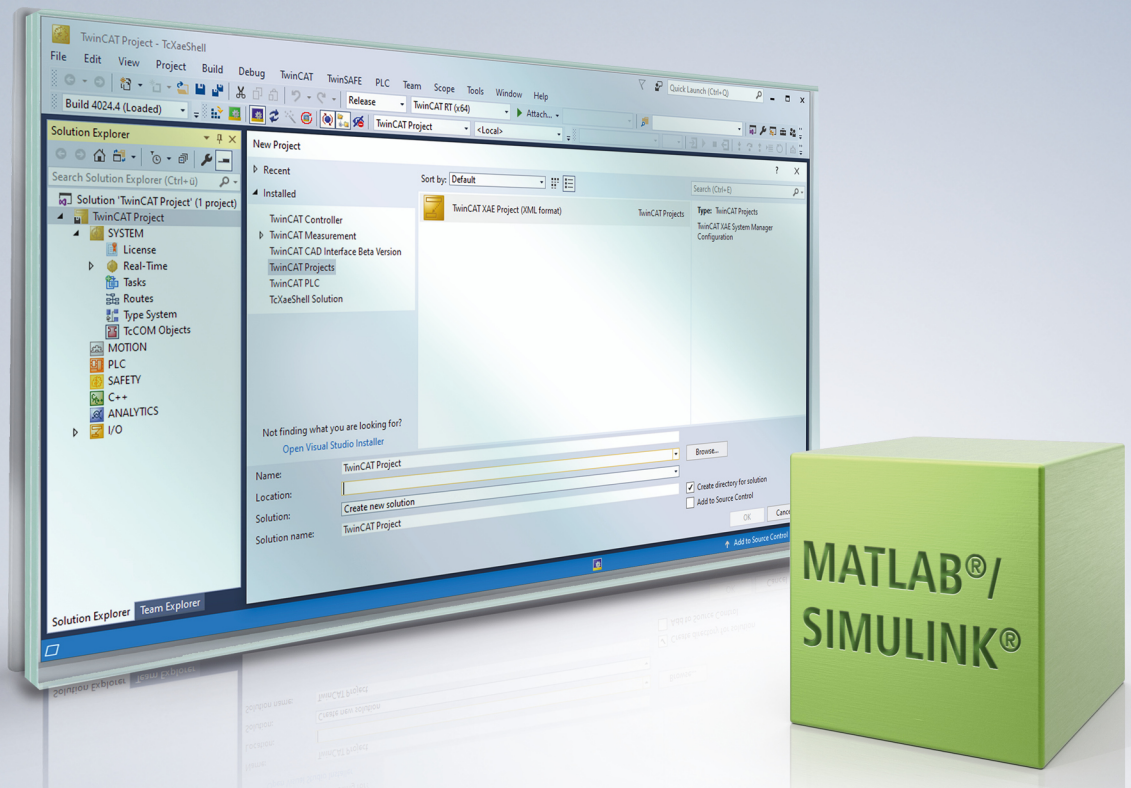
BECKHOFF New Automation Technology

Handbuch | DE

TE1401

TwinCAT 3 | Target for MATLAB®



05.08.2025 | Version: 1.3.0

Inhaltsverzeichnis

1 Vorwort 5
1.1 Hinweise zur Dokumentation 5
1.2 Zu Ihrer Sicherheit 6
1.3 Hinweise zur Informationssicherheit 7
1.4 Ausgabestände dieser Dokumentation 8
2 Übersicht 9
3 Installation 10
3.1 Ersteinrichtung der Software 12
3.1.1 Einrichtung von Default-Einstellungen und MATLAB®-Path setzen 12
3.1.2 Einrichtung der Treibersignierung 15
4 Lizenzen 20
5 Quickstart 21
6 Übersicht zu automatisch generierten Dateien 33
7 Einstellungen des TwinCAT-Modulgenerators 35
7.1 Erstellen von TMX-Archiven 40
8 Anwendung von Modulen in TwinCAT 41
8.1 Arbeiten mit dem TcCOM-Modul 41
8.1.1 MATLAB Code Darstellung 43
8.2 Arbeiten mit der SPS-Bibliothek 46
8.2.1 Online-Change der SPS-Bibliothek 50
8.2.2 Aufruf eines TcCOM-Objekts aus der SPS 51
8.3 Debugging 52
8.4 Exception Handling 55
8.5 Verwenden von Realtime Monitor Zeitmarken 63
9 FAQ 64
9.1 Build eines Sample schlägt fehl 64
9.2 Gibt es Limitierungen hinsichtlich der Ausführung von Modulen in Echtzeit? 64
10 Beispiele 66
10.1 TwinCAT Automation Interface: Verwendung in MATLAB® 66
10.1.1 Beispiel: Tc3AutomationInterface 67

1 Vorwort

1.1 Hinweise zur Dokumentation

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

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

Disclaimer

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

Marken

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

Fremdmarken

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

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

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

Haftungsausschluss

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

Qualifikation des Personals

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

Signalwörter

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

Warnungen vor Personenschäden

⚠ GEFAHR

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

⚠ WARNUNG

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

⚠ VORSICHT

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

Warnung vor Umwelt- oder Sachschäden

HINWEIS

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

Information zum Umgang mit dem Produkt

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

1.3 Hinweise zur Informationssicherheit

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

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

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

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

1.4 Ausgabestände dieser Dokumentation

Version	Änderungen
1.3.x	TwinCAT 3.1 Build 4026 • Kurz-Information zur Installation [▶ 10] • Aktualisierung der Systemvoraussetzungen

2 Übersicht

TE1401 TwinCAT Target for MATLAB®

Mit dem TwinCAT 3 Target for MATLAB® ist es möglich, die in der Skriptsprache MATLAB® entwickelten Funktionen in TwinCAT 3 nutzbar zu machen. Die Funktionen werden dabei automatisch mithilfe des MATLAB® Coder™ in C/C++-Code übersetzt und mit dem TwinCAT 3 Target for MATLAB® in TwinCAT-Objekte überführt. Diese lassen sich nahtlos im TwinCAT 3 Engineering verwenden, z. B. mit SPS-Quellcode zu einem Gesamtprojekt erweitern, debuggen und mit Feldbusteilnehmern verknüpfen. Einerseits können die automatisch generierten Module als TcCOM-Objekt und andererseits als SPS-Funktionsbaustein in die TwinCAT-Solution eingebunden werden. Die eingefügten Module werden mit dem gesamten TwinCAT-Projekt in die TwinCAT-3-Laufzeit heruntergespielt und dort, wie alle anderen Objekte, innerhalb der Echtzeitumgebung ausgeführt. TwinCAT 3 Target for Simulink® unterstützt sowohl Targets mit Windows 32 Bit und 64 Bit als auch TwinCAT/BSD.

Weitere Informationen

Technisches Kurzvideo

- TwinCAT Target for MATLAB

Produktbeschreibung

- https://www.beckhoff.com/TE1401

Webseite zu MATLAB® und Simulink® mit TwinCAT 3

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

3 Installation

Systemvoraussetzung

Im Folgenden wird zwischen dem Engineering-PC und dem Laufzeit-PC unterschieden. Dazu wird folgende Definition getroffen: Auf dem Engineering-PC werden MATLAB®-Funktionen in TwinCAT-Objekte durch Einsatz des Target for MATLAB® konvertiert. Ebenso kann, muss aber nicht, auf diesem PC eine TwinCAT-Solution erstellt werden, welche die erstellten Objekte verwendet. Die erstelle 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
  - MATLAB® und MATLAB® Coder™ Toolbox
- 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.
- 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

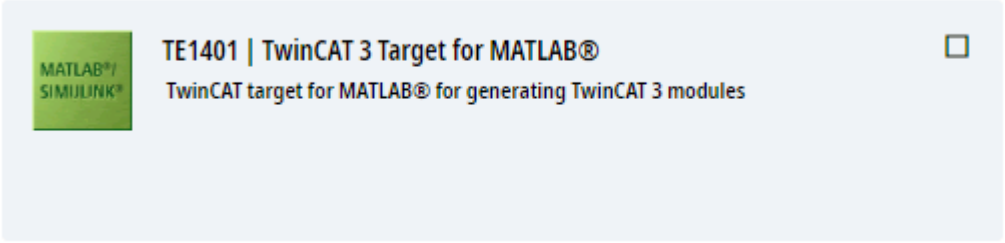
- Unterstützte Betriebssysteme
  - Windows 11, Windows 10, Windows Server (32bit und 64bit)
  - TwinCAT/BSD
- TwinCAT XAR Version 3.1.4024.7 oder höher

> **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 (TwinCAT 3.1 Build 4026)

Der Workload TE1401 | TwinCAT 3 Target for MATLAB® installiert alle notwendigen Abhängigkeiten, die zum Erstellen von TwinCAT-Objekten notwendig sind.

- Installieren Sie eine der unterstützten **Visual Studio**-Versionen, falls nicht bereits vorhanden. Beachten Sie die Installation der Option *Desktop development with C++.*
- Installieren Sie den TwinCAT Package Manager.
- Installieren Sie über den TwinCAT Package Manager folgenden Workload:



Command line: `tcpkg install TE1401.TargetForMATLAB.XAE`

- Führen Sie die Integration in das Visual Studio aus, welches Sie mit der Option *Desktop development with C++* installiert haben.

- Starten Sie MATLAB® als Administrator und führen Sie *C:\Program Files (x86)\Beckhoff\TwinCAT\Functions\TE14xx-ToolsForMatlabAndSimulink\SetupTE14xx.p* in MATLAB® aus.

Möchten Sie ein TwinCAT Projekt erstellen und dort bereits gebaute TwinCAT-Objekte verwenden, installieren Sie folgende Workloads:

- TwinCAT Standard
  `tcpkg install TwinCAT.Standard.XAE`
- TwinCAT Block Diagram Classic
  `tcpkg install TwinCAT.XAE.BlockDiagramClassic`
- In einigen Fällen benötigen Sie auch folgende Pakete:
  `tcpkg install TwinCAT.XAE.TMX.MatSimUtilities`
  (zur Konvertierung von ITcVnImage in ARRAY)

Installation (TwinCAT 3.1 Build 4024)

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

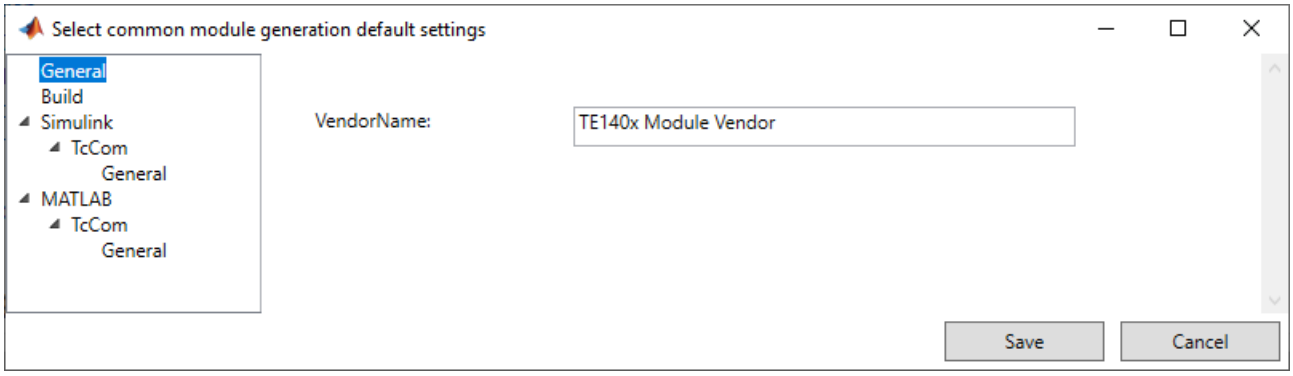
1. Starten Sie das **TwinCAT 3 XAE oder Full Setup**, falls nicht bereits vorhanden.
   Sollte eine Visual Studio- sowie TwinCAT-Installation bereits vorhanden sein, die Visual Studio Version jedoch nicht den oben genannten Anforderungen entsprechen (z. B. TwinCAT XAE Shell oder Visual Studio ohne C++-Option), 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 TE1401 TwinCAT Target for MATLAB®.
   ⇨ Die Installation des TwinCAT Target for MATLAB® 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.

Einrichtung der Software

Ausführung SetupTE14xx.p

Nach Ausführung des p-Files öffnet sich ein Dialog, in dem Sie generelle Default-Einstellungen speichern können, die dann für das System gelten. Sie können die Einstellungen direkt setzen oder zu einem späteren Zeitpunkt einstellen/verändern.

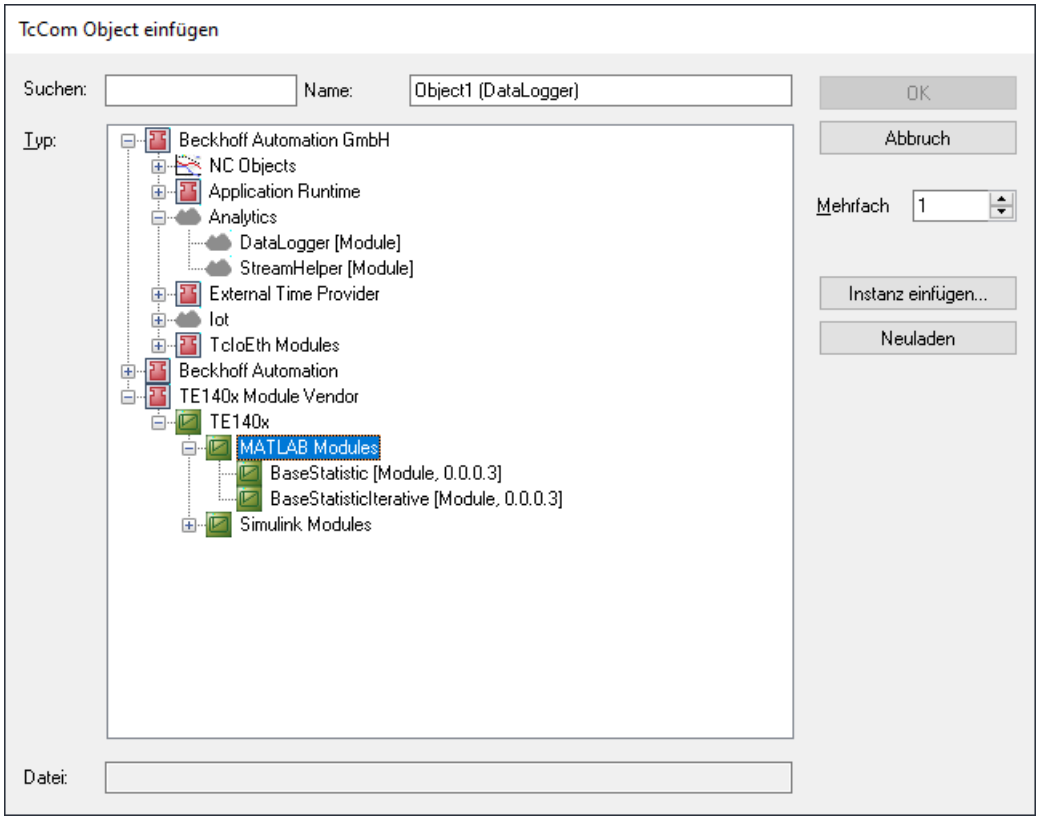


Möchten Sie das p-File ohne diesen Dialog ausführen, können Sie folgenden Befehl nutzen:
`SetupTE140x('Silent', true);`

Einstellungsmöglichkeiten im Dialog sind:

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

Diese Einstellungen beeinflussen die Hierarchie, in der die generierten TwinCAT-Objekte einsortiert werden. Siehe untenstehende Grafik. Hier sind die Einträge VendorName „TE140x Module Vendor“ und GroupName „TE140x|MATLAB Modules“ für MATLAB® und „TE140x|Simulink Modules“ für Simulink®.



Um die Default-Einstellungen zu verändern, können Sie auf den Dialog mit `TwinCAT.ModuleGenerator.Settings.Edit` in der MATLAB® Konsole zugreifen. Hier werden Ihnen auch zusätzliche Einträge angeboten, die Sie als Default hinterlegen können.

3.1 Ersteinrichtung der Software

3.1.1 Einrichtung von Default-Einstellungen und MATLAB®-Path setzen

Initiale Einrichtung der Software

MATLAB®-Pfade setzen

Wie unter Installation beschrieben, ist nach der Software-Installation die Ausführung der Files SetupTE14xx.p notwendig. Dieses Skript fügt die notwendigen Pfade Ihrem MATLAB®-path hinzu. Um MATLAB®-Pfade speichern zu können benötigt MATLAB® -Administratorrechte.

SetupTE14xx.p zusätzlich nach jedem Produktupdate ausführen

Um sicherzustellen, dass alle Pfade im MATLAB®-Path gesetzt sind, führen Sie das p-file bitte auch nach Updates des Produkts aus.

Default-Einstellungen setzen

Das Skript öffnet zudem einen Dialog, in dem Sie generelle Default-Einstellungen für das TwinCAT-Target speichern können. Diese Einstellungen gelten dann systemweit, also für *alle* installierten MATLAB®-Versionen.

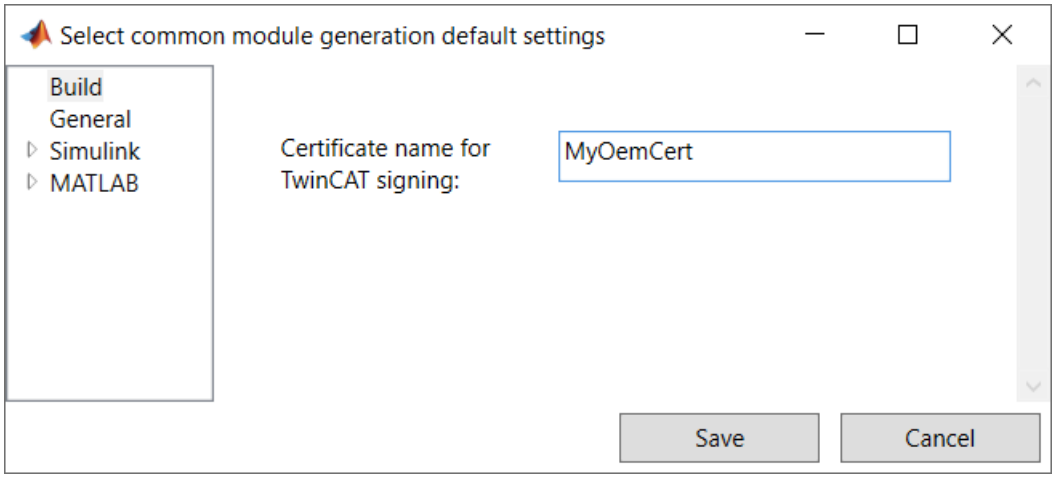
Wenn Sie ohne diesen Dialog das p-File ausführen möchten, können Sie folgenden Befehl nutzen:

```
SetupTE14xx('Silent', true);
```

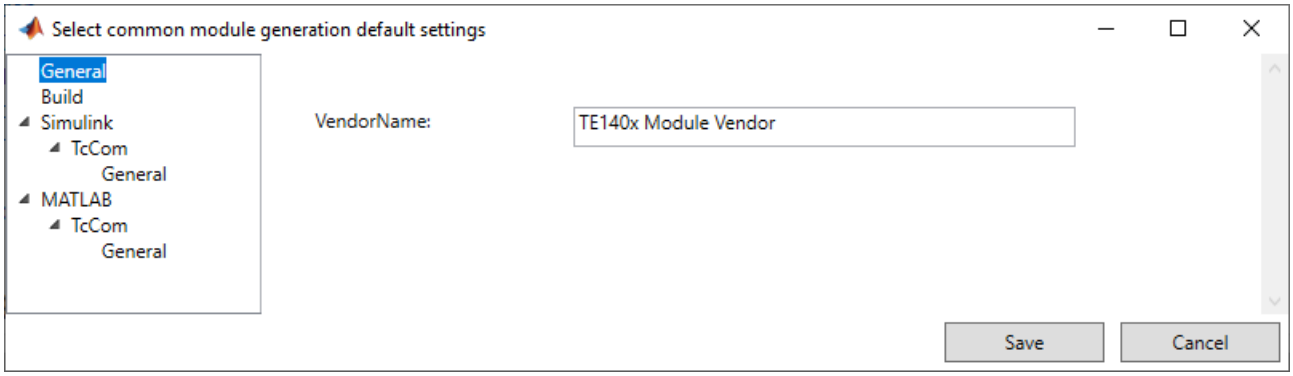
Dabei werden die Standardwerte des Dialogs gesetzt.

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

Im Dialog unter dem Reiter **Build** können Sie ein Default-Zertifikat zur Treibersignierung hinterlegen. Die Einrichtungsmöglichkeiten der Treibersignierung werden vollumfänglich im Kapitel Einrichtung der Treibersignierung [▶ 15] erläutert.



Des Weiteren können Sie im Dialog die Hierarchie der generierten TwinCAT-Objekte beeinflussen.

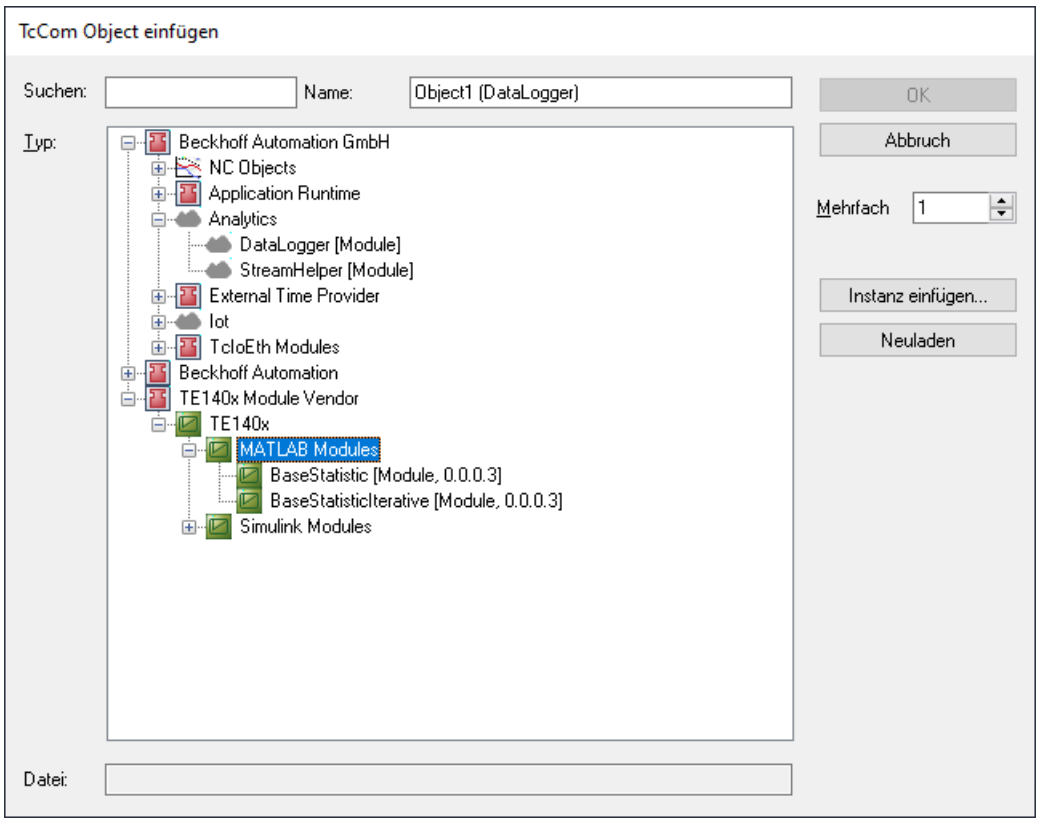


Es gibt folgende Einstellungsmöglichkeiten im Dialog:

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

Die Hierarchie zeigt sich in TwinCAT an folgendem Beispiel:

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

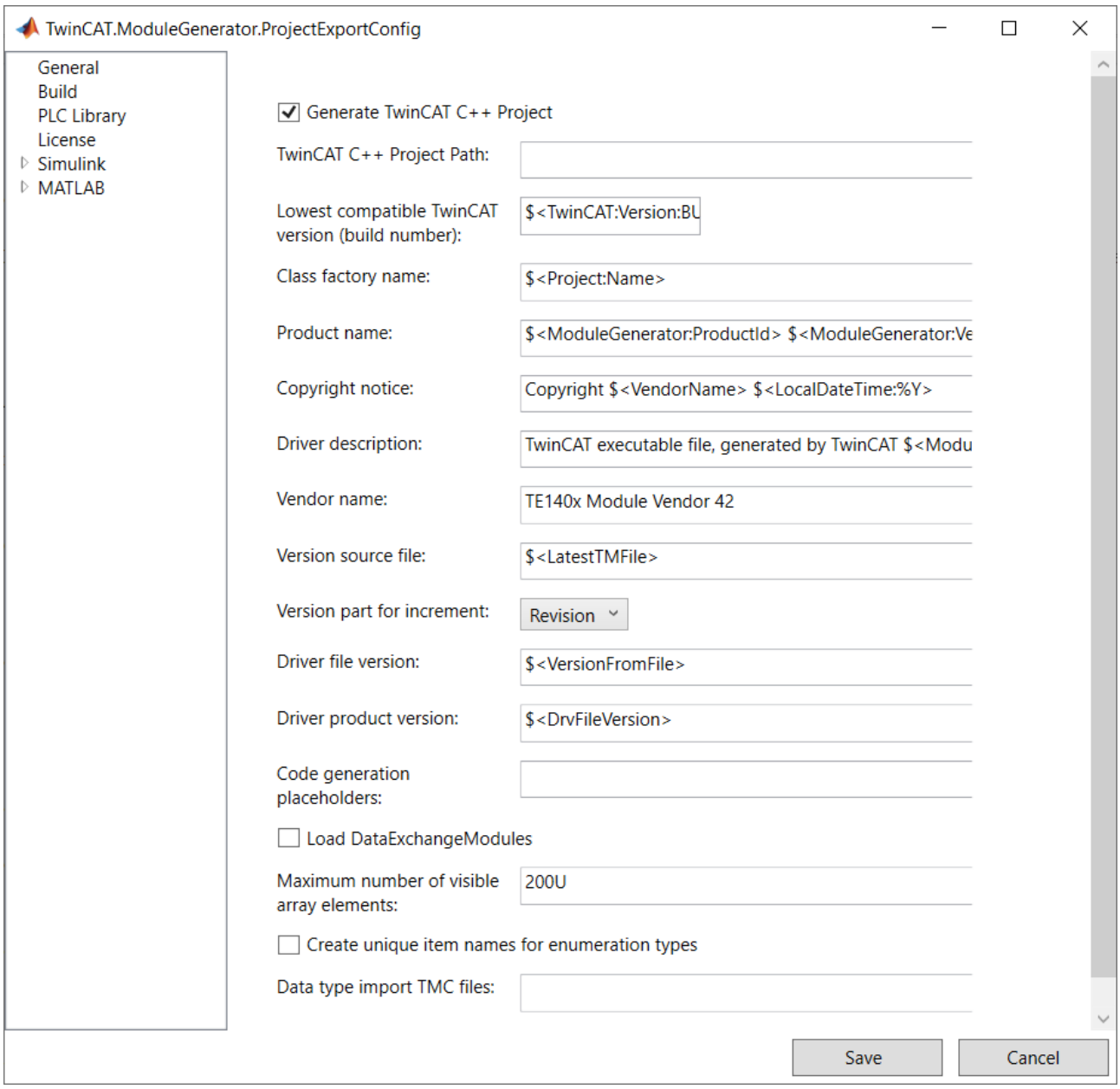


Einrichtung der Software verändern

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.

3.1.2 Einrichtung der Treibersignierung

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 der Treiber 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)*).
  - Sie werden hier aufgefordert, ein Passwort für Ihr Zertifikat anzulegen.
- Treiber können auch ohne Signierung erstellt und nachträglich signiert werden.
- Für Testzwecke in der Entwicklungsphase genügt ein nicht-gegensigniertes Zertifikat.
- Gegensignierte Zertifikate können bei Beckhoff kostenfrei bestellt werden (TC0008).

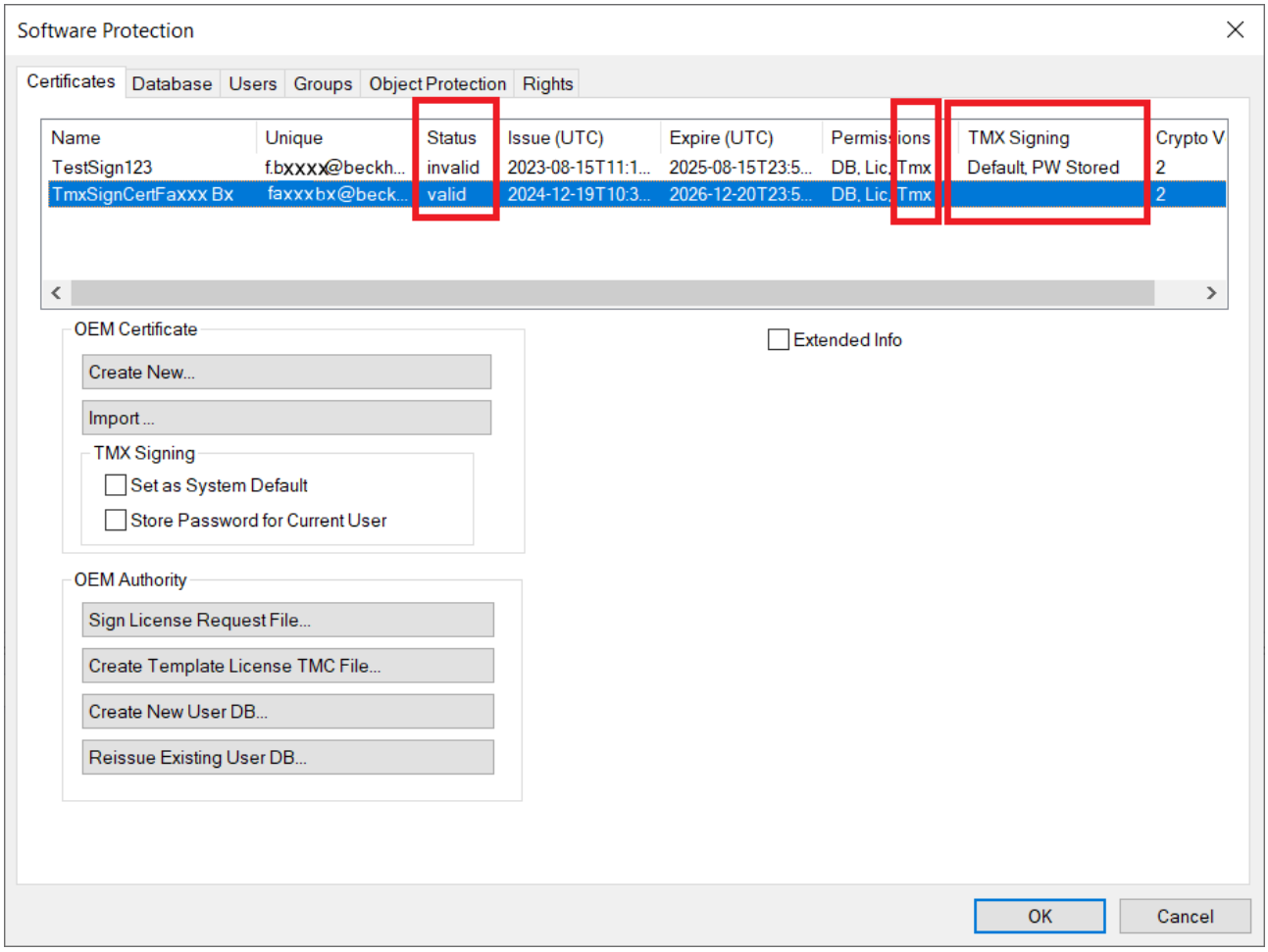
OEM-Zertifikat Level 2 unter Software Protection einrichten

TwinCAT Build 4026: Voraussetzung für den Dialog zur Einrichtung

Folgende Ausführungen zu Software Protection gelten nur für TwinCAT 3.1 Build 4026. Notwendig ist mindestens der Workload TwinCAT Standard 4026.14. Sollten Sie mit älteren Versionen arbeiten, lesen Sie bitte im unteren Abschnitt weiter „OEM-Zertifikat Level 2 zur Treibersignierung einrichten ohne den Dialog Software Protection“.

In der Oberfläche der Software Protection (**Menu bar > TwinCAT > Software Protection…**) können Sie sowohl Zertifikate erstellen (Create New…) als auch:

- Ein Zertifikat als systemweites Default-Zertifikat zur Signierung von tmx-Dateien festlegen (optional).
- Zu jedem Zertifikat das jeweils zugehörige Passwort für den angemeldeten Windows Nutzer hinterlegen (erforderlich).



In der obigen Übersicht befinden sich beispielhaft zwei Zertifikate.

Das erste Zertifikat „TestSign123“ ist nicht von Beckhoff gegensigniert, deshalb wird es im Status als *invalid* klassifiziert. Nicht gegensignierte Zertifikate können dennoch zum Signieren genutzt werden. Das Zielsystem ist dann in den Testmodus zu versetzen – siehe Abschnitt Verhalten der TwinCAT-Laufzeit [▶ 18]. Das Zertifikat „TmxSignCertFaxxxBx“ ist hingegen gegensigniert und damit als *valid* klassifiziert. Beide Zertifikate eignen sich zum Signieren von tmx-Dateien, wie unter *Permissions* zu sehen ist. In der Spalte „TMX Signing“ ist mit „Default“ vermerkt, wenn ein Zertifikat als systemweites Default-Zertifikat festgelegt ist. Der Vermerk „PW Stored“ zeigt an, dass das Passwort des Zertifikats für den angemeldeten Windows Nutzer verfügbar/ hinterlegt ist.

Zertifikat als Systemweites Default-Zertifikat festlegen (optional)

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

Markieren Sie das Zertifikat, welches Sie als Default-Zertifikat nutzen möchten aus der Liste im Dialog *Software Protection* aus und wählen Sie die Checkbox „Set as System Default“ aus.

Es wird daraufhin eine Umgebungsvariable mit dem Namen *TcSignTwinCatCertName* angelegt. Umgebungsvariablen werden unter Windows beim Starten eines Prozesses bekanntgemacht. Starten Sie daher MATLAB® neu, wenn Sie den Prozess bereits ausführen.

Weitere Möglichkeiten zur Nutzung von Zertifikaten finden Sie weiter unten in diesem Kapitel.

Passwort zu einem Zertifikat hinterlegen (erforderlich)

Das Passwort eines Zertifikats darf aus Sicherheitsgründen nicht im Projekt oder Quellcode im Simulink®-Modell oder im MATLAB®-Code eingetragen werden. Mit „Store Password for Current User“ hinterlegen Sie für Ihre Zertifikate die zugehörigen Passwörter auf ihrem System.

Die Passwörter werden obfuskiert in der Registry des Windows-Betriebssystems abgelegt. Dadurch ist das Passwort zu einem bestimmten Zertifikat im Betriebssystem (für den Current User) bekannt und wird automatisch verwendet.

Markieren Sie das Zertifikat, für welches Sie das zugehörige Passwort hinterlegen möchten im Dialog *Software Protection*. Wählen Sie „Store Password for Current User“. Sie werden aufgefordert, Ihr Passwort einzutragen. Wenn es erfolgreich überprüft und eingetragen wurde, erscheint der Vermerk „PW Stored“ unter „TMX Signing“.

Eine **alternative Variante** zum Hinterlegen eines Passworts stellt das Command Prompt mit dem TcSignTool (C:\Program Files (x86)\Beckhoff\TwinCAT\3.1\SDK\Bin) dar.

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
```

OEM-Zertifikat Level 2 zur Treibersignierung einrichten ohne den Dialog Software Protection

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

Verfügbare Zertifikate liegen unter:

Build 4026: C:\ProgramData\Beckhoff\TwinCAT\3.1\CustomConfig\Certificates
Build 4024: C:\TwinCAT\3.1\CustomConfig\Certificates

Handling des Zertifikats

Es gibt vier Varianten zur Signierung von tmx-Treibern.

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

Diese Variante ist identisch mit dem Weg über „Software Protection“ > „Set as System Default“.

Für diese Variante können Sie alternativ auch händisch eine Windows-Umgebungsvariable anlegen. Legen Sie eine neue **Umgebungsvariable** unter **User > Variables** an mit:

Variable: *TcSignTwinCatCertName*

Value: Full-Path des Zertifikats

Variante 2: Systemweites Default-Zertifikat für MATLAB®

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 Common Settings-Dialog mit *TwinCAT.ModuleGenerator.Settings.Edit* (MATLAB Command line) 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.

Das TcSignTool ist ein Kommandozeilen-Programm. Öffnen Sie bspw. das Command Prompt und führen Sie `tcsigntool sign /?` aus, um die Hilfe angezeigt zu bekommen. Das Programm finden Sie hier:

Build 4026: C:\Program Files (x86)\Beckhoff\TwinCAT\3.1\SDK\Bin

Build 4024: C:\TwinCAT\3.1\SDK\Bin

TcSignTool aus MATLAB® bedienen

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

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"
```

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:

Testmodus für nicht gegensignierte Zertifikate

Wenn Sie ein nicht-gegensigniertes OEM-Zertifikat zur Signierung nutzen, müssen Sie 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.

Whitelist für Zertifikate auf Zielsystemen

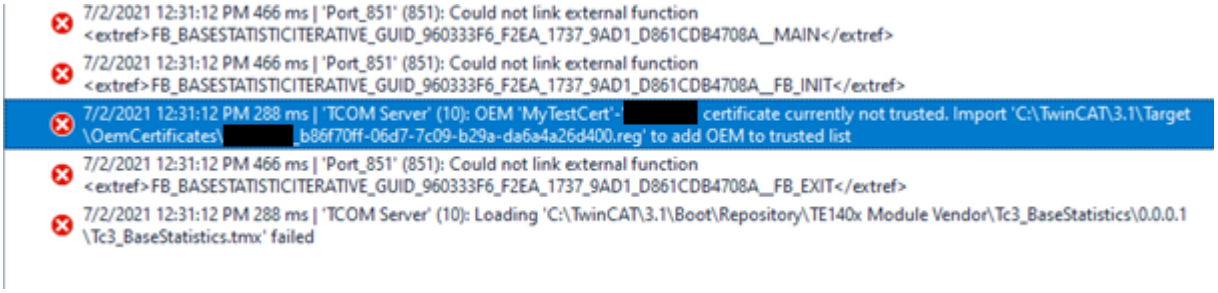
Jede TwinCAT-Laufzeit (XAR) hat eine eigene eine Whitelist an vertrauenswürdigen Zertifikaten.

Verhalten bei TwinCAT Build 4026

Das TwinCAT-XAE prüft, ob alle Zertifikate, die zur Aktivierung der Konfiguration notwendig sind, auf dem Laufzeit-System in der Whitelist stehen. Ist dies nicht der Fall, erscheint ein Pop-Up-Fenster. Sie können dort die Whitelist-Einträge setzen.

Verhalten bei TwinCAT Build 4024

Ist das Zertifikat, welches zur Signierung genutzt wurde, nicht in dieser Whitelist 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 Whitelist 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.

4 Lizenzen

Um die gesamte Funktionalität des TE1401 TwinCAT Target for MATLAB® nutzen zu können, sind zwei Lizenzen erforderlich. Zum Einen die Engineering Lizenz TE1401 zur Erstellung von TwinCAT-Objekten aus MATLAB®-Funktionen und zum Anderen eine Laufzeitlizenz zur Ausführung dieser Objekte in der TwinCAT-Laufzeit.

Engineering-Lizenz

Die Lizenz *TE1401 Target for MATLAB®* wird für das **Engineeringsystem** für das Erstellen von TcCOM und SPS-Bausteinen aus MATLAB® 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 TE1401 Lizenz sind folgende Einschränkungen zu beachten:

- Alle cpp und header files vom MATLAB® Coder™ dürfen insgesamt nicht größer als 50 kB sein.
- Funktionseingänge und Funktionsausgänge sind auf 5 Variablen limitiert.
- Sie können nicht mehrere MATLAB®-Funktionen zusammen in eine SPS-Bibliothek überführen.

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

Laufzeitlizenz

Die Lizenzen TC1320 oder TC1220 mit inkludierter SPS-Lizenz, werden benötigt, um eine TwinCAT-Konfiguration mit einem aus MATLAB® generierten TwinCAT-Objekt zu starten. Ohne aktivierte Lizenz kann das Modul und damit auch das TwinCAT-System nicht gestartet werden.

TC1320 enthält die Lizenz zum Ausführen von TwinCAT C++-Objekten sowie Objekten, die über das Target for Simulink® und über das Target for MATLAB® erstellt worden sind.

TC1220 ergänzt obige Liste der TC1320 um eine SPS-Lizenz.

Man kann für die Laufzeitlizenzen eine 7-Tage-Testlizenz erzeugen, die erste Tests ohne den Kauf der Lizenz ermöglicht.

5 Quickstart

Starten mit einer einfachen MATLAB®-Funktion

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

1. Wählen Sie als erstes einfache Samples aus, z. B. *BaseStatistics* – auch direkt aufrufbar mit `TwinCAT.ModuleGenerator.Samples.Start('BaseStatistics').`

Einsteigervideo

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

Einstieg in das Base Statistics Sample

✓ Durch Öffnen des Base Statistics Sample öffnet sich ein MATLAB Live Script, welches sowohl Dokumentationsteile als auch Sections mit Code zur Ausführung beinhaltet.

1. Führen Sie die Code Sections durch Klicken auf die jeweils eingeblendeten Run-Buttons aus.
2. Arbeiten Sie sich so Schritt für Schritt durch das Sample.

⇨ Das Bespiel zeigt, wie Sie zwei MATLAB®-Funktionen mit dem Target for MATLAB® in zwei TcCOM-Objekte und zwei Funktionsbausteine überführen können, wobei die Funktionsbausteine in einer gemeinsamen SPS-Bibliothek gebündelt werden.

Sollten Sie keine gültige TE1401 Lizenz besitzen, können Sie im Sample die Checkbox TrialLicense aktivieren. Dadurch wird nur eine MATLAB®-Funktion in ein TcCOM und einen Funktionsbaustein überführt. Das Beispiel ist dann konform mit den Demoversionsbedingungen [▶ 20] des Produkts.

Auswahl der Komponenten und Pfade

Im Bereich *General Preparation* des Beispiels wird angegeben:

Wie soll der erstellte TwinCAT-Treiber (tmx-file) heißen?

Hier wird *Tc3_BaseStatistics* gewählt.
Dieser Name wird dann verwendet an folgenden Stellen:

- Dateipfad im Engineering Repository:
  %TwinCATInstallDir% \3.1\Repository\<TE140x Module Vendor>**Tc3_BaseStatistics**\<Version>\
- Name der erstellen Dateien *.tmx, *.tml, *.tmc und *.library
- Name der erstellten SPS-Bibliothek in TwinCAT, welche dann die zwei Funktionsbausteine enthält.

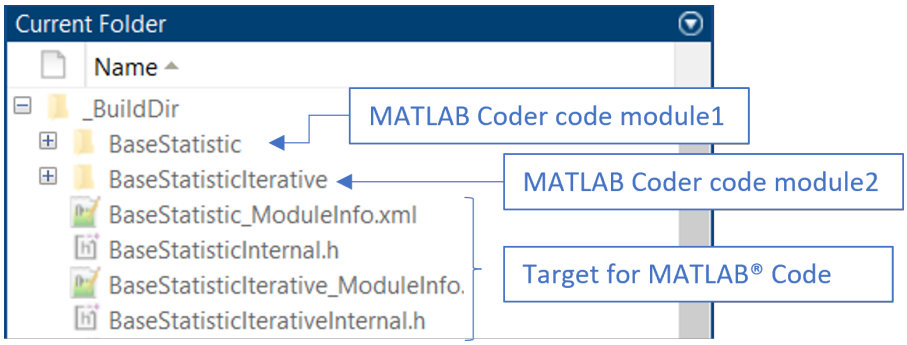
Wo sollen alle Source Dateien abgelegt werden?

Als `buildDir` wird angegeben, wo der MATLAB® Coder™ und auch das TwinCAT Target for MATLAB® alle Source Dateien, log-Files und weitere Metainformationsdateien ablegen soll. Dieser Ordner enthält alle Informationen, um von hier ausgehend die TwinCAT-Objekte zu erstellen. In diesem Fall wird ein neuer Ordner `_ buildDir` im aktuellen MATLAB®-path angelegt.

Welche MATLAB®-Funktionen sollen in TwinCAT verfügbar gemacht werden?

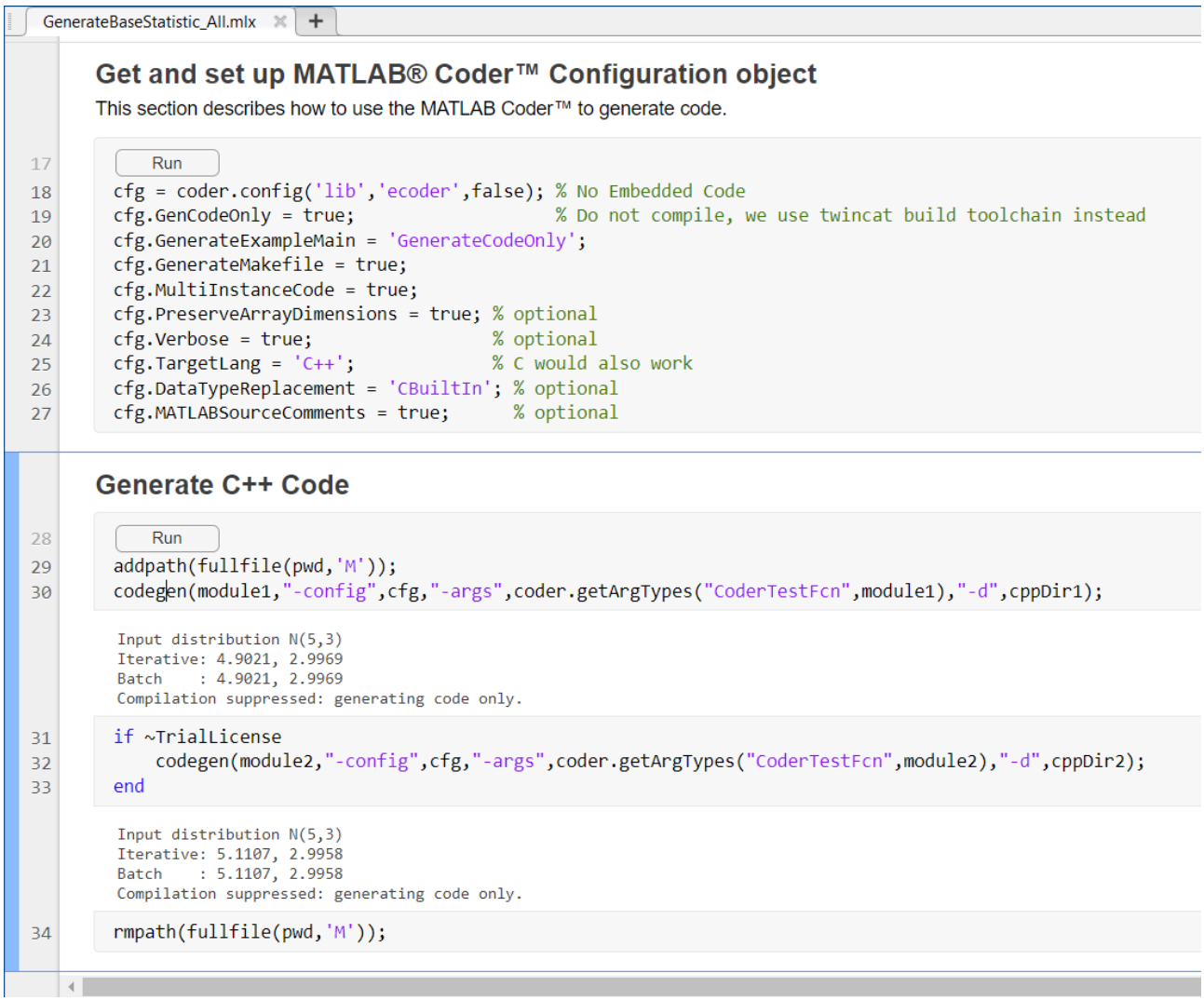
Die MATLAB®-Funktionen werden hier mit den Variablen `module1` und `module2` benannt; entsprechend sollen die zwei MATLAB®-Funktionen BaseStatistics und BaseStatisticsIteravtive (hinterlegt im Subordner M) in TwinCAT-Objekte überführt werden.

Jedes dieser Module bekommt einen eigenen Subordner im `buildDir` welcher mit `cppDir1` und `cppDir2` bezeichnet wird. In diese Subordner wird später der C++-Code vom MATLAB® Coder™ generiert.



Erstellen einer MATLAB® Coder™ Konfiguration

Im weiteren Verlauf des MATLAB® Live Script wird eine MATLAB® Coder™-Konfiguration erstellt. Diese Sektion enthält keine TwinCAT-spezifischen Bestandteile, d. h. es wird ausschließlich der MATLAB® Coder™ verwendet. Eine ausführliche Dokumentation zum MATLAB® Coder™ finden Sie in der MATLAB®-Dokumentation.



Bei der Erstellung der Coder-Konfiguration `cfg` ist zu beachten:

- Der Embedded Coder wird nicht unterstützt.
- Nur der generierte Code wird benötigt.

Dem `codegen` Befehl wird dann die Coder-Konfiguration und die entsprechende MATLAB®-Funktion, die übersetzt werden soll, übergeben. Mit dem Argument “-d“, cppDir1 wird eingestellt, dass der MATLAB® Coder™ den C++-Code in den Pfad cppDir1 ablegt.

Entsprechend liegen nach diesem Schritt der generierte C++-Code der Funktion BaseStatistic.m und BaseStatisticIterarive.m in den Ordnern _BuildDir/ BaseStatistic und _BuildDir\ BaseStatisticIterative.

Erstellen einer Target for MATLAB® Projekt-Export-Konfiguration

Die folgenden Code-Segmente im MATLAB® Live Script betreffen ausschließlich das Target for MATLAB® und sind insofern unabhängig vom MATLAB® Coder™, als dass nur noch mit dem durch den MATLAB® Coder™ bereits erstellen C++-Code gearbeitet wird.

Optional können Sie die MATLAB®-Code-Beschreibung aus dem m-File extrahieren und diese später im TwinCAT XAE anzeigen lassen, siehe MATLAB Code im TcCOM [▶ 43].

```
TwinCAT.ModuleGenerator.Matlab.ExportMCodeRepresentation('MFile',module1,'BuildDir',cppDir1);
```

Das M-File mit der Funktion module1, also BaseStatistics, muss dazu im MATLAB® Workspace liegen. Die Informationen werden dann aus dem m-File BaseStatistics.m extrahiert und im Ordner cppDir1 abgelegt.

Im nächsten Schritt wird eine Projekt-Export-Konfiguration durch den TwinCAT-Modulgenerator angelegt mit

```
TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',FullPathToVcxproj);
```

Als Argument wird der Fullpath zum neu zu erstellenden Visual Studio Projekt angegeben. In diesem Fall … _BuildDir\Tc3_BaseStatistics.vcxproj. Die Benennung des Visual Studio Projekts definiert nach dem Build auch die Benennung der erstellten Dateien *.tmx, *.tmc, *.tml und *.library. Siehe dazu Generelle Einstellungen BaseStatistics [▶ 21].

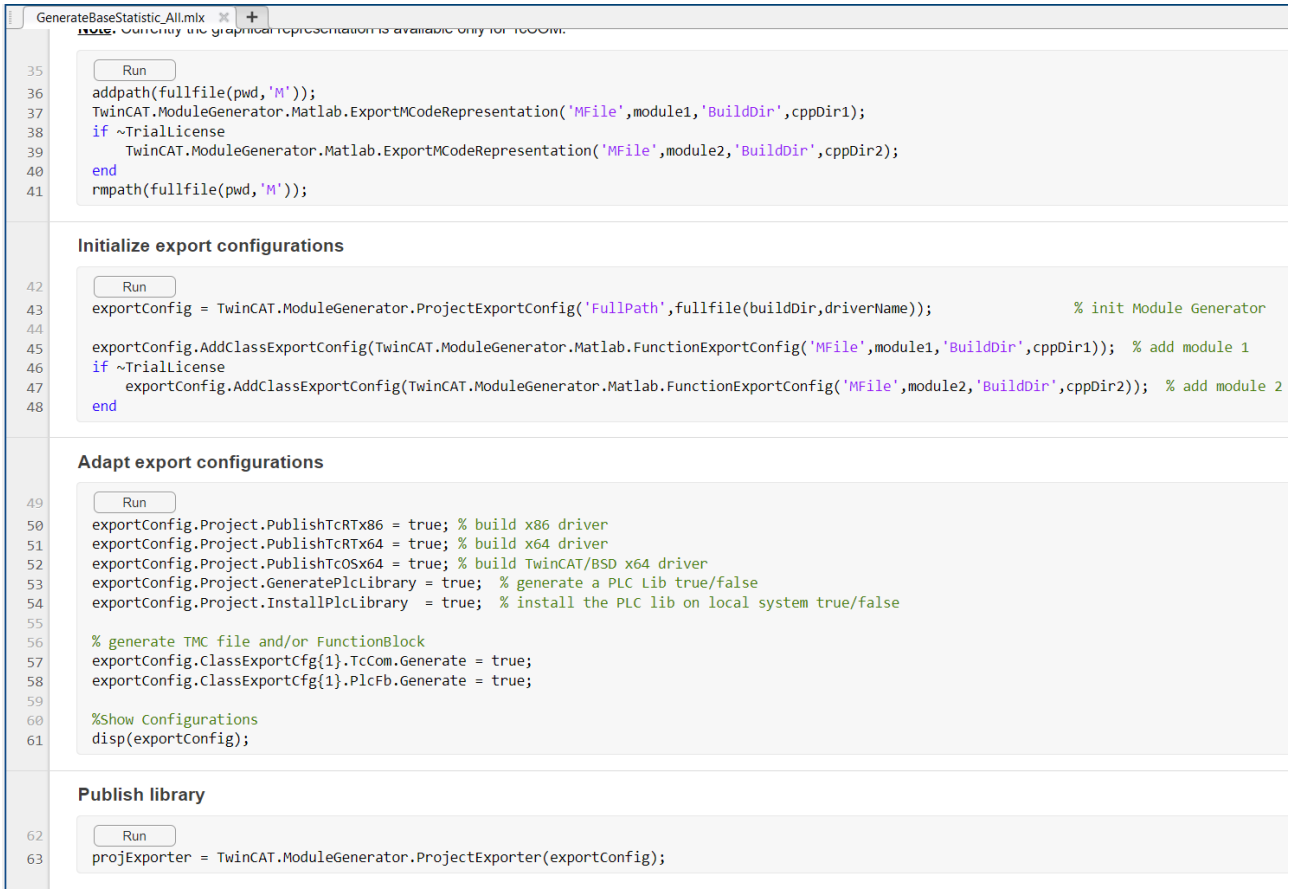


Abb. 1:

Für jede MATLAB®-Funktion muss eine Export-Konfiguration erstellt werden mit `TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig()`. Mit AddClassExportConfig wird diese Export-Konfiguration zur Projekt-Export-Konfiguration als Modul hinzugefügt.

```
exportConfig.AddClassExportConfig(TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig('MFile',module
1,'BuildDir',cppDir1));
```

Als Argument zum FunctionExportConfig() wird einmal der Pfad zum vom MATLAB® Coder™ erstellten C++-Code und der Name des m-Files mit entsprechender MATLAB®-Funktion übergeben. Beispielsweise mit module1 wird dann gesetzt, dass das zu erstellende TcCOM-Objekt und der Funktionsbaustein in der SPS-Bibliothek BaseStatistics bzw. FB_BaseStatistics heißen.

Die Projekt-Export-Konfiguration wird im Folgenden noch weiter adaptiert. So werden die Plattformen definiert, für die ein Treiber gebaut werden soll (hier für Windows 32bit, Windows 64 bit und TwinCAT/BSD® 64bit). Ebenso wird konfiguriert, dass eine SPS-Bibliothek erstellt und auch auf dem lokalen TwinCAT XAE installiert werden soll.

Für jedes Modul, welches zur Projekt-Export-Konfiguration hinzugefügt wurde, können individuell Eigenschaften gesetzt werden. Hier wird explizit gesetzt, dass für das erste hinzugefügte Modul sowohl ein SPS-Funktionsbaustein als auch ein TcCOM erzeugt werden soll.

Mit `disp(exportConfiguration)` können Sie sich die gesamte Konfiguration im Überblick anzeigen lassen.

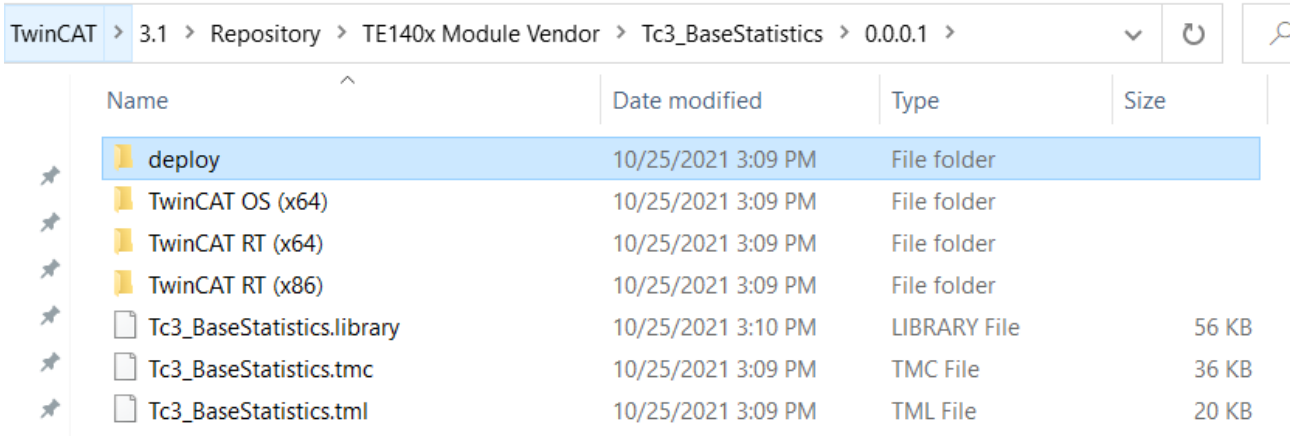
	Value	DataType	Options	DisplayNam
Project.FullPath	{["_BuildDir\Tc3_BaseStatistics"]}	"String"	""	"TwinCAT C++ Project Path"
Project.VendorName	{'TE140x Module Vendor' }	"String"	""	"VendorName"
Project.VersionSrc	{'$<LatestTMFile>' }	"String"	""	"Version source file"
Project.IncrementVersion	{'Revision' }	"Enum"	"None, Revision, Build, Minor, Major"	"Version part for increment"
Project.DrvFileVersion	{'$<VersionFromFile>' }	"String"	""	"DrvFileVersion"
Project.LowestCompatibleTcBuild	{'$<TwinCAT:Version:BUILD>' }	"Int"	""	"Lowest compatible TwinCAT vers
Project.MaxVisibleArrayElements	{'200U' }	"String"	""	"Maximum number of visible arra
Project.Publish	{'true' }	"Bool"	""	"Run the publish step after pro
Project.PublishPlatformtoolset	{'Auto' }	"Enum"	"Auto, Microsoft Visual C++ 14.1"	"Platform Toolset"
Project.PublishConfiguration	{'Release' }	"Enum"	"Release, Debug"	"Build configuration"
Project.PublishTcRTx86	{[1]}	"Bool"	""	"TwinCAT RT (x86)"
Project.PublishTcRTx64	{[1]}	"Bool"	""	"TwinCAT RT (x64)"
Project.PublishTcOSx64	{[1]}	"Bool"	""	"TwinCAT OS (x64)"
Project.SignTwinCatCertName	{0×0 char }	"String"	""	"Certificate name for TwinCAT s
Project.GeneratePlcLibrary	{[1]}	"Bool"	""	"Generate a PLC library"
Project.InstallPlcLibrary	{[1]}	"Bool"	""	"Install the generated PLC libr
Project.PreCodeGenerationCallbackFcn	{0×0 char }	"String"	""	"Pre code generation callback f
Project.PostCodeGenerationCallbackFcn	{0×0 char }	"String"	""	"Post code generation callback
Project.PostPublishCallbackFcn	{0×0 char }	"String"	""	"Post publish callback function
****** ClassExportCfg{1} (BaseStatistic) ******	{["************"]}	"************"	"************"	"************"
ClassExportCfg{1}.TcCom.Generate	{[1]}	"Bool"	""	"Generate TcCom Module (TwinCAT
ClassExportCfg{1}.TcCom.ClassName	{["BaseStatistic"]}	"String"	"<READONLY>"	"TcCOM module name"
ClassExportCfg{1}.TcCom.FpExceptionsForInit	{'CallerExceptions' }	"Enum"	"CallerExceptions, Exceptions, NoExceptions"	"Floatingpoint exception during
ClassExportCfg{1}.TcCom.FpExceptionsForUpdate	{'CallerExceptions' }	"Enum"	"CallerExceptions, Exceptions, NoExceptions"	"Floatingpoint exception during
ClassExportCfg{1}.TcCom.OnlineChange	{'false' }	"Bool"	""	"Online change support"
ClassExportCfg{1}.TcCom.GroupName	{'TE140x|MATLAB Modules' }	"String"	""	"GroupName"
ClassExportCfg{1}.TcCom.BlockDiagramExport	{'true' }	"Bool"	""	"Export BlockDiagram"
ClassExportCfg{1}.TcCom.ResolveMaskedSubsystems	{'false' }	"Bool"	""	"Resolve Masked Subsystems"
ClassExportCfg{1}.TcCom.BlockDiagramVariableAccess	{'AssignToParent' }	"Enum"	"AssignToParent, HideInBlockDiagram"	"Access to VariableGroup not re
ClassExportCfg{1}.TcCom.BlockDiagramDebugInfoExport	{'true' }	"Bool"	""	"Export BlockDiagram debug info
ClassExportCfg{1}.TcCom.MonitorExecutionTime	{'false' }	"Bool"	""	"Monitor ExecutionTime"
ClassExportCfg{1}.TcCom.InputInitValues	{'false' }	"Bool"	""	"Input: Initial values"

Sie erhalten dadurch einen Überblick über die gesetzten Werte (Value), über den Datentypen der verwendet wird (DataType), Wert-Vorschläge (Options) und eine Kurzbeschreibung (Displayname).

Mit `TwinCAT.ModuleGenerator.ProjectExporter(exportConfig)` wird der Build-Prozess der konfigurierten Plattformen angestoßen. Dadurch wird auf dem lokalen Dateisystem im Repository ein Ordner angelegt und die erstellten Treiber und Beschreibungsdateien abgelegt.

Die Pfadbeschreibung lautet:

%TwinCATInstallDir% \3.1\Repository\< VendorName >\<ProjectName>\<Version>

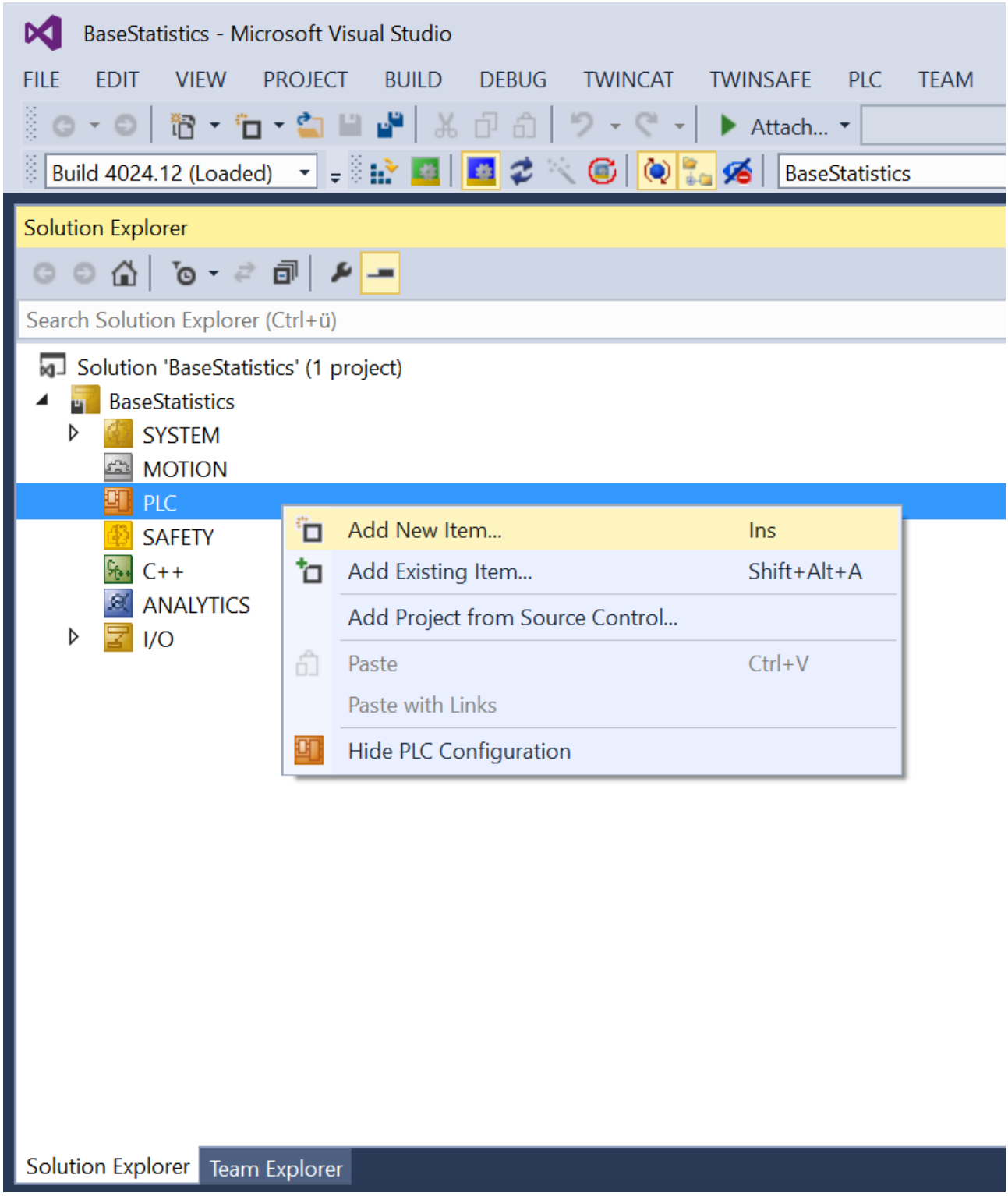


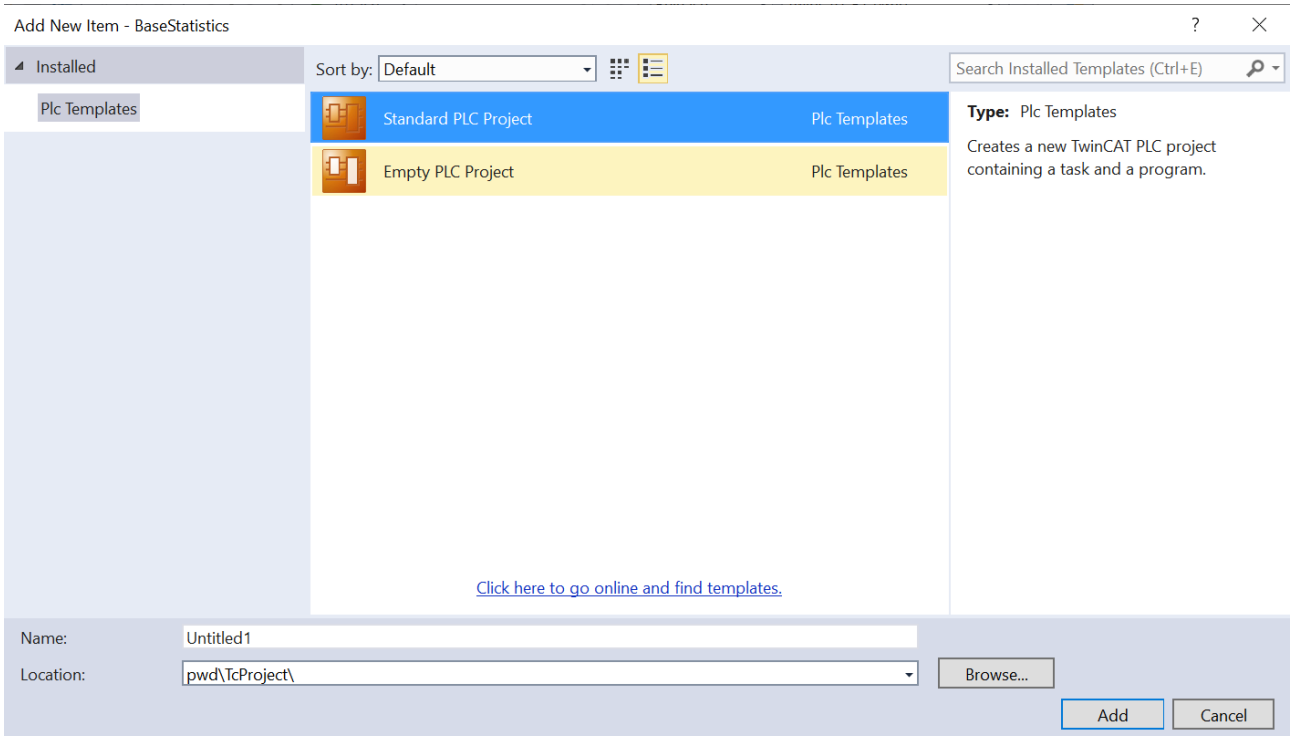
Den Ordner können Sie auf beliebig viele TwinCAT XAE Systeme kopieren, damit die Module dort verfügbar sind. Einzig die *.library müssen Sie über das PLC library repository in TwinCAT installieren.
Beachten Sie dabei, dass die Ordnerstruktur beim Kopieren nicht verändert wird.

SPS-Bibliothek in TwinCAT 3 verwenden

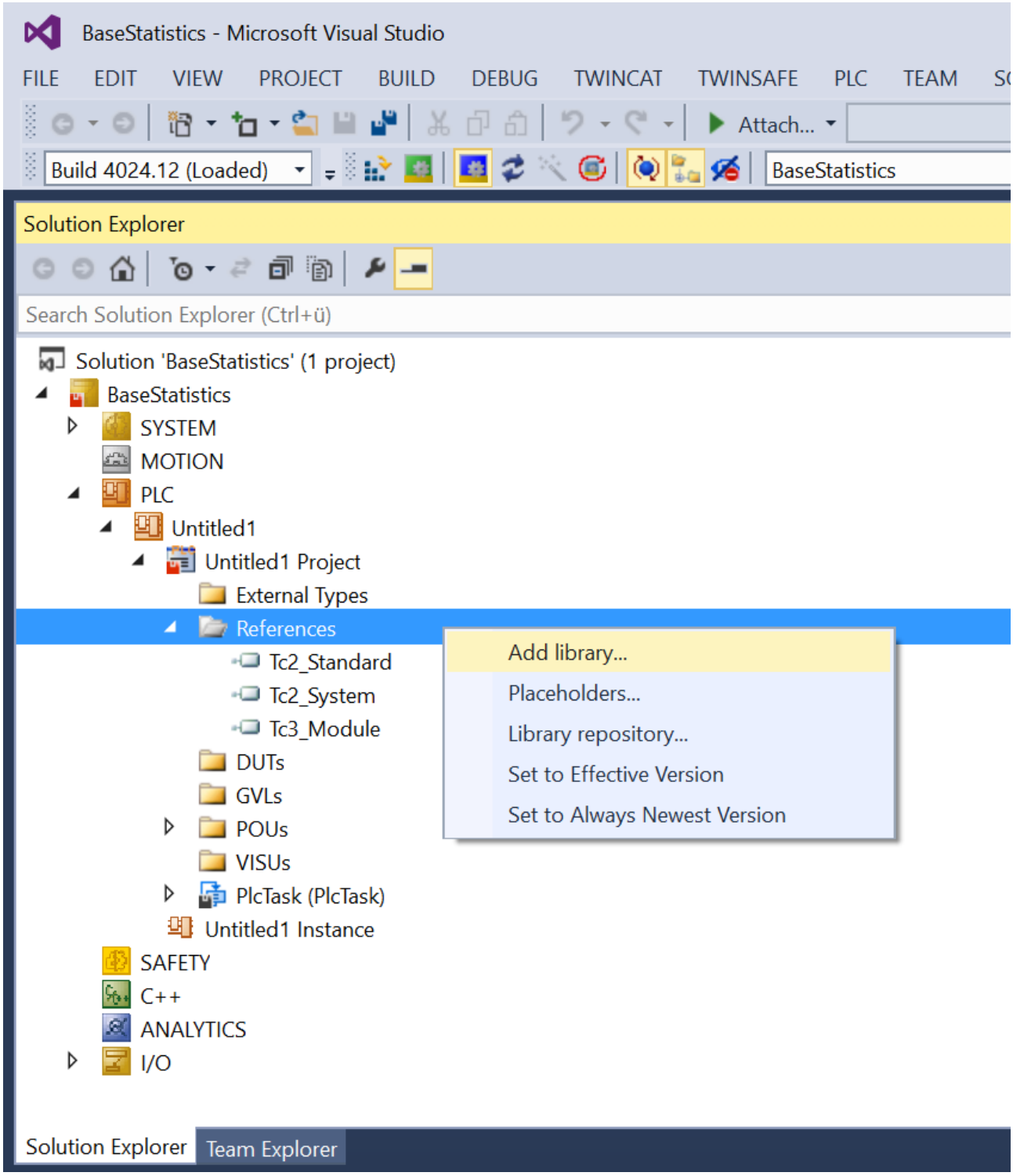
✓ Ausgehend von einer neuen TwinCAT Solution, erstellen Sie ein SPS Projekt:

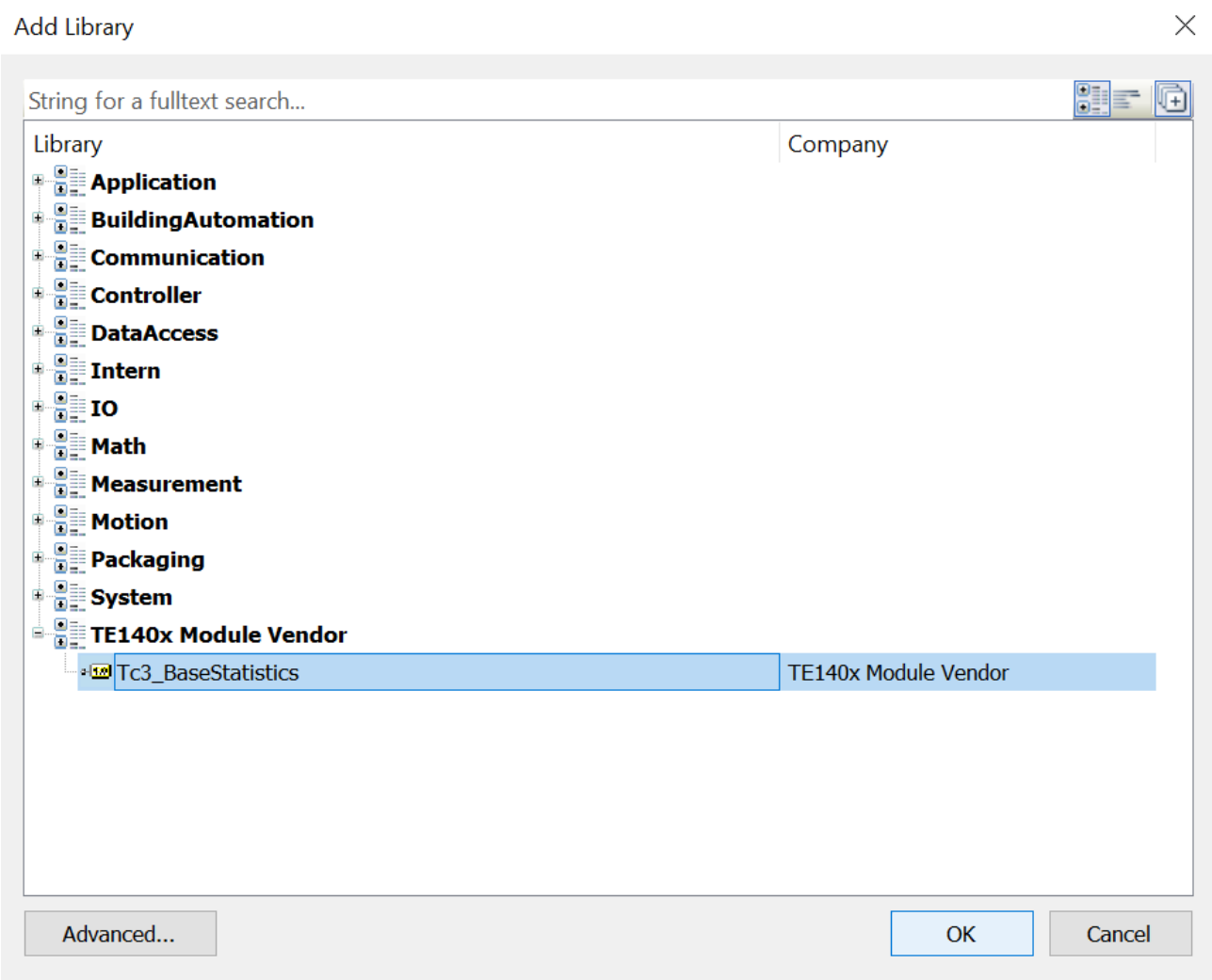
1. Führen Sie die folgenden Menüschritte aus.


BaseStatistics - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TWINCAT TWINSAFE PLC TEAM
Attach...
Build 4024.12 (Loaded)
BaseStatistics
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'BaseStatistics' (1 project)
BaseStatistics
SYSTEM
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
Add New Item... Ins
Add Existing Item... Shift+Alt+A
Add Project from Source Control...
Paste Ctrl+V
Paste with Links
Hide PLC Configuration
Solution Explorer
Team Explorer

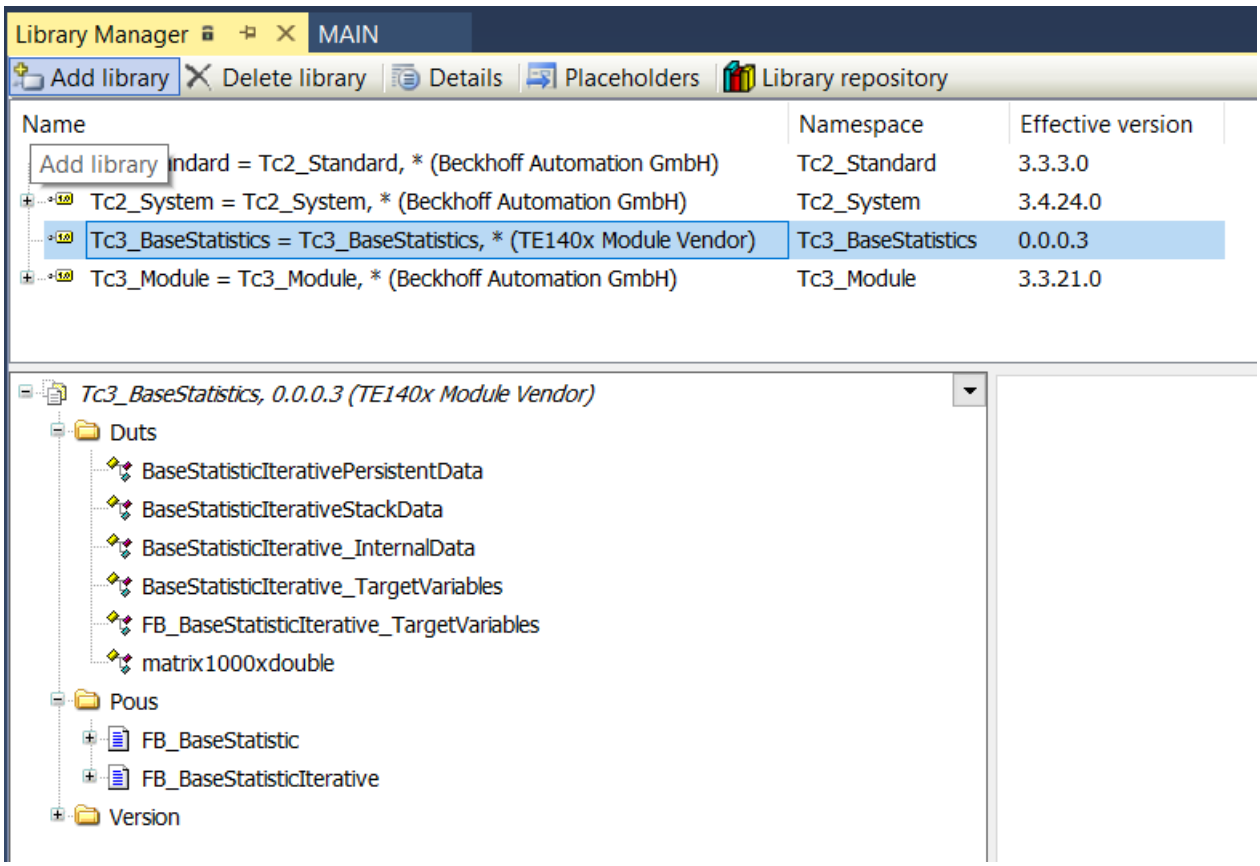


2. Fügen Sie danach die neu erstellte (und schon installierte) SPS Bibliothek hinzu:

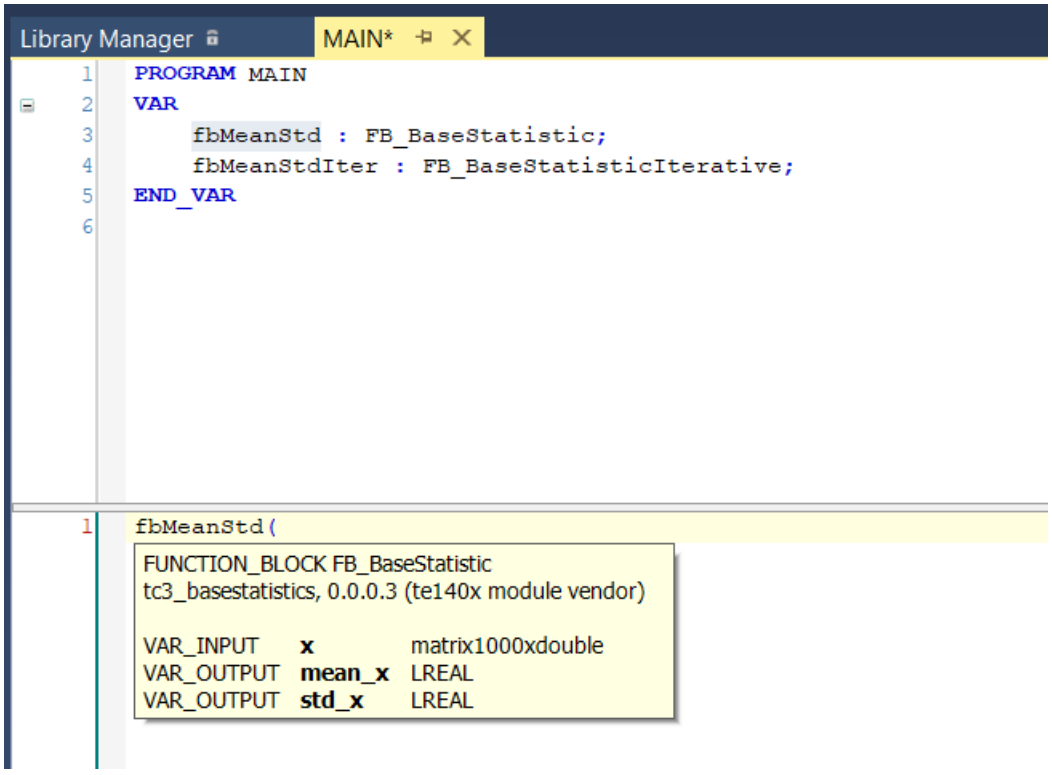

BaseStatistics - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TWINCAT TWINSAFE PLC TEAM
Attach...
Build 4024.12 (Loaded)
BaseStatistics
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'BaseStatistics' (1 project)
BaseStatistics
SYSTEM
MOTION
PLC
Untitled1
Untitled1 Project
External Types
References
Tc2_Standard
Tc2_System
Tc3_Module
DUTs
GVLs
POUs
VISUs
PlcTask (PlcTask)
Untitled1 Instance
SAFETY
C++
ANALYTICS
I/O
Add library...
Placeholders...
Library repository...
Set to Effective Version
Set to Always Newest Version
Solution Explorer
Team Explorer



3. Machen Sie sich einen Überblick über die Datentypen und Funktionsbausteine:

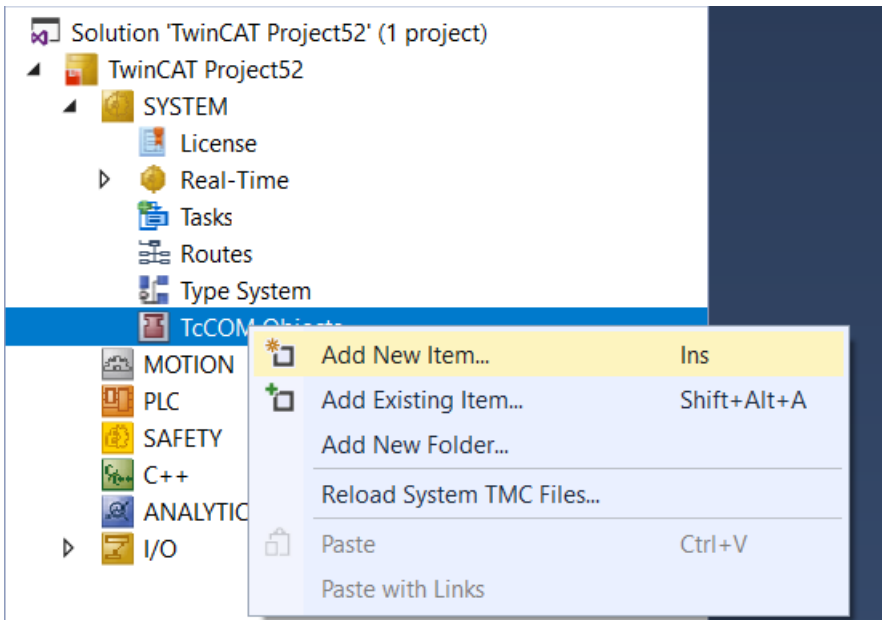


4. Fügen Sie Instanzen der Funktionsbausteine in Ihre SPS ein und nutzen Sie diese in Ihrer Applikation:

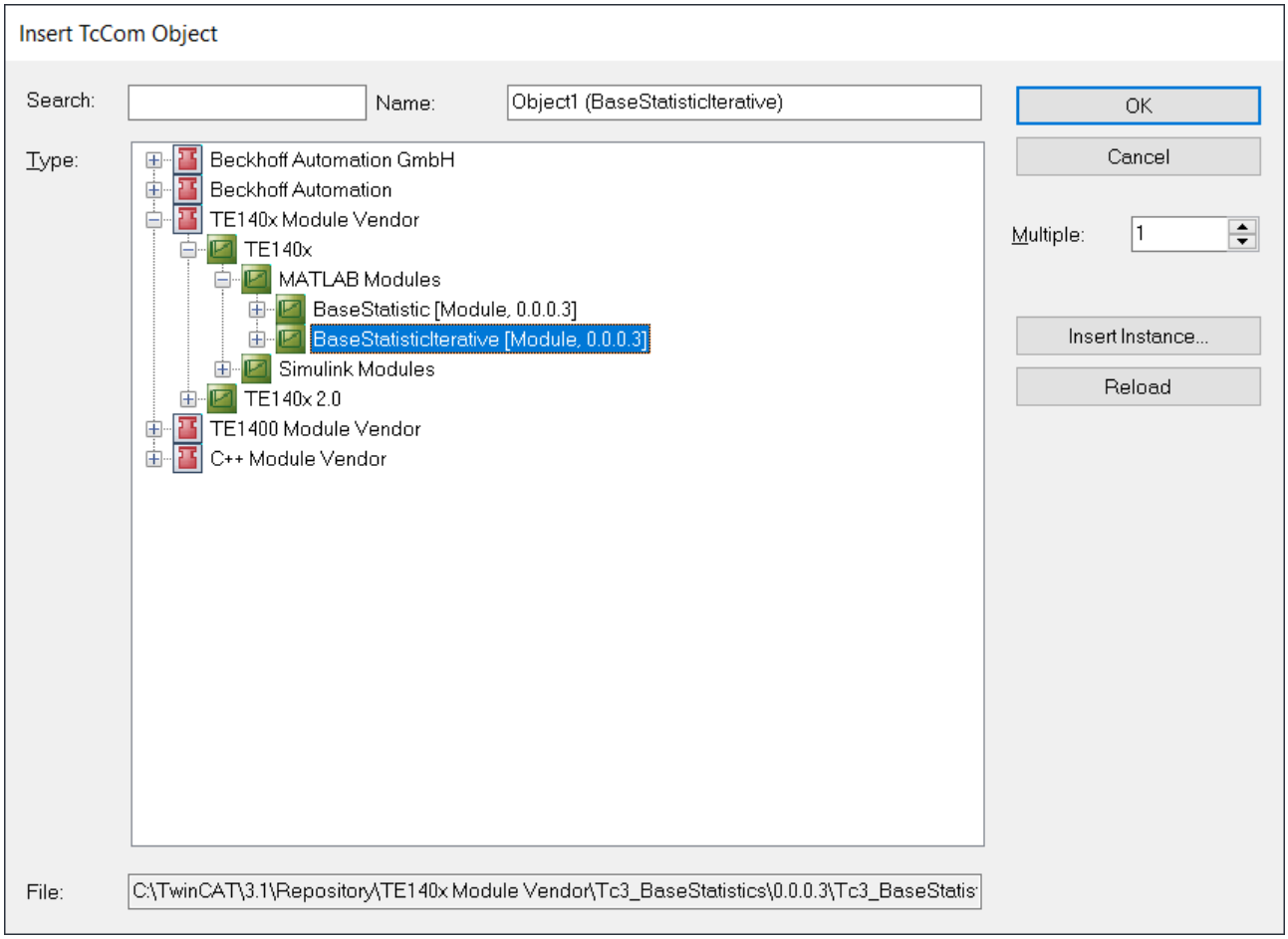


TcCOM-Objekte in TwinCAT 3 verwenden

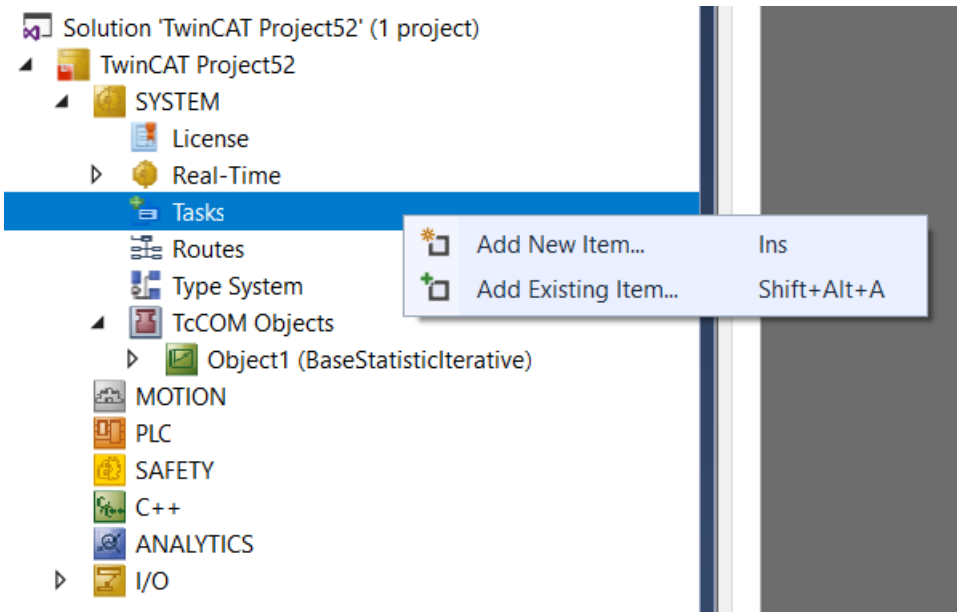
1. Fügen Sie ein neues TcCOM-Objekt ein.



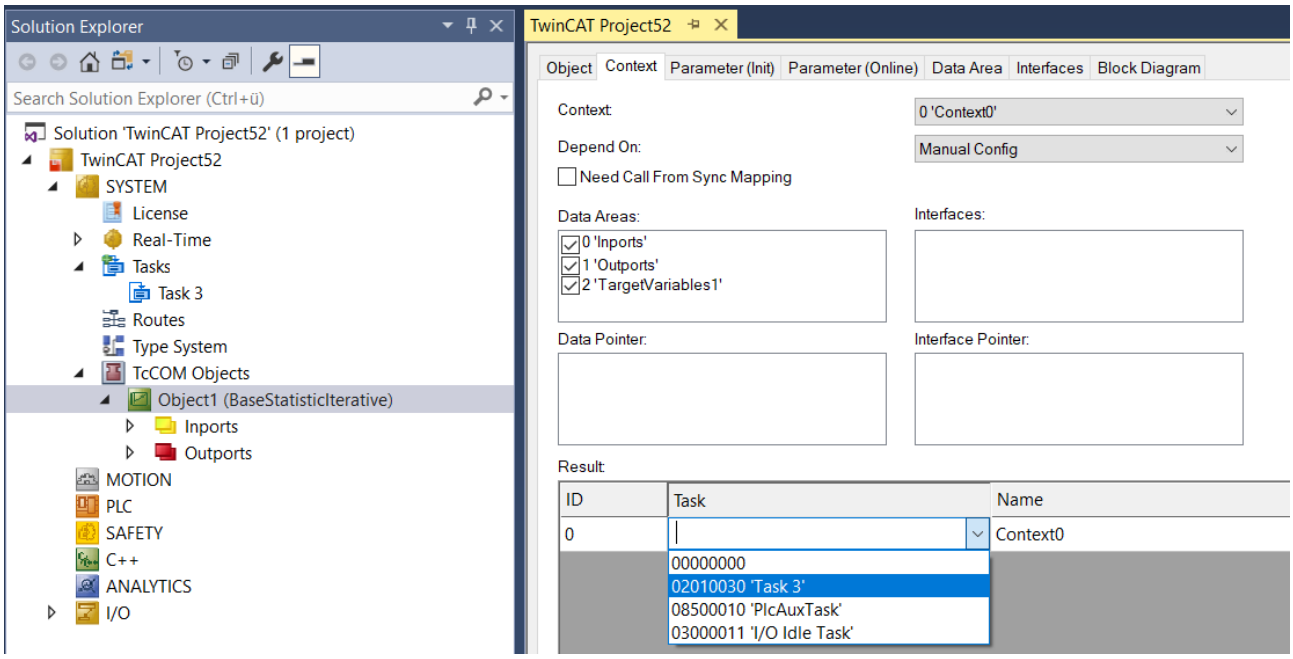
2. Wählen Sie das entsprechende TcCOM-Objekt aus:



3. Erstellen Sie eine neue zyklische Task vom Typ *TwinCAT Task.*



4. Weisen Sie dem neu erstellen TcCOM-Objekt die neu erstelle Task zu. Gehen Sie dazu auf die Instanz des TcCOM-Objekts und wählen Sie den Reiter *Context*.



⇨ Sie können nun die Konfiguration aktivieren. Um das TcCOM-Objekt vorher mit anderen Modulen in ihrer TwinCAT Solution zu verbinden, können Sie das Prozessabbild verwenden, um Mappings zu erstellen.

Sie können den MATLAB® Code unter dem Reiter Block Diagram einsehen und im laufenden Betrieb Werte beobachten und scopen. Siehe dazu MATLAB Code Darstellung [▶ 43].

6 Übersicht zu automatisch generierten Dateien

Wenn ein Build-Prozess über den TwinCAT-Modulgenerator angestoßen wird, werden einige Dateien und Ordner automatisch erzeugt. Wo liegen die Dateien, was kann damit gemacht werden und was bedeuten die Dateien – das wird im Folgenden beschrieben.

Welche Kategorien an automatisch generierten Dateien gibt es?

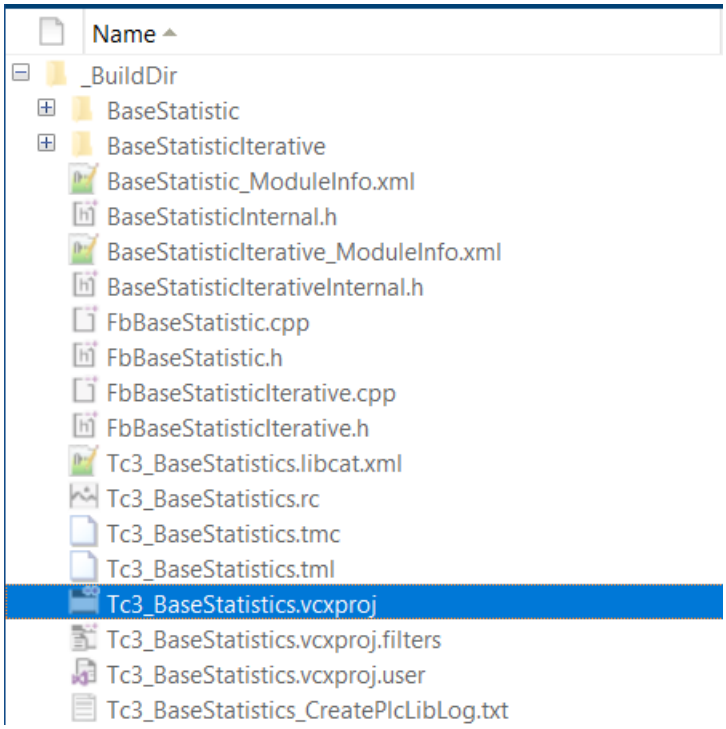
- Es wird Source Code generiert.
- Es werden Log-Files generiert.
- Es werden die TwinCAT-Objekte, Treiber (*.tmx) und Beschreibungsdateien (*.tmc, *.library, …), erzeugt.

Generierter Source Code

Alle Source-Dateien, die zum Build, also zum Erstellen der TwinCAT-Objekte, notwendig sind, werden in dem Ordner abgelegt, der bei der Initialisierung des TwinCAT-Modulgenerators angegeben wird. Hier werden präzise der Ort und Name des zu generierenden Visual Studio Projekts angegeben.

```
TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',FullPathToVcxproj);
```

In folgender Grafik ist z. B. der FullPath als *…_BuildDir\Tc3_BaseStatistics* angegeben.



Zentrale Datei für den Source Code ist die *<ProjectName>.vcxproj*. Diese Datei kann genutzt werden, um alle TwinCAT-Objekte zu erstellen. So kann zum Beispiel aus MATLAB® nur die Code-Generierung ohne Build-Prozess angestoßen und der Build-Prozess auf einem anderen System, z. B. einem Build-Server, ausgeführt werden. Setzen Sie im TwinCAT-Modulgenerator dazu `Project.Publish = false`.

Generierte Logfiles

Die generierten Logfiles werden ebenfalls in oben genannten Ordner zusammengefasst.

Die erstellen Logfiles sind die erste Anlaufstelle beim Debugging. Wenn Sie Hilfe bei unserem Support in Anspruch nehmen, senden Sie bitte immer folgende Datei mit:

<ModelName>_ModuleGenerationLog.txt

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\<Module Vendor>\<ProjectName>\<Version>

In diesem Ordner liegen die tmc-Beschreibungsdatei, die SPS-Bibliothek und die tmx-Treiber für die konfigurierten Plattformen sowie weitere Beschreibungsdateien.

Wenn der Order 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.

Zu beachten

Beschreibung der generieten C++ Dateien und Binärdateien

Versionierte C++ Projekte

7 Einstellungen des TwinCAT-Modulgenerators

Erstellen einer Projekt-Export-Konfiguration

```
exportConfig = TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',FullPathToVSproj);
```

Dem Property 'FullPath' wird der vollständige Pfad und Name des zu erstellenden Visual Studio-Projekts übergeben.

Rückgabe ist ein Objekt der Klasse TwinCAT.ModuleGenerator.ProjectExportConfig.

Beispielaufruf:

```
FullPathToVSproj = 'C:\BuildDir\MyMATLABFcn';
exportConfig = TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',FullPathToVSproj);
```

Methoden der Klasse TwinCAT.ModuleGenerator.ProjectExportConfig

`AddClassExportConfig`	Fügt eine Export-Konfiguration zum Projekt hinzu. Zur Erstellung einer Export-Konfiguration siehe Abschnitt Erstellen einer Export Konfiguration [▶ 39]. Die Export-Konfiguration wird unter Properties im Cell-Array ClassExportCfg angehängt. Beispielaufruf siehe Modul-Generator-Quickstart [▶ 23].
`Save`	Erstellt ein mat-File und speichert darin die Projekt-Export-Konfiguration. Übergabeargument ist ein Pfad, in dem das mat-File abgelegt werden soll. Beispielaufruf: `exportConfig.Save(pwd)` Speichert die Projekt-Export-Konfiguration im aktuellen Pfad.
`Load`	Lädt eine gespeicherte Projekt-Export-Konfiguration. Übergabeargument ist der Pfad, in dem das mat-File mit der gespeicherten Konfiguration liegt. Beispielaufruf: `exportConfig.Load(pwd)` Lädt die Projekt-Export-Konfiguration aus dem aktuellen Pfad.
`Edit`	Öffnet eine grafische Konfigurationsoberfläche zur Konfiguration der Projekt-Export-Konfiguration.
`disp`	Gibt eine Übersicht im MATLAB® Command Window über die aktuelle Projekt-Export-Konfiguration. Siehe Modul-Generator-Quickstart [▶ 23]. Beispielaufruf: `exportConfig.disp` alternativ `disp(exportConfig)`

	Value	DataType	Options	DisplayNam
Project.FullPath	{["_BuildDir\Tc3_BaseStatistics"]}	"String"	""	"TwinCAT C++ Project Path"
Project.VendorName	{'TE140x Module Vendor' }	"String"	""	"VendorName"
Project.VersionSrc	{'$<LatestTMFile>' }	"String"	""	"Version source file"
Project.IncrementVersion	{'Revision' }	"Enum"	"None, Revision, Build, Minor, Major"	"Version part for increment"
Project.DrvFileVersion	{'$<VersionFromFile>' }	"String"	""	"DrvFileVersion"
Project.LowestCompatibleTcBuild	{'$<TwinCAT:Version:BUILD>' }	"Int"	""	"Lowest compatible TwinCAT vers
Project.MaxVisibleArrayElements	{'200U' }	"String"	""	"Maximum number of visible arra
Project.Publish	{'true' }	"Bool"	""	"Run the publish step after pro
Project.PublishPlatformtoolset	{'Auto' }	"Enum"	"Auto, Microsoft Visual C++ 14.1"	"Platform Toolset"
Project.PublishConfiguration	{'Release' }	"Enum"	"Release, Debug"	"Build configuration"
Project.PublishTcRTx86	{[1]}	"Bool"	""	"TwinCAT RT (x86)"
Project.PublishTcRTx64	{[1]}	"Bool"	""	"TwinCAT RT (x64)"
Project.PublishTcOSx64	{[1]}	"Bool"	""	"TwinCAT OS (x64)"
Project.SignTwinCatCertName	{0×0 char }	"String"	""	"Certificate name for TwinCAT s
Project.GeneratePlcLibrary	{[1]}	"Bool"	""	"Generate a PLC library"
Project.InstallPlcLibrary	{[1]}	"Bool"	""	"Install the generated PLC libr
Project.PreCodeGenerationCallbackFcn	{0×0 char }	"String"	""	"Pre code generation callback f
Project.PostCodeGenerationCallbackFcn	{0×0 char }	"String"	""	"Post code generation callback
Project.PostPublishCallbackFcn	{0×0 char }	"String"	""	"Post publish callback function
****** ClassExportCfg{1} (BaseStatistic) ******	{["************"]}	"************"	"************"	"************"
ClassExportCfg{1}.TcCom.Generate	{[1]}	"Bool"	""	"Generate TcCom Module (TwinCAT
ClassExportCfg{1}.TcCom.ClassName	{["BaseStatistic"]}	"String"	"<READONLY>"	"TcCOM module name"
ClassExportCfg{1}.TcCom.FpExceptionsForInit	{'CallerExceptions' }	"Enum"	"CallerExceptions, Exceptions, NoExceptions"	"Floatingpoint exception during
ClassExportCfg{1}.TcCom.FpExceptionsForUpdate	{'CallerExceptions' }	"Enum"	"CallerExceptions, Exceptions, NoExceptions"	"Floatingpoint exception during
ClassExportCfg{1}.TcCom.OnlineChange	{'false' }	"Bool"	""	"Online change support"
ClassExportCfg{1}.TcCom.GroupName	{'TE140x|MATLAB Modules' }	"String"	""	"GroupName"
ClassExportCfg{1}.TcCom.BlockDiagramExport	{'true' }	"Bool"	""	"Export BlockDiagram"
ClassExportCfg{1}.TcCom.ResolveMaskedSubsystems	{'false' }	"Bool"	""	"Resolve Masked Subsystems"
ClassExportCfg{1}.TcCom.BlockDiagramVariableAccess	{'AssignToParent' }	"Enum"	"AssignToParent, HideInBlockDiagram"	"Access to VariableGroup not re
ClassExportCfg{1}.TcCom.BlockDiagramDebugInfoExport	{'true' }	"Bool"	""	"Export BlockDiagram debug info
ClassExportCfg{1}.TcCom.MonitorExecutionTime	{'false' }	"Bool"	""	"Monitor ExecutionTime"
ClassExportCfg{1}.TcCom.InputInitValues	{'false' }	"Bool"	""	"Input: Initial values"

Properties der Klasse TwinCAT.ModuleGenerator.ProjectExportConfig

`Project`

Struktur mit Einträgen zur Konfiguration der Build-Eigenschaften, der SPS-Bibliothek und der möglichen Callbacks.

`Project.FullPath`

FullPath zum zu erstellenden TwinCAT C++-Projekt.

`Project.VendorName`

Bezeichnung des Vendors. Der Vendor Name dient zur Strukturierung der TwinCAT-Objekte. Der Vendor wird im Pfad des Repositories als Ordner angelegt und ist in der Struktur beim Einfügen der SPS-Bibliothek und des TcCOM-Objekt sichtbar.

`Project.IncrementVersion`

Mögliche Werte: "None", "Revision", "Build", "Minor", Major"

Default: "Revision"

Diese Einstellung beeinflusst, an welcher der vier Stellen die Version hochgezählt werden soll. Basis ist die letzte auf dem Engineering System verfügbare Version (siehe `DrvFileVersion`).

`Project.DrvFileVersion`

Default: Sucht nach der letzten Version auf dem Engineering System. Wird keine vorhandene Version gefunden, wird mit 0.0.0.0 begonnen.

Direktes Setzen einer Version: Kann direkt als String gesetzt werden, z. B. "1.52.32.0". `IncrementVersion` wird dann nicht ausgeführt.

`Project.Publish`

Wenn TRUE, dann wird das erstellte TwinCAT C++-Projekt für die konfigurierten Plattformen gebaut.

`Project.PublishPlatformtoolset`

Konfiguriert die zu nutzende Visual Studio Version. Diese kann konkret spezifiziert werden oder auf Auto (Default) gestellt werden.

`Project.PublishTcRTx86`

Wenn TRUE, dann wird für XAR auf einer Windows 32bit Plattform gebaut.

`Project.PublishTcRTx64`

Wenn TRUE, dann wird für XAR auf einer Windows 64bit Plattform gebaut.

`Project.PublishTcOSx64`

Wenn TRUE, dann wird für XAR auf einer TwinCAT/BSD®-Plattform gebaut.

`Project.SignTwinCatCertName`

Hier kann (nicht zwingend) ein TwinCAT OEM-Zertifikat zur Treibersignierung angegeben werden. Das Passwort ist mit dem TcSignTool in die Windows Registry des Users einzutragen. Detail siehe Ersteinrichtung der Software [▶ 12].

`Project.TmxArchive`

Geben Sie hier als String einen Pfad und Dateinamen an, um ein TMX-Archiv [▶ 40] zu erzeugen. Beispiel: Project.TmxArchive = "c:\archives\[Date]-[Time]-[LibName][LibVersion].exe" erzeugt ein selbstextrahierendes TMX-Archiv unter c:\archies. Die Placeholder werden im Modulgenerator vor dem Schreiben der Datei aufgelöst.

`Project.GeneratePlcLibrary`

Wenn TRUE, dann wird eine SPS-Bibliothek (*.library) im Repository Folder für das Projekt erstellt.

`Project.InstallPlcLibrary`

Wenn TRUE, dann wird die erstellte SPS-Bibliothek auf dem lokalen TwinCAT XAE installiert.

`Project.PreCodeGenerationCallbackFcn`

Als String kann hier eine Funktion aufgerufen werden, welche noch vor der Code-Generierung, also der Erstellung des TwinCAT C++-Projekts, aufgerufen wird.

Beispielsweise kann ein m-File MyCallback.m im Workspace mit folgendem Inhalt angelegt werden:

```
function MyCallback(obj)
…
return
```

Das Property PreCodeGenerationCallbackFcn wird dann auf "MyCallback" gesetzt. Der Funktion wird standardmäßig das ProjektExporter-Objekt [▶ 39] übergeben, sodass Sie in der Callback Funktion Zugriff auf alle Daten des aktuellen Projekts haben.

`Project.PostCodeGenerationCallbackFcn`

Als String kann hier eine Funktion aufgerufen werden, welche nach der Code-Generierung, also der Erstellung des TwinCAT C++-Projekts, aufgerufen wird.

Beispielsweise kann ein m-File MyCallback.m im Workspace mit folgendem Inhalt angelegt werden:

```
function MyCallback(obj)
…
return
```

Das Property PostCodeGenerationCallbackFcn wird dann auf "MyCallback" gesetzt. Der Funktion wird standardmäßig das ProjektExporter-Objekt [▶ 39] übergeben, sodass Sie in der Callback Funktion Zugriff auf alle Daten des aktuellen Projekts haben.

`Project.PostPublishCallbackFcn`

Als String kann hier eine Funktion aufgerufen werden, welche nach dem Kompilieren, also der Erstellung der TwinCAT Objekte, aufgerufen wird.

Beispielsweise kann ein m-File MyCallback.m im Workspace mit folgendem Inhalt angelegt werden:

```
function MyCallback(obj)
…
return
```

Das Property PostPublishCallbackFcn wird dann auf "MyCallback" gesetzt. Der Funktion wird standardmäßig das ProjektExporter-Objekt [▶ 39] übergeben, sodass Sie in der Callback Funktion Zugriff auf alle Daten des aktuellen Projekts haben.

`Project.OemId` und `Project.OemLicenses`

Optional kann ein generiertes TcCOM oder ein Funktionsbaustein an eine OEM-Lizenz gebunden werden. Diese OEM-Lizenz wird beim Starten des Objekts (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**.

Ihren OEM Lizenzcheck können Sie 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.

Beispieleintrag:

```
exportConfig.Project.OemId = '{ABBAABBA-AFFE-AFFE-AFFE-ABBABBAABBAA}';

exportConfig.Project.OemLicenses = '{11111111-0000-FEFE-CCCC-BBBBBBBBBBBB}';
```

`ClassExportCfg`

Cell-Array der hinzugefügten Export-Konfigurationen. Jede Export-Konfigurationen, d. h. jede umgewandelte MATLAB®-Funktion oder jedes umgewandelte Simulink®-Modell, kann individuell konfiguriert werden.

`TcCom.Generate`

Wenn TRUE, wird ein TcCOM-Objekt erstellt, welches in der TwinCAT XAE verwendet werden kann.

`TcCom.FpExceptionsForInit`

Optionen: CallerExceptions, ThrowExceptions, SuppressExceptions, LogExceptions, LogAndHold, LogAndCatch, LogAndDump, LogHoldAndDump, LogCatchAndDump

Weiteres im Bereich Exception Handling [▶ 55].

`TcCom.FpExceptionsForUpdate`

Optionen: CallerExceptions, ThrowExceptions, SuppressExceptions, LogExceptions, LogAndHold, LogAndCatch, LogAndDump, LogHoldAndDump, LogCatchAndDump

Weiteres im Bereich Exception Handling [▶ 55].

CallerExceptions: Es werden die Einstellungen des Aufrufenden übernommen, bsw. der Task, eines anderen TcCOM oder der SPS.

`TcCom.ExecutionInfoOutput`

Wenn TRUE, wird ein weiterer Output am TcCOM mit Informationen bei auftretenden Exceptions erstellt. Weiteres im Bereich Exception Handling [▶ 55].

`TcCom.OnlineChange`

Wenn TRUE, dann kann das TcCOM durch Online Change während TwinCAT XAR im Run Modus ist, ausgetauscht werden. Siehe auch Online Change für Target for Simulink®.

`TcCom.TcComWrapperFb`

Wenn TRUE, wird ein TcCOM-Wrapper-FB in der erzeugten SPS-Bibliothek angelegt.

`TcCom.TcComWrapperFbProperties`

Wenn TRUE, werden am TcCOM-Wrapper-FB Properties für Modulparameter angelegt.

`TcCom.TcComWrapperFbPropertyMonitoring`

Optionen: NoMonitoring, CyclicUpdate, ExecutionUpdate

Spezifiziert das Monitoring-Attribut der Properties. Im Standardfall ist „No Monitoring“ eingestellt, d. h. es wird kein Attribut gesetzt.

Einstellung in MATLAB	Attribut am Property
ExecutionUpdate	{attribute 'monitoring' := 'variable'}
CyclicUpdate	{attribute 'monitoring' := 'call'}

`PlcFb.Generate`

Wenn TRUE, wird ein Funktionsbaustein in der SPS-Bibliothek erstellt, welcher in der TwinCAT XAE verwendet werden kann.

`PlcFb. FpExceptionsForInit`

Floating Point Exceptions innerhalb des Funktionsbausteins bei der Init-Phase können eingestellt werden.

Optionen: CallerExceptions, ThrowExceptions, SuppressExceptions, LogExceptions, LogAndHold, LogAndCatch, LogAndDump, LogHoldAndDump, LogCatchAndDump

Weiteres im Bereich Exception Handling [▶ 55].

`PlcFb. FpExceptionsForUpdate`

Floating Point Exceptions innerhalb des Funktionsbausteins bei der Update-Phase können eingestellt werden.

Optionen: CallerExceptions, ThrowExceptions, SuppressExceptions, LogExceptions, LogAndHold, LogAndCatch, LogAndDump, LogHoldAndDump, LogCatchAndDump

Weiteres im Bereich Exception Handling [▶ 55].

Erstellen und Laden einer Export-Konfiguration

Bei Nutzung des TwinCAT Target for MATLAB® wird zunächst der MATLAB® Coder™ benutzt, um C++-Sourcen zu erzeugen. Diese C++-Sourcen können dann zu einer Export-Konfiguration im TwinCAT-Modulgenerator zusammengefasst werden durch:

```
TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig('MFile',Name,'BuildDir',cppDir)
```

Als Properties wird mit `'BuildDir'` der Pfad zu den, vom MATLAB® Coder™ erstellen, C++-Sourcen und mit `'MFile'` der Name der MATLAB®-Funktion übergeben. Wird als Name bspw. BaseStatistics gewählt, so wird das TcCOM-Objekt diesen Namen tragen und der Funktionsbaustein in der SPS den Namen FB_BaseStatistics bekommen.

Wird das TwinCAT Target for Simulink® genutzt, ist die Herangehensweise etwas verändert. Starten Sie den Build-Vorgang aus Simulink® heraus mit deaktivierter Option „Run the publish step after project generation“. Laden Sie dann den erstellen <modelname>_tcgrt Ordner wie folgt, um die C++-Sourcen des Simulink®-Modells zur Projekt-Export-Konfiguration hinzuzufügen.

```
TwinCAT.ModuleGenerator.ProjectExportConfig.Load(<modelname>_tcgrt);
```

Erstellen von TwinCAT Objekten mit dem Modul-Generator

```
TwinCAT.ModuleGenerator.ProjectExporter()
```

Mit `TwinCAT.ModuleGenerator.ProjectExporter()` wird der Build-Vorgang für die im Property `Project` eingestellten Plattformen angestoßen. Als Argument wird das Objekt der Klasse TwinCAT.ModuleGenerator.ProjectExportConfig übergeben. Dadurch wird auf dem lokalen Dateisystem im Repository ein Ordner angelegt und die erstellten Treiber und Beschreibungsdateien abgelegt.

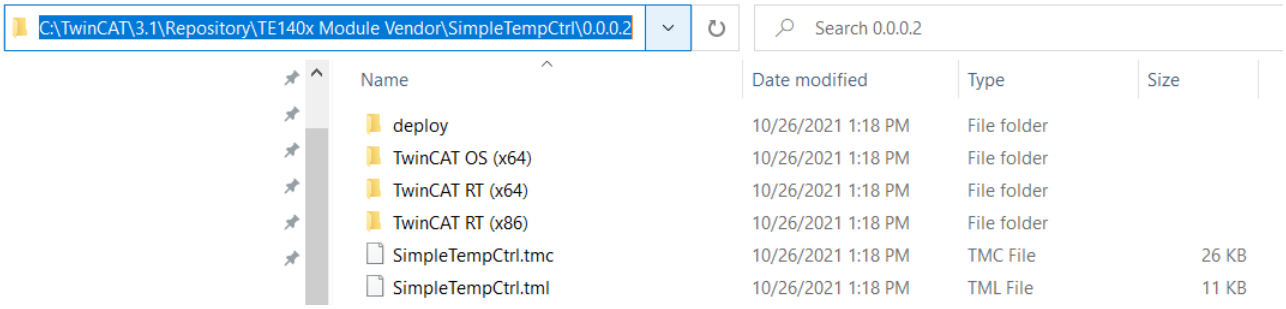
Beispielaufruf:

```
projExporter = TwinCAT.ModuleGenerator.ProjectExporter(exportConfig);
```

7.1 Erstellen von TMX-Archiven

Um mit den erstellen TwinCAT-Objekten (TcCOM und SPS Bibliothek) 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

Beispielsweise der SimpleTempCtrl in Version 0.0.0.2 liegt hier:



Händisches Kopieren auf Engineering PCs ist fehleranfällig. Einfacher ist es daher, ein sogenanntes TMX-Archiv zu erstellen. Das TMX-Archiv ist ein Archiv eines neu erstellten Projekts, beispielsweise dem SimpleTempCtrl in Version 0.0.0.2. Es muss nur das Archiv auf einen Engineering PC kopiert und ausgeführt werden. Es handelt sich um ein selbstextrahierendes Archiv, welches dann alle Dateien automatisch an die korrekte Stelle kopiert.

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

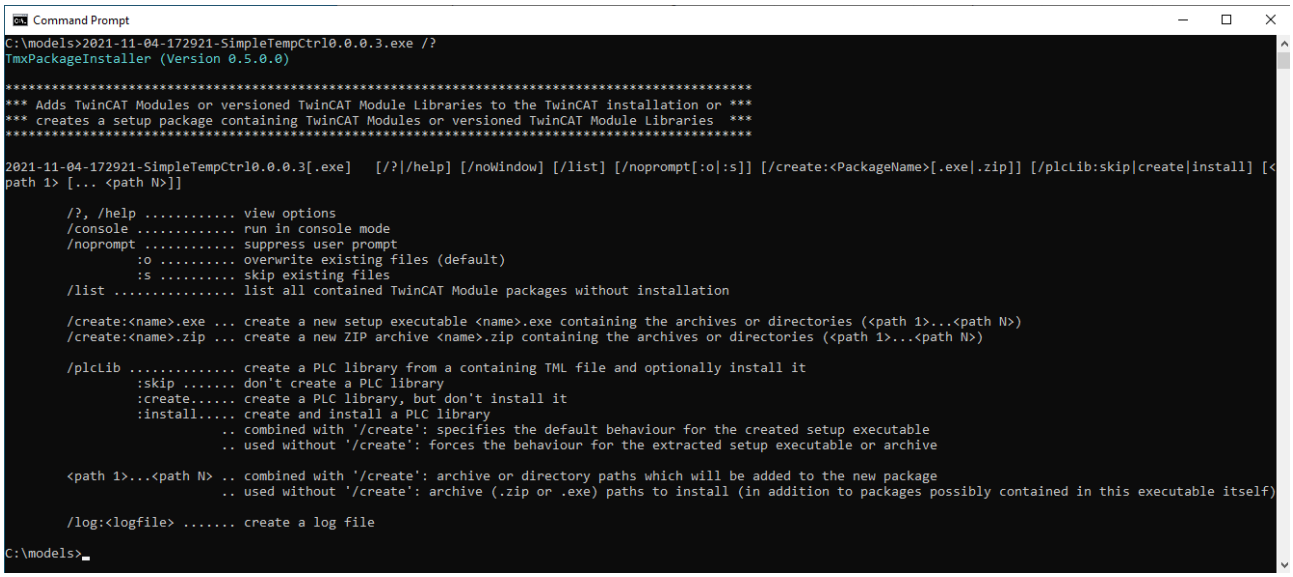
Nutzen Sie dazu das Project-Property des Modulgenerators:

```
Project.TmxArchive = "c:\archives\[Date]-[Time]-[LibName][LibVersion].exe"
```

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).

Das TMX-Archiv können Sie dann 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.

Sie können z. B. auch das Command direkt- sowie weitere Optionen nutzen:



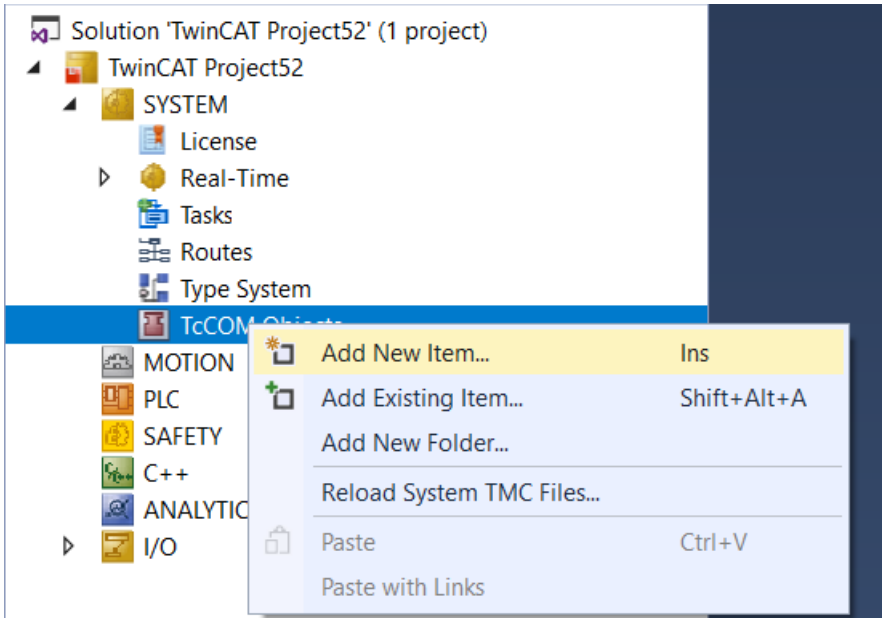
Beispielsweise erstellt (aus der *.tml-Datei) und installiert der Befehl <tmxarchive>.exe /plclib:install die PLC Bibliothek auf Ihrem lokalen Engineering PC.

8 Anwendung von Modulen in TwinCAT

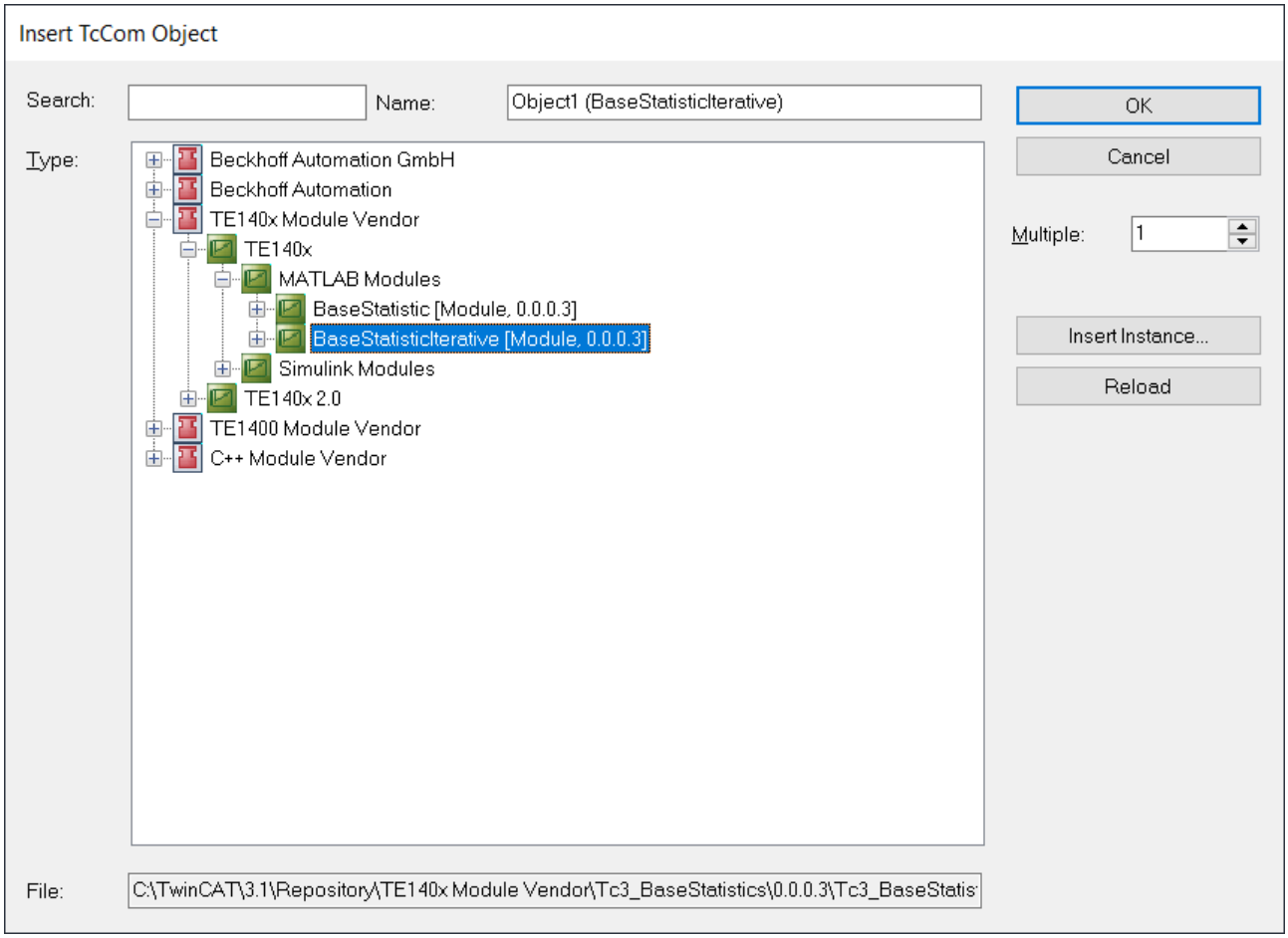
8.1 Arbeiten mit dem TcCOM-Modul

TcCOM-Objekte in TwinCAT 3 verwenden

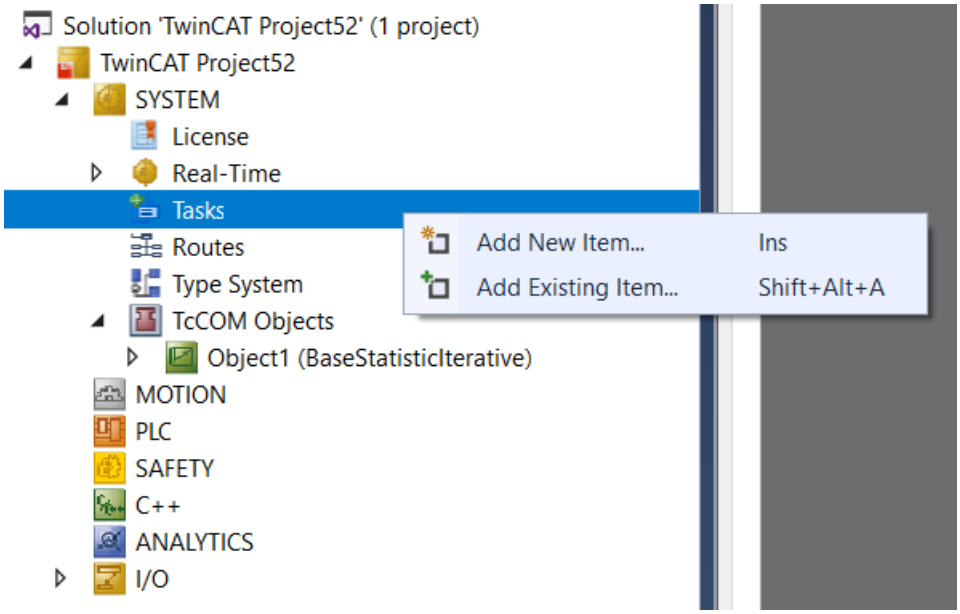
1. Fügen Sie ein neues TcCOM-Objekt ein.



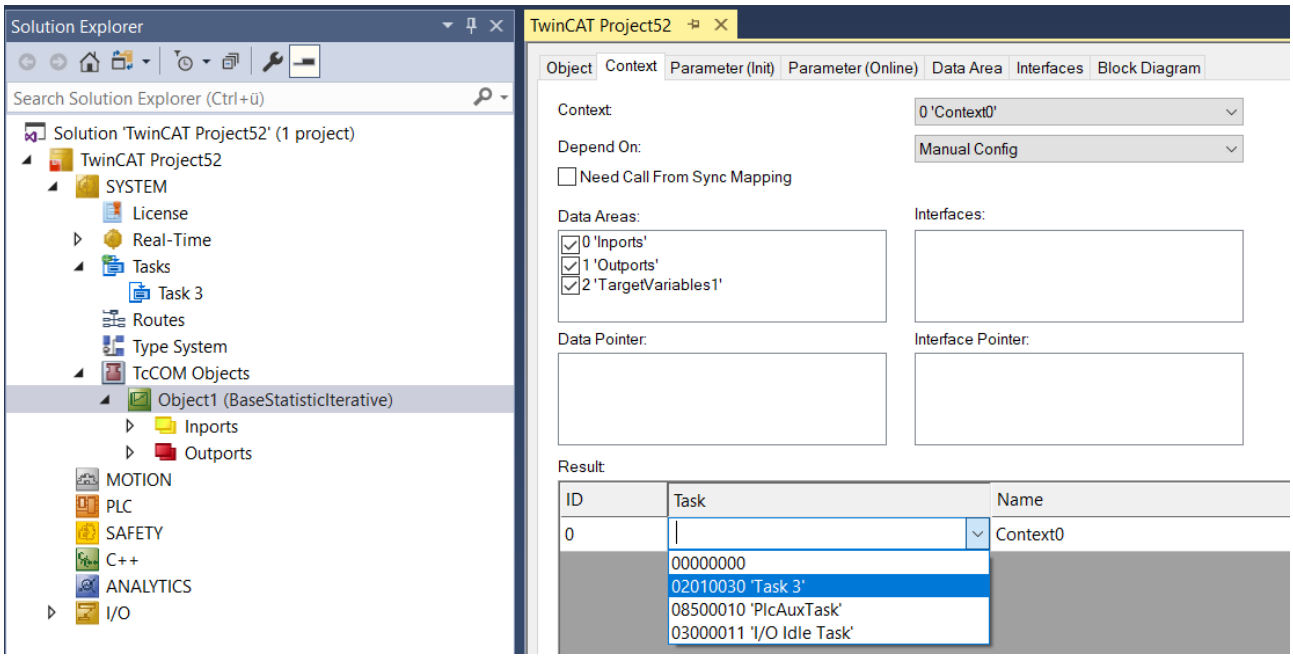
2. Wählen Sie das entsprechende TcCOM-Objekt aus:



3. Erstellen Sie eine neue zyklische Task vom Typ *TwinCAT Task.*



4. Weisen Sie dem neu erstellen TcCOM-Objekt die neu erstelle Task zu. Gehen Sie dazu auf die Instanz des TcCOM-Objekts und wählen Sie den Reiter *Context*.



⇨ Sie können nun die Konfiguration aktivieren. Um das TcCOM-Objekt vorher mit anderen Modulen in ihrer TwinCAT Solution zu verbinden, können Sie das Prozessabbild verwenden, um Mappings zu erstellen.

Sie können den MATLAB® Code unter dem Reiter Block Diagram einsehen und im laufenden Betrieb Werte beobachten und scopen. Siehe dazu MATLAB Code Darstellung [▶ 43].

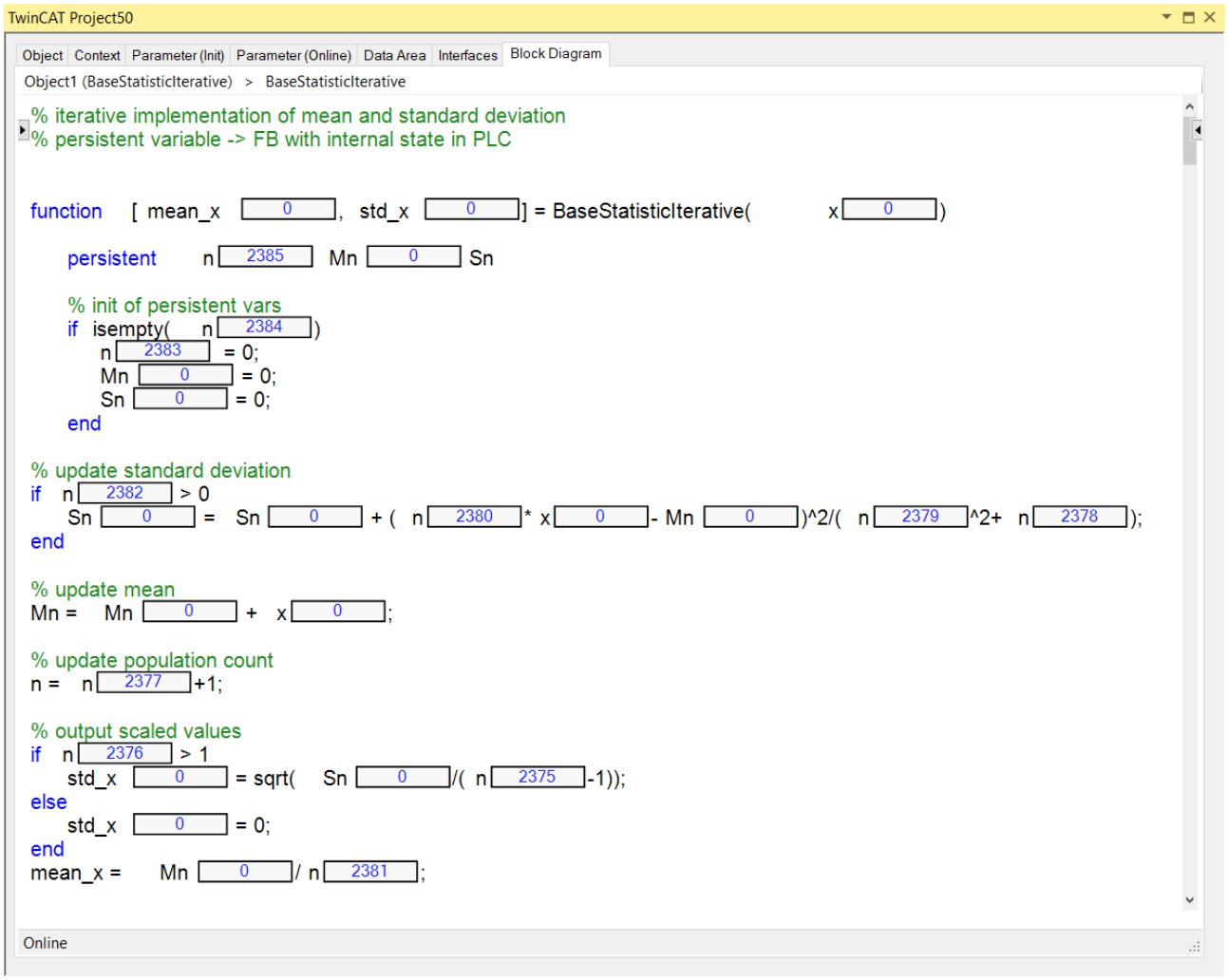
Sie können das TcCOM-Objekt auch aus der SPS heraus über den TcCOM-Wrapper-FB aufrufen oder sogar dynamisch instanziieren. Weiteres dazu siehe: Aufruf eines TcCOM-Objekts aus der SPS [▶ 51].

Neben der Arbeit mit einem TcCOM-Objekt, können Sie auch die in MATLAB® erstellten Funktionen direkt als SPS-Baustein (SPS-FB) nutzen, ohne sich Gedanken über ein TcCOM-Objekt machen zu müssen. Siehe dazu: Arbeiten mit der SPS-Bibliothek [▶ 46].

8.1.1 MATLAB Code Darstellung

8.1.1.1 MATLAB®-TcCOM

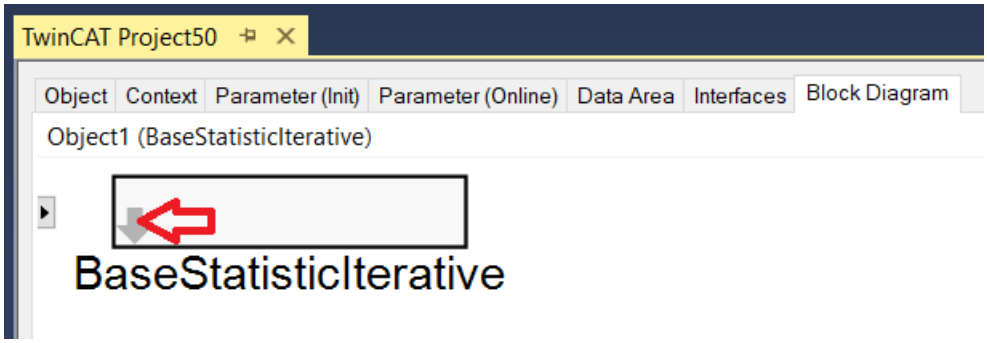
Wurde mit dem TwinCAT Target for MATLAB® ein TwinCAT-Objekt erzeugt und der MATLAB® Code Export dabei ausgeführt, kann der MATLAB® Code der MATLAB®-Function als Control in der TwinCAT XAE dargestellt werden.



8.1.1.1.1 Bedienung des Blockdiagramm-Fensters

Bei der Generierung eines TcCOM-Moduls aus MATLAB® kann der Export des MATLAB®-Codes konfiguriert werden. Wurde der Export aktiviert, findet man den Code in der TwinCAT-Entwicklungsumgebung unter dem Karteireiter **Block Diagram** der Modul-Instanz.

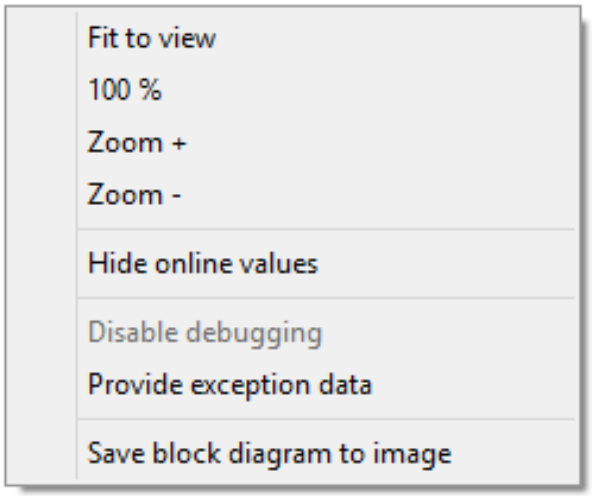
Auf oberster Ebene finden Sie das erstellte Modul in Block-Darstellung. Wählen Sie den grauen Pfeil im Block an, um den Inhalt dazustellen.



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:

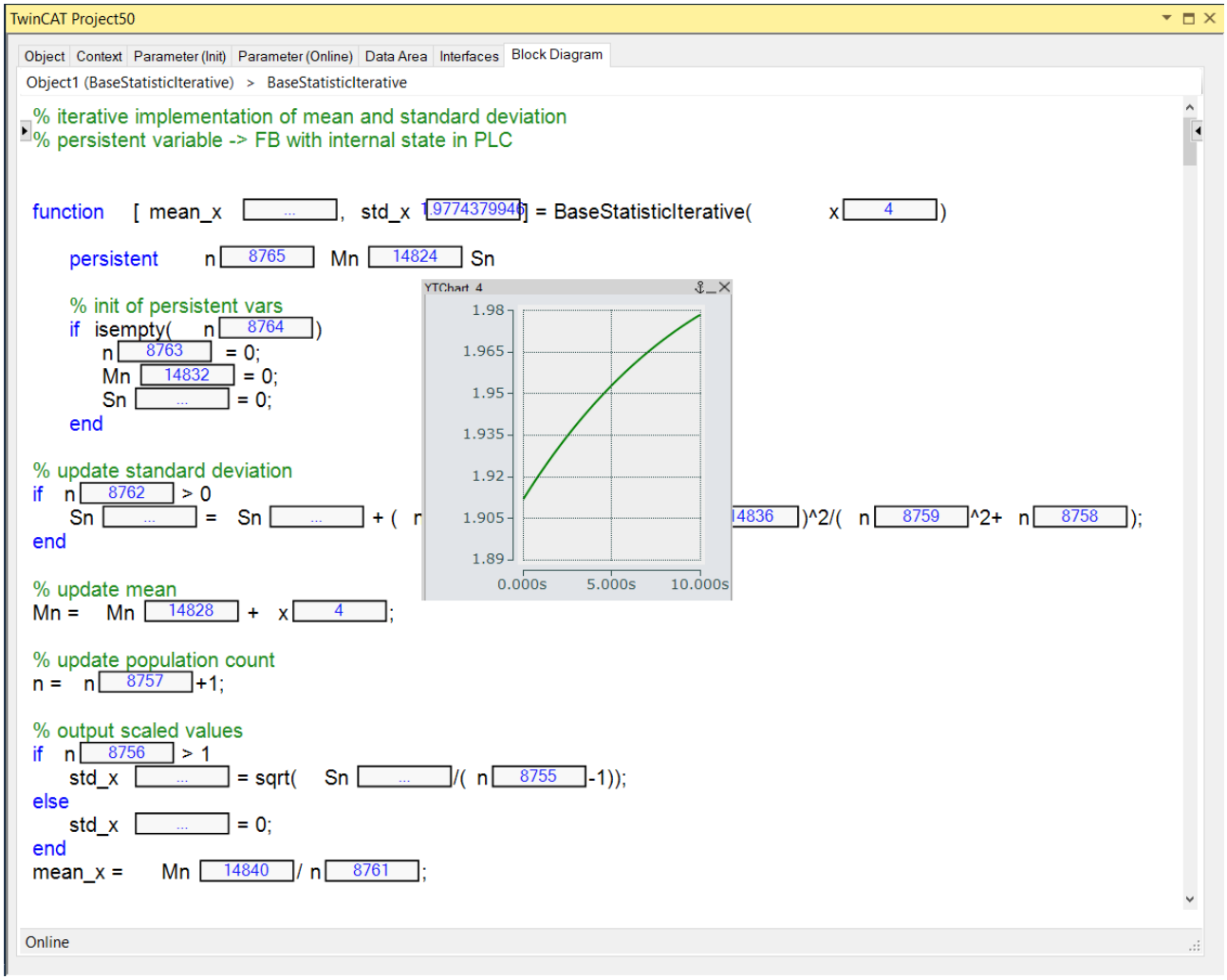


8.1.1.1.2 Anzeige von Signalverläufen

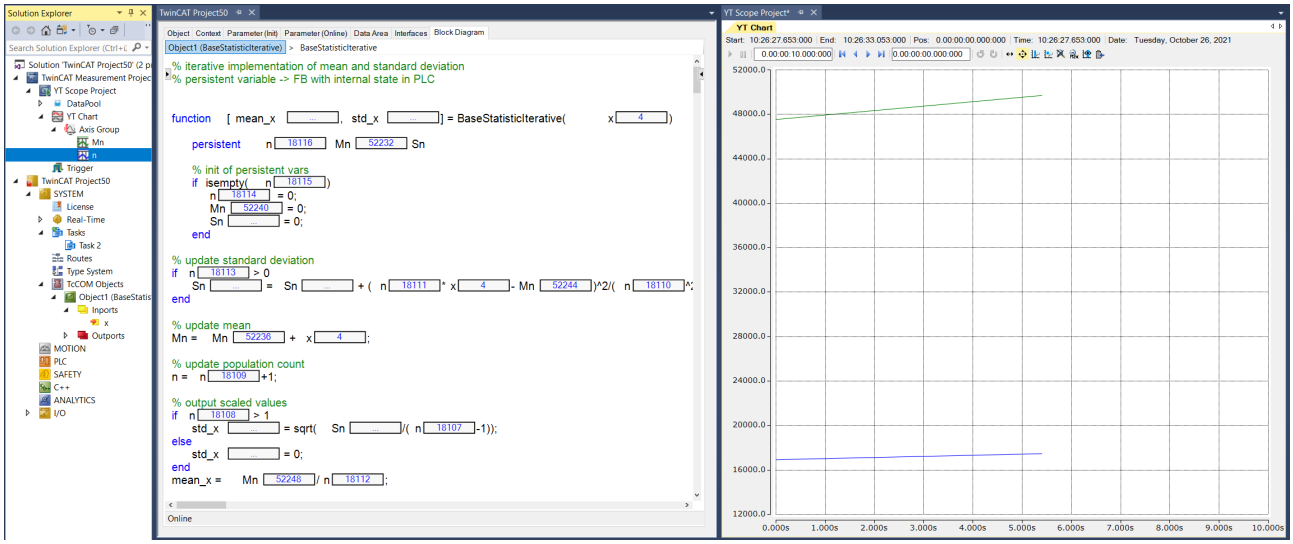
Ausgewählte Variablen können im TwinCAT XAE über ADS abgerufen werden. Somit besteht die Möglichkeit, diese in einem Mini-Scope innerhalb des Block-Diagramms, oder mit dem TwinCAT Scope innerhalb eines Measurement-Projektes, darzustellen.

Variablen, welche im Scope dargestellt werden können, haben einen nachgestellten schwarzen Rahmen in der Code-Darstellung. In diesem Rahmen werden im laufenden Betrieb die Werte in Blau dargestellt.

Durch Drag&Drop einer „blauen Variablen“ auf das Block-Diagramm-Fenster öffnet sich ein Mini.Scope.



Durch Drag&Drop einer „blauen Variablen“ auf die Axis Group eines Charts im TwinCAT Measurement-Projekt, werden die Variablen zum TwinCAT Scope hinzugefügt.



Welche Variablen sind als „blaue Variablen“ im TwinCAT XAE sichtbar?

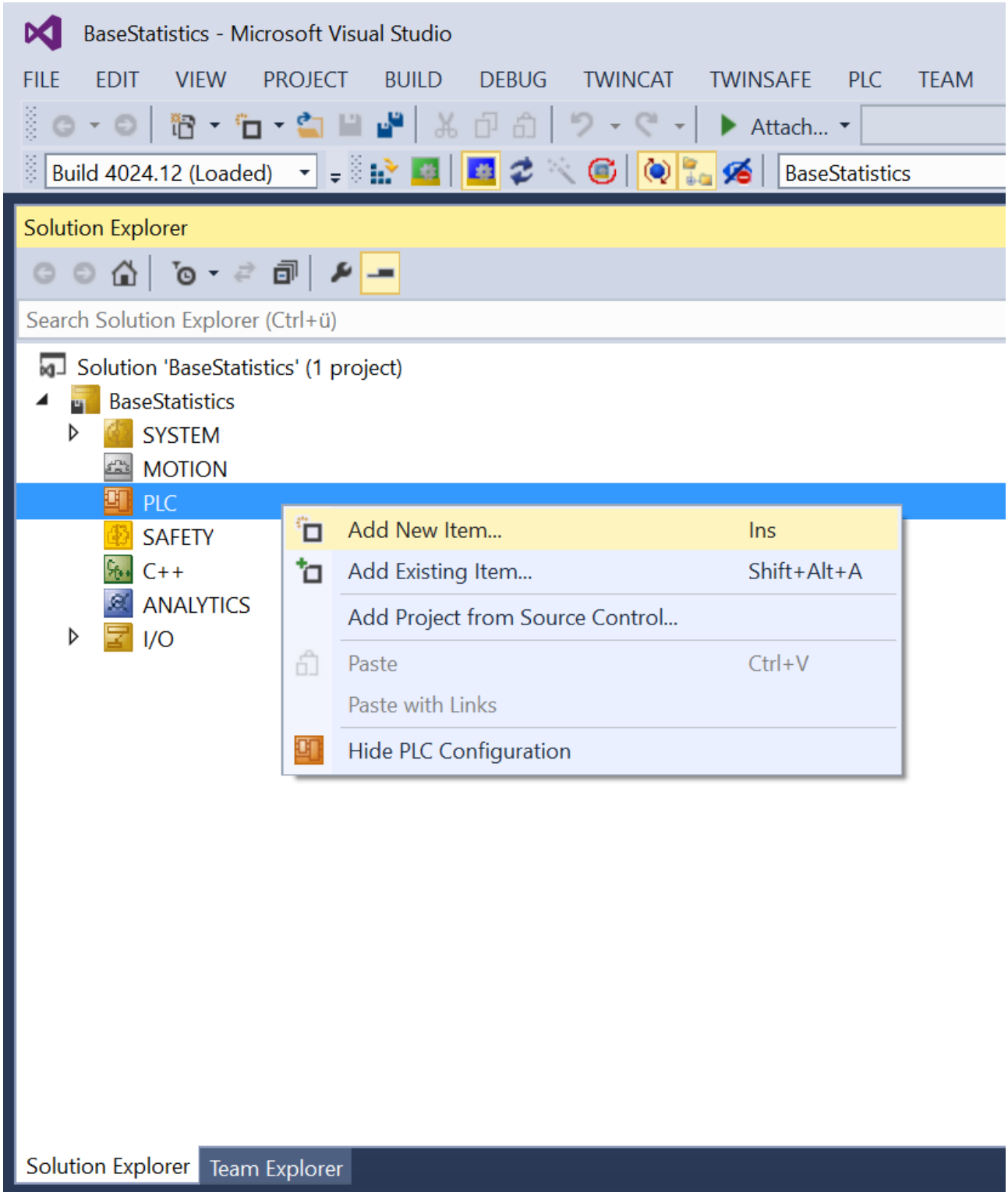
- Die Eingangsgrößen
- Die Ausgangsgrößen
- Persistente Variablen (MATLAB-Definition `persistent var1 … varN`)

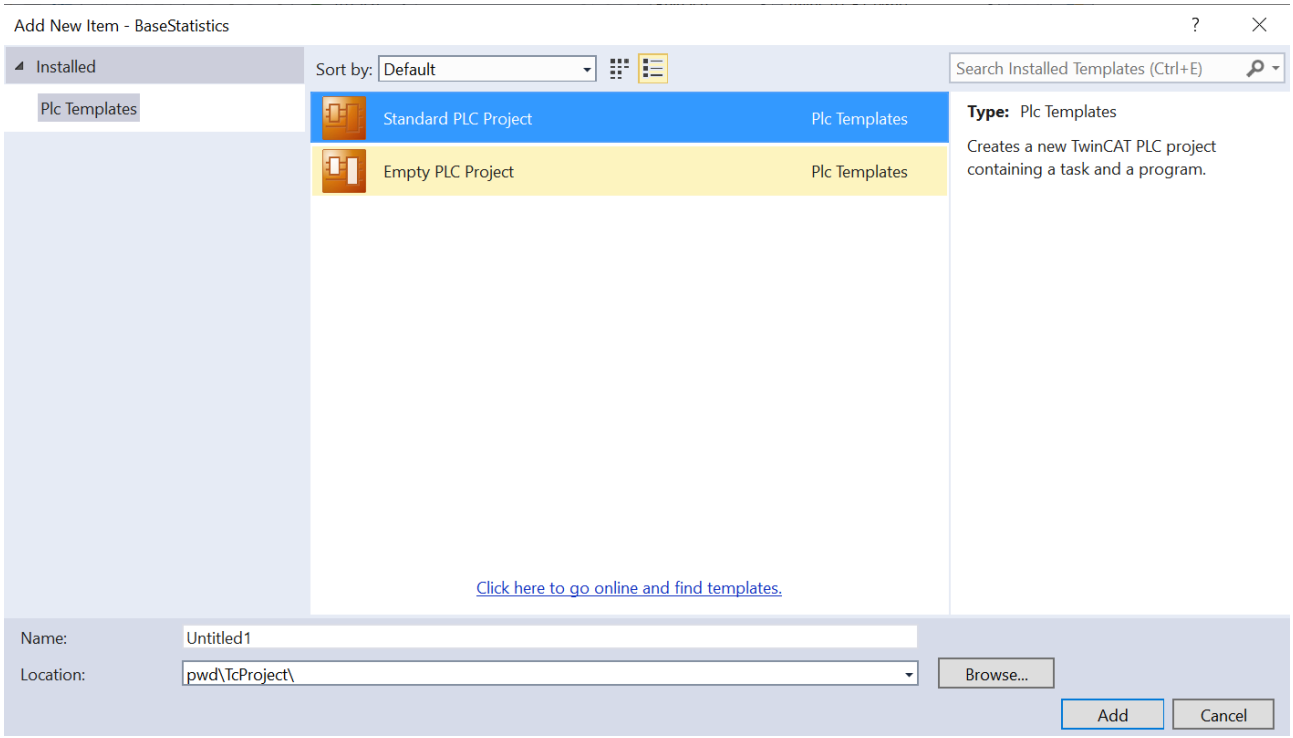
8.2 Arbeiten mit der SPS-Bibliothek

SPS-Bibliothek in TwinCAT 3 verwenden

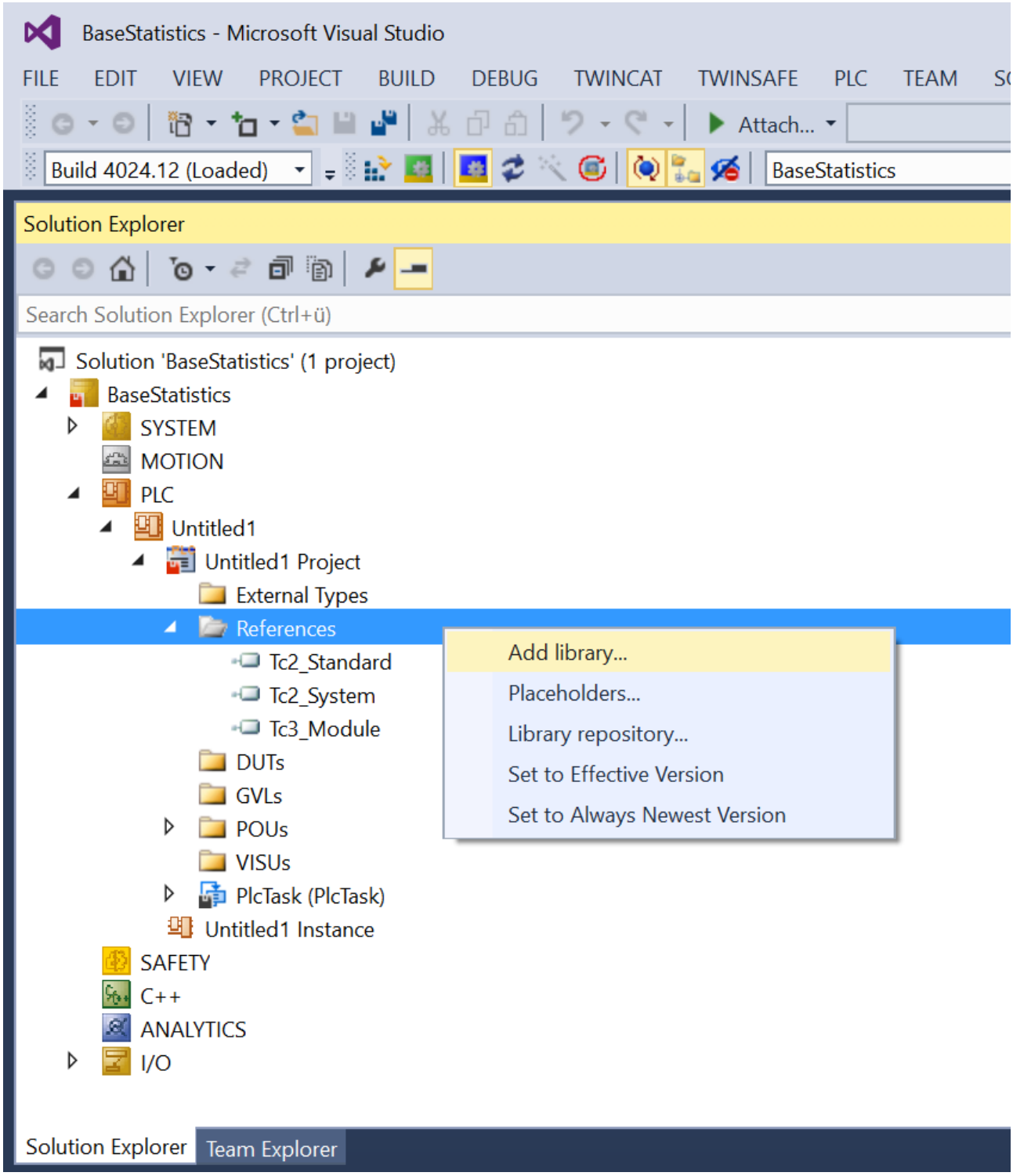
✓ Ausgehend von einer neuen TwinCAT Solution, erstellen Sie ein SPS Projekt:

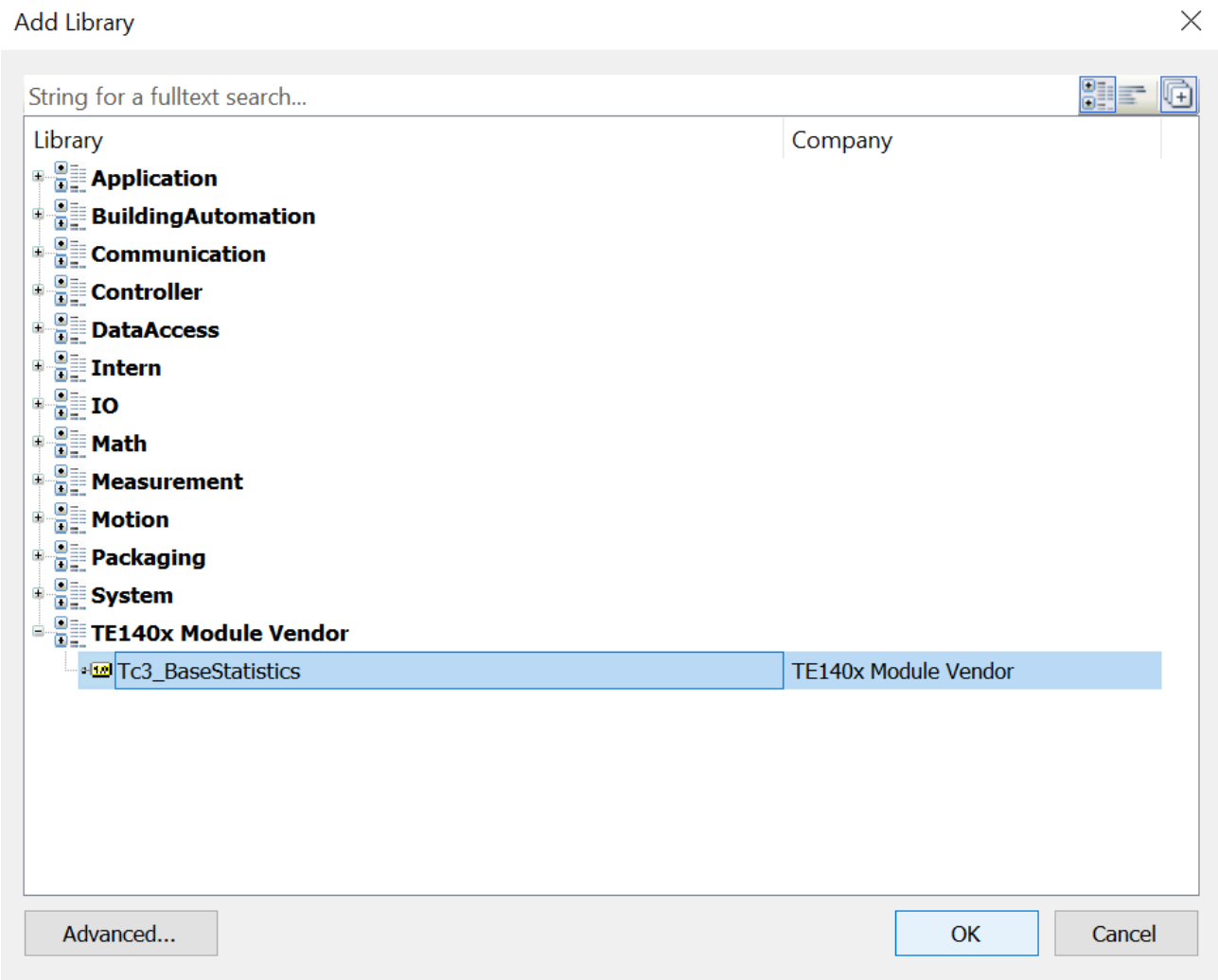
1. Führen Sie die folgenden Menüschritte aus.



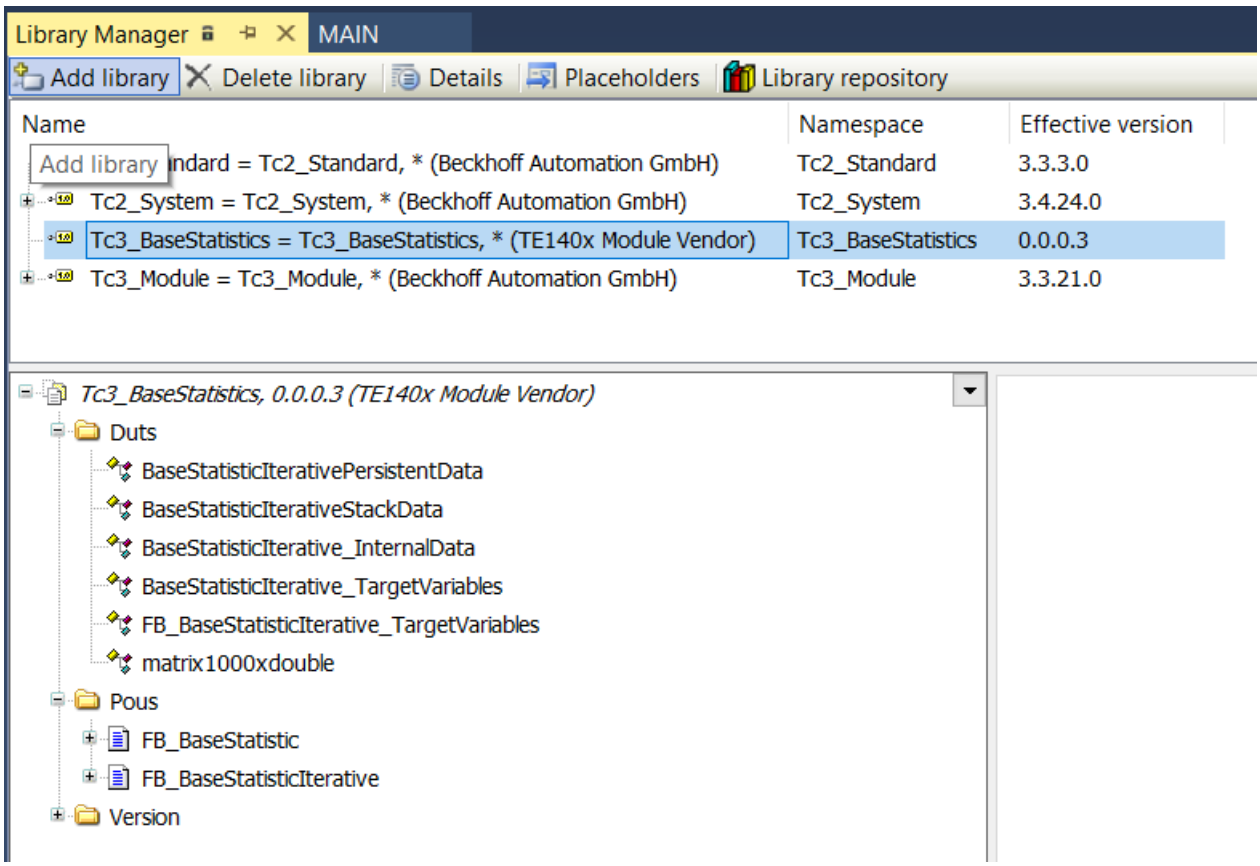


2. Fügen Sie danach die neu erstellte (und schon installierte) SPS Bibliothek hinzu:

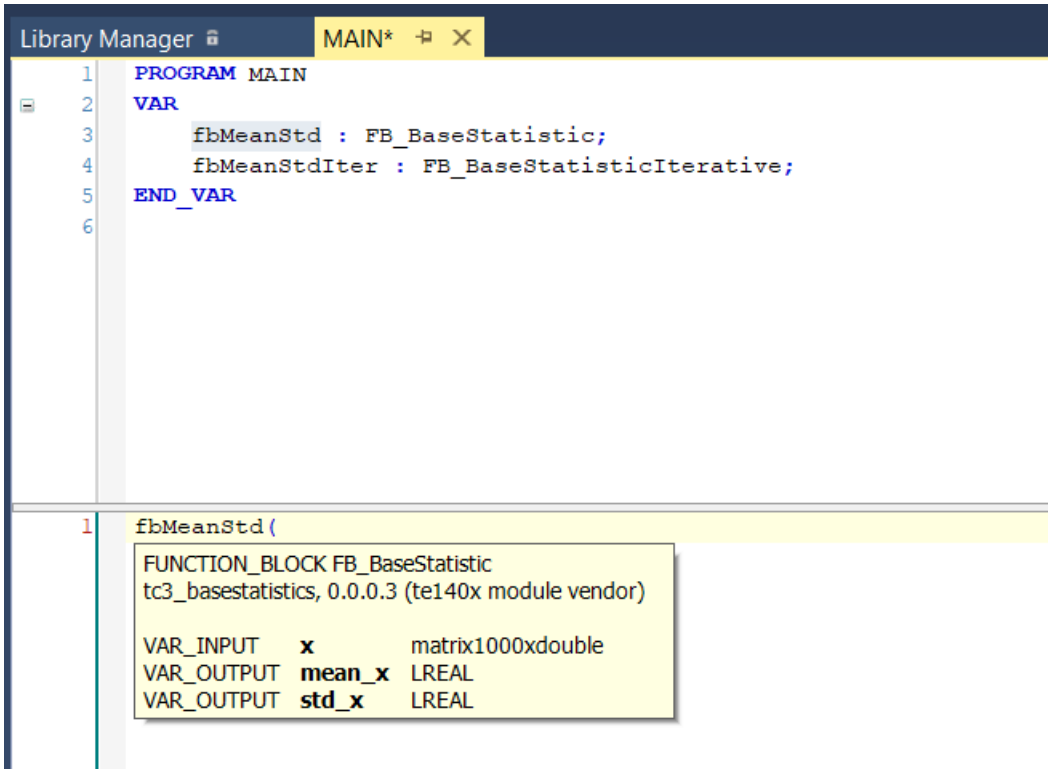

BaseStatistics - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TWINCAT TWINSAFE PLC TEAM
Attach...
Build 4024.12 (Loaded)
BaseStatistics
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'BaseStatistics' (1 project)
BaseStatistics
SYSTEM
MOTION
PLC
Untitled1
Untitled1 Project
External Types
References
Tc2_Standard
Tc2_System
Tc3_Module
DUTs
GVLs
POUs
VISUs
PlcTask (PlcTask)
Untitled1 Instance
SAFETY
C++
ANALYTICS
I/O
Add library...
Placeholders...
Library repository...
Set to Effective Version
Set to Always Newest Version
Solution Explorer
Team Explorer



3. Machen Sie sich einen Überblick über die Datentypen und Funktionsbausteine:



4. Fügen Sie Instanzen der Funktionsbausteine in Ihre SPS ein und nutzen Sie diese in Ihrer Applikation:



8.2.1 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 MATLAB®.
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 ihre MATLAB-Funktion(en) und erstellen Sie daraus eine SPS-Bibliotheksversion (0.0.0.2).
5. Wählen Sie in TwinCAT - SPS - **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).

8.2.2 Aufruf eines TcCOM-Objekts aus der SPS

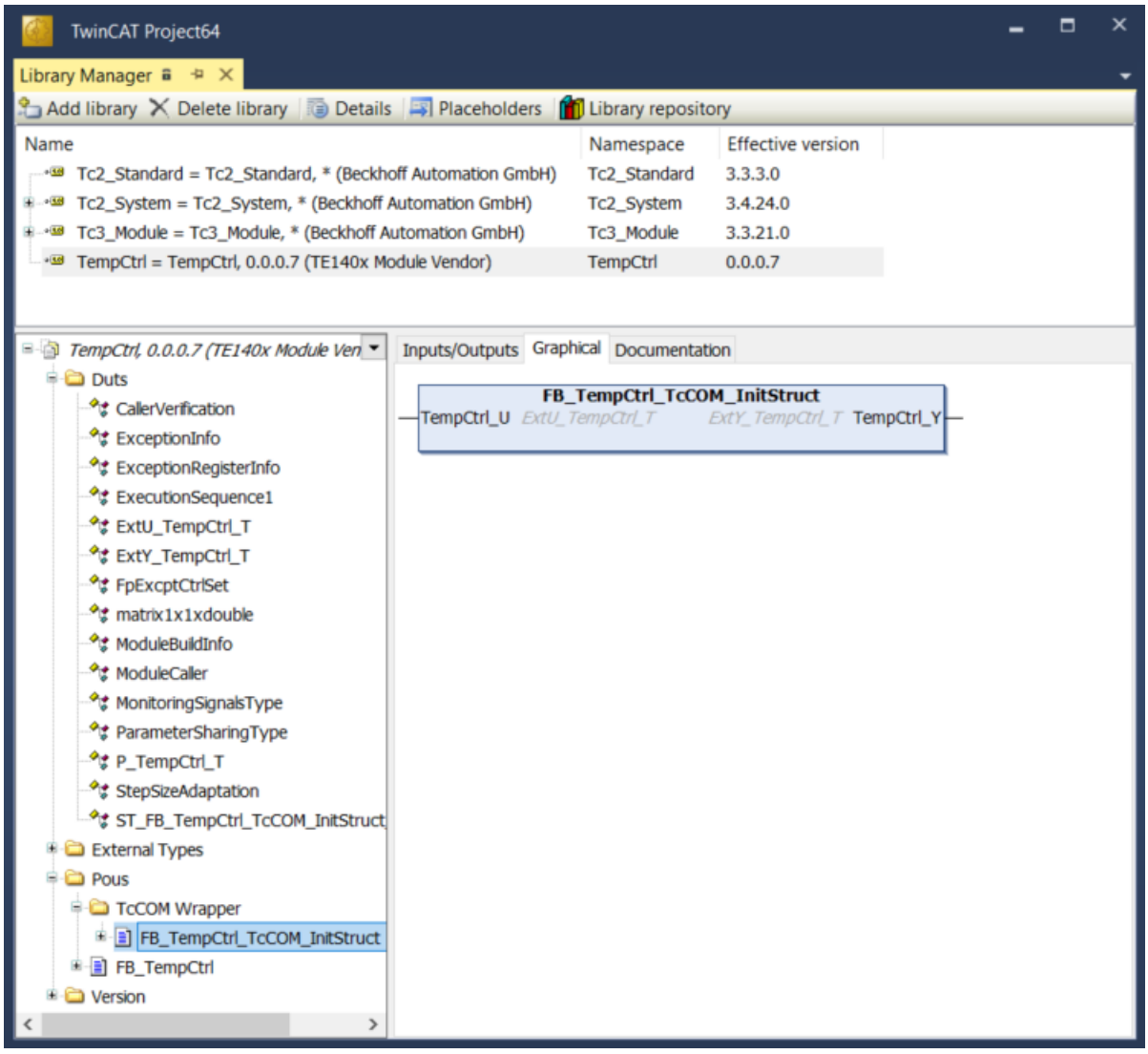
Erstellen eines TcCOM-Wrapper-FB

In der Export-Konfiguration ist zu setzen:

```
TcCom.TcComWrapperFb = 'true';
TcCom.TcComWrapperFbProperties = 'true'; % optional
```

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.

- 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.

Deklaration

```
// link wrapper with a static instance
InitStrStatic      : ST_FB_TempCtrl_TcCOM_InitStruct := (noid := 16#01010010);   // OID from object1
 in System > TcCOM Objects
fbTempCtrStatic    : FB_TempCtrl_TcCOM_InitStruct(InitStrStatic);
Inputs             : ST_TempCtrl_U_T;      // data type defined in TempCtrl library
Outputs            : ST_TempCtrl_Y_T;
```

Ausführungs-Code

```
fbTempCtrStatic(stTempCtrl_U := Inputs, stTempCtrl_Y => Outputs);
```

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.

- 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.

Deklaration

```
// dynamic instance: create TcCOM from PLC
InitStrDyn         : ST_FB_TempCtrl_TcCOM_InitStruct := (
                       nTaskOid      := 16#02010030,             // take TaskOID of PlcTask
                       eModuleCaller   := E_ModuleCaller.Module );      // set module caller to "ca
ll by module"
fbTempCtrDyn       : FB_TempCtrl_TcCOM_InitStruct(InitStrDyn);
OutputsDyn         : ST_TempCtrl_Y_T;
```

Ausführungs-Code

```
fbTempCtrDyn(stTempCtrl_U := Inputs, stTempCtrl_Y => OutputsDyn);
```

Der Quellcode zur oben gezeigten Grafik ist verfügbar in MATLAB® über das Command Window

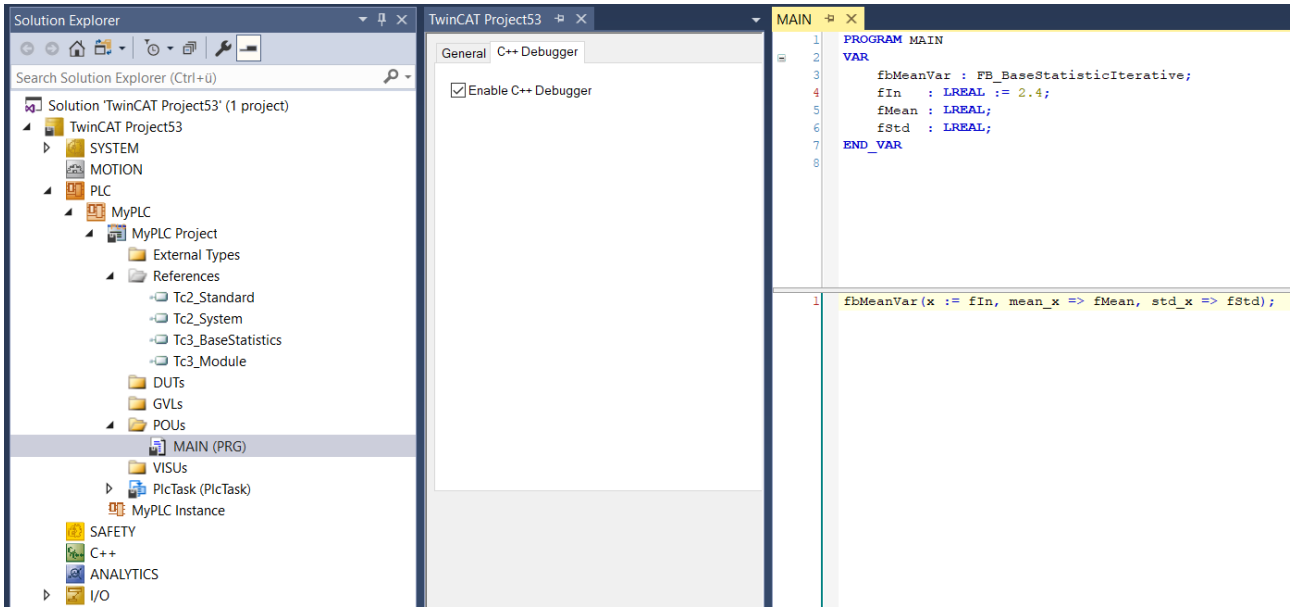
```
TwinCAT.ModuleGenerator.Samples.Start("TcCOM Wrapper Function Blocks")
```

8.3 Debugging

Folgende Schritt-für-Schritt Anleitung gilt für die Verwendung von TcCOM-Objekten und Funktionsbausteinen, die mit dem Target for MATLAB® erstellt worden sind, gleichermaßen. Gezeigt wird im Folgenden das Debugging für einen SPS-Funktionsbaustein.

✓ Schritt-für-Schritt Vorgehen:

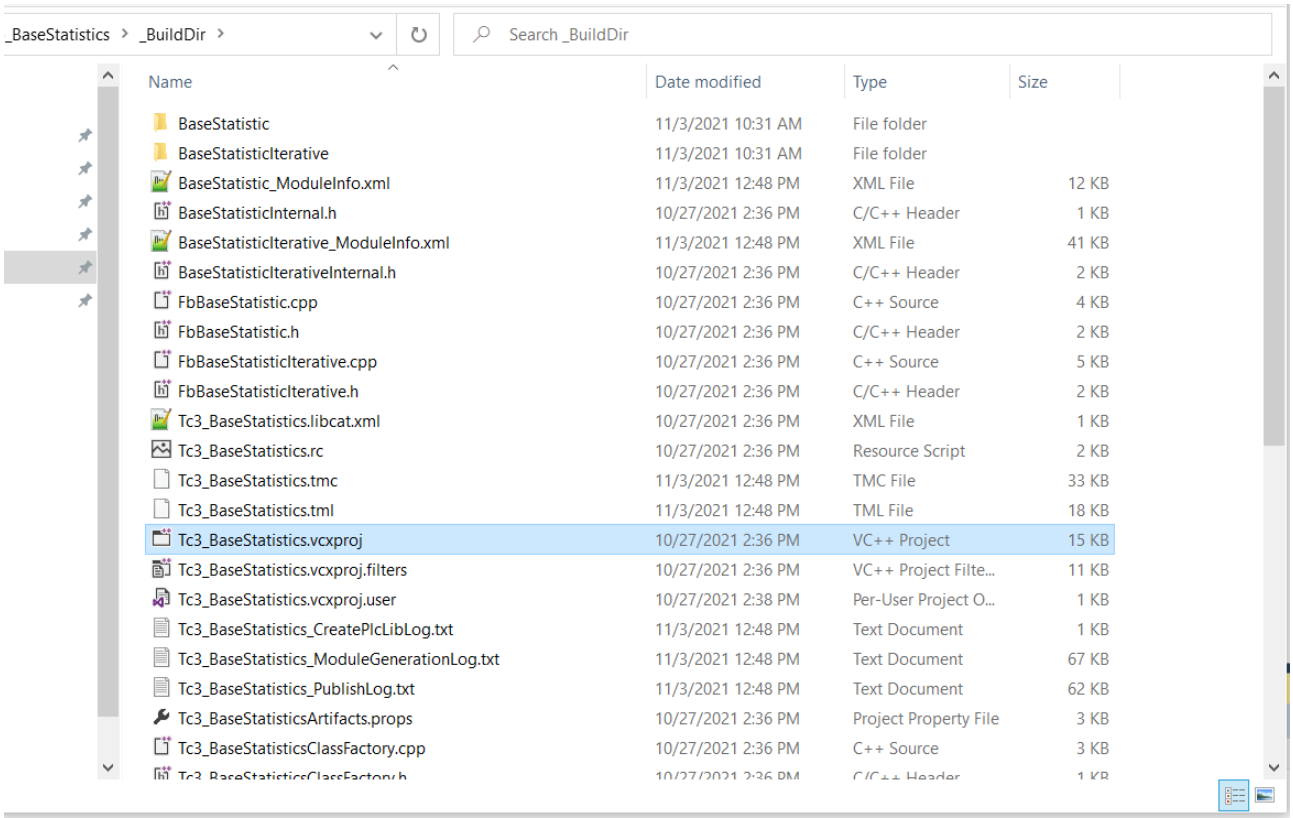
1. Stellen Sie sicher, dass Ihre TwinCAT-Applikation mit aktiviertem C++-Debugger aktiviert wurde.



2. Öffnen Sie das bei der Code-Generierung erstellte TwinCAT C++-Projekt, das zu dem Modul gehört, welches Sie debuggen möchten.

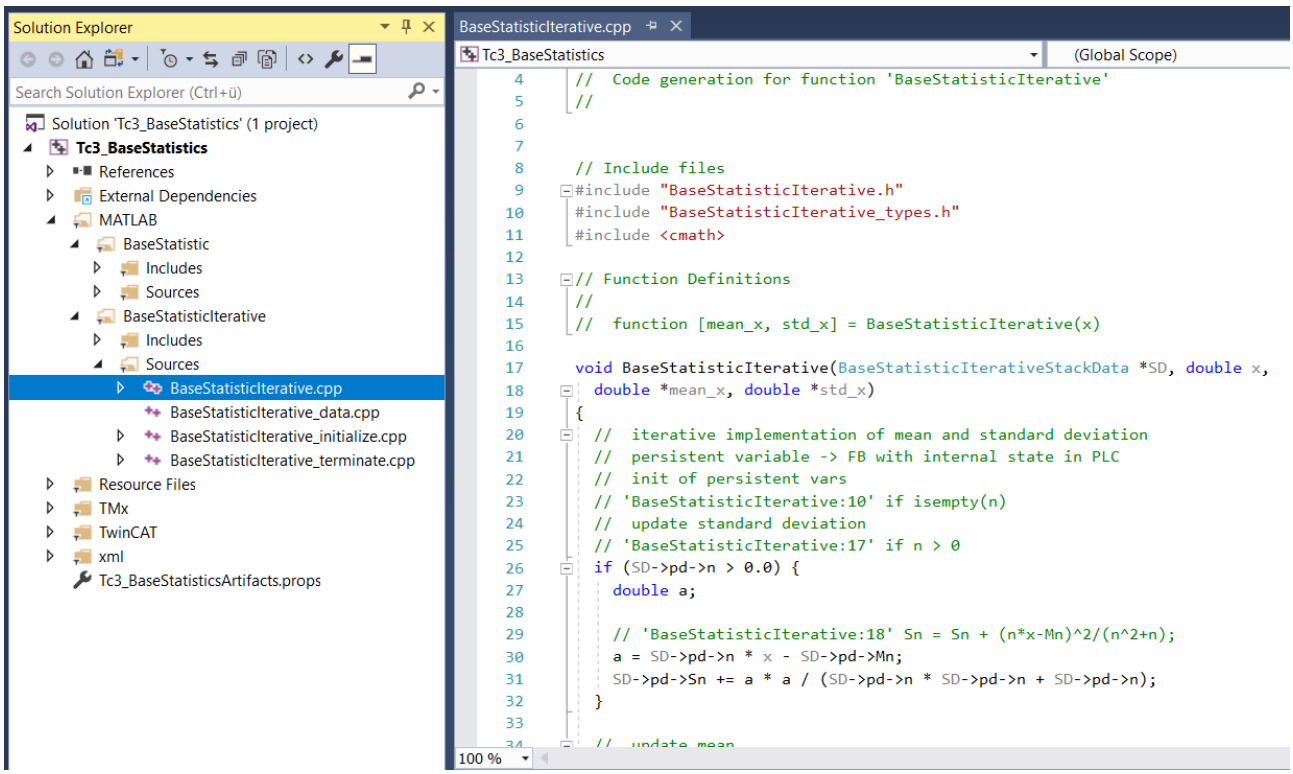
   ⇨ Den Projektort haben Sie bei der Initialisierung der Projekt-Export-Konfiguration angegeben, siehe Generierter Code TE1401 [▶ 33].

   ⇨ Sie können das Visual Studio Projekt direkt öffnen, oder es in Ihre TwinCAT Solution unter C++ mit „Add existing Item“ hinzufügen.

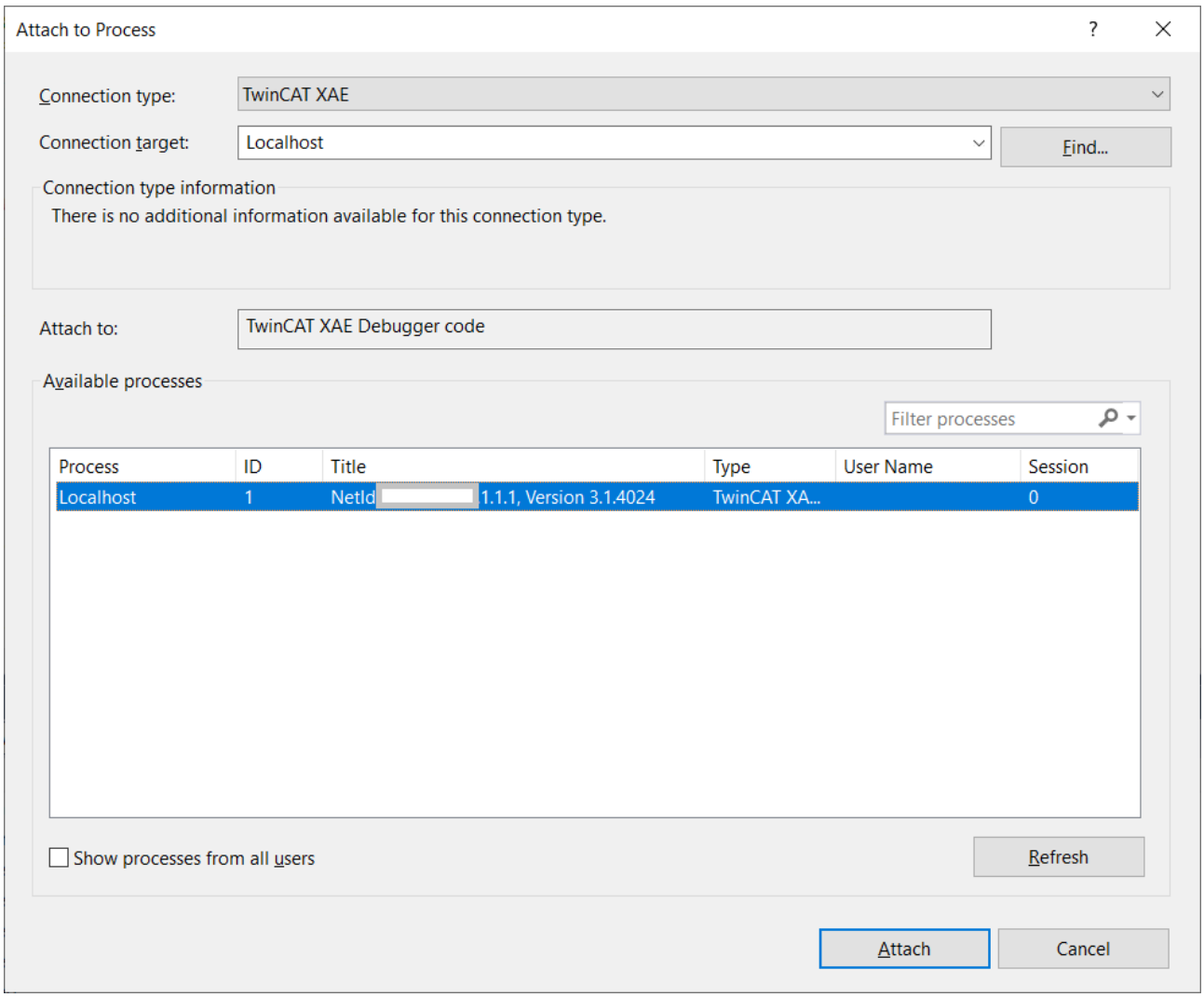


3. Sehen Sie sich in der Visual Studio Solution im Ordner MATLAB® die Sub-Ordner an, welche den Namen der erstellten MATLAB®-Funktion tragen.

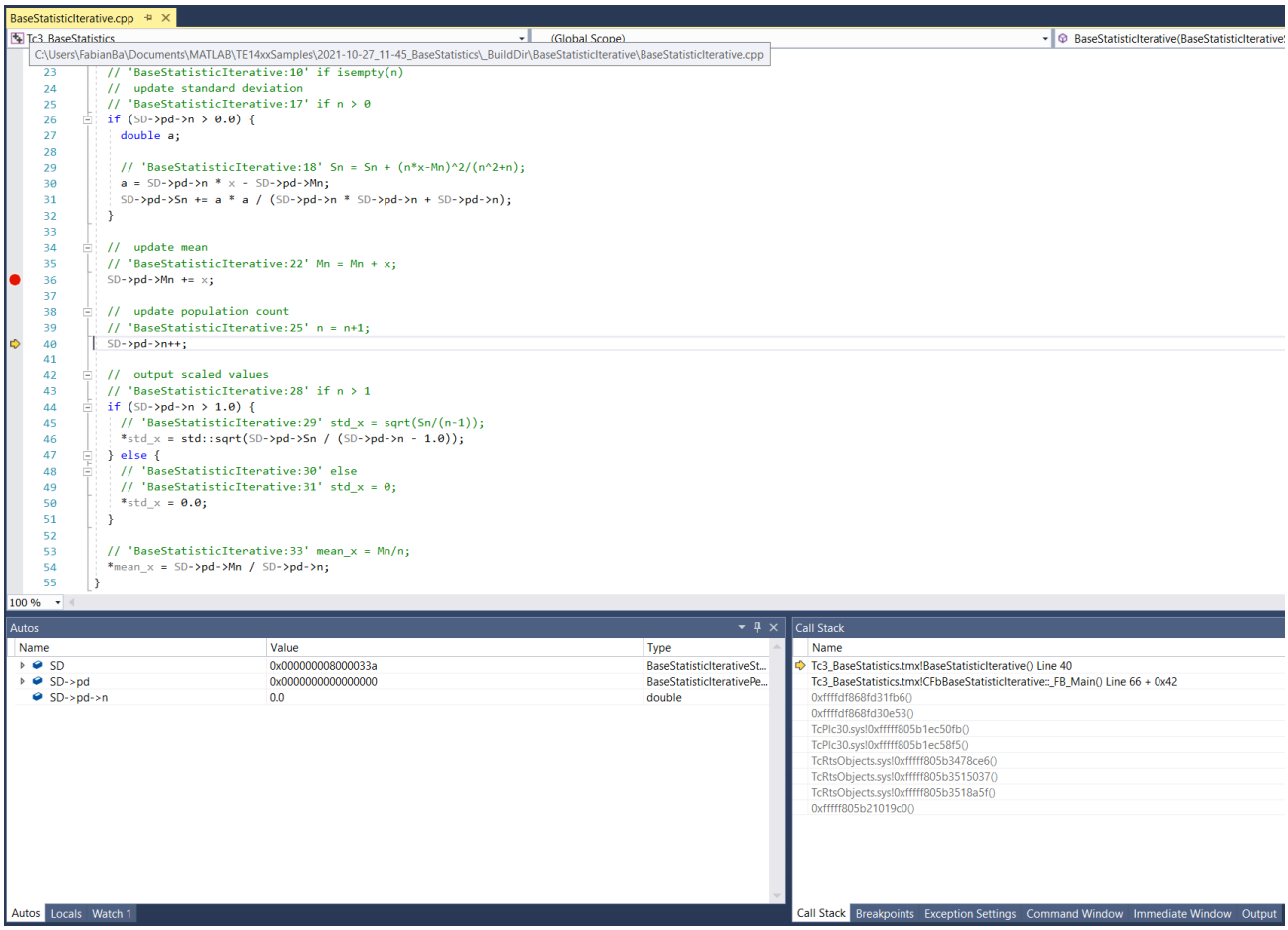
   ⇨ Im Sub-Order Sources finden Sie den ausgeführten Code, der vom MATLAB® Coder™ generiert wurde.



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.



8.4 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.

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, 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.

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.

Für das TcCOM-Objekt können Sie einen zusätzlichen Ausgang „ExecutionInfo [▶ 36]“ aktivieren:

```
exportConfig.ClassExportCfg{nModuleCount}.TcCom.ExecutionInfoOutput = true;
```

ExecutionInfo Output für PLC-FB

Aktuell ist die ExecutionInfo nur für das TcCOM-Objekt verfügbar. Wenn Sie den Code aus der PLC heraus aufrufen möchte, nutzen Sie dazu den TcCOM-Wrapper-FB.

Der Ausgang ExecutionInfo ist eine Struktur mit folgenden Einträgen:

ExcecutionInfo 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

ExceptionCode	DINT	Code der Exception, vgl. Microsoft Hilfe
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
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 den *ReturnAddr* eine 0.

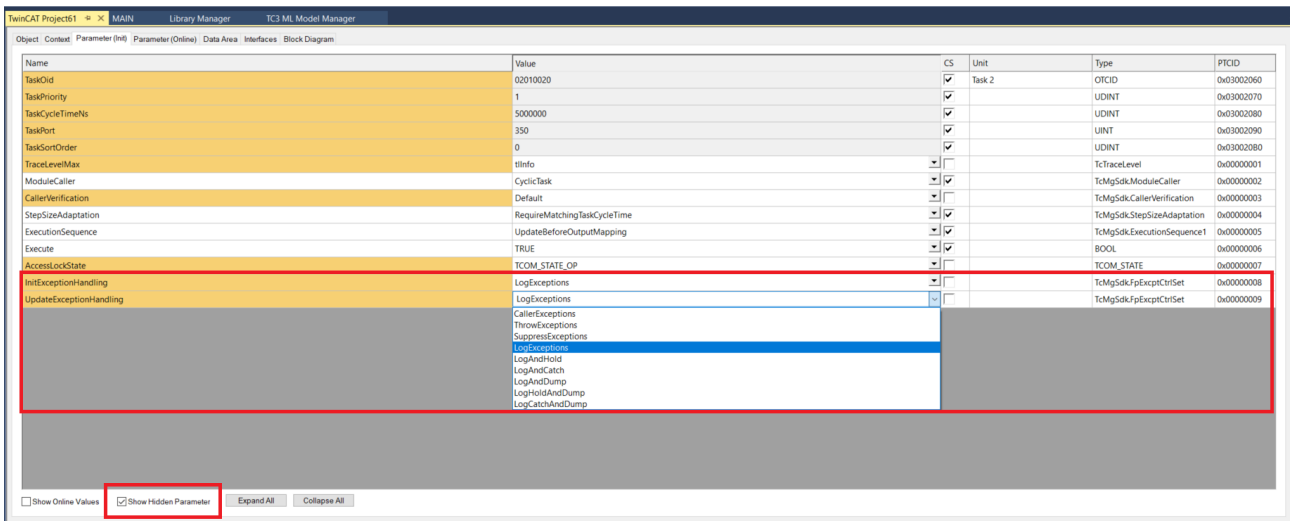
Definition des Objekt-Verhaltens bei auftretenden Exceptions

Das Verhalten eines TcCOM-Objekts bei auftretenden Exceptions können Sie für die Initialisierungsphase und für die UpdatePhase getrennt unter den Properties der Klasse TwinCAT.ModuleGenerator.ProjectExportConfig einstellen:

```
exportConfig.ClassExportCfg{ nModuleCount }.TcCom.UpdateExceptionHandling = 'CallerExceptions';
exportConfig.ClassExportCfg{ nModuleCount }.TcCom.InitExceptionHandling = 'CallerExceptions';
```

Optionen sind: `CallerExceptions`, `ThrowExceptions`, `SuppressExceptions`, `LogExceptions`, `LogAndHold`, `LogAndCatch`, `LogAndDump`, `LogHoldAndDump`, `LogCatchAndDump`.

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**.



Die Einstellungen für den PLC-FB FB_<ModelName> in der SPS Bibliothek sind unabhängig von den Einstellungen zum TcCOM-Objekt.

`exportConfig.ClassExportCfg{nModuleCount}.PlcFb.UpdateExceptionHandling`

`exportConfig.ClassExportCfg{nModuleCount}.PlcFb.InitExceptionHandling`

Beachten Sie, dass der andere SPS-Funktionsbaustein (FB_<ModelName>_TcCOM) ein Wrapper für ein TcCOM-Objekt [▶ 51] ist und somit bei dessen Verwendung die Exception Einstellungen aus dem Bereich TcCOM gültig sind.

Es stehen insgesamt 9 unterschiedliche Einstellungen zur Verfügung.

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.
- **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.

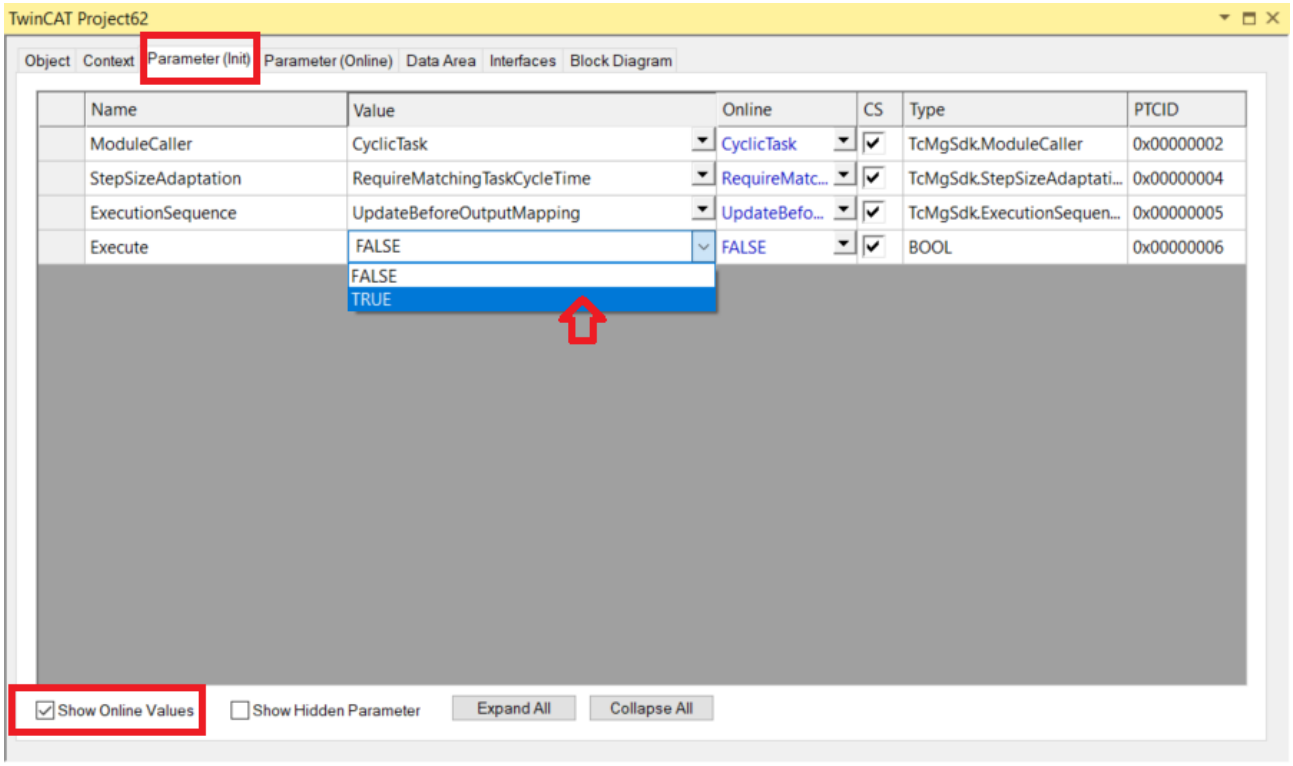
- **LogAndDump, LogHoldAndDump und LogCatchAndDump**
  - Verhalten wie LogExceptions
  - Zusätzlich wird ein Dump File auf dem Laufzeitsystem im TwinCAT Ordner *Boot* abgelegt. Weiteres zu Dump Files siehe hier [▶ 62].

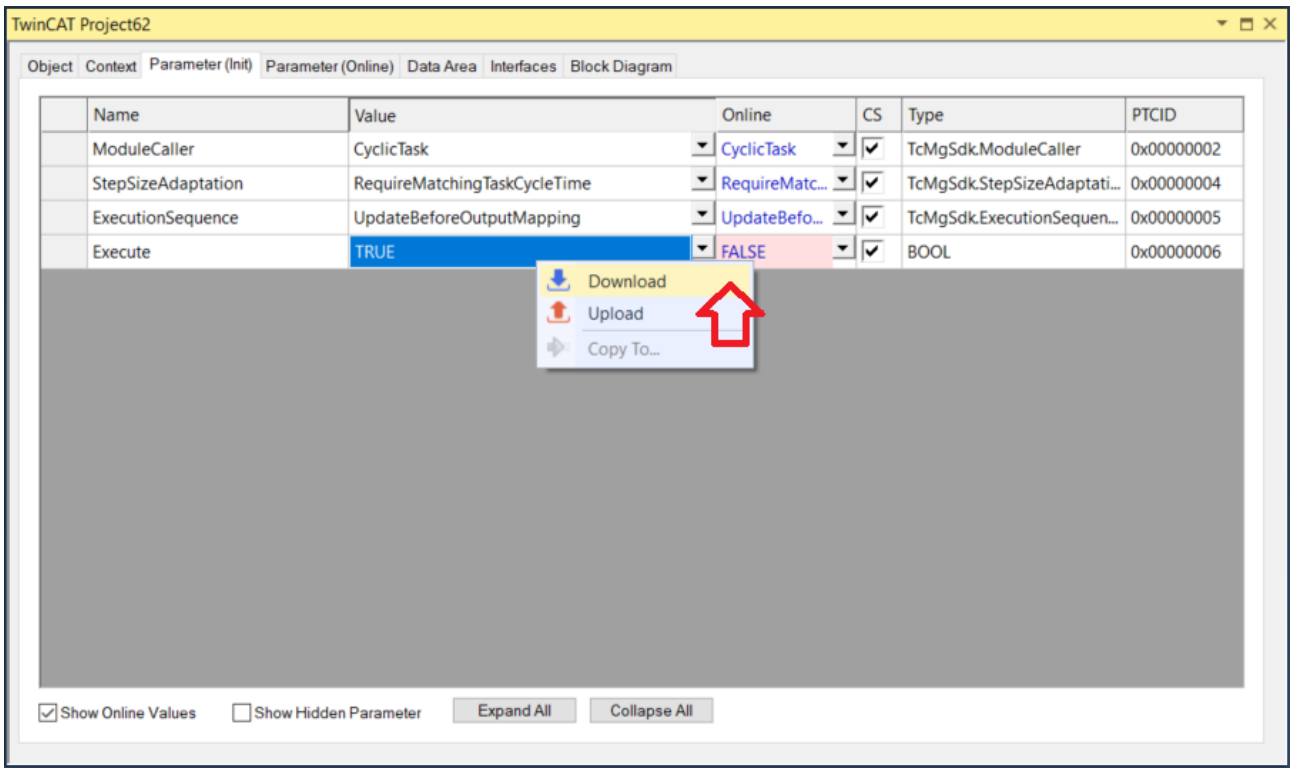
Ausführungsstopp eines TwinCAT-Objekts behandeln

LogAndHold und LogHoldAndDump

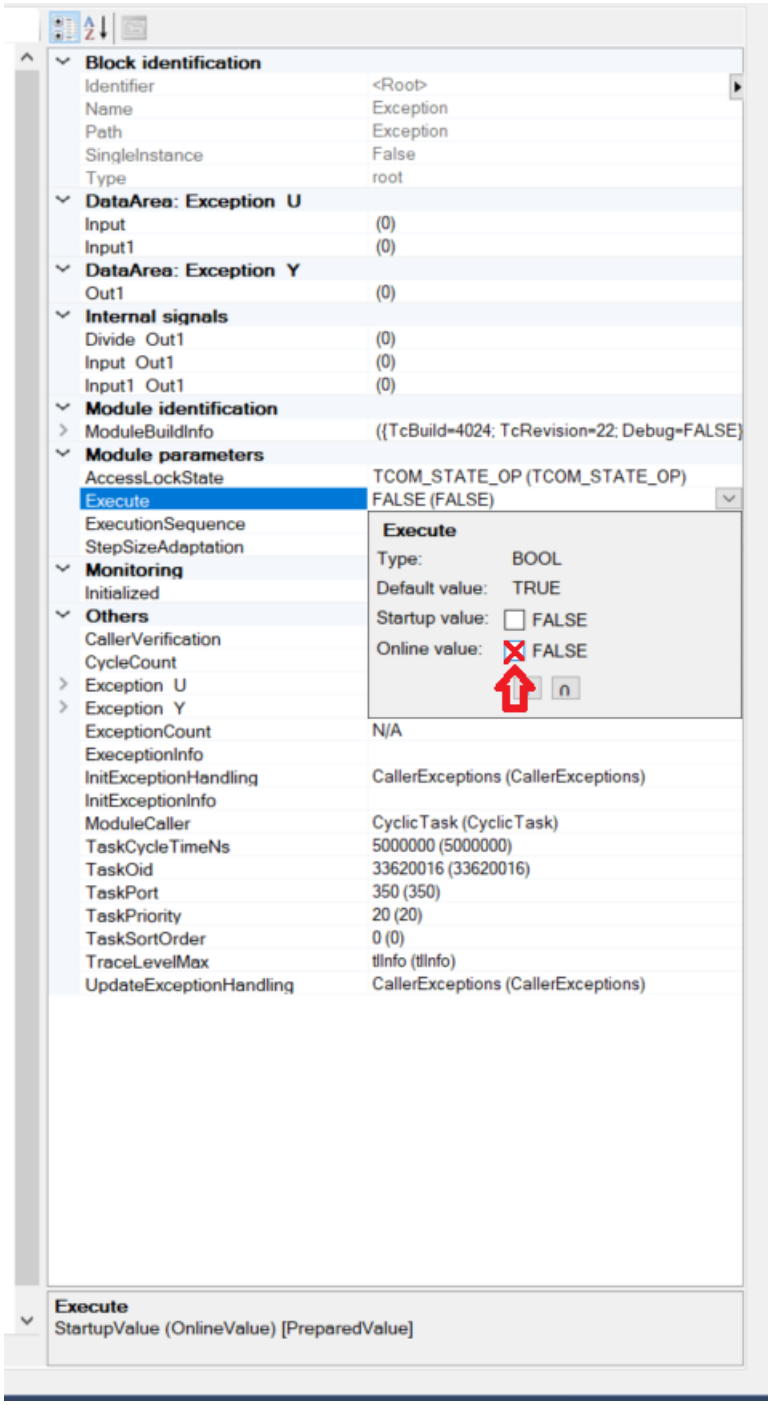
Im Falle einer Exception wird die Ausführung des Codes im betreffenden TcCOM-Objekt durch Setzen des Parameters Execute auf FALSE gestoppt. Der Paramater kann aus dem XAE und per ADS gelesen bzw. geschrieben werden.

Im XAE können Sie auf dem TcCOM-Objekt unter Parameter (Init) dessen Online-Werte anzeigen lassen und verändern.

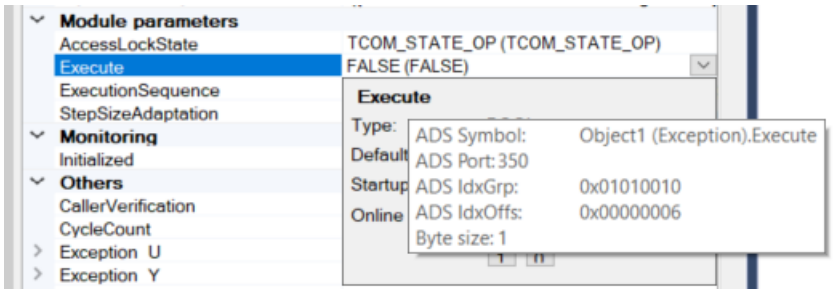




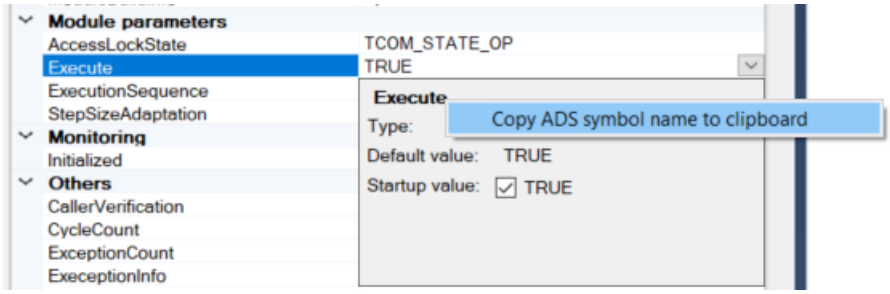
Im Blockdiagramm wird Ihnen der Parameter unter *Module parameters* angeboten.



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.



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 das Modul wieder eine Berechnung durchführen kann. Dies ist notwendig, da interne Zustände nicht mehr konsistent sein können. Eine Reinitialisierung kann nur erfolgen, indem das TcCOM-Objekt in den Zustand „Init“ zurückgefahren und erneut nach OP gefahren wird.

Zur Laufzeit können nur TcCOM-Objekte runtergefahren werden, welche keine Mappings haben, sonst würden aktive Mappings das Herunterfahren blockieren. Eine neue Initialisierung ist nur möglich, indem die gesamte TwinCAT-Laufzeit neu gestartet wird.

Eine flexiblere Alternative ist die Nutzung des TcCOM-Wrapper-FB [▶ 51] in der SPS. 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.

```
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.

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.

8.5 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")
```

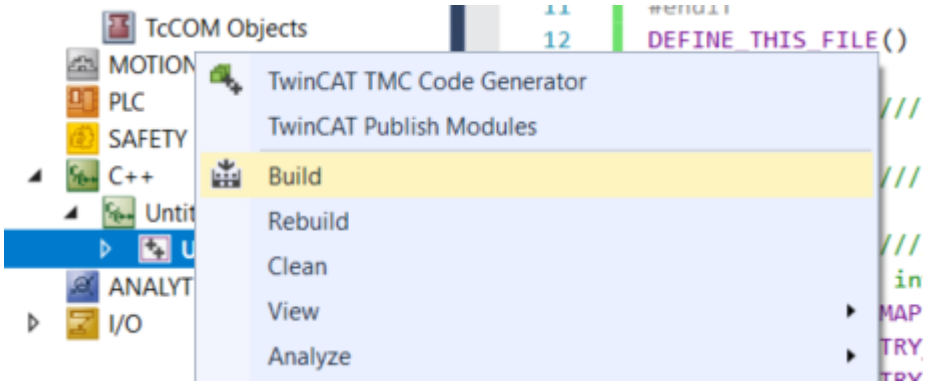
9 FAQ

9.1 Build eines Sample schlägt fehl

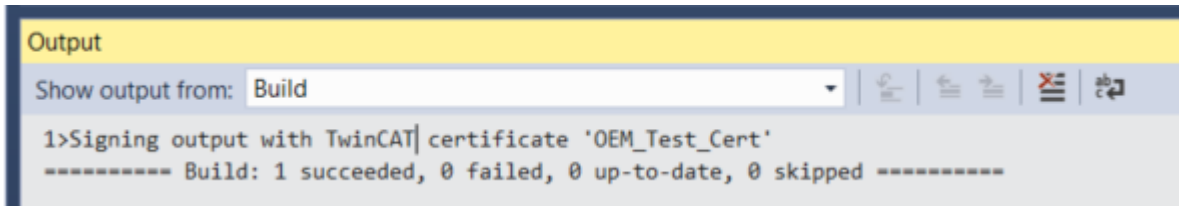
Alle mitgelieferten Beispiele (Liste durch `TwinCAT.ModuleGenerator.Samples.List` im MATLAB® Command Window) sind bei Beckhoff Automation durch Tests geprüft worden. Sollte ein Build eines Samples dennoch nicht erfolgreich durchlaufen, liegt die Vermutung nahe, dass bei der Einrichtung auf Ihrem Engineering 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.



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

Nicht alle Zugriffe, die in MATLAB® 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 der Zugriff auf das Dateisystem des IPC eingeschränkt. Zum Schreiben und Lesen von Dateien aus TwinCAT nutzen Sie: fopen, fclose, fread und fwrite. Siehe Beispiel: `TwinCAT.ModuleGenerator.Samples.Show('FileAccess')`.
- **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. 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.
- **Precompiled libraries**: Es ist möglich, dass bei der Code-Generierung durch den MATLAB® Coder™ kein plattformunabhängiger C/C++ Code erzeugt wird, sondern vorkompilierte Bibliotheken eingebunden werden. In diesen Fällen ist keine Echtzeitausführung in TwinCAT möglich. Die Einstellung coder.HardwareImplementation hilft bei der Prüfung, ob generic C/C++ Code erzeugt

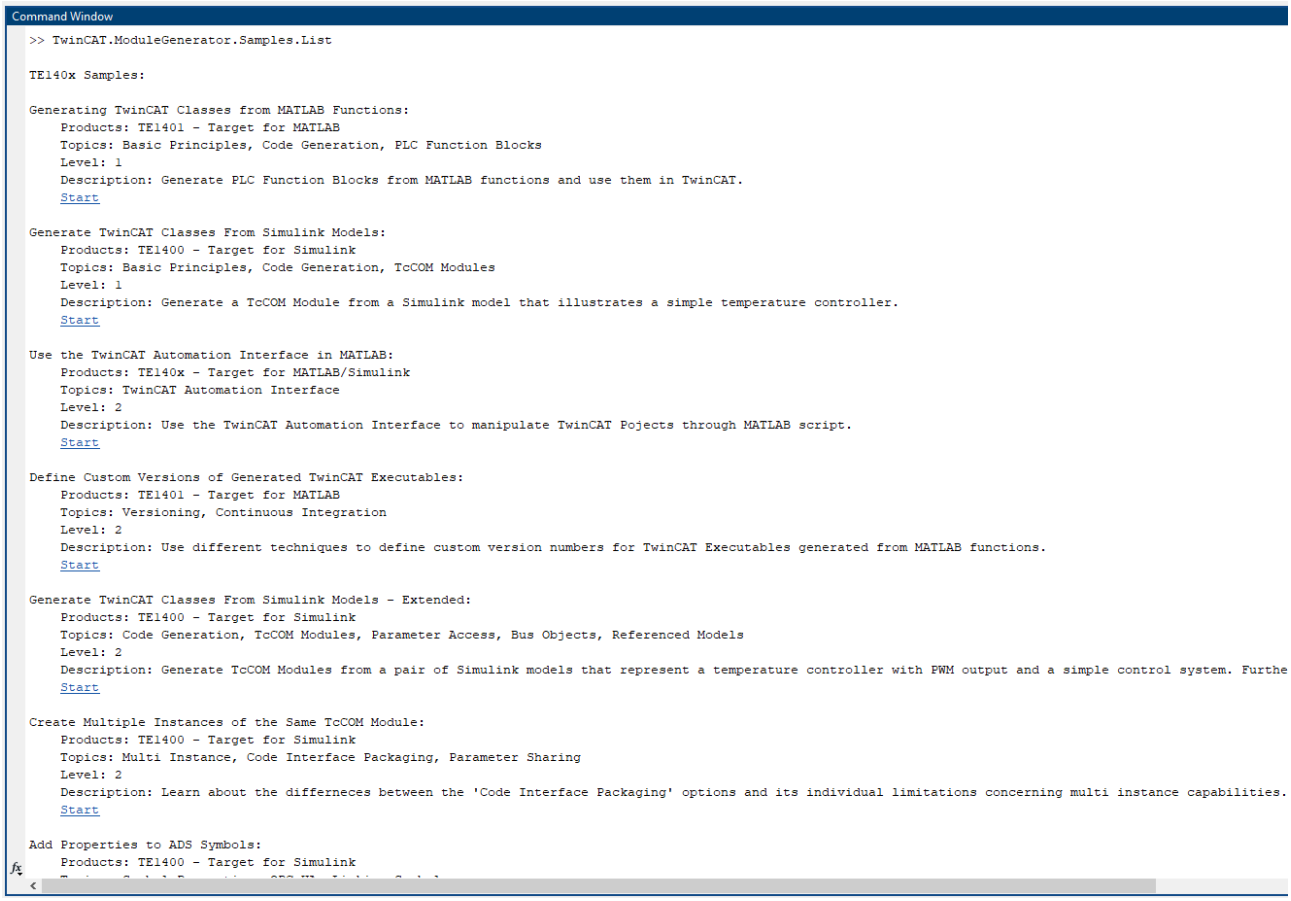
werden kann. Beispielsweise `coder.HardwareImplementation.ProdHWDeviceType = 'Generic->32-bit x86 compatible';` und `coder.HardwareImplementation.TargetHWDeviceType = 'Generic->32-bit x86 compatible';`

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 Click 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

```
TwinCAT.ModuleGenerator.Samples.Show(SampleName)
```

```
TwinCAT.ModuleGenerator.Samples.Start(SampleName)
```

Zum Anzeigen und Starten einzelner Samples. Das Argument SampleName ist als String zu übergeben, z.B.

```
TwinCAT.ModuleGenerator.Samples.Start('BaseStatistics')
```

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.

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

https://infosys.beckhoff.com/content/1031/te1401_tc3_target_Matlab/Resources/5776206091.zip

Trademark statements

Beckhoff®, ATRO® , EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

DSP System Toolbox, Embedded Coder, MATLAB, MATLAB Coder, MATLAB Compiler, MathWorks, Predictive Maintenance Toolbox, Simscape, Simscape™ Multibody™, Simulink, Simulink Coder, Stateflow and ThingSpeak are registered trademarks of The MathWorks, Inc.

Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Mehr Informationen:
www.beckhoff.com/te1401

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com