen Überblick zu verwendet. Für genauere Analysen empfiehlt es sich, die Signale in einem TwinCAT Measurement-Projekt zu analysieren.

Anzeigen von Signalverläufen im TwinCAT 3 Scope

Erfolgt der Drop nicht auf das Blockdiagramm Control sondern auf eine Axis Group in einem TwinCAT Measurement Projekt, wird das Signal dort hinzugefügt.



Hinzufügen eines Signals in ein TwinCAT 3 Scope

4.8.1.3.1.3 Modul-Parametrierung im Blockdiagramm

Zur Parametrierung einer TcCOM-Instanz kann das **Parameter-Fenster** direkt im Blockdiagramm verwendet werden. Außerdem kann die Eigenschaftstabelle genutzt werden, die am rechten Rand des Blockdiagramms ein- und ausgeklappt werden kann. Grundsätzlich wird zwischen unterschiedlichen Parameter-Werten unterschieden:

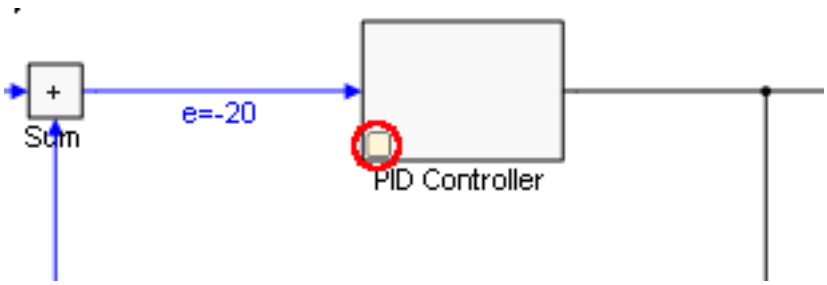
„Default“, „Startup“, „Online“ und „Prepared“

Im Drop-down-Menü der Eigenschaftstabelle des Blockdiagramms findet man folgende Wertetypen:

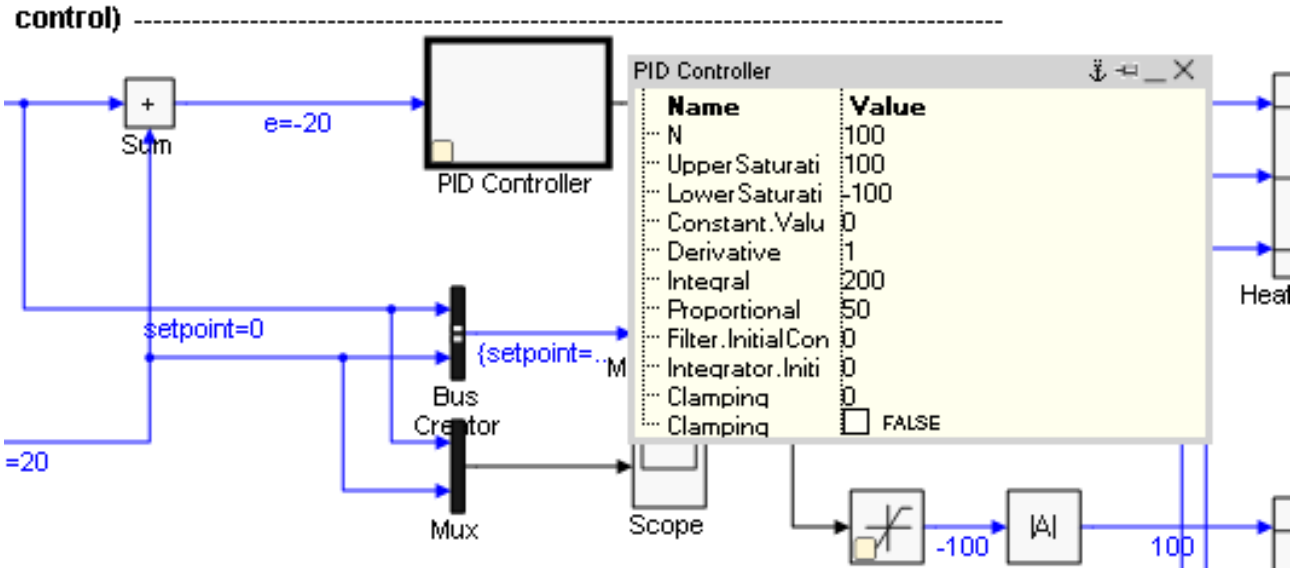
- **Default-Werte** sind die Parameterwerte beim Generieren des Codes. Sie sind unveränderlich in der Modulbeschreibungsdatei gespeichert und ermöglichen es, nach Parameteränderungen die "manufacturing settings" wiederherzustellen.
- **Startup-Werte** werden in der TwinCAT-Projektdatei gespeichert und in die Modulinstanz heruntergeladen, sobald TwinCAT die Modulinstanz startet.
  In Simulink®-Modulen können auch Startwerte für das Eingangs-Prozessabbild festgelegt werden. Dadurch kann das Modul mit Eingangswerten ungleich Null gestartet werden, ohne dass die Eingänge mit anderen Prozessabbildern verknüpft werden müssen. Interne Signale und Ausgangssignale haben keine Startwerte, da sie in jedem Fall im ersten Zyklus überschrieben würden.
- **Onlinewerte** sind nur verfügbar, wenn das Modul auf dem Zielsystem gestartet wurde. Sie zeigen den aktuellen Wert des Parameters im laufenden Modul. Dieser kann auch während der Laufzeit geändert werden. Das entsprechende Eingabefeld muss dazu allerdings erst über das Kontextmenü freigeschaltet werden, um versehentliche Eingaben zu vermeiden.
- **Prepared-Werte** können immer dann festgelegt werden, wenn auch Onlinewerte verfügbar sind. Mit ihrer Hilfe können verschiedene Werte gespeichert werden, um sie konsistent in das Modul zu schreiben. Wenn vorbereitete Werte festgelegt wurden, sind diese in einer Tabelle unterhalb des Blockdiagramms zu sehen. Mit den Schaltflächen rechts neben der Liste können die vorbereiteten Werte als Onlinewert heruntergeladen und/oder als Startwert gespeichert oder auch gelöscht werden.

Parametrieren im Blockdiagramm

Parametrierbare Blöcke werden im Blockdiagramm mit einem gelben Kasten markiert.



Durch Doppelklick auf den Block oder durch einen einzelnen Klick auf den gelben Kasten wird ein Fenster mit den veränderbaren Parametern angezeigt.



Wird ein Wert geändert, kann dieser mit folgenden Tastenbefehlen übernommen werden:

STRG + Enter	Onlinewert direkt setzen
SHIFT + Enter	Startup-Wert setzen
Enter	Prepared-Wert setzen

Die Symbole in der Titelleiste haben folgende Funktionen:

	Fenster schließen
	Fenster über alle Blockdiagrammhierarchieebenen im Vordergrund halten
	Fenster auf der aktuellen Blockdiagrammhierarchieebene offen halten
	Fenster auf Titelleiste minimieren

4.8.1.3.1.4 Debuggen

Um Fehler innerhalb eines mit MATLAB®/Simulink® erstellten TcCOM-Moduls zu finden bzw. das Verhalten des Moduls in der Gesamtarchitektur des TwinCAT-Projekts zu analysieren, stehen unterschiedliche Wege zur Verfügung.

Debuggen im Blockdiagramm

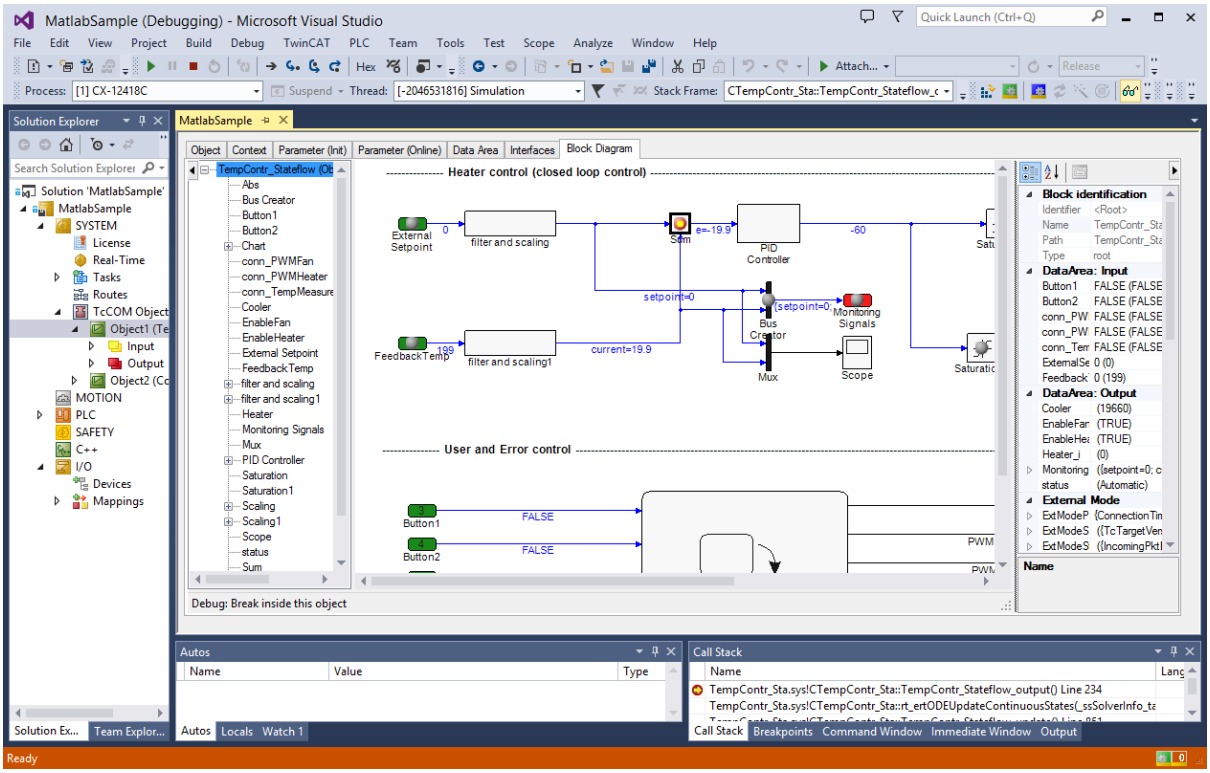
Wurde bei der Generierung des TcCOM-Moduls das Blockdiagramm exportiert, kann dieses in der TwinCAT-Entwicklungsumgebung angezeigt und unter anderem zum Debuggen innerhalb der entsprechenden Modulinstanz verwendet werden. Dazu nutzt das Blockdiagramm den Microsoft Visual Studio Debugger, der über den TwinCAT Debugger-Port mit der TwinCAT-Laufzeit verbunden werden kann. Das Verbinden (Attachen) des Debuggers erfolgt wie im C++-Bereich unter Debuggen beschrieben.

Voraussetzungen für das Debuggen innerhalb des Blockdiagramms sind:

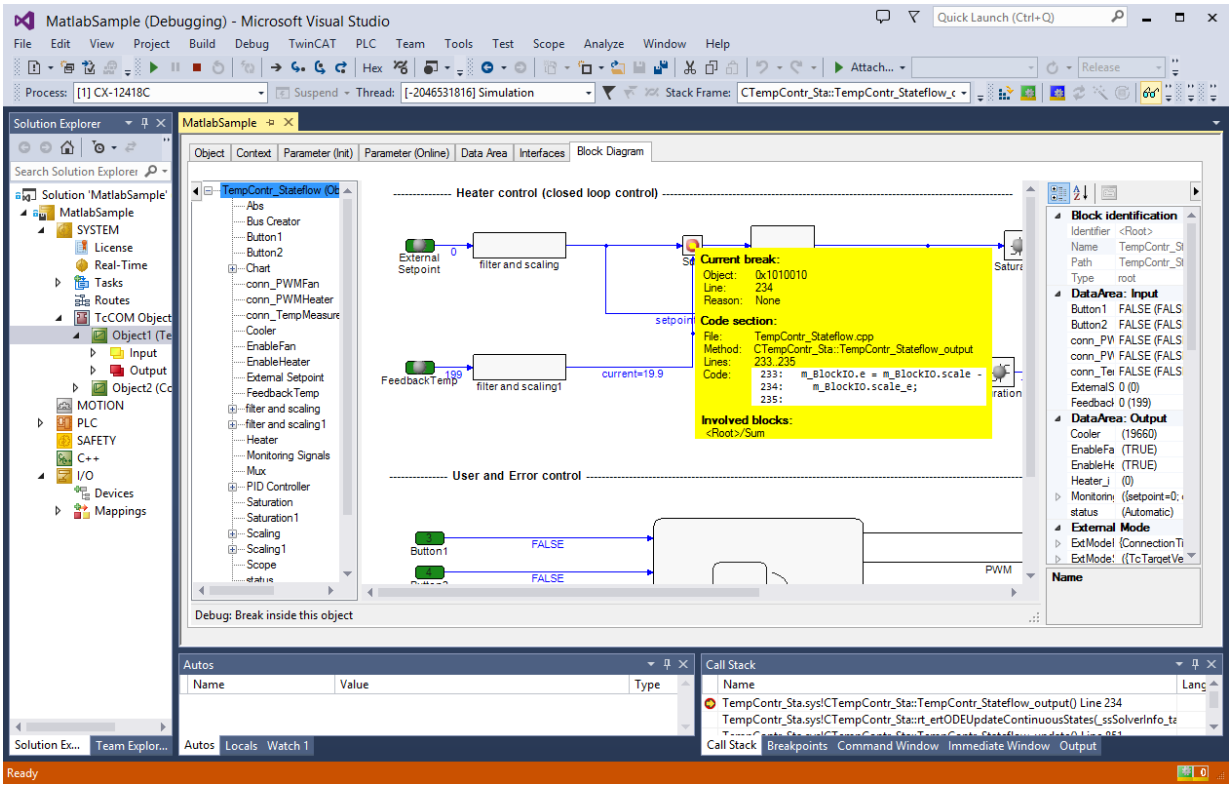
- Der C/C++-Quellcode des TcCOM-Moduls muss auf dem Engineering-System vorhanden sein und vom Visual Studio-Debugger gefunden werden können. Idealerweise sollte das Debuggen daher auf dem System stattfinden, auf dem auch die Codegenerierung ausgeführt wurde. Wurde das Modul auf einem anderen System erstellt, kann der zugehörige C/C++-Quellcode i.d.R. durch Einbindung des Visual Studio Projektes in den C++ Bereich von TwinCAT bekannt gemacht werden. Die Datei <ModelName>.vcxproj liegt im build-Verzeichnis, siehe Welche Dateien werden automatisch bei der Codegenerierung und dem Publish erstellt? [▶ 62]
- Das Modul muss mit der Konfiguration **Debug** erstellt worden sein. Beim Publish direkt im Anschluss an die Codegenerierung, muss im Bereich Modulgenerierung (Tc Build) [▶ 23] unter **publish configuration** die Einstellung **Debug** ausgewählt werden. Beim publish des Moduls aus dem C++ Bereich in TwinCAT muss der Debugger im C++ Knoten der Solution freigegeben sein, siehe Dokumentation C/C++ - Debuggen.
- Bei der Codegenerierung müssen in den Coder-Einstellungen unter **Tc Advanced** die Optionen **Export block diagram** und **Export block diagram debug information** aktiviert sein.
- Im TwinCAT-Projekt muss der Debugger-Port aktiviert sein, wie unter TwinCAT 3 C++ Enable C++ debugger beschrieben.

Setzen von Breakpoints im Blockdiagramm

1. Nach dem Verbinden (Attachen) des Debuggers mit der TwinCAT-Laufzeit, werden die möglichen Breakpoints im Blockdiagramm den Blöcken zugeordnet und als Punkte dargestellt. Durch Anklicken des gewünschten Breakpoints kann dieser aktiviert werden, um die Ausführung der Modulinstanz bei der nächsten Abarbeitung des zugehörigen Blocks zu stoppen. Die Farbe des Punktes gibt Auskunft über den aktuellen Zustand des Breakpoints:
   - Grau: Breakpoint inaktiv
   - Rot: Breakpoint aktiv. Bei der nächsten Abarbeitung dieses Blocks wird der Programmablauf angehalten
   - Gelber Punkt in der Mitte: Breakpoint Hit. Die Programmabarbeitung ist im Augenblick an dieser Stelle angehalten
   - Blauer Punkt in der Mitte: Breakpoint Hit (wie gelb), allerdings in einer anderen Instanz des Moduls.



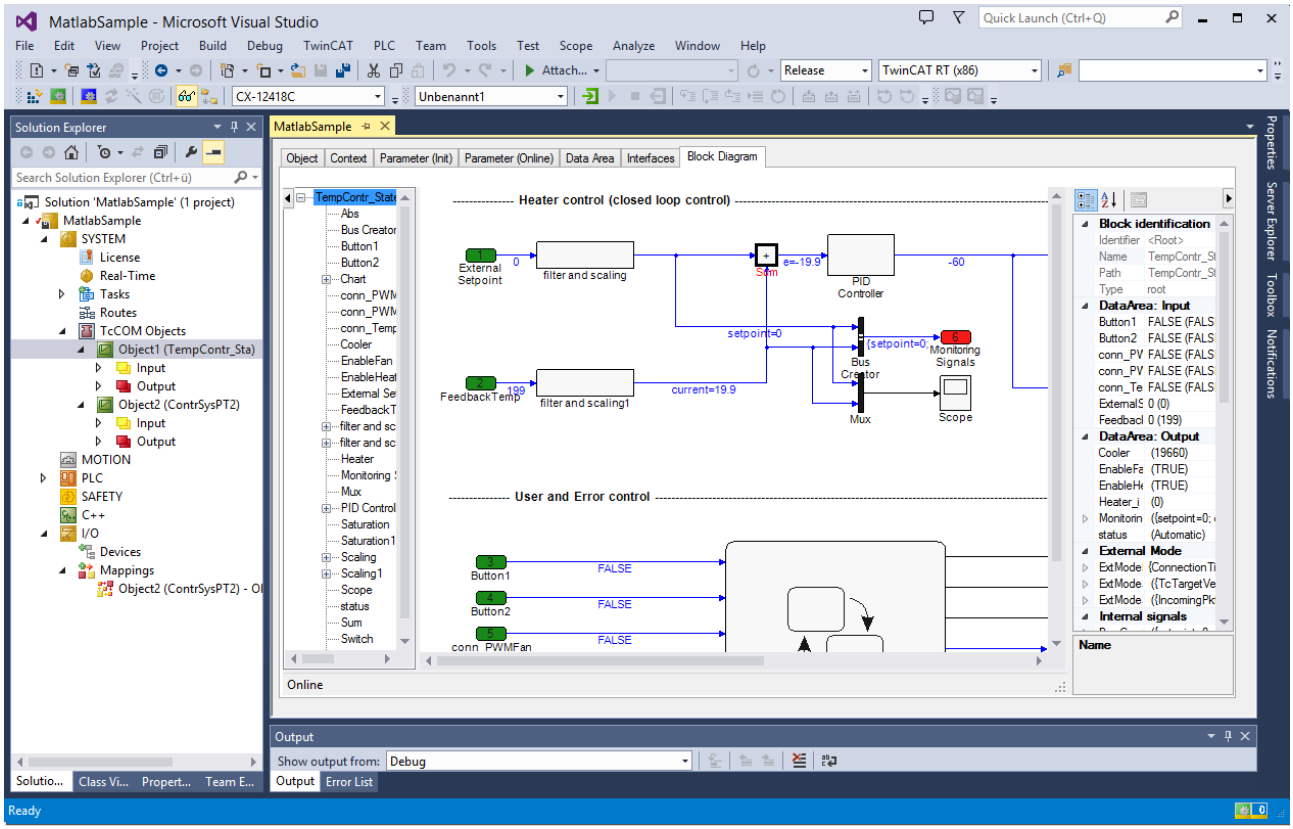
2. Im Tool-Tip des Breakpoints findet man zusätzliche Informationen, wie z. B. den zugehörigen C++-Code-Abschnitt:



Breakpoints werden nicht immer einem einzelnen Block zugeordnet. Im zugrundeliegenden C++-Code sind häufig Funktionalitäten mehrerer Blöcke in einem Codeabschnitt oder sogar einer Zeile zusammengefasst. Weil sich daher oft mehrere Blöcke den gleichen Breakpoint teilen, ändert sich bei der Aktivierung eines Breakpoints im Blockdiagramm häufig auch die Darstellung der Punkte an anderen Blöcken.

Auswertung von Exceptions

Treten während der Abarbeitung eines TcCOM-Modules Exceptions, wie z.B. eine Division durch Null, auf, so kann die Stelle an der diese Exception versursacht wurde im Blockdiagramm dargestellt werden. Dazu muss das TcCOM-Modul die oben genannten Voraussetzungen erfüllen und der C++-Debugger muss im TwinCAT-Projekt aktiviert sein (TwinCAT 3 C++ Enable C++ debugger). Nachdem der Debugger verbunden (attached) wurde, was vor aber auch noch nach dem Auftreten der Exception erfolgen kann, wird der verursachende Block im Blockdiagramm hervorgehoben, sofern die verursachende Codezeile einem Block zugeordnet werden kann. Der Name des Blockes wird rot dargestellt und der Block selbst wird fett markiert.



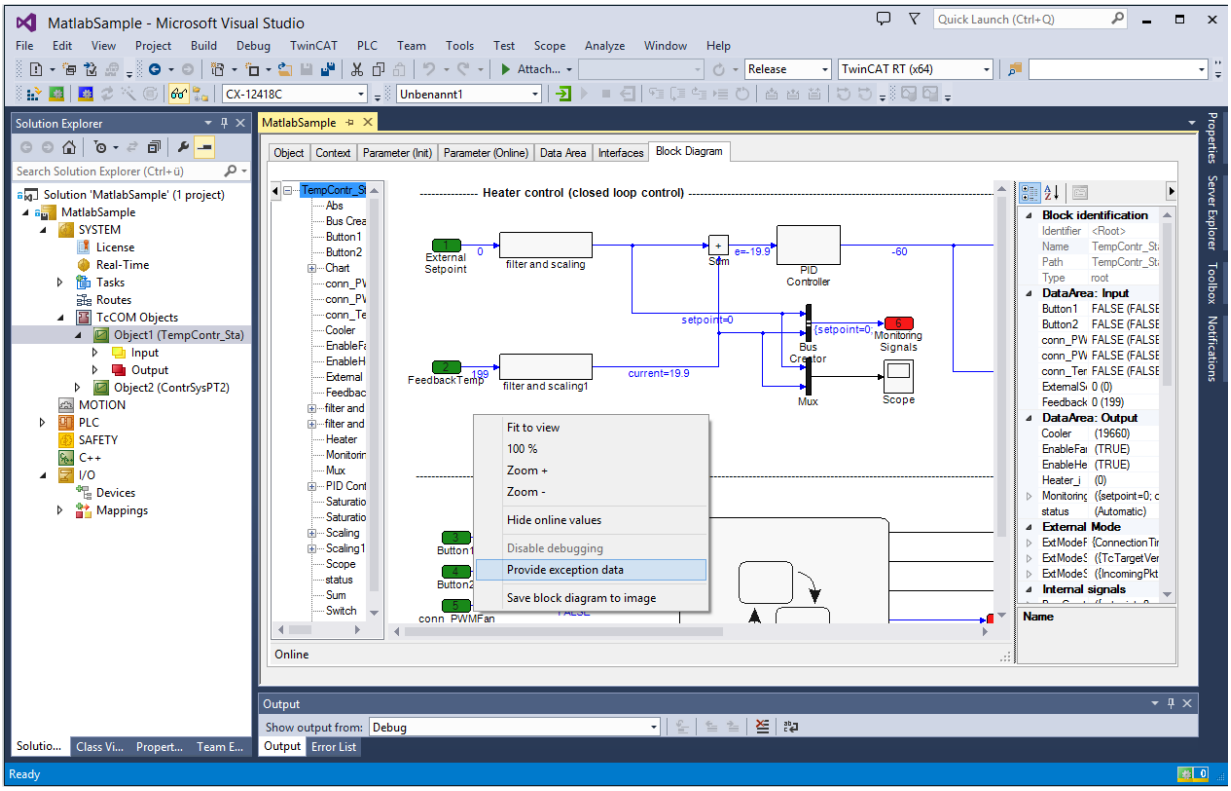
Manuelle Auswertung von Exceptions ohne Quellcode

Auch wenn auf dem Engineering-System nicht der Source Code des Modules verfügbar ist oder der C++-Debugger nicht aktiviert wurde, kann man nach Auftreten einer Exception die Fehlerstelle im Blockschaltbild hervorheben.

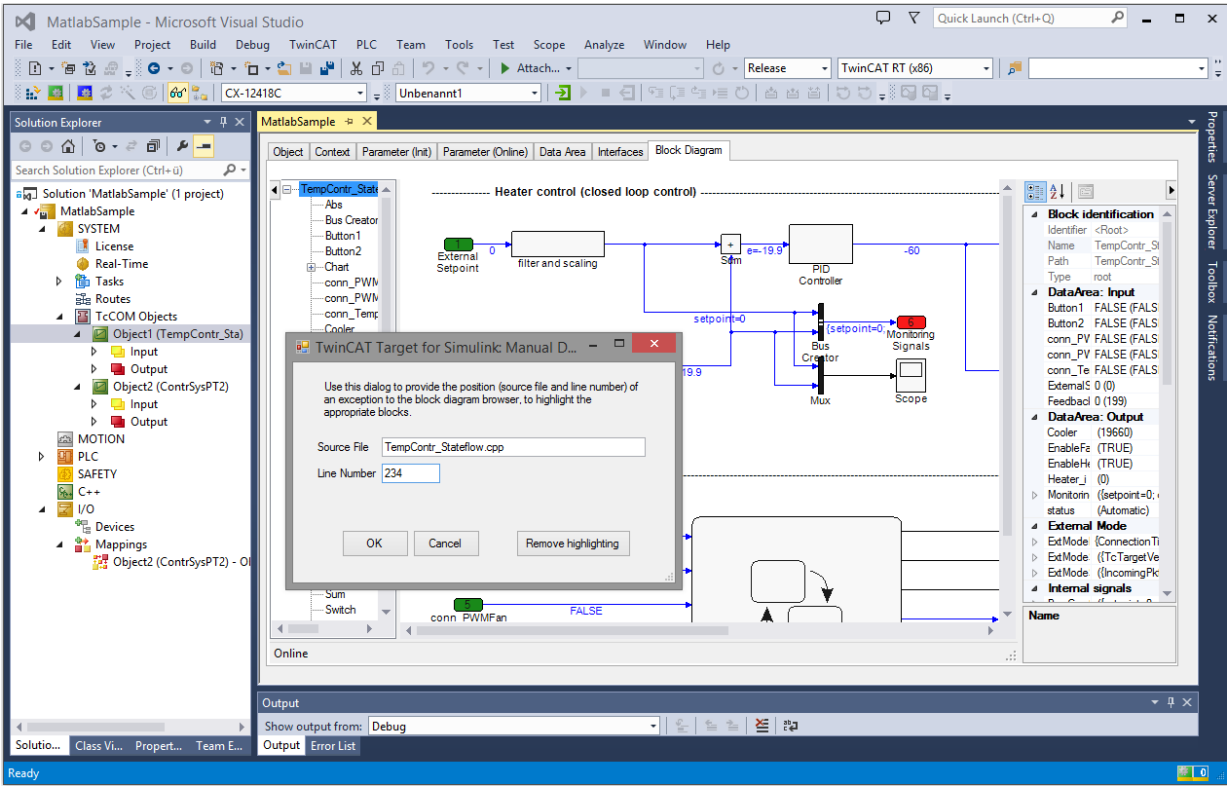
Typischerweise wird beim Auftreten eines Fehlers immer eine Fehlermeldung generiert, in der die Quellcode-Datei sowie die Zeile im Quellcode angegeben ist. Über diese Information lässt sich eine Exception in vielen Fällen einem Block des Blockschaltbildes zuordnen. Dazu kann man wie folgt vorgehen:

✓ Voraussetzung für das Hervorheben der Fehlerstelle innerhalb des Blockdiagramms ist, dass die Debuginformationen erzeugt wurden (Option **Export block diagram debug information** in den Coder-Einstellungen unter **Tc Advanced**).

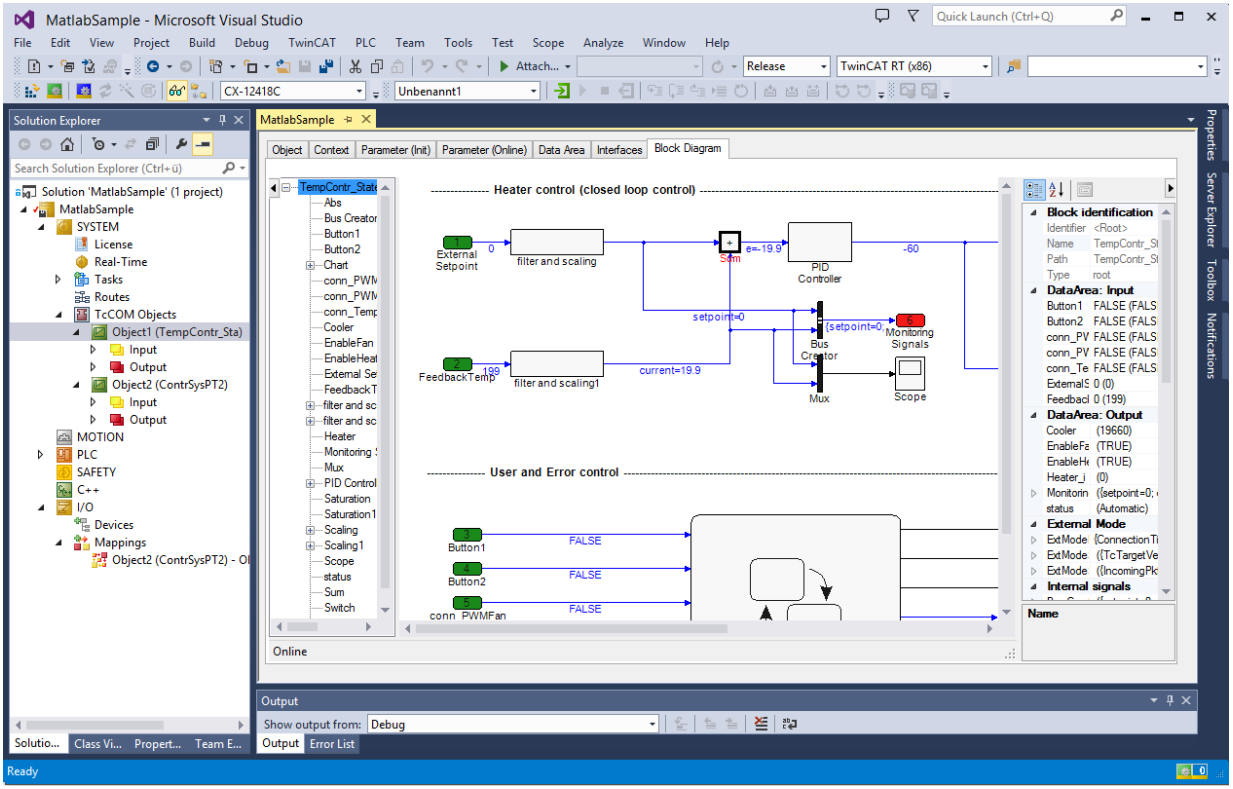
3. Aus dem Kontext-Menü des Blockschaltbildes ist der Eintrag **Provide exception data** zu wählen:



4. In dem sich öffnenden Dialog sind die in der Fehlermeldung bereitgestellte Quellcode-Datei und Zeilennummer einzutragen:



5. Der Name des Blockes, welchem die Zeilennummer zugeordnet ist, wird rot dargestellt und der Block selbst wird fett markiert:



4.8.1.4 Online Change von TcCOM zur Laufzeit

Mit dem „Online Change“ können Sie TcCOM Objekte zur Laufzeit, d.h. ohne TwinCAT Stopp, auf einem Laufzeit-PC austauschen.

Beschreibung des Online Change für TcCOM siehe: Erstellung versionierter Treiber [▶ 142].

Beispiel in MATLAB® öffnen

Öffnen Sie ein Beispiel in MATLAB® mit:

```
TwinCAT.ModuleGenerator.Samples.Start('OnlineChange_TemperatureController')
```

4.8.1.5 To File Block und MAT-file logging

Sie können Ihre Simulink®-Modelle so konfigurieren, dass sie in Form eines TcCOM-Objekts in der TwinCAT-Laufzeit MAT-files auf dem Dateisystem des Laufzeit-PCs erzeugen.

Schreibrechte auf dem Laufzeit-PC beachten

Beachten Sie die Schreibrechte auf dem Pfad, auf dem Sie schreiben möchten.

MAT-file logging

Konfiguration seitens Simulink®:

- Aktivieren Sie MAT-file logging unter **Code Generation > Interface > MAT-file logging**, vgl. MathWorks®-Dokumentation.
- Aktivieren Sie **Code Generation > TC General > Load DataExchangeModules.**

Wenn mit diesen Einstellungen ein TcCOM-Objekt erzeugt und in einer TwinCAT-Konfiguration auf einem Laufzeit-System aktiviert wird, werden entsprechend der von MathWorks® vorgegebenen Eigenschaften Modellsignale in ein MAT-file gespeichert.

Das MAT-file wird auf dem Dateisystem des Laufzeit-PCs im TwinCAT-Boot-Verzeichnis erzeugt.

To File Block

Konfiguration in Simulink®:

- Aktivieren Sie MAT-file logging unter **Code Generation > Interface > MAT-file logging**.
- Aktivieren Sie **Code Generation > TC General > Load DataExchangeModules**.
- Geben Sie im **ToFile Block** den Fullpath an, z. B. *C:\Logs\MyLog.mat.* Wenn Sie nur den Dateinamen angeben wird das MAT-file im TwinCAT-Boot-Verzeichnis erzeugt.
- Wählen Sie in den **Block Parameters** des **To File Blocks** *Save format: Array*.

Wenn mit diesen Einstellungen ein TcCOM-Objekt erzeugt und in einer TwinCAT Konfiguration auf einem Laufzeit-System aktiviert wird, wird an konfigurierter Stelle ein MAT-file erzeugt.

- Das MAT-file wird zur Laufzeit mit neuen Daten gefüllt und wächst in seiner benötigten Speichergröße entsprechend mit fortlaufender Zeit an.
- Die Terminate-Methode zum Abschließen des MAT-files wird in der Transition Preop-Init durchgeführt. Fahren Sie dazu entweder das TcCOM-Objekt in den Init State oder versetzen Sie die TwinCAT-Laufzeit in den Config mode.

HINWEIS

Ausreichender Speicherplatz

Beachten Sie, dass Sie genügend Speicherplatz auf dem Zielsystem vorhalten müssen, um unvorhergesehenes Verhalten des Laufzeit-PCs zu vermeiden.

TwinCAT File Writer

Sie können mit dem TwinCAT File Writer das Daten-Logging zur Laufzeit präzise steuern. Der TwinCAT File Writer kann Dateipakete definierter Größe abschließen und eine vorgegebene Menge an Dateien auf dem Laufzeitsystem erzeugen. Dadurch besteht keine Gefahr, an das Limit des Zielsystem-Speichers zu gelangen.

- Maximale Größe der .mat files einstellbar
- Maximale Anzahl der .mat files einstellbar
- Schreiben optional pausieren über einen TcCOM Modul Parameter
- Unterstützt nicht alle Datentypen

Dokumentation des Blocks finden Sie hier [▶ 120].

4.8.1.6 Aufruf des TcCOM aus der SPS

Bitte zu Abschnitt Anwenden des TcCOM-Wrapper-FB [▶ 214] wechseln.

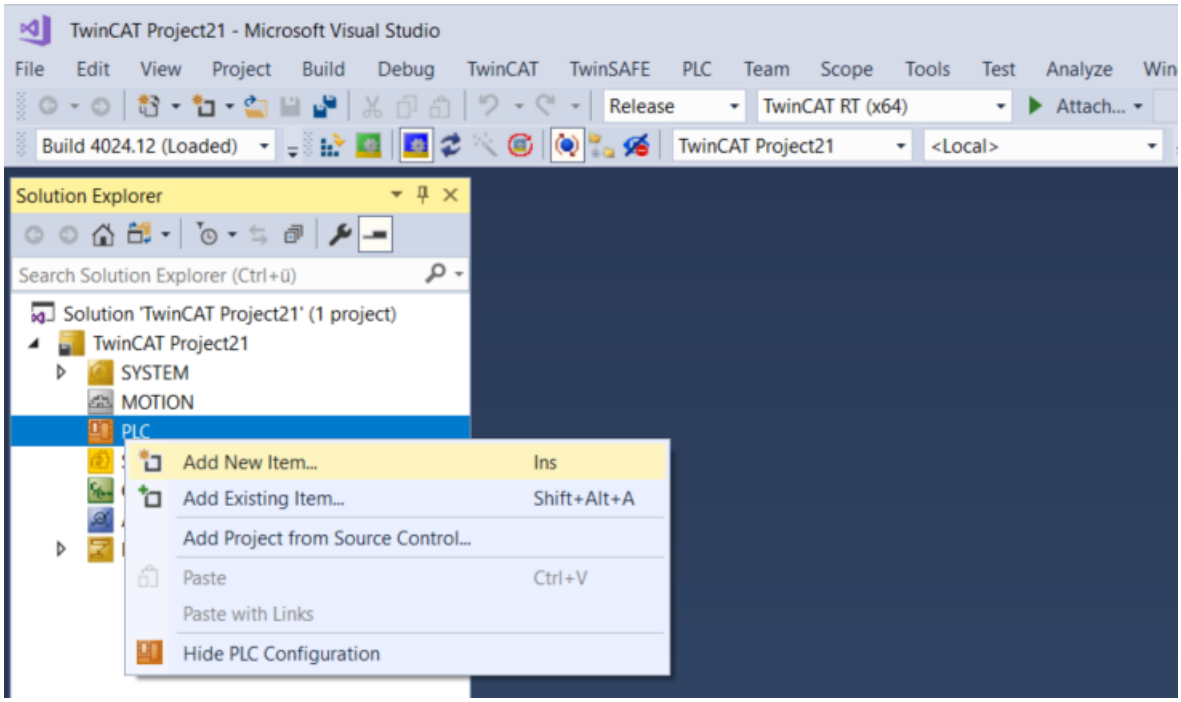
4.8.2 Arbeiten mit der SPS-Bibliothek

Neben dem TcCOM-Objekt kann ebenfalls eine SPS-Bibliothek erstellt werden. Diese SPS-Bibliothek beinhaltet **zwei** unterschiedliche **Typen** von Funktionsbausteinen:

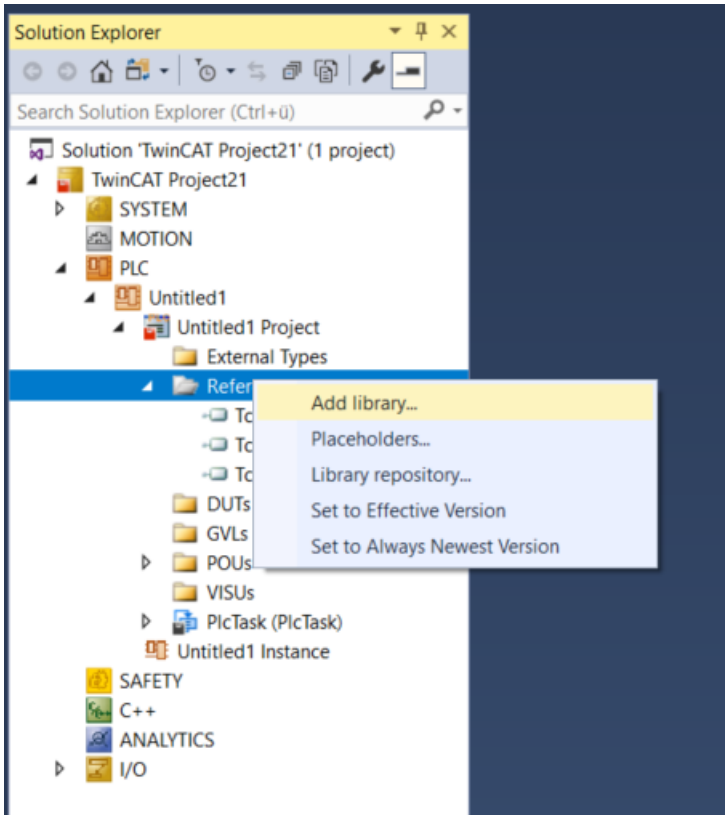
- Der Funktionsbaustein *FB_<modelname>* im Ordner *POU* beinhaltet die volle Funktionalität des Simulink®-Modells, d. h. der Source Code ist im FB direkt verankert. Dieser FB wird im Folgenden kurz **PLC-FB** genannt.
- Im Unterordner *POU/TcCOM Wrapper* sind FBs gelistet, welche sich als Wrapper für ein TcCOM-Objekt verstehen, d. h. die Funktionalität ist nicht direkt im FB, sondern ausgelagert in einer Instanz eines TcCOM. Die Wrapper werden im Folgen kurz **TcCOM-Wrapper-FB** genannt.

Kurzüberblick

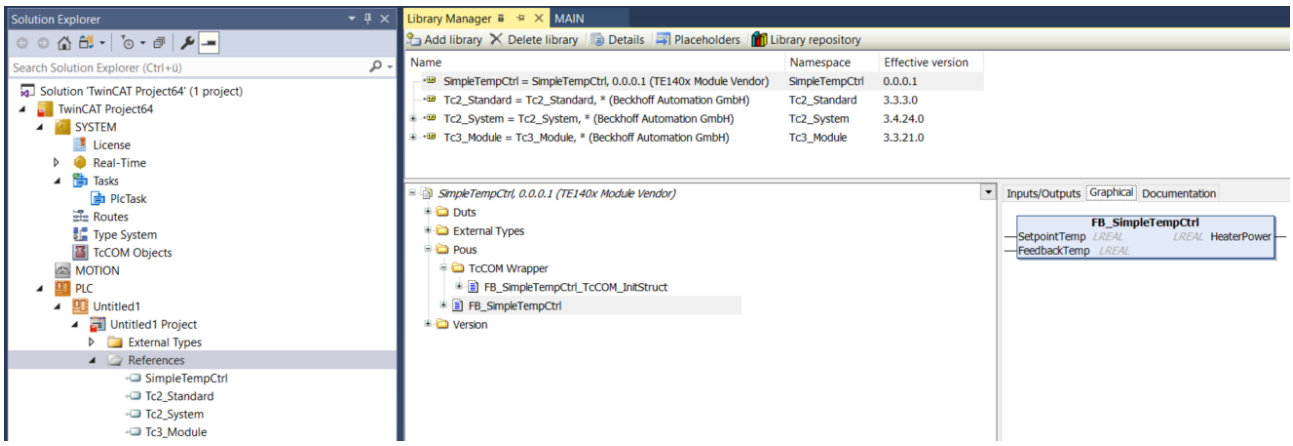
- SPS-Projekt anlegen:



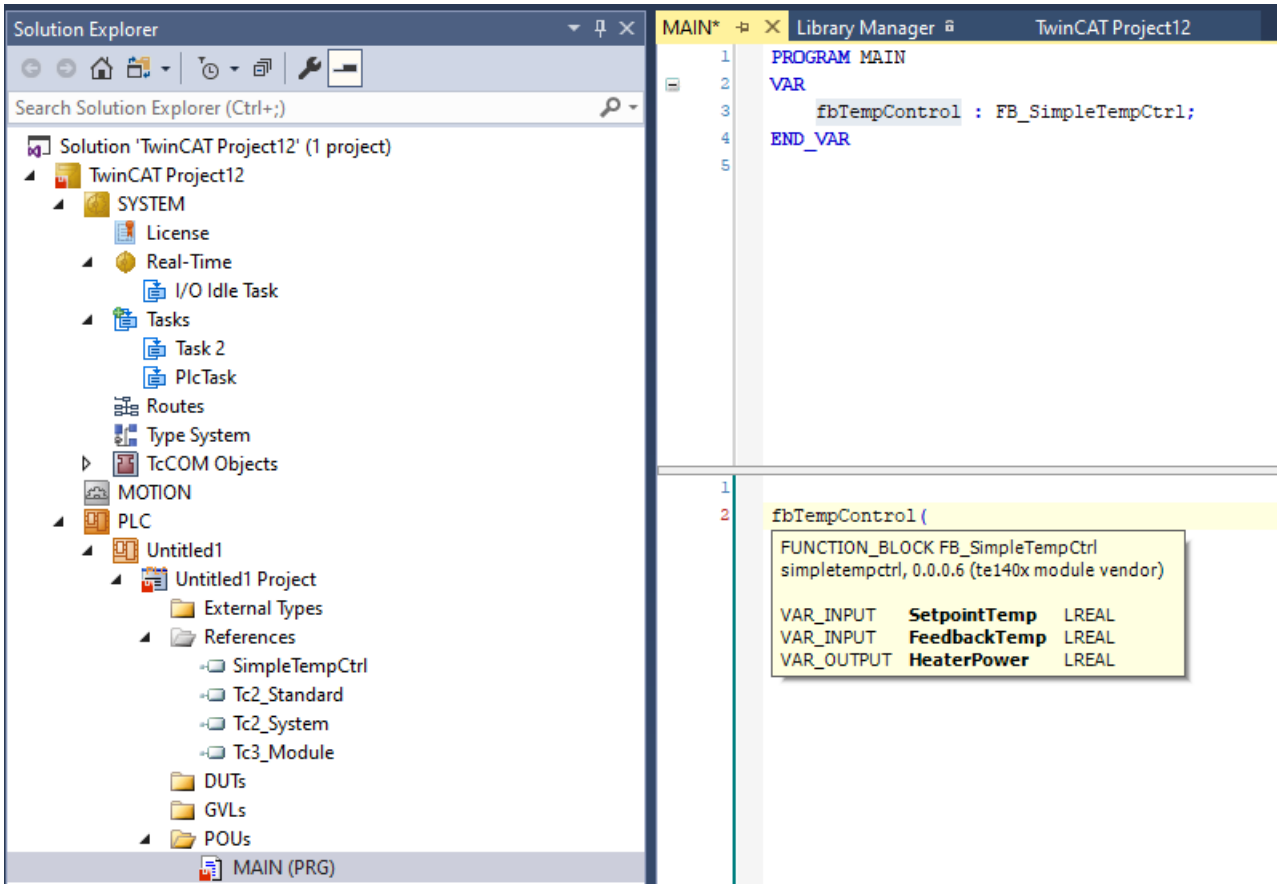
- SPS-Bibliothek laden:



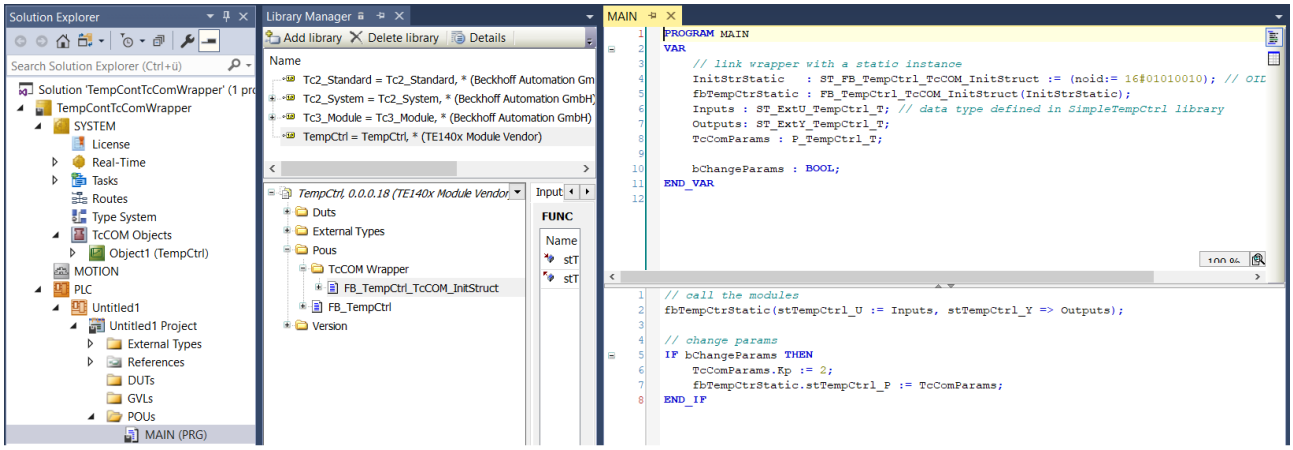
- Inhalt der Bibliothek betrachten:



- Funktionsbaustein *FB_<modelname>* (PLC-FB [▶ 219]) instanziieren und im SPS-Code verwenden:



- Alternativ: TcCOM-Wrapper-FB [▶ 214] nutzen und im SPS-Code verwenden:



4.8.2.1 SPS-Bibliothek erzeugen und installieren

Inhalt der SPS-Bibliothek konfigurieren

Wie unter Arbeiten mit der SPS-Bibliothek [▶ 208] beschrieben, kann die SPS-Bibliothek unterschiedliche Funktionsbausteine enthalten.

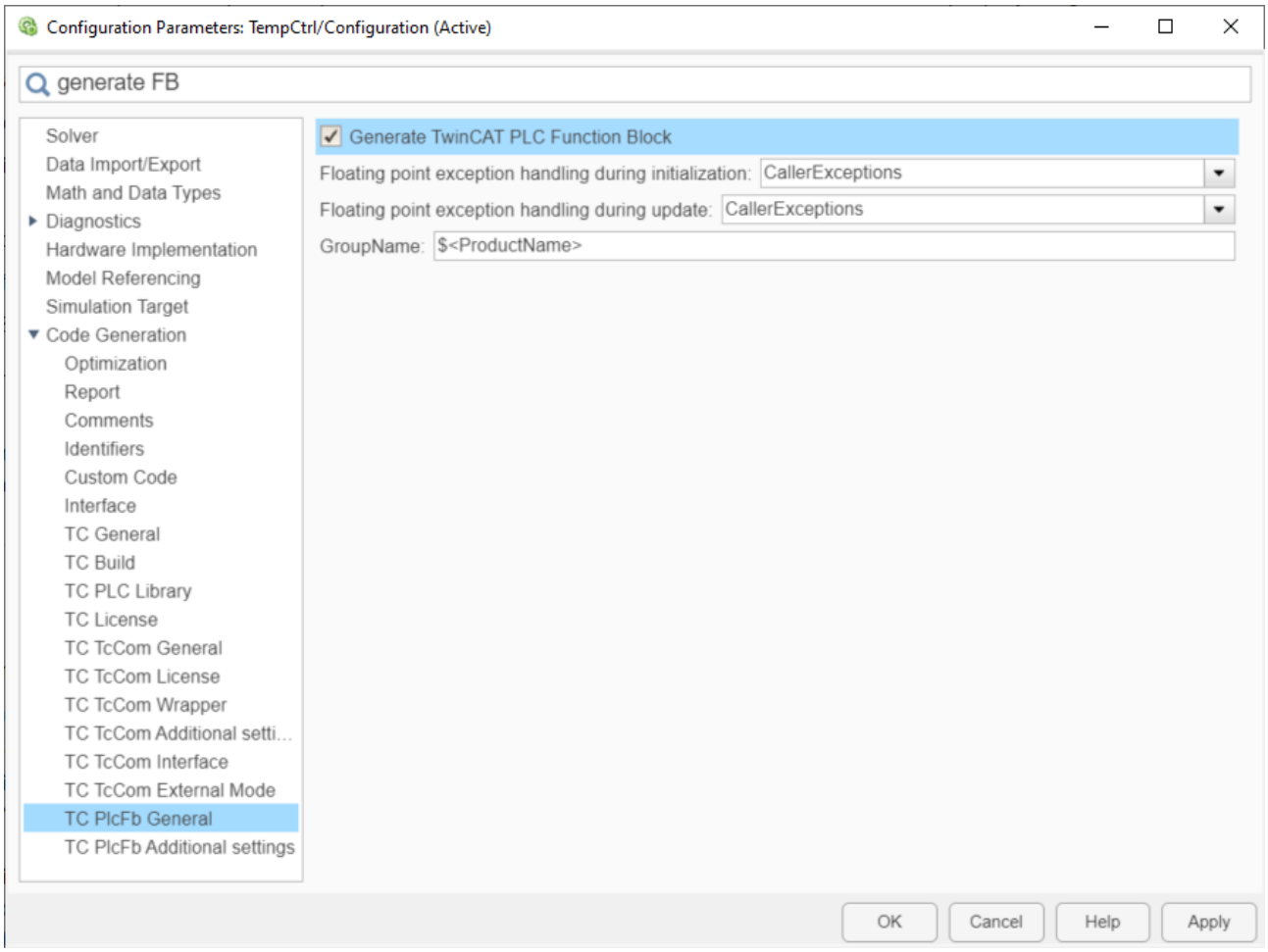
- Der Funktionsbaustein *FB_<modelname>* im Ordner *POU* beinhaltet die volle Funktionalität des Simulink®-Modells, d. h. der Source Code ist im FB direkt verankert. Dieser FB wird im Folgenden kurz **PLC-FB** genannt.
- Im Unterordner *POU/TcCOM Wrapper* sind FBs gelistet, welche sich als Wrapper für ein TcCOM-Objekt verstehen, d. h. die Funktionalität ist nicht direkt im FB, sondern ausgelagert in einer Instanz eines TcCOM. Die Wrapper werden im Folgen kurz **TcCOM-Wrapper-FB** genannt.

Im Folgenden erfahren Sie, wie Sie die beiden Funktionsbausteine konfigurieren können und wie Sie die erstellte SPS-Bibliothek installieren.

4.8.2.1.1 Erstellen und Konfigurieren des PLC-FB

Navigieren Sie zu **Configuration Parameters > Code Generation > TC PlcFb General**.

Aktivieren Sie hier die Checkbox *Generate TwinCAT PLC Function Block*. In der Standardkonfiguration ist die Checkbox aktiviert.

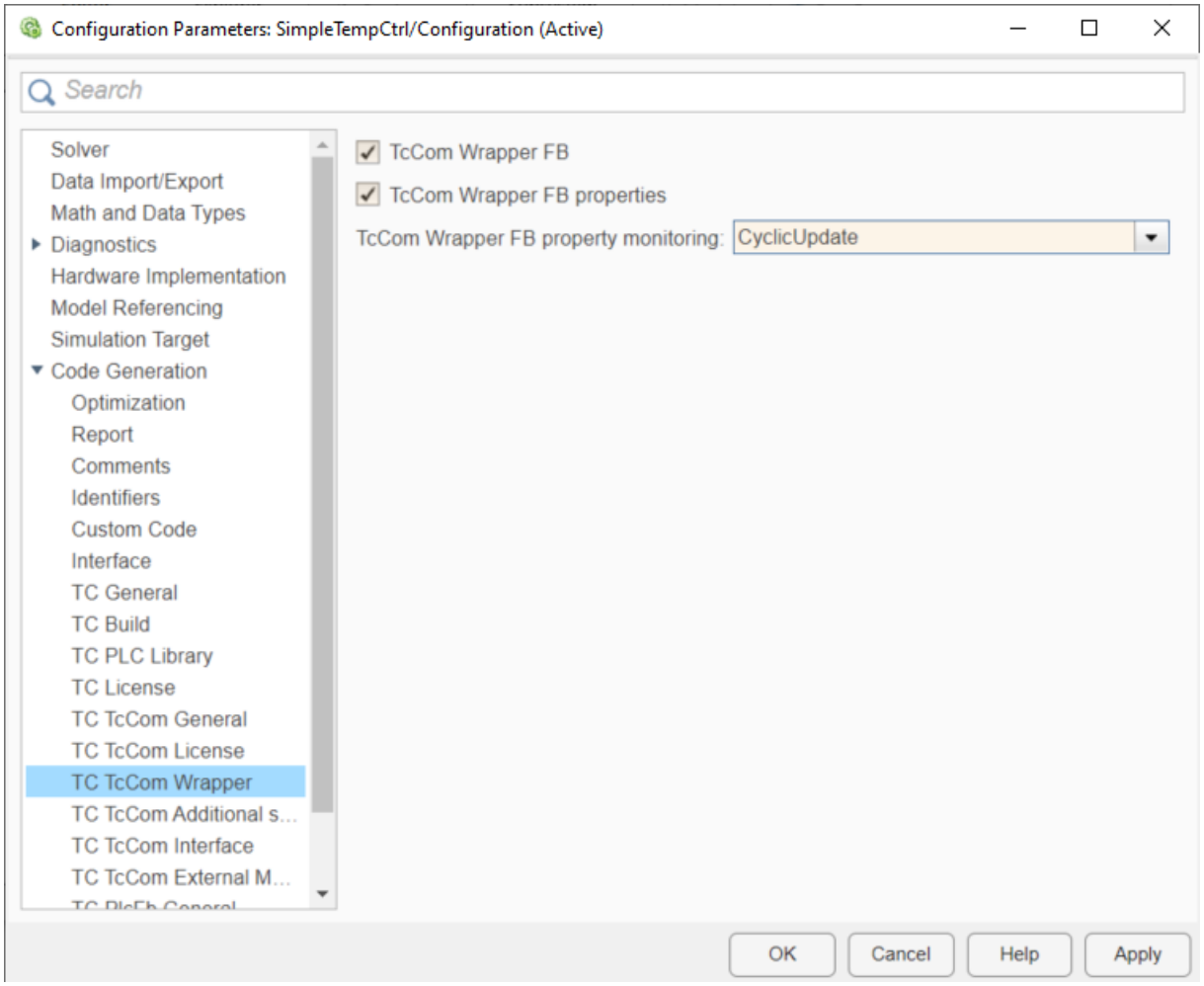


Die weiteren Einstellungen betreffen die Handhabung von Floating Point Exceptions für den PLC-FB. Diese müssen separat zu den Einstellungen für das TcCOM-Objekt getroffen werden, siehe Exception Handling [▶ 225].

4.8.2.1.2 Erstellen und Konfigurieren des TcCOM-Wrapper-FB

Navigieren Sie zu **Configuration Parameters > Code Generation > TC TcCom Wrapper**.

Aktivieren Sie die Checkbox *TcCom Wrapper FB*. In der Standardkonfiguration ist die Checkbox nicht aktiviert.



Über die *Checkbox TcCom Wrapper FB properties* können Sie konfigurieren, ob Modul-Parameter am FB als Properties erstellt werden sollen. Siehe dazu Konfiguration des Datenzugriffs auf Daten eines TcCOM-Objekts [▶ 147], insbesondere *Parameter: Initial values*.

Wie Sie den erzeugten Wrapper anwenden, erfahren Sie hier: Anwenden des TcCOM-Wrapper-FB [▶ 214].

Beispiel

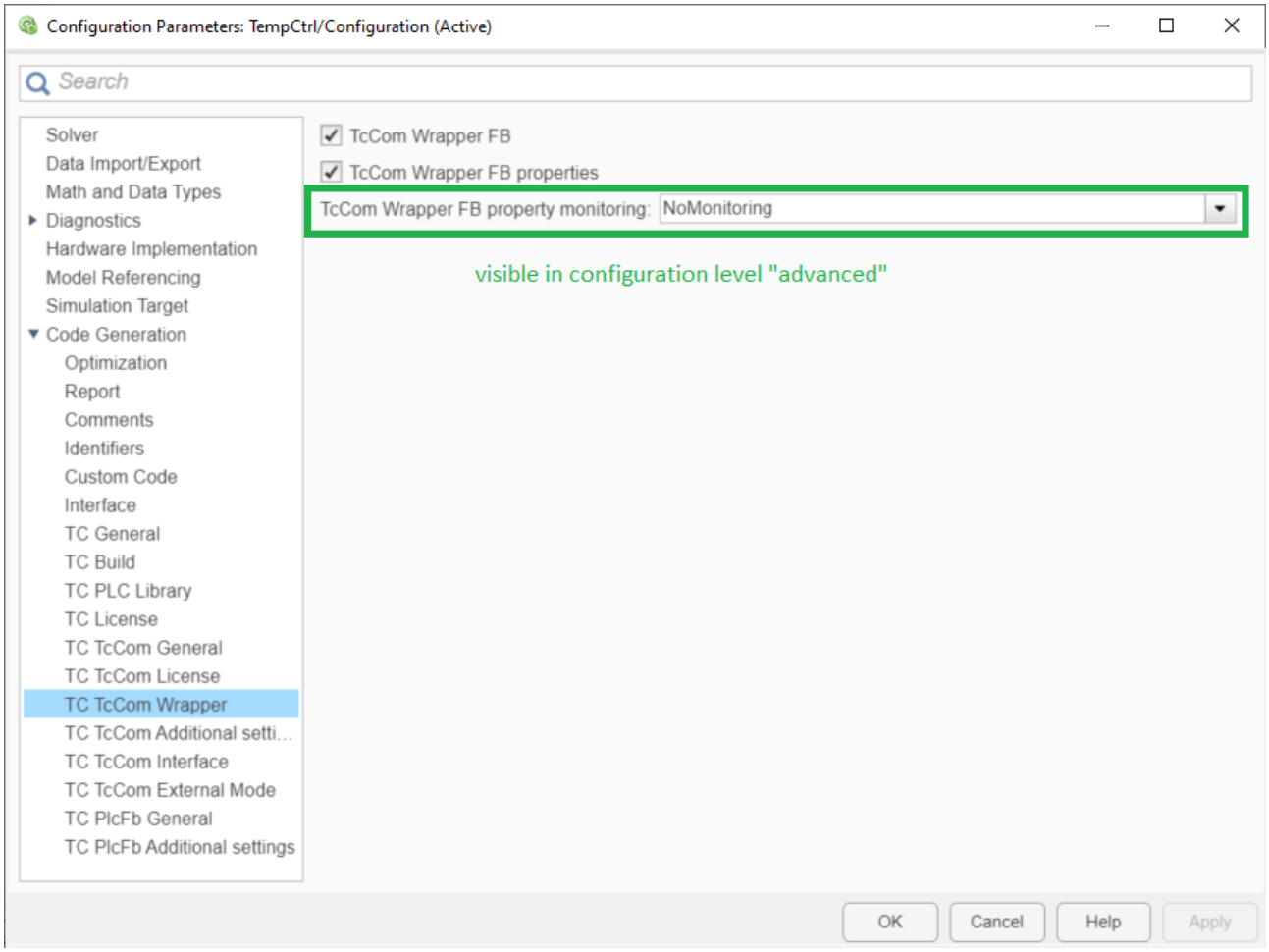
Durch Setzen des *Parameter: Initial Values* unter Tc TcCom Interfaces werden die Modell-Parameter als Modul-Parameter angelegt (standardmäßig eingeschaltet). Erzeugen Sie nun den „TcCom Wrapper FB“ mit der Option „TcCom Wrapper FB properties“. Setzen Sie das property monitoring auf „CyclicUpdate“, um die Wert-Änderung des Property im Online-View direkt sehen zu können.

Dann können Sie beispielsweise wie folgt auf die Modul-Parameter zugreifen:

```
PROGRAM MAIN
VAR

   // dynamic instance: create TcCOM from PLC
   InitStrDyn : ST_FB_TempCtrl_TcCOM_InitStruct_InitStruct := (
                TaskOid:= 16#02010030,                    // take TaskOID of PlcTask
                eModuleCaller:= ModuleCaller.Module );        // set module caller to "call by module
"
   fbTempCtrDyn : FB_TempCtrl_TcCOM_InitStruct(InitStrDyn);

   Outputs    : ExtY_TempCtrl_T;   // input
   Inputs     : ExtU_TempCtrl_T;   // output
   Parameters : P_TempCtrl_T;      // parameter
```

```
   bChange: BOOL;
END_VAR
```

```
fbTempCtrDyn(TempCtrl_U := Inputs, TempCtrl_Y => Outputs);

IF bChange THEN
   Parameters.Kp := 10;
   fbTempCtrDyn.TempCtrl_P := Parameters;
END_IF
```



Im configuration level “advanced” kann auch das Monitoring-Attribut der Properties präzisiert werden. Im Standardfall ist „No Monitoring“ eingestellt, d. h. es wird kein Attribut gesetzt.

Einstellung in Simulink®	Attribut am Property
ExecutionUpdate	{attribute 'monitoring' := 'variable'}
CyclicUpdate	{attribute 'monitoring' := 'call'}

Monitoring Attribute beeinflussen die Sichtbarkeit der Attribut-Werte im Online-View, d. h. wenn man sich in die SPS eingeloggt hat und die aktuellen Werte der Properties am FB beobachten möchte.

- No Monitoring: Die Werte sind im Online-View nicht sichtbar.
- Cyclic Update: Die Werte der Properties werden zyklisch aktualisiert und angezeigt.
  Im eingeloggten Zustand in der SPS wird dadurch zusätzlicher Code ausgeführt.
- Execution Update: Die Werte der Properties werden nur dann im Online-View aktualisiert, wenn im Ausführungscode getter-/setter-Methoden für die Properties aufgerufen werden. **Dieses führt schnell zu Irritationen und ist nur in seltenen Fällen relevant.**

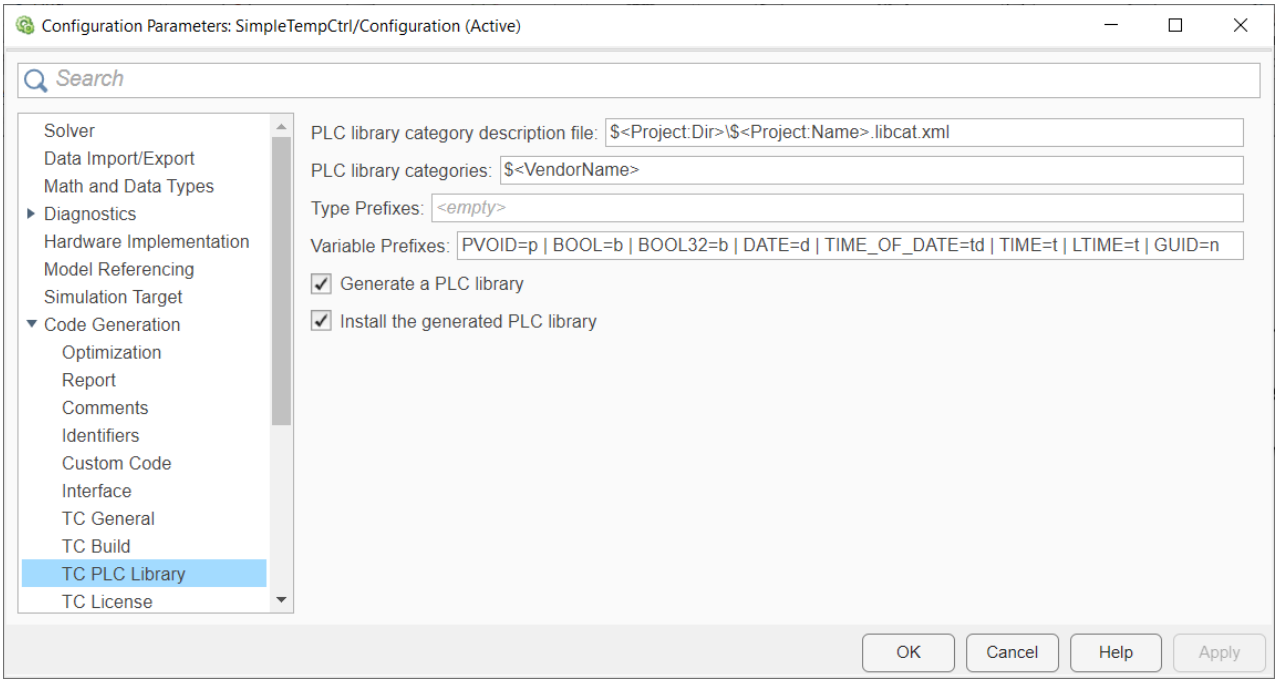
Kein TcCOM-Wrapper für Online Change-fähige Module

Wenn das TcCOM Online Change-fähig ist, wird kein TcCOM-Wrapper-FB erzeugt, da die Version der SPS-Bibliothek und die Version des TcCOM-Objekts immer zueinander passen müssen, was beim Online Change des TcCOM nicht gewährleistet werden kann.

4.8.2.1.3 Konfigurieren und Installieren der SPS-Bibliothek

Konfigurieren der Typ- und Variablen-Präfixe

Sie können unter **Code Generation > TC PLC** Library Ihre individuellen Typ- und Variablen-Präfixe konfigurieren, die in der erstellen SPS-Bibliothek genutzt werden sollen. Als Default-Einstellung werden die Variablen-Präfixe entsprechend dieser Konvention (siehe Identifiers for variables and intances) erzeugt.



Tragen Sie die gewünschten Präfixe als *pipe-seperated list* ein.

Installieren der SPS-Bibliothek

Wie oben beschrieben, können Sie konfigurieren, welche Funktionsbausteine (PLC-FB [▶ 211] und/oder TcCOM-Wrapper-FB [▶ 212]) in Ihrer SPS-Bibliothek enthalten sein sollen. Um die erstellten Funktionsbausteine in der SPS nutzen zu können, muss die entsprechende SPS-Bibliothek auf Ihrem TwinCAT Engineering System installiert sein.

Situation 1:

✓ Sie nutzen das Target for Simulink® auf demselben PC, auf dem Sie Ihre SPS programmieren möchten.

1. Erzeugen Sie eine SPS-Bibliothek und installieren Sie diese direkt auf Ihrem lokalen Engineering System aus Simulink® heraus.
2. Wählen Sie dazu unter **TC PLC Library** entsprechende Checkboxen aus:

⇨ Nach erfolgreichem Build ist die neue Version der SPS-Bibliothek direkt im lokalen TwinCAT XAE verfügbar.

Situation 2:

✓ Sie möchten die SPS-Bibliothek auf beliebigen TwinCAT-XAE-Systemen nutzen.

1. Erstellen Sie ein TMX-Archiv [▶ 139].
2. Kopieren Sie das TMX-Archiv [▶ 139] auf ein TwinCAT Engineering System Ihrer Wahl.
3. Installieren Sie die SPS-Bibliothek beim Entpacken des TMX-Archiv [▶ 139]s.

⇨ Die im Archiv enthaltene Bibliothek ist im TwinCAT XAE verfügbar.

4.8.2.2 Anwenden des TcCOM-Wrapper-FB

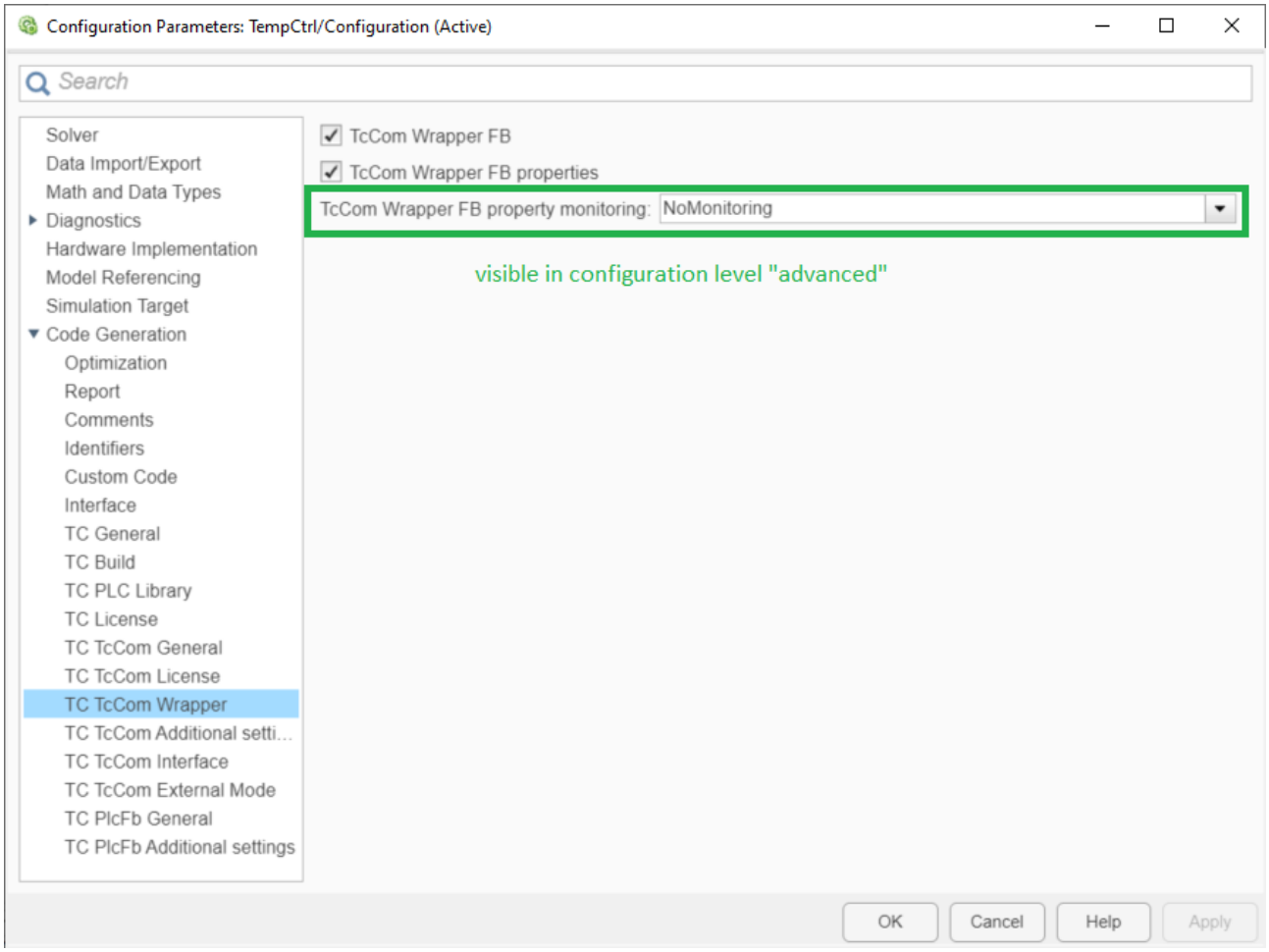
Es gibt zwei Wege, ein TcCOM-Objekt aus der SPS aufzurufen:

1. Referenzieren einer statischen Objekt-Instanz
2. Dynamisches Instanziieren eines Objekts aus der SPS

⇨ Für beide Wege wird der TcCOM-Wrapper-Funktionsblock aus der generierten SPS-Bibliothek verwendet.

Konfiguration des TcCOM-Wrapper-Funktionsblocks in Simulink®

Navigieren Sie zu: **Configuration Parameters > Code Generation > TC TcCom Wrapper**. Aktivieren Sie hier die Checkbox **TcCom Wrapper FB**. In der Standardkonfiguration ist die Checkbox nicht gesetzt. Über die Checkbox **TcCom Wrapper FB properties** können Sie konfigurieren, ob die Modellparameter am Funktionsblock als Properties erstellt werden sollen.



Im configuration level “advanced” kann auch das Monitoring-Attribut der Properties präzisiert werden. Im Standardfall ist „No Monitoring“ eingestellt, d. h. es wird kein Attribut gesetzt.

Einstellung in Simulink®	Attribut am Property
ExecutionUpdate	{attribute 'monitoring' := 'variable'}
CyclicUpdate	{attribute 'monitoring' := 'call'}

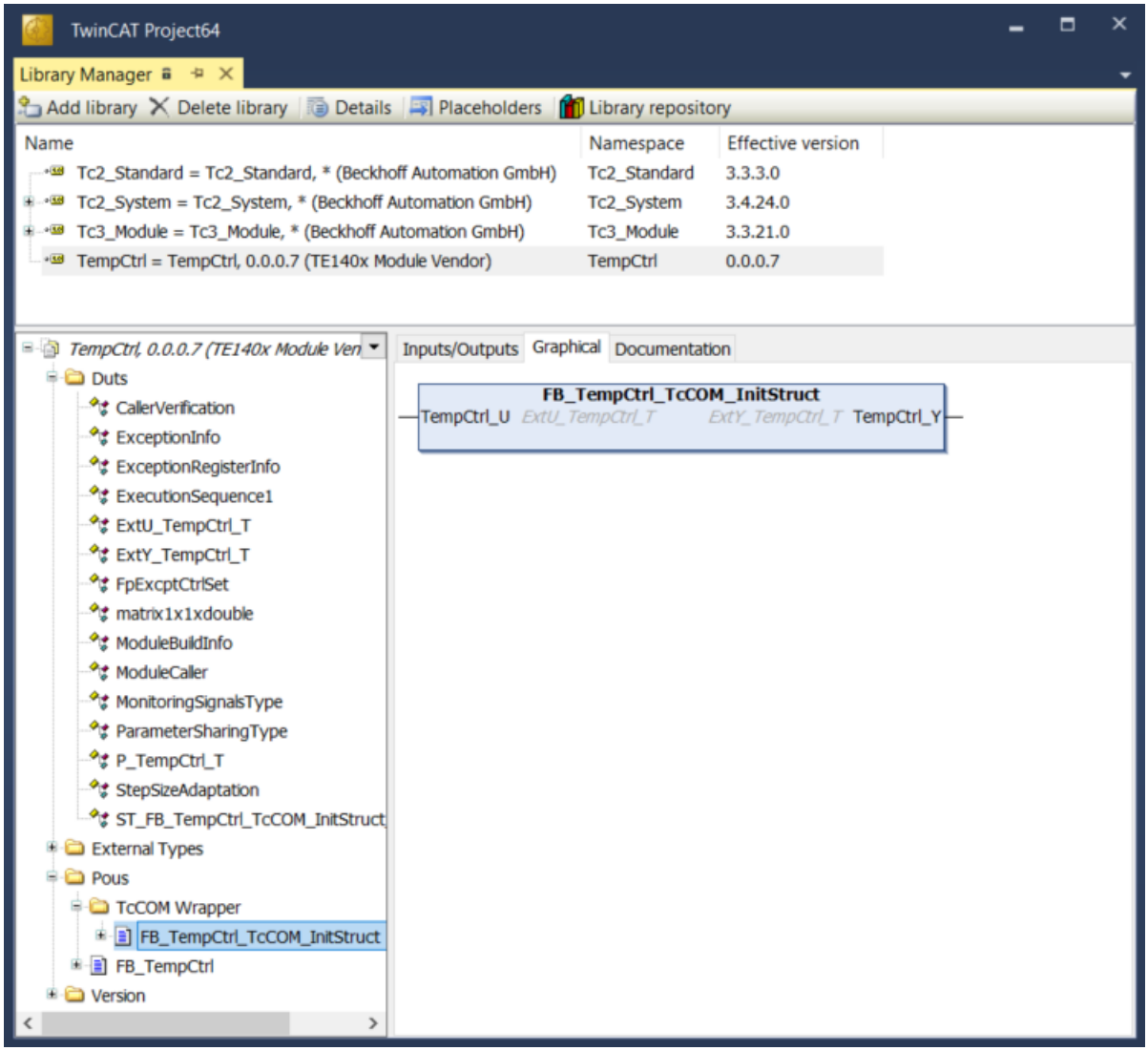
Monitoring Attribute beeinflussen die Sichtbarkeit der Attribut-Werte im Online-View, d. h. wenn man sich in die SPS eingeloggt hat und die aktuellen Werte der Properties am FB beobachten möchte.

- No Monitoring: Die Werte sind im Online-View nicht sichtbar.
- Cyclic Update: Die Werte der Properties werden zyklisch aktualisiert und angezeigt.
  Im eingeloggten Zustand in der SPS wird dadurch zusätzlicher Code ausgeführt.
- Execution Update: Die Werte der Properties werden nur dann im Online-View aktualisiert, wenn im Ausführungscode getter-/setter-Methoden für die Properties aufgerufen werden. **Dieses führt schnell zu Irritationen und ist nur in seltenen Fällen relevant.**

Für weitere Details siehe auch SPS-Bibliothek erzeugen und installieren [▶ 211] bzw. Create the TcCOM-Wrapper-FB.

Instanz des TcCOM-Wrapper-Funktionsblock erzeugen

1. Erstellen Sie ein SPS-Projekt.
2. Fügen Sie die gewünschte Bibliothek unter **References** hinzu.



⇨ Sie erhalten unter **Pous/TcCOM Wrapper** einen Funktionsblock, den Sie in der SPS instanziieren können. Darüber hinaus werden notwendige Datentypen im Ordner *Duts* angelegt.

Variante 1: Referenzieren einer statischen Modul-Instanz

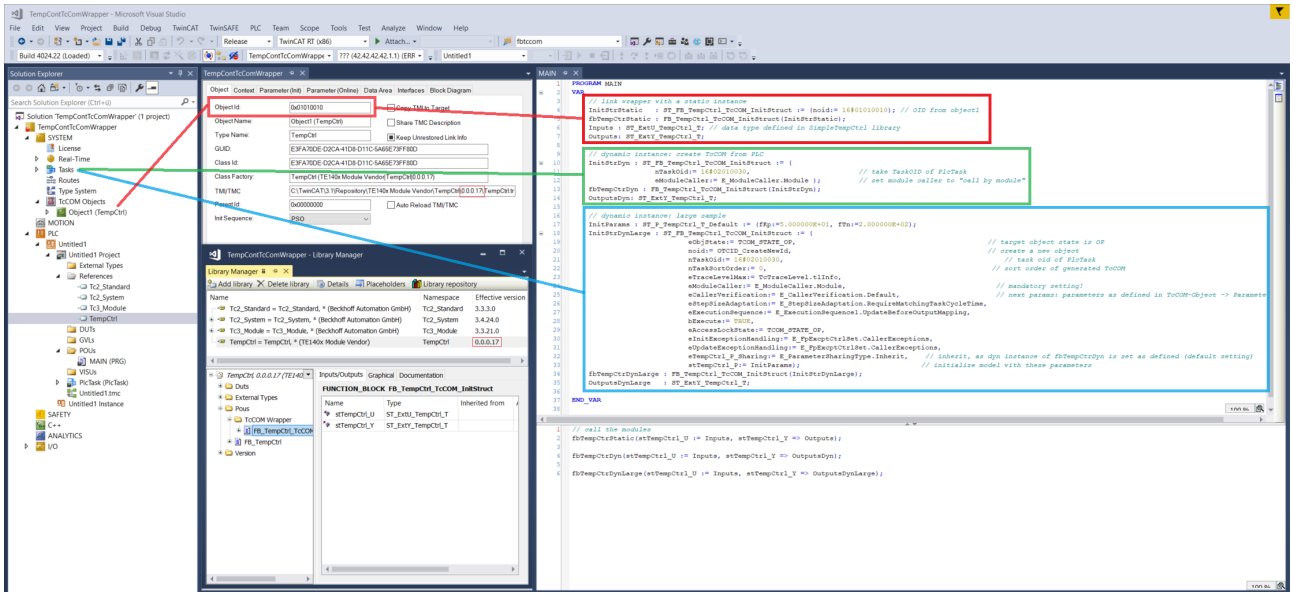
Der Funktionsblock kann genutzt werden, um auf vorher im XAE, z. B. unter **System > TcCOM Objects** angelegte Modulinstanzen, zuzugreifen. Für diesen statischen Fall muss die Objekt-ID der entsprechenden Modulinstanz bei der Deklaration der Funktionsblock-Instanz übergeben werden. Siehe roter Bereich in untenstehender Grafik.

- Die Instanz des TcCOM-Objekts und die aufrufende SPS müssen in derselben Task laufen.
- Auf der Instanz des TcCOM-Objekts ist darauf zu achten, dass unter Parameter (Init) der Eintrag *ModuleCaller* auf *Module* steht und nicht auf *CyclicTask*.
- Der benötigte Speicher für das TcCOM wird in diesem Fall aus dem *non paged pool* des Systems bezogen.

Variante 2: Dynamisches Instanziieren und Referenzieren aus der SPS

Der Funktionsblock kann auch so genutzt werden, dass ein TcCOM-Objekt aus der SPS heraus erzeugt und mit dem Wrapper verknüpft wird. In untenstehender Grafik sind das der grüne Bereich als Minimalbeispiel und der blaue Bereich mit erweiterter Parametrierung.

- Über die TaskOid der SPS Task muss angegeben werden, in welcher Echtzeit-Task der Wrapper aufgerufen wird.
- Der ModuleCaller muss auch hier (über die Init Struktur) auf *Module* eingestellt werden.
- Der benötigte Speicher für das TcCOM wird in diesem Fall aus dem Router Memory bezogen.



Grafik in der Webbrowser Ansicht

Klicken Sie mit der rechten Maustaste auf das Bild und wählen Sie „open image in new tab“, um das Bild besser lesen zu können.

Der Quellcode zur oben gezeigten Grafik ist verfügbar in MATLAB® über das Command Window:

```
TwinCAT.ModuleGenerator.Samples.Start('TcComWrapper TemperatureController')
```

Arbeiten mit den Properties des TcCOM-Wrapper-FB

Properties am FB bieten eine einfache Möglichkeit, mit Modul-Parametern eines TcCOM zu interagieren, siehe auch Best Practice: Zugriff auf Daten des TcCOM [▶ 151].

Beispiel

Durch Setzen des *Parameter: Initial Values* unter Tc TcCom Interfaces werden die Modell-Parameter als Modul-Parameter angelegt (standardmäßig eingeschaltet). Erzeugen Sie nun den „TcCom Wrapper FB“ mit der Option „TcCom Wrapper FB properties“. Setzen Sie das property monitoring auf „CyclicUpdate“, um die Wert-Änderung des Property im Online-View direkt sehen zu können.

Dann können Sie beispielsweise wie folgt auf die Modul-Parameter zugreifen:

```
PROGRAM MAIN
VAR

   // dynamic instance: create TcCOM from PLC
   InitStrDyn : ST_FB_TempCtrl_TcCOM_InitStruct_InitStruct := (
                TaskOid:= 16#02010030,                    // take TaskOID of PlcTask
                eModuleCaller:= ModuleCaller.Module );        // set module caller to "call by module
"
   fbTempCtrDyn : FB_TempCtrl_TcCOM_InitStruct(InitStrDyn);

   Outputs     : ExtY_TempCtrl_T;    // input
   Inputs      : ExtU_TempCtrl_T;    // output
   Parameters  : P_TempCtrl_T;       // parameter

   bChange: BOOL;
END_VAR
```

```
fbTempCtrDyn(TempCtrl_U := Inputs, TempCtrl_Y => Outputs);

IF bChange THEN
   Parameters.Kp := 10;
   fbTempCtrDyn.TempCtrl_P := Parameters;
END_IF
```

Arbeiten mit dem ADI Interface

⚠ WARNUNG

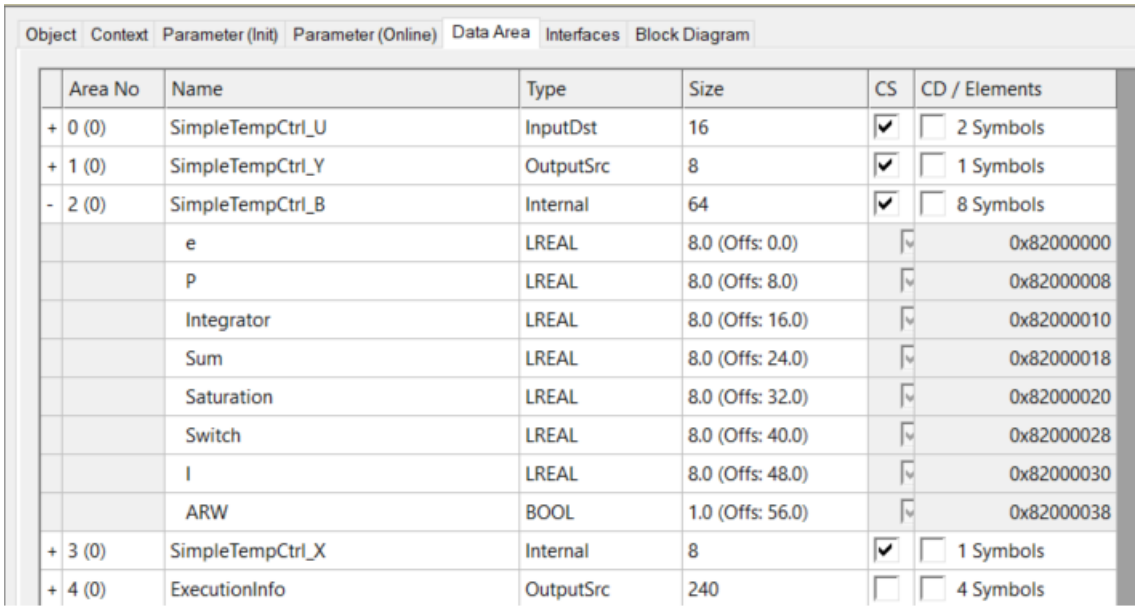
Uneingeschränkter Lese- und Schreibzugriff

Über das ITc_ADI Interface bekommen Sie einen Pointer auf den Speicherbereich einer DataArea. Entsprechend können Sie dort uneingeschränkt lesen und schreiben.

Im Folgenden ist ein Beispiel dargestellt, wie man lesend auf die DataArea des BlockIO zugreifen kann:

```
stInitTemp : ST_Funktionsblock_SimpleTempCtrl_TcCOM_InitStruct := (nOid := 16#01010010);
FunktionsblockTempCtr  : Funktionsblock_SimpleTempCtrl_TcCOM_InitStruct(stInitTemp);
stTempCtr_BlockIO  : ST_B_SimpleTempCtrl_T;
pStTempCtr_BlockIO : POINTER TO ST_B_SimpleTempCtrl_T;
hr : HRESULT;
```

```
(* read data are via ADI Interface *)
// get a pointer to Data Area
hr := FunktionsblockTempCtr.ipADI.GetImagePtr(size := SIZEOF(stTempCtr_BlockIO), offs := 0, adi_x :=
 2, ppData := ADR(pStTempCtr_BlockIO));
IF hr = 0 THEN
   // copy data to a local variable
   MEMCPY(ADR(stTempCtr_BlockIO), pStTempCtr_BlockIO, SIZEOF(stTempCtr_BlockIO));
   // always release the pointer!
   FunktionsblockTempCtr.ipADI.ReleaseImagePtr(pData := pStTempCtr_BlockIO);
END_IF;
```



	Area No	Name	Type	Size	CS	CD / Elements
+	0 (0)	SimpleTempCtrl_U	InputDst	16	☑	☐ 2 Symbols
+	1 (0)	SimpleTempCtrl_Y	OutputSrc	8	☑	☐ 1 Symbols
-	2 (0)	SimpleTempCtrl_B	Internal	64	☑	☐ 8 Symbols
		e	LREAL	8.0 (Offs: 0.0)		0x82000000
		P	LREAL	8.0 (Offs: 8.0)		0x82000008
		Integrator	LREAL	8.0 (Offs: 16.0)		0x82000010
		Sum	LREAL	8.0 (Offs: 24.0)		0x82000018
		Saturation	LREAL	8.0 (Offs: 32.0)		0x82000020
		Switch	LREAL	8.0 (Offs: 40.0)		0x82000028
		I	LREAL	8.0 (Offs: 48.0)		0x82000030
		ARW	BOOL	1.0 (Offs: 56.0)		0x82000038
+	3 (0)	SimpleTempCtrl_X	Internal	8	☑	☐ 1 Symbols
+	4 (0)	ExecutionInfo	OutputSrc	240	☐	☐ 4 Symbols

Der Methode GetImagePtr werden unter anderem `adi_x` und `offs` übergeben. Diese bestimmen die DataArea an sich, in diesem Fall DataArea Nummer 2 (SimpleTempCtrl_B), und den Datenbereich in der Area, der gelesen/geschrieben werden soll, in diesem Fall ohne Offset und die gesamte Größe der DataArea (also die gesamte DataArea).

Beim Schreiben sind im obigen Beispiel entsprechend beim `MEMCPY` die Quelle und das Ziel zu tauschen.

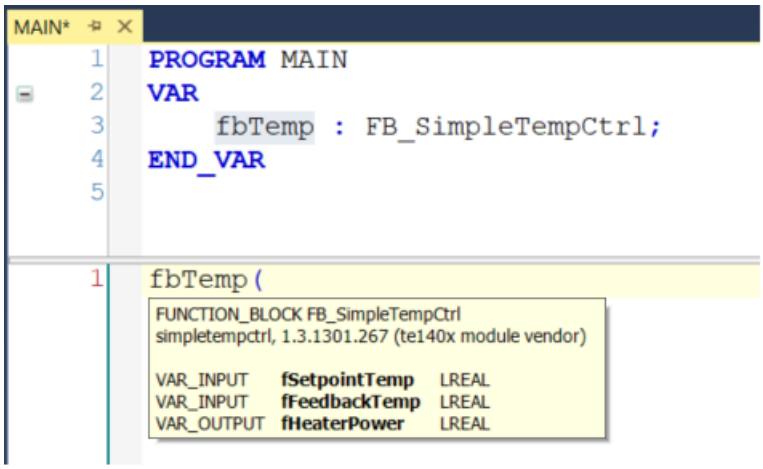
Für weitere Hinweise zur Interaktion mit dem TcCOM beachten Sie Best Practice: Zugriff auf Daten des TcCOM [▶ 151].

4.8.2.3 Anwenden des SPS Funktionsbausteins (PLC-FB)

Die Anwendung des SPS Funktionsbausteins (PLC-FB, FB_<modelname>) in der SPS Bibliothek ist fokussiert auf dessen Einfache Anwendung. Damit einher gehen auch ein paar Einschränkungen gegenüber dem TcCOM (bzw. dem SPS-Wrapper-FB).

Anwendung

Erstellen Sie eine oder mehrere Instanzen des PLC-FBs aus der erstellten SPS Bibliothek. Beim Schreiben des Quellcodes steht Ihnen die IntelliSense zur Verfügung, sodass Sie bequem die erwarteten Inputs und Outputs sehen können.



Sollten in Simulink® Bus-Objekte als Inputs oder Outputs verwendet worden sein, werden diese Datentypen automatisch in der SPS Bibliothek als Strukturen definiert.

Einschränkungen

Der PLC-FB lässt keinen Zugriff auf die Modellparameter über Properties am Funktionsbaustein zu. Es gibt zwei Möglichkeiten die Modellparameter dennoch zu verändern:

- Über den External Mode [▶ 222]
- Bei Verwendung von Reusable Code [▶ 196], kann eine TcCOM-Instanz die Parameter definieren. Alle Instanzen des PLC-FB sind automatisch auf „inherit“ eingestellt und übernehmen die Parameter der vorgebenden TcCOM-Instanz.

Der PLC-FB enthält zudem kein Block Diagramm [▶ 198] im TwinCAT XAE. Debugging des Bausteins erfolgt über den TwinCAT C++ Debugger, wie hier [▶ 220] beschrieben oder mittels External Mode.

4.8.2.3.1 Online Change der SPS-Bibliothek

Sie können während TwinCAT im Run-Modus ist, im TwinCAT XAE die SPS-Bibliotheksversion tauschen und diese per Online Change in die laufende Applikation spielen. Somit lassen sich alle in einer SPS-Bibliothek befindlichen Funktionsbausteine ohne TwinCAT-Neustart updaten.

Schritt-für-Schritt vorgehen:

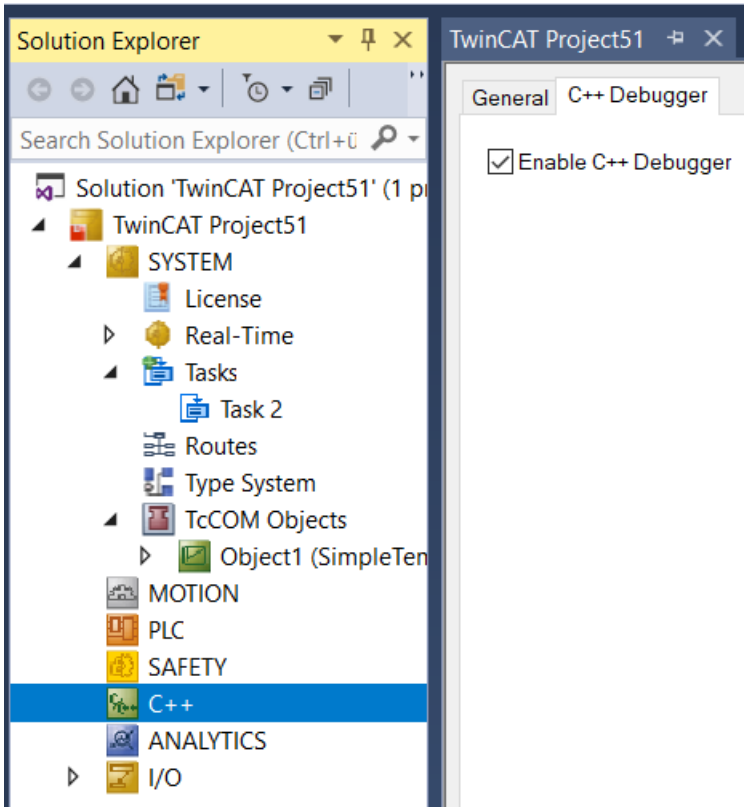
1. Erstellen Sie eine erste SPS-Bibliotheksversion mit dem TwinCAT Target for Simulink®.
2. Binden Sie diese SPS-Bibliotheksversion in ein SPS-Projekt ein.
3. Aktivieren Sie ihre TwinCAT-Konfiguration mit der ersten SPS-Bibliotheksversion (z. B. Version 0.0.0.1).
4. Adaptieren Sie ihr Simulink®-Modell und erstellen Sie daraus eine SPS-Bibliotheksversion (0.0.0.2).
5. Wählen Sie in der SPS unter **Referenzen** die neu erstelle SPS-Bibliotheksversion aus (ggf. müssen Sie die neue Bibliothek auf dem XAE-System installieren).
6. Wählen Sie **Build > Build Solution**, um das Projekt neu zu bauen.
7. Wählen Sie **Login > Login with online change** (mehr Informationen in der SPS Dokumentation).

4.8.3 Debugging

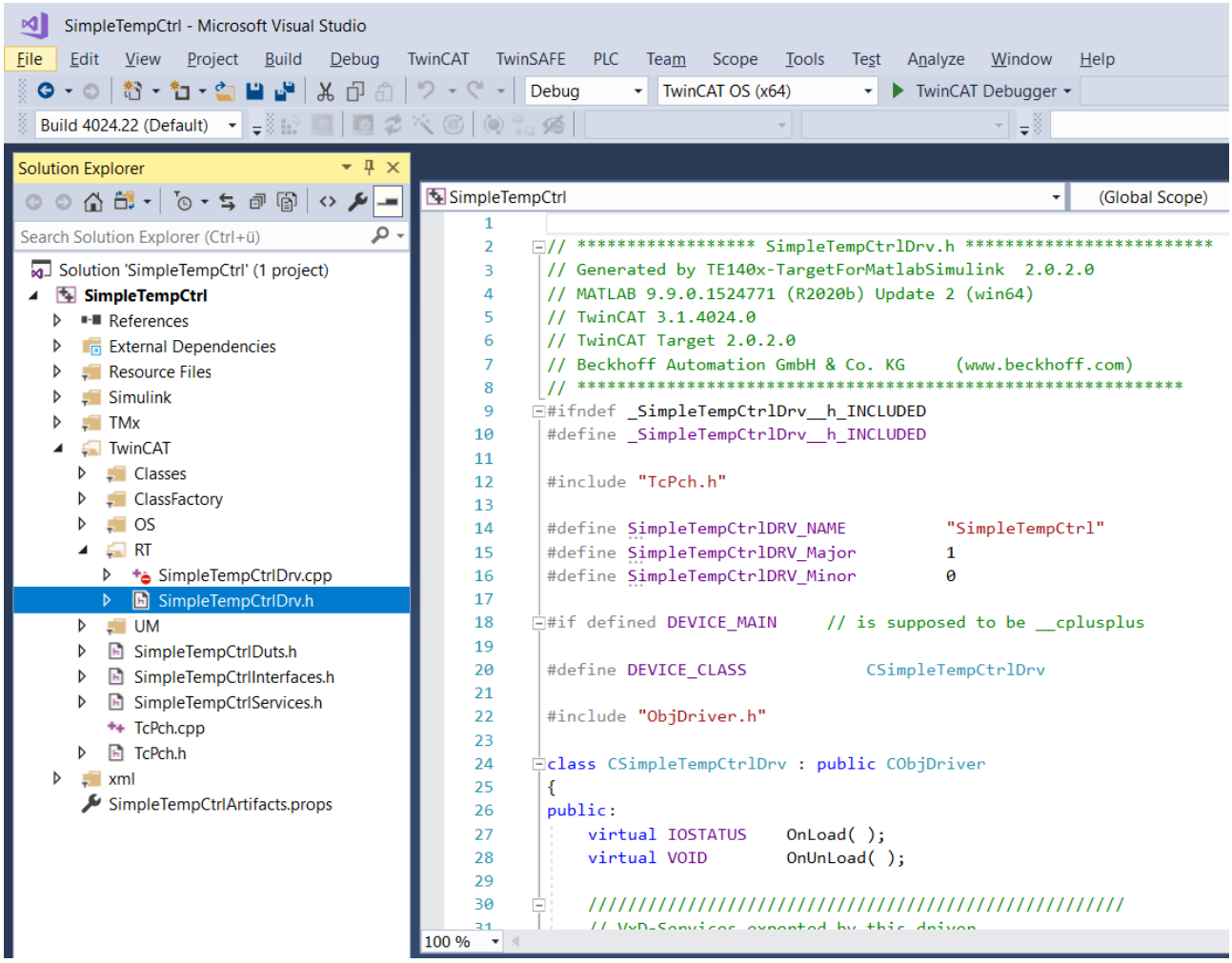
Neben dem Debugging über den External Mode [▶ 222] und über das Block Diagram im TwinCAT XAE [▶ 202], können Sie auch ganz klassisch das erstelle C++-Projekt zum Debugging verwenden.

Schritt-für-Schritt vorgehen:

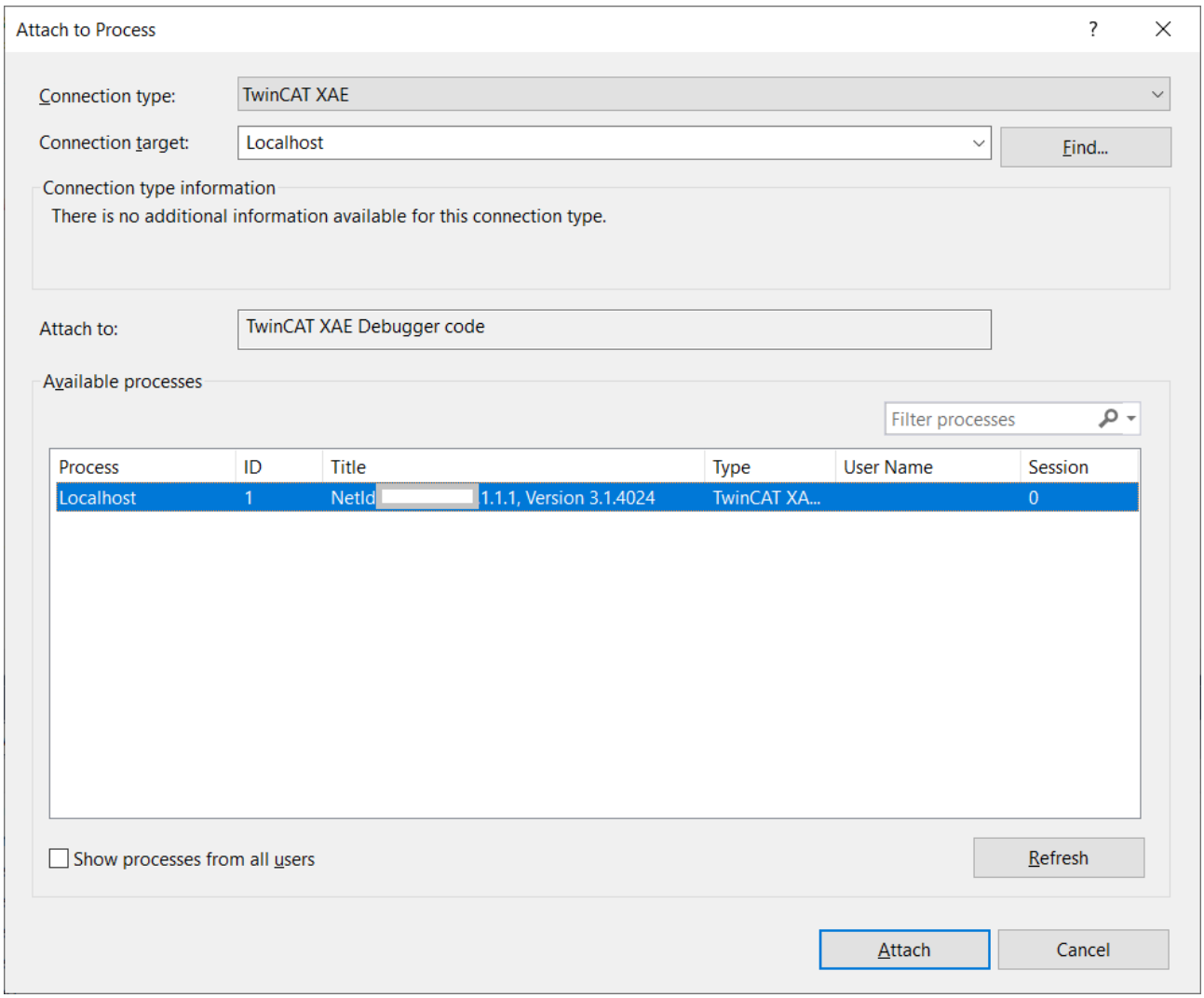
1. Stellen Sie sicher, dass Ihre TwinCAT-Applikation mit aktiviertem C++-Debugger aktiviert wurde.



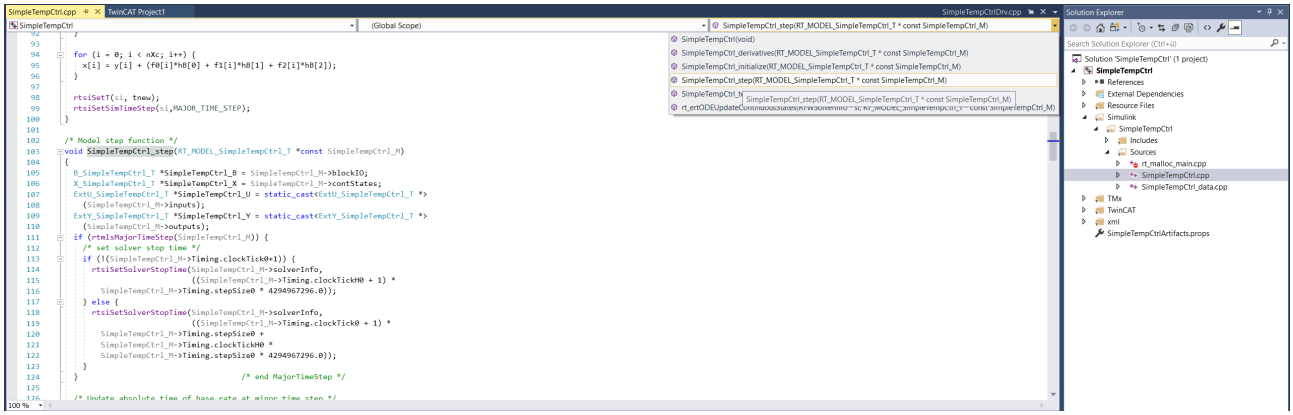
2. Öffnen Sie das bei der Code-Generierung erstellte C++-Projekt, das zu dem Modul gehört, welches Sie debuggen möchten.
   Das Projekt finden Sie im Ordner < *SimulinkModelName*>*_tcgrt*, welcher im aktuellen MATLAB® -Pfad erstellt wird, wenn Sie den Code-Generierungsprozess starten.
3. Suchen Sie in diesem Ordner nach der Datei <*SimulinkModelName*>*.vcxproj*.
   Sie können das <*SimulinkModelName*>*.vcxproj* in Visual Studio alleine öffnen oder auch die vcxproj-Datei in Ihrer TwinCAT-Solution unter C++ mit „Add existing Item“ hinzufügen.



4. Wählen Sie in der Menüleiste **Debug > Attach to Process** und wählen Sie als Connection Type „TwinCAT XAE“ sowie unter Connection target Ihr gewünschtes Zielsystem. Wählen Sie dann **Attach**.



5. Setzen Sie im C++-Code Breakpoints und steppen Sie wie gewohnt durch Ihren Code. Tipp: Bei der Ausführung des Codes wird die Step-Function verwendet, welche Sie im Ordner **Simulink > Sources > <SimulinkModelName>.cpp** finden.



4.8.4 Verbinden mit dem External Mode

Sie können sich per External Mode aus Ihrer Simulink®-Umgebung auf ein laufendes TcCOM-Objekt oder eine Instanz des SPS-Funktionsbausteins in der TwinCAT XAR verbinden.

Einschränkung zum Code interface packaging

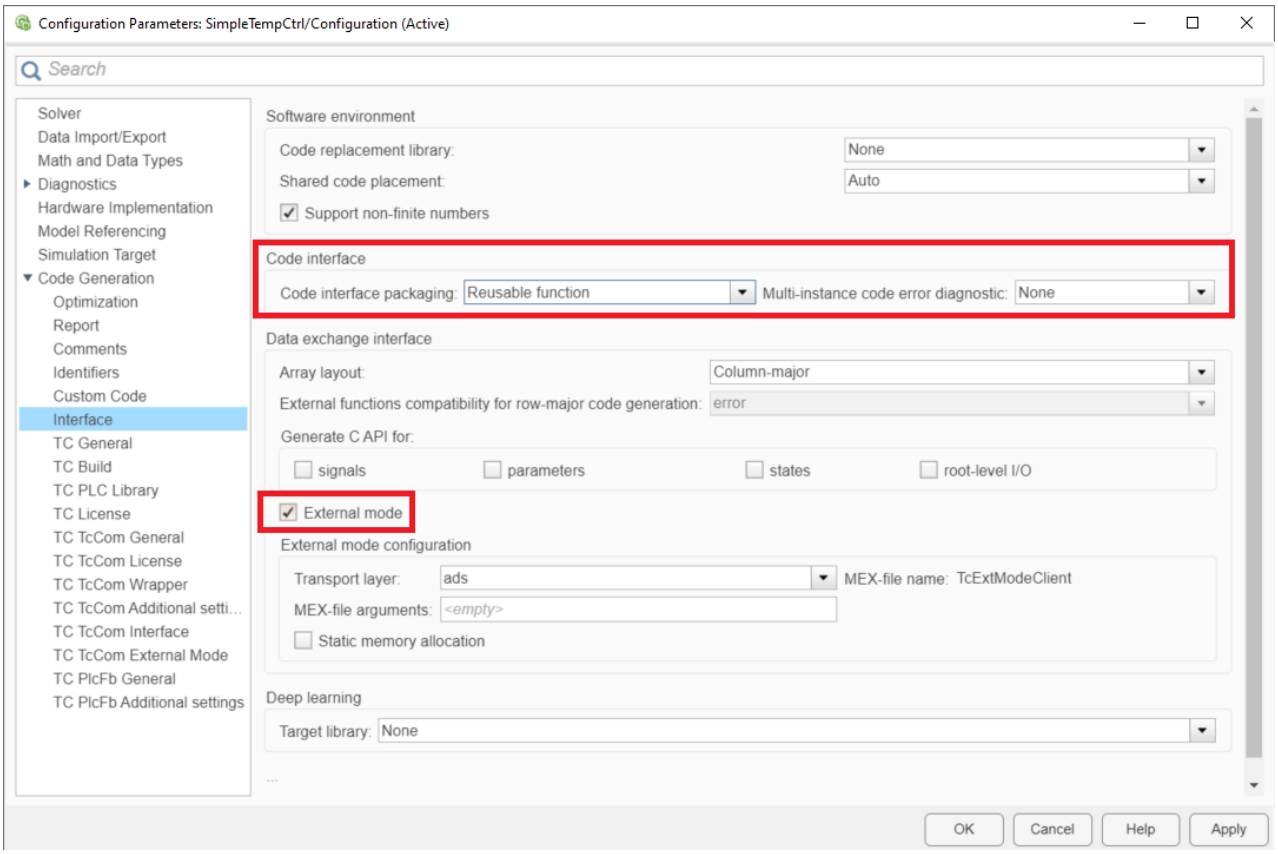
Das Code interface packaging definiert das Verhalten bei mehreren Instanzen einer erstellen Klassen in TwinCAT [▶ 196]. Soll der External Mode genutzt werden, ist die Einstellung C++ Class nicht erlaubt.

Simulationszeit in Simulink® auf „inf“ setzen

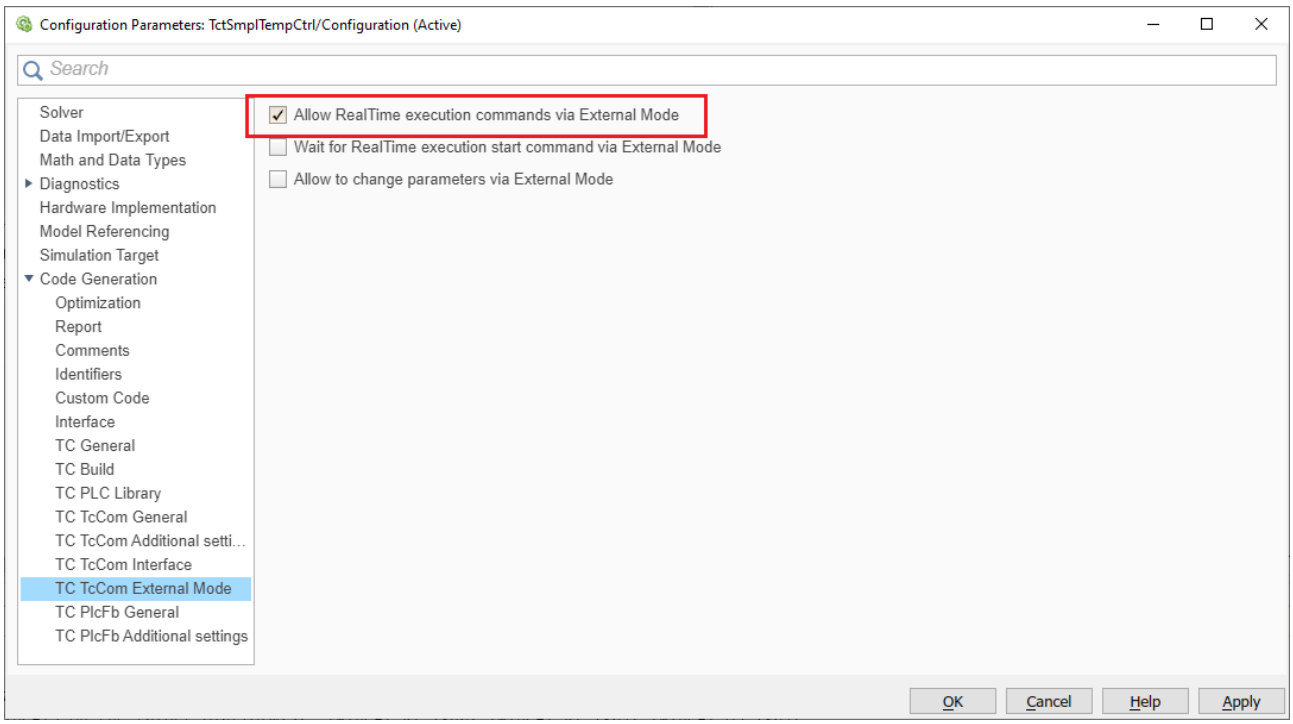
Setzen Sie die Simulink®-Simulationszeit auf „inf“. Für den Betrieb in TwinCAT ergibt es keinen Sinn, nach einer definierten Zeit die Ausführung des Moduls zu stoppen.

✓ Einstellungen bei der Code-Generierung in Simulink®

1. Setzen Sie unter **Code Generation > Interface** den Parameter **External mode**.
   - Beachten Sie beim **Code interface packaging**:
     - Nonreusable function: erlaubt
     - Reusable function: erlaubt, Multi-instance code error diagnostic auf **None** stellen
     - C++ Class: nicht erlaubt

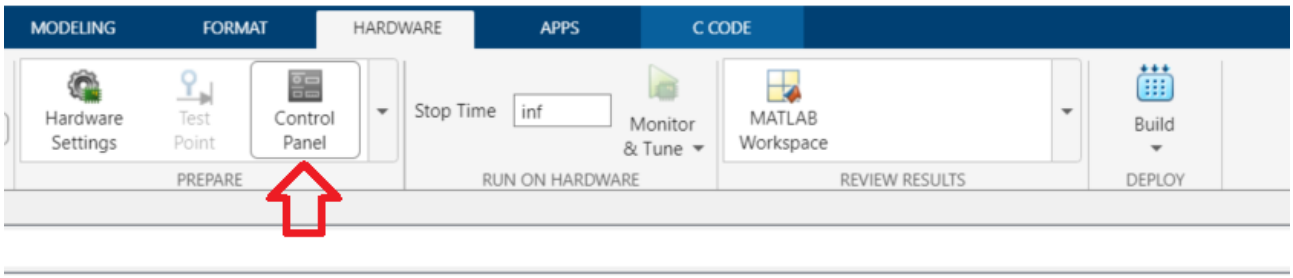


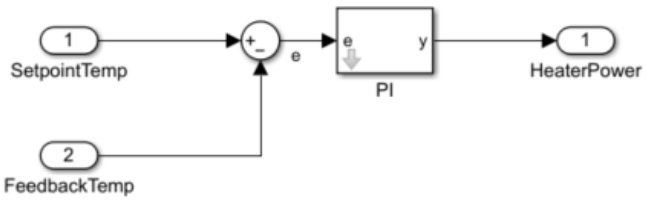
2. Definieren Sie die Berechtigungen des External mode (separat für TcCOM und PlcFb einstellbar).
   ⇨ Im folgenden Screenshot exemplarisch für **TcCOM** gezeigt.



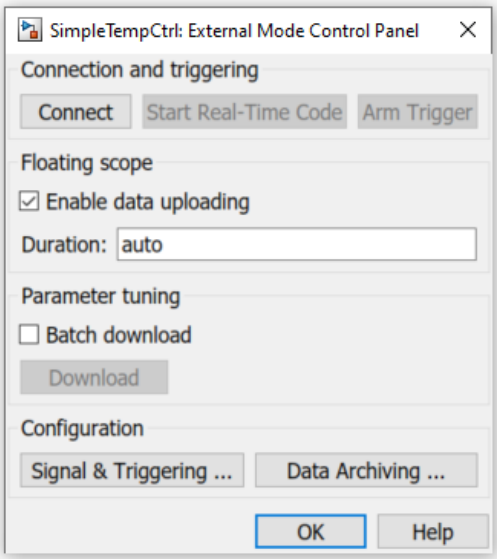
✓ Verbinden auf ein Laufzeit-Objekt mit dem External Mode

3. Öffnen Sie das External Mode **Control Panel**.
4. Wählen Sie **Connect**.

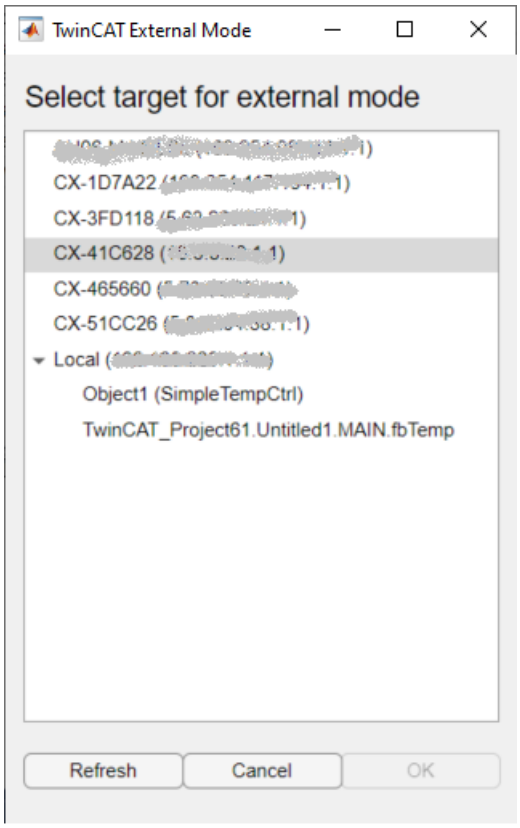




TwinCAT Target for MATLAB/Simulink
Sample model "SimpleTempCtrl"
Subjects:
- Basics



5. Wählen Sie das verbundene Target und die Objekt-Instanz aus.



⇨ Nach Auswahl von **OK** sind Sie mit dem Objekt verbunden. Der Button **Connect** auf dem External Mode Control Panel hat sich zu **Disconnect** geändert und Sie können in Simulink® die vom Target übertragene Simulationszeit sehen.

Wie in obiger Grafik zur Auswahl des Objekts in einem Target zu sehen ist, ist der External Mode sowohl für TcCOM Instanzen als auch für PLC-FB Instanzen verfügbar.

Bidirektionale ADS-Route notwendig

Für den External Mode ist eine bidirektionale ADS-Route notwendig. Unidirektionale Routen führen zu einem Timeout in der Kommunikation.

4.8.5 Exception Handling

Bei der Abarbeitung des aus MATLAB® oder Simulink® autogenerierten C++-Codes in TwinCAT kann es zur Laufzeit zu Floating Point Exceptions kommen, wenn zum Beispiel ein bei der Programmierung unerwarteter Wert in eine Funktion übergeben wird. Die Behandlung von solchen Exceptions wird im Folgenden beschrieben.

Was ist eine Floating Point Exception?

Eine Floating Point Exception tritt dann auf, wenn eine arithmetisch nicht exakt ausführbare Rechenoperation in der Floating Point Unit der CPU beauftragt wird. IEEE 754 definiert diese Fälle: *inexact, underflow, overflow, divide-by-zero, invalid-operation*. Tritt einer dieser Fälle auf, wird ein Status Flag gesetzt, welches auf die nicht exakt ausführbare Rechenoperation hinweist. Es wird des Weiteren definiert, dass jede Rechenoperation ein Ergebnis liefern muss, und zwar ein solches, das in der Mehrzahl der Fälle dazu führt, dass man die Exception ignorieren kann.

Beispielsweise ergibt eine Division durch Null +inf oder -inf. Wird im weiteren Code ein Wert durch inf geteilt, ergibt dies Null, sodass keine Folgeprobleme zu erwarten sind. Wird inf allerdings multipliziert, oder werden andere Rechenoperationen mit inf ausgeführt, sind dies *invalid operations,* deren Ergebnis als Not-a-Number (NaN) dargestellt wird.

Wie reagiert die TwinCAT-Laufzeit bei Exceptions?

> **TwinCAT C++ Debugger nicht aktiv**
>
> Folgende Ausführungen gelten nur für den Fall, dass der C++ Debugger **nicht** auf dem TwinCAT-Laufzeitsystem aktiviert ist. Bei aktiviertem C++ Debugger werden Exceptions vom Debugger abgefangen und können behandelt werden, siehe Debugging [▶ 220].

Standardverhalten

Default-Einstellung in TwinCAT ist, dass bei „divide-by-zero“ und „invalid-operation“ die Ausführung des Programms gestoppt wird und TwinCAT eine Fehlermeldung ausgibt.

Task-Einstellung: Floating Point Exceptions

Diese Default-Einstellung lässt sich auf Ebene einer jeden TwinCAT Task verändern. Wird die Checkbox „Floating Point Exception“ deaktiviert, führt eine Exception **nicht** zu einem TwinCAT-Stopp und es wird **keine** Fehlermeldung ausgegeben. Diese Einstellung ist dann gültig für alle Objekte, die durch diese Task aufgerufen werden. In der Folge müssen Sie in der Applikation darauf achten, dass NaN- und inf-Werte entsprechend im Programmcode behandelt werden.

Prüfen auf NaN und Inf

Wird beispielsweise ein NaN per Mapping an ein TwinCAT-Objekt weitergegeben, welches Floating Point Exceptions aktiviert hat, führt eine Rechenoperation mit NaN in diesem Objekt zu einer Exception und in der Folge zu einem TwinCAT-Stopp. Daher muss direkt nach dem Mapping auf NaN bzw. inf geprüft werden. In der SPS stehen dazu entsprechende Funktionen in der Bibliothek Tc2_Utilities zur Verfügung, bspw. LrealIsNaN.

Try-Catch-Anweisung

Eine weitere Möglichkeit zum Umgang mit Exceptions ist die Einbettung in eine Try-Catch-Anweisung. In der SPS stehen dazu die Anweisungen __TRY, __CATCH, __FINALLY, __ENDTRY zur Verfügung. Sind auf der aufrufenden Task Floating Point Exceptions aktiviert und tritt innerhalb des Try-Catch eine Exception auf, so wird diese im Catch-Zweig gefangen und kann behandelt werden. Entsprechend werden bei dieser Vorgehensweise keine Variablen auf inf oder NaN gesetzt. Wichtig ist aber zu bemerken, dass der Code in Try-Zweig nur bis zu der Stelle der Exception durchlaufen wird und dann ein Sprung zum Catch-Zweig erfolgt. Im Applikationscode ist entsprechend zu beachten, dass interne Zustände im Try-Zweig ggf. nicht konsistent sind.

Dump Files

Ab TwinCAT 3.1.4024.22 (XAR) können zur Laufzeit Dump Files im Falle von Exceptions im TcCOM-Objekt erstellt werden.

Präzisierung des Verhaltens bei Exceptions auf Objekt-Ebene

Neben der Möglichkeit, das Verhalten bei Exceptions auf Task-Ebene zu beeinflussen, kann auch das Verhalten auf Ebene eines TwinCAT-Objekts, d. h. des generierten TcCOM oder des generierten SPS-Funktionsbausteins (PLC-FB [▶ 219]), präzisiert werden.

Auf Objektebene ist mit dem TwinCAT Target for Simulink® eine Fülle an Möglichkeiten realisierbar. Im Grunde basieren aber alle im Folgenden dargelegten Optionen auf oben genannten Prinzipien.

Definition des Objekt-Verhaltens bei auftretenden Exceptions

Es stehen insgesamt 9 unterschiedliche Einstellungen zur Verfügung.

- **CallerExceptions** (default): Exceptions werden so ausgelöst, wie an der aufrufenden Task konfiguriert.
- **ThrowExceptions**: Exceptions im TwinCAT-Objekt werden in jedem Fall ausgelöst, unabhängig davon, wie die Task konfiguriert ist.
  - Eine Exception verursacht eine TwinCAT Fehlermeldung und einen TwinCAT Stopp
- **SuppressExceptions**: Exceptions werden nicht ausgelöst, unabhängig davon wie die Task konfiguriert ist.

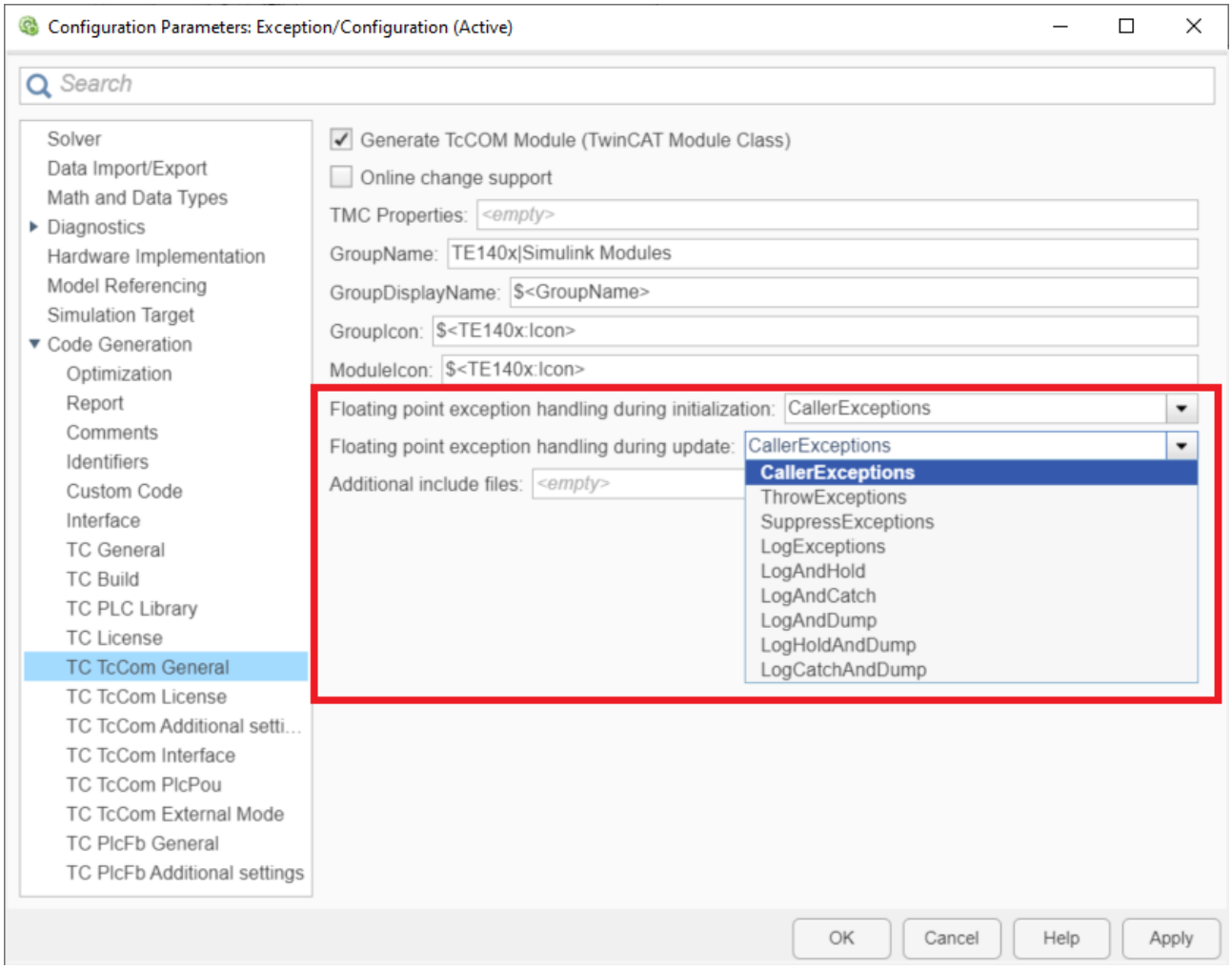
  - Eine Exception verursacht keinen TwinCAT-Stopp.
  - Ausgänge oder interne Zustände können NaN oder inf sein.
- **LogExceptions**: Exceptions werden ausgelöst, führen aber nicht zu einem TwinCAT-Stopp.
  - Eine Exception verursacht keinen TwinCAT-Stopp.
  - Ausgänge oder interne Zustände können NaN oder inf sein.
  - Der Ausgang ExecutionInfo wird mit Informationen über eine Exception im aktuellen Zyklus gefüllt. Treten mehrere Exceptions in einem Zyklus auf, wird nur die erste Exception am Ausgang angezeigt. Beim erneuten Aufruf des TwinCAT-Objekts werden die Informationen zurückgesetzt.
- **LogAndHold**: Exceptions werden ausgelöst. Die Ausführung des TwinCAT-Objekts wird gestoppt.
  - Eine Exception verursacht keinen TwinCAT-Stopp.
  - Ausgänge oder interne Zustände können NaN oder inf sein.
  - Der Ausgang ExecutionInfo wird mit Informationen über eine Exception im aktuellen Zyklus gefüllt. Treten mehrere Exceptions in einem Zyklus auf, wird nur die erste Exception am Ausgang angezeigt. Beim erneuten Aufruf des TwinCAT-Objekts werden die Informationen zurückgesetzt.
  - Die Ausführung des TwinCAT-Objekts wird nach Auftreten einer Exception gestoppt. TwinCAT selbst bleibt im Run-Modus. Ausführung wieder Starten: ReleaseObjectStop [▶ 231].
- **LogAndCatch**: Exceptions werden mit try-catch im TwinCAT-Objekt gefangen. Die Ausführung des TwinCAT-Objekts wird gestoppt.
  - Eine Exception verursacht keinen TwinCAT-Stopp.
  - Ausgänge oder interne Zustände können **keine** NaN oder inf enthalten.
  - Der Ausgang ExecutionInfo wird mit Informationen über eine Exception im aktuellen Zyklus gefüllt.
  - Die Ausführung des Codes endet an der Stelle der Exception. Von dort wird in den Catch-Zweig gesprungen, d. h. interne Zustände können inkonsistent sein.
  - Die Ausführung des TwinCAT-Objects wird nach Auftreten einer Exception gestoppt. TwinCAT selbst bleibt im Run-Modus. Ausführung wieder Starten: ReleaseObjectStop [▶ 231].
- **LogAndDump, LogHoldAndDump und LogCatchAndDump**
  - Verhalten wie LogExceptions
  - Zusätzlich wird ein Dump File auf dem Laufzeitsystem im TwinCAT Ordner *Boot* abgelegt. Weiteres zu Dump Files siehe hier [▶ 235].

Versionsempfehlung für 64bit-Zielsysteme

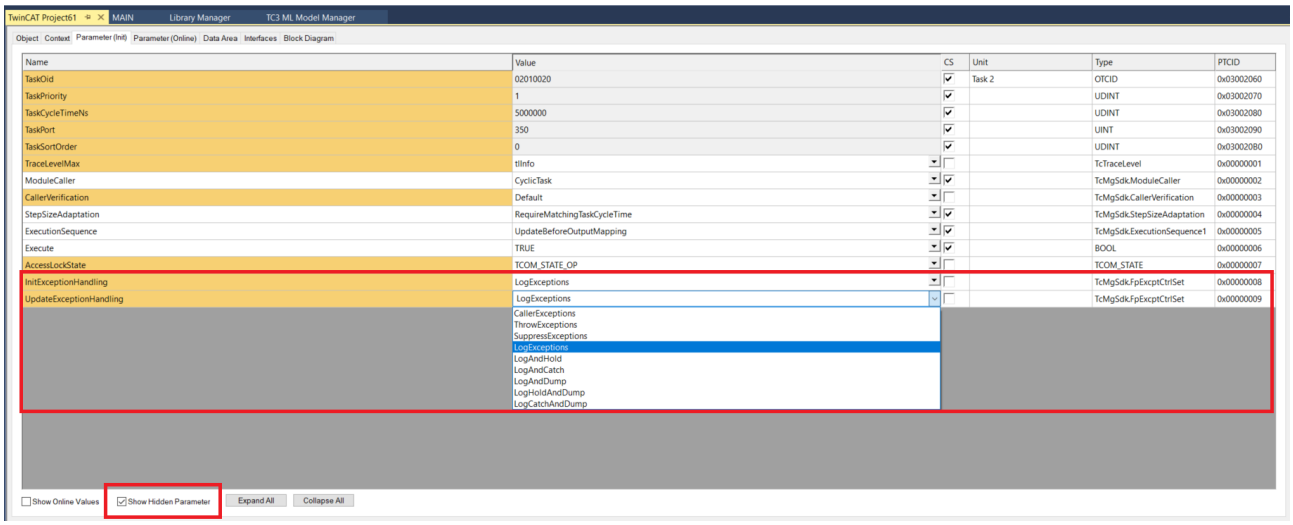
Bei Verwendung der Einstellungen „LogExceptions“, „LogAndHold“, „LogAndDump“, LogHoldAndDump“ wird die Nutzung einer XAR-Version von mindestens 3.1.4024.35 und TE1400-Version von mindestens 2.4.2.0 empfohlen.

<< Einstellung für TcCOM >>

Das Verhalten eines TcCOM-Objekts bei auftretenden Exceptions können Sie unter TC TcCOM General in den Code Generation Settings in Simulink® definieren. Das Verhalten ist dabei getrennt für die Initialisierungsphase des TcCOM und für die Laufzeitphase (update phase) zu definieren.



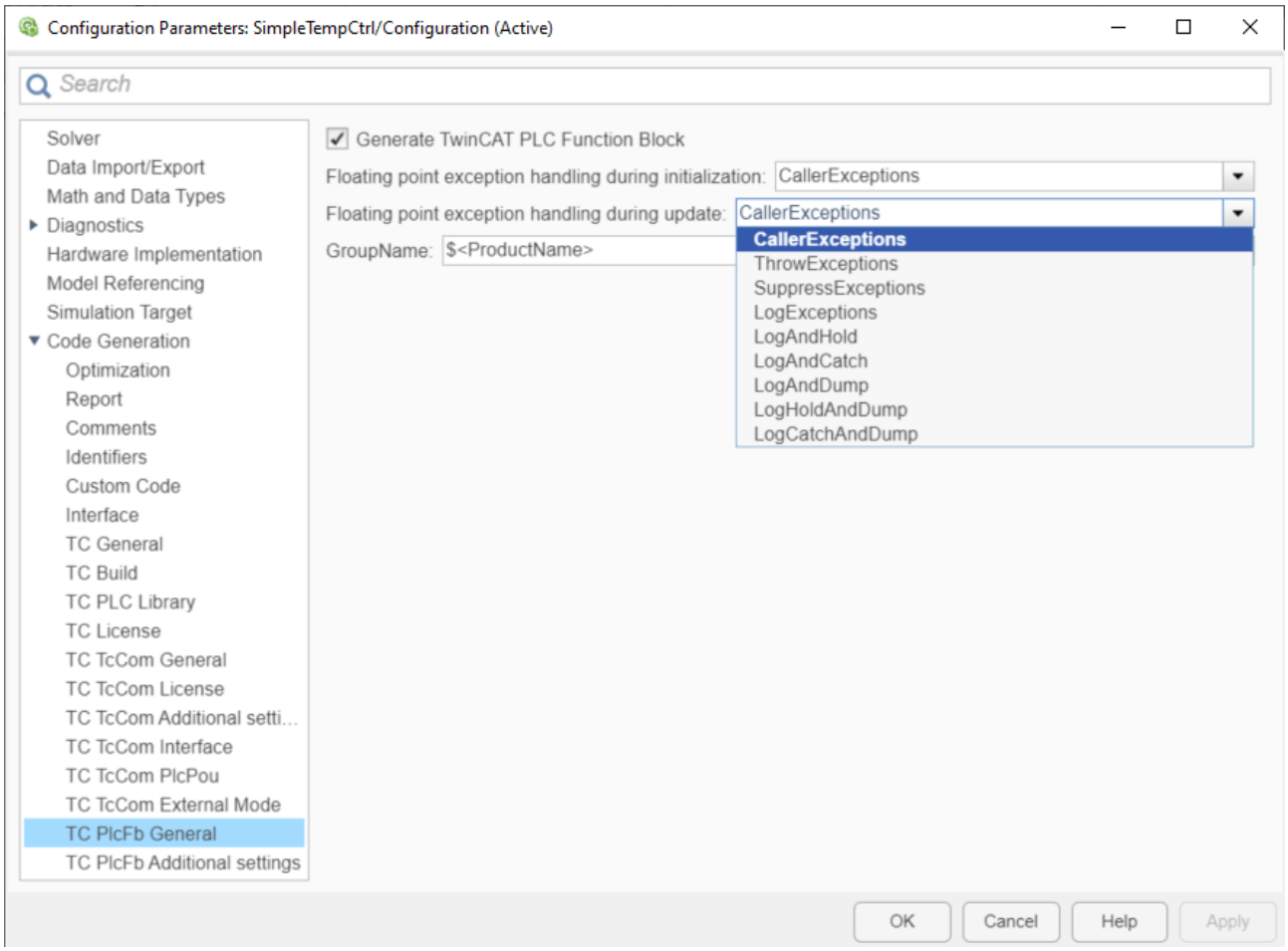
Sollten Sie mit einem bereits kompilierten TcCOM in TwinCAT arbeiten, können Sie auch nachträglich die Einstellungen auf der Objekt-Instanz verändern. Nutzen Sie dazu den Reiter Parameter (init) und wählen Sie **Show hidden Parameters**.



<< Einstellung für PLC-FB >>

Die Einstellungen für den PLC-FB (SPS-Funktionsbaustein FB_<ModelName> in der SPS-Bibliothek) sind unabhängig von den Einstellungen zum TcCOM-Objekt unter **TC PlcFb General** vorzunehmen. Eine nachträgliche Anpassung der Exception-Optionen bei Verwendung des Funktionsbausteins in TwinCAT ist nicht vorgesehen.

Beachten Sie, dass der andere SPS-Funktionsbaustein FB_<ModelName>_TcCOM ein Wrapper für ein TcCOM-Objekt ist und somit bei dessen Verwendung die Exception-Einstellungen aus dem Bereich TcCOM gültig sind.

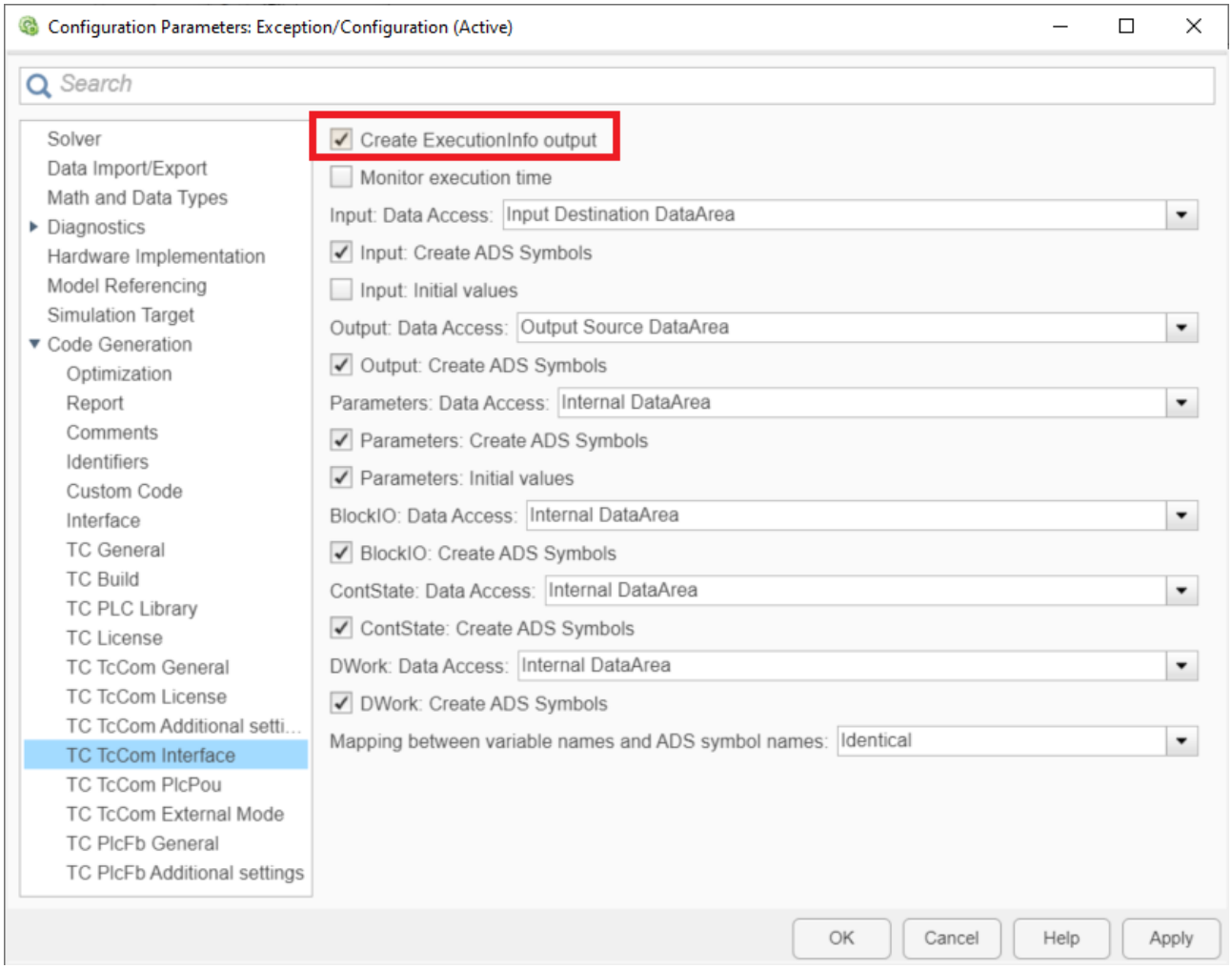


Optionaler ExecutionInfo Output

Werden auf Objektebene Exceptions behandelt, ist es sinnvoll, entsprechende Informationen über aufgetretene Exceptions am Objektausgang zugänglich zu machen. So kann an diesem Ausgang abgefragt werden, ob eine Exception aufgetreten ist, was für eine Exception es war, ob ein Dump File geschrieben wurde etc.

<< Einstellung für TcCOM >>

Für das TcCOM-Objekt können Sie einen zusätzlichen Ausgang „ExecutionInfo“ aktivieren über den Eintrag TC TcCom Interface.



Der Ausgang ExecutionInfo ist eine Struktur mit folgenden Einträgen:

ExecutionInfo Struktur

Eintrag	Datentyp	Bedeutung
CycleCount	ULINT	Aktueller Cycle Count (unabhängig von einer Exception)
ExceptionCount	ULINT	Anzahl der bislang aufgetretenen Exceptions
ActException	TcMgSdk.ExceptionInfo	Näherer Erläuterung zur aktuellen Exception (nur erste Exception im aktuellen Zyklus)

TcMgSdk.ExceptionInfo

Eintrag	Datentyp	Bedeutung
ExceptionCode	DINT	Exception code
TmxName	STRING(127)	Name des tmx-Treibers, der die Exception geworfen hat.
TmxVersion	ARRAY[0..3] OF UDINT	Version des tmx-Treibers, der die Exception geworfen hat.
InstructionAddr	UDINT	Relative Adresse im Speicher; Ort an dem die Exception aufgetreten ist.
ReturnAddr	ARRAY[0..3] OF UDINT	Rücksprungadressen

Eintrag	Datentyp	Bedeutung
DumpCreated	BOOLEAN	TRUE, wenn ein Dump File zur Exception erstellt wurde.

Mit der *InstructionAddr* ist es möglich, zu beurteilen, ob die Exception mit dem angegebenen *ExceptionCode* immer an der gleichen Stelle im Quellcode auftritt. Ist die *InstructionAddr* für wiederholende Exceptions gleich, tritt diese immer an der gleichen Stelle im Code auf. Über die *ReturnAddr* sehen Sie, woher die Aufrufe gekommen sind, die zur Stelle der Exception geführt haben. Sie können also beurteilen, ob der Aufruf, der zur Exception führt, immer denselben Aufrufweg nimmt. Wird der Code von außerhalb des Tmx-Treibers aufgerufen, steht in *ReturnAddr* eine 0.

Exception code	Bedeutung
0xC000008E	Divide by zero
0xC000008F	Inexact result
0xC0000090	Invalid operation

Nutzen Sie den TcCOM-Wrapper-FB [▶ 214], steht Ihnen die ExecutionInfo Struktur am Funktionsbaustein zur Verfügung. Beachten Sie, dass die Einträge entsprechend der TwinCAT Programmierkonventionen Präfixe entsprechend dem Datentyp tragen.

<< Einstellung für PLC-FB >>

Der PLC-FB beinhaltet als Properties immer `nExceptionCount` und `stActiveException` entsprechend obiger Definition. D.h. es muss keine Checkbox separat gesetzt werden, um diese Properties zu erhalten. Einziger, im Vergleich zum TcCOM, nicht vorhandener Parameter ist der Cycle Count, da dieser in der SPS bei bedarf sehr einfach selbst implementiert werden kann.

Ausführungsstopp eines TwinCAT-Objekts behandeln

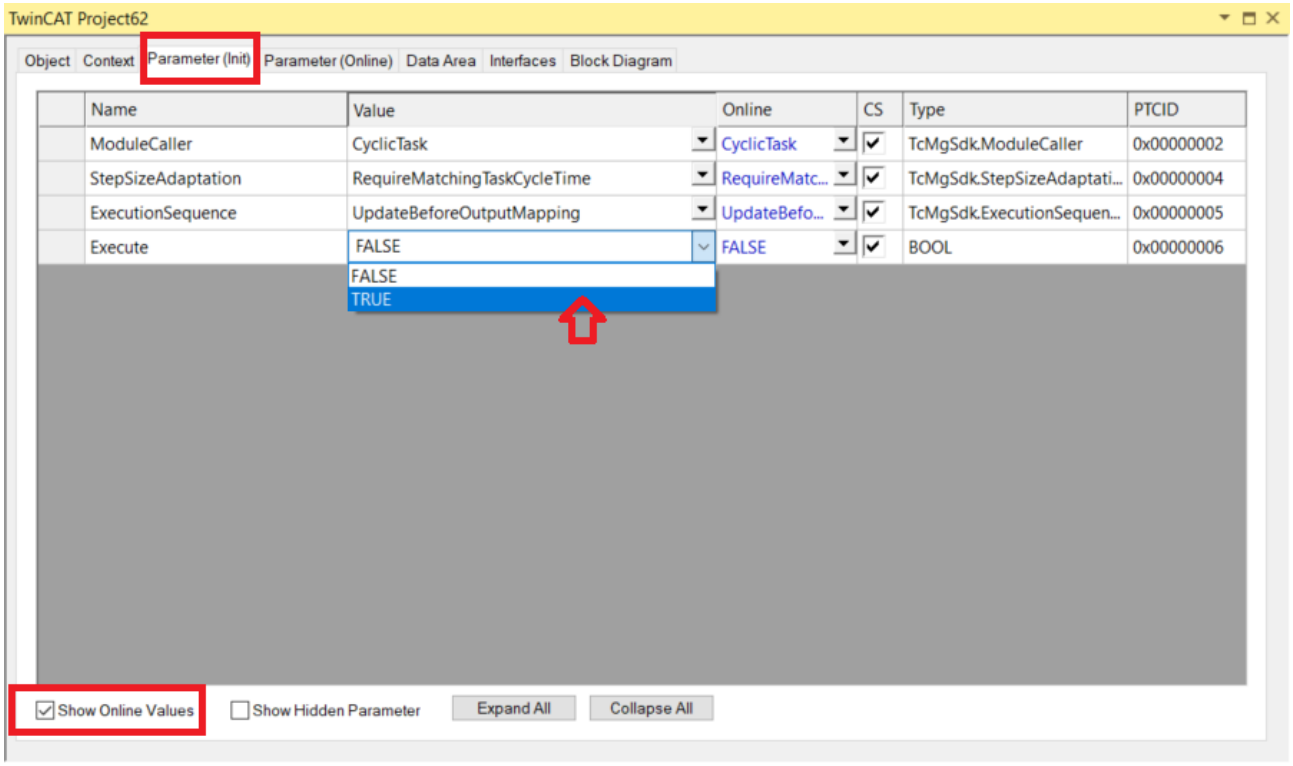
LogAndHold und LogHoldAndDump

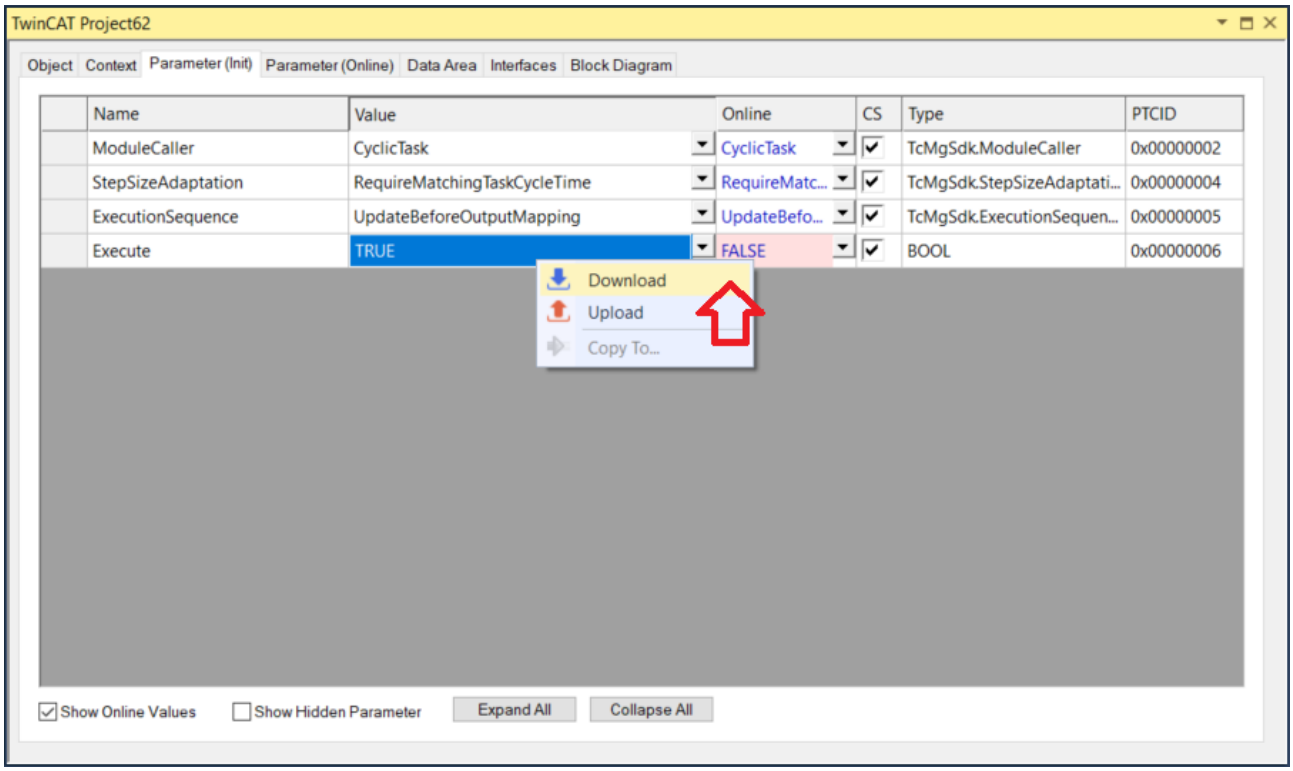
Im Falle einer Exception wird die Ausführung des Codes im betreffenden TcCOM-Objekt oder SPS-Funktionsbaustein (PLC-FB) durch Setzen des Parameters Execute auf FALSE gestoppt.

<< Einstellung für TcCOM >>

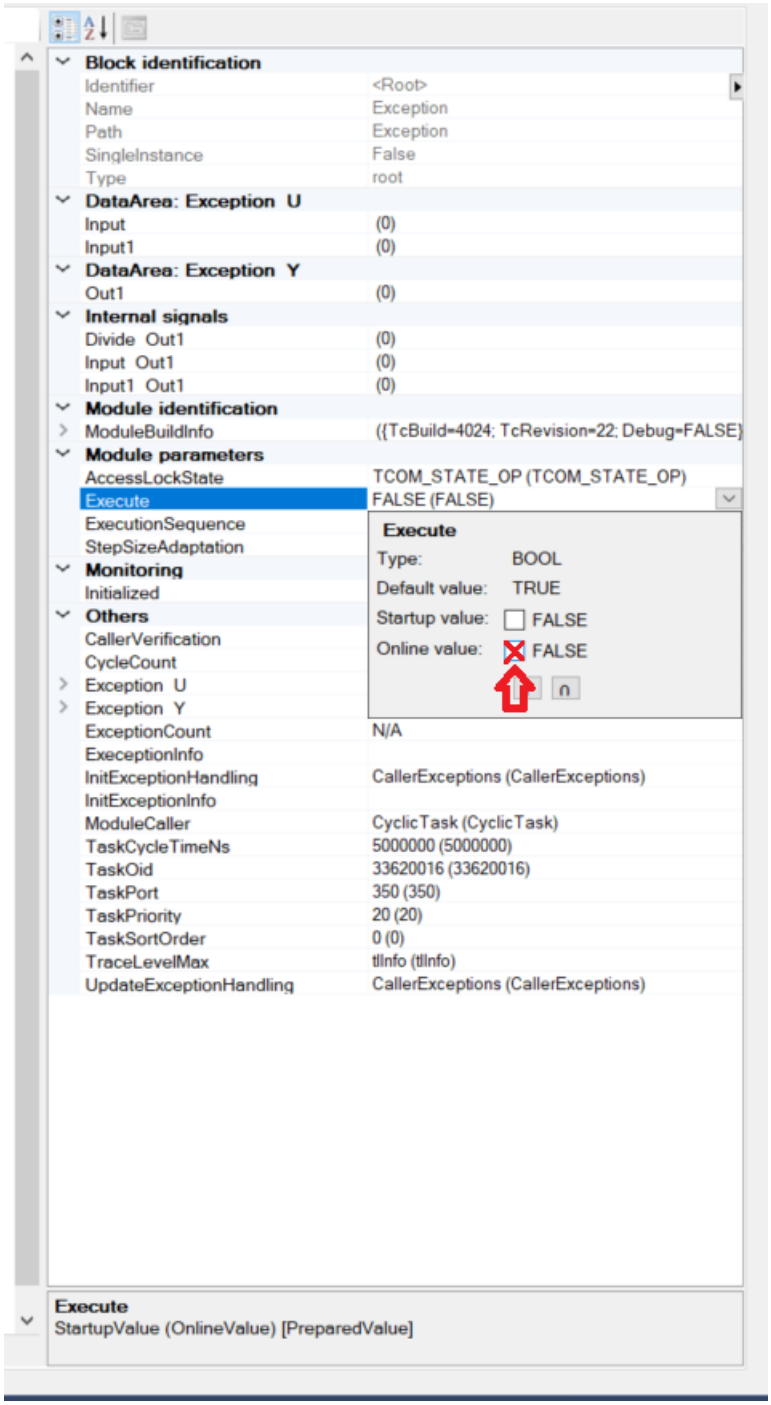
Der Paramater Execute kann aus dem XAE und per ADS gelesen bzw. geschrieben werden.

Im XAE können Sie auf dem TcCOM-Objekt unter Parameter (Init) dessen Online-Werte anzeigen lassen und verändern.

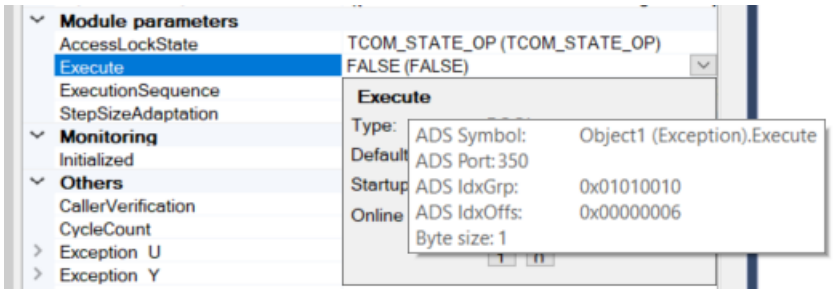




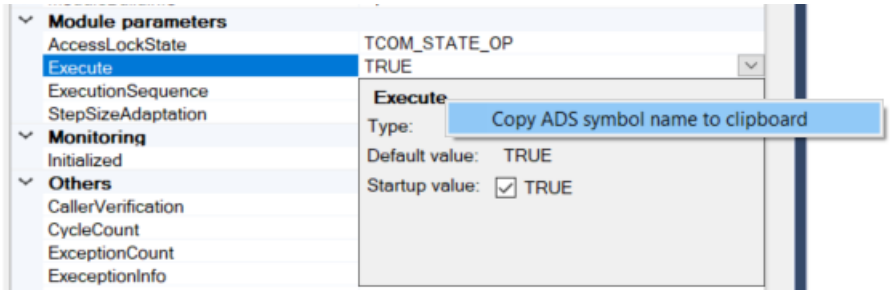
Im Blockdiagramm wird Ihnen der Parameter unter *Module parameters* angeboten.



Wenn Sie die Maus über den Namen Execute im Änderungsdialog führen, wird Ihnen, wie bei allen anderen Parametern auch, die ADS-Adresse des Parameters gezeigt. Damit können Sie den Parameter auch per ADS setzen.



Durch Rechtsklick auf den Namen **Execute** können Sie die ADS-Symbolinformationen auch in der Zwischenablage speichern. Auch dies gilt für alle anderen Parameter.



Nutzen Sie den TcCOM-Wrapper FB, können Sie den Parameter Execute durch Schreiben auf dem Property bExecute verändern.

<< Einstellung für PLC-FB >>

Der Parameter Execute ist am FB als Property `bExecute` mit Lese- und Schreibrechten vorhanden. Nutzen Sie dieses Property, um die Ausführung des FB wieder zu starten.

LogAndCatch und LogCatchAndDump

Zusätzlich zum Parameter *Execute* wechselt, im Fall von LogAndCatch und LogCatchAndDump, auch der Online-Parameter *Initialized* nach FALSE. Das Modul muss neu initialisiert werden, bevor es wieder eine Berechnung durchführen kann. Dies ist notwendig, da interne Zustände, durch den Abbruch der Code-Ausführung an der Stelle der Exception, inkonsistent sein können.

<< Einstellung für TcCOM >>

Eine Reinitialisierung kann nur erfolgen, indem das TcCOM-Objekt in den Zustand „Init“ zurückgefahren und erneut nach OP gefahren wird. Zur Laufzeit können aber nur TcCOM-Objekte runtergefahren werden, welche keine Mappings haben, sonst würden aktive Mappings das Herunterfahren blockieren. Eine neue Initialisierung ist im Fall von aktiven Mappings am TcCOM nur möglich, indem die gesamte TwinCAT-Laufzeit neu gestartet wird. Es wird daher empfohlen den TcCOM-Wrapper-FB [▶ 214] zu nutzen. Dieser kann genutzt werden, um das TcCOM aus des SPS aufzurufen und benötigt keine Mappings um auf dessen Eingänge und Ausgänge zuzugreifen. Entsprechend kann das TcCOM-Objekt auch während der Laufzeit neu initialisiert werden.

Property-Einstellungen für den TcCOM-Wrapper-FB

Im Folgenden Beispiel-Code werden Properties, z.B. bExecute, gelesen. Erzeugen Sie den TcCOM-Wrapper-FB mit gesetzten TcCom Wrapper FB Properties und mit der Option “CyclicUdate“ für die Properties, damit untenstehender Code zum Wrapper passt.

```
PROGRAM MAIN
VAR
   stInitTemp : ST_FB_SimpleTempCtrl_TcCOM_InitStruct := (nOid := 16#01010010);
   fbTempCtr  : FB_SimpleTempCtrl_TcCOM_InitStruct(stInitTemp);
   Inputs     : ST_ExtU_SimpleTempCtrl_T;
   Outputs    : ST_ExtY_SimpleTempCtrl_T;
   ExecutionOut : ST_ExecutionInfo2;
END_VAR
```

```
// check if TcCOM is in OP mode and all set
IF fbTempCtr.bExecute = TRUE AND fbTempCtr.bInitialized = TRUE AND fbTempCtr.nObjectState = TCOM_STA
TE.TCOM_STATE_OP THEN

    // call the module
    fbTempCtr(stSimpleTempCtrl_U := Inputs, stSimpleTempCtrl_Y => Outputs, stExecutionInfo => Execut
ionOut);

    // handle exceptions
    IF ExecutionOut.ActException.ExceptionCode <> 0 THEN
       // collect exception information
       (* ..... *)

       // reinit TcCOM
       fbTempCtr.Reinit(stReInit := stInitTemp);
    END_IF

 END_IF
```

Beachten Sie, dass die ReInit-Methode synchron ausgeführt wird, d. h. je nach Zykluszeit und benötigter Zeit zum neu initialisieren, kann es zu Zyklusüberschreitungen kommen.

<< Einstellung für PLC-FB >>

Zur Prüfung, ob das hinterlegte Modul nicht (mehr) initialisiert ist, können Sie das Property `bInitialized` am FB nutzen. Sie haben hier nur lesenden Zugriff. Eine Reinitialisierung ist derzeit nicht über eine Methode am FB möglich. Es muss die SPS-Laufzeit, alternativ die gesamte TwinCAT-Laufzeit, neu gestartet werden.

Dump-Files

Das Schreiben des Dump-Files kann einige Zyklen in Anspruch nehmen. Am besten sollte für das betreffende TcCOM-Objekt oder den PLC-FB eine separate Task verwendet werden, die keine wichtigen Tasks blockiert.

Dump-Files werden nur mit einer TwinCAT XAR Version >= 3.1.4024.22 geschrieben, ansonsten erhält man eine entsprechende Warnung.

Im Fall von *LogAndDump* wird die Ausführung des Codes nach Auftreten einer Exception zyklisch fortgeführt, es können entsprechend zyklisch Exceptions auftreten, die zu andauernden Zykluszeitüberschreitungen führen könnten. Daher wird der Online-Wert des Parameters *UpdateExceptionHandling* nach dem Schreiben des Dump-Files auf *LogExceptions* gesetzt, d. h. das Schreiben von Dump-Files wird deaktiviert, kann anschließend aber z. B. per ADS oder Eingriff über das XAE unter Parameter (Init) wieder eingeschaltet werden.

Das erstellte Dump-File wird auf dem Laufzeit-PC im Boot-Ordner abgelegt und kann von dort zur Analyse auf einen anderen PC kopiert werden. Bei Verwendung einer TwinCAT Version kleiner 3.1.4024.x können Sie die Dump-Files mit WinDbg öffnen und Ihre Analyse starten.

4.8.6 Verwenden von Realtime Monitor Zeitmarken

MATLAB®-Kommandos, wie tic und toc, sind beliebte Mittel, um die Performance von Code-Abschnitten in MATLAB® zu analysieren. In der TwinCAT-Laufzeit sind diese Kommandos in dieser Form nicht nutzbar.

TwinCAT stellt für diesen Zweck den sogenannten TwinCAT Realtime Monitor zur Verfügung, welcher Zeitmarken im Quellcode auswertet und zur Analyse darstellt. Das Setzen von Realtime Monitor Zeitmarken wird in MATLAB® Code unterstützt, d. h. die Zeitmarken werden in MATLAB® gesetzt und werden nach Code-Generierung und Instanziierung in TwinCAT über den Realtime Monitor auswertbar. Ausführung der Zeitmarken in MATLAB® führt zu Ausgaben in der MATLAB® Konsole.

Klasse: TwinCAT.ModuleGenerator.Realtime.LogMark

Methoden: Start, Stop und Mark

MATLAB®-Dokumentation: `doc("TwinCAT.ModuleGenerator.Realtime.LogMark")`

Beispiel in MATLAB®

```
TwinCAT.ModuleGenerator.Samples.Start("BaseStatisticsLogMark")
```

Die Nutzung der Zeitmarken ist auf MATLAB® Code beschränkt und erfordert für die Verwendung in Simulink® eine entsprechende Einbettung in MATLAB® Function blocks.

4.9 FAQ

4.9.1 Modell-Parameter zur Laufzeit verändern

Kann ich Modell-Parameter während der Laufzeit in TwinCAT verändern?

Ja, beachten Sie dabei folgende Einstellungen:

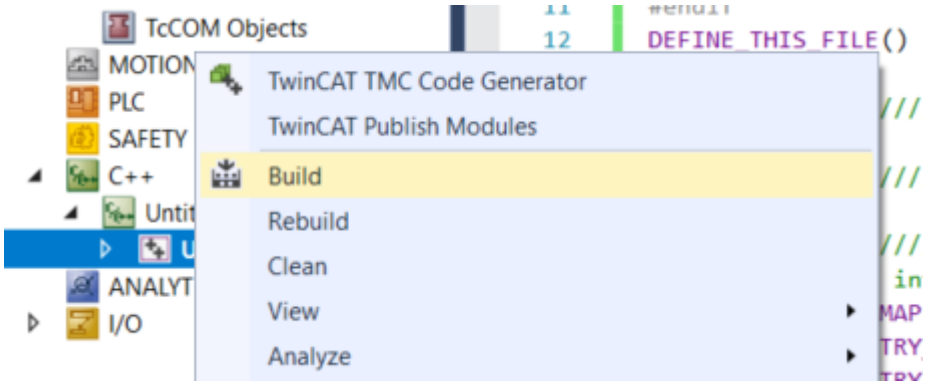
- Optimization > Default parameter behavior: Tunable
  Wenn der Parameter so gesetzt ist, werden Modell-Parameter zur Laufzeit einstellbar. Siehe auch Parametrierung einer Modulinstanz [▶ 191].
- Interface > Code interface packaging
  Sie haben hier die Optionen „Nonreusable function“, „Reusable function“ und „C++ Class“. Die Einstellungen haben Einfluss darauf, ob Sie mehrere Instanzen eines TcCOM in TwinCAT instanziieren können und ob Sie diese dann auch deren Modell-Parameter individuell oder abhängig voneinander einstellen können. Siehe auch Parametrierung mehrerer Modulinstanzen [▶ 196].

4.9.2 Build eines Sample schlägt fehl

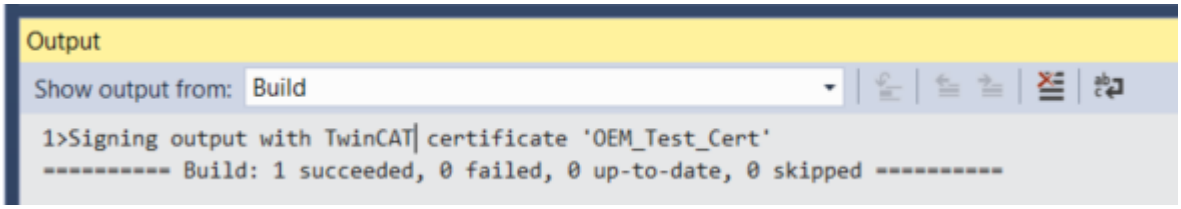
Alle mitgelieferten Beispiele (Liste durch `TwinCAT.ModuleGenerator.Samples.List` im MATLAB® Command Window) sind bei Beckhoff Automation durch Tests geprüft worden. Sollte ein Build eines Samples dennoch nicht erfolgreich durchlaufen, liegt die Vermutung nahe, dass bei der Einrichtung auf Ihrem Engineering PC etwas angepasst werden muss.

✓ Um das Plattformtoolset ohne den Einfluss von MATLAB® zu testen, erstellen Sie bitte in TwinCAT (öffnen Sie TwinCAT in Visual Studio) ein TwinCAT Versioned C++-Projekt .

1. Rechtsklicken Sie **Add New Item** auf C++ Tree Item.
2. Wählen Sie dann **TwinCAT Module Class with Cyclic Caller**.
   ⇨ Es erscheint ein C++-Projekt im TwinCAT Tree unter C++.
3. Bauen Sie das C++-Projekt und betrachten Sie das Output Window in TwinCAT.



⇨ Das Output-Fenster sollte „1 suceeded“ für den Build-Prozess liefern. Wenn dies nicht der Fall ist, prüfen Sie, ob Sie die Option **Desktop development with C++** im Visual Studio installiert haben.



4.9.3 Probleme bei der Blockdiagramm-Darstellung im TwinCAT XAE

Mit dem TwinCAT Target for Simulink® Version 2.x.xxxx.x und höher erstellte TcCOM-Module benötigen eine TC3 BlockDiagram-Version 1.4.1419.0 und höher zur korrekten Anzeige im TwinCAT XAE.

Wo finde ich die Version des TC3 BlockDiagram?

- Unter Programme und Feature in der **Systemsteuerung > Beckhoff TwinCAT 3 BlockDiagram**.
- Im Blockdiagram im TwinCAT XAE > Rechtsklick im Fenster > **About TC3 BlockDiagram.**

Wenn Ihre TwinCAT XAE-Installation eine frühere Version des TC3 BlockDiagram beinhaltet:

- Können Sie das *TwinCAT Tools for MATLAB and Simulink* Setup installieren. Dieses beinhaltet eine neue TC3 BlockDiagram Version.

- Können Sie sich beim Beckhoff Support melden, um ein separates TC3 BlockDiagram Setup zu erfragen.

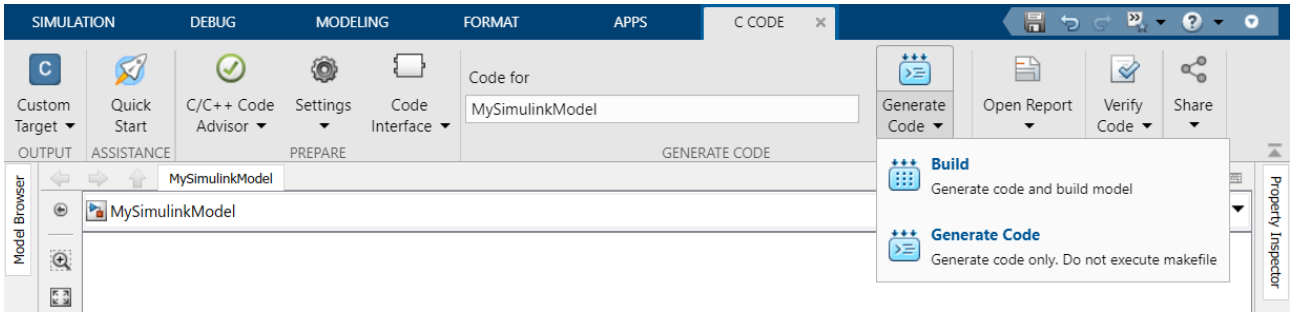
4.9.4 Kann ich TE1400 Version 1.2.x und Version 2.x gleichzeitig verwenden?

Ja, das ist möglich. Installieren Sie einfach beide Produkte auf Ihrem System und wählen Sie das entsprechende Target aus, um zwischen den beiden Versionen zu differenzieren.

TwinCatGrt.tlc für Version 2.x und *TwinCAT.tlc* für 1.2.x

4.9.5 Was ist der Unterschied zwischen "Build" und "Generate code"?

In der Simulink Coder™ App können Sie zwischen „Build“ und „Generate Code“ wählen:



Wenn Sie als Target *TwinCatGrt.tlc* eingestellt haben, haben beide Optionen dieselbe Funktion, da das *TwinCatGrt.tlc* nicht über das makefile von MathWorks® läuft.

Wenn Sie den Build-Prozess nicht ausführen, sondern nur den C++-Code generieren möchten, deaktivieren Sie unter TC Build die Checkbox **Run the publish step after project generation**.

Was ist der Unterschied zwischen „Build“ und „Publish“?

Unter „Publish“ wird das sukzessive Ausführen des Build-Prozesses für bestimmte TwinCAT-Plattformen verstanden. Aus Simulink® heraus werden dann für die unter TC Build aktivierten Plattformen die entsprechenden Binaries nacheinander erstellt, sodass im Nachgang entscheiden werden kann, für welche Zielplattform die kompilierten Funktionen genutzt werden sollen.

4.9.6 Ich kann in TwinCAT die Paramater eines Modules nicht verändern

Als Standardwert für den Parameter **Default parameter behavior** ist *im TwinCarGrt.tlc* „Inlined“ eingestellt. Verändern Sie diesen auf „Tunable“ oder konfigurieren Sie, welche Parameter als „Tunable“ markiert werden sollen über den Button **configure**…

4.9.7 Mapping geht verloren bei Reload TMI/TMC

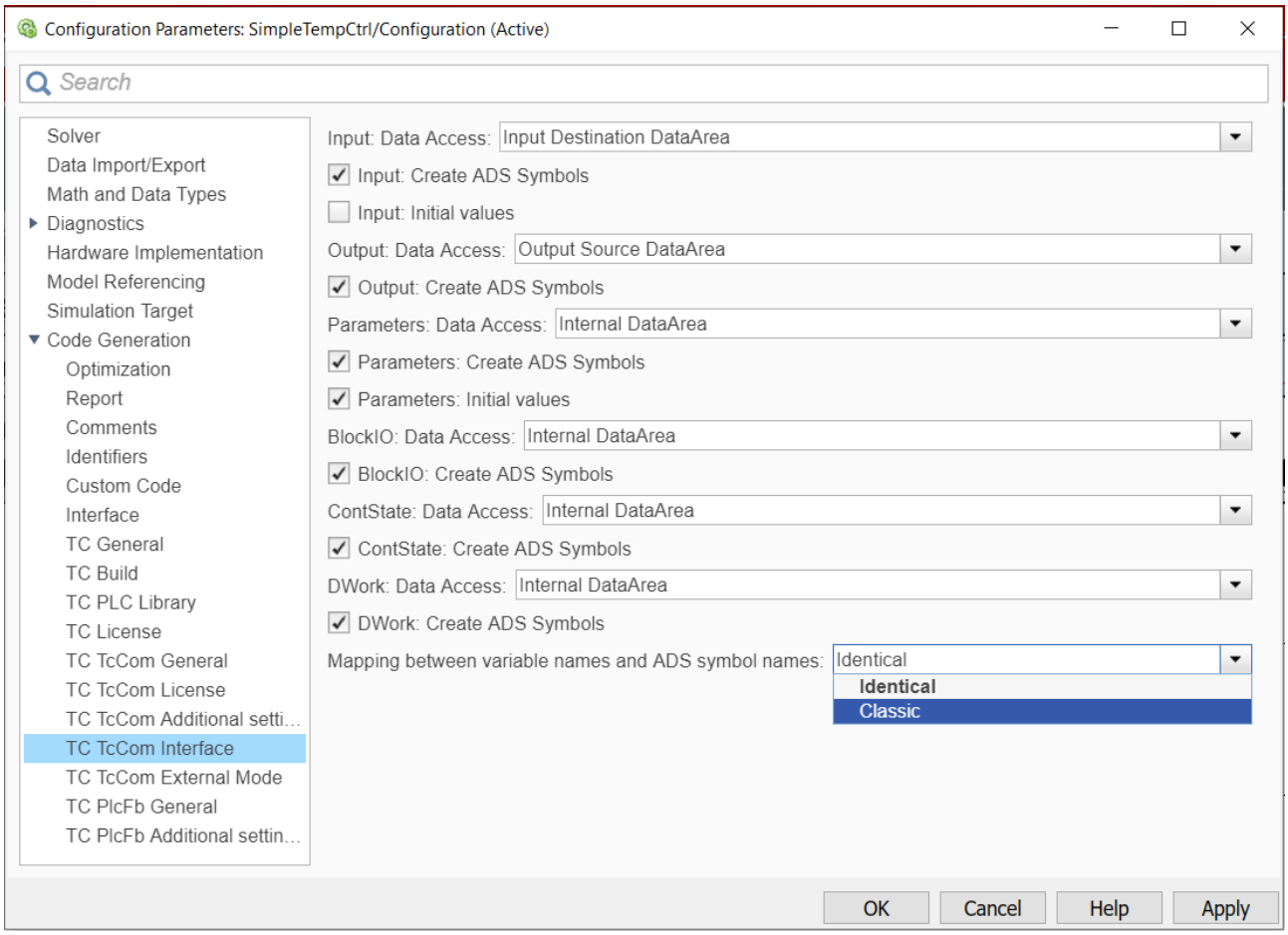
Herausforderung:

Sie haben bereits TcCOM-Objekte in Ihrer TwinCAT Solution, welche Sie mit dem Target for Simulink® Version 1.2.xxxx.x erstellt haben. Sie möchten nun mit dem Target for Simulink® Version 2.x.x.x ein neues TcCOM-Objekt erstellen und das neu erstellte TcCOM mit Reload TMI/TMC File… in Ihrer bestehenden TwinCAT Solution ersetzen.

In den Default-Einstellungen verlieren Sie durch dieses Vorgehen die Mapping-Informationen.

Lösung:

Stellen Sie unter TC TcCOM Interface das „mapping between variable names and ADS Symbol names” auf “Classic” und erstellen Sie damit das neue TcCOM-Objekt.



Dadurch bliebt das Mapping erhalten, wenn Sie nun in TwinCAT mit Reload TMI/TMC File… ihr altes TcCOM-Objekt ersetzen.

4.9.8 Einbinden des Blockdiagramm-Controls in .NET

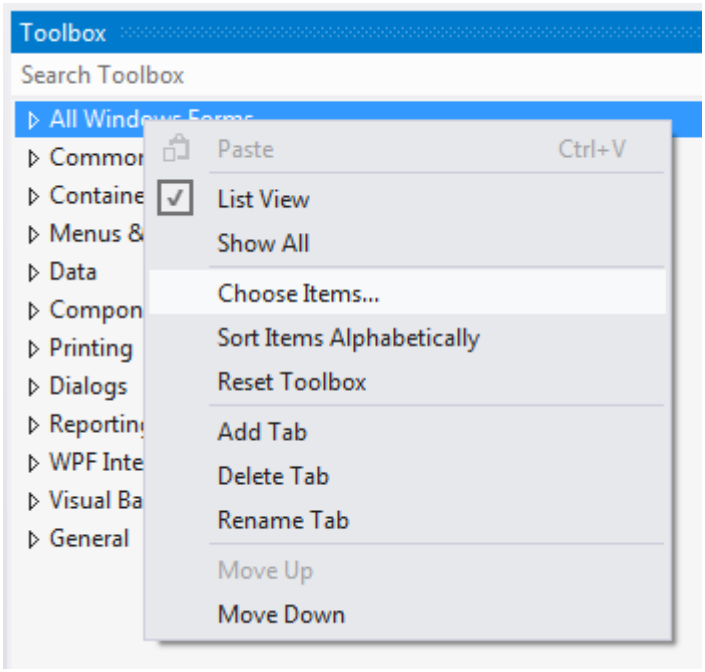
Das Control, welches das Blockdiagramm in der TwinCAT XAE Umgebung darstellt, kann auch als Control in eigene Visualisierungen eingebunden werden.

Ein Beispielprogramm kann hier heruntergeladen werden: https://infosys.beckhoff.com/content/1031/te1400_tc3_target_Matlab/Resources/11697311755.zip

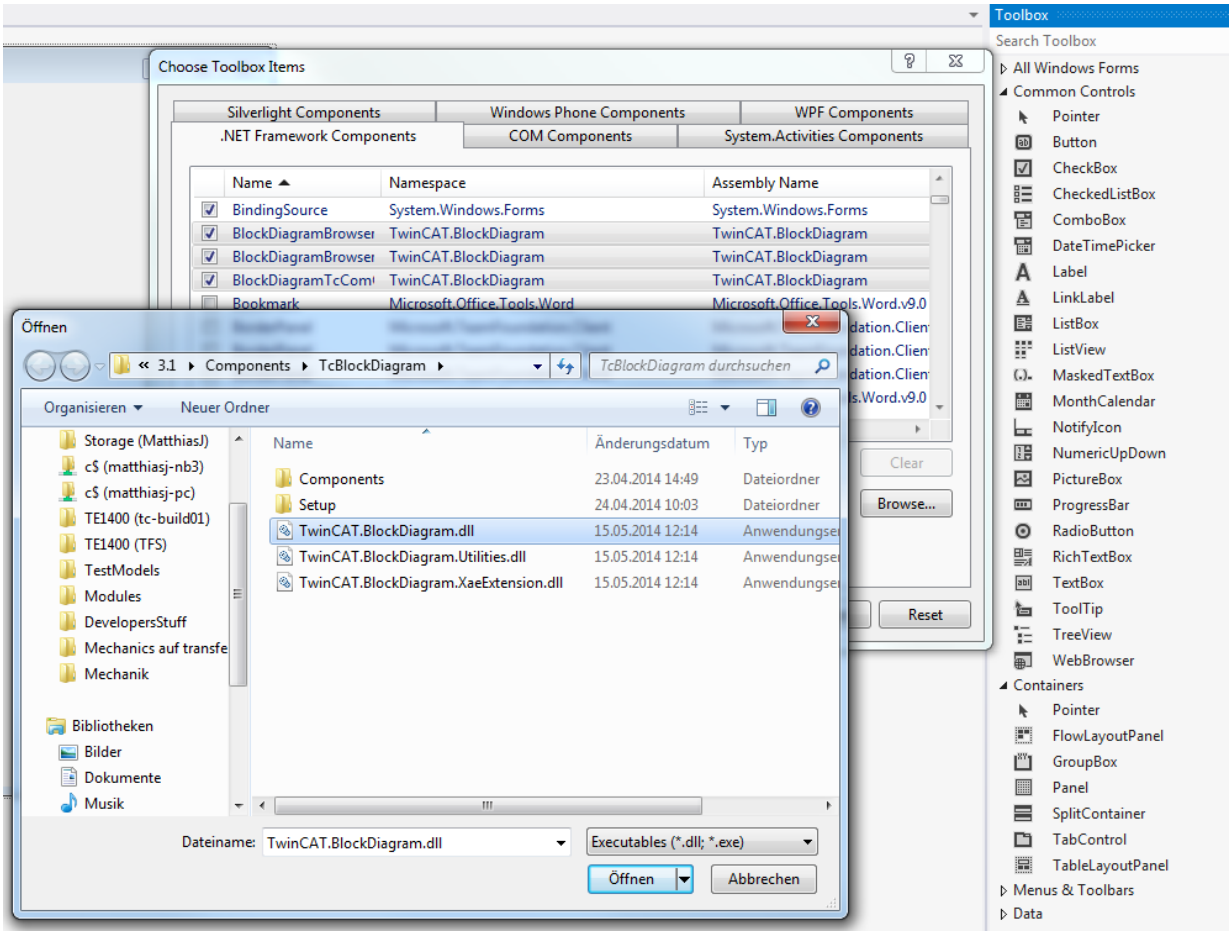
✓ Folgende Schritte sind notwendig:

1. Erstellen Sie eine Windows-Forms-Applikation.
2. Fügen Sie TwinCAT.BlockDiagram.dll zur Toolbox hinzu.

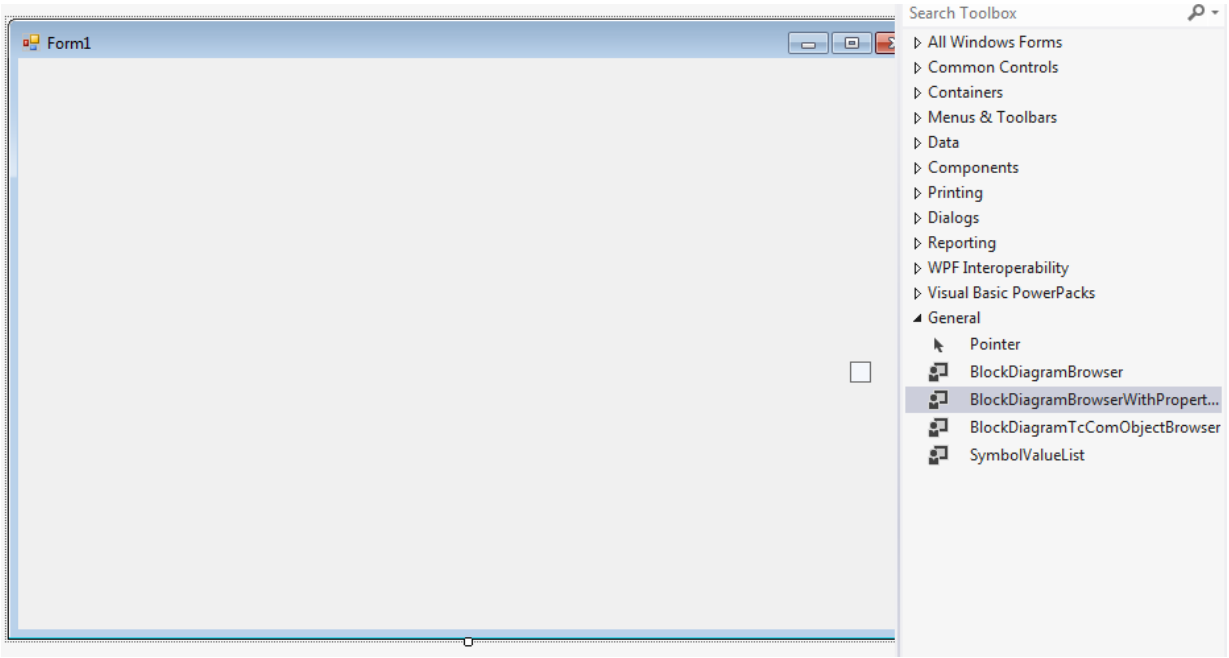
3. Wählen Sie dafür im Kontextmenü den Eintrag **Choose Items…** aus.



4. Browsen Sie zur TwinCAT.Blockdiagram.dll, welche sich unter *<TwinCAT-Installationspfad>\3.1\Components\TcBlockDiagram* befindet.



5. Fügen Sie eine TcBlockdiagram-Control-Instanz zum Windows-Forms-Object per Drag-and-Drop hinzu (BlockDiagramBrowser bzw. BlockDiagramTcComObjectbrowser).



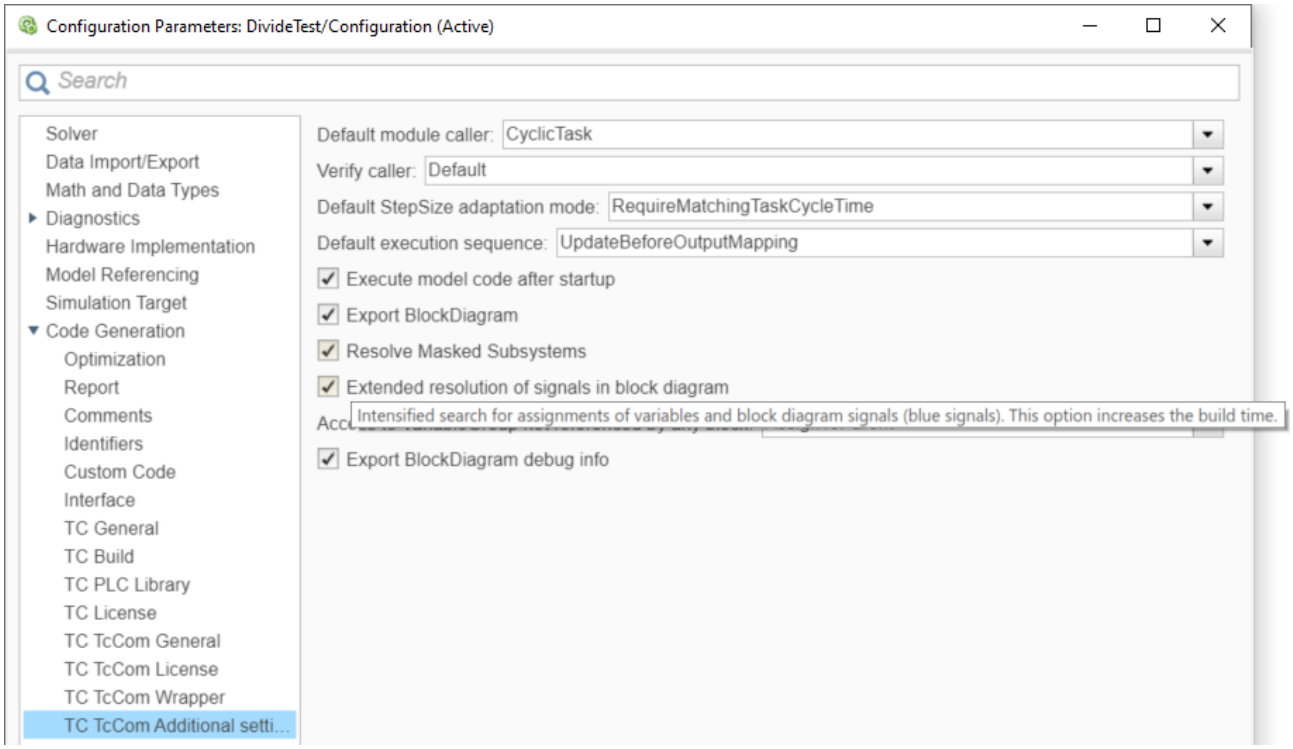
4.9.9 Beobachtbare Signale im TwinCAT Blockdiagramm

Welche Einstellungen beeinflussen die Anzahl der „blauen Signale“ im TwinCAT Blockdiagramm?

Grundsätzlich benötigt das TwinCAT Target for Simulink® eine Variable im C/C++-Code, welche dem Signal im Blockdiagramm zugeordnet werden kann. Ist ein Signal im Blockdiagramm nicht blau gekennzeichnet, bedeutet das, dass entweder keine Variable existiert (da diese bei der Code-Generierung durch Optimierung des Codes weggefallen ist) oder die Variable nicht zugeordnet werden konnte.

Um ein Wegfallen der Variablen durch Code-Optimierung zu unterdrücken, können Sie **Test Points** in Simulink® nutzen: Siehe *Configure Signals as Test Points* in der Simulink® Dokumentation.

Unter **TC TcCom Additional settings** können Sie zudem den Eintrag **Extended resolution of signals in block diagram** setzen. Ist dieser Eintrag aktiv, wird eine intensivierte Suche nach Zuordnungen von Variablen und Signalen durchgeführt. Beachten Sie, dass diese Suche Zeit beansprucht, sodass der Export des Blockdiagramms länger dauert.



4.9.10 Verwenden von Simulink® Strings

Simulink® Strings sind ausdrücklich erlaubt und können mit dem TwinCAT Target for Simulink® genutzt werden.

Einschränkung

Je nach MATLAB® Release Version, Code-Interface-Packaging-Einstellung sowie gesetztem C/C++-Standard, übersetzt der Simulink Coder™ einen Simulink® String in den Datentyp `std::string`. Wird dieser Simulink® String dann als Modell *Eingang* oder Modell *Ausgang* genutzt, ist zu beachten, dass diese Einträge nicht durch Mapping mit anderen Objekten in TwinCAT verbunden werden können. Das Mapping in TwinCAT geht von einer statischen Datentypgröße aus, was bei `std::string` nicht der Fall ist.

Handhabung in TwinCAT

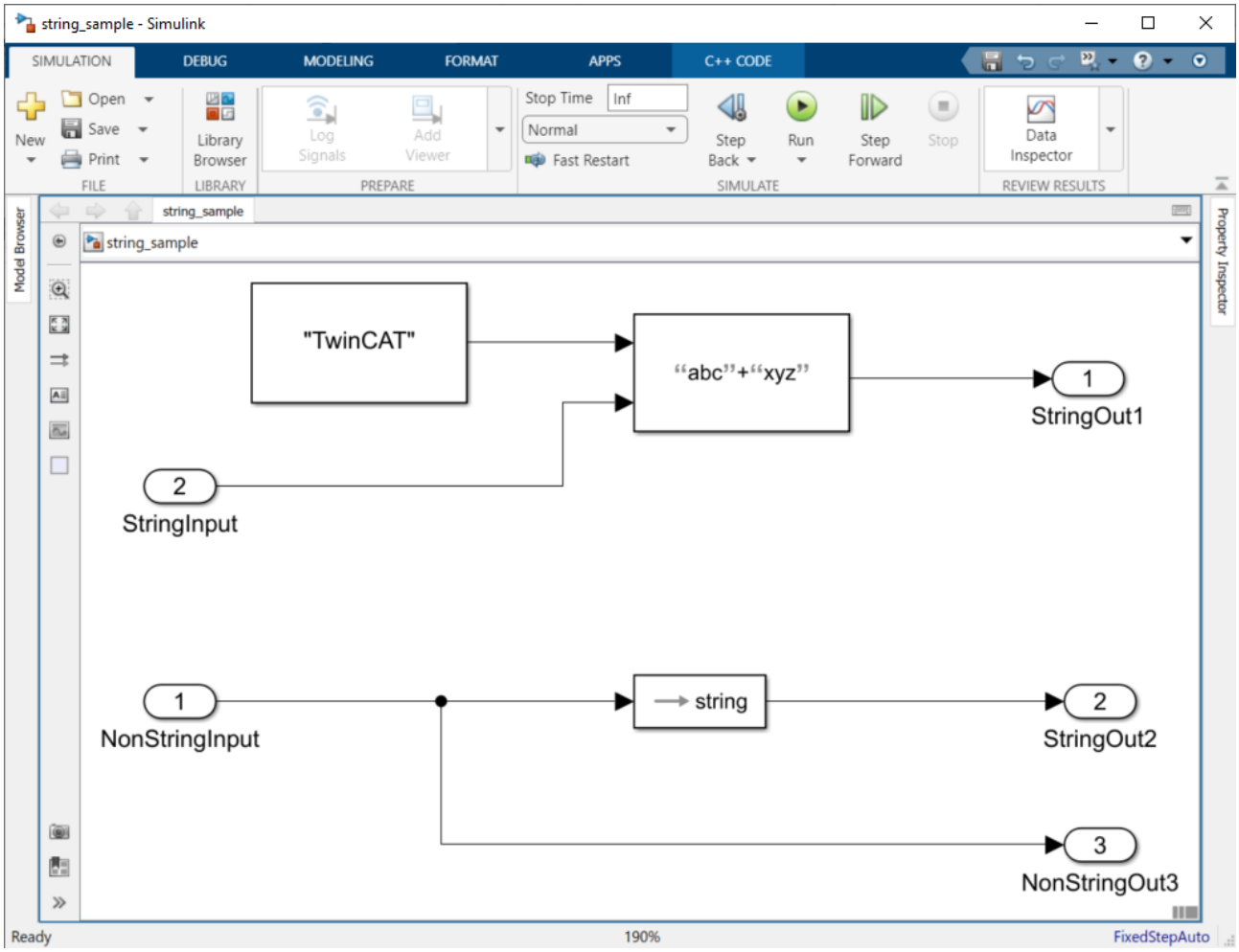
Damit Sie dennoch die Ein- und Ausgänge nutzen können, wird die Verwendung des PLC-FB (keine Mappings notwendig) oder des TcCOM-Wrapper-FB empfohlen. Bei den Standard Ein- und Ausgängen der FBs werden in beiden Fällen die Simulink®-String-Einträge nicht angezeigt. Diese sind über Getter und Setter Methoden am FB separat zu setzen.

Simulink® Bus mit Simulink® Strings: Eingeschränkte Verwendung

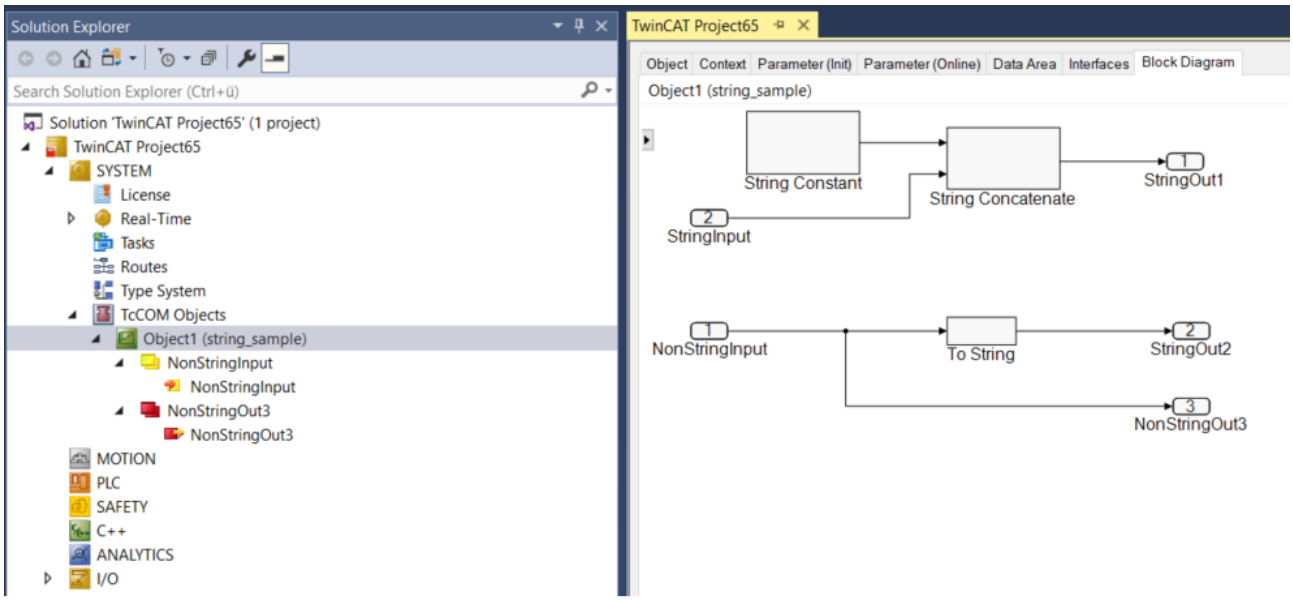
Folgende Situation wird aktuell nicht unterstützt: Ein Simulink® String kann nicht in einem Simulink® Bus verwendet werden, der als Ein- oder Ausgang des Modells dient, falls dieser als `std::string` vom Simulink Coder™ abgebildet wird.

Beispiel

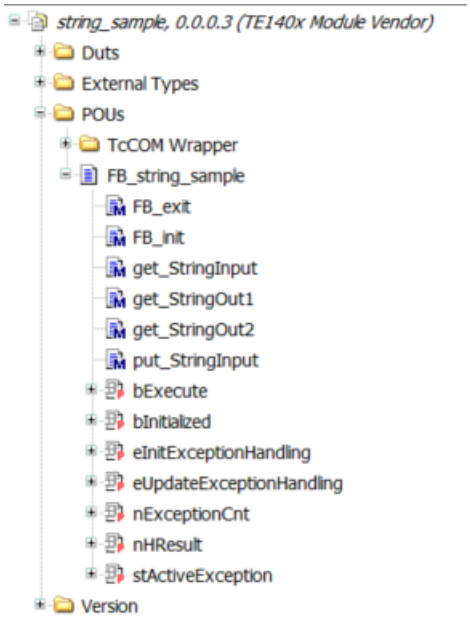
Folgendes Simulink®-Modell sei gegeben mit einer Mischung aus String und Nicht-String-Ein- und -Ausgängen.

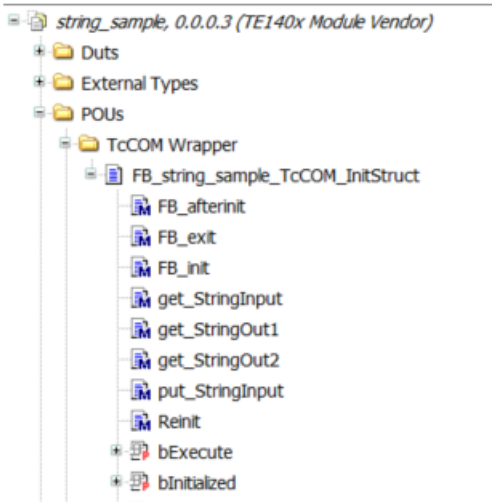


Wenn dieses Modell mit MATLAB R2022a und Code Interface Packaging „C++ Class“ übersetzt wird, wird der Datentyp `std::string` vom Simulink Coder™ erzeugt. In folgender Abbildung ist zu erkennen, dass in TwinCAT die String-Ein- und -Ausgänge **nicht** im Prozessabbild vorhanden sind.
Nur die Nicht-String-Ein- und -Ausgänge sind im Prozessabbild vorhanden.



Um die Strings schreiben und lesen zu können, muss entweder der TcCOM Wrapper FB oder der PLC FB verwendet werden. An den Funktionsbausteinen werden automatisch entsprechende Getter und Setter Methoden erstellt.





Beispielcode unter Verwendung des PLC-FB:

```
VAR
   fbStringSample : FB_string_sample;

   myStringIn   : T_MaxString;
   myStringOut1 : T_MaxString;
   myStringOut2 : T_MaxString;
   nSize : ULINT;
   nSize2: ULINT;
   fIn   : LREAL;
   fOut  : LREAL;
END_VAR
```

```
// put sting input
fbStringSample.put_StringInput(c_str := ADR(myStringIn));

// call function
fbStringSample(fNonStringInput := fIn, fNonStringOut3 => fOut);

// get string outputs
nSize := SIZEOF(myStringOut1);
fbStringSample.get_StringOut1(c_str := ADR(myStringOut1), size := nSize);  // size in VAR IN OUT!

nSize2 := SIZEOF(myStringOut2);
fbStringSample.get_StringOut2(c_str := ADR(myStringOut2), size := nSize2);
```

4.9.11 Gibt es Limitierungen hinsichtlich der Ausführung von Modulen in Echtzeit?

Nicht alle Zugriffe, die in Simulink® unter Nicht-Echtzeit-Bedingungen möglich sind, können in der TwinCAT-Echtzeit-Umgebung durchgeführt werden. Im Folgenden werden bekannte Limitierungen beschrieben:

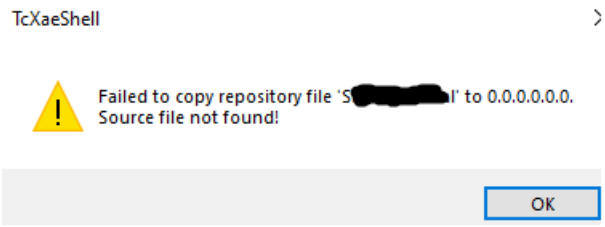
- **Direkter Dateizugriff**: Aus der TwinCAT-Runtime ist kein direkter Zugriff auf das Dateisystem des IPC realisierbar. Zum Schreiben von .mat-Dateien nutzen Sie bitte den TwinCAT-File-Writer-Block anstatt den ToFile-Block von Simulink®.
- **Direkter Hardware-Zugriff**: Ein direkter Zugriff auf Geräte/Schnittstellen setzt einen entsprechenden Treiber voraus, z. B. RS232, USB, Netzwerkkarte, …
  Aus dem Echtzeitkontext kann nicht auf die Gerätetreiber des Betriebssystems zurückgegriffen werden. Zum Beispiel ist es daher nicht einfach möglich, mit der Instrument Control Toolbox™ eine RS232-Kommunikation für den Nicht-Echtzeit-Betrieb herzustellen und diese dann direkt in der TwinCAT-Runtime zu nutzen. Zur Anbindung von externen Geräten kann aber seitens TwinCAT auf eine Vielzahl von Kommunikationsmöglichkeiten zurückgegriffen werden, siehe TwinCAT 3 Connectivity TF6xxx.

- **Zugriff auf die Betriebssystem-API**: Es ist nicht möglich, aus der TwinCAT-Runtime die API des Betriebssystems direkt zu nutzen; ein Beispiel ist die Einbindung der *windows.h* in C/C++-Code. Diese wird bspw. durch den Simulink Coder™ bei Verwendung der FFTW-Implementierung (aber nicht bei der Radix-2-Implementierung) des FFT-Blocks aus der DSP Systems Toolbox™ eingebunden.
- **Precompiled libraries**: Es ist möglich, dass bei der Code-Generierung durch den Simulink Coder™ kein plattformunabhängiger C/C++-Code erzeugt wird, sondern vorkompilierte Bibliotheken eingebunden werden. In diesen Fällen ist keine Echtzeitausführung in TwinCAT möglich.

4.9.12 Meldung: Failed to copy repository

Nach Aktivierung der TwinCAT Configuration erscheint folgende Fehlermeldung im TwinCAT XAE?

Failed to copy repository file …



Die Ursache ist in der Regel, dass in Ihrem Engineering Repository keine oder nicht alle Dateien, insbesondere die TMX-Treiber, gefunden wurden. Das TwinCAT XAE versucht, den Treiber auf das XAR-System zu kopieren, findet aber nicht die korrekte Datei (kontrollieren Sie, ob Sie einen Treiber für die Zielplattform des ausgewählten XAR-Systems im Engineering Repository haben).

Folgefehler im Error Dialog im XAE ist beispielsweise:

'TCOM Server' (10): Error loading repository driver 'C:\TwinCAT\3.1\Boot\Repository\<model vendor>\<model name>\<version>\<tmx-name>' - hr = 0xc0000225

Da die TMX-Datei nicht auf das XAR-System kopiert werden konnte, konnte es nicht geladen werden. Weiterer Folgefehler ist dann ein Link-Error „Could not link external function“.

✓ **Informationen und Abhilfe zu diesem Fehlerbild:**

1. Was ist das Engineering Repository und wo finde ich es?

   ⇨ Siehe automatisch erstellte Dateien [▶ 124].

2. Wie stelle ich sicher, dass beim Kopieren von kompilierten Modellen auf andere XAE-Systeme immer alle notwendigen Dateien übertragen werden und die Ordnerstruktur korrekt bleibt?

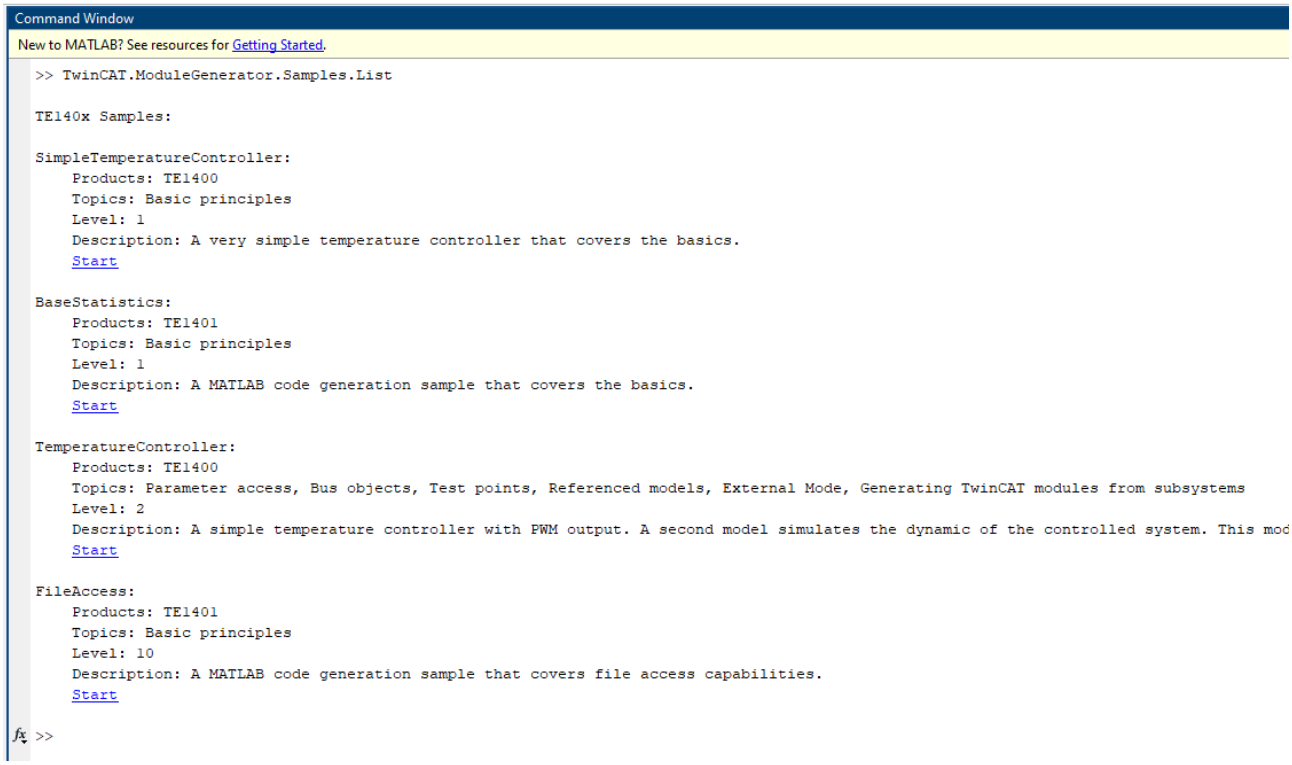
   ⇨ Siehe TMX-Archive [▶ 139].

4.10 Beispiele

Von Beckhoff Automation bereitgestellte Beispiele werden mit dem *TwinCAT Tools for MATLAB® and Simulink®* Setup auf Ihrem System installiert.

Sie können sich mit folgendem Befehl alle verfügbaren Samples ausgeben lassen:

```
TwinCAT.ModuleGenerator.Samples.List
```



Sie können die Samples durch Klicken auf den blauen Start-Link aufrufen. Dazu wird der Sample-Code dann in Ihr User-Verzeichnis kopiert, sodass Sie das Original-Sample nicht verändern. Sie können entsprechend mit der Kopie des Samples arbeiten und ausprobieren.

Ebenso verfügbar sind zum Anzeigen und Starten einzelner Samples:

```
TwinCAT.ModuleGenerator.Samples.Show(SampleName)
```

```
TwinCAT.ModuleGenerator.Samples.Start(SampleName)
```

Das Argument SampleName ist als String zu übergeben, z. B.:

```
TwinCAT.ModuleGenerator.Samples.Start('SimpleTemperatureController')
```

Leerzeichen im SampleName sind im Argument mit Underline zu ersetzen, z. B. „Combine Modules“ -> TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules').

4.10.1 TwinCAT Automation Interface: Verwendung in MATLAB®

Kurzbeschreibung des Automation Interface

Mit dem TwinCAT Automation Interface können TwinCAT XAE-Konfigurationen per Programmier-/ Skriptcodes automatisch erzeugt und bearbeitet werden. Die Automatisierung einer TwinCAT-Konfiguration steht dank sogenannter Automation Interfaces zur Verfügung, auf die über alle COM-fähigen Programmiersprachen (z. B. C++ oder .NET) und auch über dynamische Scriptsprachen, wie Windows PowerShell, IronPython, oder sogar das (veraltete) Vbscript, zugegriffen werden kann. Ebenfalls ist die Verwendung aus der MATLAB®-Umgebung möglich.

Eine ausführliche Dokumentation der Produkts finden Sie hier: TwinCAT Automation Interface

Verwendung in MATLAB®

In MATLAB® können Sie das Automation Interface durch das Kommando NET.addAssembly sichtbar machen. Damit sind Sie in der Lage, die in der Produktdokumentation beschriebenen Interfaces (Automation Interface API) zu nutzen. Ebenfalls finden Sie in der Produktdokumentation viele Programmierbeispiele für die Nutzung aus C# und PowerShell (Automation Interface Configuration).

Um Ihnen den Einstieg aus MATLAB® zu vereinfachen, finden Sie im Folgenden eine Beispielimplementierung für MATLAB® auf Basis einer MATLAB®-Klasse, welche Sie nutzen, verändern und erweitern können.

4.10.1.1 Beispiel: Tc3AutomationInterface

Übersicht

Der Beispiel-Code besteht aus zwei m-Files:

- *Tc3AutomationInterface.m*: MATLAB®-Klasse, welche einige häufig genutzte Methoden implementiert.
- *Tc3AutomationInterfaceGuide.mlx*: MATLAB® Live-Script, welches die MATLAB®-Klasse beispielhaft aufruft.

Beispiel mit MATLAB® aufrufen

Das TwinCAT Tool for MATLAB® and Simulink® Setup installiert das Beispiel auf Ihrem System. Rufen Sie das Beispiel mit dem MATLAB® Command Window auf:
`TwinCAT.ModuleGenerator.Samples.Start('AutomationInterface').`

Das MATLAB®-Script

Das MATLAB®-Script liefert ein Beispiel, wie Sie eine TwinCAT-Solution erzeugen, den EtherCAT-Master nach I/O scannen, zwei TcCOM-Module instanziieren, verlinken und das Projekt auf einem Target aktivieren können.

Um das Script ausführen zu können, müssen die beiden verwendeten TcCOM in Ihrem *publish directory %TwinCATDir%\\CustomConfig\Modules* vorhanden sein. Laden Sie dazu das Beispiel Temperature Controller aus der TE1400 | Target for MATLAB®/Simulink® herunter. Die Dateiordner aus dem Verzeichnis *.\TE1400Sample_TemperatureController_PrecompiledTcComModules\Actual TwinCAT versions* kopieren Sie dann in das *publish directory*.

Führen Sie das m-File *Tc3AutomationInterface_Testbench.m* aus. Es wird im Hintergrund die aktuellste auf Ihrem System verfügbare Visual Studio-Instanz geöffnet und die TwinCAT Solution konfiguriert, speichert und aktiviert.

Die MATLAB®-Klasse

Die Properties

In den Properties der Klasse *Tc3AutomationInterface* werden alle zur Instanz der Klasse gehörigen Variablen und Interfaces gehalten. So können mehrere TwinCAT Solutions in einem MATLAB®-Script aufgebaut werden, indem für jede Solution eine Instanz der Klasse erzeugt wird. Damit ergeben sich dann keine Überschneidungen.

Der Konstruktor

```
function this = Tc3AutomationInterface
```

Der Konstruktor lädt alle notwendigen Assemblies und setzt bei Erfolg das Property AssembliesLoaded auf TRUE. Die geladenen Assemblies sind:

- EnvDTE und EnvDTE80: Bibliotheken für das Visual Studio Core Automation. Notwendig zur Konfiguration von Visual Studio.
- TCatSysManagerLib: TwinCAT Automation Interface Bibliothek zur Konfiguration einer TwinCAT Solution in Visual Studio.
- TwinCAT.Ads: ADS Bibliothek, z. B. zum Lesen und Verändern des XAR state.
- System.Xml: Bibliothek zum Parsen von XML Dateien.

Ausgewählte Methoden der Klasse

```
function TcComObject = CreateTcCOM(this, Modelname)
```

Nutzen Sie die Hilfefunktion von MATLAB®, um die Funktion und die Parameter der Methode einzusehen.

```
>> help Tc3_AI.CreateTcCOM
--- help for Tc3AutomationInterface/CreateTcCOM ---

  CreateTcCOM  creates a new instance of a TcCOM

    TcComObject = CreateTcCOM(Modelname)
    Instanciates the TcCOM with the specified name (Modelname).
    Also a task with a matching cycle time is created and linked to
    the TcCOM-Object.

    set properties: TcCOM

    see also:
    Beckhoff Infosys
```

Ebenfalls wird bei einigen Methoden ein Link ins das Beckhoff Infosys angeboten. Diese verweisen auf Dokumentationsbeispiele aus der TwinCAT Automation Interface Dokumentation, sodass Sie direkt einen Vergleich zur Implementierung in MATLAB®, C# und PowerShell einsehen können. Ebenfalls finden Sie in einigen Sektionen im Kommentar einen Link zum Beckhoff Infosys, sodass Sie die Quelle der Information einsehen können.

Die Methode CreateTcCOM beginnt zunächst mit dem Parsen der *<modelname>.tmc* Datei. Daraus wird mit der *System.Xml* die ClassID, die Task Zykluszeit sowie Task Priorität extrahiert. Dann wird mit dem Automation Interface ein entsprechendes TcCOM instanziiert sowie eine (oder mehrere) zugehörige Task(s) erzeugt. Abschließend wird (werden) die Task(s) dem TcCOM zugeordnet.

```
function ActivateOnDevice(this, AmsNetId)
```

Um den aktuellen Status einer TwinCAT Runtime, z. B. config oder run, zu erfragen bzw. zu verändern, wird TwinCAT ADS genutzt. In der Methode ActivateOnDevice wird zunächst die XAR mit der spezifizierten AmsNetId in den Config-Modus geschaltet und dann die aktuelle TwinCAT-Konfiguration aktiviert sowie das System gestartet. Zwischen den Einzelschritten sind Pausen eingetragen, da dieser Vorgang ggf. etwas Zeit benötigt.

Statische Methoden

Statische Methoden sind auch ohne Instanz der Klasse verfügbar.

```
function vsVersions = GetInstalledVisualStudios
```

Vorbereitet ist hier eine Funktion, welche über Registry Key Einträge die auf dem System verfügbaren Visual Studio Installationen detektiert und auflistet. Die Implementierung ist auf VS 2010 bis VS 2017 limitiert.

Dokumente hierzu

- AutomationInterfaceMATLAB (Resources/zip/5776206091.zip)

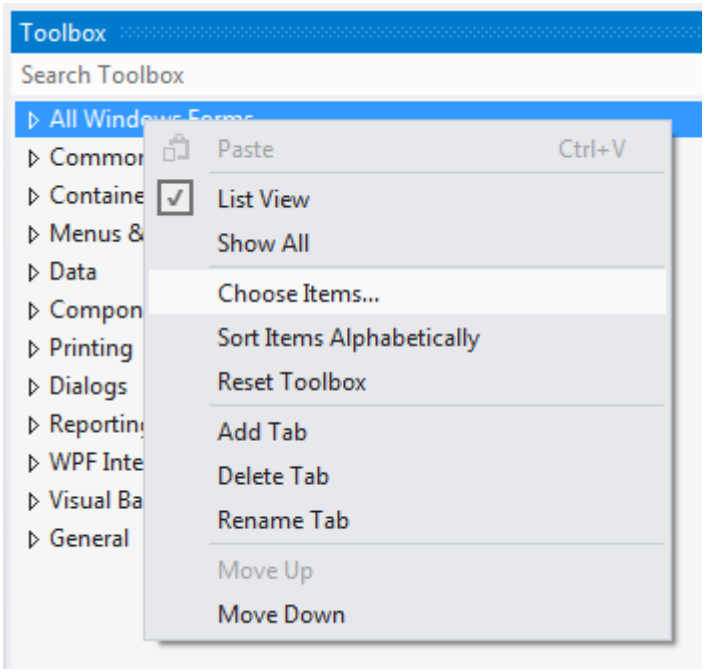
4.10.2 Einbinden des Blockdiagramm-Controls

Das Control welches das Block-Diagramm in der TwinCAT XAE Umgebung darstellt, kann auch als Control in eigene Visualisierungen eingebunden werden.

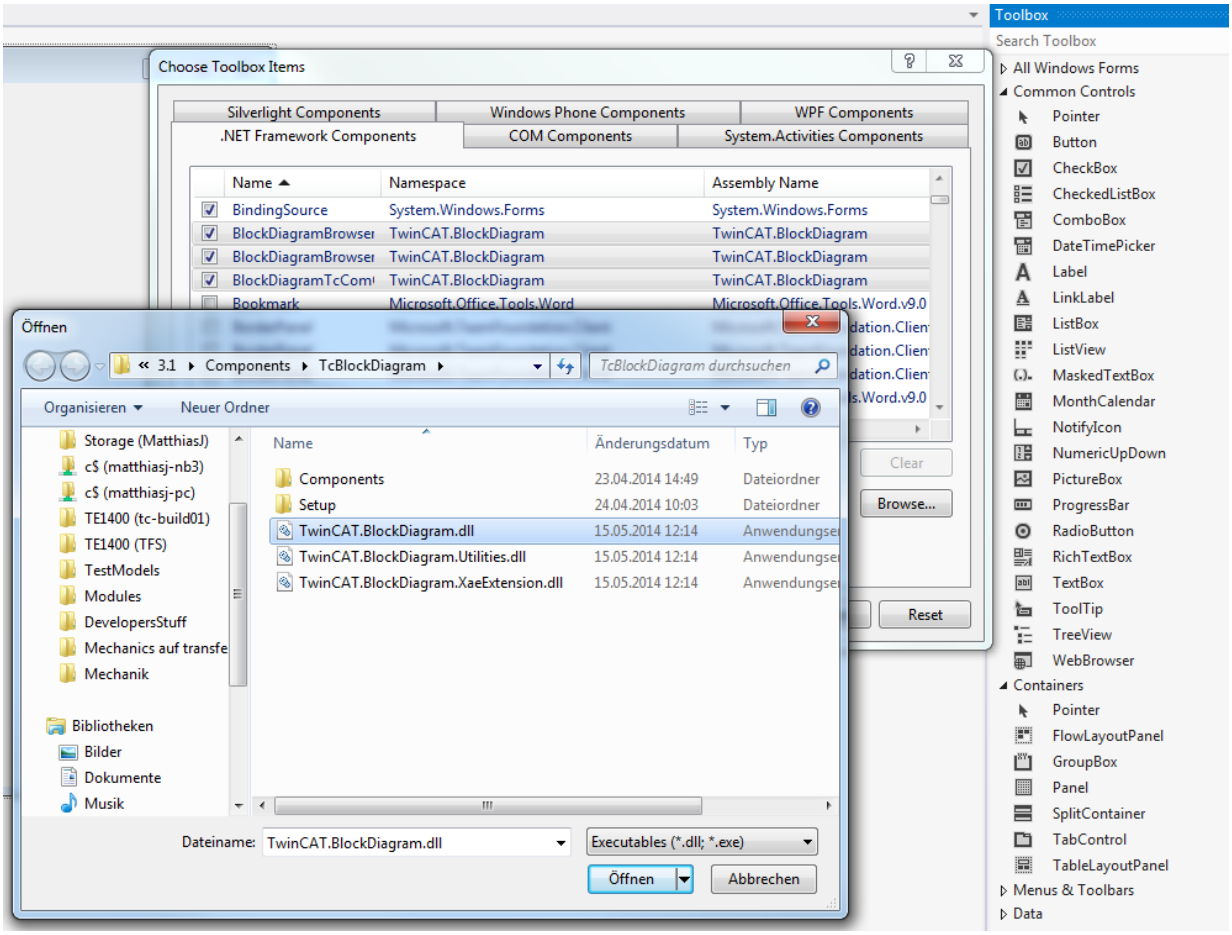
Folgende Schritte sind dafür notwendig:

1. Erstellen Sie eine Windows-Forms-Applikation.
2. Fügen Sie TwinCAT.BlockDiagramm.dll zur Toolbox hinzu:

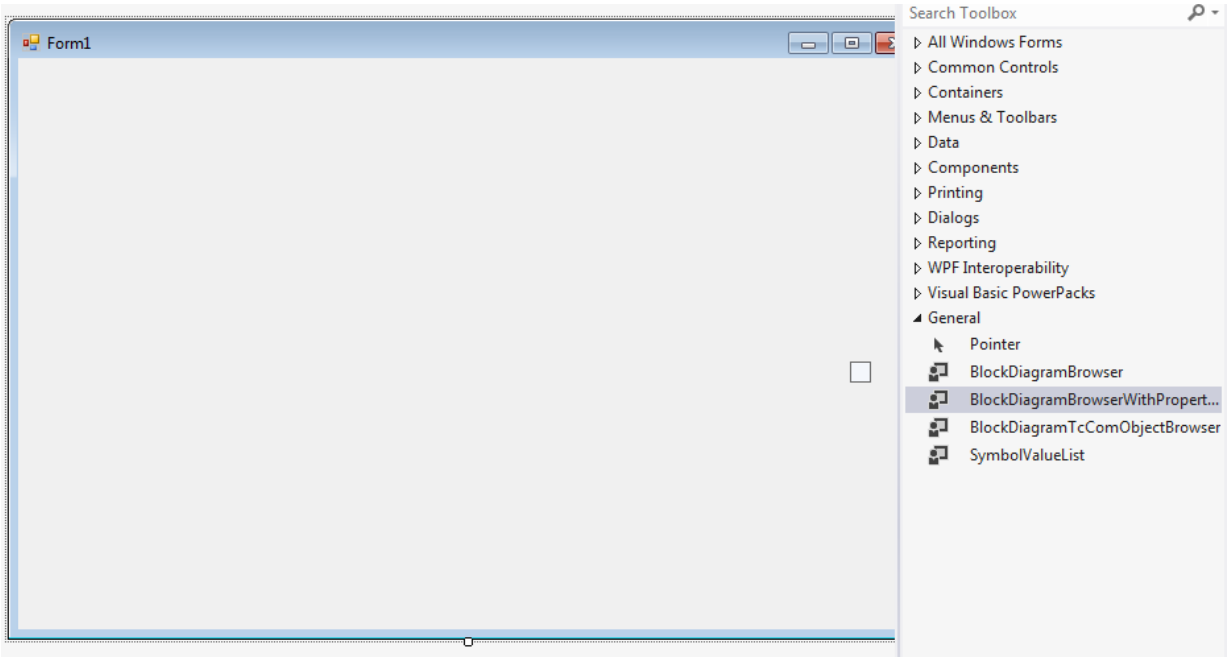
3. Wählen Sie dafür im Kontextmenü den Eintrag „Choose Items…“ aus.



4. Browsen Sie zur TwinCAT.Blockdiagram.dll, welche sich unter *<TwinCAT-Installationspfad>\3.1\Components\TcBlockDiagram* befindet.



5. Fügen Sie eine TcBlockdiagram-Control-Instanz zum Windows-Forms-Object per Drag&Drop hinzu.

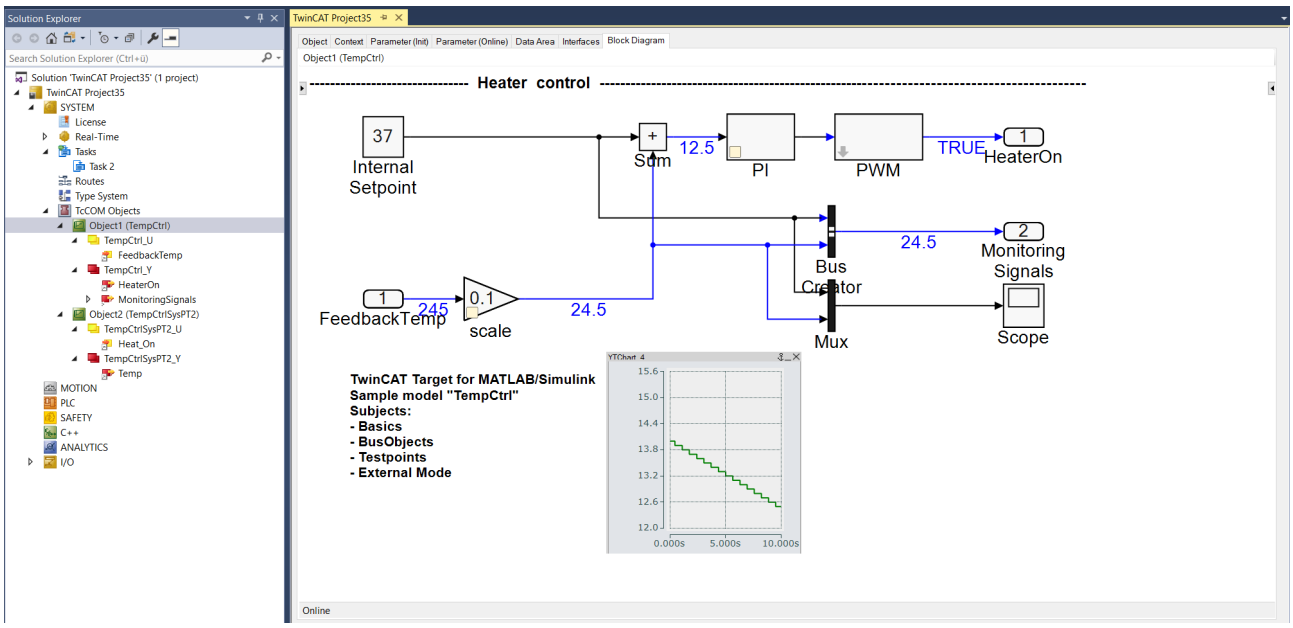


4.10.3 Erstellte TwinCAT-Objekte selbst ausprobieren

✓ Sie haben kein MATLAB® oder möchten einfach so mit dem TwinCAT Target for Simulink® kompilierte TwinCAT-Objekte ausprobieren?
Dann folgen Sie der untenstehenden Beschreibung.

1. https://infosys.beckhoff.com/content/1031/te1400_tc3_target_Matlab/Resources/14632780043.zip,
2. ZIP entpacken und tmx-Archiv ausführen.
   ⇨ Im Zip liegen zwei tmx-archive [▶ 139] als Executables.
3. Führen Sie diese aus.
   ⇨ Dadurch entpacken Sie die TwinCAT-Objekte in den korrekten TwinCAT-Pfad (Engineering Repository).
4. XAE öffnen und Instanzen anlegen.
   ⇨ Sie können nun in TwinCAT 3 neue TcCOM-Objekte anlegen. Sie finden die zwei Objekte unter „TE140x Module Vendor“ – TE140x – Simulink Modules.
5. Legen Sie die Instanzen an und weisen Sie eine TwinCAT-Task zu.
6. Aktivieren Sie die Konfiguration.
7. Binden Sie das zur Signierung verwendete Zertifikat in die Trust-List des Targets ein.
   ⇨ Das Target-System muss die verwendeten tmx-files laden. Diese tragen eine Signierung, erstellt mit einem OEM-Zertifikat. Das verwendete OEM-Zertifikat ist vermutlich noch nicht auf dem Target-System in der White-List.
8. Fügen Sie entsprechend das Zertifikat zur White-List hinzu [▶ 98].
9. Aktivieren Sie die Konfiguration.

⇨ Nach Aktivierung der Konfiguration, können Sie im Block Diagram [▶ 198] das Verhalten der Objekte beobachten und Parameter verändern.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Project35' (1 project)
TwinCAT Project35
SYSTEM
License
Real-Time
Tasks
Task 2
Routes
Type System
TcCOM Objects
Object1 (TempCtrl)
TempCtrl_U
FeedbackTemp
TempCtrl_Y
HeaterOn
MonitoringSignals
Object2 (TempCtrlSysPT2)
TempCtrlSysPT2_U
Heat_On
TempCtrlSysPT2_Y
Temp
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
TwinCAT Project35
Object
Context
Parameter (Init)
Parameter (Online)
Data Area
Interfaces
Block Diagram
Object1 (TempCtrl)
Heater control
37
Internal
Setpoint
Sum
12.5
PI
PWM
TRUE
1
HeaterOn
24.5
2
Monitoring
Signals
Bus
Creator
1
FeedbackTemp
245
0.1
scale
24.5
Mux
Scope
TwinCAT Target for MATLAB/Simulink
Sample model "TempCtrl"
Subjects:
- Basics
- BusObjects
- Testpoints
- External Mode
15.6
15.0
14.4
13.8
13.2
12.6
12.0
0.000s
5.000s
10.000s
Online

Mehr Informationen:
www.beckhoff.com/te1400

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com