

BECKHOFF New Automation Technology

Manual | EN

TE1400

TwinCAT 3 | Target for Simulink®

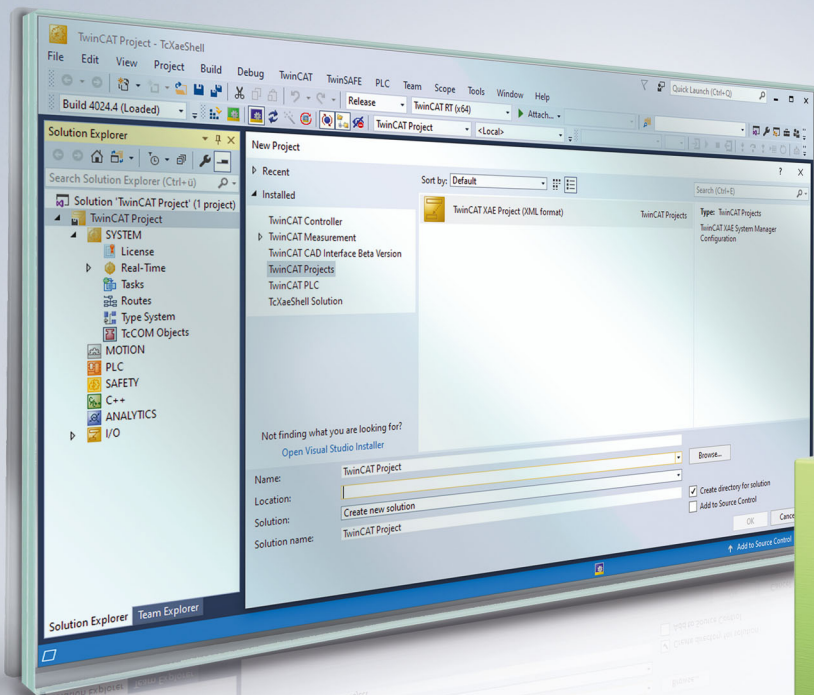


Table of contents

1	Foreword	7
1.1	Notes on the documentation	7
1.2	For your safety	7
1.3	Notes on information security.....	9
2	Overview	10
3	Up to version 1.2.xxxx.x	12
3.1	Installation	12
3.2	Licenses	14
3.3	Quickstart	15
3.4	TwinCAT Library in Simulink®	18
3.5	Parameterization of the code generation in Simulink.....	21
3.5.1	Module generation (Tc Build).....	22
3.5.2	Data exchange (Tc Interfaces).....	26
3.5.3	External mode (Tc External Mode)	30
3.5.4	Advanced settings (Tc Advanced)	33
3.6	Application of modules in TwinCAT	39
3.6.1	Parameterization of a module instance.....	39
3.6.2	Executing the generated module under TwinCAT	41
3.6.3	Calling the generated module from a PLC project	43
3.6.4	Using the ToFile block.....	48
3.6.5	Signal access via TwinCAT 3 Scope	54
3.7	FAQ.....	55
3.7.1	Does code generation work even if I integrate S-Functions into my model?	55
3.7.2	Why do FPU/SSE exceptions occur at runtime in the generated TwinCAT module, but not in the Simulink model?	55
3.7.3	After updating TwinCAT and/or TE1400 I get an error message for an existing model... ..	56
3.7.4	Why do the parameters of the TcCOM instance not always change after a "Reload TMC/TMI" operation?.....	56
3.7.5	After a "Reload TMC/TMI" error "Source File <path> to deploy to target not found.....	57
3.7.6	Why do I have a ClassID conflict when I start TwinCAT?	58
3.7.7	Why can the values transferred via ADS differ from values transferred via output mapping?	58
3.7.8	Are there limitations with regard to executing modules in real-time?.....	58
3.7.9	Which files are created automatically during code generation and publishing?.....	59
3.7.10	How do I resolve data type conflicts in the PLC project?	60
3.7.11	Why are the parameters of the transfer function block in the TwinCAT display not identical to the display in Simulink?.....	61
3.7.12	Why does my code generation/publish process take so long?	61
3.8	Examples	61
3.8.1	TemperatureController_minimal	62
3.8.2	Temperature Controller	68
3.8.3	SFunStaticLib.....	77
3.8.4	SFunWrappedStaticLib	83
3.8.5	Module generation Callbacks.....	88

4	From version 2.x.xxxx.x	89
4.1	Installation	89
4.2	Licenses	92
4.3	Setting up driver signing.....	93
4.3.1	User certificates for delivery without test mode.....	96
4.4	Quick start	102
4.5	TwinCAT Library in Simulink®	108
4.5.1	TwinCAT Input and Output modules	109
4.5.2	TwinCAT Environment View	116
4.5.3	TwinCAT File Writer	116
4.6	Overview of automatically generated files.....	117
4.7	Parameterization of the code generation in Simulink®.....	121
4.7.1	Overview table of all configuration parameters.....	124
4.7.2	Parameterization of the code generation via an m-file.....	131
4.7.3	Bundling of several models in one TwinCAT driver	133
4.7.4	Sharing created TwinCAT objects.....	135
4.7.5	Creation of versioned drivers	137
4.7.6	Configuration of data access to data of a TcCOM object.....	143
4.7.7	Shared memory between TcCOM instances	150
4.7.8	Creating a module with OEM license query.....	156
4.7.9	Integration of own C/C++ code	158
4.7.10	Configuration of the TMX file properties.....	159
4.7.11	Multitask, Concurrent Execution and OpenMP	159
4.7.12	Symbol Properties and Attribute Pragmas	166
4.7.13	Available placeholders	172
4.7.14	Working with callbacks.....	182
4.8	Application of modules in TwinCAT	183
4.8.1	Working with the TcCOM module	183
4.8.2	Working with the PLC library.....	203
4.8.3	Debugging.....	214
4.8.4	Connecting to External mode.....	217
4.8.5	Exception handling.....	220
4.8.6	Using Realtime Monitor time stamps	230
4.9	FAQ.....	230
4.9.1	Change model parameters at runtime.....	230
4.9.2	Build of a sample fails	231
4.9.3	Problems with the block diagram representation in TwinCAT XAE	231
4.9.4	Can I use TE1400 version 1.2.x and version 2.x at the same time?.....	232
4.9.5	What is the difference between "Build" and "Generate code"?.....	232
4.9.6	I can't change the parameters of a module in TwinCAT	232
4.9.7	Mapping is lost with Reload TMI/TMC	232
4.9.8	Integrating the block diagram controls in .NET	233
4.9.9	Observable signals in the TwinCAT block diagram.....	235
4.9.10	Using Simulink® Strings.....	236
4.9.11	Are there limitations with regard to executing modules in real-time?.....	238
4.9.12	Message: Failed to copy repository	239

4.10	Samples	239
4.10.1	TwinCAT Automation Interface: use in MATLAB®.....	240
4.10.2	Integrating the block diagram controls	242
4.10.3	Try out created TwinCAT objects yourself	244

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.

For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.

The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without notice.

No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

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

If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT®

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The distribution and reproduction of this document as well as the use and communication of its contents without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.

Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings**⚠ DANGER**

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment**NOTICE**

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our <https://www.beckhoff.com/secguide>.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at <https://www.beckhoff.com/secinfo>.

2 Overview

TE1400 TwinCAT Target for Simulink®

With the TwinCAT 3 Target for Simulink® it is possible to make models developed in Simulink® usable in TwinCAT 3. Various toolboxes such as SimScape™ or Stateflow™ or DSP System Toolbox™ can be integrated in Simulink®. Embedded MATLAB® function blocks are also supported. The models are automatically transcoded in C/C++ code with the aid of the Simulink Coder™ and transformed into TwinCAT objects with TwinCAT 3 Target for Simulink®. These TwinCAT objects can then be executed in real-time in the TwinCAT runtime. These TwinCAT objects can be TcCOM objects for direct instantiation and linking with real-time tasks or function blocks for instantiation and processing in a PLC project.

Areas of application and application examples

The areas of application of TwinCAT Target for Simulink® can be summarized by the following keywords:

- Rapid Control Prototyping
- Real-time simulation
- SiL (**S**oftware in the **L**oop) simulation
- HiL (**H**ardware in the **L**oop) simulation
- Model-based design
- Model-based monitoring

The following application examples are intended to illustrate possible areas of application:

- **Example 1: Rapid Control Prototyping**

During the simulation development stage in Simulink®, a controller is implemented as a Simulink® model, which is integrated into the simulation model of the control loop via *model referencing*. This enables the closed control loop to be designed and tested in a simulation (**M**odel in the **L**oop simulation (MiL)). Before the controller model is compiled unchanged into a TwinCAT module via mouse click, which then operates as real-time controller for an actual system. Since standard Simulink® function blocks are used as inputs and outputs, they can be used in the higher-level Simulink® model as well as in the module generated later in TwinCAT.

- **Example 1a: Real-time simulation of a controlled system**

The controlled system is also implemented as a Simulink® model, which is integrated into the model of the closed control loop via *Model Referencing*. The TcCOM module generated from this is used to perform a real-time simulation, in which a controller implemented in IEC61131-3, C++ or Simulink® can be tested.

- **Example 2: Real-time simulation of a machine/Virtual commissioning**

A TcCOM module is generated from a machine model created in Simulink®. This can be used to test a PLC program in real-time, before the actual machine is connected (virtual commissioning). Depending on the configuration, SiL or HiL simulations can be performed in this way. See also Overview.

- **Example 2a: SiL simulation of plant components**

According to VDI/VDE 3693 Part 1, Software in the loop (SiL) is defined as a stage following MiL simulation, in which the control code is available as series code. The series code can be executed in an emulated controller, for testing against a system simulation model.

According to this definition, there are two options for a SiL simulation of systems (components) with TwinCAT:

- The system model remains in Simulink® and uses ADS to communicate with the series code, which is executed in the TwinCAT runtime. See also TE1410 Interface for MATLAB Simulink.
- The system model is also compiled into a TcCOM module and executed in real-time (see example 1a).

- **Example 2b: HiL simulation of system components**

According to VDI/VDE 3693 Part 1, Hardware in the loop (HiL) is defined as an advanced testing stage, in which the actual target control code is tested on an actual controller against a system model. The latter is executed in a simulation tool, which acts as a bus device and therefore uses the actual communication networks of the automation system for communication with the actual controller.

Based on this definition, the model of the system or the system components is converted to TcCOM modules and executed on a second Industrial PC, taking into account the real-time requirements. The

function Overview is used to configure this IPC such that it makes the mirrored process image available to the actual controller. In this way, it is possible to use the actual controller and the actual configuration to communicate with the "simulation IPC" in hard real-time.

- **Example 3: Model-based monitoring of system components**

In many cases, measured variables are of interest that are not directly accessible or would result in excessive effort/costs. By using a physically representative model with measurable input variables, non-measurable variables can still be determined. An example is temperature measurement at locations that are inaccessible, such as the permanent magnet temperature in an electric motor. Based on a thermal model of the motor, the temperature can be estimated by means of secondary parameters such as electric current, rotational speed and cooling temperature.

Further Information

Technical short videos

- [TwinCAT Target for Simulink](#)

Product descriptions

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

Customer application videos

- [Overview of customer applications](#)
- [Success Story Vintecc bv](#)
- [Success Story Magway](#)

Website for MATLAB® and Simulink® with TwinCAT 3: <http://www.beckhoff.com/matlab>

3 Up to version 1.2.xxxx.x

TE1400 Target for Simulink® versions lower than 1.2.xxxx.x support MATLAB R2010b to MATLAB R2019a.

TE1400 Target for Simulink® versions higher than 2.x.xxxx.x support MATLAB R2019a and higher.

3.1 Installation

System requirements

Initially, the same requirements apply to the target for MATLAB®/Simulink® as for TwinCAT 3 C/C++. For a detailed description of the TwinCAT 3 C/C++ requirements, please refer to Chapter 4, "Requirements", of the TwinCAT 3 C++ manual.

In the following sections, these requirements are only referred to briefly, not in detail.

On the engineering PC

- Microsoft Visual Studio 2010 (with Service Pack 1), 2012, 2013, 2015 or 2017 Professional, Premium, Ultimate or Community Edition
 - Installation under Windows always with right-click **run as admin...**
 - For Visual Studio 2015 check the Visual C++ checkbox during installation
 - For Visual Studio 2017, manually select "Desktop development with C++"
- Microsoft "Windows Driver Kit" version 7.1.0 (only required for TwinCAT versions older than TwinCAT 3 build 4024.0)
 - It is sufficient to install the "Build Environments".
 - Set the environment variable (variable name WINDDK7, variable value <Installation directory> e.g. C:\WinDDK\7600.16385.1)
- TwinCAT 3 XAE

On the runtime PC

- IPC or Embedded CX PC with Microsoft operating system based on "Windows NT kernel" (Win XP, Win 7 and corresponding embedded versions, Win 10)
- TwinCAT 3 XAR
 - TwinCAT 3.0 only supports 32-bit operating systems on the target
 - TwinCAT 3.1 supports 32-bit and 64-bit operating systems. If the target is a x64 system, the created drivers must be signed. The TE1400 supports OS driver signing. See "x64: driver signing" in the TwinCAT 3 C++ manual

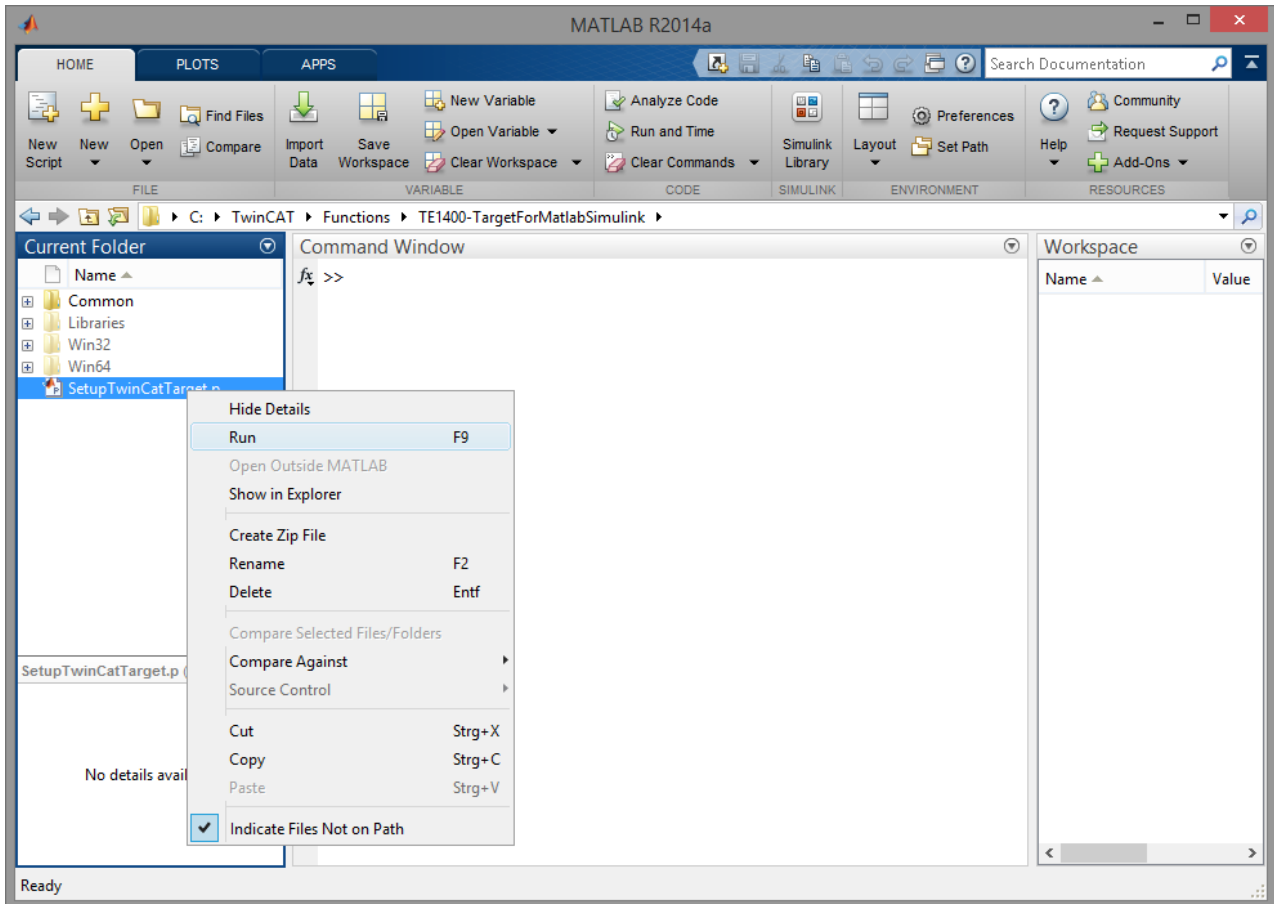
In addition to the above requirements, which originate from the requirements of TwinCAT 3 C/C++, the following requirements apply to the engineering PC:

- MATLAB®/Simulink® R2010b up to and including R2019a. As of and including R2019a, the use of TE1400 version 2.x.xxxx.x is recommended.
- Simulink Coder™ (in MATLAB® versions prior to R2011a: Real-Time Workshop®)
- MATLAB Coder™ (in MATLAB® versions prior to R2011a: part of the Real-Time Workshop®)
- Installing the TE1400 Target for MATLAB®/Simulink®

Setup instructions

- ✓ Install one of the supported Visual Studio versions, if not already installed. Pay attention to the installation of the C++ components.
1. Start TwinCAT 3 Setup, if it does not already exist.
 - ⇒ If a Visual Studio and a TwinCAT installation already exists but the Visual Studio version does not meet the requirements mentioned above (e.g. Visual Studio Shell or Visual Studio without Visual C++), you first have to install a suitable Visual Studio version (install Visual C++, if necessary). Then run TwinCAT 3 Setup to integrate TwinCAT 3 into the new (or modified) Visual Studio version.

2. If necessary, install the Microsoft Windows Driver Kit (see Installation "Microsoft Windows Driver Kit (WDK)" in the TwinCAT 3 C/C++ manual).
The order in which the Windows Driver Kit was installed is irrelevant.
 3. If you do not have a **MATLAB®** installation on your system, install it. The order in which MATLAB® was installed is irrelevant.
 4. Start the setup *TE1400-TargetForMatlabSimulink* to install the TE1400.
The TE1400 is installed in the TwinCAT folder, i.e. it is separate from the MATLAB® installation. A MATLAB® version that exists on the system can be linked to the TE1400 according to point 6.
 5. Start MATLAB® as administrator and execute `%TwinCAT3Dir%\Functions\TE1400-TargetForMatlabSimulink\SetupTwinCatTarget.p` in MATLAB®.
- ⇒ A setup window opens. See the following section.



- The p-file links the MATLAB® version used to the TE1400. If a new MATLAB® version is installed on the system, the p-file must be executed in the new version.
- If a new TE1400 version is installed on top of an existing TE1400 version, the p-file should also be run again.

i User Account Control

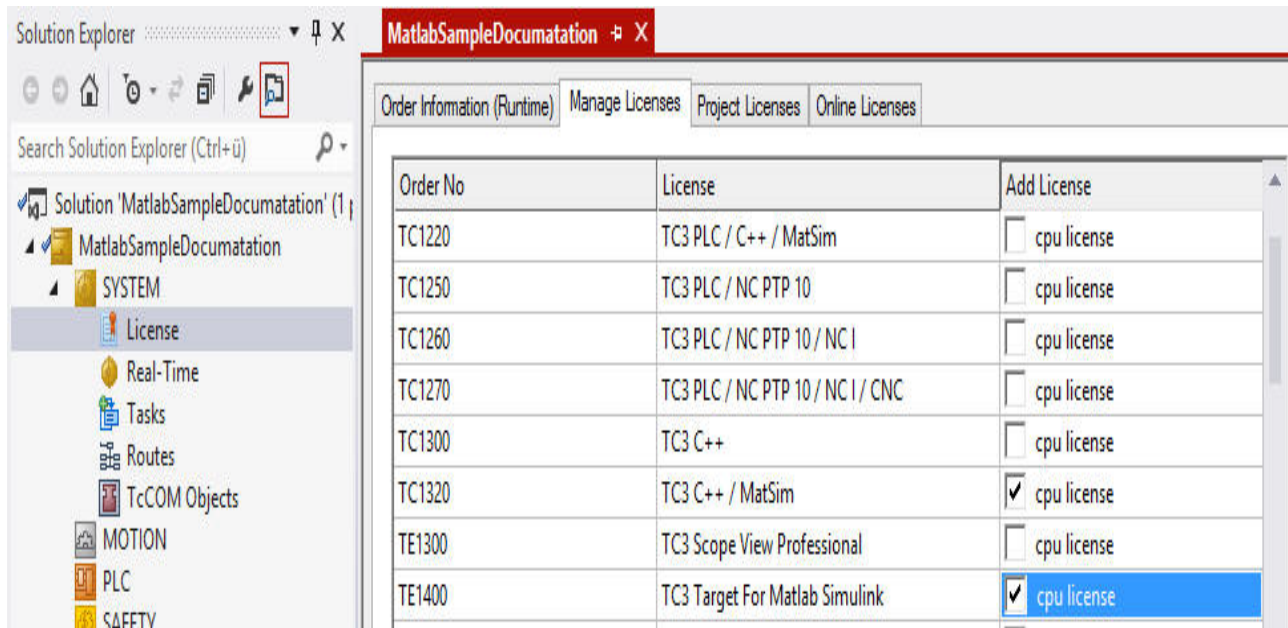
If MATLAB® is executed in a system with activated User Account Control (UAC) without administrator rights, the MATLAB® path cannot be stored permanently. In this case, SetupTwinCatTarget.p must be executed every time MATLAB® is started, since otherwise some files required for generating TwinCAT modules cannot be found.

i Driver signing for targets with x64 operating system

To use an x64-operating system as runtime PC, the drivers must be signed. Details can be found in the TC3 C++ manual under [Driver signing](#).

3.2 Licenses

Two licenses are required to use the full functionality of the TE1400 Target for MATLAB®/Simulink® (see Ordering and activation of TwinCAT 3 standard licenses).



Order No	License	Add License
TC1220	TC3 PLC / C++ / MatSim	<input type="checkbox"/> cpu license
TC1250	TC3 PLC / NC PTP 10	<input type="checkbox"/> cpu license
TC1260	TC3 PLC / NC PTP 10 / NC I	<input type="checkbox"/> cpu license
TC1270	TC3 PLC / NC PTP 10 / NC I / CNC	<input type="checkbox"/> cpu license
TC1300	TC3 C++	<input type="checkbox"/> cpu license
TC1320	TC3 C++ / MatSim	<input checked="" type="checkbox"/> cpu license
TE1300	TC3 Scope View Professional	<input type="checkbox"/> cpu license
TE1400	TC3 Target For Matlab Simulink	<input checked="" type="checkbox"/> cpu license

Required licenses for TE1400

TE1400: TC3 Target-For-Matlab-Simulink (module generator license)

This license is required for the **engineering system** for the module generation from MATLAB®/Simulink®. For testing purposes, the module generator of the TE1400 can be used in demo mode without a license.



A fully functional 7-day trial license is not available for this product.

Restrictions in the demo version

The module generator has the following restrictions without a license.

Allowed are models with maximum :

- 100 function blocks
- 5 input signals
- 5 output signals



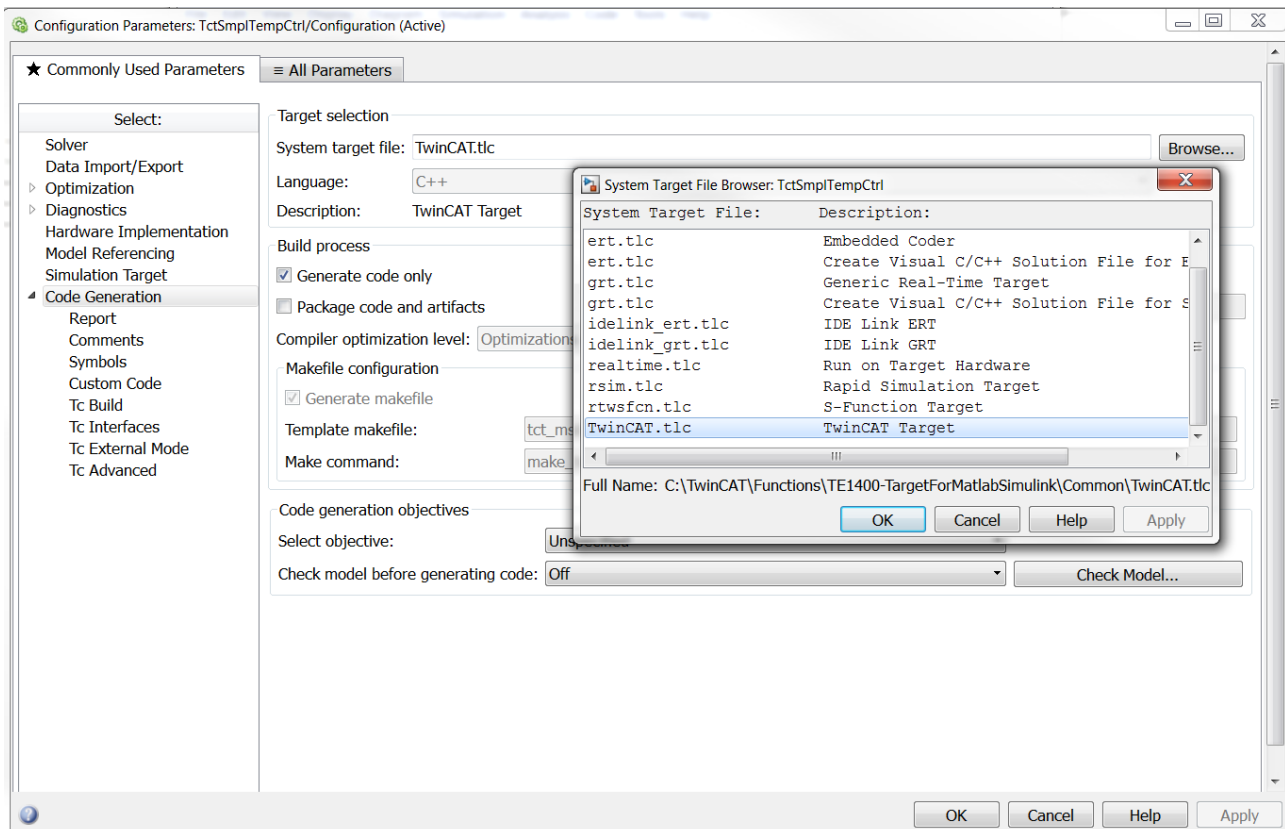
Modules created with a demo license may only be used for non-commercial purposes!

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

The license TC1320 (or TC1220 with PLC license) is required to start a TwinCAT configuration with a module generated from Simulink®. Without activated license, the module and consequently the TwinCAT system cannot be started. In this case you get error messages relating to the license violation. You can generate a 7-day trial license, which enables initial tests without purchasing the license.

3.3 Quickstart

Configuration of the Simulink® model



The coder settings can be accessed via the Model Explorer in the **View** menu of the Simulink environment, via **Code Generation** (previously **Real-Time Workshop**) > **Options** in the **Tools** menu, or via the **Configuration Parameters** dialog. In the tree view, select **Configuration -> Code Generation**. Then, open the **General** tab and select *TwinCAT.tlc* as "System target file". Alternatively, use the **Browse** button to open a selection window and select **TwinCAT Target** as target system.

In addition, a fixed-step solver must be configured in the solver settings, to ensure real-time capability of the Simulink model.

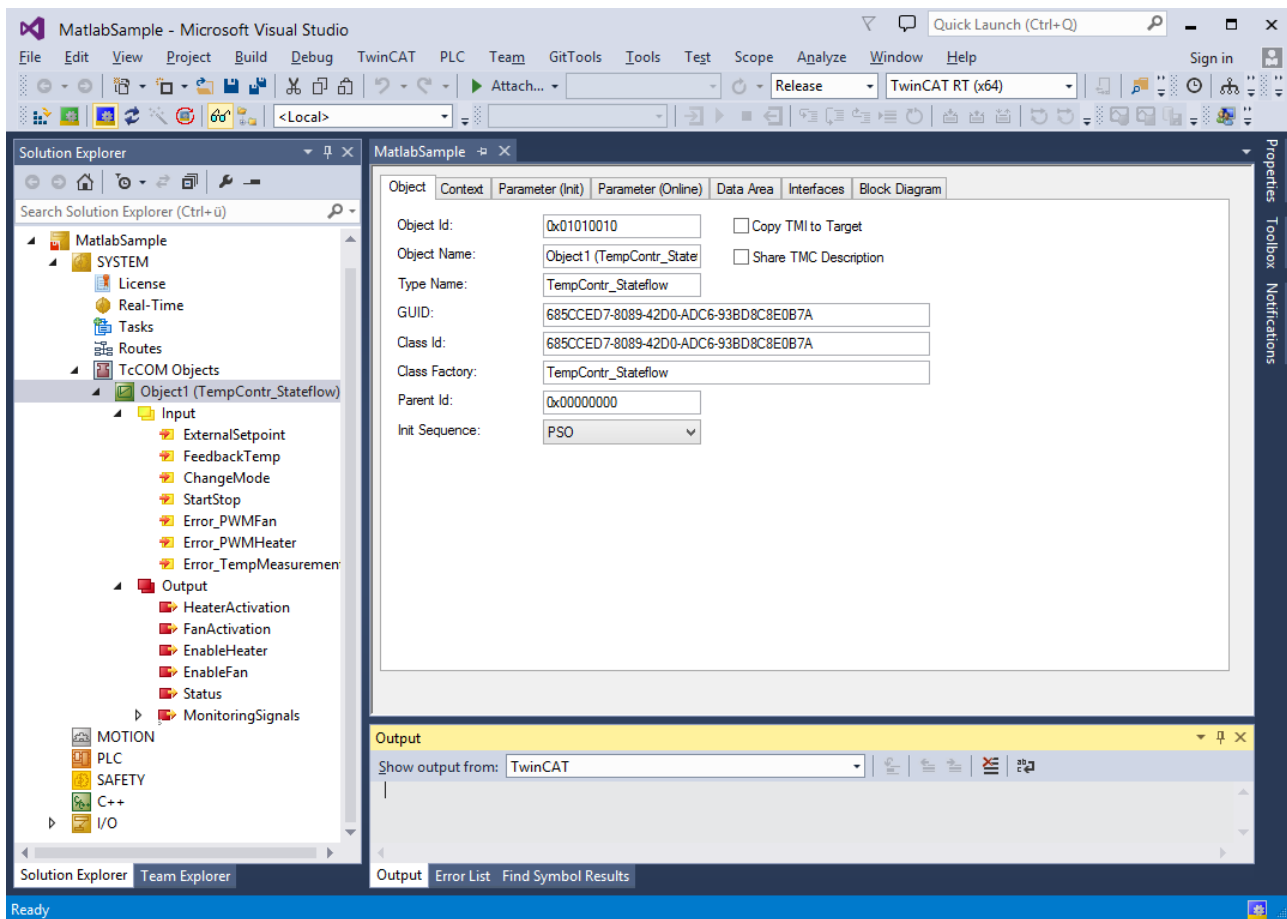
Generating a TcCOM module from Simulink

Generation of the C++ code or the TcCOM module can be started with the **Build** button (or **Generate code**) in the lower section of the window for the code generator options. If the option **Publish module** is activated under **TC Build** (default setting), the build process for generating executable files starts immediately after the C++ code has been generated, and a TcCOM module is created. Otherwise, the module generator stops after the C++ code and the project file for Visual Studio™ has been generated. For further information please refer to [Publish Module](#) [► 22].

Integration of the module in TwinCAT 3

After module export with "Publish"

If the option **Publish Module** was enabled before the module was generated, the module will already be available in compiled form. A TwinCAT Module Class (TMC file) was created during this process and can be instantiated directly in the project. A TwinCAT Module Instance (TMI) is referred to as TcCOM object or module instance below.

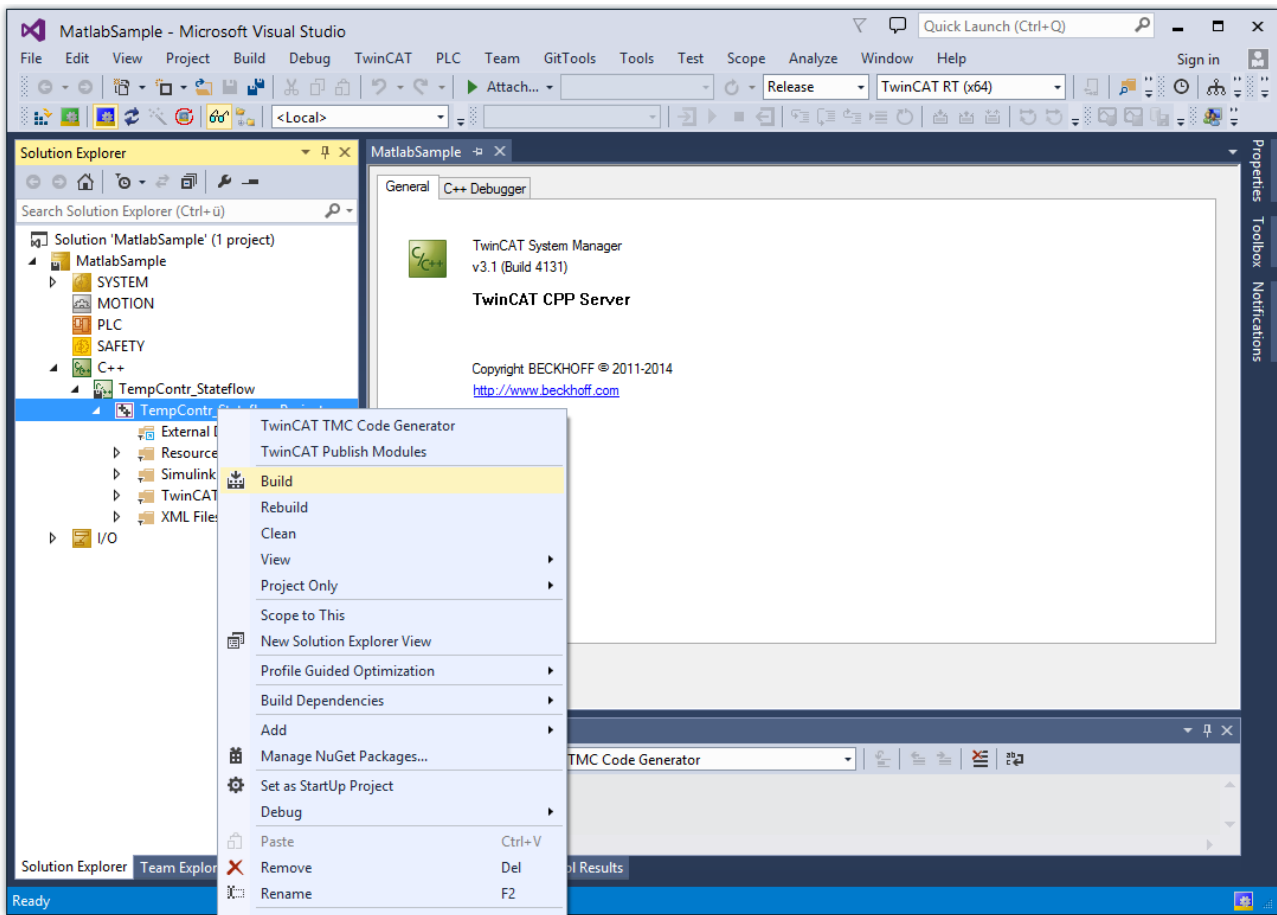


Instances of the generated module can be integrated in a TwinCAT3 project any number of times. TcCOM objects are usually appended to the node **TcCOM Objects** via the **Add New Item** context menu. A selection list of the modules that are available on the system can be obtained via this option. The modules generated by Simulink can be found under **TE1400 Module Vendor > Generated Modules**.

Compiling the code without "Publish"

If the option **Publish Module** was disabled prior to the module generation, the C/C++ code pertaining to the module still has to be compiled, before it can be executed.

The C++ project can be inserted into the TwinCAT project via **Add Existing Item** in the context menu of the C++ node. The C++ project file is located in the Build directory "`<MODELNAME>_tct`" and has the name of the module with the file extension `.vcxproj`. The module can then be created in the TwinCAT development environment (XAE):



Multiple instances of the module can be created via the context menu of the parent node of the C++ project. These are listed under the project node. Further information about the build process of C++ projects in the TwinCAT development environment (XAE) and about the instantiation of modules created in this way can be found in section "Creating a TwinCAT3 C++ project".

Cyclic call by a real-time task

Object Context Data Area Interfaces Block Diagram

Context: 0

Depend On: Task Properties

Need Call From Sync Mapping

Data Areas:

- 0 'ExternalInputs'
- 1 'ExternalOutputs'
- 2 'BlockIO'

Data Pointer:

Interface Pointer:

Result:

ID	Task	Name	Priority	Cycle Time (µs)	ADS Port
0	02000105	Task 1	5	5000	350

Under the **Context** tab of the module instance, you will find all the contexts of the module, which have to be assigned to a real-time task. If **Depend on: Task Properties** are assigned automatically to tasks for which the cycle time and the priority match the displayed values. If there are no matching tasks or if the setting **Depend on: Manual Config** was selected, tasks can be created under **System Configuration -> Task Management**. Further information on cyclic calling of module instances can be found in section "[Cyclic Call \[p. 41\]](#)".

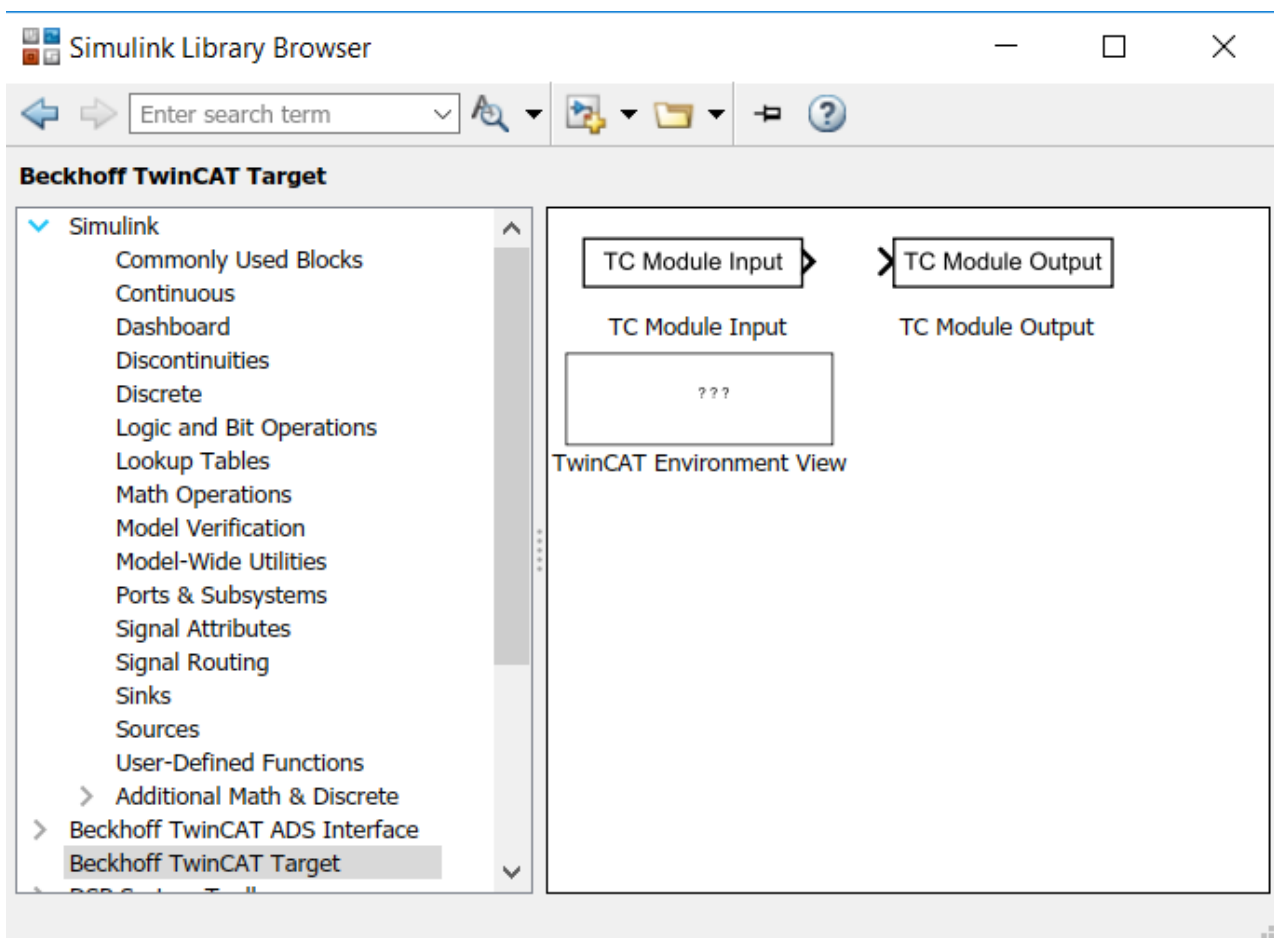
Data exchange with other modules or fieldbus devices

The process images of the module inputs and outputs can be expanded below the module instance node in the TwinCAT development environment. Here you will find all ports that have been defined in the Simulink model with the aid of the function blocks **In1** and **Out1** (components of the standard Simulink library). All signals within this process images can be linked to signals of another process images via the **Change Link** context menu.

3.4 TwinCAT Library in Simulink®

In Simulink®, *TwinCAT-specific* input and output function blocks can (not mandatory) be used to define the signals/buses connected to these function blocks as inputs or outputs in the subsequent TcCOM in TwinCAT. A common way is to use the standard input ports (In) and output ports (Out) of Simulink®. This is usually also the *best practice* way, unless the additional functions of the TwinCAT-specific input and output function blocks described below are required.

The TwinCAT-specific input and output function blocks can be found in the **Library Browser** under **Beckhoff TwinCAT Target**.



If you use the input and output function blocks provided by Beckhoff, you will benefit from the following additional functionalities, compared to the standard Simulink® input and output ports:

- You can also define signals and buses from subsystems directly as inputs or outputs for TcCOM, without first transferring the signals/buses from the subsystem to the top system.

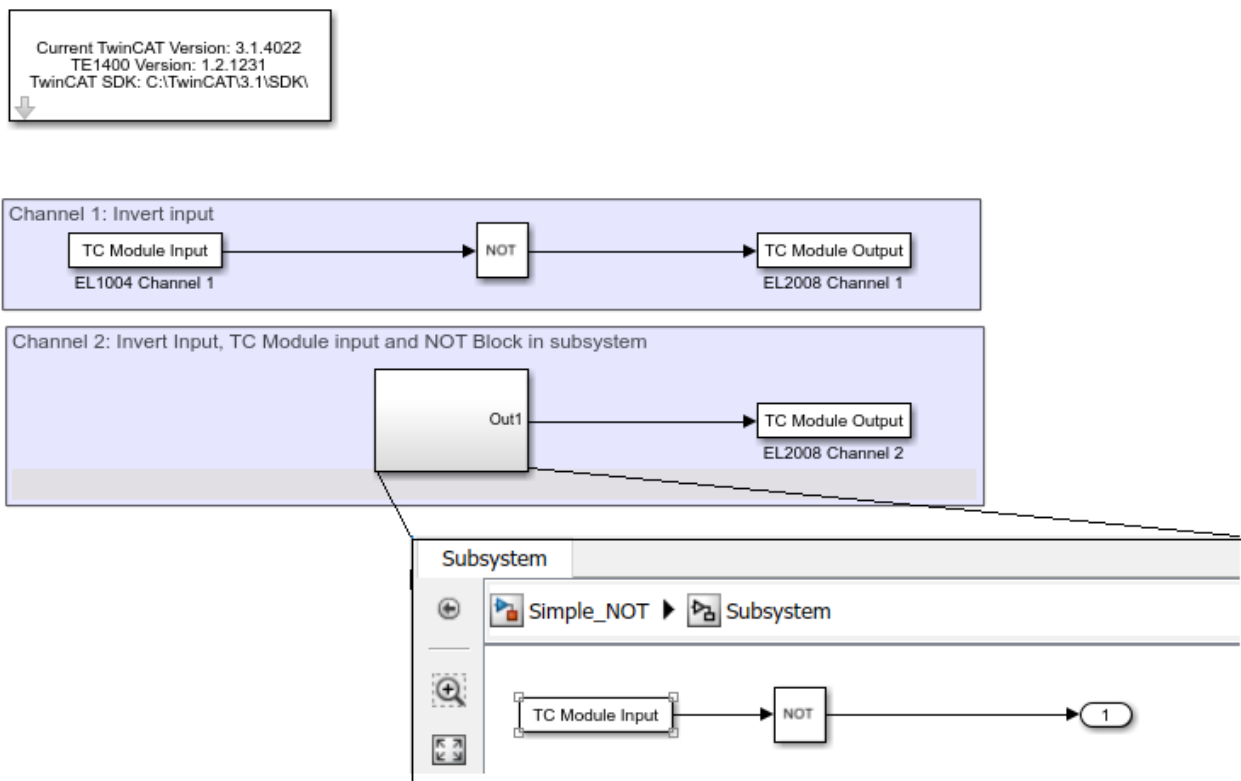
- You can (not mandatory) store an automatic mapping to other TcCOM or I/Os in the function block parameters, so that the mapping is executed directly and automatically when the TcCOM is instantiated.
- You can use initial values for inputs. To do this, set the Value of the Tc Module Inputs to any value.

When using automatic mapping, please note that if the TcCOM is instantiated more than once in TwinCAT, you will end up with a mapping conflict which you must resolve by manual mapping. This option is therefore not recommended for multiple instantiations.

In addition to TwinCAT-specific input and output function blocks, a TwinCAT Environment View Block is also provided. This can be used in the Simulink® environment to simply display TwinCAT and TE1400 versions on the system.

Example

A Simulink® model is created, which outputs two negated inputs. An input is placed in a subsystem, see figure below.

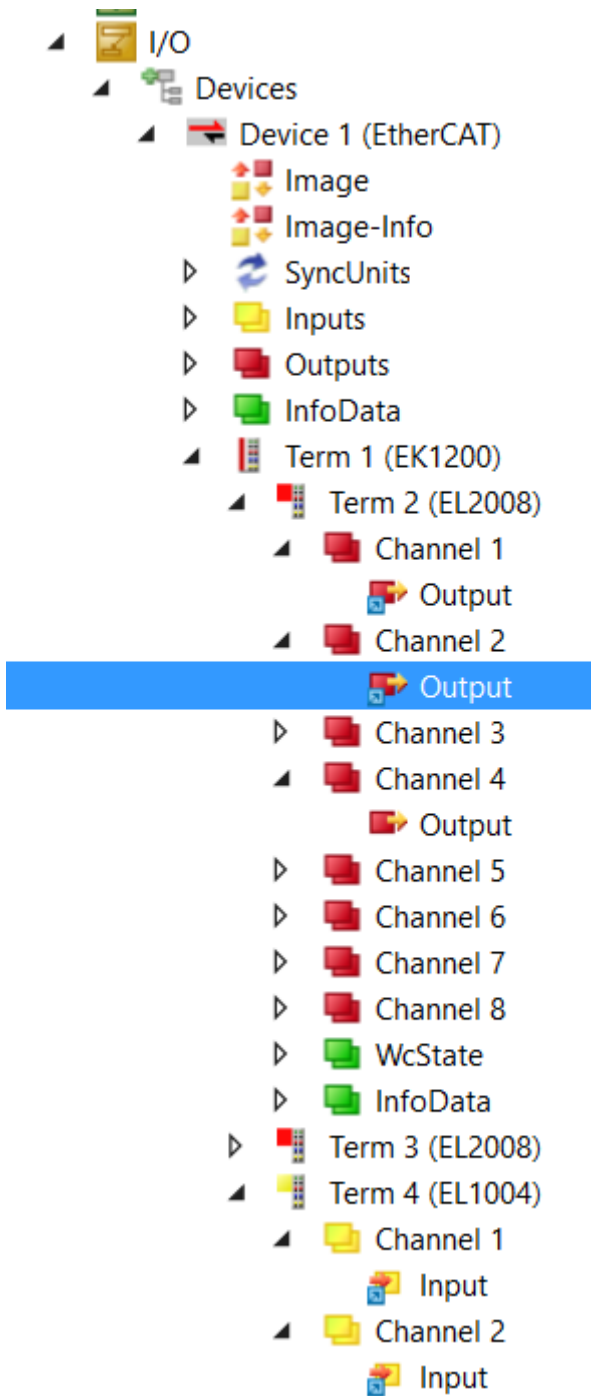


The inputs and outputs of the model are automatically mapped to digital inputs and outputs via the properties of the TC Modules Input and Output. The necessary tree items can be found in TwinCAT 3 by selecting the desired input or output and then copying the string in the **Variable** tab under **Full Name**.

Variable	Flags	Online
Name:	Output	
Type:	BIT	
Group:	Channel 4	Size: 0.1
Address:	26.3	User ID: 0
Linked to...		
Comment		
ADS Info:	Port: 11, IGrp: 0x3040010, IOffs: 0xC10000D3, Len: 1	
Full Name:	TIID^Device 1 (EtherCAT)^Term 1 (EK1200)^Term 2 (EL2008)^Channel 4^Output	

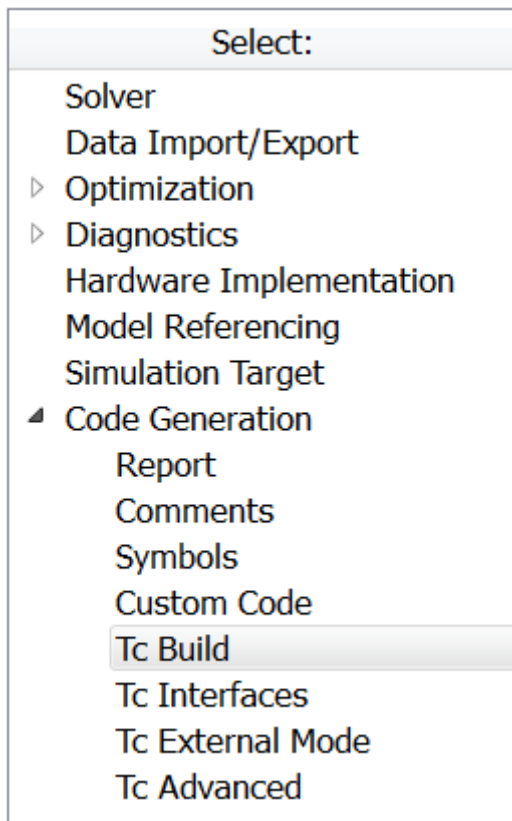
A list of shortcuts for quick access can be found in the documentation of the **Automation Interface > API > ITcSysManager > ITcSysmanager::LookupTreeItem**.

When the Simulink® model described above is compiled and integrated into TwinCAT 3, a mapping to the corresponding inputs and outputs is automatically created. The automatically generated mappings are marked with a blue symbol to distinguish them from manual mapping, while manual mapping symbols appear white.



3.5 Parameterization of the code generation in Simulink

Within MATLAB® Simulink® a wide range of configuration setting options for the TcCOM module to be generated are available. To this end the tree structure under Code Generation is extended with the entries Tc Build, Tc Interfaces, Tc External Mode and Tc Advanced. Many parameters can be modified in TwinCAT 3 at the module instance level; see [Application of modules in TwinCAT \[► 39\]](#).



The corresponding setting options are described below.

i Tooltips

Hover with the cursor over the text fields of the dialog boxes to bring up a detailed description of the option as a tooltip (pop-up window).

3.5.1 Module generation (Tc Build)

The Publish mechanism can be used to compile TwinCAT C++ projects for several TwinCAT platforms and export them to a central Publish directory. In the first step, the modules for all selected platforms are built. Then, all files required for instantiation and execution of the module under TwinCAT 3 are copied to the Publish directory.

The section "Export TC3 modules" under **TC3 Engineering > C/C++ > Modules Handling** describes how the publish mechanism is applied to TC3 C++ modules. The following section describes how Simulink has to be configured in order to export TwinCAT modules directly after the code has been generated with the aid of the Publish mechanism.

Publish directory

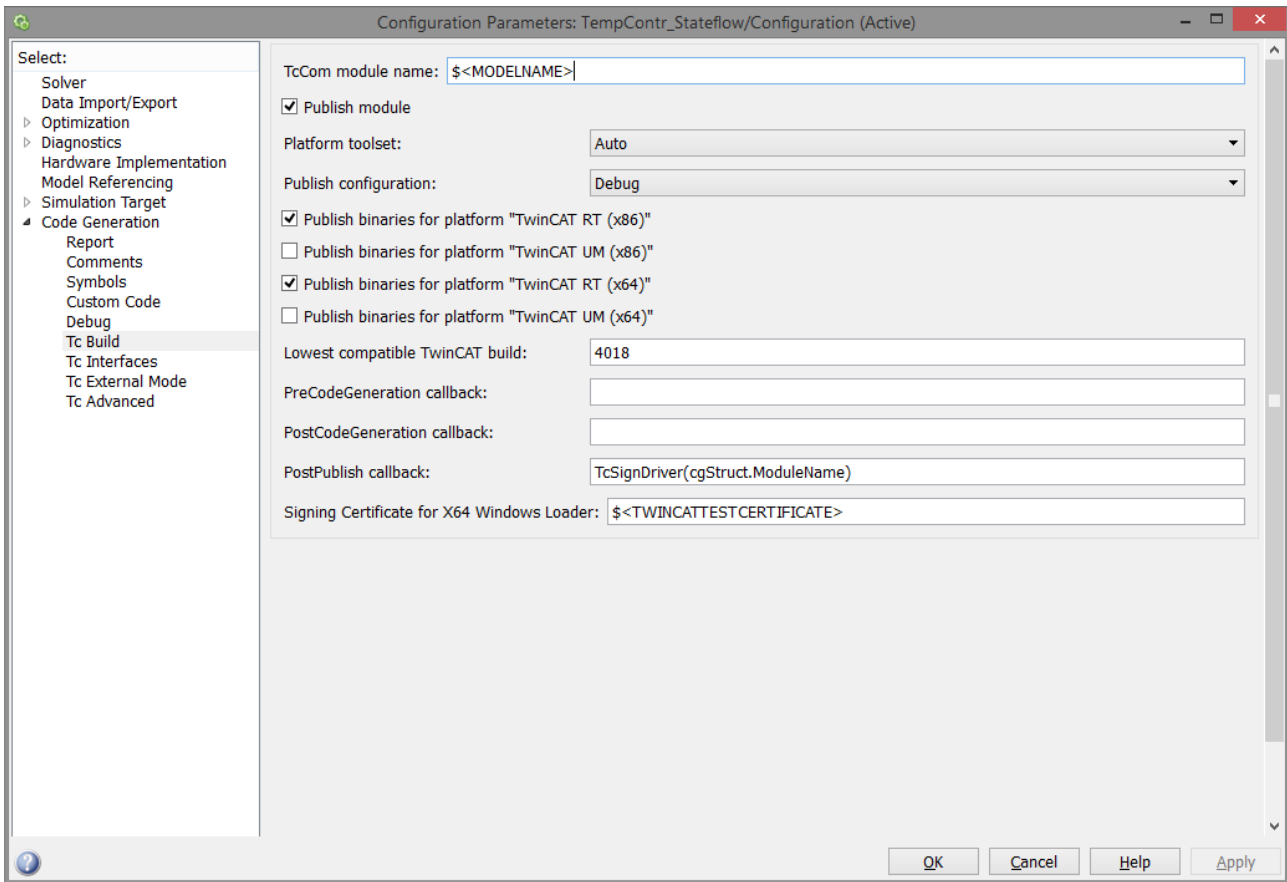
The files relating to exported modules are copied to the directory `%TwinCat3Dir%CustomConfig\Modules\<MODULENAME>\`. To instantiate the module on another development PC, this folder can be copied to the appropriate directory on the other computer.

Application

It makes sense to publish modules only once they have reached a stage at which they are only rarely modified, and if they are used in several TwinCAT projects. Otherwise, it may be more efficient to integrate the whole C++ project in the TwinCAT project, e.g. if the Simulink model is still under development and regular modifications are therefore to be expected, or if the module is only used in a special TwinCAT project.

Configuration in Simulink

The publish mechanism can be configured at **Tc Build**: (Export options for TwinCAT modules)



Publish module:

- **disabled:** The module generator is stopped once the C++ project has been generated. The generated C++ project has to be compiled manually, so that the module can be executed in TwinCAT 3. This can be done directly from the TwinCAT development environment, once the generated C++ project has been integrated into the TwinCAT project.
- **enabled:** "Publish" is executed automatically, once the C++ project has been generated. The module is then available on the development PC in compiled form for all TwinCAT projects and can be instantiated directly in the TwinCAT development environment (XAE). The other Publish settings relate to the target platforms for which the module is intended. Due to the sequential building for the different platforms, these settings can have a significant effect on the module generation duration.

i "Generate code only" option

The option **Generate code only** (in the **Build process** part of the window for the **Code Generation** settings) has no function, because the TwinCAT Publish mechanism is used instead of the MATLAB Make mechanism.

Platform toolset:

Enables selection of a certain platform toolset (compiler) for building the module drivers. The options available for selection depend on the VisualStudio versions installed on the system. If **Auto** is selected, a compiler is selected automatically.

Publish configuration:

Select **Debug** here in order to enable debugging of the exported block diagram in TwinCAT 3. If no debugging is required, e.g. in a release version, **Release** can be selected here.

- **Publish binaries for platform „<PLATFORMNAME>“:**
 - Select all TwinCAT platforms on which the module is intended to run. The drivers are then built successively for all selected platforms.

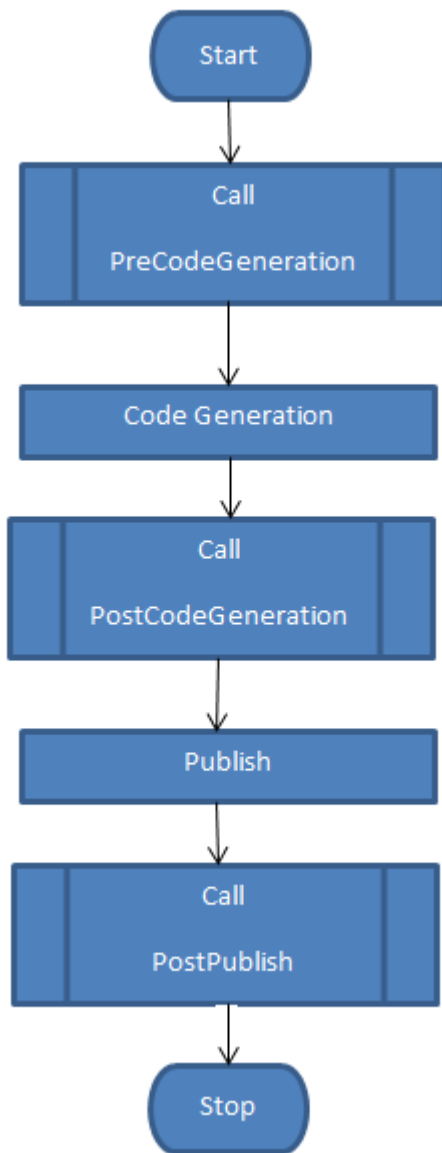
- **Lowest compatible TwinCAT build:**

- Enter the build number of the oldest TwinCAT version, with which the module is still to be compatible. If the module is subsequently used with an older TwinCAT version, it may fail to start. Also, the generated code may be uncompileable, if the SDK of an older TwinCAT version is used. The following table provides an overview of main TwinCAT version-dependent properties of the generated module:

Property	TC3 Build	Description
Large DataAreas	< 4018	DataAreas > 16 MB are not supported
	>= 4018	DataAreas > 16 MB utilize the data areas of several DataArea IDs, using the "OBJDATAAREA_SPAN_xxx" macros.
Project subdirectory "_ModuleInstall"	< 4018	During instantiation of a module that was previously exported via "Publish", only the TMC description is imported into the TwinCAT project. The module instance still refers to files within the <u>Publish directory</u> [► 22]. To load the TwinCAT project on other development computer, the Publish directories of the modules in use have to be copied manually into the corresponding directories of the other computers. Otherwise the project cannot be activated, and the block diagram is not displayed.
	>= 4018	During instantiation of an exported module, all associated files are copied to subdirectory "_ModuleInstall" of the project directory. The project can now be opened on another development PC (even if it is compressed as an archive), without having to copy additional files manually. Another advantage is that the files in the <u>Publish directory</u> [► 22] are now completely decoupled from the TwinCAT project. The module description, which is part of the TwinCAT project after it is instantiated, and the associated files (e.g. the drivers) are kept consistent. Files in the Publish directory can be overwritten, while the project uses a different version of the module up to "Reload TMC" and can still be re-activated on a target system.

PreCodeGeneration / PostCodeGeneration / PostPublish callback:

MATLAB functions can be entered here, which are called before and after the code generation, or after Publish: (callback sequence)



To execute model- or module-specific actions, the structure cgStruct can be accessed here. It contains the following subelements:

Name	Value	Comment
ModelName	Name of the Simulink model	
StartTime	Return value of the MATLAB function "now ()" at the start of the code generation	
BuildDirectory	Current build directory	From "PostCodeGeneration"
ModuleName	Name of the generated TwinCAT module	From "PostCodeGeneration"
ModuleClassId	ClassId of the generated TwinCAT module	From "PostCodeGeneration"
<UserDefined>	Additional custom fields can be added to the structure, in order to transfer additional information to subsequent callbacks.	

For example, in the simplest case additional information could then be output between the individual module generation phases:

```

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

See also: [Callback samples \[► 62\]](#)

Signing Certificate for x64 Windows Loader:

Defines the certificate used for signing of the driver for the "TwinCAT RT (x64)" platform. The default value \$ (TWINCATTESTCERTIFICATE) refers to the environment variable TWINCATTESTCERTIFICATE, which is described under "Driver signing" (**TC3 Engineering > C/C++ > Preparation**). Alternatively, the certificate name can be entered directly here, or different placeholders can be used, depending on the desired signing behavior:

Value	Behavior
\$(ENVIRONMENT VARIABLE)	This placeholder is resolved at an early stage during code generation. The value is written into the generated C++ project. If the specified environment variable is not found, the code generation process terminates with a corresponding error message.
\$(ENVIRONMENT VARIABLE)	This placeholder is not resolved until the generated C++ project is built. If the environment variable is not found, only a warning appears. The x64 driver can then still be built, although it cannot be loaded by the Windows loader on a target system.
CertificateName	The name of the certificate is written into the generated C++ project.
	If the field remains empty, only a warning appears. The x64 driver can then still be built, although it cannot be loaded by the Windows loader on a target system.

3.5.2 Data exchange (Tc Interfaces)

Configuration in Simulink

Depending on the Simulink model, there are several groups of internal variables in addition to the input and output variables. ADS access and the process image type can be configured as required. These settings affect how the variables are linked with other process images in the TwinCAT development environment, and how they can exchange data. The following groups can be configured:

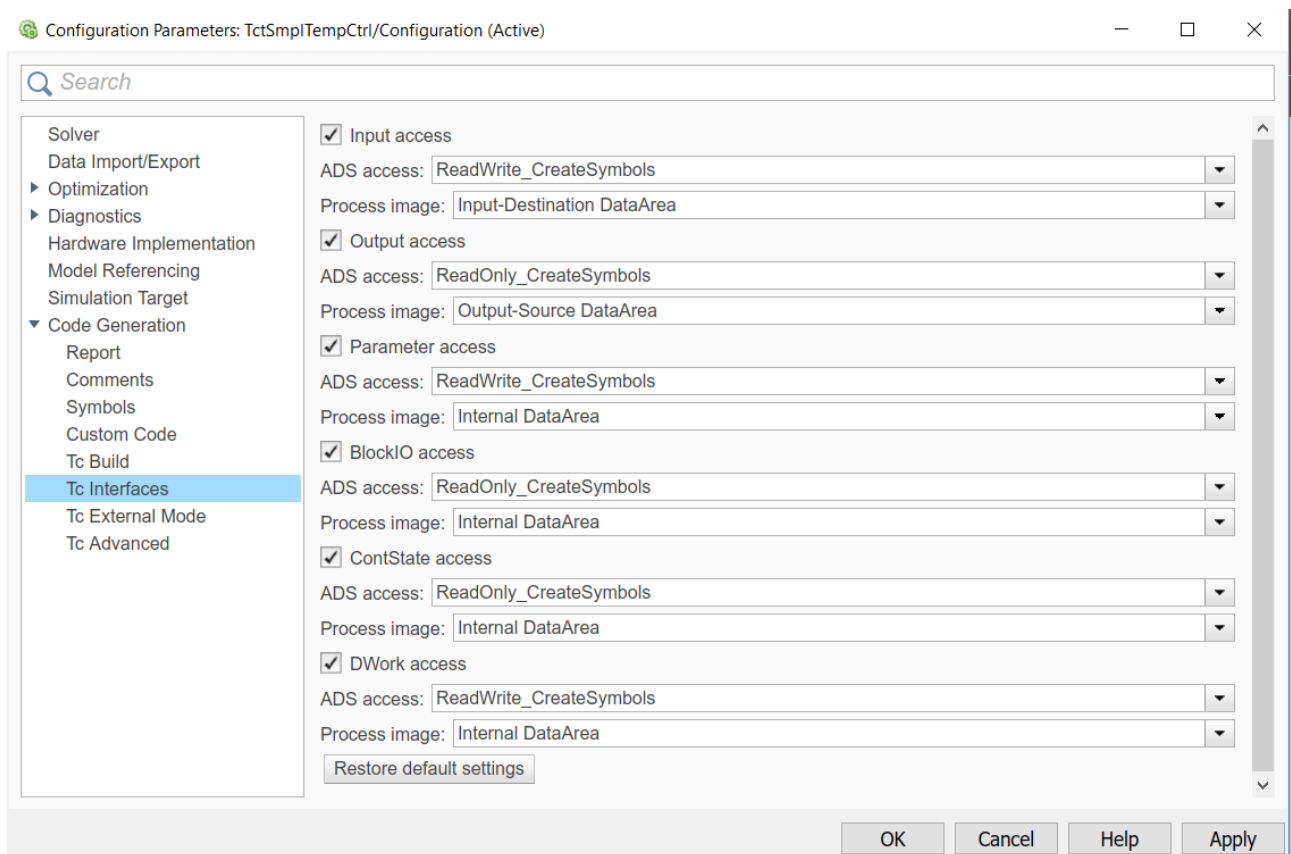
Group	Description
Input	Model inputs
Output	Model outputs
Parameter	Model-specific parameters: Parameters of Simulink blocks that can be "set"
BlockIO	Global output signals of Simulink blocks: Internal signals for which a "test point" was set or which were declared as global due to code optimizations of the code generator.
ContState	Continuous state variables
DWork	Time-discrete state variables

On the configuration page **TC Interface** in the coder settings, there are several possible settings for each of these variable groups. The options available for selection depend on the group, i.e. not all the described options are available in all cases:

Parameter	Options	
<i>GROUP</i> access (checkbox)	TRUE	The module enables access to variables of this group.
	FALSE	The module denies access to variables of this group.
ADS access	Only relevant if " <i>GROUP</i> access"=TRUE	
	No ADS access	No ADS access
	ReadOnly_NoSymbols	No ADS write access, ADS communication is only possible via the Index group and the Index Offset information
	ReadWrite_NoSymbols	Full ADS access, ADS communication is only possible via the Index group and the Index Offset information
ReadOnly_CreateSymbols	No ADS write access, ADS symbol information is generated	

Parameter	Options	
	ReadWrite_CreateSymbols	Full ADS access, ADS symbol information is generated
Process image	Only relevant if "GROUP access"=TRUE	
	No DataArea	Link to DataArea or I/O: no Link to DataPointer: no
	Standard DataArea	Link to DataArea or I/O: no Link to DataPointer: yes
	Input-Destination DataArea	Link to DataArea or I/O: yes Link to DataPointer: yes
	Output-Source DataArea	Link to DataArea or I/O: yes Link to DataPointer: yes
	Internal DataArea	Link to DataArea or I/O: no Link to DataPointer: no
	Retain DataArea	Enables linking to a "retain handler" (see retain data [▶ 27]) for remanent data management.

The above setting options can be realized in the following mask via the corresponding drop-down lists.



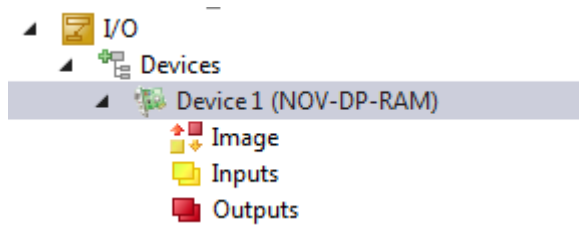
The **restore default settings** option can be used to undo all changes and reset the default settings. The default settings are shown in the diagram above.

3.5.2.1 Retain data

This section describes the option to make data available even after an ordered or spontaneous system restart. The NOV-RAM of a device is used for this purpose. The EL6080 cannot be used for these retain data, because the corresponding data must first be transferred, which leads to corresponding runtimes. The following section describes the retain handler, which stores data and makes them available again, and the application of the different TwinCAT 3 programming languages.

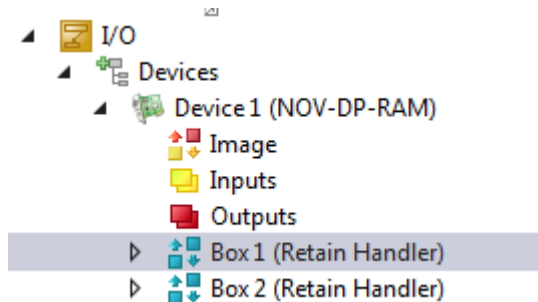
Configuring a retain device

1. The retain data are stored and made available by a retain handler, which is part of the NOV-DP-RAM device in the IO section of the TwinCAT solution. Create a NOV-RAM DP Device in the IO area of the



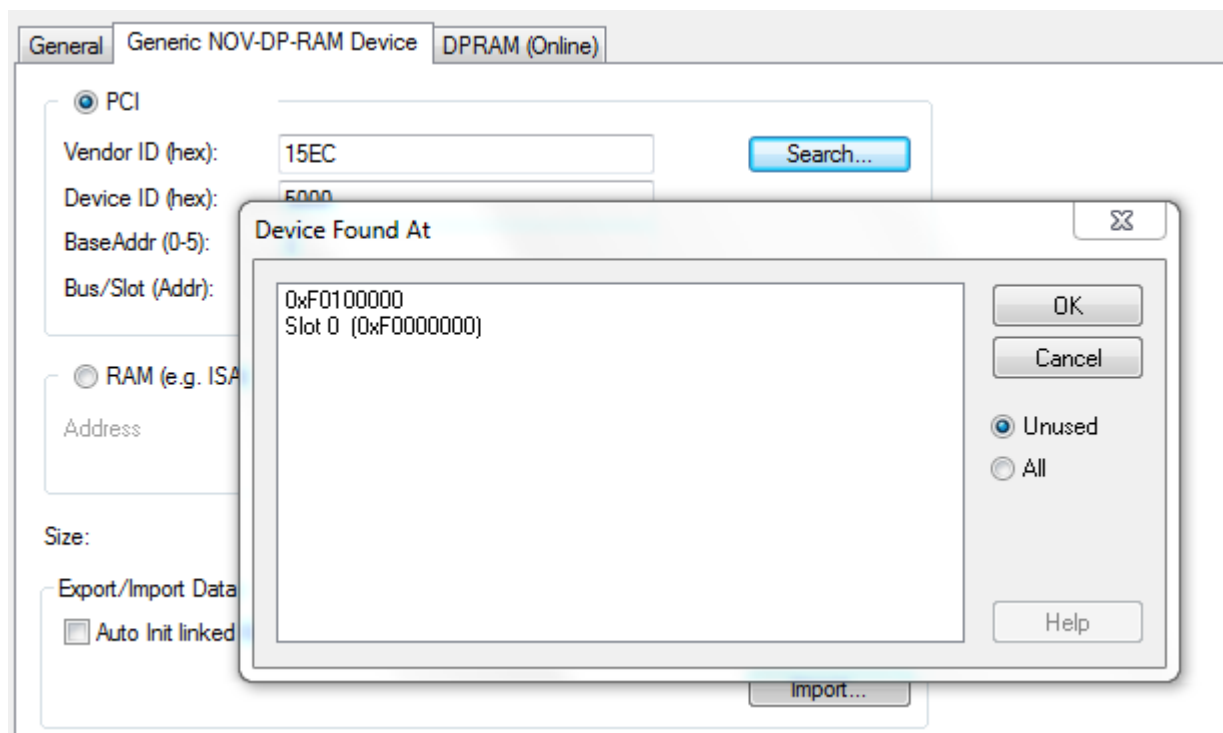
Solution.

2. Create one or more Retain Handler below this device.



Storage location: NOV-RAM

3. Configure the NOV-DP RAM device. In the **Generic NOV-DP-RAM Device** tab, use **Search...** to define the area to be used.



4. An additional retain directory for the symbols is created in the TwinCAT boot directory.

Using the retain handler with a PLC project

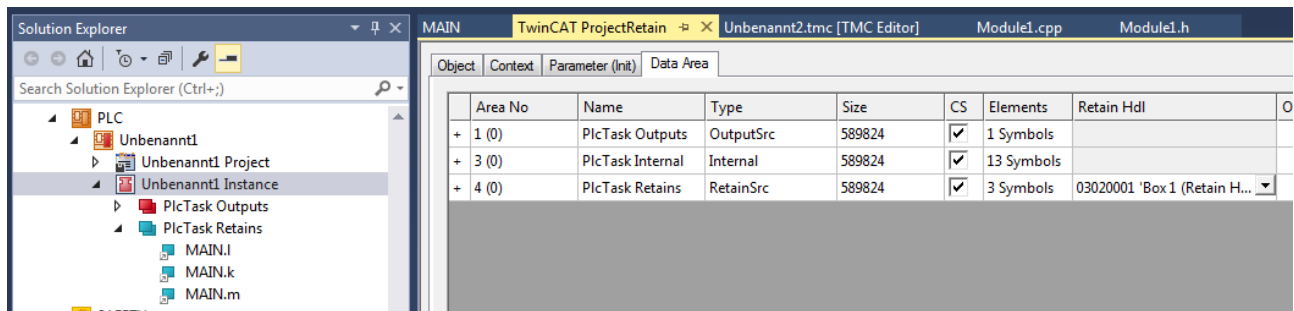
In a PLC project the variables are either created in a VAR RETAIN section or identified with the attribute TcRetain.

```
PROGRAM MAIN
VAR RETAIN
  l: UINT;
  k: UINT;
END_VAR
VAR
```

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

Corresponding symbols are created after a "Build".

The assignment to the retain handler of the NOV-DP-RAM device is done in column **Retain Hdl**.

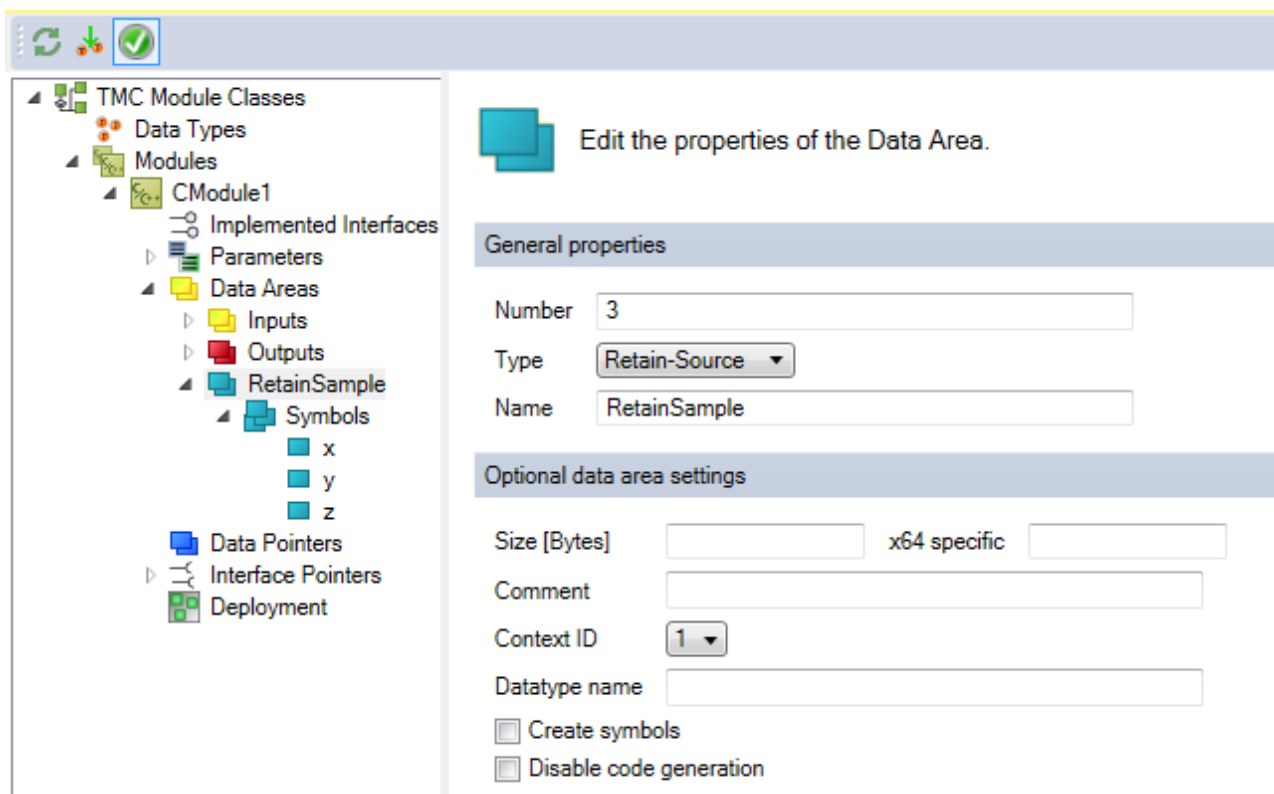


If self-defined data types (DUTs) are used as retain, the data types must be available in the TwinCAT type system. You can either use the option **Convert to Global Type** or you can create structures directly as `STRUCT RETAIN`. However, the Retain Handler then handles all occurrences of the structure.

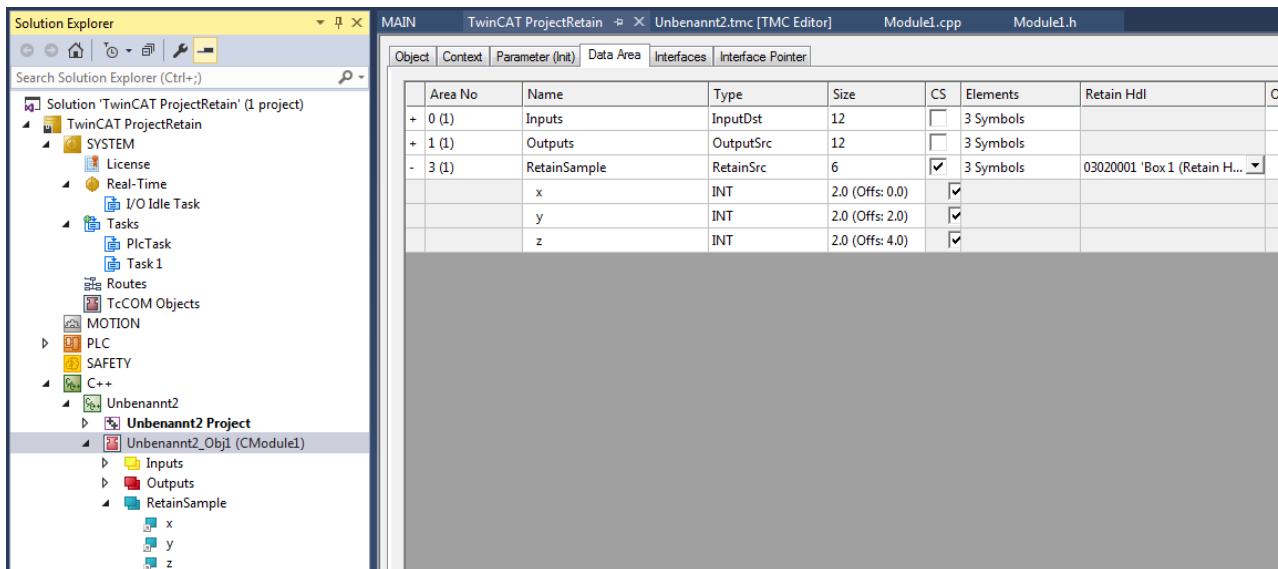
Retain data cannot be used for POU's (function blocks) as a whole. However, individual elements of a POU can be used.

Using the retain handler with a C++ module

In a C++ module a data area of type Retain Source is created, which contains the corresponding symbols.

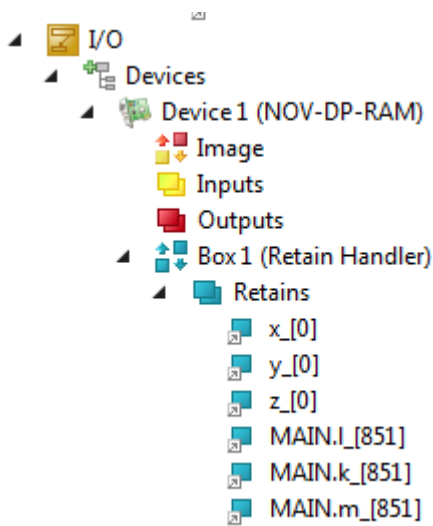


At the instances of the C++ module, a retain handler of the NOV-DP-RAM device to be used for this data area is defined in column **Retain Hdl**.



Conclusions

When a retain handler is selected as target in the respective project, the symbols under retain handler and a mapping are created automatically after a "Build".

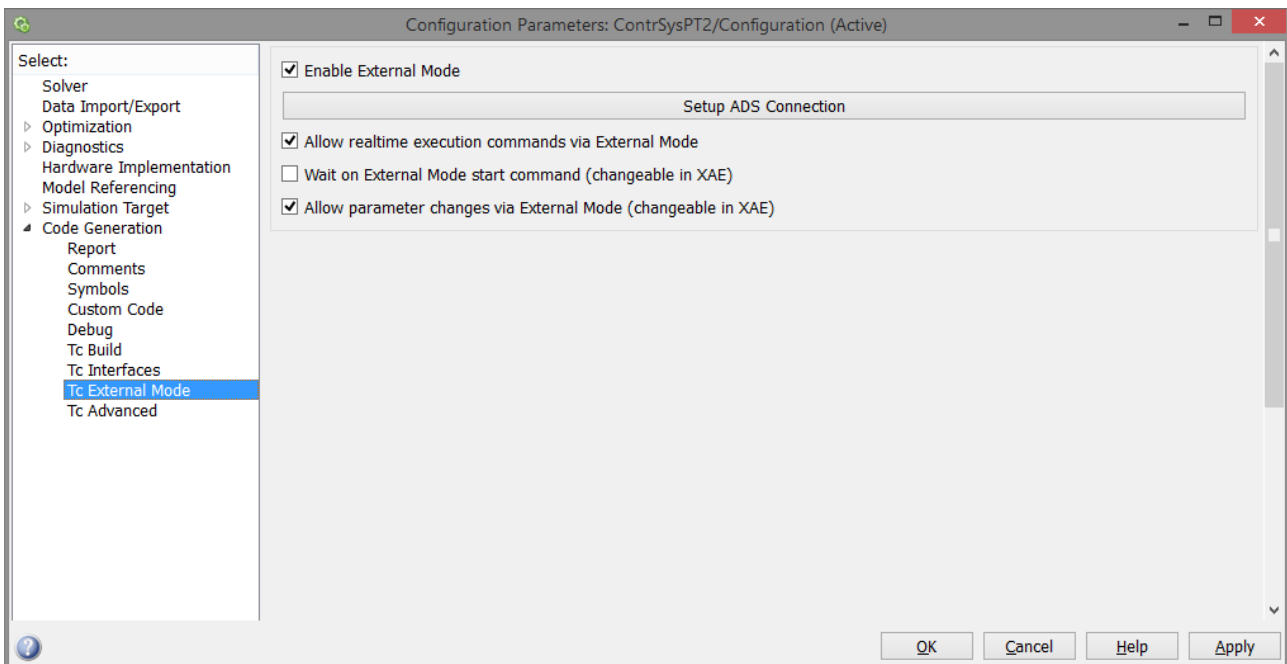


3.5.3 External mode (Tc External Mode)

Simulink offers various execution modes. In addition to "Normal mode", in which the Simulink model is calculated directly in the Simulink environment, an "External mode" is available. In this mode Simulink only acts as a graphical interface, without performing calculations in the background. Once the model with the corresponding settings has been converted into a TcCOM module, Simulink can link to the instantiated TcCOM object that is currently running in the TwinCAT real-time environment. In this case the internal module signals are transferred to Simulink via ADS, where they can be recorded or shown with the corresponding Simulink blocks. Parameters that were modified in Simulink can be written online into the TcCOM object. However, such an online parameter modification is only possible for parameters that are defined as "tunable".

Configuration of the module generator

An External Mode connection is only possible if the generated module supports it. To this end **External Mode** must be activated in the settings for the Simulink Coder under **TC External Mode** before the module is generated:



In addition, there is a button for preconfiguring the "External Mode" connection. For information on configuring the "External Mode" connection see section "Establishing a connection". Further parameters under this tab are:

Parameter	Description	Default value
Allow real-time execution commands via External Mode	Defines the default value of the <u>module parameter</u> [▶ 31] "AllowExecutionCommands", which specifies whether the module should process start and stop commands from Simulink. Special behavior of this parameter: The module parameter "AllowExecutionCommands" is ReadOnly, if the value is FALSE. In this case the code is optimized in terms of the execution time and therefore does not contain the code sections for processing start/stop commands.	FALSE
By default wait on External Mode start command	Default value of the <u>module parameter</u> [▶ 31] "WaitForStartCommand"	FALSE

Module parameter

For configuring the behavior in **external mode** (on the XAE side) the parameter External Mode is defined as a structure in generated modules, which contains the following elements:

Parameter	Description	Default value	
Activated	ReadOnly . Determines whether the generated module supports the external mode.	Setting the module generator	
AllowExecutionCommands	Only relevant if "Activated"=TRUE. ReadOnly if the default value is FALSE, since in this case the code sections are not included in the generated code. That is, this parameter can disable the processing of start and stop commands, but it cannot enable it, if it was not created during code generation.	Setting the module generator	
	TRUE		Enables Simulink to start and stop the module execution via the "External Mode" connection.
	FALSE		Start and stop commands are ignored in the module.

Parameter	Description	Default value	
WaitForStartCommand	Only relevant if "Activated" is TRUE and "AllowExecutionCommands" is TRUE	Setting the module generator	
	TRUE		When TwinCAT has started the module is not executed automatically but waits for the start command from Simulink.
	FALSE		The module starts immediately with the assigned TwinCAT task. (default)

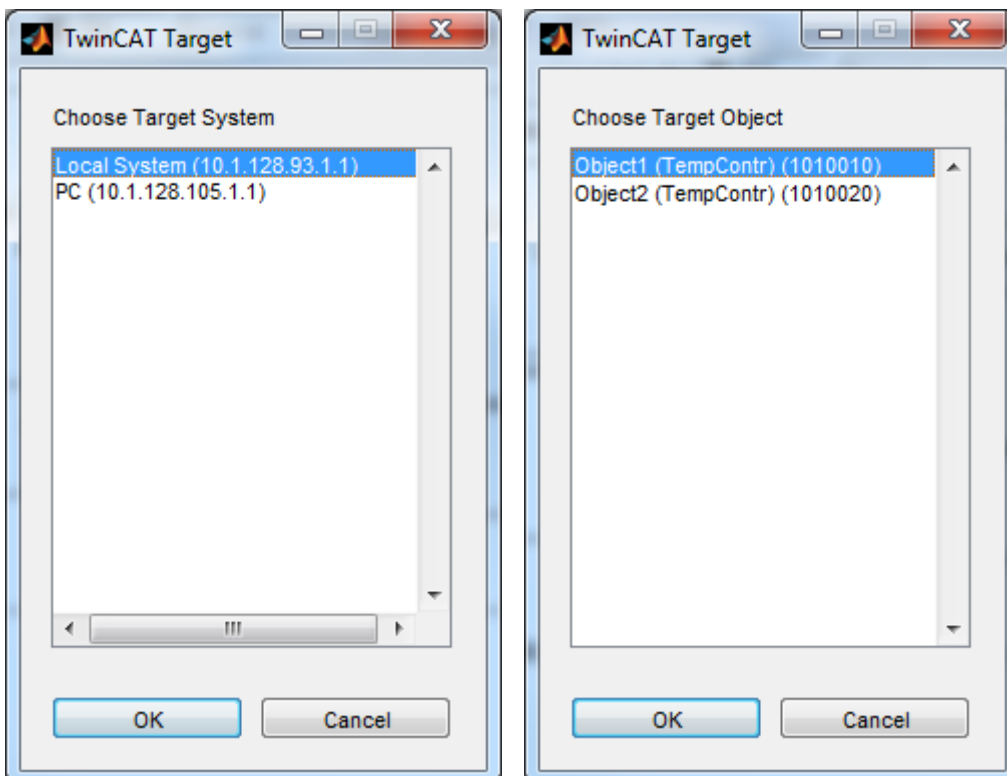
For further information on adapting the module parameters in XAE see section "Parameterization of the generated module".

Establish connection from Simulink®

The "External Mode" connection can be started from Simulink via the **Connect to Target** icon, which appears in the Simulink toolbar when **External** mode is selected:



If connection data are missing or incorrect, the following dialogs are displayed one after the other, so that the user can reconfigure the connection:



The first dialog box shows a list of target systems, the second box shows a list of the available module instances on the selected target system. In the following dialog the user can specify whether the new connection data should be stored. Once the connection data have been stored, the connection is established automatically, if the connection data point to a valid and suitable module.

The stored connection data can be modified at any time in the coder settings via the button **Setup ADS Connection** under **TC External Mode**.

Transfer of the calculation results for "minor time steps"

Under certain circumstances, signal values transferred via ADS may differ from the values that were copied to other process images via "output mapping". See [Transfer of the calculation results for "minor time steps" \[► 58\]](#).

Parameterization in TwinCAT

Large models necessitate communication of large data volumes between TwinCAT and Simulink. This takes place via ADS. On the TwinCAT side, buffers are created as part of the process. The buffers can be adapted for incoming and outgoing data (default: 10,000 bytes), and the timeout threshold can be adjusted (default: 3.0 seconds).

Object	Context	Parameter (Init)	Parameter (Online)	Data Area	Interfaces	Block Diagram					
Name	Value	CS	Unit	Type	PTCID						
CallBy	CyclicTask	<input checked="" type="checkbox"/>		TctModuleCallByTy...	0x00000000						
ExecutionSequence	StateUpdateAfterOutputMappi...	<input checked="" type="checkbox"/>		TctModuleExecutio...	0x00000001						
StepSize	RequireMatchingTaskCycleTime	<input checked="" type="checkbox"/>		TctStepSizeType	0x00000002						
- ExtModeParameters	...	<input checked="" type="checkbox"/>			0xBF002000						
.ConnectionTimeout	3.0			LREAL							
.InitIncomingPktBufferSize	10000			UDINT							
.InitOutgoingPktBufferSize	10000			UDINT							
.Activated	TRUE	<input checked="" type="checkbox"/>		BOOL							

3.5.4 Advanced settings (Tc Advanced)

The advanced settings can be used to set parameters that affect the execution and call behavior of the module and also the display and properties of the exported block diagram:

Configuration Parameters: Simple_NOT/Configuration (Active)

Search

- Solver
- Data Import/Export
- ▶ Optimization
- ▶ Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- ▼ Code Generation
 - Report
 - Comments
 - Symbols
 - Custom Code
 - Tc Build
 - Tc Interfaces
 - Tc External Mode
 - Tc Advanced**

Task assignment: ManualConfig

Default task priority: 5

CallBy: CyclicTask

Execution sequence: StateUpdateAfterOutputMapping

Step size: UseTaskCycleTime

Auto start cyclic execution

Monitor execution times

Export block diagram

Resolve masked Subsystems

Access to Parameter variables, not referenced by any block: Assign to parent block

Access to BlockIO variables, not referenced by any block: Assign to parent block

Access to ContState variables, not referenced by any block: Assign to parent block

Access to DWork variables, not referenced by any block: Hide in block diagram

Export block diagram debug information

Show parameters table in XAE

Use original input and output block names (including special characters).

Set testpoints at Simulink Scope signals before code generation
(Caution This requires a change in the Simulink model. Testpoints are not removed automatically, when deactivating!)

Maximum number of visible array elements: 200

Hide DataTypes defined in TMC: '\$<TwinCat3Dir>\CustomConfig\Modules\\$<CLASSFACTORYNAME>\\$<CLASSFACTC

Skip caller verification

PLC Function Block (TcCom wrapper): Module specific FB

OEMID: <empty>

LicenseIDs: <empty>

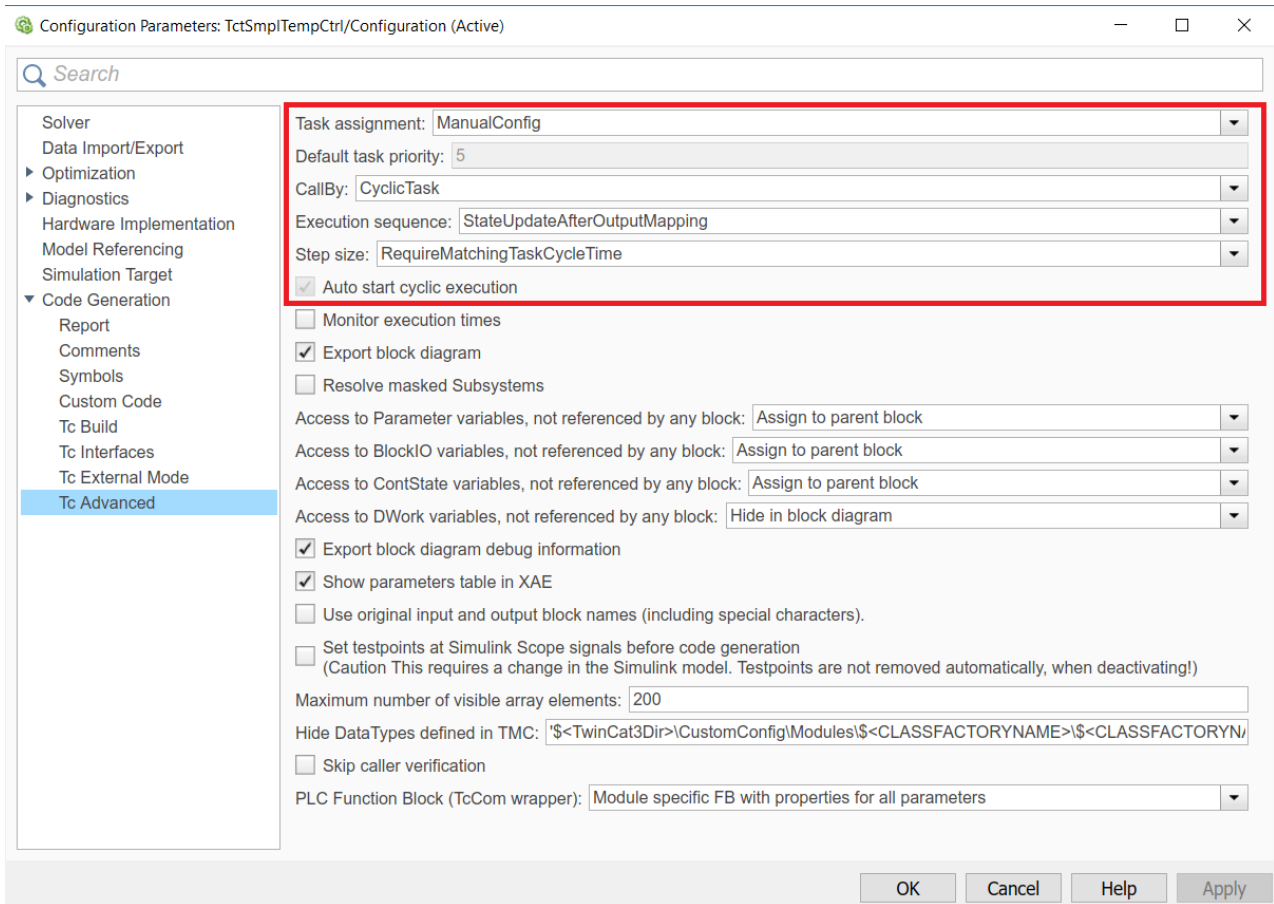
OK Cancel Help Apply

Execution behavior of the generated module

In TwinCAT 3, a Simulink module can be called directly from a cyclical real-time task or from another TwinCAT module, e.g. a PLC. The behavior of the generated module class can be parameterized in Simulink under Tc Advanced. To specify individual module instances behavior that differs from the class behavior, the execution type can be adjusted in the TwinCAT 3 development environment via the TcCOM parameter list in the **Parameter (Init)** tab or via the parameter range of the block diagram.

Configuration of the default settings in Simulink

The default values of the call parameters can be configured in the Simulink coder settings, in order to reduce the parameterization effort for the individual objects (module instances):



Task assignment

The assignment type for a TwinCAT task can be defined under **Task assignment**.

"Depend On" setting	Description
Manual Config	The tasks can be assigned manually in the context table, by selecting or entering the object IDs of the tasks in the Task column. The selected tasks must meet all the criteria that were configured via the "Call parameters"
Parent Object	Can only be used if the parent node of the module instance is a task in the project tree. In this case, the parent object is used as cyclic caller of the module.
Task Properties	The tasks are automatically assigned to the module when the cycle time and the priority correspond to the values specified in Simulink. If there is no corresponding task, new tasks can be created and parameterized as required under the node "System Configuration -> Task Management".

If the Task properties option is active, the priority of the corresponding task must be specified.

Cyclic call

If the value "CyclicTask" was set for the call parameter "CallBy", all task cycle times are verified when the module starts. The conditions for the cycle times of the associated tasks can be specified via the call parameter "Step size". If all cycle times meet their conditions, the module can start. Otherwise the module start and the TwinCAT runtime terminate with corresponding error messages.

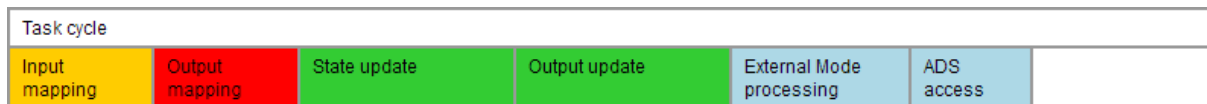
Call from another TwinCAT module

If the call parameter "CallBy" was set to the value "Module", the assigned tasks do not call the module automatically. To call the generated TcCOM module via another module instead, the interfaces of that module can be accessed. This can be done from a C++-module or from a TwinCAT PLC, as shown in "[Calling the generated module from a PLC project \[► 43\]](#)".

Execution order

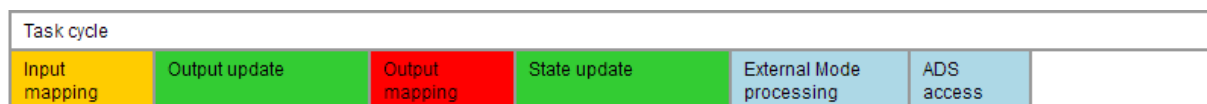
In modules that were created with TE1400 from version 1.1, an execution order can be specified, in order to optimize the jitter and the response times for the respective application. Older versions use always the order "StateUpdateAfterOutputMapping". The following table illustrates the advantages and disadvantages of the supported call sequences:

- IoAtTaskBegin



- Longest response time of all options
- + Smallest possible jitter in response time

- StateUpdateAfterOutputMapping

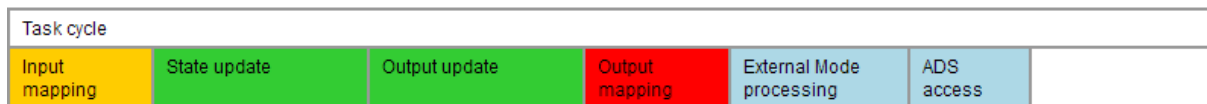


- + Shortest response time
- o Average jitter in response time

Results from "Minor Time Steps" are transferred via ADS

I Signal values transferred via ADS may differ from the values that were copied to other process images via "output mapping". The reason is that "State update" may overwrite values, depending on the selected solver. For further information see [Transfer of the results from "Minor Time Steps" \[► 58\]](#).

- StateUpdateBeforeOutputUpdate



- o Average response time
- largest jitter in the response time, since dependent on execution times for "State update" and "Output update"

Step size adjustment (step size)

The behavior of the TcCOM that was generated with regard to the step size (corresponding to the cycle time in TwinCAT) is defined here.

- RequireMatchingTaskCycleTime
The module expects the "Fixed Step Size" specified in Simulink as cycle time for the allocated task. If another cycle time is set, the TcCOM start sequence is aborted with an error message. Multitasking modules expect that all allocated tasks were configured with the associated Simulink sample time.
- UseTaskCycleTime
The module enables cycle times, which differ from the "Fixed Step Size" specified in Simulink. In multitasking modules, all task cycle times must match the corresponding Simulink sample times. If the cycle time deviates from the Simulink sample time, a warning is issued in TwinCAT indicating the deviation.
- UseModelStepSize
The module uses the sample time set in Simulink for all internal calculations. This setting is primarily intended for use in simulations within the TwinCAT environment and can be used for accelerated or slowed-down simulation.

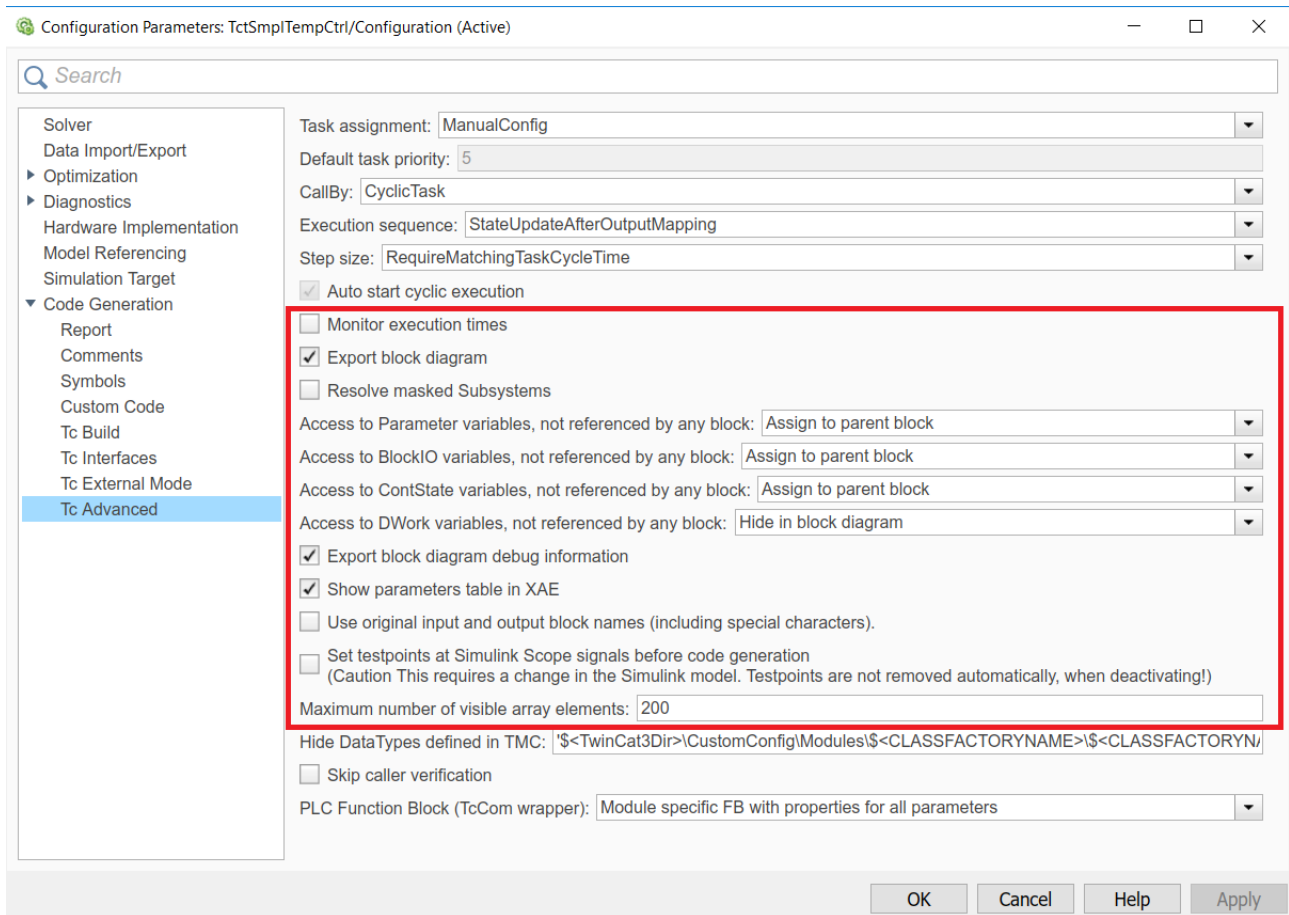
Auto start cyclic execution

If this option is enabled (default), the TcCOM module is set to OP state on startup, and the generated model code is executed directly. If this option is disabled, the module is also set to OP state, but the model code is not executed. In the instantiated module this option can be found in the "Parameter (init)" tab and in the parameter range of the block diagram under "Module parameters" as variable "ExecuteModelCode", where it can also be adjusted.

Adapting the display, debugging and parameterizability

Modules generated from Simulink offer a wide range of options for parameterizing the module and model parameters, even after code generation and instantiation. The parameterization options can be adjusted before the code generation, so that in the development phase debugging options are enabled and parameters of masked subsystems are resolved, which are to be hidden in the release version. The use of modules in TwinCAT 3 requires a display that can be configured according to requirements. For example, debugging information can be included in the block diagram export.

The following coder parameters enable adaptation of the block diagram export, the parameter and signal representation, and advanced functions:



Parameters	Description		Default value
Monitoring execution times	TRUE	The execution times of the TC module are calculated and made available as ADS variable for monitoring.	FALSE
	FALSE	Calculation of execution times disabled.	
Export block diagram	TRUE	The Simulink block diagram is exported and displayed in XAE under its "Block Diagram" tab once the module has been instantiated.	TRUE
	FALSE	The Simulink block diagram is not exported, and the "Block Diagram" tab is not displayed in XAE.	
Resolve masked Subsystems	Only relevant if "Export block diagram"=TRUE.		FALSE

Parameters	Description		Default value
	TRUE	Masked subsystems are dissolved. All contents of masked subsystems are visible to users of the generated module in XAE.	
	FALSE	Masked subsystems are not resolved. The contents of the masked subsystems are not visible to users of the generated module.	
Access to <i>VariableGroup</i> , not referenced by any block	Assign to parent block	Variables of this group, which belong to a block within an unresolved subsystem, are assigned to the next higher, visible subsystem.	
	Hide in block diagram	Variables of this group, which cannot be allocated to a Simulink block, are not displayed in the block diagram.	
	Hide, No access	Variables of this group, which cannot be allocated to a Simulink block, are hidden and made inaccessible. This is only possible, if "No DataArea" was selected for the process image of this variable group (configuration in Simulink [► 26]).	
Export block diagram debug information	TRUE	Debugging information generated for the block diagram enables allocation of row numbers in the generated code to displayed blocks. Required for debugging [► 197] .	TRUE
	FALSE	No debugging information is generated	
Show parameter table in XAE	TRUE	The "Parameter (Init)" tab is displayed in XAE and enables parameterization of the module via the parameter list.	TRUE
	FALSE	The "Parameter (Init)" tab is not displayed in XAE.	
Use original input and output block names	FALSE	Inputs and outputs of the module have the names that were created by the Simulink Coder as variable names. Spaces and special characters are not allowed.	FALSE
	TRUE	Inputs and outputs of the module have the names that were used in the Simulink model. The names may include spaces and special characters.	
Set testpoints at Simulink Scope signals before code generation		Scope blocks are ignored by the Simulink coder, i.e. the signals are generally not available in the generated TwinCAT module and cannot be displayed. To force the generation of variables for these signals, test points can be defined in the Simulink model. This parameter can be used to automatically create test points for all scope input signals.	
Maximum number of visible array elements		Specifies the maximum number of array elements to be displayed in the TwinCAT development environment. Larger arrays cannot be opened, and the elements cannot be linked individually, for example.	

Hide Datatypes defined in TMC

In each TMC file the required data types are specified and notified in the system through import in TwinCAT 3. The data types are assigned a unique GUID. Accordingly, the GUID remains unchanged if a TMC file is re-imported in which a data type has not changed. If enums or structures are used, for example, changes (e.g. additional model parameters in a structure) may result in the data type name of the modified data type and the previous data type being identical, with different GUIDs. This unique assignment via GUIDs is not available in the PLC, where the data type name is used for identification. If a TcCOM instance is called from the PLC, a mechanism must be provided that prevents this kind of ambiguity.

The **Hide Data Types defined in TMC** ensures that the last imported TMC or its data type names and data types are used for the PLC. Any existing data type names with other data types are hidden for the PLC. See also [How do I resolve data type conflicts in the PLC project? \[► 60\]](#).

Skip caller verification

This option disables queries when calling a TcCOM from the PLC, see [Calling the generated module from a PLC project \[► 43\]](#). This leads to faster processing of the module call. On the other hand, the user must make sure that the call is executed correctly and from the correct context.

This option should only be activated if it is necessary for performance reasons, and if the project has previously been tested with activated queries.

PLC Function Block

The POU type for calling a Simulink object from the PLC can be defined in more detail here. A more detailed description can be found under "[Calling the generated module from a PLC project \[► 43\]](#)".

OEM license check

Optionally, a generated TcCOM can be linked to an OEM license. This OEM license is checked when starting TcCOM (besides the Beckhoff runtime license TC1220 or TC1320) in TwinCAT 3. If no valid license is available, the module does not start up and an error message appears.

How to create and manage OEM certificates can be found under TwinCAT3 > TE1000 XAE > Technologies > Security Management.

In Simulink, you can insert the OEM License Check by naming your OEM ID and your license ID or multiple license IDs to be queried. You can find your OEM ID in the Security Management Console (Extended Info activated). The license ID can be viewed by double-clicking on the corresponding license entry in TwinCAT under System > License. Both IDs are also included in the generated License Request File when a Request File is generated with your OEM license.

Note GUID form

The IDs to be entered must be transferred as a string in GUID form, i.e. in Simulink the data must be entered in quotation marks. No spaces are allowed in the specifications.

Sample entry:

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

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

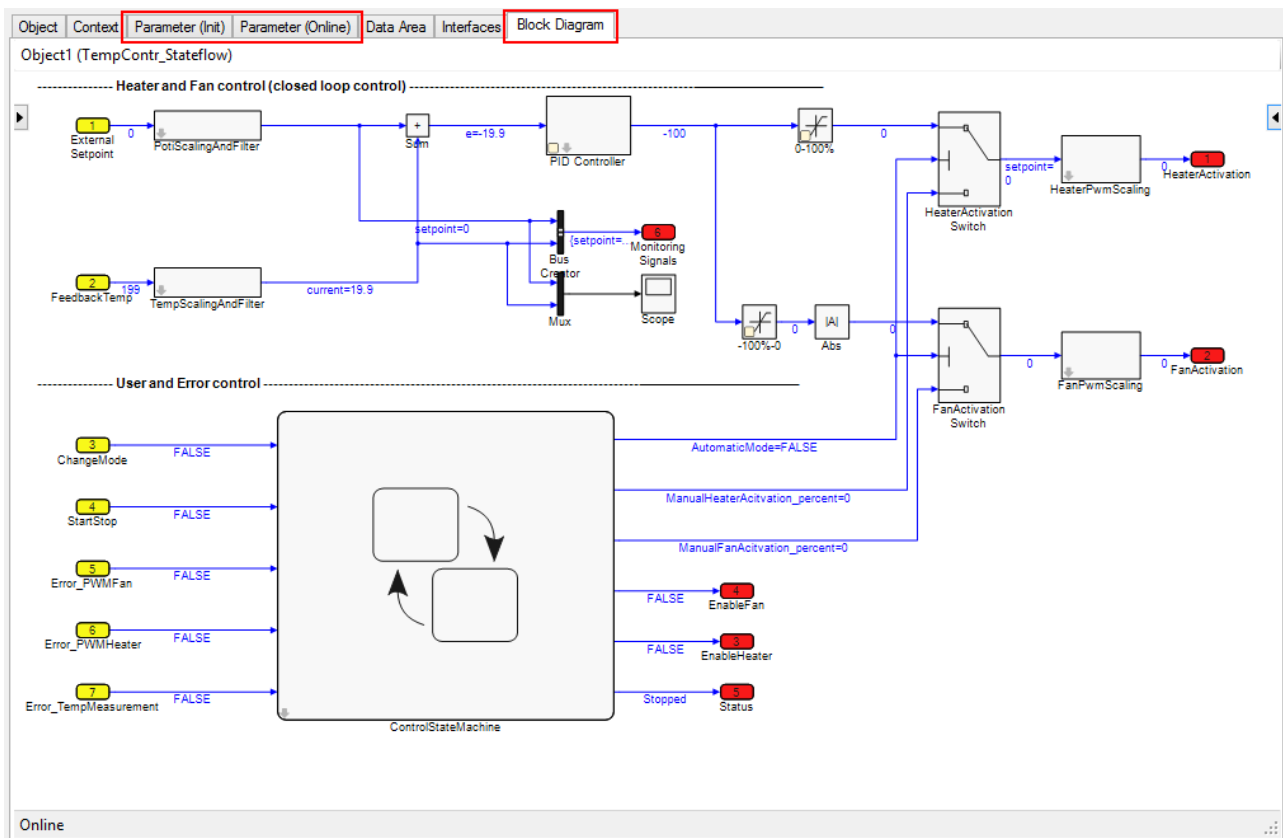
3.6 Application of modules in TwinCAT

The data of modules exported from Simulink are stored in directory `%TwinCat3Dir\CustomConfig\Modules\<MODULENAME>\%`, and from where they can be copied to any number of development PCs with TwinCAT XAE. A Simulink license is not required on these systems. TwinCAT nevertheless offers further extensive parameterization options for the generated modules. Cyclic execution of TcCOM modules through calls via a task and calls of modules from a PLC project are described below.

3.6.1 Parameterization of a module instance

Parameter representation in XAE

The block diagram in the Browser Parameters tab:



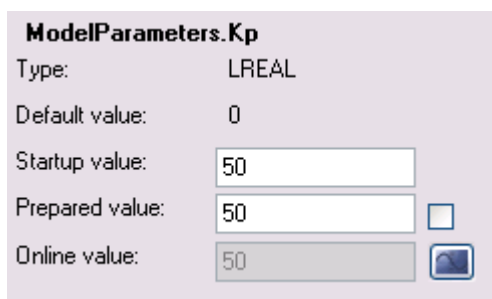
In general, TcCOM modules can be parameterized via the parameter list under the **Parameter (Init)** tab in the TwinCAT 3 development environment (XAE). Simulink modules can also be parameterized via the block diagram, if block diagram export is enabled in the Simulink coder settings under **Tc Advanced**.

Module- and model-specific parameters

The parameter list contains module- and model-specific parameters. Examples of module-specific parameters are "[Call Parameter \[▶ 411\]](#)" or "[External Mode Parameter \[▶ 30\]](#)". In the block diagram these parameters are only shown in the parameter section (on the right-hand side of the window) if the top level of the block diagram ("`<root>`") is selected.

Model-specific parameters are defined as "tunable" parameters in the Simulink blocks. The parameter list displays them as a structure.

In the block diagram these model parameters are assigned to a block or indeed several blocks. The values can be adjusted when the corresponding block is selected. The parameter values (startup, prepared or online) can then be adjusted in the drop-down menu of the property table or in the parameter window directly in the block diagram:



Hover with the mouse over the name of the drop-down menu (in this case ModelParameters.Kp) to show its ADS information as a tooltip. Right-click on the name to copy the ADS symbol information to the clipboard.

Access to the model-specific parameters is only possible, if

- the Simulink optimization option **Inline parameters** is disabled, or workspace variables were selected as model parameters in the advanced options under **Inline parameters**

- ADS access to parameters is enabled under **Tc interfaces** ([Data exchange \(Tc Interfaces\)](#) [▶ 26]).

"Default", "Startup", "Online" and "Prepared"

The following value types can be found in the drop-down menu of the Property table of the block diagram:

- **Default values** are the parameter values during code generation. They are invariably stored in the module description file and enable the manufacturing settings to be restored after parameter changes.
- **Startup values** are stored in the TwinCAT project file and downloaded to the module instance as soon as TwinCAT starts the module instance.
Startup values for the input process image can also be specified in Simulink® modules. This allows the module to be started with non-zero input values, without the need for linking the inputs with other process images. Internal signals and output signals have no starting values, since they would, in any case, be overwritten in the first cycle.
- **Online values** are only available if the module was started on the target system. They show the current parameter value in the running module. This value can also be changed during runtime. Although in this case the corresponding input field has to be enabled via the context menu, in order to prevent accidental inputs.
- **Prepared values** can be specified whenever online values are available. They can be used to save various values, in order to write them consistently to the module. If prepared values have been specified, they are displayed in a table below the block diagram. The buttons to the right of the list can be used to download prepared values as online values and/or save them as starting value, or delete them.

3.6.2 Executing the generated module under TwinCAT

In TwinCAT 3, a TcCOM module can be called directly from a cyclical real-time task or from another module, e.g. a PLC. To specify the behavior of the individual module instances, the method of execution can be defined in the TwinCAT 3 development environment.

Context settings

A list of all Simulink sample times for the module can be found under the **Context** tab of the module instance. If "SingleTasking" is selected in the solver settings of the Simulink model, the number of tasks is limited to 1:

ID	Task	Name	Priority	Cycle Time (µs)	ADS Port
0	02000105	Task 1	5	5000	350

For each of the contexts listed in the table a task has to be specified, through which the module is to be called. The task assignment varies, depending on the settings under "Depend On":

"Depend On" setting	Description
Manual Config	The tasks can be assigned manually in the context table, by selecting or entering the object IDs of the tasks in the Task column. The selected tasks must meet all the criteria that were configured via the "Call parameters".
Parent Object	Can only be used if the parent node of the module instance is a task in the project tree. In this case, the parent object is used as cyclic caller of the module.
Task Properties	The tasks are automatically assigned to the module when the cycle time and the priority correspond to the values specified in Simulink. If there is no corresponding task, new tasks can be created and parameterized as required under the node "System Configuration -> Task Management".

Configuration in XAE

Parameters that affect the behavior of Simulink module execution are:

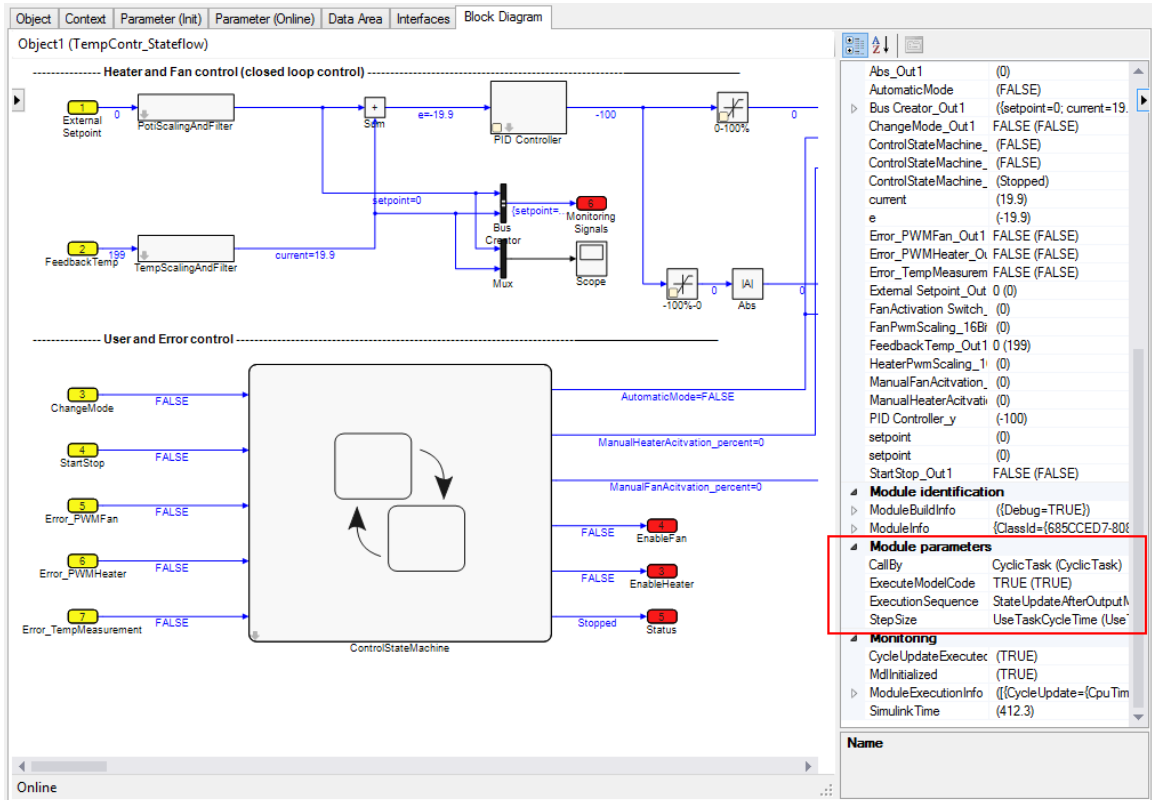
Parameter	Options / description	
CallBy	Task	The module automatically appends itself to the tasks specified in the context settings [► 41] , when TwinCAT is switched to Run mode. The tasks call the module cyclically until TwinCAT is stopped.
	Modules	The module is not called directly by the assigned tasks, but it can be called from the PLC or another module. Important: The calling module must be assigned the same task as the TcCOM objects to be called.
Step size	RequireMatchingTaskCycleTime	The module expects the "Fixed Step Size" specified in Simulink as cycle time for the allocated task. Multitasking modules expect that all allocated tasks were configured with the associated Simulink sample time. Otherwise the module (and TwinCAT) cannot be started. The start sequence is then aborted with corresponding error messages.
	UseTaskCycleTime	The module enables cycle times, which differ from the "Fixed Step Size" specified in Simulink. In multitasking modules, all task cycle times must match the corresponding Simulink sample times.
	UseModelStepSize	The module uses the SampleTime set in Simulink for all internal calculations. This setting is primarily intended for use in simulations within the TwinCAT environment.
ExecutionSequence	This parameter is only available in modules that were generated with TE1400 Version 1.1 or higher. It can be used to adjust the order of the calculation and communication process, in order to optimize jitter and reaction time for the respective application. Modules generated with TE1400 version 1.0 always use the order " StateUpdateAfterOutputMapping ". The differences between the different options are described under " order of execution [► 36] ".	
	IOAtTaskbeginn	Execution order: Input mapping -> Output mapping -> State update -> Output update -> External mode processing -> ADS access
	StateupdateAfterOutputMapping	Execution order: Input mapping -> Output update -> Output mapping -> State update -> External mode processing -> ADS access
	StateupdateBeforeOutputUpdate	Execution order: Input mapping -> State update -> Output update -> Output mapping -> External mode processing -> ADS access

Access to these parameter in the TwinCAT development environment (XAE) is provided via the object node under the following tabs:

- **Parameter (Init) :**

Object	Context	Parameter (Init)	Parameter (Online)	Data Area	Interfaces	Block Diagram
PTCID	Name	Value	Unit	Type	Comment	
0x00000000	CallBy	CyclicTask		✓		TctModuleCallByType
0x00000001	ExecutionSequence	StateUpdateAfterOutputMapping		✓		TctModuleExecution...
0x00000002	StepSize	UseTaskCycleTime		✓		TctStepSizeType
0xBF002000	ExtModeParameters	...				
0x00000009	ExecuteModelCode	TRUE		✓		BOOL

• Block diagram :



If none of these tabs are displayed, the Simulink coder settings need to be adjusted for parameter representation in XAE.

3.6.3 Calling the generated module from a PLC project

If the call parameter "CallBy" was set to the value "Module", the assigned tasks do not call the module automatically. To call the generated TcCom module via another module instead, the interfaces of that module can be accessed. This can be done from a C++ module or, as shown below, from the TwinCAT PLC. A PLCopen XML file is generated during code generation. This file can be found in the build directory <MODEL_DIRECTORY>\<MODEL_NAME>_tct and - if the module was exported via the Publish step - also in the Publish directory [▶ 22] of the module. The file contains POUs that simplify calling a Simulink object from the PLC by encapsulating the handling of the interface pointers. The POUs can be imported via **Import PLCopenXML** in the context menu of a PLC project node.



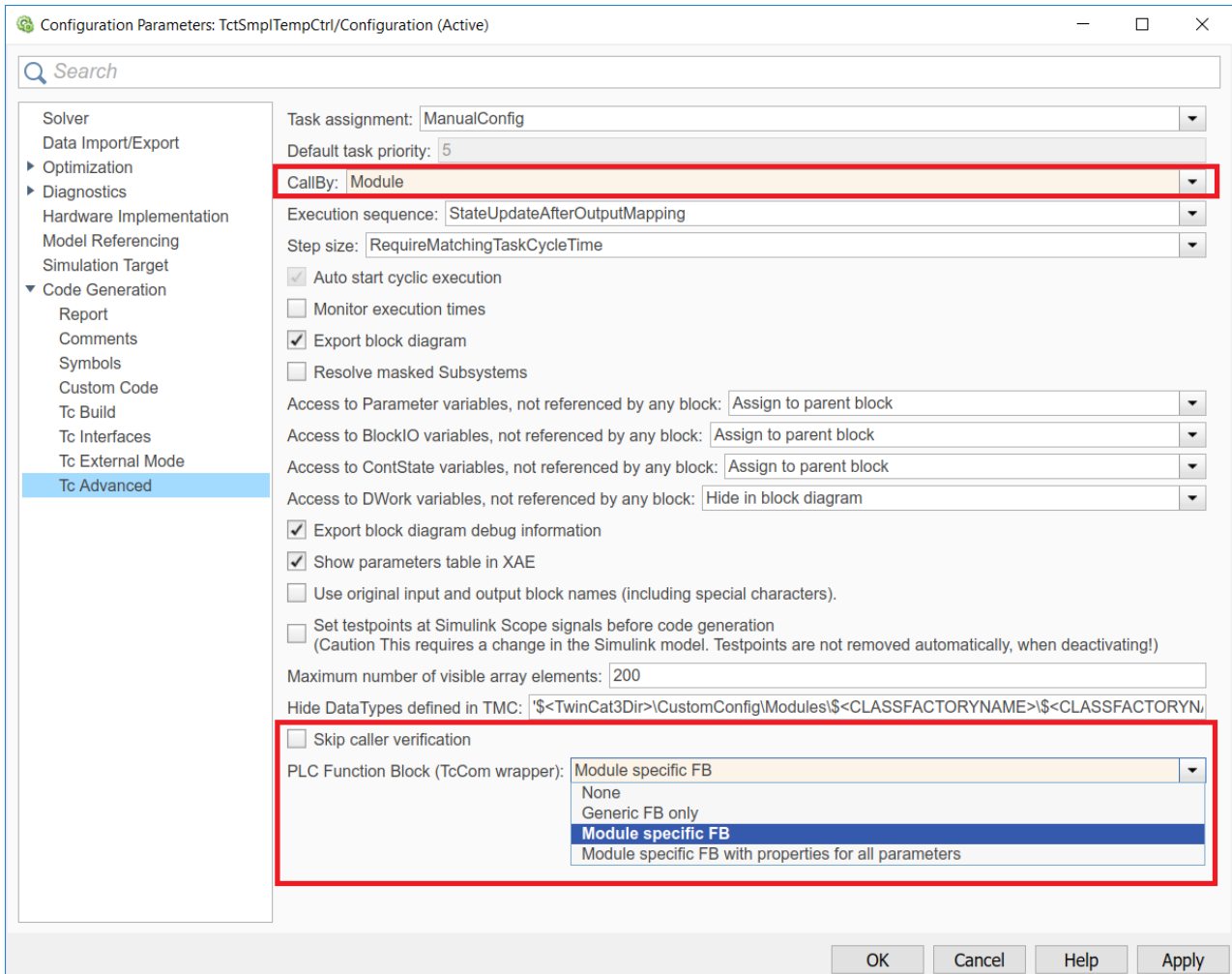
The following descriptions apply from version 1.2.1216.0 of TE1400!

Configuration in Simulink

In the settings under **Tc Advanced**, **CallBy** is initially set to **Module** (can be changed later in TwinCAT Engineering).

The parameter **Skip caller verification** is visible from TE1400 version 1.2.1230.

The parameter **PLC Function Block (TcCOM wrapper)** is available from TE1400 version 1.2.1216.0:



The following options are available for selection:

Option	Description
None	No PLCopen XML file is generated
Generic FB only	Only the function block FB_TcMatSimObject (see also here), which applies to all modules generated with TE1400 and data types used by it, is generated. The data exchange takes place via generic methods. The user must know module-specific data such as byte-sizes, parameter index offsets or DataArea IDs. Version note: Up to and including TE1400 1.2.1216.0, this generic FB can only be used in a meaningful way, if an FB derived from it is implemented, which deals with the initialization of internal variables. From version 1.2.1217.0 additional methods are available, which enable direct initialization, i.e. without derived FB.
Module specific FB	In addition to the generic FB_TcMatSimObject , the module-specific function block FB_<ModuleName> and associated data types are generated. The structure of the input and output variables exactly matches the structure of the corresponding data areas of the module. For exchanging data, the input and output variables can be assigned directly, without having to explicitly specify the size of the data areas or the DataArea IDs, for example.
Module specific FB with properties for all parameters	The module-specific function block FB_<ModuleName> is assigned additional properties. Based on these properties, module parameters can be read and also written, if appropriate. For each module parameter the function block is assigned two properties: " <i>ParameterName_Startup</i> " and " <i>ParameterName_Online</i> ".

The module-specific function block

FB_<ModuleName> is derived from FB_TcMatSimObject and provides the methods and properties described above. In addition, the following properties are implemented:

Public Properties:

Method	Data type	Description
InitData	ST_<ModuleName>_InitData	Stores the startup values of the module parameters for initializing a module instance. During module state transitions the values are transferred to the module from INIT to PREOP via SetObjState(). Required for dynamic instantiation.
<ParameterName>_Startup	<ParameterType>	Available for all parameters, if the coder is configured accordingly. Enables transparent access to the corresponding element of the InitData structure (read/write).
<ParameterName>_Online	HRESULT	Available for all parameters, if the coder is configured accordingly. Reads or writes the online values of the corresponding module parameter.

Notes regarding FB with properties for all parameters

If **Process image** is set to **Internal DataArea** under Tc Interfaces in the **Parameter access** area, a property is created for all parameters. These must then be read and written as an entity:

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

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

If **Process image** is set to **No DataArea** under Tc Interfaces in the **Parameter access** area, a separate property is created for each model parameter. These can then be read and written directly without a local PLC variable.

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

Referencing a static module instance:

The FB can be used to access module instances previously created in the XAE, e.g. under **System > TcCOM Objects**. For this static case, the object ID of the corresponding module instance must be transferred during declaration of the FB instance:

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

The object ID can be found in the instance of TcCOM under the **Object** tab.

Sample code

The following code sample illustrates the application of a module-specific function block in a simple PLC program in ST code, using an object of module class "TempContr" with ObjectID 0x01010010 as an example:

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

```

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

```

Generating and referencing a dynamic module instance:

If <ObjectId> = 0, the FB attempts to generate an instance of the TcCom module dynamically:

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

In this case, the module instance does not appear in the XAE configuration tree, but only appears at runtime (i.e. after the initialization of the PLC instance) in the "Project Objects" table of the **System > TcCOM Objects** node.

A prerequisite for dynamic instantiation of a TcCOMmodule if that the corresponding "Class Factory" is loaded. To this end, the **Load** checkbox (or the **TC Loader** checkbox if the "TwinCAT Loader" is used) must be set for the "Class Factory" of the module under the **Class Factories** tab of the **System > TcCOM Objects** node. The name of the "Class Factory" of a TcCOM module generated from Simulink usually matches the module name, although the ClassFactory name is limited to fewer characters.

A further condition for dynamic instantiation of a module is adequate availability of dynamic memory. To this end, the ADS router memory must be set to an adequate size.

Sample code:

```

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

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

```

Task context setting

The PLC task from which the call is made must be configured in the context settings [▶ 41] of a **static** module instance.

Result:

ID	Task	Name
0	02000114	PlcTask

The object ID of the PLC task must be transferred to a **dynamic** module instance via the InitData structure. If available, the corresponding InitData element can be set via the property "ContextInfoArr_<contextId>_TaskOid_Startup".

When the TcCOM module is called, a context verification is performed. An error message is displayed if the context is not correct. This verification takes time and is performed with each call. For this reason, the verification can be deactivated via the checkbox **Skip caller verification** in the **Tc Advanced** dialog, see Skip caller verification [▶ 39].

Import of several PLCopen-XML: FB_TcMatSimObject

The generic function block **FB_TcMatSimObject** is identical for **all** modules generated with TE1400 (from V1.2.12016.0). **Even if it is used for different modules, it only has to be imported once into the PLC project.**

Description of the function block:

Public Methods

Method	Return data type	Description
Execute	HRESULT	Copies the data of the InputDataArea structures from the FB to the module instance (of the object), calls the cyclic methods of the object and copies the data from the output data areas back into the corresponding data structures of the FB
GetObjPara	HRESULT	Reads parameter values from the object via PID (Parameter ID = ADS Index Offset)
SetObjPara	HRESULT	Writes parameter values to the object via PID (Parameter ID = ADS Index Offset)
ResetHresult		Acknowledges error codes that have occurred during initialization of the FB or when calling "Execute()".
SaveOnlineParametersForInit	HRESULT	Reads the current online values of the parameters from the object and saves them in the parameter structure after the FB variable plnitData, if it exists
SetObjState	HRESULT	Tries to bring the TCOM_STATE of the object to the required target state, step-by-step
AssignClassId		From TE1400 1.2.1217.0: Sets the expected class ID for the case of referencing a static object . This is compared with the class ID of the referenced module to avoid problems due to incompatibilities. If no class ID is assigned, this compatibility check is omitted. To generate a dynamic object , the class ID must be defined via this method.
SetInitDataInfo		From TE1400 1.2.1217.0: Transfers the pointer to the InitData structure to be used. This structure must be created when dynamic objects are used. It must be initialized with parameters, as required. For static objects, this structure is optional. It

Method	Return data type	Description
		enables subsequent re-initialization of the object. In this case, the structure may also be initialized by calling "SaveOnlineParametersForInit".
SetDataAreaInfo		From TE1400 1.2.1217.0: Transfers the pointer to an array of DataArea information of type ST_TcMatSimObjectDataAreaInfo, and the number of elements in that array. This information is required for cyclic data exchange within the "Execute" method.

Public Properties

Method	Data type	Description
ClassId	CLSID	Returns the ClassId of the module
Error	BOOL	Returns TRUE if a pending error requires acknowledgement
ErrorCode	HRESULT	Returns the current error code
State	TCOM_STATE	Returns the current TCOM_STATE of the object, or tries to bring it into the target state, step-by-step

Referencing a module instance

Just like the module-specific FB derived from it, FB_TcMatSimObject can be instantiated with the object ID of a static module instance or with 0:

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

3.6.4 Using the ToFile block

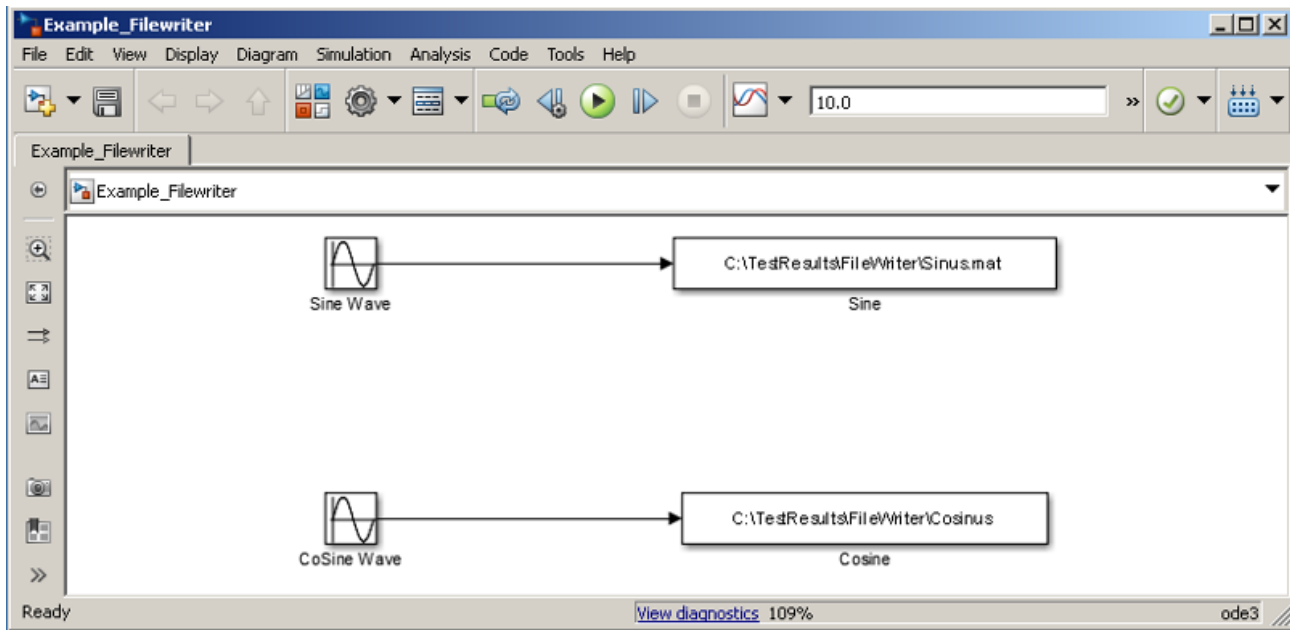
The ToFile block from the Simulink default library can be used to log signals in a MAT file. From within a created TcCOM, this block can still be used from the TwinCAT runtime.

For file system access from the real-time, an additional TcCOM "TcExtendedFilewriter" is created and linked to the TcCOM with the ToFile block (referred to as Simulink TcCOM below). The TcExtendedFilewriter then receives the data from the assigned TcCOM and writes it to a MAT file (mat4).

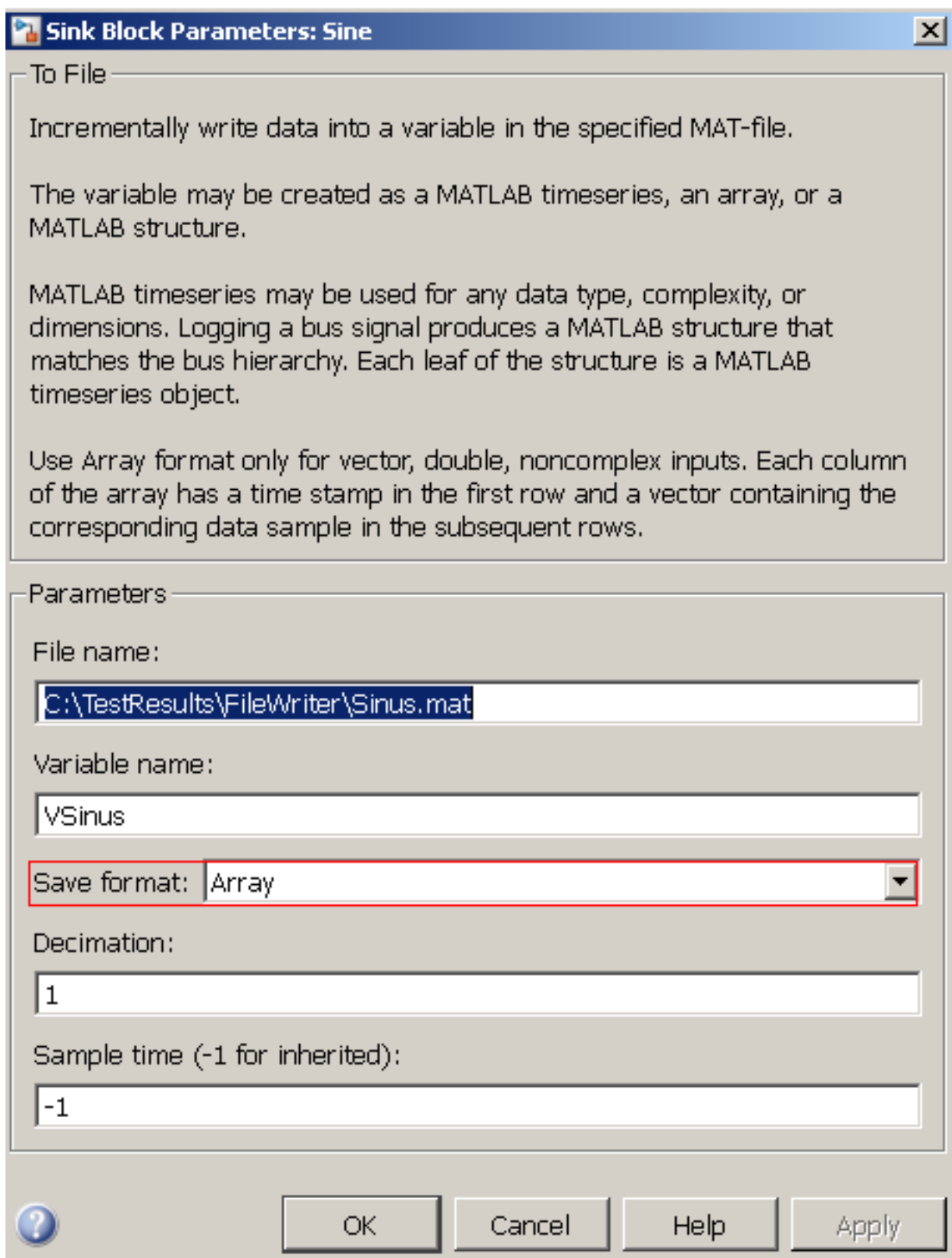
The settings in Simulink and TwinCAT are described step by step below, based on an example.

Configuration in the Simulink model

A model with a sine and a cosine source serves as a simple example. Each signal is to be logged with a ToFile block.



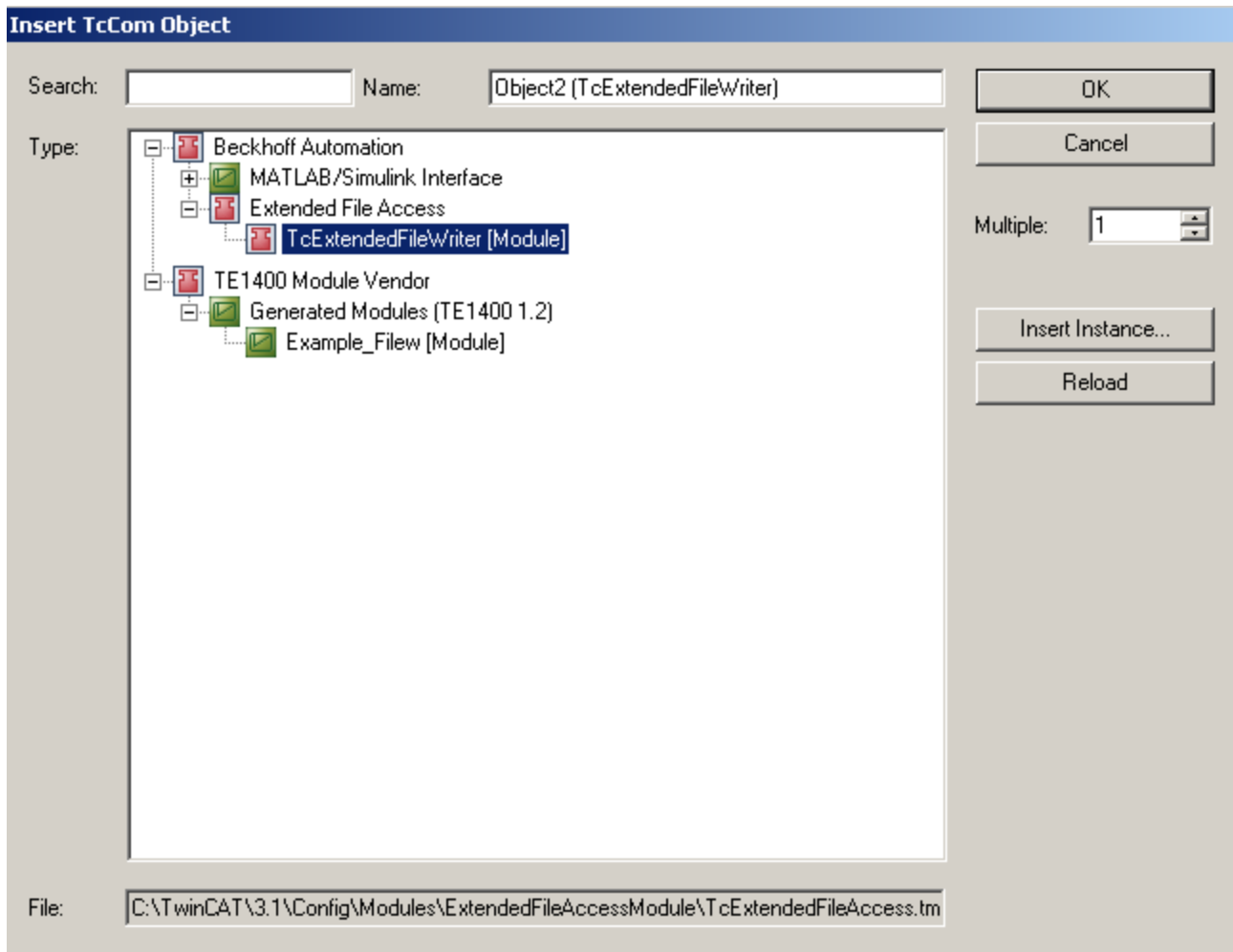
To enable code generation for the ToFile blocks, the format must be set to **Array**:



The model is now ready for code generation.

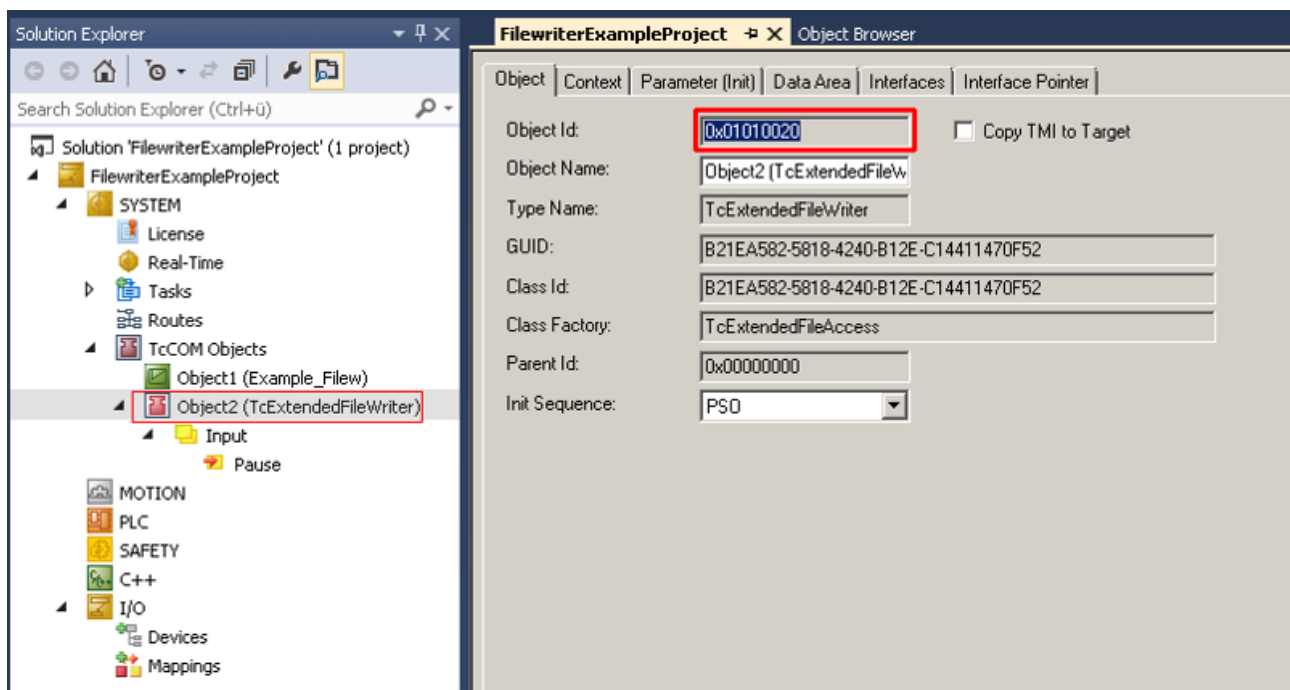
Configuration in TwinCAT

To write from the generated Simulink TcCOM, the TcCOM *TcExtendedFilewriter* installed with the TE1400 is required. It accepts data from the Simulink object and stores the data in the file system. The module can be found in the TcCOM browser under *Beckhoff Automation -> Extended File Access -> TcExtendedFileWriter*.



Initially, both TcCOMs are instantiated. Both objects can be linked to a joint task or separate task. In order to establish a link between the two objects, the ObjectID of the *TcExtendedFileWriter* instance is communicated to the Simulink TcCOM.

The ObjectID can be found under the **Object** tab.



The ObjectID is then inserted under the "Parameters (Init)" tab of the Simulink TcCOM for the parameter **ExtendedFileAccessOID**:

Object	Context	Parameter (Init)	Parameter (Online)	Data Area	Interfaces	Block Diagram
	PTCID	Name	Value	CS	Unit	Type
	0x0000...	CallBy	CyclicTask	<input checked="" type="checkbox"/>		TctModule..
	0x0000...	ExecutionSequence	StateUpdateAfterOutputM...	<input checked="" type="checkbox"/>		TctModule..
	0x0000...	StepSize	UseTaskCycleTime	<input checked="" type="checkbox"/>		TctStepSize.
	0x0000...	ExtendedFileAccessOID	00000000	<input type="checkbox"/>		OTCID
+	0xBF00...	ExtModeParameters	00000000	<input type="checkbox"/>		
	0x0000...	ExecuteModelCode	01010020 'Object2 (TcExtendedFileWriter)'	<input type="checkbox"/>		BOOL
+	0x8200...	ModelParameters	...	<input type="checkbox"/>		
	0x0300...	ContextInfoArr_0_TaskOid	00000000	<input checked="" type="checkbox"/>		OTCID
	0x0300...	ContextInfoArr_0_TaskPriority	0	<input checked="" type="checkbox"/>		UDINT
	0x0300...	ContextInfoArr_0_TaskCycleTimeNs	0	<input checked="" type="checkbox"/>		UDINT
	0x0300...	ContextInfoArr_0_TaskPort	0	<input checked="" type="checkbox"/>		UINT
	0x0300...	ContextInfoArr_0_TaskSortOrder	0	<input checked="" type="checkbox"/>		UDINT

It is possible to link several Simulink TcCOMs with one *TcExtendedFileWriter* instance. Ensure that filename conflicts are avoided. Several *TcExtendedFileWriter* instances can be used in parallel. For example, each Simulink TcCOM with a ToFile block can use its own *TcExtendedFileWriter* instance.

Parameterization of the TcExtendedFileWriter instance

The behavior of the object can be adapted under the Parameter (init) tab of the *TcExtendedFileWriter* instance.

Object	Context	Parameter (Init)	Data Area	Interfaces	Interface Pointer		
	Name	Value	CS	Unit	Type	P...	Comment
	Timeout	1000	<input type="checkbox"/>	ms	UDINT	0x...	Timeout used for a file access
	WorkingDirectory	C:\	<input type="checkbox"/>		STRING(...)	0x...	Anchor directory for relative Filepaths
	NumberOfFiles	0	<input type="checkbox"/>	Files	UDINT	0x...	Limit number of files. 0 - unlimited, 1..n - 1..n Files
	MaxFileSize	1024	<input type="checkbox"/>	KByte	UDINT	0x...	Size of one file until a new one is opened. filesize = 0 >= Max...
	InternalBufferSize	12	<input type="checkbox"/>	KByte	UDINT	0x...	The Data is stored in a Buffer before being transferred to syst...
	SegmentSize	2	<input type="checkbox"/>	KByte	UDINT	0x...	Size of the Segments transferred to system service
	.. Input	...	<input type="checkbox"/>			0x...	

Timeout:

A timeout can be set

Working directory:

If a relative path is used in the ToFile block, e.g. /logData, the full path is resolved via the Working Directory parameter.

Number of Files:

It is possible to limit the number of files. If the parameter is 0, limitation is inactive.

Max File Size:

Once the specified file size (default: 1 MB) has been reached, the file is closed and a new file is opened, in order to ensure that the logged data can be accessed while the module is running.

Internal Buffer Size:

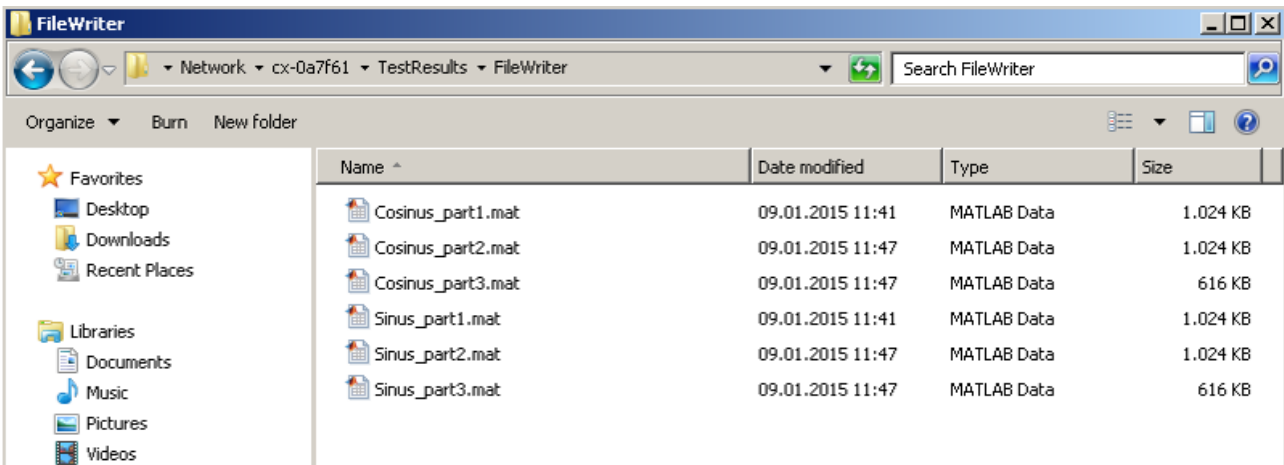
A buffer with the size InternalBufferSize is created on the TwinCAT side, from which the data is then written.

Segment Size:

With each write command of the *TcExtendedFileWriter* instance, a segment with the size SegmentSize is written from the internal buffer to the specified file. The maximum theoretically possible data rate for writing is composed of the SegmentSize and the cycle time of the TcExtendedFileWriter (the *TcExtendedFileWriter* instance does not have to have the same cycle time as the assigned Simulink TcCOM module). However, be aware that a write command may not yet be complete when the next cycle starts. If this is the case, the write command is suspended in this cycle. It is therefore a *best case* assessment.

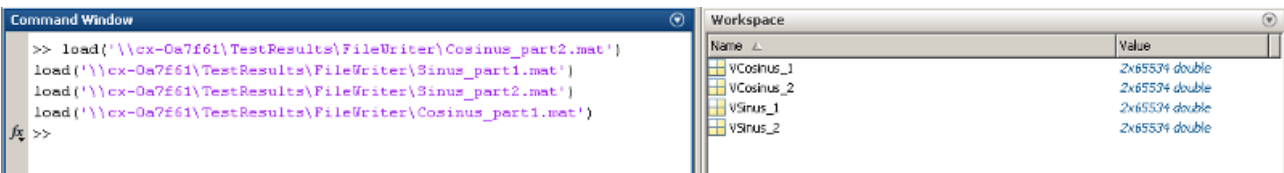
The TwinCAT project can now be activated.

Once the specified file size (default: 1 MB) has been reached, the file is closed and a new file is opened, in order to ensure that the logged data can be accessed while the module is running (in the diagram: *_part1.mat and *_part2.mat are completed, while writing of *_part3.mat is still in progress):

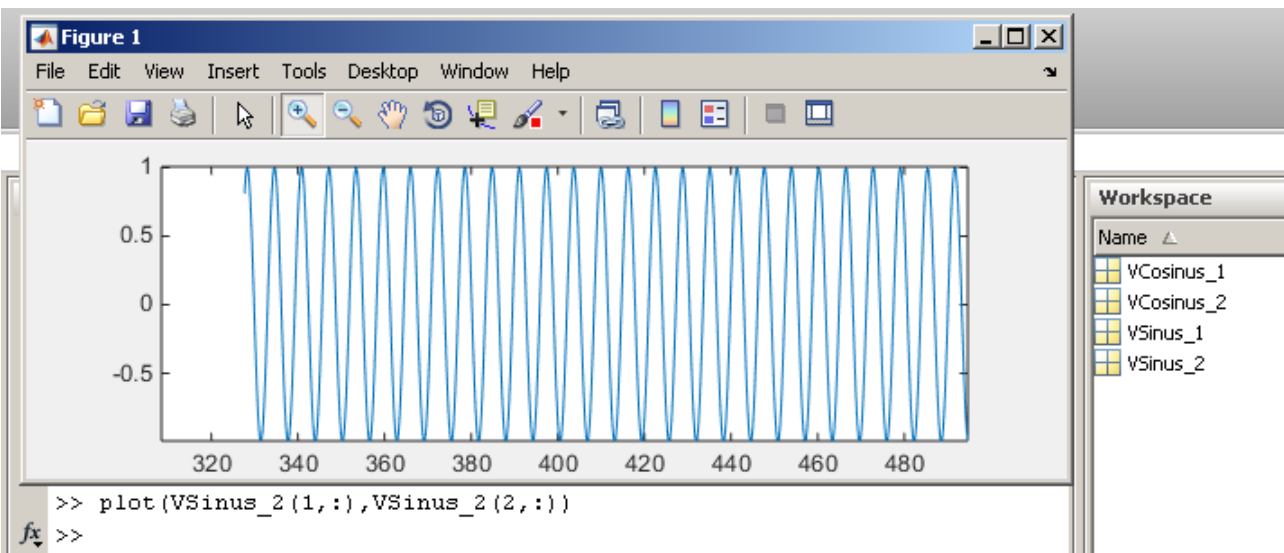


The TcExtendedFilewriter object has a Pause input to prevent continuous writing. If the input is set to TRUE, the file currently in use for write access is closed, and all further incoming data is discarded. If the input is set to FALSE again, a new file for logging incoming data is opened.

The closed files can be opened as usual in MATLAB:

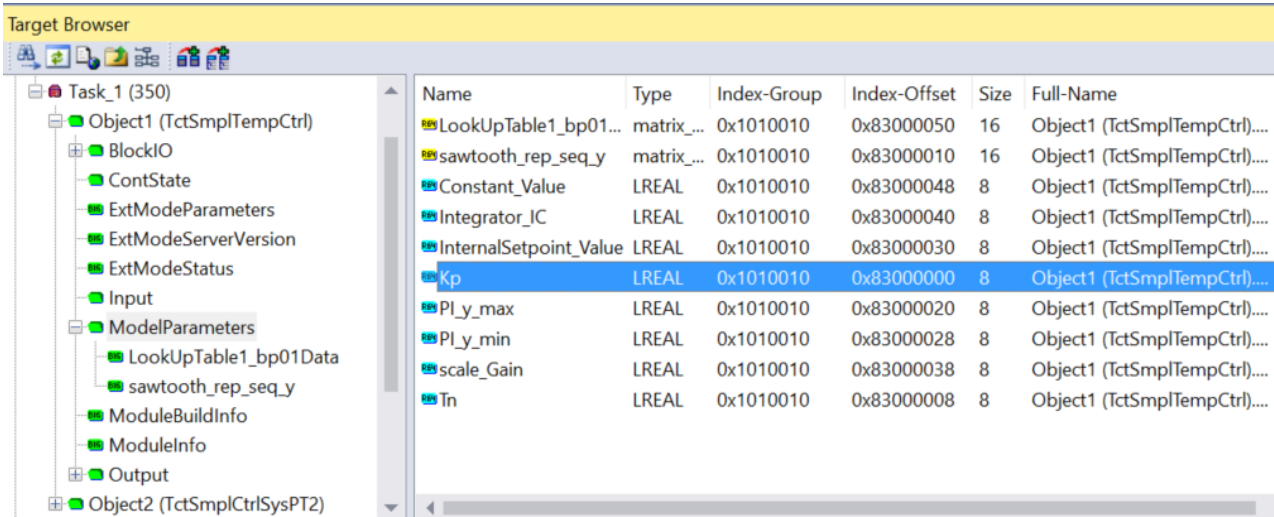


The plot shows the expected sine wave:

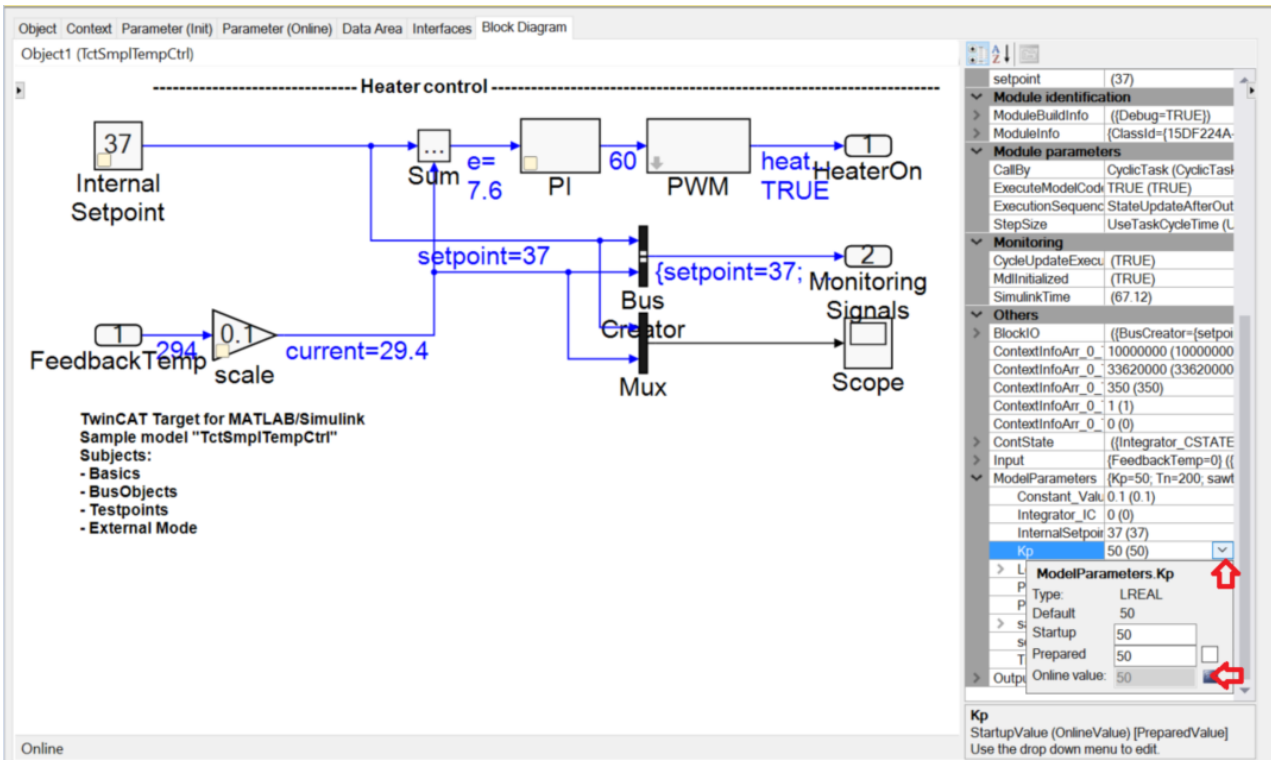


3.6.5 Signal access via TwinCAT 3 Scope

TwinCAT 3 Scope enables access to all variable groups for which at least read ADS access was enabled, see [Tc Interfaces \[▶ 26\]](#) in Simulink. To use the **Target Browser** for configuring the Scope, the option ... **_CreateSymbols** must be selected under Tc Interfaces in Simulink. Without the corresponding symbol information, the signals to be captured have to be configured manually in Scope via Index group and Index offset.



Alternatively, Scope can be started via the corresponding icon directly from the TwinCAT development environment (XAE). In the drop-down window of the **block diagram browser**, the button **Show in Scope** is shown for each available signal when the module instance runs on the target system.



The signals can also be conveniently dragged into the Scope configuration using the right mouse button (drag & drop) from the drop-down menu (bar on the right in the diagram above) or from the blue signal lines in the block diagram (main window in the diagram above). The right mouse button can also be used to drag a whole block into the Scope configuration, in order to record all inputs and outputs for this function block.

3.7 FAQ

3.7.1 Does code generation work even if I integrate S-Functions into my model?

S-Functions can be integrated into Simulink® models, which can then be built for use in the TwinCAT runtime.

There are various workflows, which are based on different circumstances. The most common four cases are briefly explained here, and the appropriate solution for integration into the code generation process is shown.

Case 1: I have access to the source code used in the S-Function.

In this case, the location of the source code can be specified in the S-Function. The code generation process can be started directly without any further steps. The source code is found and compiled for use in TwinCAT.

Case 2: I have an inlined S-Function (TLC file)

In this case, the code generation process can be started directly without any further steps, since the code of the S-Function to be inserted is contained in the TLC file. How to create a TLC file for an S-function can be found in the documentation of The MathWorks®: <https://de.mathworks.com/help/simulink/sfg/how-to-implement-s-functions.html>

Case 3: I have a compiled MEX file without access to the source code

In this case, a function was created by third parties and compiled as a MEX file. The source code or the TLC file was not included, e.g. to protect intellectual property. In this case, the third party supplying the MEX file must compile the source code as a TwinCAT-capable library, so that this library can be linked in real-time. A guide can be found under Samples: [SFunStaticLib \[▶ 77\]](#).

Case 4: I integrate a MEX library, whose source code I do not have, into my S-Function (whose source code is available).

In this case, too, the third party supplying the MEX file must compile the source code as a TwinCAT-compatible library. A guide can be found under Samples: [SFunWrappedStaticLib \[▶ 83\]](#).

3.7.2 Why do FPU/SSE exceptions occur at runtime in the generated TwinCAT module, but not in the Simulink model?

In the default settings, Simulink may treat floating point exceptions differently than TwinCAT 3.

In order to adjust the behavior for floating point exceptions between Simulink and TwinCAT, in the Signals box under Model Configuration Parameters in Simulink in section **Diagnostics >Data Validity** you can choose between:

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

To **debug** an SSE exception in TwinCAT, please use the C++ debugger, see [Debug \[▶ 197\]](#) in the TE1400 documentation. Provided you have built your model as a "debug" module with the C++ debugger activated, it is sufficient to attach to the process after TwinCAT has started, if the exception occurs during the initial cycles. In many cases the SSE exception occurs directly in the first cycle. In this case, a division by zero can occur quickly if certain signals are initialized with zero.

Another way to encounter SSE exceptions is to **disable** floating point exceptions. These can be deactivated in the System Manager under Tasks (uncheck the *floating point exceptions* checkbox). This setting then applies to all modules that are addressed by this task. If an exception occurs, a NaN is generated and no error is output.

NOTICE**Deactivating floating point exceptions**

NaN values may only be used in other PLC libraries, in particular as control values in functions for Motion Control and for drive control, if they are expressly approved! Otherwise, NaN values can lead to potentially dangerous malfunctions!

3.7.3 After updating TwinCAT and/or TE1400 I get an error message for an existing model.

Description of the situation:

You have already successfully converted a Simulink model into a TcCOM. You have then carried out an update of the TwinCAT XAE and/or the TE1400. You now want to recompile the Simulink model (e.g. you have used a new TE1400 feature, changed something on the model, or you have not changed anything). Now you receive error messages during publishing.

Possible cause and solution:

A folder named <modelname>_tct already exists in the Build directory, see [Which files are created automatically during code generation and publishing? \[► 59\]](#). This order was created with the sources of the previous software version(s). Under certain circumstances, conflicts may arise at this point if a new software release triggers a new publishing process that wants to store the sources in the same folder.

A simple solution is to delete the corresponding folder, so that all sources are reconfigured with the current version of all components when you build the module.

3.7.4 Why do the parameters of the TcCOM instance not always change after a "Reload TMC/TMI" operation?

Observation:

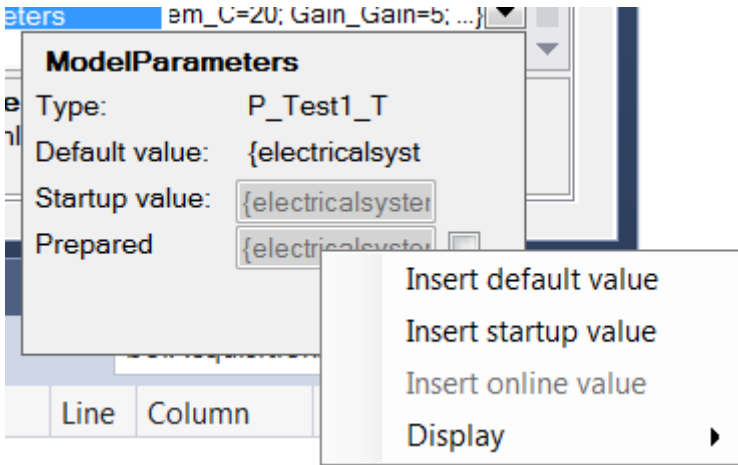
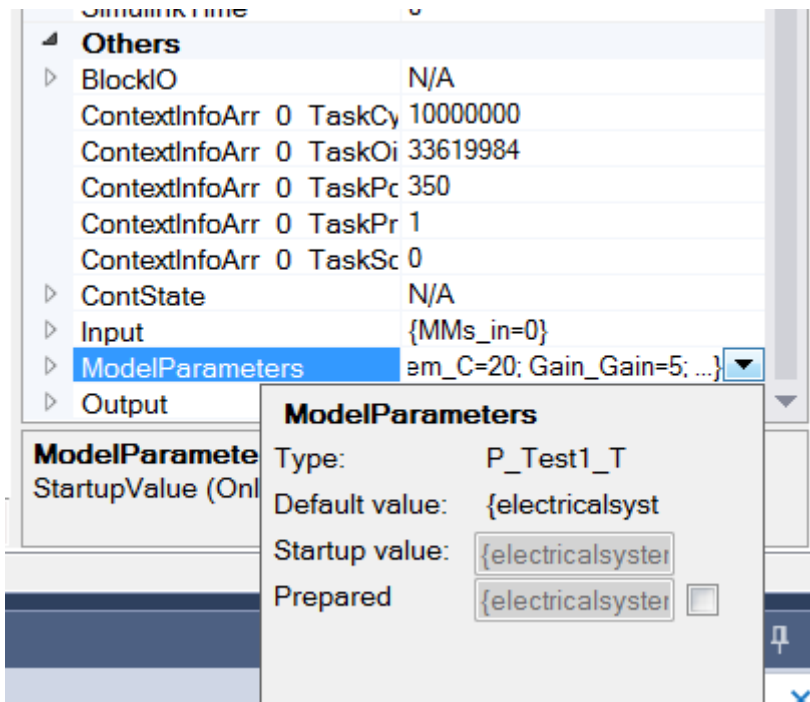
TwinCAT 3 contains an existing instance of a TcCOM object.

As already explained, the model parameters, e.g. the parameters of a PID controller, can be modified in TwinCAT via the exported block diagram or via the **Parameter (init)** tab of the TcCOM object outside of Simulink. If you change your Simulink model in Simulink and create a new TcCOM object, you can, of course, update this via the call **reload TMC/TMI** by right-clicking on the corresponding TcCOM object in TwinCAT. In this case all links are preserved, as long as the process image remains unchanged.

A distinction is made between two different cases

- Only model parameters were modified in Simulink, e.g. PID control parameters
- Model parameters were modified, and further structural changes were made in the model

In the former case you will note that the parameters of your TcCOM object have **not** changed after the call **Reload TMC/TMI**. The startup values are taken from the previous TcCOM instance, so that your settings from TwinCAT for this module instance are not lost. To load the model parameters from Simulink, you can select them by navigating to the **ModelParameters** dropdown menu in the right part of the block diagram window: right-click on **Startup value** or **Prepared** and select **Insert default value**. The default values are loaded from the TMC file, so that the parameter settings are taken from Simulink.



Alternatively, you can delete the old TcCOM object and insert the new TcCOM object. In this case all previous model parameters are lost, and the newly added object has the same model parameters as the corresponding Simulink model.

If additional changes were made apart from the model parameters, the model code also changes, which means retention of the previous model parameters settings is only possible to a limited degree. In this case the TwinCAT module parameters from the previous instance are retained, and the System Manager is still able to assign them unambiguously.

3.7.5 After a "Reload TMC/TMI" error "Source File <path> to deploy to target not found

When you perform a TMC/TMI reload, make sure you use the TMC file from the Publish directory: %TwinCAT3Dir%\CustomConfig\Modules\<MODULENAME>, **not** the file from the Build directory in folder <MODULENAME>_tct.

If you use the TMC file from the Build directory, TwinCAT cannot find the corresponding driver and you get the error message shown in the heading when you start TwinCAT.

3.7.6 Why do I have a ClassID conflict when I start TwinCAT?

The class ID establishes a unique relationship between the tmc file and the associated real-time driver.

If you have created a TcCOM module from Simulink® with the TE1400 and have instantiated it in a TwinCAT project, the class ID is anchored in the TcCOM instance and the instance expects a corresponding driver with this class ID. Now go back to Simulink® and create a new TcCOM with the same name as the already instantiated module. A new tmc file and new drivers will be stored in the Publish directory with a new class ID. If you now activate the TwinCAT configuration without informing TwinCAT that the class ID has changed, you will see the following behavior:

Behavior for TwinCAT version < 4018:

You will get an error message informing you that the class IDs do not match.

Behavior for TwinCAT version ≥ 4018

The driver from the _ModullInstall project folder, which matches the existing instance in the TwinCAT project, is used. The behavior of the module instance remains unchanged for the TwinCAT project.

Important: The lowest compatible TwinCAT build ≥ 4018 must also be entered under Tc Build in order for the latter behavior to occur. See also [Module generation \(Tc Build\)](#) [► 22].

Solution:

To be able to use the behavior of the newly generated TcCOM module in your TwinCAT project, you can right-click on the corresponding instance of TcCOM and select TMI/TMC-File -> **Reload TMI/TMC File**. Now select the tmc file in your Publish directory and confirm with **OK**. If you call the module from the PLC and have imported the PLCopen.xml file for this purpose, you must reimport it and select **Replace the existing object** in the dialog box.

3.7.7 Why can the values transferred via ADS differ from values transferred via output mapping?

Transfer of the results for "minor time steps"

Depending on the configured [processing sequence](#) [► 41] of the module instance, the transferred ADS values may differ from the expected values. Differences may occur if the time-continuous state variables are updated after the "output mapping", in order to obtain the shortest response time:

Task cycle time						
Input mapping	Output update	Output mapping	State update	External mode processing	ADS access	

Signal values transferred via ADS may differ from the values that were copied to other process images via "output mapping". The reason is that some values are overwritten in a state update. In other words: The transferred values are the result of the calculations within subordinate time steps of the solver that was used ("minor time steps"), while during "output mapping" the results of higher-level time steps are copied. This also applies for data that are transferred via [External Mode](#) [► 30].

3.7.8 Are there limitations with regard to executing modules in real-time?

Not all access operations that are possible in Simulink® under non-real-time conditions can be performed in the TwinCAT real-time environment. Known limitations are described below.

- **Direct file access:** No direct access to the file system of the IPC can be realized from the TwinCAT runtime. An exception is the Simulink® sink function block "To File". As described under [Using the ToFile block](#) [► 48], the TcExtendedFileWriter module that realizes the file access can be instantiated in TwinCAT.

- **Direct hardware access:** Direct access to devices/interfaces requires a corresponding driver, e.g. RS232, USB, network card, ... It is not possible to access the device drivers of the operating system from the real-time context. At present it is therefore not easily possible to establish an RS232 communication for non-real-time operation with the Instrument Controller Toolbox™ and then use this directly in the TwinCAT runtime. However, TwinCAT offers a wide range of communication options for linking external devices, see [TwinCAT 3 Connectivity TF6xxx](#).
- **Access to the operating system API:** The API of the operating system cannot be used directly from the TwinCAT runtime. An example is the integration of *windows.h* in C/C++ code. This is integrated by the Simulink Coder® if the FFTW implementation of the FFT block from the DSP Systems Toolbox™ is used (but not with the Radix 2 implementation), for example.

3.7.9 Which files are created automatically during code generation and publishing?

Files are created in two separate folders as soon as you start the build process from Simulink. Which files are created depends on the selected configuration.

Publish directory: %TwinCATDir%\CostumConfig\Modules\

All the files required for instantiation of the TcCOM in TwinCAT are stored in this directory.

File	Purpose
<ModelName>.tmc	TwinCAT module class file
<ModelName>_ModuleInfo.xml	Block diagram information and summary of the engineering system versions (Matlab version, TC version, ...)
<ModelName>_PlcOpenPOUs.xml	Optional file. Can be included for the call of TcCOM from the PLC, see Calling the generated module from a PLC project [► 43] .
<ModelName>.sys	In the subdirectories TwinCAT RT (x64) and TwinCAT RT (x86). Real-time driver of the created module.
<ModelName>.pdb	In all subdirectories. Debug information file.
<ModelName>.dll	In the subdirectories TwinCAT UM (x64) and TwinCAT UM (x86). Driver for the user-mode runtime.

To use the TcCOM described in this directory on other engineering systems, the entire folder can be copied to the appropriate folder on the engineering system.

Build directory

The Build directory is usually the current matlab path, which is active at the start of the build process. Two subdirectories are created in the Build directory. On the one hand, the Simulink Coder creates the directory slprj, in which Simulink stores specific cache files, on the other hand the TE1400 creates a directory <ModelName>_tct, in which all the important resources are combined.

File	Purpose
Subfolder html	Summary of code generation and publishing process in html format.
<ModelName>_codegen_rpt.html	
*.cpp and *.h	Source code of automatic code generation
<ModelName>.vcxproj	Visual Studio project of automatic code generation. Can be included in the TwinCAT C++ node as an <i>existing project</i> and published from there.
<ModelName>_PublishLog.txt	Text file with Publish log.
<ModelName>_ModuleInfo.xml	Block diagram information and summary of the engineering system versions (Matlab version, TC version, ...)

File	Purpose
<ModelName>_PlcOpenPOUs.xml	Optional file. Can be included for the call of TcCOM from the PLC, see Calling the generated module from a PLC project [► 43].

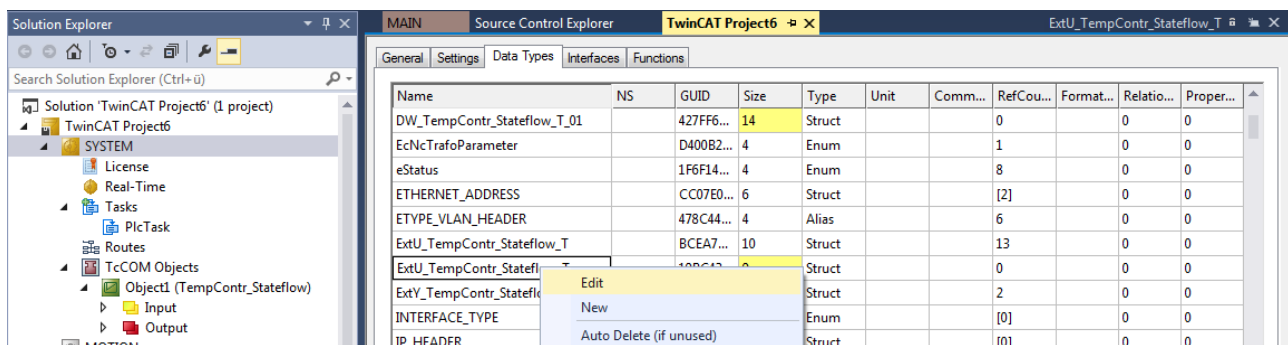
The files stored in the Build directory are suitable for transfer to other engineering systems, just like the files in the Publish directory. On the corresponding engineering systems, the publish process must then be performed manually via the C++ area in TwinCAT. In addition to the resources for the publish process, all other relevant data for tracing the origin of the generated source code (without Matlab or Simulink source code) can be found here.

3.7.10 How do I resolve data type conflicts in the PLC project?

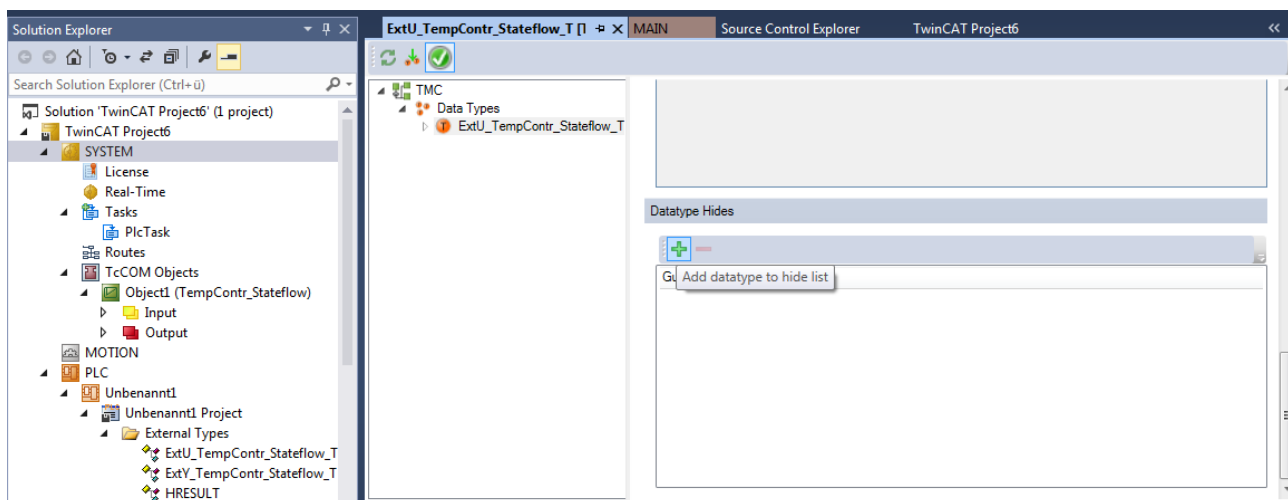
If inputs, outputs, parameters or state variables of a Simulink model are changed, the corresponding data types in the TwinCAT module generated from it also change. After the update, the data types have the same name but a different GUID. The type system of the TwinCAT development environment (XAE) can manage several data types of the same name with different GUID. However, a PLC project is not allowed to have several data types with the same name.

Especially after a module instance has been updated via "Reload TMC", several data types of the same name may exist in the type system, of which typically only the type related to the currently instantiated module class should be used. In some cases the user has to specify manually which of the data types should be available in the PLC project, particularly if PLC function blocks generated by the TE1400 are used.

To this end, the data type editor can be started via the context menu of the type to be used in the table **SYSTEM > Data types**:



By adding **Datatype Hides**, you can selectively exclude obsolete data types from being used in PLC projects:



3.7.11 Why are the parameters of the transfer function block in the TwinCAT display not identical to the display in Simulink?

The Simulink Coder® generates real-time capable code; all transfer function representations are transformed into the state-space representation. Accordingly, the matrices of the state-space representation (A, B, C, D) are used in the code generated by the Simulink Coder®, which in turn can be displayed and modified in TwinCAT 3.

In MATLAB the transformation of the transfer function representations into the state-space representation can take place via the function `[A,B,C,D] = tf2ss(NUM,DEN)`, for example.

3.7.12 Why does my code generation/publish process take so long?

The entire process of generating instantiable TcCOM modules runs through two phases. code generation and the publish process. The *diagnostic viewer* of Simulink® shows:

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

Notes on the duration of the code generation

The duration of the code generation depends to a large extent on the individual model and is made up of the code generation of the Simulink Coder and the code generation for the TcCOM framework. Accordingly, the TE1400 only has influence on the TcCOM framework.

If **large parameter lists**, e.g. look-up tables, are marked as tunable, the look-up table is entered in the tmc file to be generated, which may result in extended code generation duration.

Notes on the duration of the publish process

The Publish process consists of compiling the C/C++ code with the MS Visual C++ compiler, linking, and copying the module files to the Publish directory (<TwinCAT folder>\3.1\CustomConfig\Modules). Accordingly, the compiler performance is crucial for this step. It depends on the compiler version and the settings (e.g. debug or release).

In Simulink® under **Tc Build** it is possible to compile binaries for different **target systems**. These are created in a successive process. If you want to build a large model, it is advisable to focus on the platform(s) that you will actually use later.

3.8 Examples

Example models for generating TcCom modules:

Example	Topics	Description
TemperatureController_minimal ▶ 621	<ul style="list-style-type: none"> Basic principles 	A very simple temperature controller that covers the basics.

Example	Topics	Description
TemperatureController [► 68]	<ul style="list-style-type: none"> Parameter access Using bus objects Using test points Using referenced models Using external mode Generating TwinCAT modules from subsystems 	A very simple temperature controller with PWM output. Provides a quick overview of how to use the module generator. Also uses Simulink BusObjects (structures) for an output and includes a test point, which affects the accessibility of internal signals via ADS. ExternalMode is also used in the example.
SFunStaticLib [► 77]	<ul style="list-style-type: none"> SFunction Static library 	Generates TwinCAT modules from Simulink models with SFunctions that are provided by third parties without source code.
SFunWrappedStaticLib [► 83]	<ul style="list-style-type: none"> SFunction Static library 	Generates TwinCAT modules from Simulink models with SFunctions, for which the source code is available, but is dependent on static libraries.

Examples for module generation callbacks [► 24]:

Example	Topics	Description
Packaging module files into ZIP archives [► 88]	<ul style="list-style-type: none"> PostPublish callback Archiving generated module files 	This simple example illustrates the automatic archiving of generated module files.

3.8.1 TemperatureController_minimal

Description

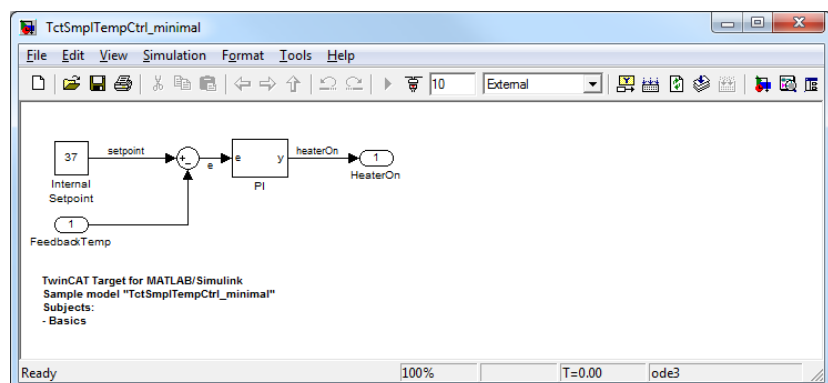
The following example shows the basics of generating a TwinCAT module from a Simulink model.

Overview of project directory

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/1539966475/.zip contains all the files required for reproducing this example:

TctSmpIMinTempCtrl.mdl

Simulink much of a simple PI temperature controller.



TctSmpITempCtrlParameters.mat Contains all the necessary model parameters.

TctSmpIMinCtrlSysPT2.mdl Simulink model of a simple PT2 controlled system (not used in the following description)

_PrecompiledTcComModules This subdirectory contains readily compiled TwinCAT modules that were generated from the enclosed Simulink models. They enable the integration of a module in TwinCAT to be tested, without having to

generate the module first. They can be used in situations where a MATLAB license is not yet available, for example. A quick reference guide for module installation on the development PC is also enclosed.

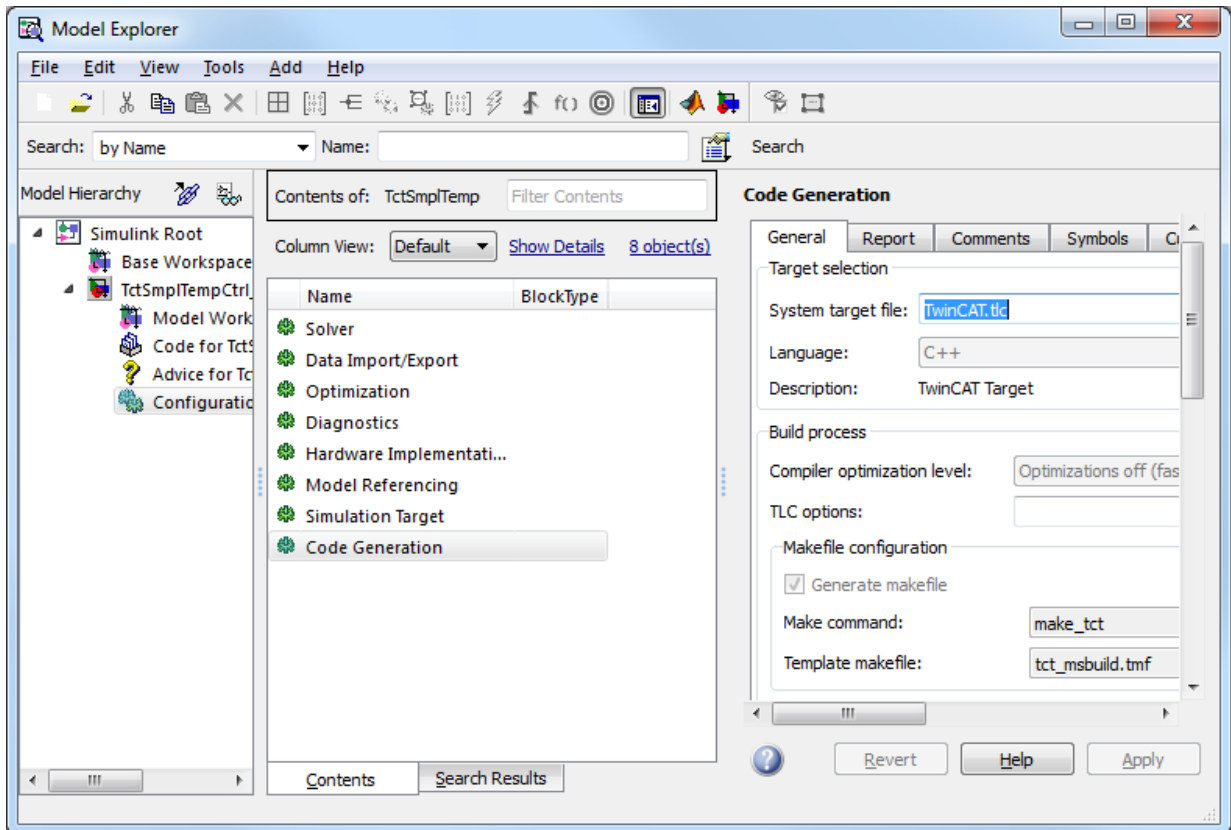
Info: To start the module on an x64 target system, the system must be switched to test mode!

_PreviousSimulinkVersions

The MDL files described above are stored in the file format of the current Simulink version. This subdirectory contains the models in the file format of elder Simulink versions.

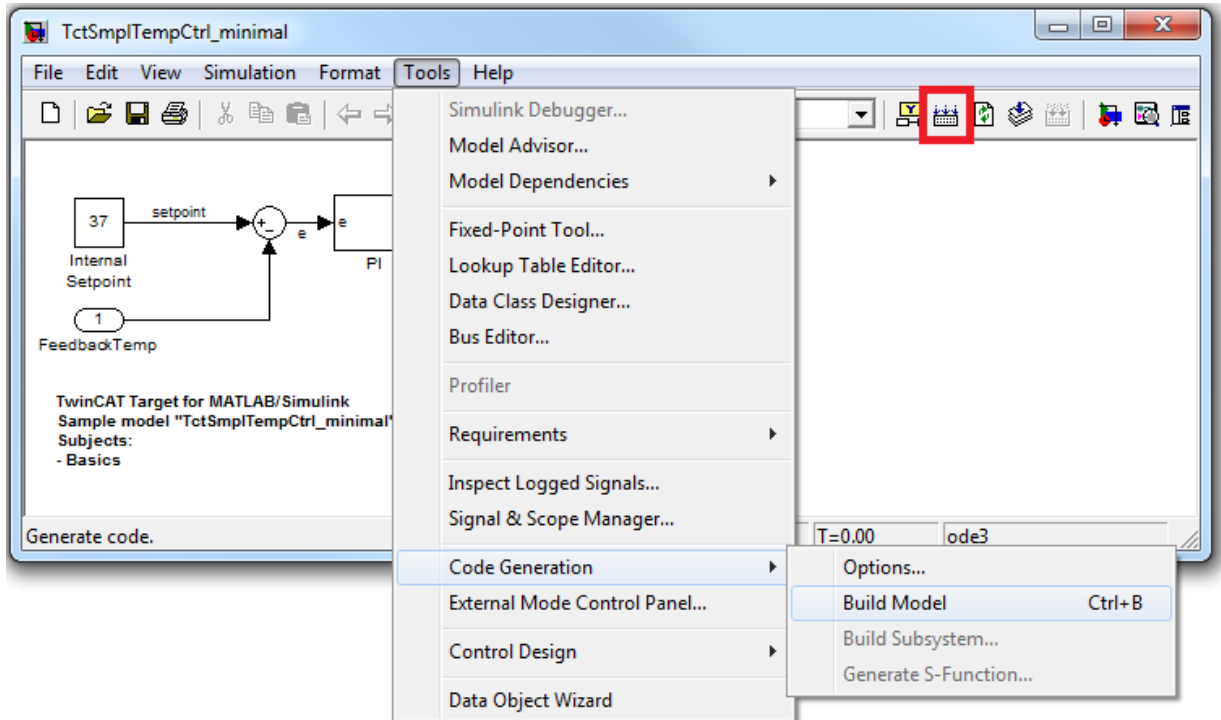
Generating a TwinCAT module

1. Open *TctSmpMinTempCtrl.mdl* in Simulink
2. Start **Model Explorer**
3. Under **Configuration -> Code Generation**, select the **System target file *TwinCAT.tlc*** - either key in manually or use the **Find** button:



4. Close **Model Explorer**

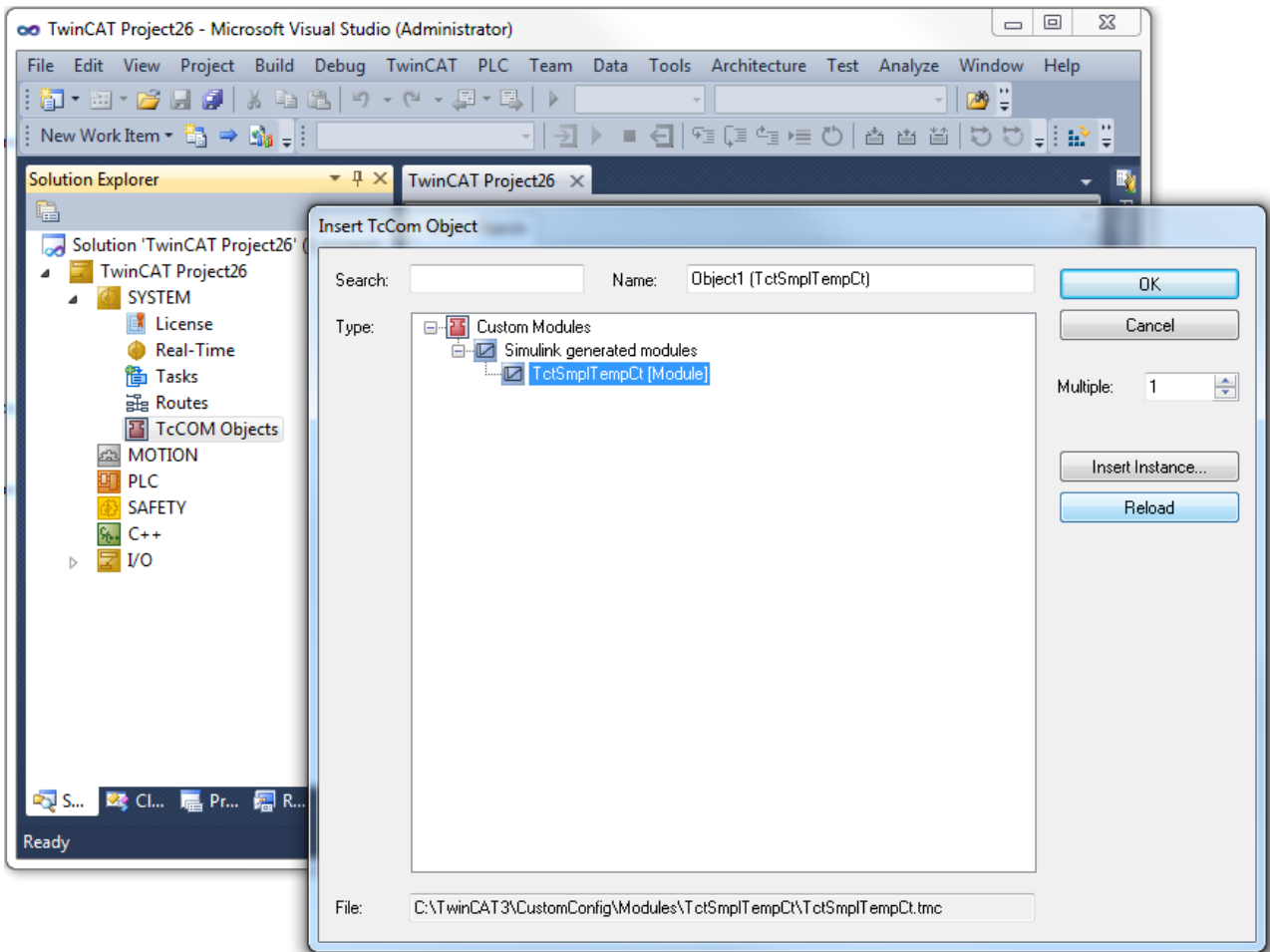
5. Start code generation via the Simulink menu item **Tools->Code Generation-> Build Model** or via the toolbar icon **Incremental build**



⇒ The progress of the code generation is shown in the MATLAB command window.

Using the generated TwinCAT module

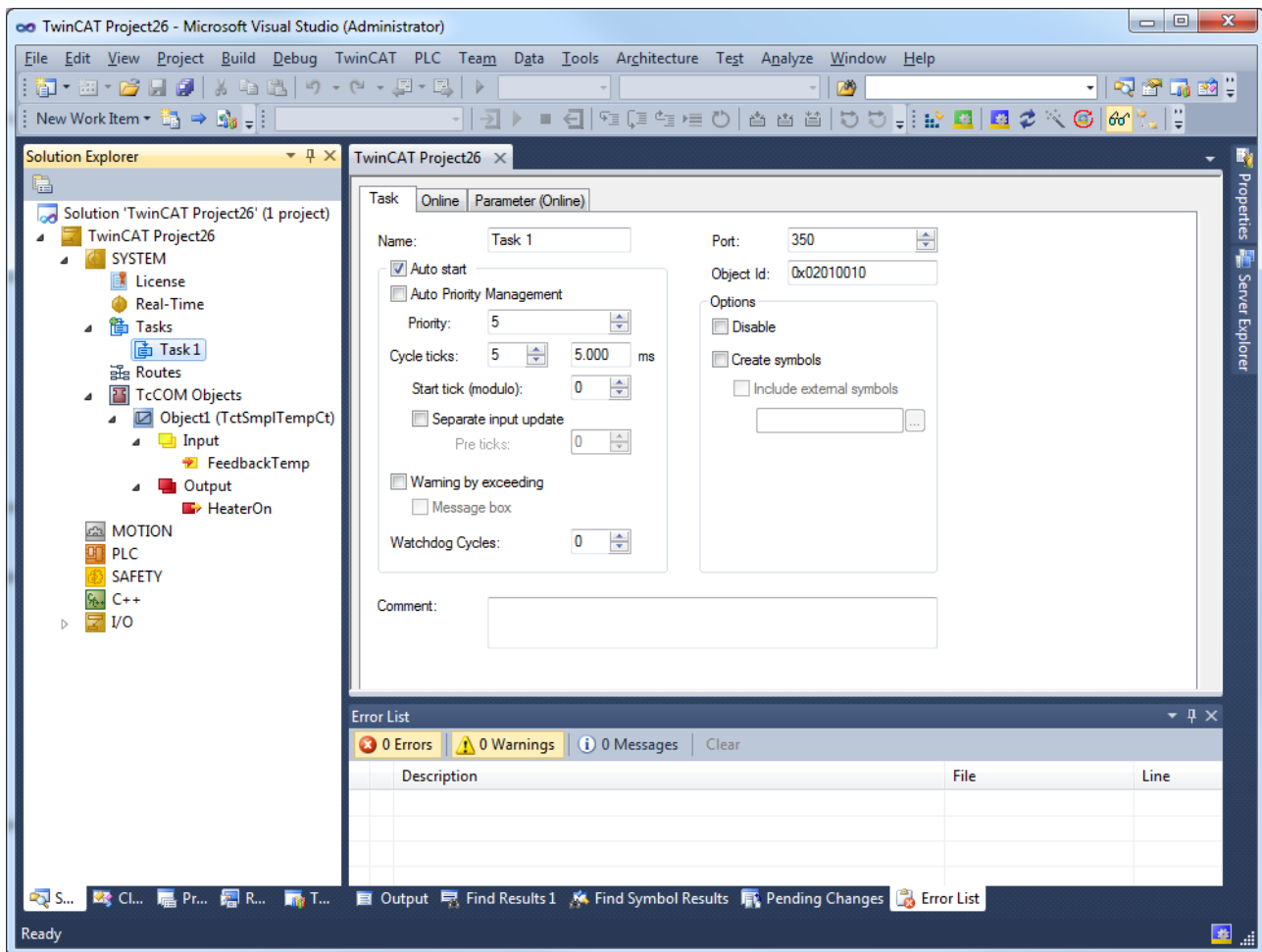
Open the TwinCAT development environment and create a new TwinCAT project. Expand node **System** in the **Solution Explorer**. Select the menu item **Add new item** in the context menu of node *TcCOM Objects*. The following dialog is displayed:



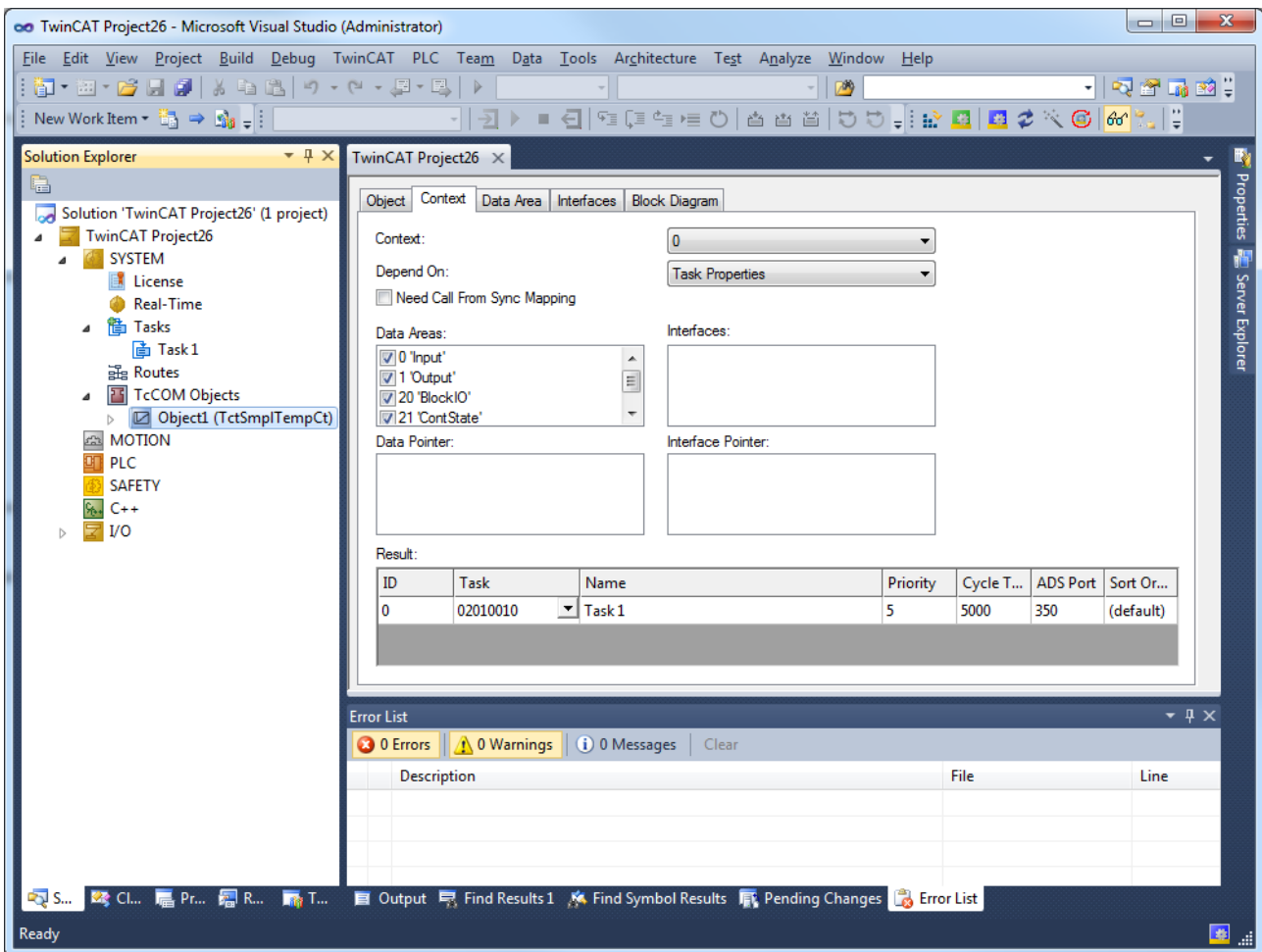
Select the generated module from the group **Custom Modules -> Simulink generated modules**. If XAE was started before the end of the code generation, first press the **Reload** button.

Add a new task using the context menu of the node **System ->Tasks** and configure the new task with the default parameters of the generated module:

- Priority: 5
- Cycle Time: 5 ms



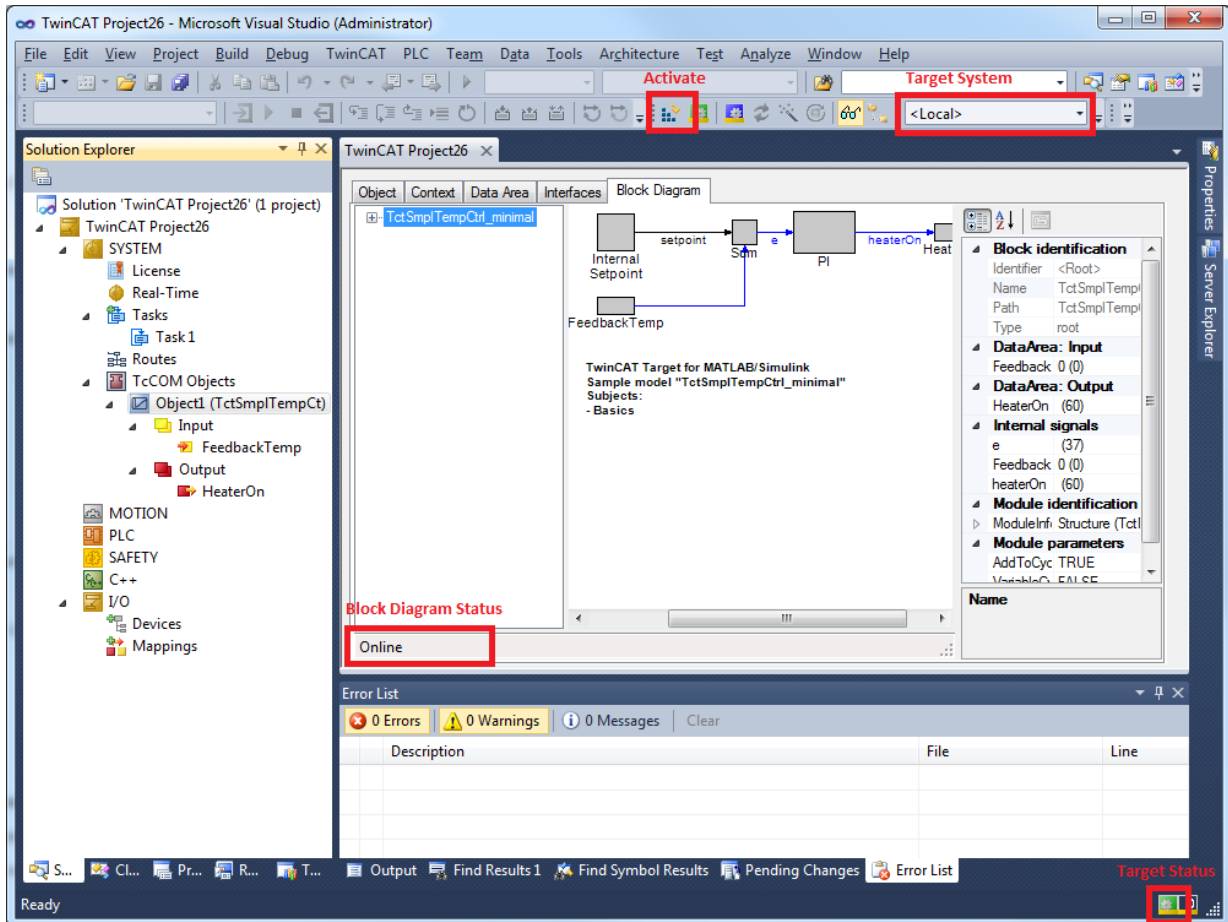
The module (with its default settings) should then have been configured automatically for attaching to this task. To verify this, select the object node **Object1 (TctSmpITempCt)** and open the **Context** tab. The **Result** table should contain the object ID and the object name of the task, as shown in the figure below:



The configuration is now completed and can be activated on the target system.

1. Select the target system, the current configuration should be activated.
2. If there is no license, activate a free trial license in order to execute the modules generated with Simulink (TC1320 or TC1220) on the target system.
3. Activate the configuration on your target system. Confirm the question to overwrite the current configuration, and start the TwinCAT system.
4. The status symbol on the target should change its colors to green (running).

5. If the **Block Diagram** tab was selected, the block diagram state changes to "Online", and the Properties table shows some online values.



3.8.2 Temperature Controller

Description

The following example extends the basics, shown in example "TemperatureController_minimal" by the following subjects:

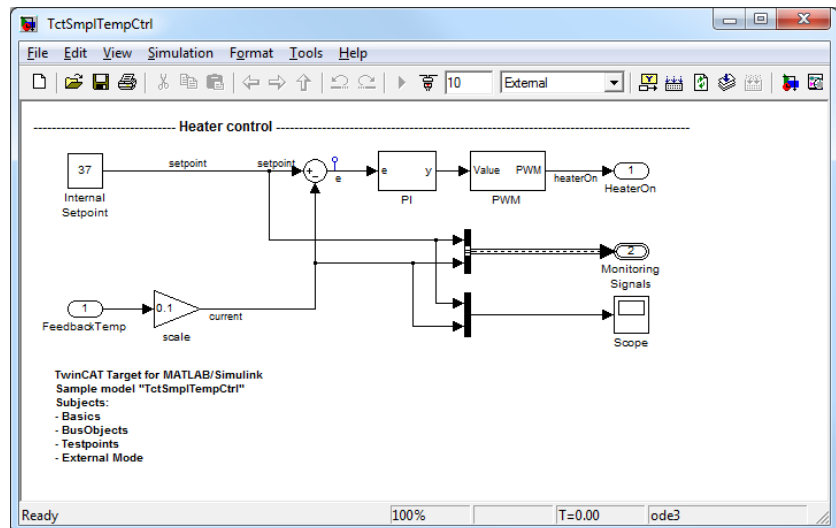
- [Parameter access \[► 69\]](#)
- [Using Bus Objects \[► 71\]](#)
- [Using Test Points \[► 72\]](#)
- [Using Referenced Models \[► 74\]](#)
- [Using External Mode \[► 76\]](#)
- [Generating TwinCAT modules from SubSystems \[► 77\]](#)

Overview of project directory

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/1539964811/.zip contains all the files for this example:

TctSmpITempCtrl.mdl

More advanced (but still very simple) temperature controller.



TctSmpICtrlSysPT2.mdl

Simple PT2 model for the controlled system.

TctSmpIClosedLoopCtrl.mdl

Model of a closed control loop, which was implemented through referencing of the controller models and the controlled system.

TctSmpITempCtrlParameters.mat Contains all the necessary model parameters.

TctSmpITempCtrlBusObjects.mat Contains all the required Simulink BusObjects (structure definitions).

_PrecompiledTcComModules

This subdirectory contains readily compiled TwinCAT modules that were generated from the enclosed Simulink models. They enable the integration of a module in TwinCAT to be tested, without having to generate the module first. They can be used in situations where a MATLAB license is not yet available, for example. A quick reference guide for module installation on the development PC is also enclosed.

Info: To start the module on an x64 target system, the system must be switched to test mode!

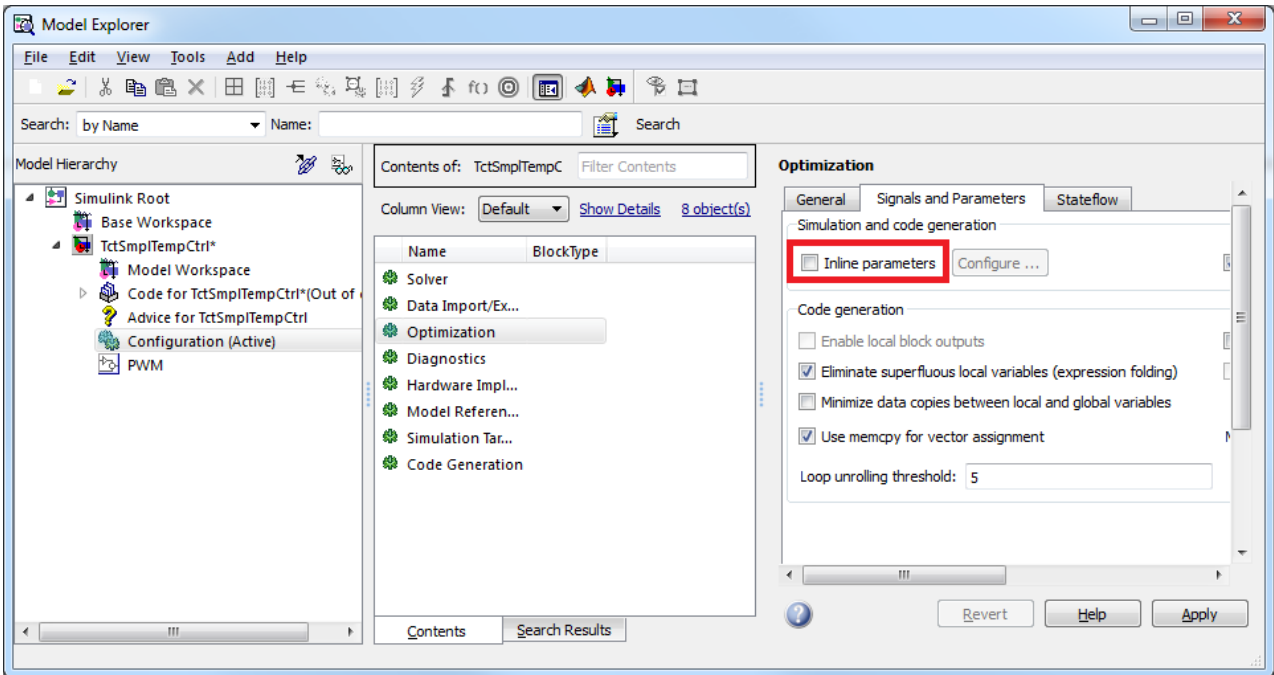
_PreviousSimulinkVersions

The MDL files described above are stored in the file format of the current Simulink version. This subdirectory contains the models in the file format of elder Simulink versions.

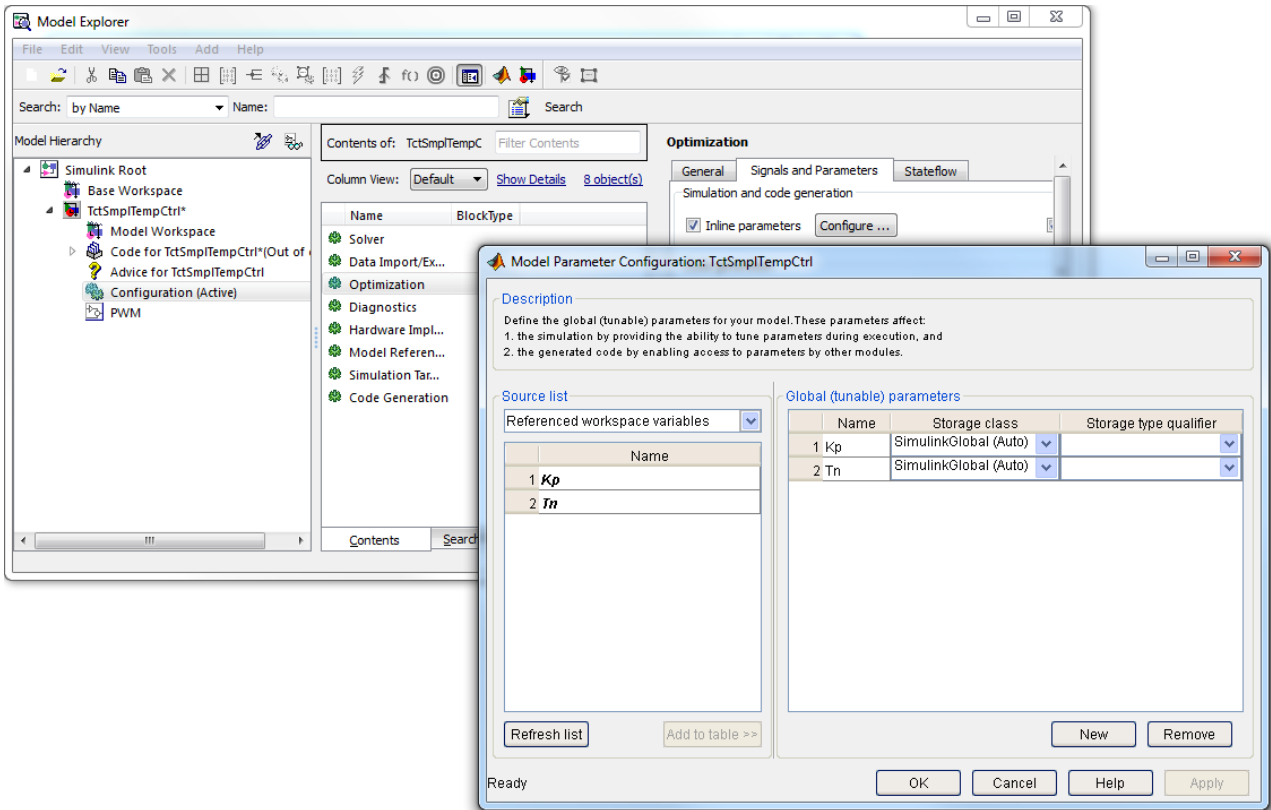
Parameter access

TctSmpITempCtrl.mdl has no embedded parameter values (inline parameters), i.e. the parameter values are stored in the corresponding model parameter structure. In addition, under the tab **TCT Advanced** of the coder settings, the module generator is configured such that ADS access to the parameters and generation of ADS symbols is allowed. ADS access is then possible from TwinCAT Scope View or other ADS clients.

The **Block diagram** tab in TwinCAT XAE is an ADS client. Access to its parameter depends on these settings.



If the option **Inline parameters** is activated without further configurations, all parameter values in the generated module codes are fixed. The **Configure...** button next to **Inline parameters** can be used to open a configurator, in which you can select the variables of the MATLAB working area that are to remain configurable in the generated module:



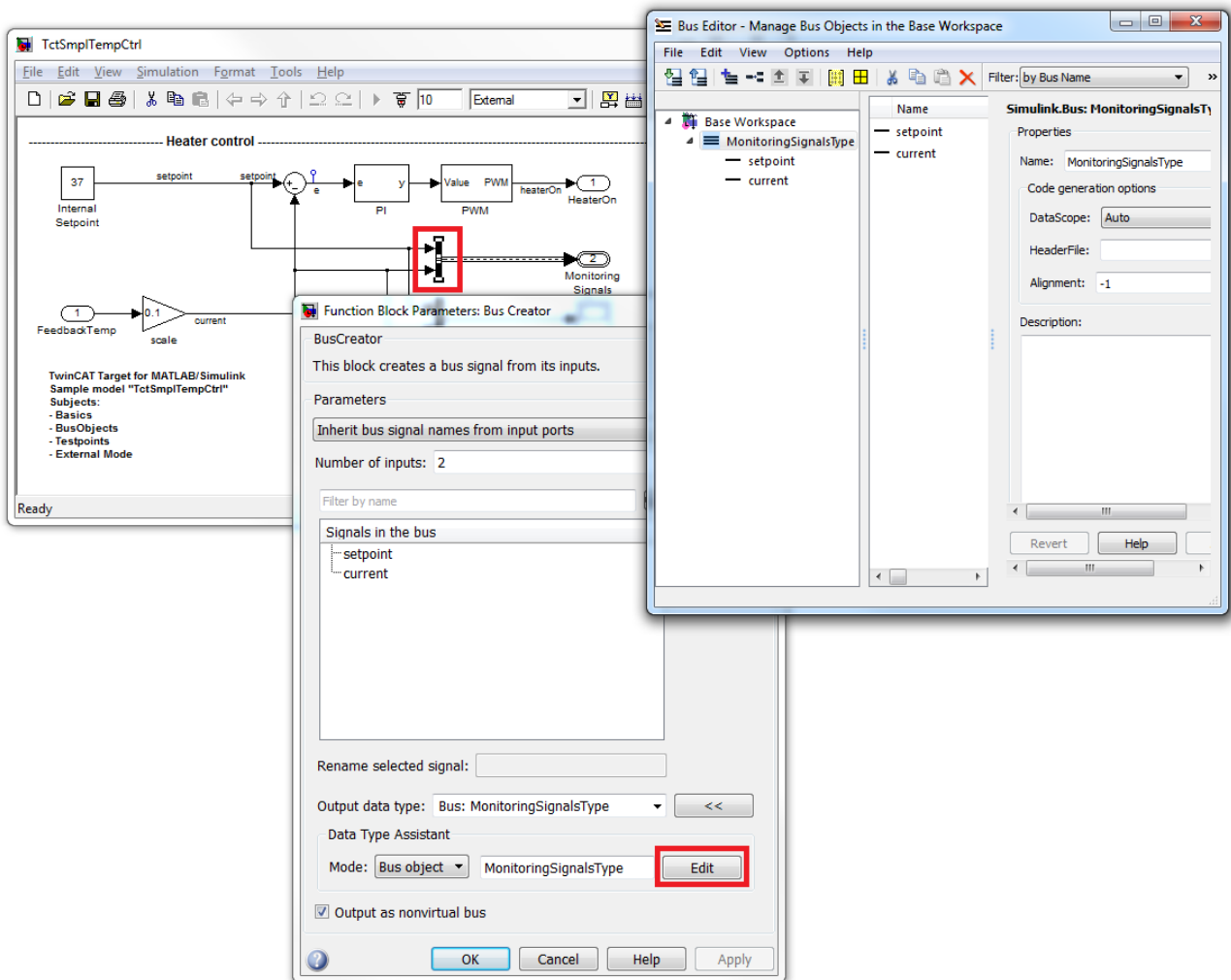
In the sample shown, only the workspace variables *Kp* and *Tn* remain configurable, which means that only the Simulink block parameters that depend on these parameters are configurable. The parameter structure is reduced to these two elements.

For further information on *parameter inlining*, see [Simulink documentation](#).

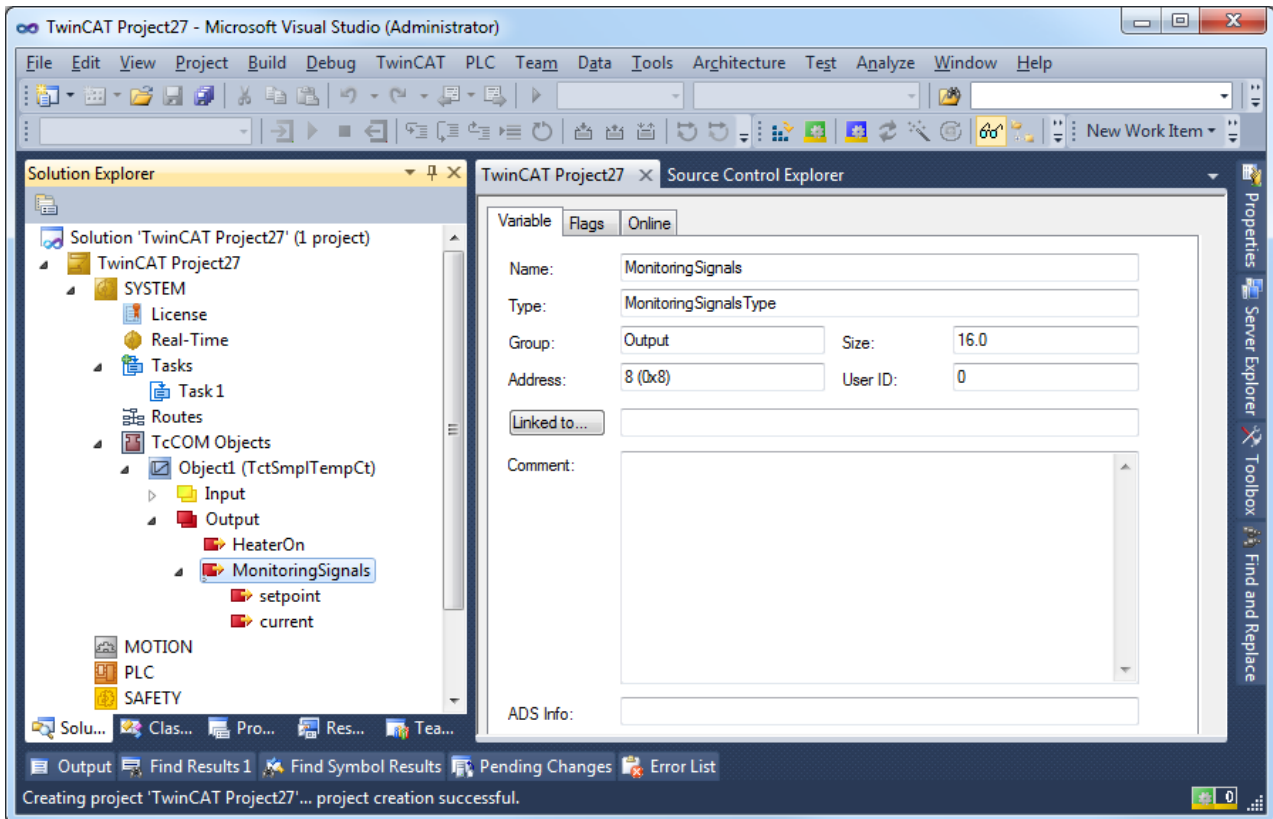
Using bus objects

Simulink BusObjects enable access to TwinCAT modules generated in Simulink via structured symbols. This sample contains a predefined BusObject called *MonitoringSignalsType*. It is an output structure, i.e. it assigns the signals it contains to a PLC module.

The configuration of a BusObject is started by double-clicking on the **BusCreator** block. To start the Bus Editor, click the **Edit** button in the Welcome screen, as shown in the figure below. Further information on using BusObjects can be found in the [Simulink documentation](#).



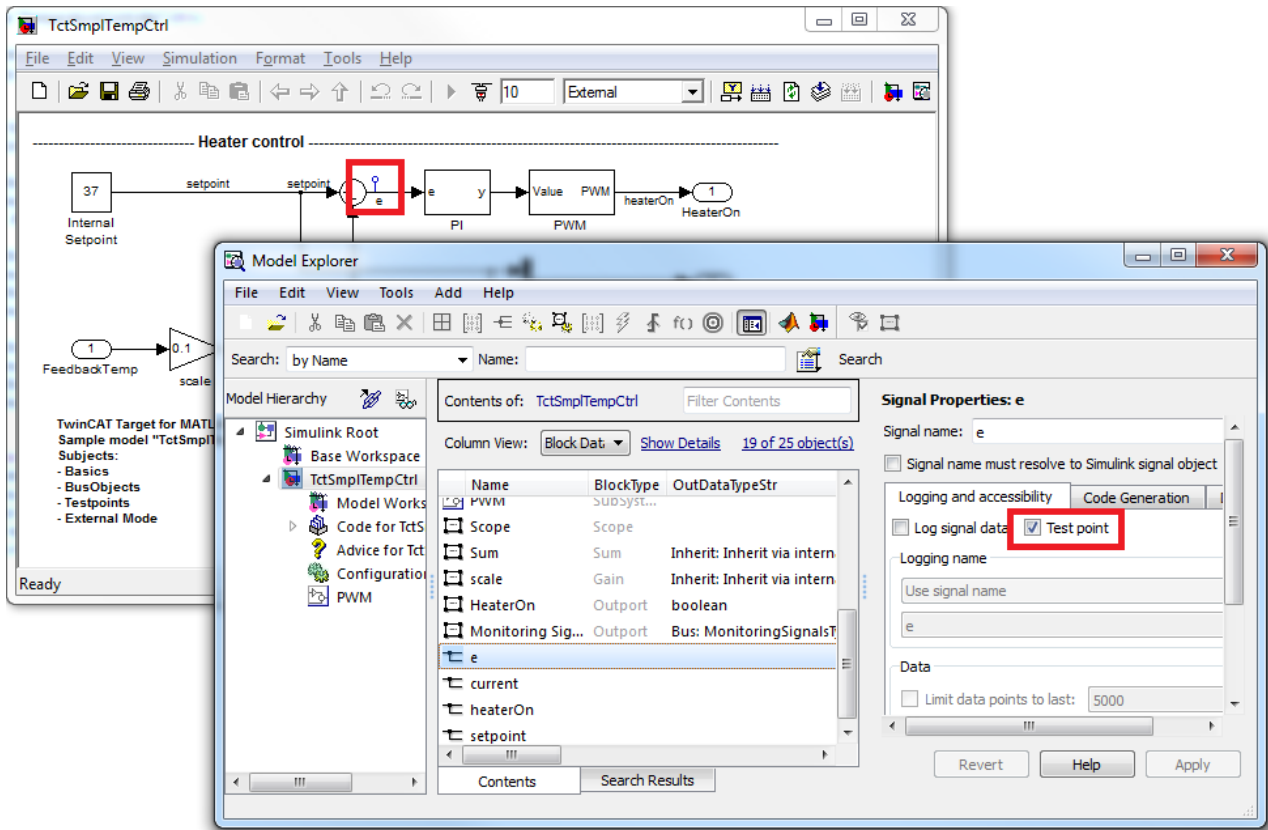
During instantiation of the generated module in a TwinCAT project, the specified BusObject is imported into the TwinCAT project as a global TwinCAT data type. This data type is used by the generated module itself for displaying the output structure, although it can also be used by other modules, such as a PLC, for example, which are linked to this output structure.



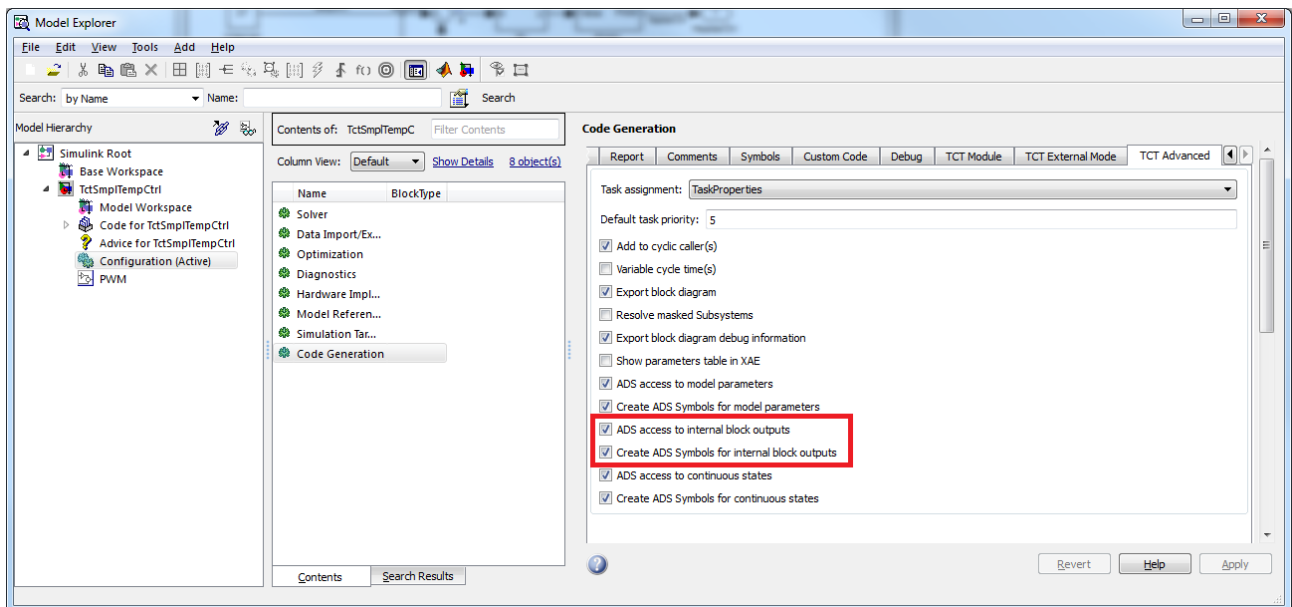
Using test points

In Simulink you can specify test points on signals for monitoring by Simulink "Floating Scope", for example. If the TwinCAT Target module generator is used, signals with such test points are invariably declared as member variable for the generated TwinCAT module. This enables ADS access to the signal. For more information on test points, see [Simulink documentation](#).

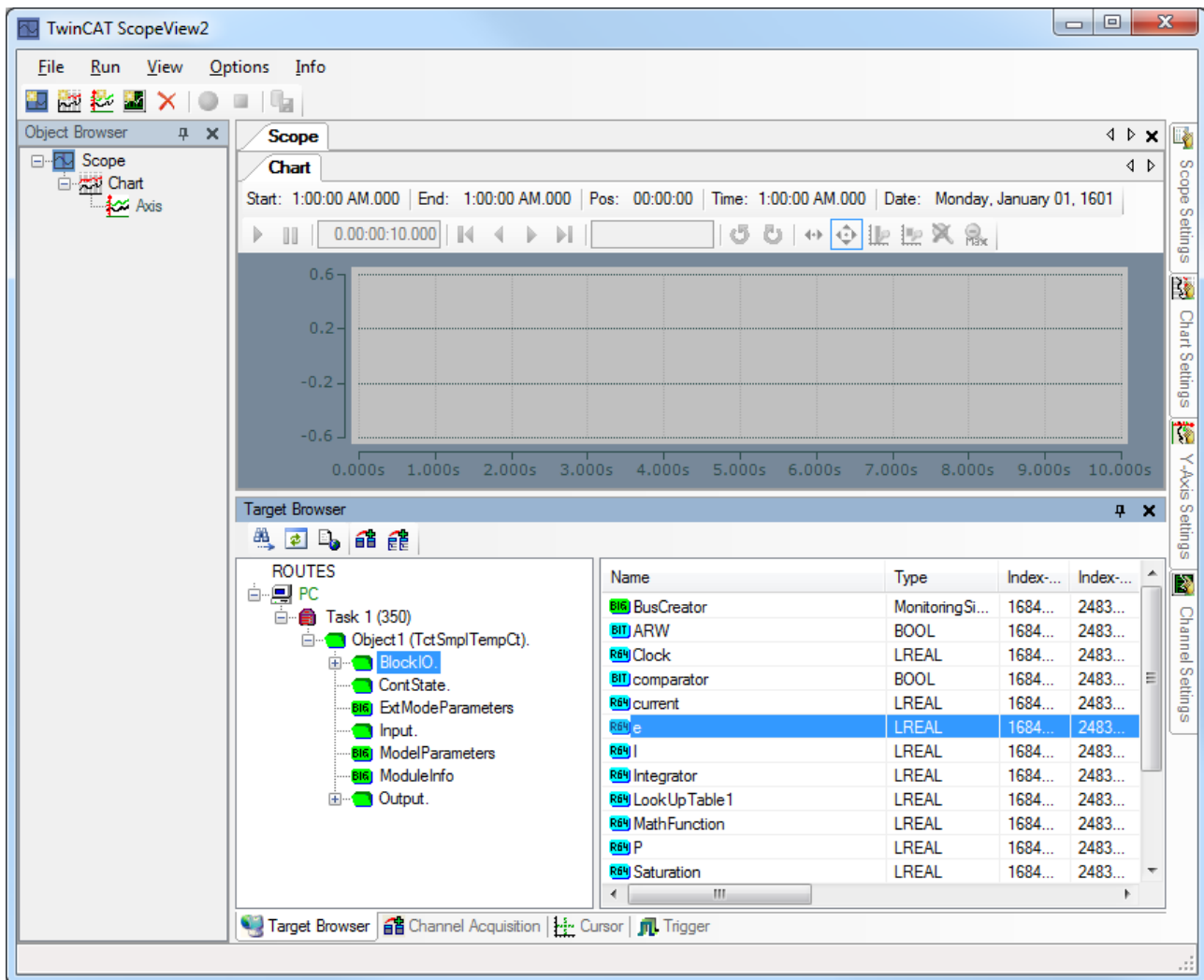
In this sample, the **Model Explorer** is used to define a test point for the control difference e :



To enable ADS access, enable **internal block output** in the code settings under the **TCT Advanced** tab:

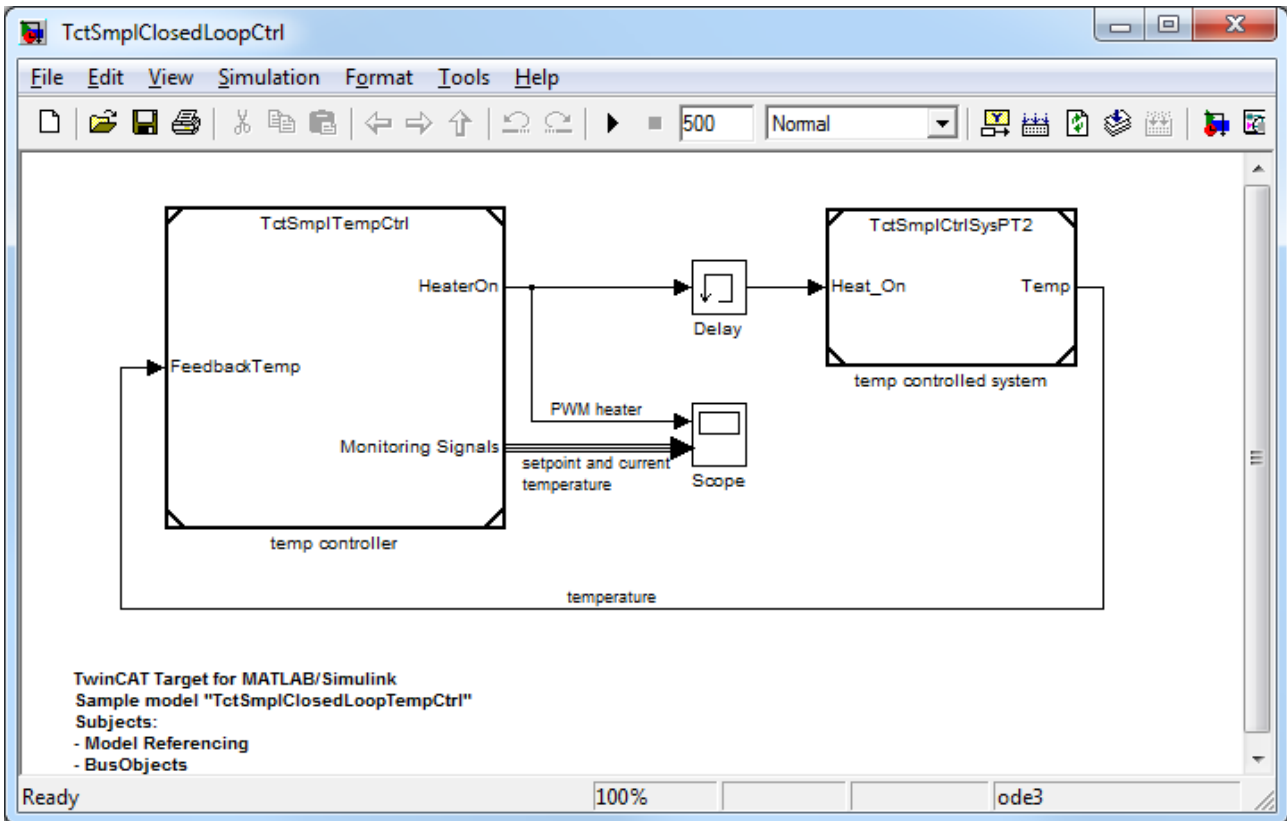


In this way you can use TwinCAT Scope View to access the signal with test points and some other block output variables when the generated TwinCAT module is executed.



Using referenced models

Open the model *TctSmpI ClosedLoopCtrl.mdl*, which contains two model references. Referenced models are the temperature controller described above and a simple P-T2 model of a temperature control system.



Such model referencing has several advantages, both in general and in combination with TwinCAT Target. Two basic options for structured modelling and, particularly in this example, for controller design are:

Simulation for optimizing the controller:

Optimization of the controller design based on simulation of the control loop with MATLAB/Simulink, followed by transfer of the optimized controller into the real-time environment of TwinCAT 3. Thanks to the use of standard Simulink input and output blocks for the definition of the TwinCAT module process images, no changes in the controller model are required before module generation commences.

Reuse and faster creation of models:

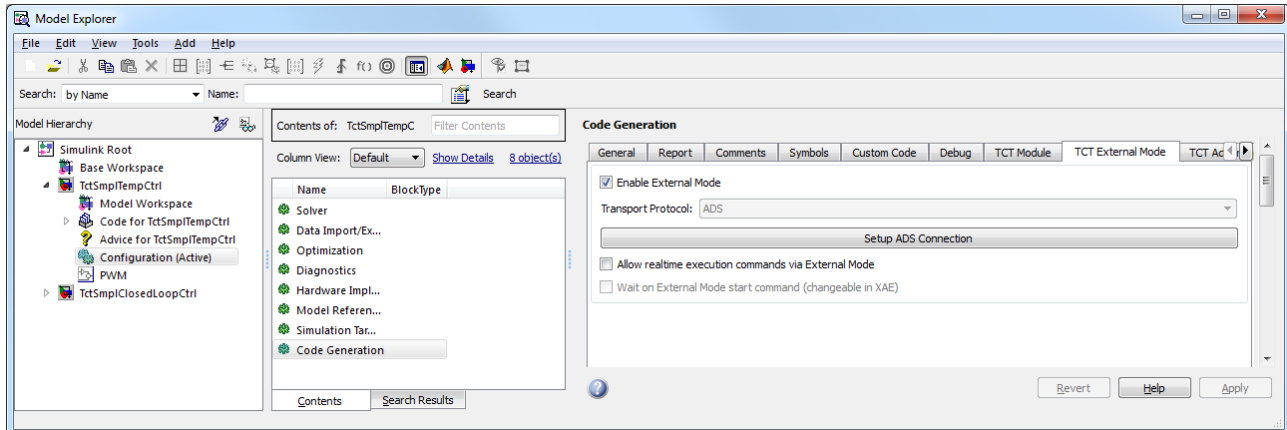
A model can be referenced several times in one or several higher-level models. In this way, the models can be divided into reusable functional units, similar to text programming languages, where the code is structured into functions or methods. This improves the readability of complex models. The generated code of referenced models is compiled into static libraries, which are only updated if the referenced model was modified since the last code generation. This can speed up the development of complex models, if parts that are only rarely modified are stored in referenced models. In this example, model generation can be started for a control loop model, and a real-time control loop simulation can be executed in the TwinCAT runtime.

Note on licenses:

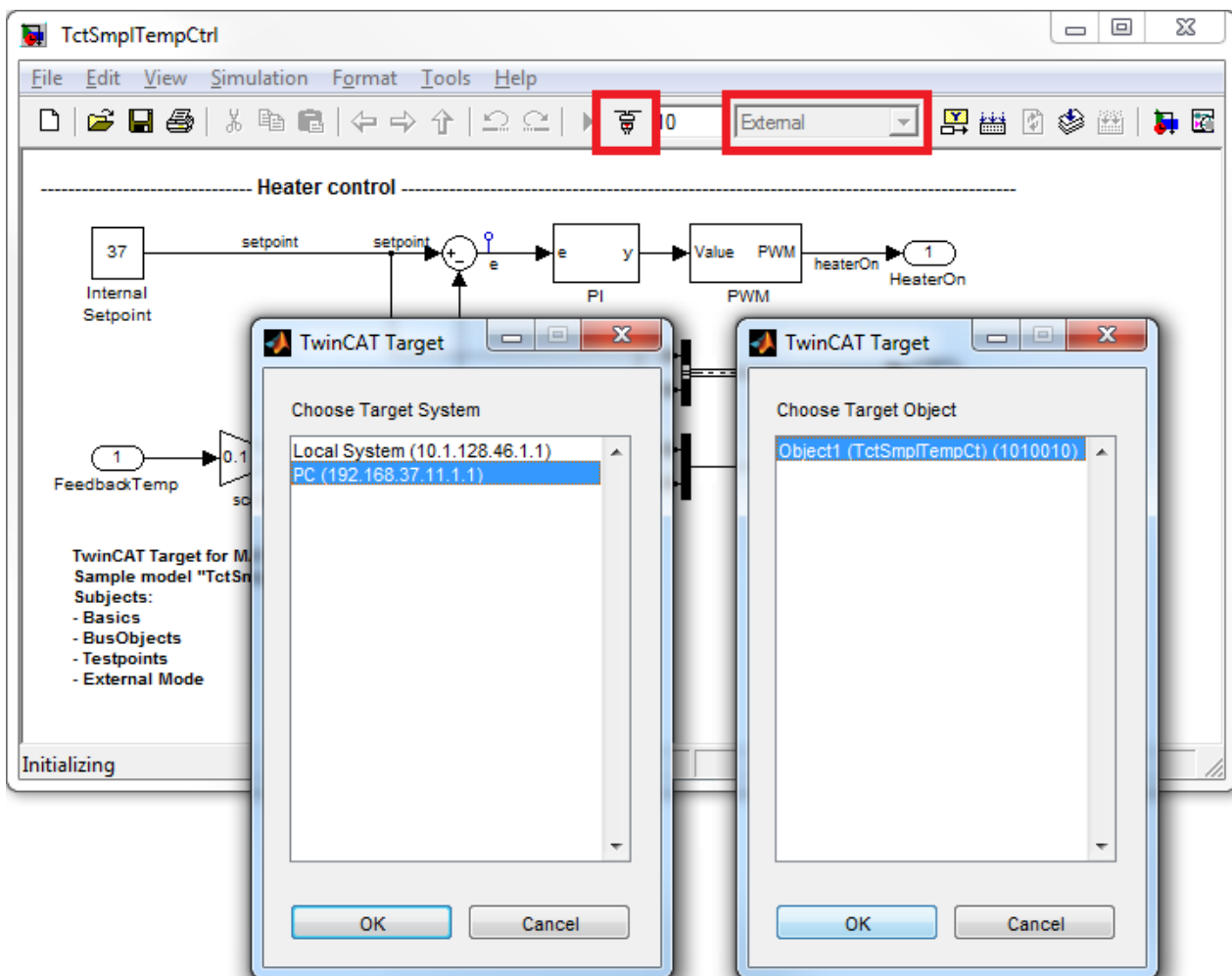
The control loop model of this example can only be compiled into a TwinCAT module with a valid TwinCAT Target license (TE1400). Otherwise, this model exceeds the limits for unlicensed models.

Using external mode

The temperature controller model *TctSmpITempCtrl.mdl* has been preconfigured so that ExternalMode connections are permitted:



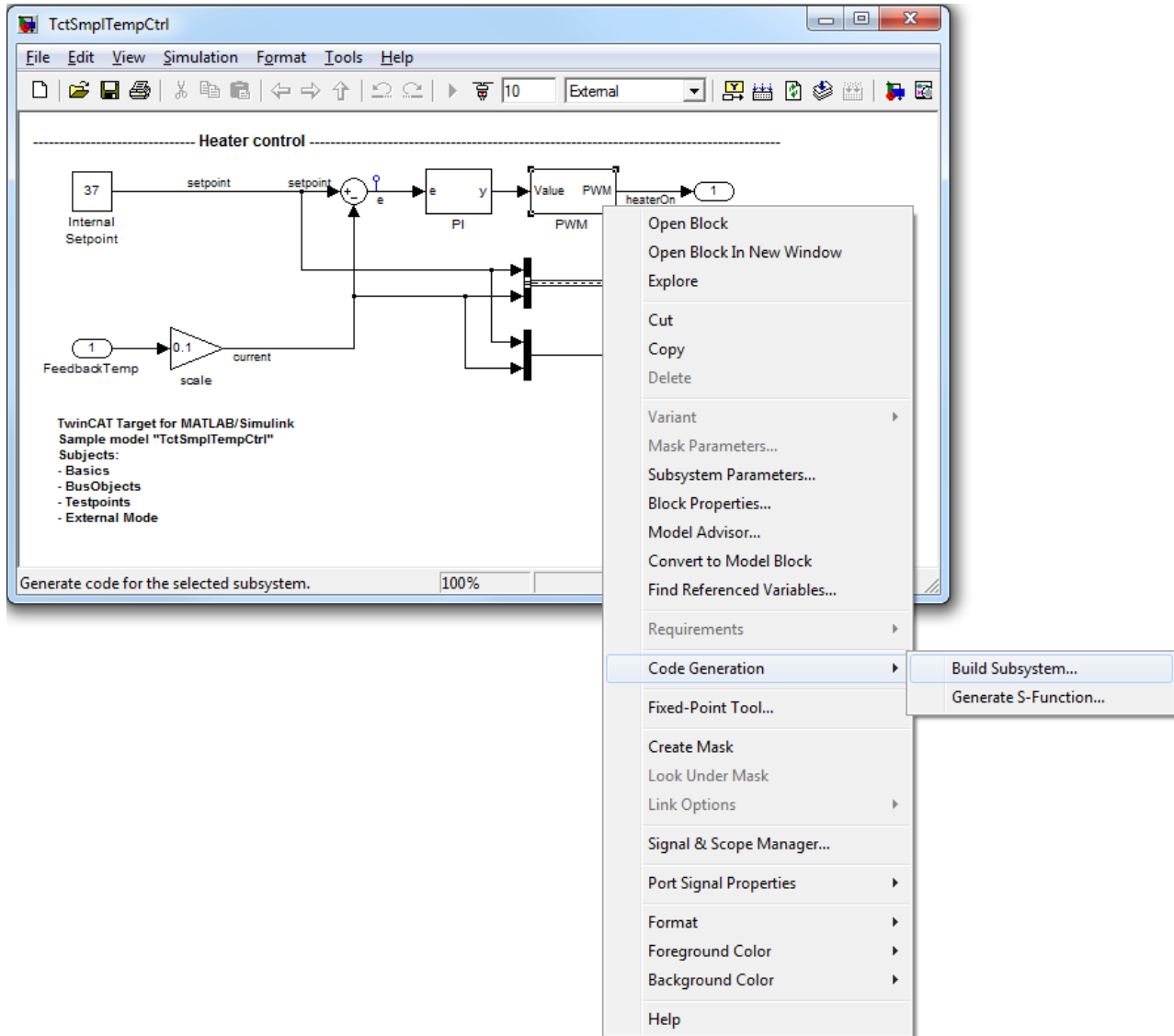
Because of these configurations, you can use the **Connect to Target** icon in the Simulink toolbar to establish a connection with the generated temperature controller via ExternalMode. The module must have been previously generated and started on a TwinCAT system and an ADS route must have been configured between your development system and the corresponding target system. A number of dialogs are displayed to help you navigate to the desired module instance.



You can now use the **Scope** block in Simulink to monitor the real-time signals of the generated and now connected TwinCAT module. You can also change the value of the **Internal Setpoint** block, for example. As soon as the parameter change is confirmed, it is downloaded directly to the target module. This is only possible for adjustable parameters if the model parameters *are not inlined* (see "Parameter access [▶ 69]").

Generating TwinCAT modules from subsystems

Creating a TwinCAT module in a Simulink subsystem, instead of the entire model, via the subsystem context menu:



3.8.3 SFunStaticLib

Use cases

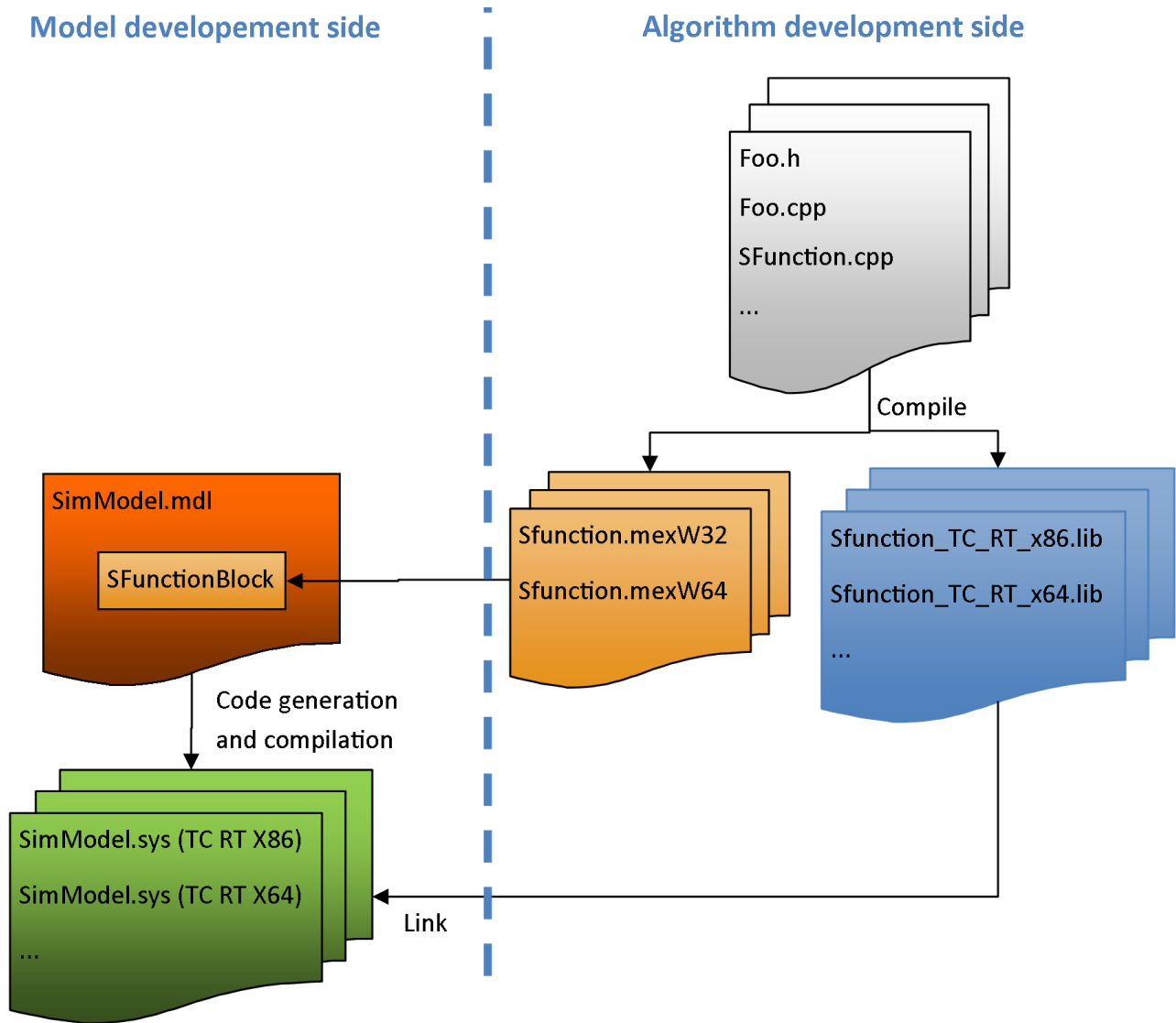
Encapsulating own code within static libraries can be useful to

- speed up module generation, if the code contains algorithms, which are not changed frequently
- deliver TwinCAT Target compatible SFunction algorithms to customers, without the need to hand out the source code but only the compiled libraries

Description

The following example illustrates how TwinCAT modules are generated using SFunctions from Simulink models, for which no source code is available. In this case, the SFunction functionality can be integrated into the generated TwinCAT module via static libraries. However, this presupposes that suitable libraries are available for all TwinCAT platforms, for which the module is to be created.

The following diagram illustrates the typical workflow when using third-party algorithms in a user-created Simulink model:



In this example, the source code for the "algorithm development side" is available and can be compiled for all TwinCAT platforms. It shows how

- SFunctions are generated with suitable TwinCAT libraries
- such libraries are made available (e.g. to customers)
- such libraries are used in own models

Project directory overview

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/1539910283/.zip contains all the files necessary to reproduce this example:

TctSmpISFunStaticLib.mdl	contains the model that references the SFunction.
BuildLibsAndSFunction.m	contains an M-script that creates the static library for all currently available TwinCAT platforms and creates the SFunction.
OpenLibProject.m	contains an M-script that defines the MATLAB build environment for Visual Studio, such as MATLAB Include and library directories. The static library is then opened in Microsoft Visual Studio 2010 with the predefined environment variables.
Subdirectory SFunLibProject	contains the SFunction project.
Subdirectory BuildScripts	contains several M-scripts for "BuildLibsAndSFunction.m" and "OpenLibProject.m".

<p>_PrecompiledTcComModules</p>	<p>This subdirectory contains readily compiled TwinCAT modules that were generated from the enclosed Simulink models. They enable the integration of a module in TwinCAT to be tested, without having to generate the module first. They can be used in situations where a MATLAB license is not yet available, for example. A quick reference guide for module installation on the development PC is also enclosed.</p> <p>Info: To start the module on an x64 target system, the system must be switched to test mode!</p>
<p>_PreviousSimulinkVersions</p>	<p>The MDL files described above are stored in the file format of the current Simulink version. This subdirectory additionally contains the models in the file format elder Simulink versions.</p>

Build SFunction and appropriate static link libraries

Build requirements

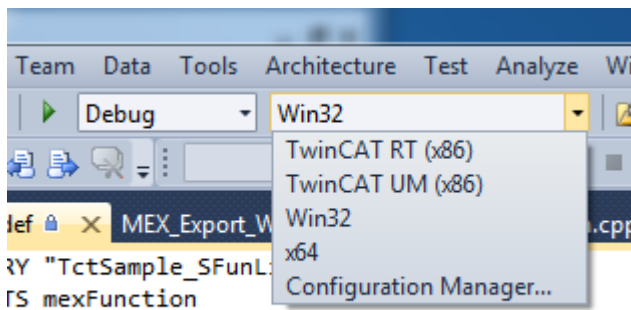
It is recommended to have TwinCAT 3 installed on your system to build the binaries, but not required. Requirements are:

- Windows Driver Kit** installed on your system and the environment variable *WinDDK* set to its path.
- TwinCAT SDK** installed on your system and the environment variable *TwinCatSdk* set to its path.

For more information about this requirement, see the system requirements in the TwinCAT 3 documentation.

Creating binary files manually

The binary files can be created manually with Visual Studio. To do this, execute *OpenLibProject.m*. This prepares the required environment variables and opens the SFunction project in Visual Studio. Create a project for all platforms that should be supported.



- TwinCAT xx(xxx)** Creates the platform-specific static library, which is linked to the generated TwinCAT module.
- Win32** Creates the .MEXW32 SFunction for running the simulation of the model with Simulink in 32-bit MATLAB. It can only be created if Visual Studio was started from 32-bit MATLAB.
- x64** Creates the .MEXW64 SFunction for running the simulation of the model with Simulink in 64-bit MATLAB. It can only be created if Visual Studio was started from 64-bit MATLAB. In order to create the MEX files used in this example for 32- and 64-bit MATLAB, Visual Studio must be started from both MATLAB versions.

Build the binaries via build script

Alternatively to the manual build procedure, in order to speed up the build procedure for a quick overview, run **BuildLibsAndSFunction.m**. This prepares the build environment variables and invokes MSBUILD multiple times to build the .LIB and .MEXWxx files for each TwinCAT platform and for the current MATLAB platform architecture (32 or 64 Bit). To build the MEX files of this example for 32 and 64 Bit MATLAB *BuildLibsAndSFunction.m* has to be executed with both MATLAB variants.

After the build procedure, all the build output files are copied to the subfolder *LibProject\TctSample_SFunLib\BuildOutput*. All necessary binaries are additionally copied to *LibProject\TctSample_SFunLib\LibPackage*.

Deliver the binaries

LibProject\TctSample_SFunLib\LibPackage is the folder which can be delivered to customers who want to use the now built - TwinCAT Target compatible - SFunction. Copy the content of this folder to the users system, more precisely to the folder *%TwinCat3Dir%\Functions\TE1400-TargetForMatlabSimulink\Libraries*. If *BuildLibsAndSFunction.m* was used for building the binaries, this has already been done for the local system. The content of this folder should be:

Subfolders TwinCAT xx (xxx)

contain the static link libraries for different TwinCAT platforms. These are used when generating TwinCAT modules from appropriate Simulink models.

Subfolders Win32 / Win64

contain the MEX files (and optionally some static link libraries) for the different MATLAB platform architectures (32 and/or 64 Bit). Either subfolder Win32 or Win64 is added to the MATLAB path when setting up TwinCAT Target via *SetupTwinCatTarget.m*. Thus, SFunction MEX files are found by MATLAB can be used directly from this location.

Run simulation

To check if everything works, open "TctSmpISFunWrappedStaticLib.mdl" and start simulation. If the simulation starts without error messages, everything is prepared as needed.

Generate TwinCAT module

Configuring a model

The general settings for generating a TwinCAT module must be set, e.g. a fixed-step solver must be configured, and the system target file "TwinCAT.tlc" must be selected under the "General" tab in the model coder settings. For further information on the general configuration of the TwinCAT module generator see Quickstart.

In addition, the code generator must know which static libraries have to be linked to the generated code and where to find them. Enter this information in the corresponding option fields of the Simulink coder, as shown in the figures below.

Code Generation

General Report Comments Symbols Custom Code Debug TCT Module TCT External Mode TCT

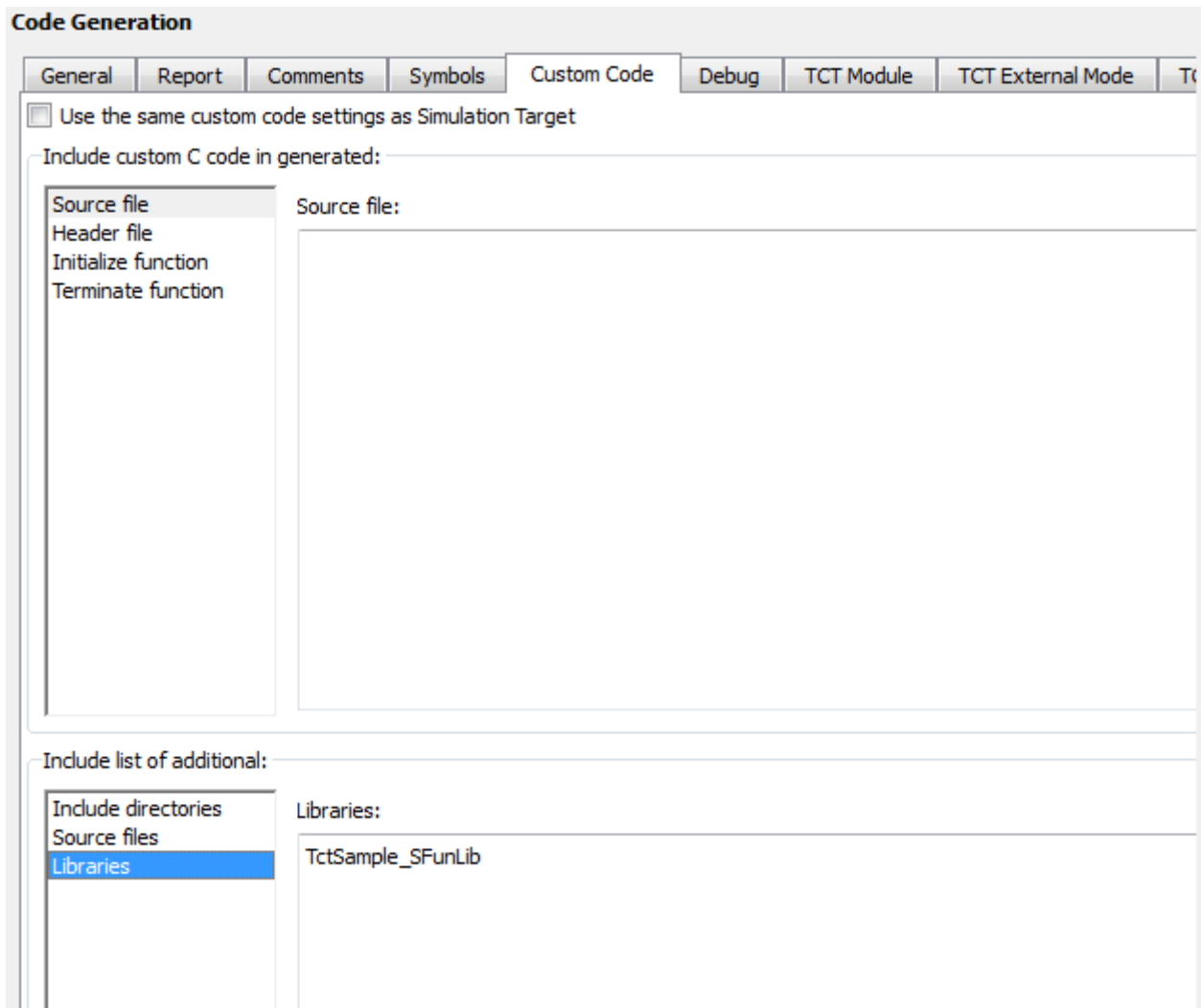
Use the same custom code settings as Simulation Target

Include custom C code in generated:

Source file	Source file:
Header file	
Initialize function	
Terminate function	

Include list of additional:

Include directories	Include directories:
Source files	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)"
Libraries	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Release"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Debug"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Release"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Debug"



The Include folder should already have been created automatically by TwinCAT Target. The Libraries setting must contain the names of the static libraries to be linked.

Background information for these settings:

In this example (and in most other cases) there are several instances of these libraries in the specified folders. MSBUILD decides which version is linked to the module when the generated TwinCAT module binary files are linked.

How to use this example as a template

The following list provides a short overview of the easiest way, to create an own TwinCAT Target compatible SFunction:

1. Copy the sample folder
2. Replace the MDL file by your own
3. Rename the VCXPROJ file to the desired name of your SFunction
4. Copy your source files to the directory where the VCXPROJ file is located
5. Adapt the scripts *BuildLibsAndSFunction.m* and *OpenLibProject.m* to the new project name
6. Open the VCXPROJ file using *OpenLibProject.m*

7. Remove the existing CPP files from the project
8. Add your own CPP files to the project
9. Adapt the DEF file contents to the new project name
10. If necessary, add include directories, dependency directories and libraries to the compiler and linker settings
11. Build the project (for different platforms and/or configurations)
12. Close the VCXPROJ file
13. Run *BuildLibsAndSFunction.m*

3.8.4 SFunWrappedStaticLib

Use cases

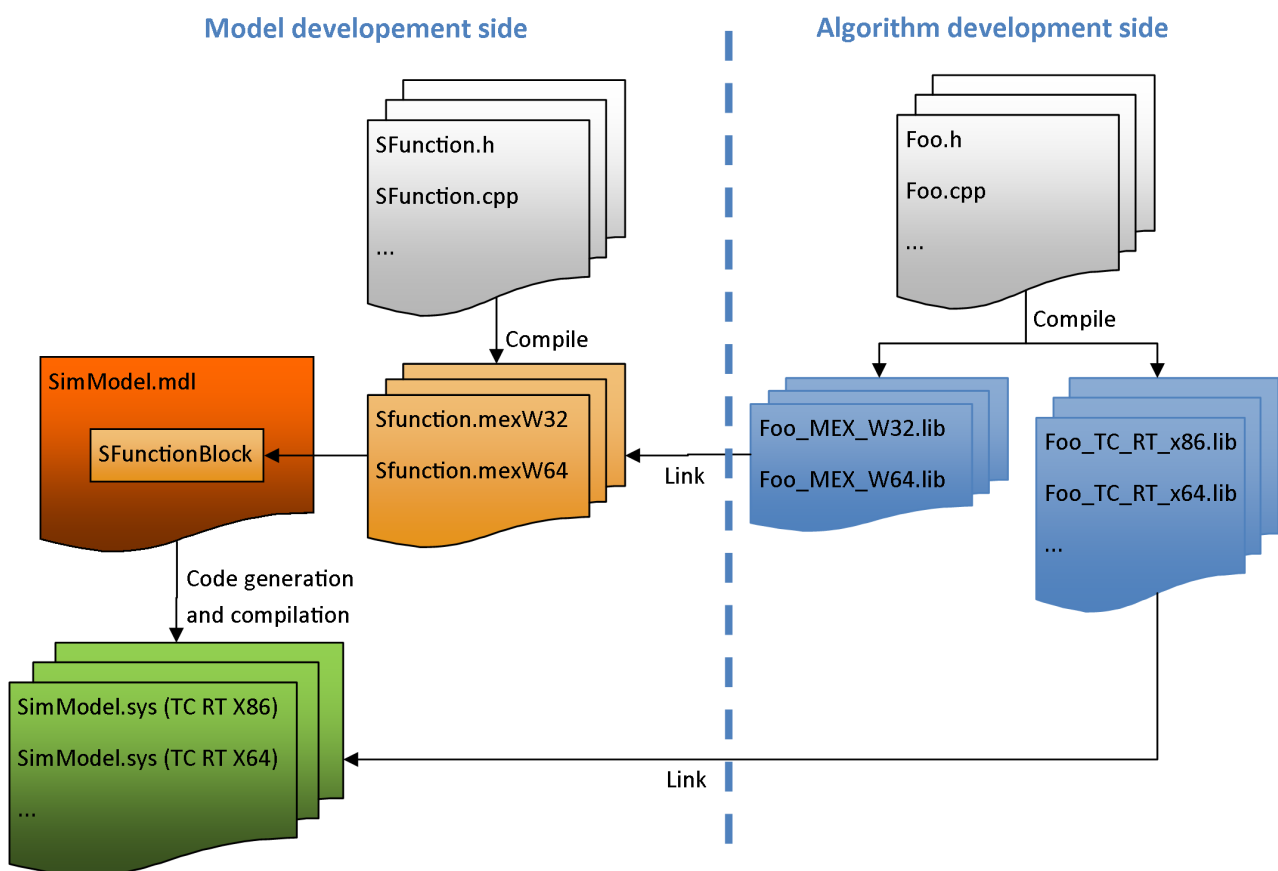
Encapsulating own code within static libraries can be useful to

- speed up module generation, if the code contains algorithms, which are not changed frequently
- deliver TwinCAT Target compatible SFunction algorithms to customers, without the need to hand out the source code but only the compiled libraries

Description

The following example shows the configuration of the module generator when using Sfunctions that depend on statically linked libraries. For this type of code integration, a suitable library must be available for all TwinCAT platforms for which the module is to be created.

The following diagram illustrates the typical workflow when using third-party algorithms in a custom Simulink model:



In this example, the source code for the "algorithm development side" is available and can be compiled for all TwinCAT platforms. It shows how

- dependent libraries are created

- such libraries are made available (e.g. to customers)
- such libraries are used in own models

Overview of project directory

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/1539911947/.zip contains all the files necessary to reproduce this example:

TctSmpISFunWrappedStaticLib.mdl	contains the model that references the SFunction.
TctSample_SFunLibWrapper.cpp	must be present on the target system. Contains the source code of the SFunction.
StaticLib.cpp	Simple example of source code of a static library.
BuildLibsAndSFunction.m	contains an M-script that creates the static library for all currently available TwinCAT platforms and creates the SFunction.
OpenLibProject.m	contains an M-script that defines the MATLAB build environment for Visual Studio, such as MATLAB Include and library directories. The static library is then opened in Microsoft Visual Studio 2010 with the predefined environment variables.
Subdirectory LibProject	contains the static library.
Subdirectory BuildScripts	contains several M-scripts for "BuildLibsAndSFunction.m" and "OpenLibProject.m".
_PrecompiledTcComModules	This subdirectory contains readily compiled TwinCAT modules that were generated from the enclosed Simulink models. They enable the integration of a module in TwinCAT to be tested, without having to generate the module first. They can be used in situations where a MATLAB license is not yet available, for example. A quick reference guide for module installation on the development PC is also enclosed. Attention: To start the module on an x64 target system, the system must be switched to test mode!
_PreviousSimulinkVersions	The MDL files described above are stored in the file format of the current Simulink version. This subdirectory contains the models in the file format of elder Simulink versions

Create SFunction and corresponding statically linked libraries

Build requirements

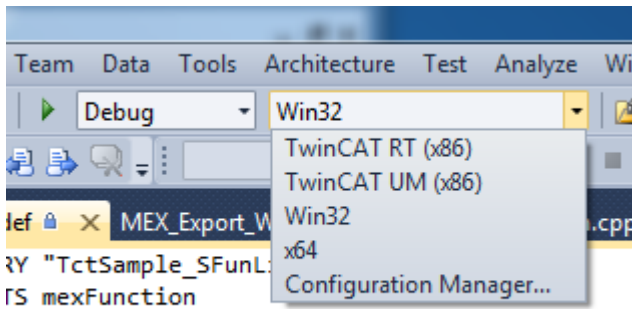
It is recommended to have TwinCAT 3 installed on your system to build the binaries, but not required. Requirements are:

Windows Driver Kit	installed on your system and the environment variable <i>WinDDK</i> set to its path.
TwinCAT SDK	installed on your system and the environment variable <i>TwinCatSdk</i> set to its path.

For more information about this requirement, see the system requirements in the TwinCAT 3 documentation.

Creating static libraries manually

The static libraries can be created manually with Visual Studio. To do this, execute *OpenLibProject.m*. This prepares the required environment variables and opens the SFunction project in Visual Studio. Create a project for all platforms that should be supported. The output file for all platforms is a static library:



Build the static link libraries via build script

Alternatively to the manual build procedure, run *BuildLibsAndSFunction.m*. This prepares the build environment and invokes MSBUILD multiple times to build the lib files for each platform.. Afterwards, all the build output files are copied to the subfolder *LibProject\TctSample_WrappedStaticLib\BuildOutput*. The .LIB files, which are necessary to build the SFunction and the generated TwinCAT modules are additionally copied to *LibProject\TctSample_WrappedStaticLib\LibPackage*.

Deliver the static libraries

LibProject\TctSample_WrappedStaticLib\LibPackage is the folder which can be delivered to users, which want to use this library inside their own - TwinCAT Target compatible - SFunctions. Copy the content of this folder to the users system, more precisely to the folder *%TwinCat3Dir%\Functions\TE1400-TargetForMatlabSimulink\Libraries*. This is also done by *BuildLibsAndSFunction.m* for the local system. The content of this folder should be:

Subfolders TwinCAT xx (xxx)

contain the static link libraries for different TwinCAT platforms. These are used when generating a TwinCAT module from an appropriate Simulink model.

Subfolders Win32 / Win64

contain the static link libraries for the different MATLAB platform architectures (32 and/or 64 Bit). These are used when generating a TwinCAT module from an appropriate Simulink model. To build the libraries for this example for 32 and 64 Bit MATLAB *BuildLibsAndSFunction.m* has to be executed with both MATLAB variants.

Compile mex file code

Before the SFunction can be used inside the Simulink model, it has to be built, too. Of course this can be done manually, as in any other SFunction. However, the MEX compiler has to be advised to link the static library to the SFunction.

When executing *BuildLibsAndSFunction.m*, this is also done automatically. Afterwards the file "SFunStaticLib.mexw32" should be located inside your working folder.

To check if everything works, open "TctSmpISFunWrappedStaticLib.mdl" and start simulation. If the simulation starts without error messages, everything is prepared as needed.

Generating a TwinCAT module

Configuring a model

The general settings for generating a TwinCAT module must be set, e.g. a fixed-step solver must be configured, and the system target file "TwinCAT.tlc" must be selected under the "General" tab in the model coder settings. For further information on the general configuration of the TwinCAT module generator see [Quickstart \[▶ 15\]](#).

In addition, the code generator must know which static libraries have to be linked to the generated code and

where to find them. Enter this information in the corresponding option fields of the Simulink coder, as shown in the figures below.

Code Generation

General Report Comments Symbols Custom Code Debug TCT Module TCT External Mode TCT

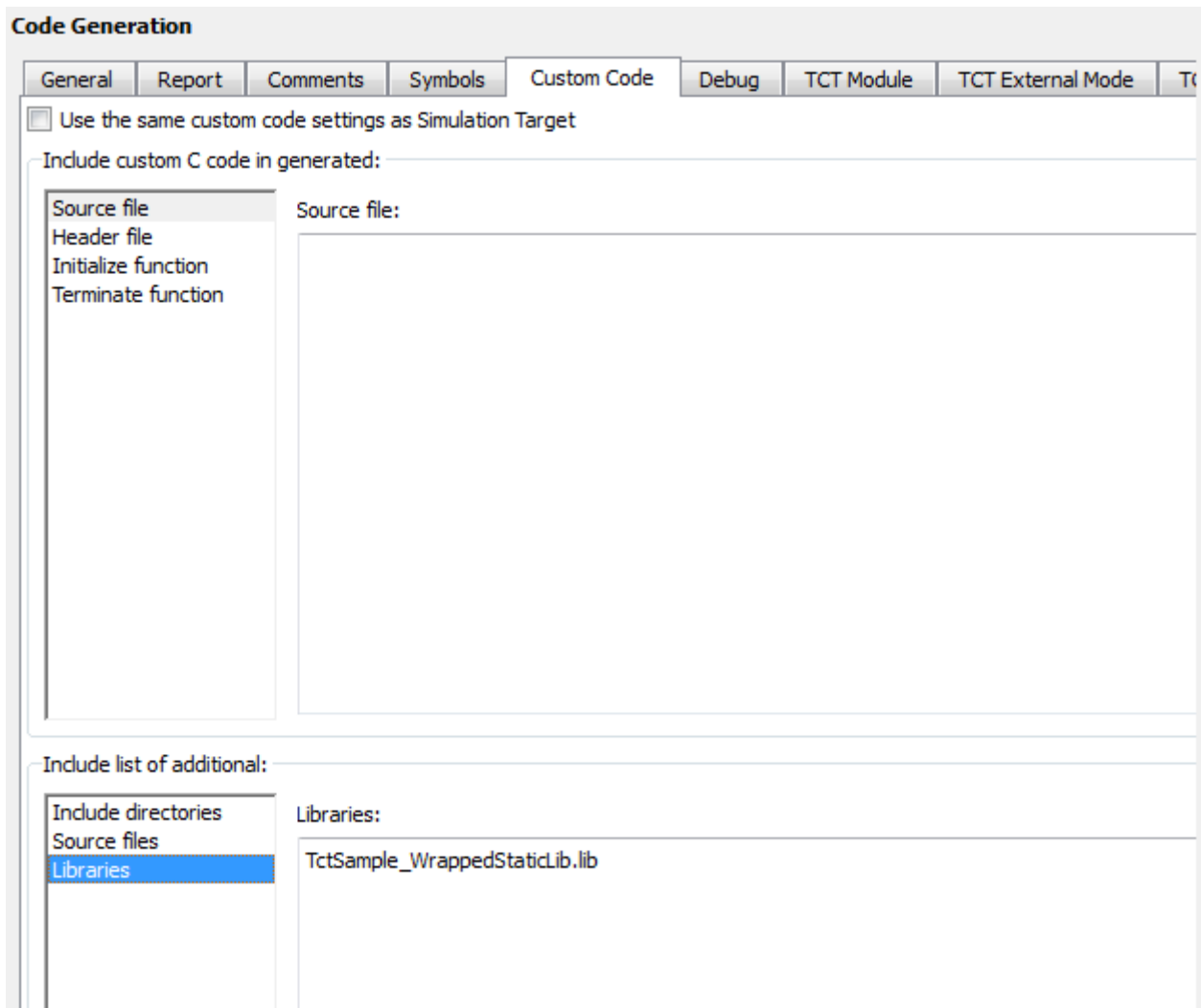
Use the same custom code settings as Simulation Target

Include custom C code in generated:

Source file	Source file:
Header file	
Initialize function	
Terminate function	

Include list of additional:

Include directories	Include directories:
Source files	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)"
Libraries	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Release"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Debug"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Release"
	"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Debug"



The Include folder should already have been created automatically by TwinCAT Target. The Libraries setting must contain the names of the static libraries to be linked.

Background information for these settings:

In this example (and in most other cases) there are several instances of these libraries in the specified folders. MSBUILD decides which version is linked to the module when the generated TwinCAT module binary files are linked.

How to use this example as a template

The following list provides a short overview of the easiest way, to create an own TwinCAT Target compatible SFunction dependency:

1. Copy the sample folder
2. Replace the MDL file by your own
3. Rename the VCXPROJ file to the desired name of your SFunction
4. Copy your source files to the directory where the VCXPROJ file is located
5. Adapt the scripts *BuildLibsAndSFunction.m* and *OpenLibProject.m* to the new project name
6. Open the VCXPROJ file using *OpenLibProject.m*
7. Remove the existing CPP files from the project
8. Add your own CPP files to the project
9. If necessary, add include directories, dependency directories and libraries to the compiler and linker settings
10. Build the project (for different platforms and/or configurations)

11. Close the VCXPROJ file
12. Run *BuildLibsAndSFunction.m*

3.8.5 Module generation Callbacks

Examples for **module generation callbacks** [► 24]:

Example	Topics	Description
Packaging module files into ZIP archives [► 88]	<ul style="list-style-type: none"> • PostPublish callback • Archiving generated module files 	This simple example illustrates the automatic archiving of generated module files.

3.8.5.1 Packaging module files into ZIP archives

Callbacks can be used to store generated module files in as a ZIP archive, for example. First create the directory *C:\MyGeneratedTcComModules*, then copy the following command in the **PostPublish callback** field of the code generator settings of the Simulink model under **Tc Build**:

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

The screenshot shows the 'Tc Build' configuration window. On the left is a tree view with 'Code Generation' expanded and 'Tc Build' selected. The main area contains the following settings:

- TcCom module name: \$<MODELNAME>
- Publish module
- Platform toolset: Auto
- Publish configuration: Debug
- Publish binaries for platform "TwinCAT RT (x86)"
- Publish binaries for platform "TwinCAT UM (x86)"
- Publish binaries for platform "TwinCAT RT (x64)"
- Publish binaries for platform "TwinCAT UM (x64)"
- Lowest compatible TwinCAT build: 4018
- PreCodeGeneration callback: (empty)
- PostCodeGeneration callback: (empty)
- PostPublish callback: `zip(fullfile('C:\MyGeneratedTcComModules', cgStruct.ModuleName), '*', fullfile(getenv('TwinCat3Dir'), 'CustomConfig', 'Modules', cgStruct.ModuleName))`
- Signing Certificate for x64 Windows Loader: \$<TWINCATTESTCERTIFICATE>

4 From version 2.x.xxxx.x

- TE1400 Target for Simulink® versions lower than 1.2.xxxx.x support MATLAB R2010b to MATLAB R2019a.
- TE1400 Target for Simulink® versions higher than 2.x.xxxx.x support MATLAB R2019a and higher.
- The installations for both versions can be used in parallel on one engineering system.
- Compatibility of the created modules: see [Mapping is lost with Reload TMI/TMC \[► 232\]](#).

4.1 Installation

System requirements

In the following, a distinction is made between the engineering PC and the runtime PC. The following definition applies: Simulink® models or MATLAB® functions are converted into TwinCAT objects on the engineering PC by using TwinCAT Target for Simulink® or Target for MATLAB® respectively. Likewise, a TwinCAT solution can, but does not have to, be created on this PC, which uses the created objects. The created TwinCAT solution is then loaded from the engineering PC to a runtime PC in the TwinCAT runtime environment for execution of the project.

On the engineering PC

- MATLAB R2019a or higher
 - Simulink® and Simulink Coder™ Toolbox for using Target for Simulink®
 - MATLAB® and MATLAB Coder™ Toolbox for using Target for MATLAB®
- Visual Studio 2017 or higher (Professional, Ultimate or equivalent edition)
 - During installation, the option *Desktop development with C++* must be selected manually. The option can also be installed later.
 - The Visual Studio version must be supported by the TwinCAT XAE setup, see [here](#).
- TwinCAT 3.1.4024.7 or higher
 - Install TwinCAT 3 XAE or Full Setup only after Visual Studio has been installed with *Desktop development with C++*.
- TwinCAT Tools for MATLAB® and Simulink® setup

On the runtime PC

- Supported operating systems
 - Windows 7, Windows 10, Windows Server (32-bit and 64-bit)
 - TwinCAT/BSD®
- TwinCAT XAR version 3.1.4024.7 or higher

Built objects can be easily forwarded

TwinCAT objects built on an engineering PC (or Build Server) can be easily forwarded to other people. They only need the TwinCAT XAE development environment in order to use the created objects (TcCOM or PLC function blocks) in a TwinCAT solution.

Installation

- ✓ Install one of the supported **Visual Studio** versions, if not already installed. Note the installation of the *Desktop development with C++* option.
1. Start **TwinCAT 3 XAE or Full Setup**, if it is not already present.
If a Visual Studio and a TwinCAT installation are already present but the Visual Studio version does not meet the requirements mentioned above (e.g. TwinCAT XAE Shell or Visual Studio without C++ option), you need to install a suitable Visual Studio version first (install C++ option, if necessary). Then run TwinCAT 3 setup to integrate TwinCAT 3 into the new (or modified) Visual Studio version.
 2. If you have not yet installed **MATLAB®** on your system, install it. The order in which MATLAB® has been installed is irrelevant.

3. Start **TwinCAT Tools for MATLAB® and Simulink®** setup to install the TE1400.

⇒ The TE1400 is installed in the TwinCAT folder structure, i.e. it is separate from the MATLAB® installation.

4. Start MATLAB® as administrator and run

`%TwinCAT3Dir%.. \Functions\TE14xx-ToolsForMatlabAndSimulink\SetupTE14xx.p` in MATLAB®.

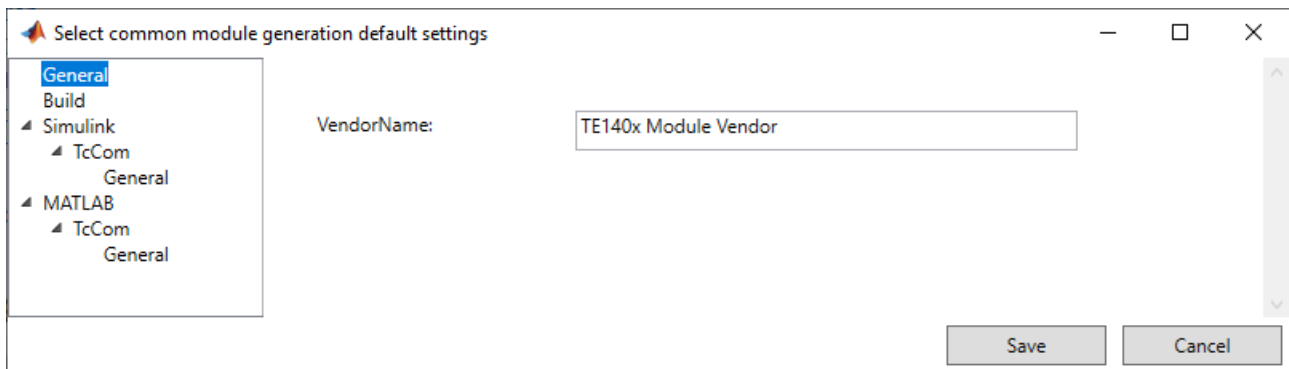
⇒ A setup window opens. See the following section.

Initial setup of the software (Common Settings dialog)

Run SetupTE14xx.p

After executing the p-file (see step 4 under Installation), a dialog opens in which you can save general default settings. These settings then apply system-wide, i.e. for *all* installed MATLAB® versions.

You can make the settings at this time or make/change them at a later time.



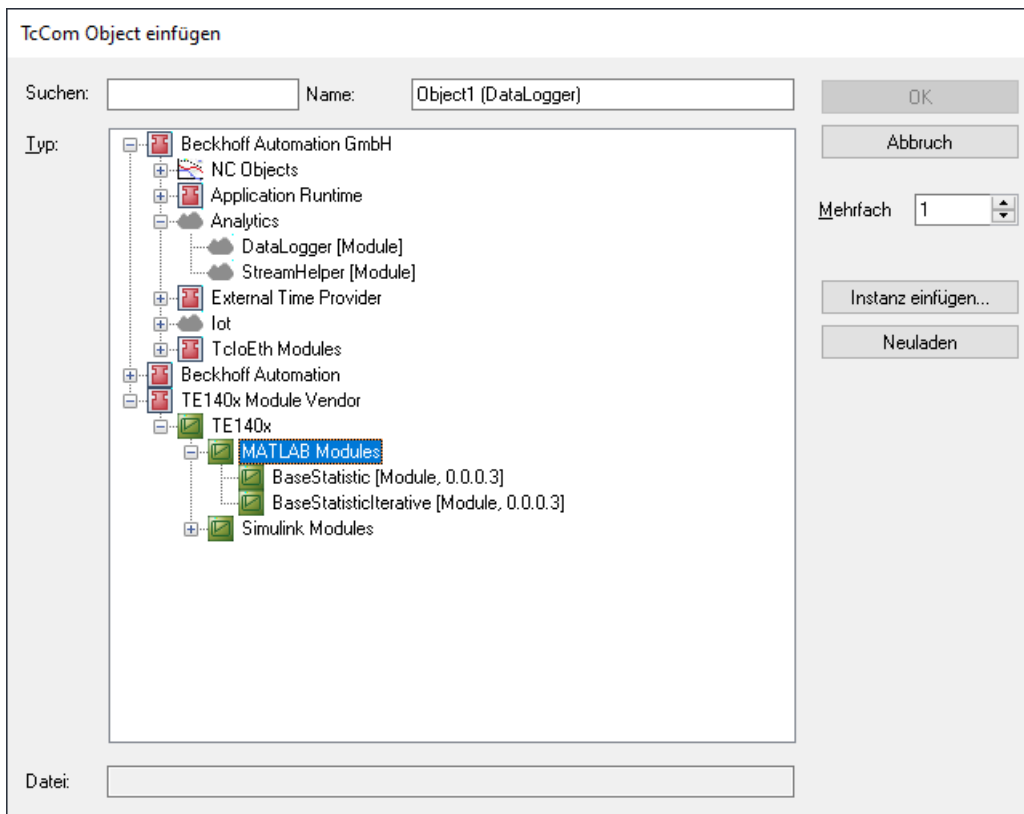
The following setting options are available in the dialog:

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

These settings influence the hierarchy in which the generated TwinCAT objects are sorted. See diagram below.

There you can see the entries:

- VendorName "TE140x Module Vendor"
- GroupName "TE140x"
 - "MATLAB Modules" for MATLAB® and
 - "Simulink Modules" for Simulink®



In the described dialog under the **Build** tab, you can store a default certificate for driver signing. The setup options for driver signing are explained in full in the chapter [Setting up driver signing \[► 93\]](#).



Entering a certificate at this point is optional and not mandatory.

Run SetupTE14xx.p "silent"

If you want to execute the p-file without this dialog, you can use the following command:

```
SetupTE140x('Silent', true);
```

This sets the default values of the dialog.

Change the software setup

To change the default settings of the TwinCAT target, you can access a dialog in the MATLAB® Console as follows:

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

Here you are offered various entries that you can store as default values.

TwinCAT.ModuleGenerator.ProjectExportConfig

General
Build
PLC Library
License
▸ Simulink
▸ MATLAB

Generate TwinCAT C++ Project

TwinCAT C++ Project Path: _____

Lowest compatible TwinCAT version (build number): \$<TwinCAT:Version:BU_____

Class factory name: \$<Project:Name>_____

Product name: \$<ModuleGenerator:ProductId> \$<ModuleGenerator:Ve_____

Copyright notice: Copyright \$<VendorName> \$<LocalDateTime:%Y>_____

Driver description: TwinCAT executable file, generated by TwinCAT \$<Modu_____

Vendor name: TE140x Module Vendor 42_____

Version source file: \$<LatestTMFile>_____

Version part for increment: Revision ▾

Driver file version: \$<VersionFromFile>_____

Driver product version: \$<DrvFileVersion>_____

Code generation placeholders: _____

Load DataExchangeModules

Maximum number of visible array elements: 200U_____

Create unique item names for enumeration types

Data type import TMC files: _____

Save Cancel

Accept changes

1. Enter the new default settings in the dialog box.
 2. Confirm with the **Save** button.
 3. Restart MATLAB®.
- ⇒ The changes have been adopted.

4.2 Licenses

Two licenses are required to use the full functionality of the TE1400 TwinCAT Target for Simulink®. On the one hand, the TE1400 engineering license for creating TwinCAT objects from Simulink® and, on the other hand, a runtime license for executing these objects during the TwinCAT runtime.

Engineering license

The license *TE1400 Target for Simulink®* is required for the **engineering system** for creating TcCOM and PLC function blocks from Simulink®. For testing purposes, the product can be used in demo mode without a license as a demo version.



A 7-day trial license with full functionality is not available for this product.

Restrictions in the demo version

Without valid TE1400 license models are allowed with maximum:

- All cpp and header files from Simulink Coder™ (incl. dependent libraries) must not exceed 100 kB in total.
- 5 input signals
- 5 output signals



Modules created with a demo license may only be used for non-commercial purposes!

Runtime license

A runtime license is required to start a TwinCAT configuration containing one or more TwinCAT objects generated from Simulink®.

Required licenses for execution are:

TF1400 TwinCAT 3 Runtime for MATLAB®/Simulink® **and** TC1300 TwinCAT 3 C++. Both licenses are included in the TC1320 and TC1220 **license bundles**. TC1320 bundles TC1300 and TF1400 licenses (C++ and MATLAB®/Simulink® Runtime). TC1220 bundles the TC1200, TC1300 and TF1400 licenses (PLC, C++ and MATLAB®/Simulink® Runtime).

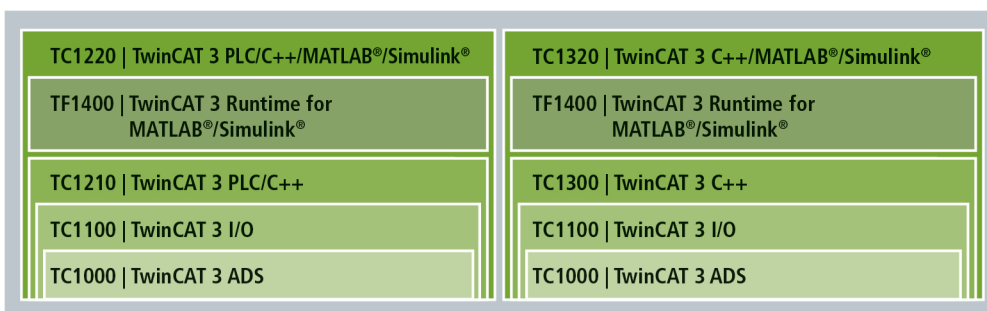


License TF1400 License only from TwinCAT 3.1.4026.0

The TF1400 license is only supported from TwinCAT 3.1.4026.0. For earlier versions only the license bundles TC1220 or TC1320 are available. The license bundles are still supported with build 4026 and higher. The TF1400 license is only required if the TwinCAT 3 C++ license is already available on the target system via a license.

Without activated license, the module and consequently the TwinCAT system cannot be started.

You can generate a 7-day trial license for the named runtime licenses, which allows initial testing without purchasing the license.



4.3 Setting up driver signing

Create an OEM certificate level 2

TwinCAT objects generated from MATLAB® or Simulink® are based on a tmx driver (TwinCAT Module Executable), as are TwinCAT C++ objects. These drivers must be signed with an OEM certificate level 2, so that it can be loaded on the runtime PC during the TwinCAT runtime.

See the following links for detailed documentation on how to create an OEM certificate for driver signing.

- [General documentation on OEM certificates](#)

- [Application-related documentation for tmx driver signing](#)

The most important facts in brief:

- You can create your own certificate. To do this, go to Visual Studio at: **Menu bar > TwinCAT > Software Protection...**
- You need an OEM certificate Crypto Version 2 (option: *Sign TwinCAT C++ executables (*.tmx)*).
- All drivers (for 32-bit and for 64-bit operating systems) must be signed.
- Drivers can also be created without signing and signed afterwards.
- For testing purposes in the development phase, a non-countersigned certificate is sufficient.
- Countersigned certificates can be ordered free of charge from Beckhoff (TC0008).

Use of an OEM level 2 certificate for driver signing

To sign tmx drivers, you need a certificate and a password associated with the certificate.

Handling of the certificate

There are four possible variants for signing tmx drivers.

Variant 1: System-wide default certificate for TwinCAT C++ and TE14xx

You can set a default certificate on an engineering PC, which is always used for TwinCAT C++, Target for MATLAB® and Target for Simulink®, unless you explicitly specify a different certificate.

For this variant, you use a Windows environment variable. Create a new **environment variable** at **User > Variables** with:

Variable: *TcSignTwinCatCertName*

Value: Name of the desired certificate



Available certificates can be found at *TwinCAT\3.1\CustomConfig\Certificates*.

Variant 2: System-wide default certificate for TE14xx

You can set a default certificate in your MATLAB® environment, which is always used for Target for MATLAB® and Target for Simulink® (**not** TwinCAT C++), unless you explicitly specify a different certificate.

Open the [Common Settings dialog](#) [► 90] mentioned above with *TwinCAT.ModuleGenerator.Settings.Edit* and enter the desired default certificate under **Build> Certificate name for TwinCAT signing**. This certificate is stored in your user directory as default and is used by all MATLAB® versions on your system as default.

Variant 3: Certificate in the configuration of the Simulink® model

You can explicitly name a certificate for each build operation. For Variant 3 you do not have to make any further settings in advance. Before each build process, you can define a certificate of your choice for precisely this build process.

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

Target for MATLAB®: Property *SignTwinCatCertName*

Variant 4: Build without certificate and sign later with TcSignTool

You can build without a certificate and sign afterwards with the **TcSignTool**. For **Variant 4** you can use the TcSignTool.

The TcSignTool is a command line program located in the path `..\TwinCAT\3.1\sdk\Bin\`. For example, open the command prompt and execute `tcsigntool sign /?` to display the help.



Operating TcSignTool from MATLAB®

From MATLAB®, the tool can be started with the command `system()` or with `!`.

```

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

Sign files with certificate:

TcSignTool.exe sign /f certificatefile [/p password]/g) [/q] file1 [file2 ...]

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

/q quiet mode

returns 0 = succeeded, 1 = failed

ans =

    1

fx >> |
    
```

Sample call for signing a tmx driver for TwinCAT:

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

Handling the certificate password

The password is only entered directly at *Variant 4* of the certificate handling. For *Variants 1 to 3*, the associated password must be entered in addition to the certificate. For security reasons, the password should not be entered in the source code in the Simulink® model or in the MATLAB® code. With the TcSignTool you can store obfuscated passwords belonging to your certificates in the registry of the Windows operating system. This means that the password for a specific certificate is known in the operating system (for the corresponding user) and is used automatically.

The password is stored with the following call:

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

The obfuscated password is stored in the registry under:

```
HKEY_CURRENT_USER\SOFTWARE\Beckhoff\TcSignTool\
```

The password is deleted with the following call:

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

Behavior of the TwinCAT runtime

If a TwinCAT object created from MATLAB® or Simulink® with a signed driver is used in a TwinCAT Solution and loaded onto a target system with **Activate Configuration**, the following must be observed:

Each TwinCAT runtime (XAR) has its own white list of trusted certificates. If the certificate used for signing is not included in this white list, the driver will not be loaded. A corresponding error message is output in TwinCAT Engineering (XAE).

```

7/2/2021 12:31:12 PM 466 ms | 'Port_851' (851): Could not link external function
<extref>FB_BASESTATISTICITERATIVE_GUID_960333F6_F2EA_1737_9AD1_D861CDB4708A_MAIN</extref>
7/2/2021 12:31:12 PM 466 ms | 'Port_851' (851): Could not link external function
<extref>FB_BASESTATISTICITERATIVE_GUID_960333F6_F2EA_1737_9AD1_D861CDB4708A_FB_INIT</extref>
7/2/2021 12:31:12 PM 288 ms | 'TCOM Server' (10): OEM 'MyTestCert' - certificate currently not trusted. Import 'C:\TwinCAT\3.1\Target\OemCertificates\..._b86f70ff-06d7-7c09-b29a-da6a4a26d400.reg' to add OEM to trusted list
7/2/2021 12:31:12 PM 466 ms | 'Port_851' (851): Could not link external function
<extref>FB_BASESTATISTICITERATIVE_GUID_960333F6_F2EA_1737_9AD1_D861CDB4708A_FB_EXIT</extref>
7/2/2021 12:31:12 PM 288 ms | 'TCOM Server' (10): Loading 'C:\TwinCAT\3.1\Boot\Repository\TE140x Module Vendor\Tc3_BaseStatistics\0.0.0.1\Tc3_BaseStatistics.tmx' failed
    
```

The error message contains the instruction to execute a registry file, which was automatically created on the target system, on the target system as administrator. This process adds the used certificate to the white list.

● Registry file is only dependent on the OEM certificate

i The registry file can also be used on other target systems. It only contains information about the OEM certificate used and is not target system dependent.

If you use a non-countersigned OEM certificate for signing, you must also put your target system into test mode. To do this, run the following command as an administrator on the target system:

```
bcdedit /set testsigning yes
```

If you are using a countersigned OEM certificate, this step is not necessary.

4.3.1 User certificates for delivery without test mode

● System requirements

i - Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 or TwinCAT/BSD (on the target system)

With TwinCAT Build 4024, Beckhoff offers existing customers the issuing of a "TwinCAT 3 OEM user certificate", which can be used for signing TMX files created with TwinCAT 3 in C++.

- This certificate requires secure validation of the applicant data, since it is used in the Windows environment. TwinCAT 3 user certificates must therefore be officially ordered for validation of the address and contact data, and are only issued to existing Beckhoff customers.
- Order number: **TC0008**
- The issuing of this TwinCAT 3 user certificate is free of charge.
- Directory for saving the certificate: **C:\TwinCAT\3.1\CustomConfig\Certificates**

● The TwinCAT 3 user certificate is not required for using the TwinCAT 3 TMX files

i The TwinCAT 3 user certificate is used exclusively for the one-time signing of the TMX files and is not required for the use of the TMX files signed with it.

● On which computers is the TwinCAT 3 user certificate TC0008 required?

i The TwinCAT 3 user certificate should be located exclusively on the engineering computer on which the TMX files are signed - i.e. **NOT** on each target system.

Validity of the TwinCAT 3 user certificate

The validity of the TwinCAT 3 user certificate is limited to two years for security reasons.

● What happens if the certificate has expired?

i You can no longer sign new TMX files.
However, the use of already signed TMX files is still possible without any restrictions.

You can apply for a renewal of your certificate before the expiry of the two years (and even after that).

To extend a TwinCAT 3 user certificate, the same process applies as for requesting a new certificate. In this case, the certificate must also be ordered (the order numbers for a certificate extension are the same as for a new certificate request).

In contrast to a new certificate, you do not generate a new OEM Certificate Request File but send your existing certificate to the Beckhoff certificate section for renewal. Please notify us in the email that this is a certificate extension and not a new issue. Otherwise, the same criteria apply regarding the content of the email as for the application for a new certificate.

The existing certificate receives a new expiration date, is then re-signed and is valid for another 2 years.

The newly signed certificate is thus fully compatible with the original version.

4.3.1.1 Request TwinCAT 3 user certificate

Overview of the ordering and validation process

An official order is required to request a TwinCAT user certificate.

- Order number: **TC0008** (TwinCAT 3 Certificate Extended Validation)
- The issuing (and renewal) of a TwinCAT 3 user certificate is free of charge.
- Since a TwinCAT 3 user certificate is a digital ID card, verification of the inquirer's contact data is required according to the usual market standards.
- A TwinCAT 3 user certificate is therefore only issued to existing Beckhoff customers.

Overview of the ordering and validation process



Your email address must be a company email account (freemailers such as GMail or similar are not permitted) and correspond with the company name of the inquirer.

1. Contact your Beckhoff sales contact and announce the request of a TwinCAT 3 OEM certificate. Order "TC0007" or "TC0008".
2. Important: as the inquirer, please provide your contact details as the delivery address (= contact name and email address) and the area of use of the certificate (company name, address).
3. The contact details provided in the order will be verified and you (the inquirer named in the delivery address) will be contacted by Beckhoff Sales.
4. When requesting a new OEM certificate, [Creation of the Certificate Request file for TC0008 \[▶ 97\]](#).
5. Determine the "File Fingerprint" of the OEM certificate file using TwinCAT Engineering (see [Determining the file fingerprint of the OEM certificate file \[▶ 100\]](#)). Please inform the Beckhoff sales contact of this File Fingerprint as part of your contact data verification. The transmission of the File Fingerprint must be done by a different communication channel than the one used for sending the OEM certificate request file.
6. Now send the "OEM certificate file" to the Beckhoff sales contact.
7. After signing the certificate file at the Beckhoff headquarters, you will receive it by e-mail from your contact person.

Please note that it may take a few days to validate your contact details and issue the certificate.

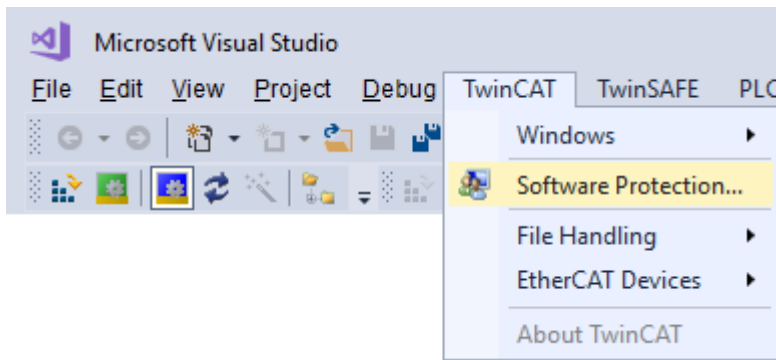
4.3.1.2 Creation of the Certificate Request file for TC0008



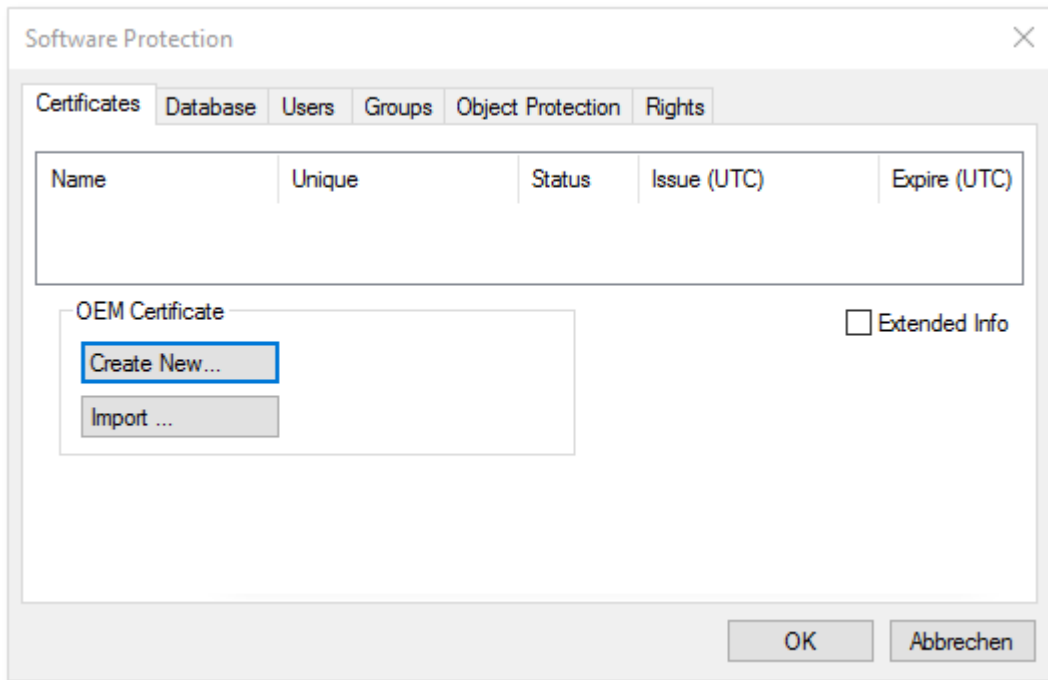
System requirements

- Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 or TwinCAT/BSD (on the target system)

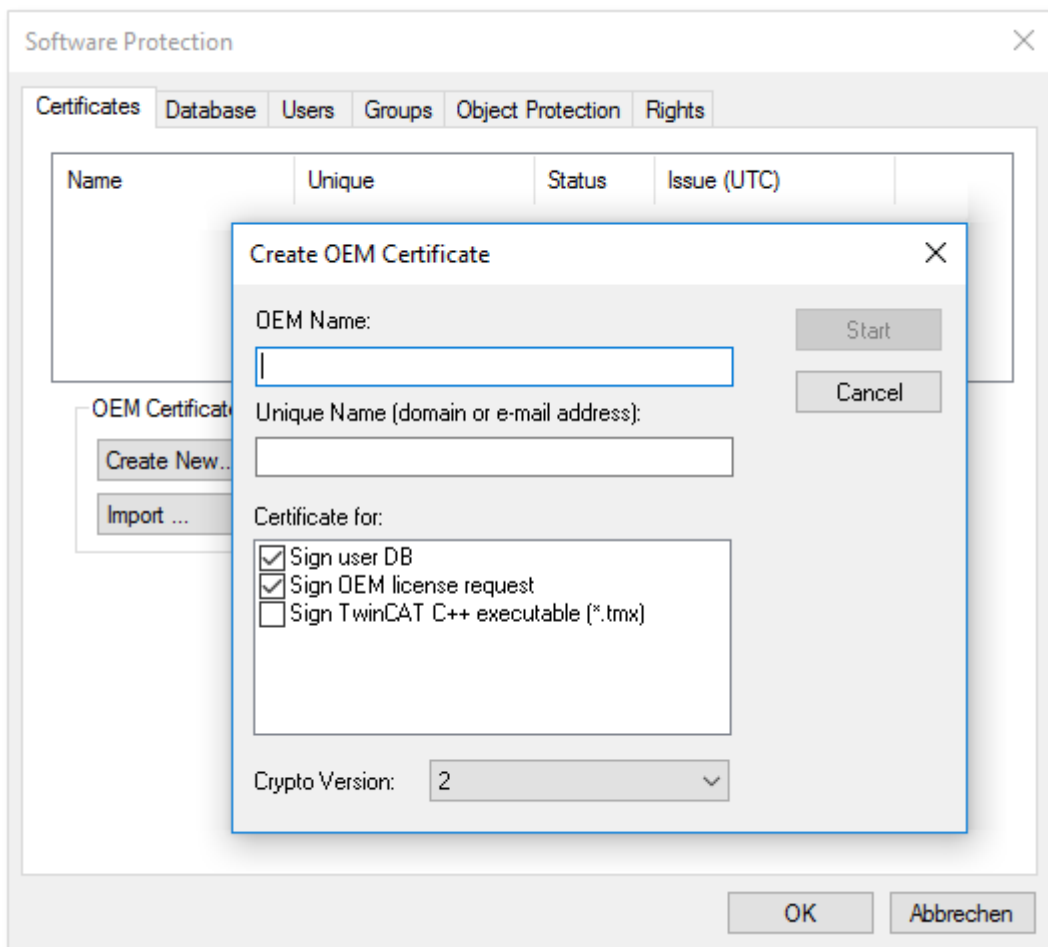
1. Call up the Software Protection configurator. To do this, select the menu item **Software Protection** in the main menu below the item **TwinCAT**:



- In the window that opens, select the **Certificates** tab.
Click **Create New....**:

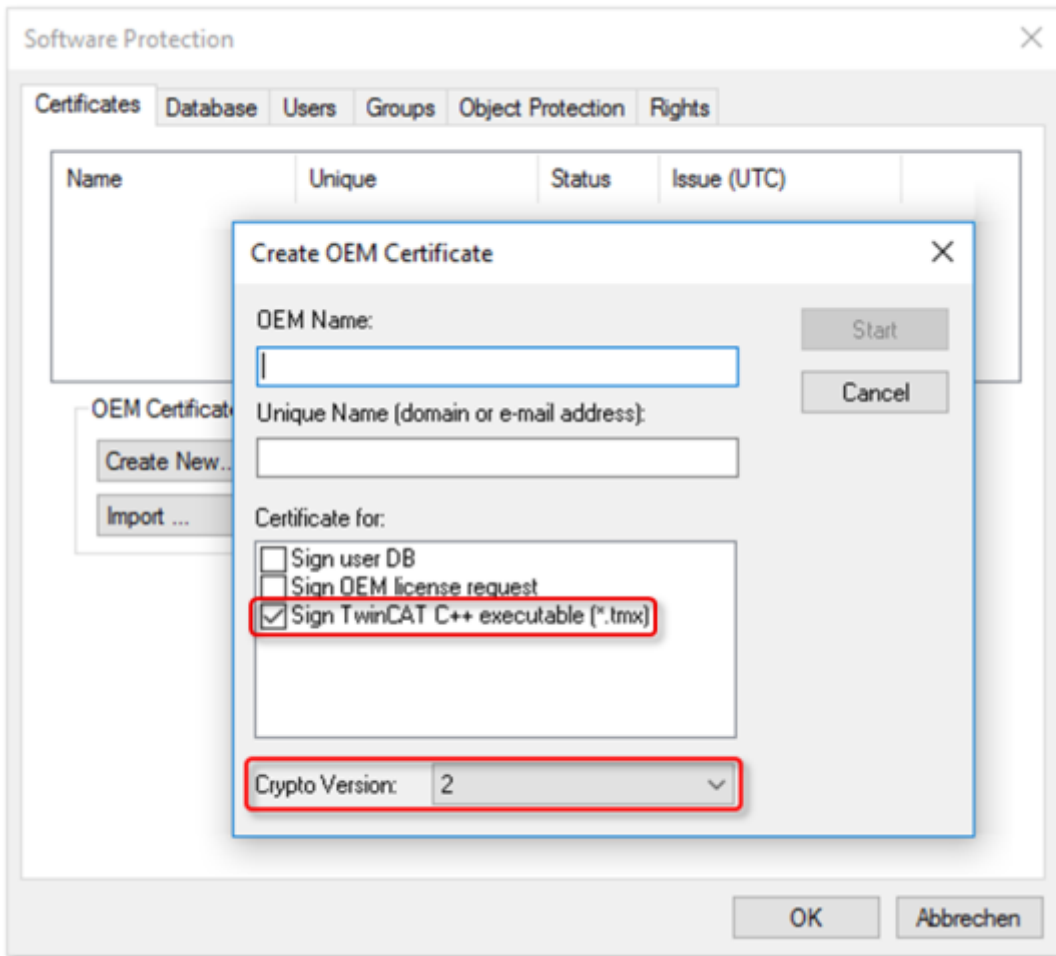


- The **Create OEM Certificate** input window opens:



- Enter the required data:
 - Enter your company name in the **OEM Name** text field. The name must have a clear reference to your company or your business unit.

- Enter a **Unique Name**. The "OEM Unique Name" must be a unique name that uniquely identifies the owner of the certificate worldwide, preferably the URL of your company's website or your email address. The email address must be a company email address, i.e. it must be possible to assign it unambiguously to your company.
- Check the checkbox **Sign TwinCAT C++ executables**:



If you only want to sign TwinCAT driver software with this certificate, uncheck the other two checkboxes. (These are only used in the PLC area)

- Make sure that Crypto version 2 (for the encrypted content of the certificate content) is set. (standard setting)
5. Once you have entered the data, click **Start** and select a directory to save the file. **You can simply accept the suggested directory "c:\twincat\3.1\customconfig\certificates". You need the newly created file in this directory in order to be able to read out the "File Fingerprint" for this file [▶_100] in a subsequent step.**
- ⇒ A dialog for selecting a password for the OEM Private Key opens.
6. Issue a password for the OEM Private Key.

● Important: Password security!



Be sure to use a strong password for your certificate!
Protect your password with suitable measures so that it cannot fall into unauthorized hands!

● Password cannot be restored if lost



Beckhoff is unable to recover or reset your password. If you forget or lose the password for your certificate, you can no longer use it and have to request a new certificate.

7. Confirm the password by entering it again and close the dialog with **OK**.

⇒ The file is saved.

The "Certificate Request File" generated in this way must now be signed by the Beckhoff certificate section in order to be valid. The procedure is described in chapter [Requesting a certificate](#) [▶ 97].

4.3.1.3 Determining the file fingerprint of the OEM certificate file

You need this functionality to request a **TwinCAT OEM Certificate Extended Validation (TC0008)**.

i System requirements

This functionality requires TwinCAT 3.1 Build 4024 of higher.

i The OEM Certificate Request File becomes the TwinCAT OEM certificate once it is signed by Beckhoff. The files do not differ except for this signature. For this reason, the term "TwinCAT OEM certificate file" is used for both file versions in the following sections.

Reading the "file fingerprint" of an OEM certificate file via TwinCAT 3 Engineering

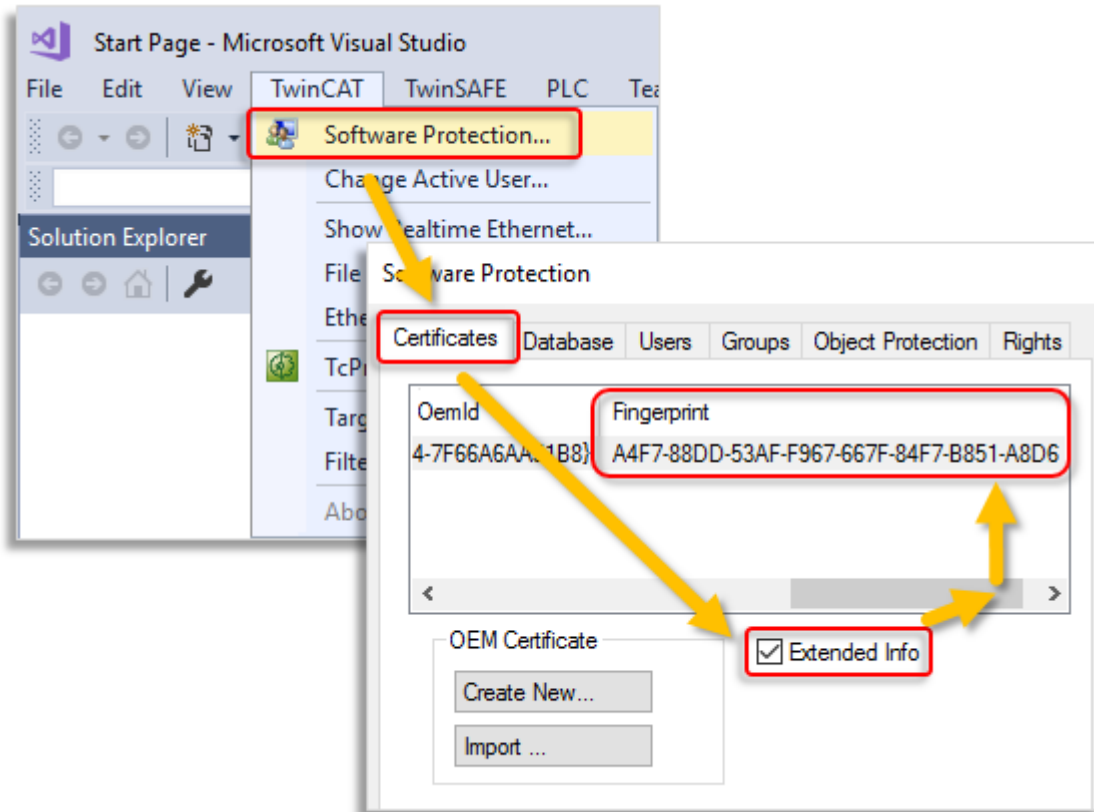
For this function it is necessary that the OEM certificate file is located in this directory: "c:\twincat\3.1\customconfig\certificates".

This directory contains your OEM certificate, if you already have a certificate and want to renew it.

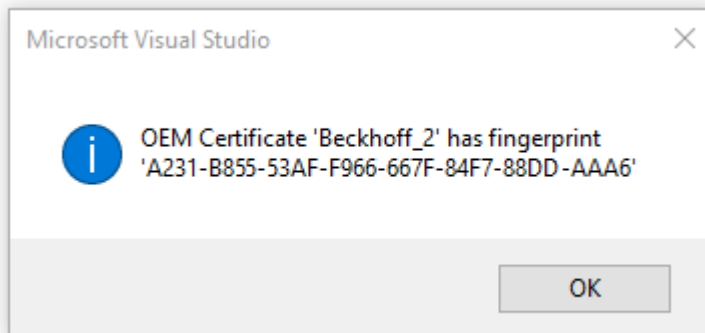
If you did not change the suggested directory when creating the "OEM Certificate Request File", the file is already in this directory.

Procedure:

1. Call up the TwinCAT 3 Software Protection configurator.



2. Select the **Certificates** tab.
3. Check the **Extended Info** checkbox.
4. In the window scroll to the right until you see the **Fingerprint** column. (As an alternative, you can simply double-click the certificate line. The Fingerprint file is then displayed in a pop-up window:



The shortcut [Ctrl] + [C] can be used to copy the fingerprint data from the message window to the Windows clipboard.

4.3.1.4 Saving the signed TwinCAT user certificate

Recommended directory for saving the certificate: **C:\TwinCAT\3.1\CustomConfig\Certificates**



System requirements

- Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 or TwinCAT/BSD (on the target system)

● The TwinCAT 3 user certificate is not required for using the TwinCAT 3 TMX files

i The TwinCAT 3 user certificate is used exclusively for the one-time signing of the TMX files and is not required for the use of the TMX files signed with it.

● On which computers is the TwinCAT 3 user certificate TC0008 required?

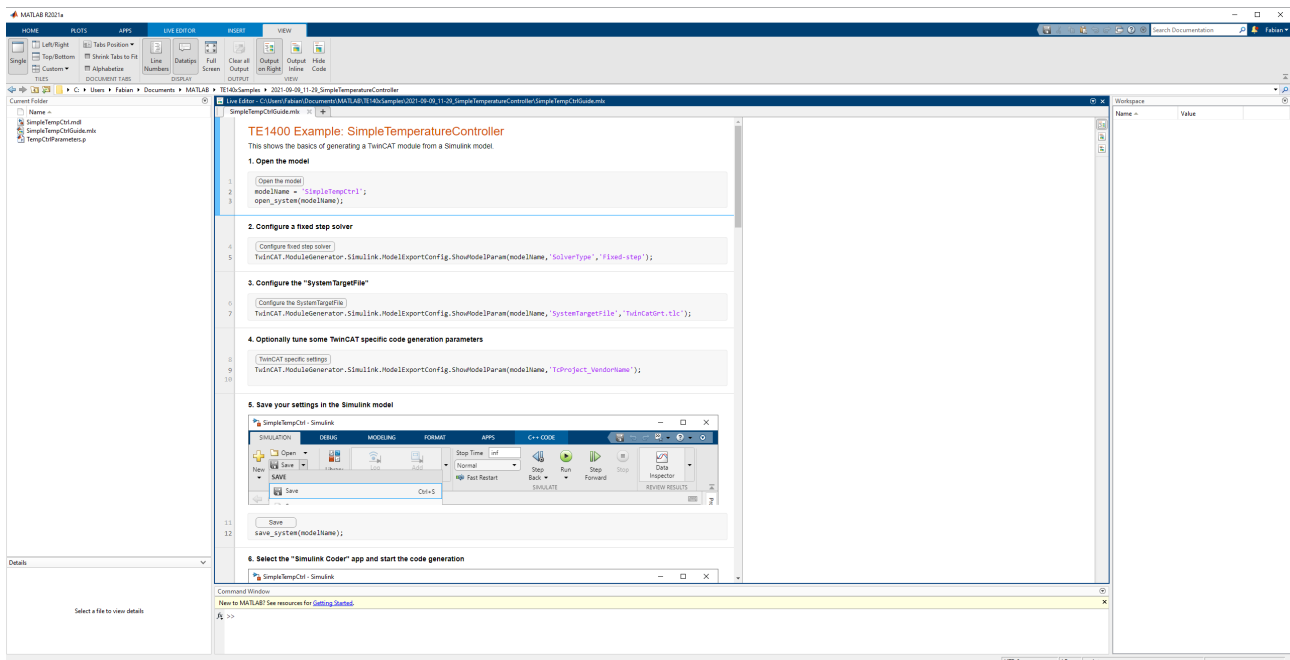
i The TwinCAT 3 user certificate should be located exclusively on the engineering computer on which the TMX files are signed - i.e. **NOT** on each target system.

4.4 Quick start

Starting with a simple Simulink® model

✓ Feel free to use our built-in samples for first steps with the TwinCAT Target for Simulink®. The MATLAB® Command Window provides a list of available samples via `TwinCAT.ModuleGenerator.Samples.List`

1. For example, select the `SimpleTemperatureController` and start the sample using the Start link in the Command Window.



⇒ In the following, the Quick start is executed along this sample.

2. Start at the beginning by selecting the button **Open the model** in the Live Script.

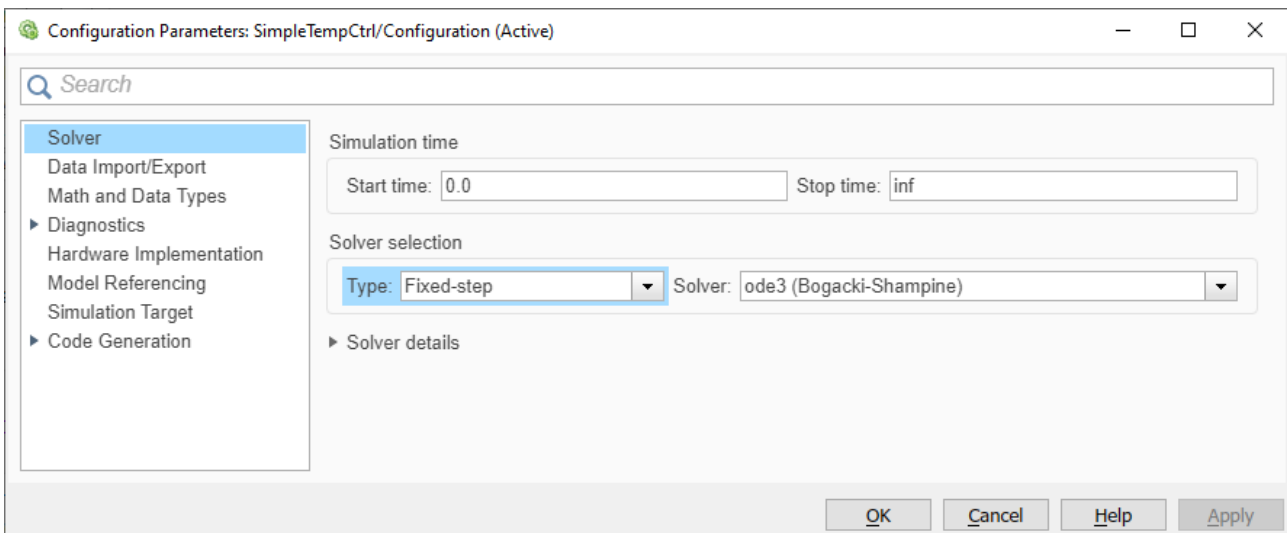
⇒ For the further configuration steps in Simulink® you can simply click the next button in the Live Script.

● Beginner video

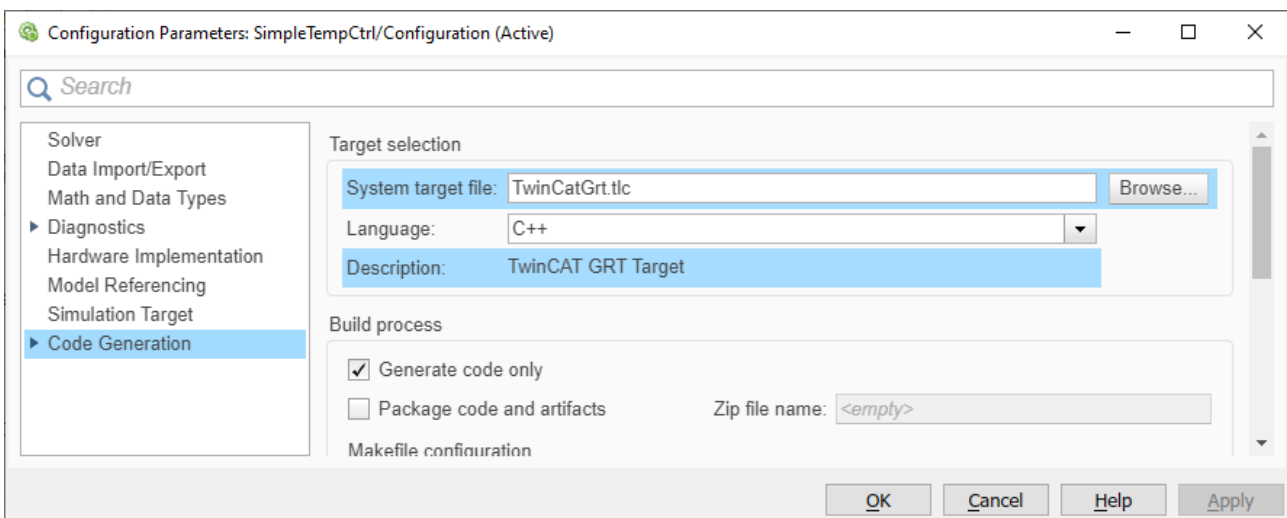
i The following video (only available in English) can also be used as an introduction: [TwinCAT Target for Simulink®](#)

The configuration steps in Simulink®

1. Select a fixed-step solver. To do this, go to the **Configuration Parameters** of the model.

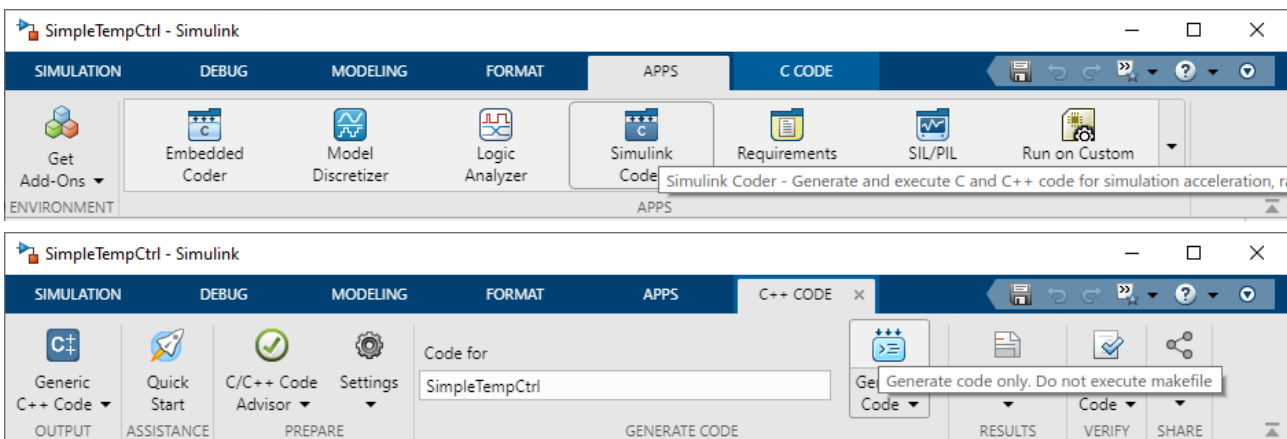


2. Select the system target file to "TwinCatGrt.tlc".



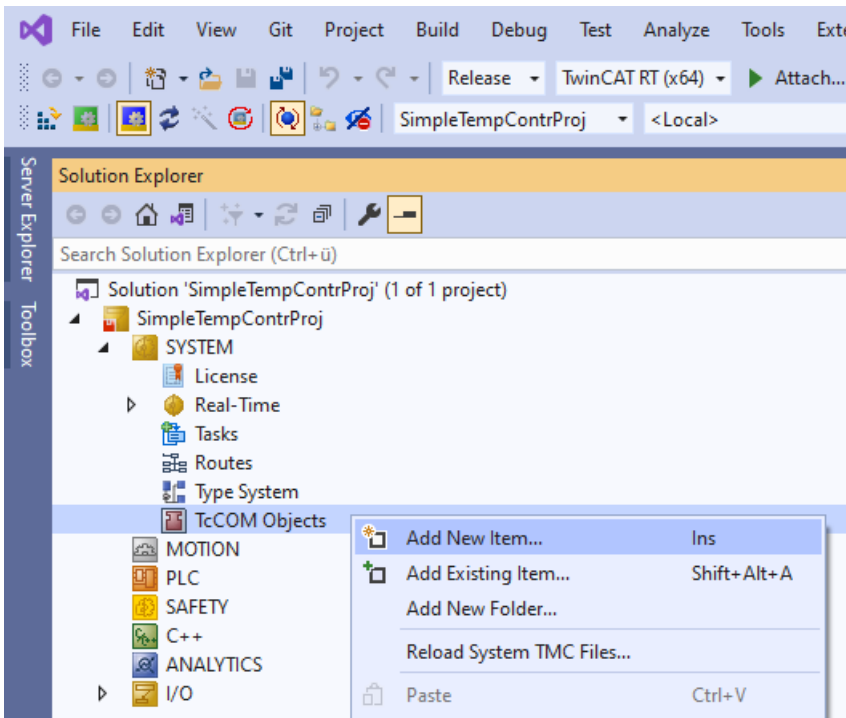
Optional: Under *Optimization*, set the parameter *Default parameter behavior* to Tunable so that you can continue to change model parameters in TwinCAT. See also [Parameterization of a module instance](#) [► 185].

3. Save your changes in the Simulink® model.
4. Start code generation via the Simulink Coder™ App.

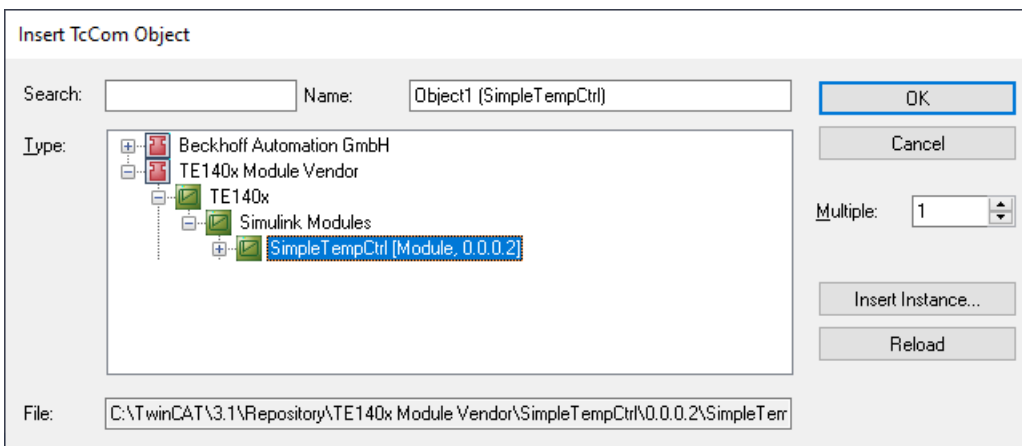


Insert TcCOM in TwinCAT

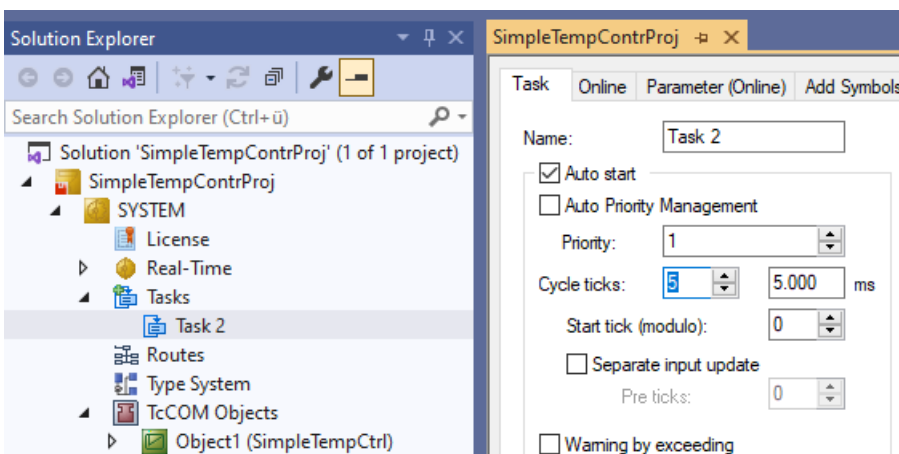
1. Open TwinCAT (TwinCAT XAE or TwinCAT in a Visual Studio environment).
2. Instantiate a new TcCOM object.



3. Select the desired object.

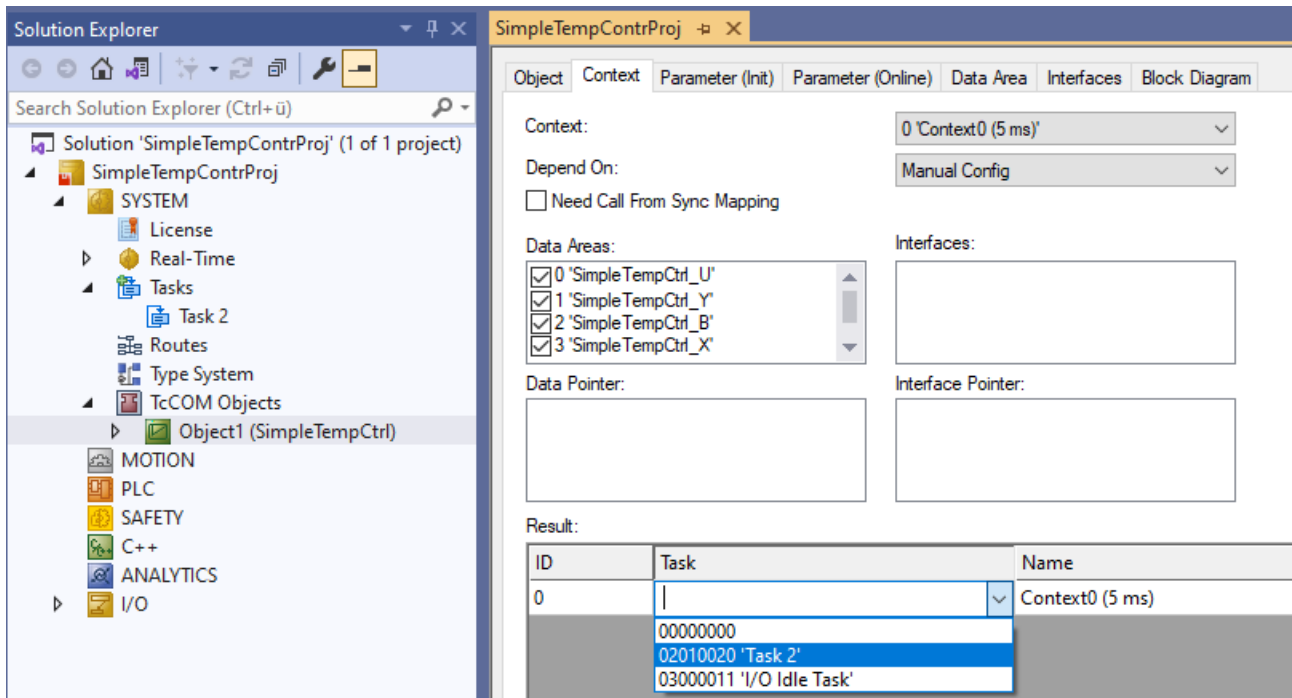


4. Create a cyclic task.

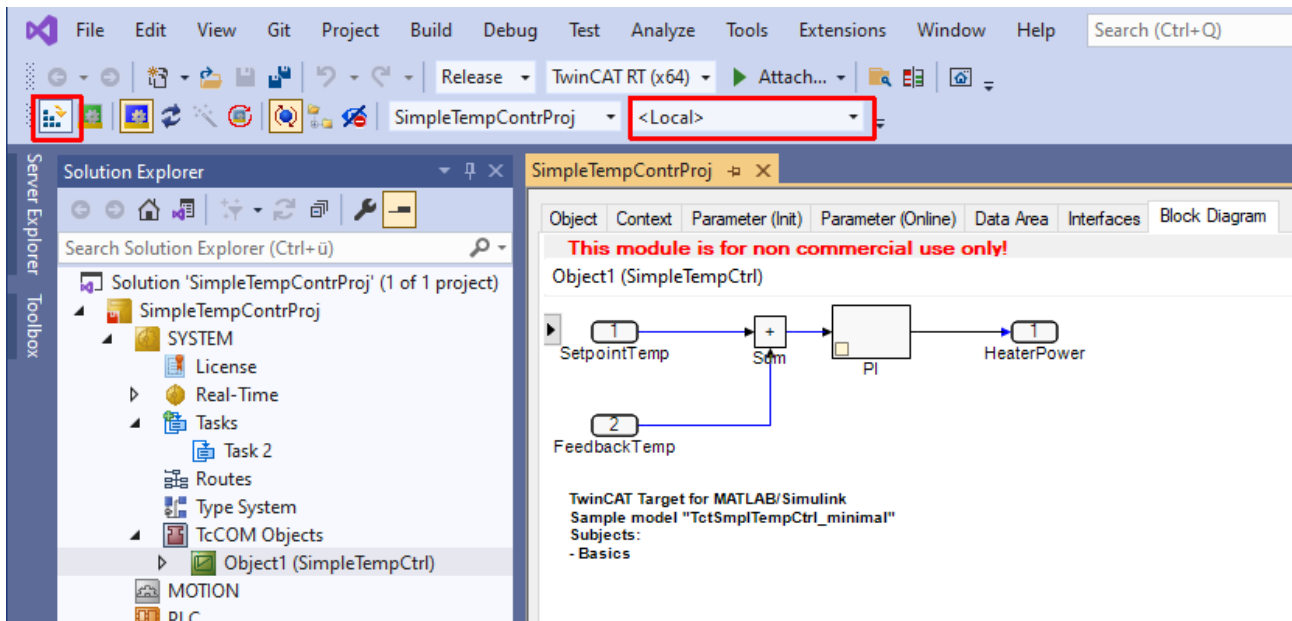


5. Assign the created task to your TcCOM instance.

Note that the cycle time of the task and the SampleTime in Simulink® (here 5 ms) match.



6. Activate the configuration.



Configure and link TcCOM instance

The data exchange of the TcCOM instance takes place via mappings of the process image. Simulink® inputs and Simulink® outputs are automatically mapped as inputs or outputs in the process image and can be linked to I/O or other objects.

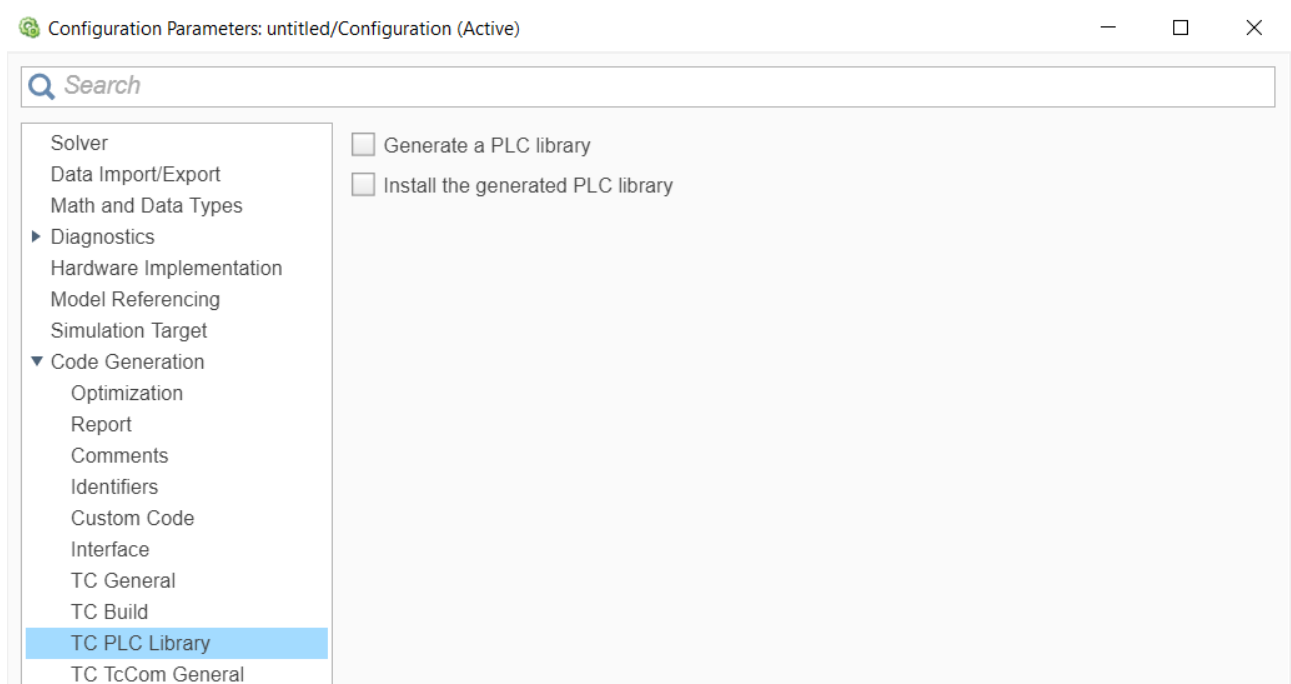
In the Parameters (Init) area of the TcCOM instance, you can optionally configure the instance differently than specified when it was created from Simulink®.

Name	Wert	CS	Typ	PTCID
ModuleCaller	CyclicTask	✓	TcMgSdk.ModuleC...	0x0000...
StepSizeAdaptation	RequireMatchingTaskCycleTime	✓	TcMgSdk.StepSize...	0x0000...
Execute	TRUE	✓	BOOL	0x0000...
SimpleTempCtrl_P_Sharing	Define	✓	ParameterSharingT...	0x0000...
- SimpleTempCtrl_P	...	✓		0x0000...
.Kp	50.0		LREAL	
.Tn	200.0		LREAL	

For example, you can set the parameter Kp to "52" here. The TcCOM module would then use this value as the startup value for this instance.

Insert as PLC function block

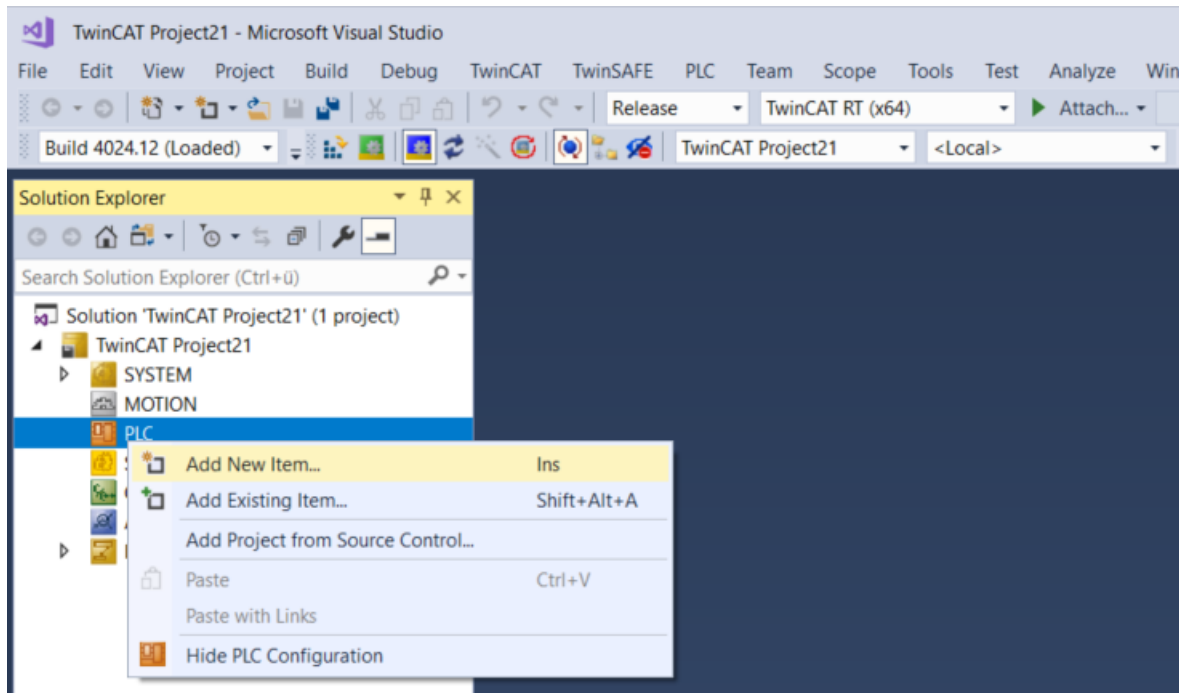
The PLC library used in the following is only available if the following parameters are set in Simulink®:



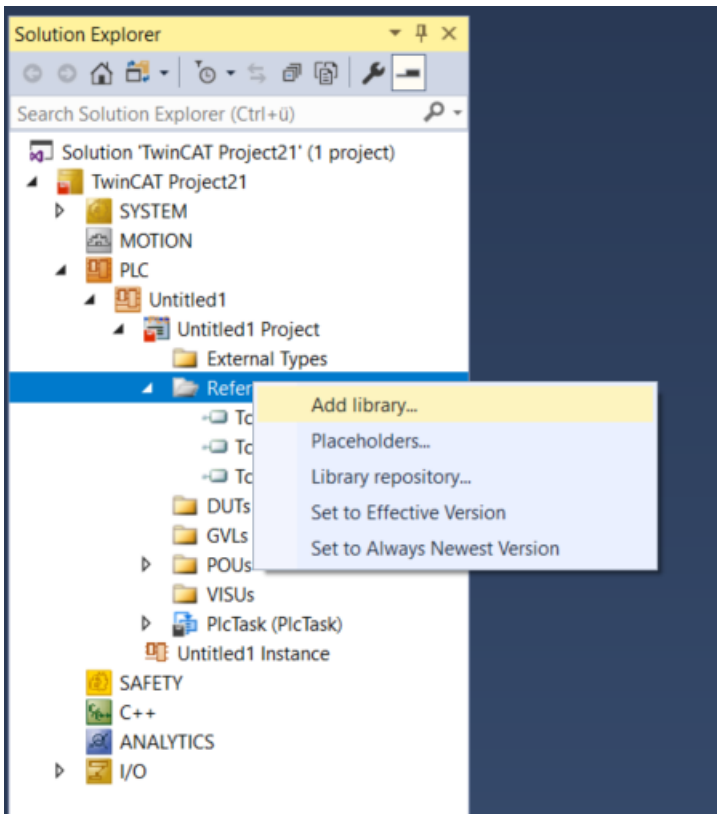
If these options have not been set, this can be done subsequently without using Simulink® and the TwinCAT Target for Simulink®. See [Create and install PLC library \[► 206\]](#).

Brief overview of action steps PLC library/function block

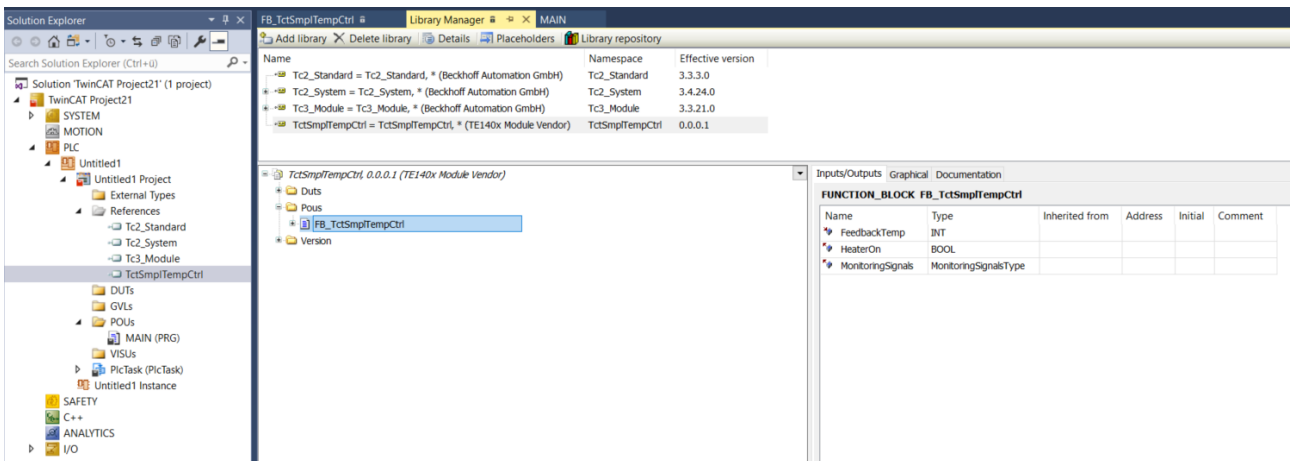
- Create PLC in TwinCAT:



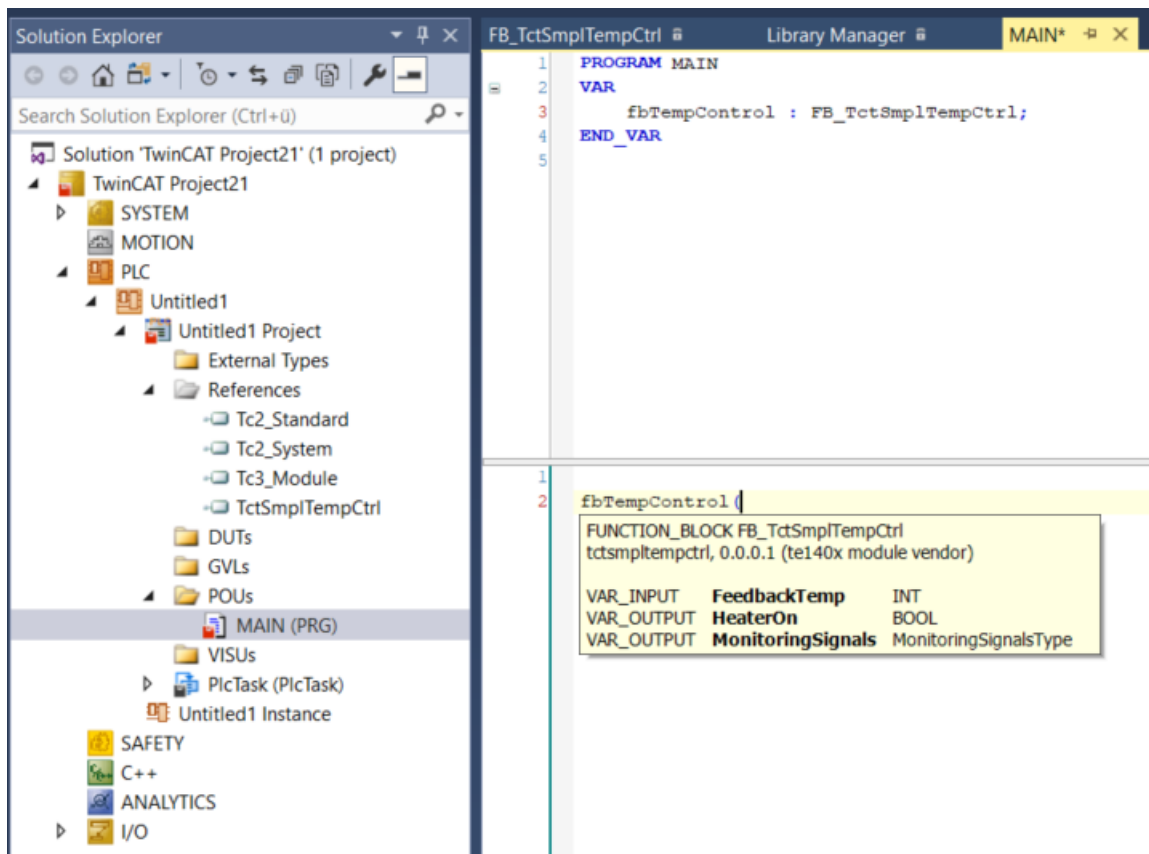
- Add PLC library:



- Select PLC library and view content:



- Use the function block from the library in the PLC:

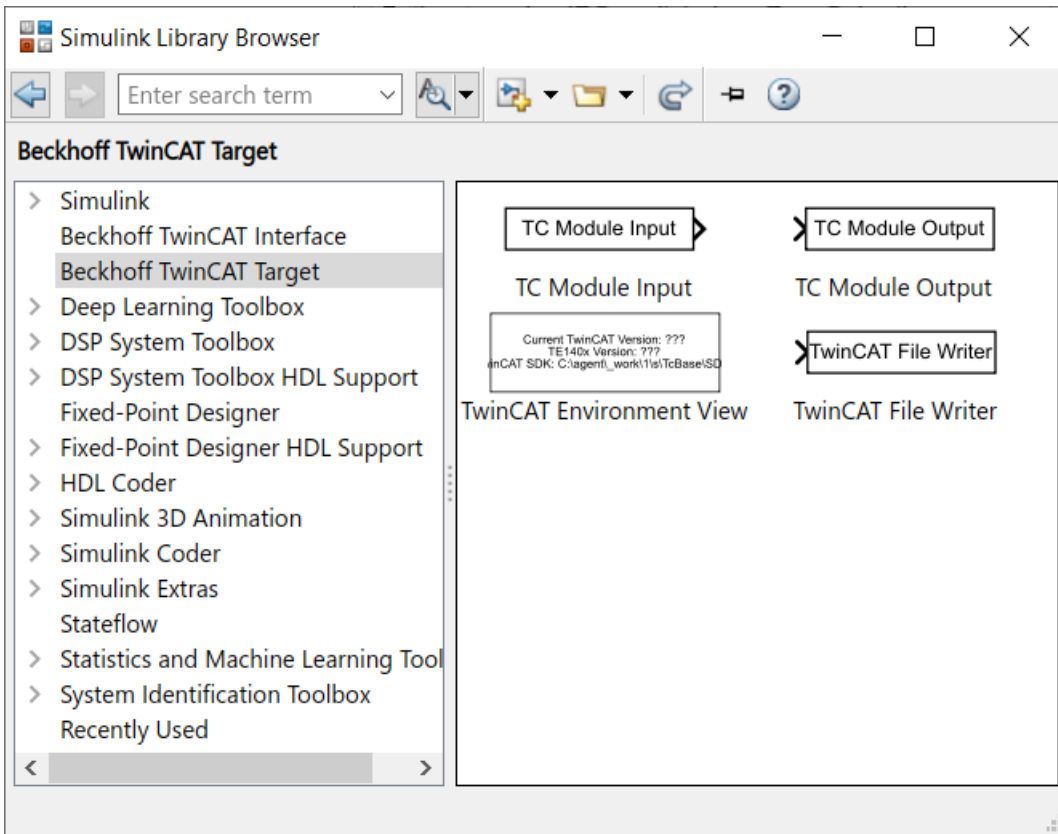


● Calling a TcCOM object also possible from the PLC

i In addition to the variant described here, you can also call an instance of a TcCOM from the PLC. See [Applying the TcCOM Wrapper FB \[► 209\]](#).

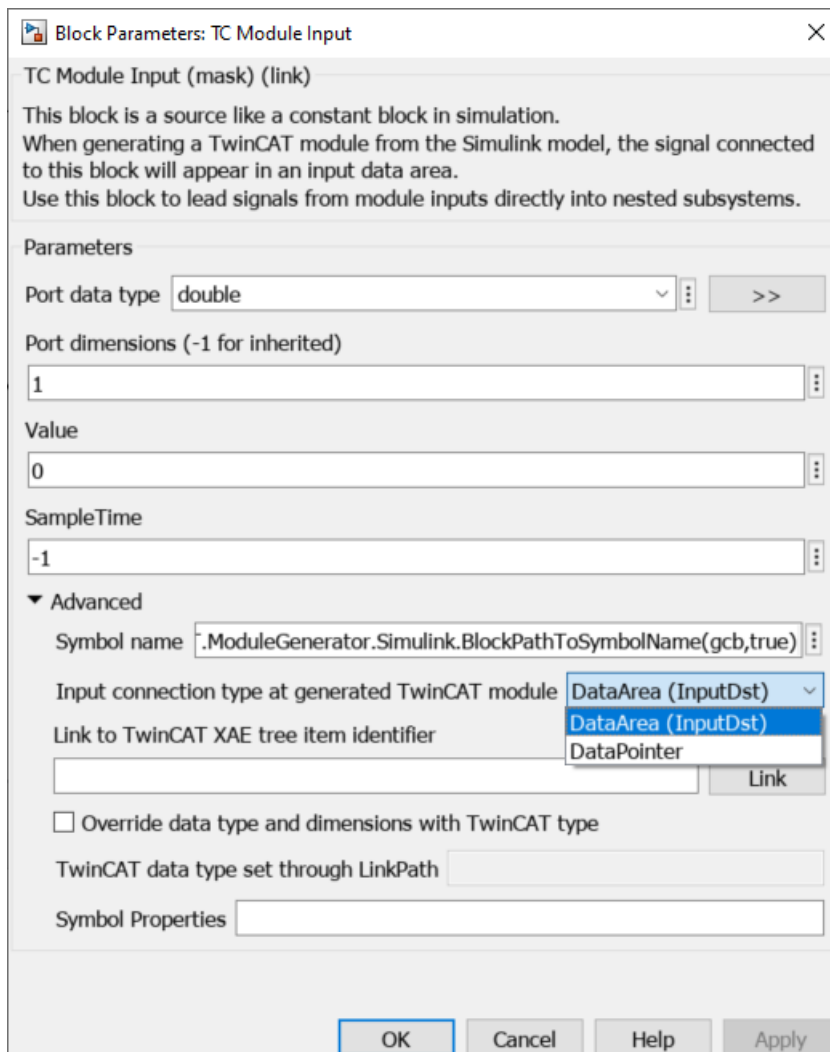
4.5 TwinCAT Library in Simulink®

Specific blocks for the TwinCAT Target for Simulink® are located in the **Library Browser > Beckhoff TwinCAT Target** area.



4.5.1 TwinCAT Input and Output modules

TwinCAT-specific input and output function blocks can optionally be used in Simulink®. Another valid way is to use the standard input ports (In) and output ports (Out) of Simulink®. This is usually also the *best practice* way, unless the additional functions of the TwinCAT input and output modules described below are required.



Additional functions of the TC Module blocks

If you use the input function blocks (TC Module Input) and output function blocks (TC Module Output) provided by Beckhoff, you will benefit from the following additional functionalities, compared to the standard Simulink® input and output ports:

- You can also define signals and buses from **subsystems directly as inputs or outputs for TcCOM**, without first transferring the signals/buses from the subsystem to the top system. Sample: [Subsystem inputs and outputs](#) [► 111].
- You can optionally store an automatic mapping to other TcCOM or I/Os in the block parameters so that the **mapping is executed automatically** directly when the TcCOM is instantiated, see [Automatic mapping](#) [► 111].
- You can select individually between Mapping and **DataPointer** for the *connection type* of each TC Module Inputs/Outputs. You can make all Standard Input Ports accessible via mapping, for example, and make others accessible via the TC Module Inputs via DataPointer, see [DataArea or DataPointer](#) [► 113].
Sample: [Shared memory between TcCOM instances](#) [► 150].
- You can influence the **Symbol Name**, i.e. change the name and hierarchy of the input or output in the process image, for example, see [Symbol Name](#) [► 114].
- You can equip signals and buses with specific **Symbol Properties**, see [Symbol properties](#) [► 114].
- You can use **initial values** for inputs. To do this, set the Value of the TC Module Inputs to any value, see [Initial values](#) [► 115].



Initial values can also be implemented for Standard Input Ports, see [Option Input: Initial values](#) under TC TcCom Interfaces [▶ 143].

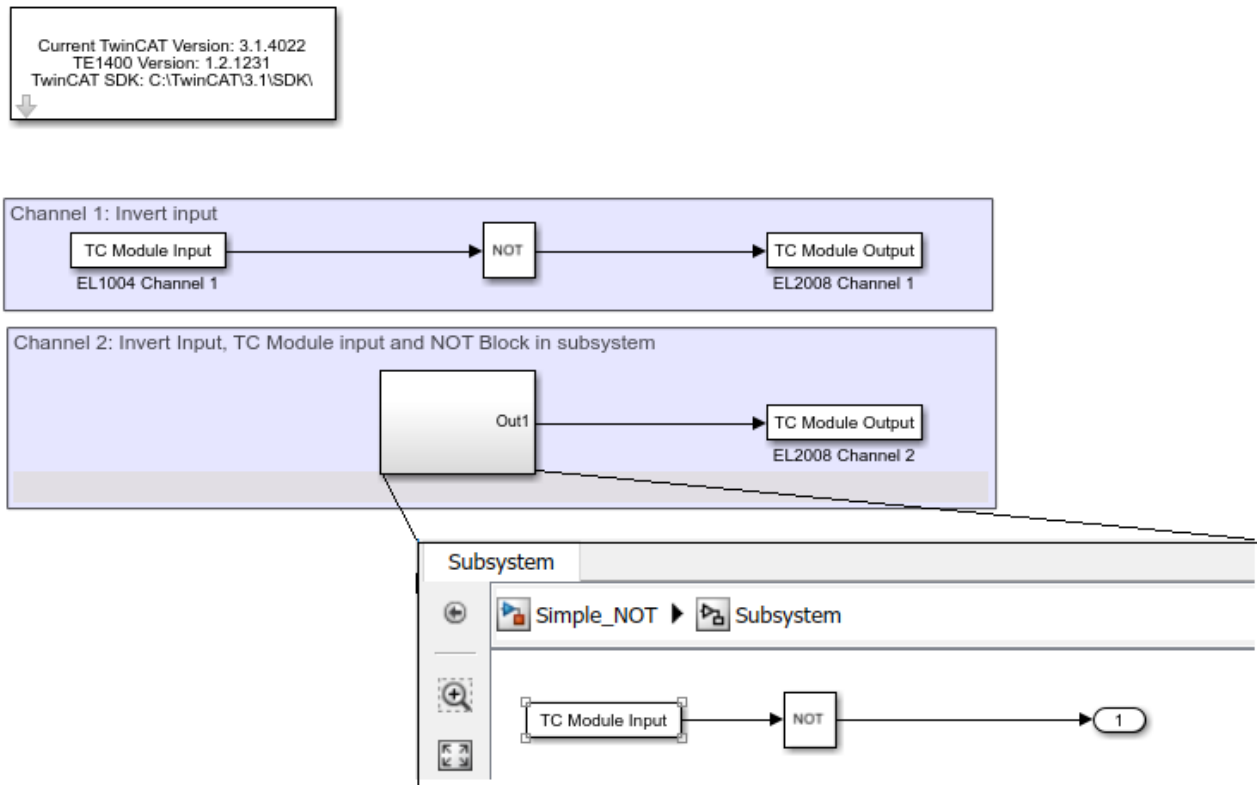
When using automatic mapping, please note that if the TcCOM is instantiated more than once in TwinCAT, you will end up with a mapping conflict which you must resolve by manual mapping. This option is therefore not recommended for multiple instantiations.

4.5.1.1 Subsystem inputs and outputs

Creating a TcCOM input/output from a Simulink® subsystem

A Simulink® model is created, which outputs two negated inputs. An input is placed in a subsystem, see figure below.

According to the described property of the TC Module Input/Output blocks, the TC Module Input in the Simulink® subsystem in TwinCAT is also included in the process image of the TcCOM. Thus, it is not necessary to route to the top level of the Simulink® model, as is the case with standard in-ports or out-ports.



4.5.1.2 Automatic mapping

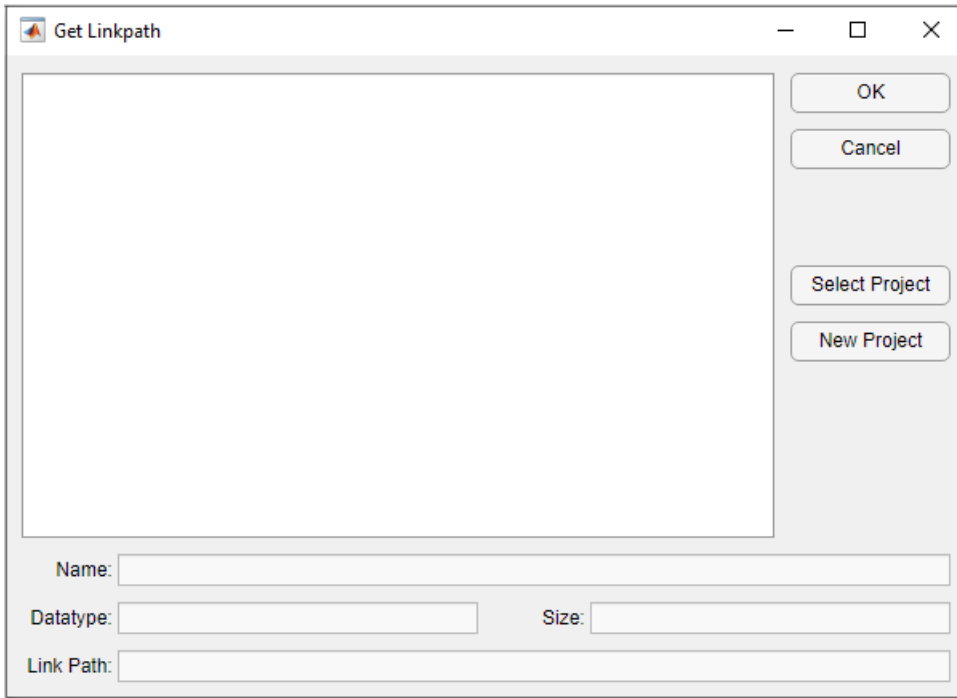
The inputs and outputs of the Simulink® model are automatically mapped to digital inputs and outputs in the following. This means that after instantiation of the TcCOM in a TwinCAT project, the mappings are created automatically. Manual linking is no longer necessary.

Navigate to the parameter **Link to TwinCAT XAE tree item identifier** in the Block Parameters of the Tc Module In or Tc Module Out under **Advanced**. You can either enter the Tree Item Identifier and the data type manually, or select them from Simulink® via a browser.

By selecting the **Link** button a new dialog opens. You can now load an existing TwinCAT project (**Select Project**) and browse the existing inputs or outputs or you can create a new project (**New Project**) and automatically scan the EtherCAT fieldbus in the new project to link it with the detected I/Os.

- ✓ Create a new project and display and select inputs and outputs of the target:
 1. Create a new project.

2. For this purpose, select a memory path of the new TwinCAT project to be created.
 3. Select the target to which you want to download the project.
 4. Automatically scan the I/O tree of the target.
- ⇒ All inputs and outputs of the target are displayed and can be selected.

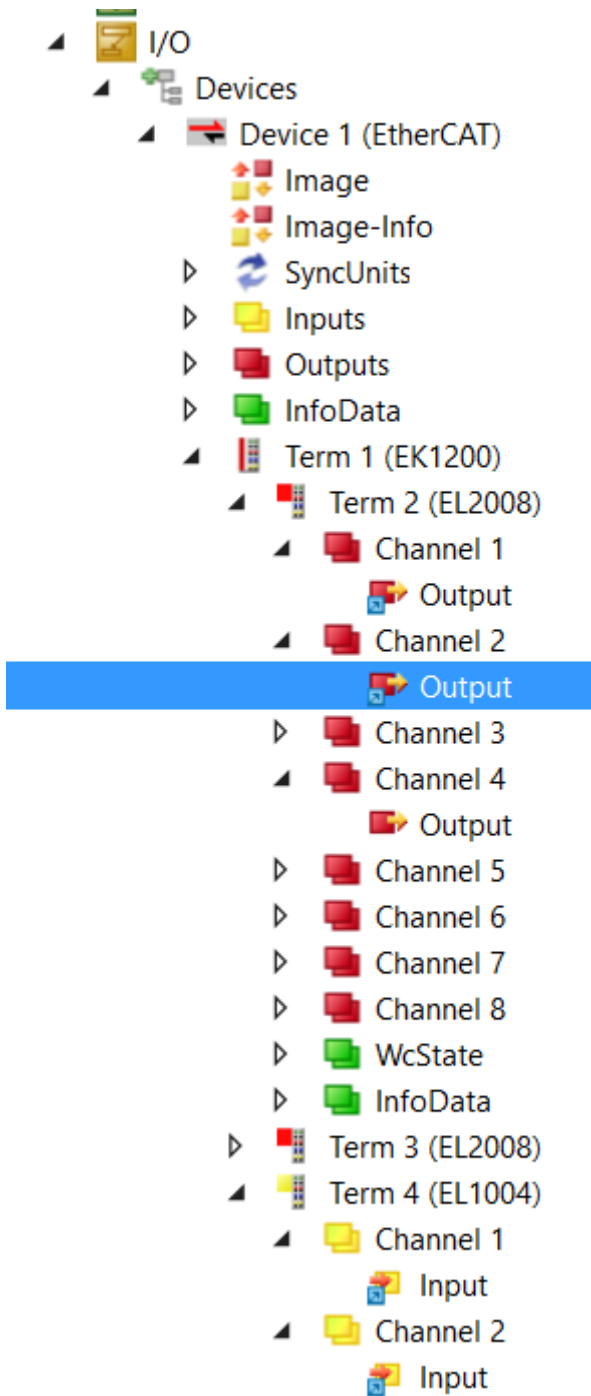


By selecting the input or output you want to link, the tree item identifier is automatically set and the appropriate data type is automatically entered in Simulink®.

If the Simulink® model described above is compiled into a TcCOM and integrated in a TwinCAT 3 Solution, a mapping to the inputs and outputs selected in Simulink® is automatically created.

Coloring of the symbols for differentiation:

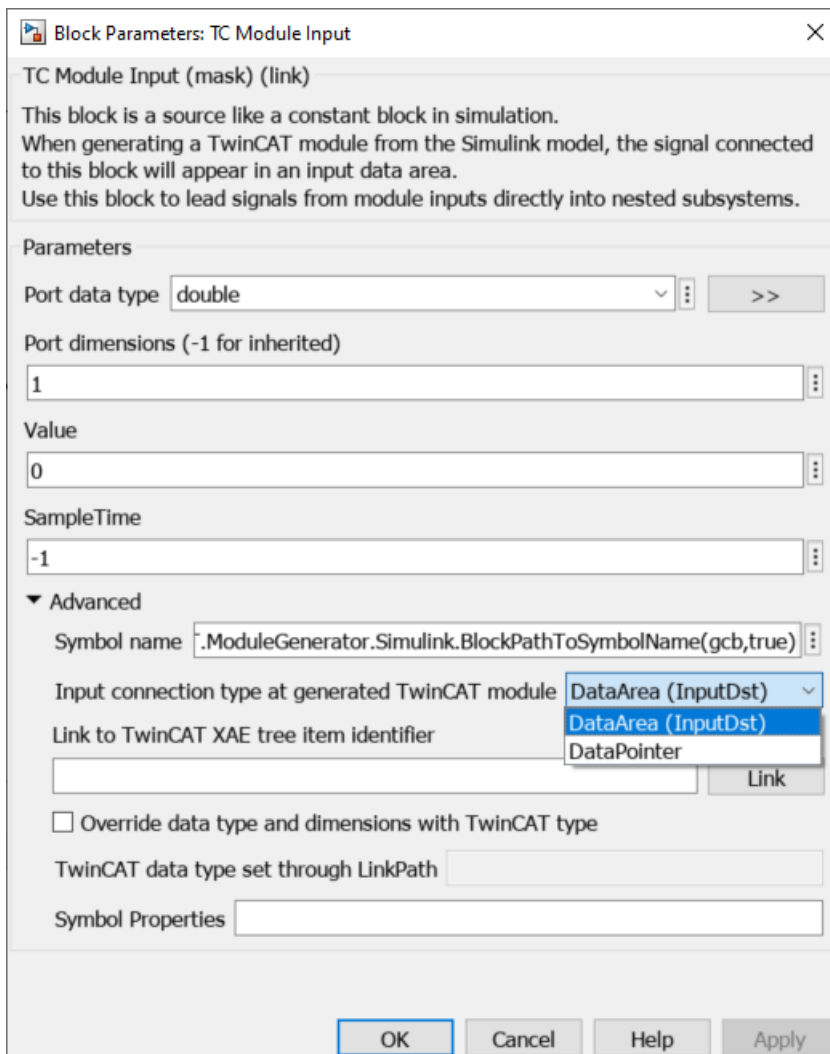
the automatically generated mappings are provided with a *blue* symbol, while manual mapping symbols are displayed *white*.



4.5.1.3 DataArea or DataPointer

By default, "DataArea (InputDst)" is selected as **Input connection type at generated TwinCAT module**. This means that a TC Module Input is created as *Input Destination DataArea* and a TC Module Output accordingly as *Output Source DataArea*.

Alternatively, "DataPointer" can also be selected here, see [Shared memory between TcCOM instances \[p. 150\]](#).



If the Simulink® model shown above is created with the **Target for Simulink®**, 2 inputs and 2 outputs appear in the process image on the TcCOM instance if "mapping" is selected as **Input connection type at generated TcCOM module**.

4.5.1.4 Symbol properties

You can define specific symbol properties for the TC Module In/Out blocks, see [Symbol Properties and Attribute Pragmas](#) [► 166].

Sample: OPC.UA.DA.=1



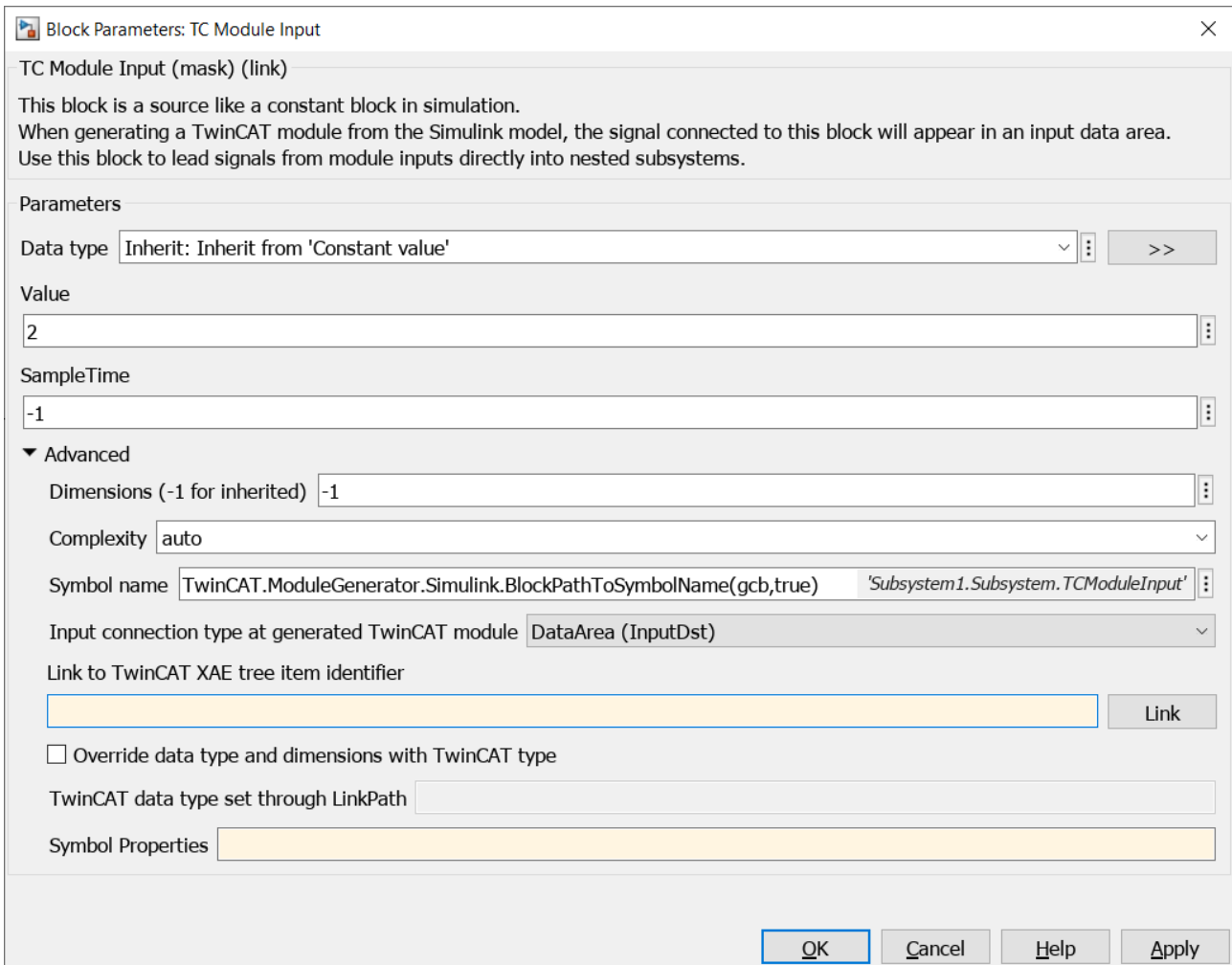
Restriction

The symbol name must not be nested when using symbol properties.

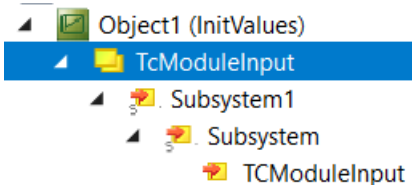
4.5.1.5 Symbol Name

You can change the name and thus the display of the symbol in the process image at *Symbol name*.

In the following sample, a TC Module Input is nested in two subsystems. The default name resolution is (as can be seen to the right of the name symbol) <Subsystem>.<Subsystem>.<TcModuleName>.



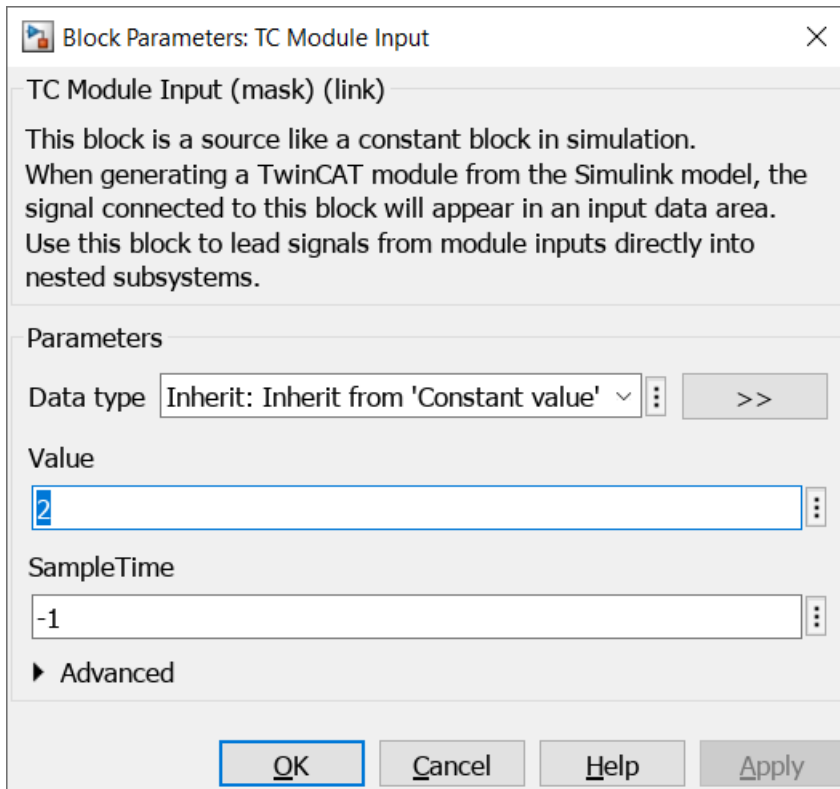
The process image looks like this:



Especially in cases where there is very deep nesting, it may be desirable to shorten the symbol name. You therefore have the option of entering the symbol name manually as text, e.g. 'Subsystem.InputValue'.

4.5.1.6 Initial values

In addition to data type, dimension and SampleTime, an optional **Value** field is editable in addition to the standard Simulink® in-port and out-port. This field allows you to specify an initial value for the input (only available for TC Module Input). If the input in TwinCAT is not linked to a corresponding output, the value entered here is used as the input value.



4.5.2 TwinCAT Environment View

Drag the TwinCAT Environment View into your Simulink® environment to directly get the available TwinCAT XAE version and the currently installed TE1400 version displayed, for example for support cases.

4.5.3 TwinCAT File Writer

The TwinCAT File Writer block writes .mat files from the TwinCAT environment. The block only fulfills this function if the Simulink® model has already been transferred to a TcCOM or FB and is executed in a TwinCAT Runtime. If the Simulink® model is executed in Simulink®, this block has no function.



Parameter	Description	Note
Port data type	Data type of the incoming signal	Support is offered for: <ul style="list-style-type: none"> Integer Types float double boolean enums bus objects
Port dimension	Dimensionality of the incoming signal	-1 -> Inherit Otherwise [1.2], [1.5], ..., for example
SampleTime	Block Sample Time in seconds	-1 -> Inherit

Parameter	Description	Note
file name	File name of the .mat file	Full path or relative path possible. Relative path relative to <i>TwinCAT\3.1\Boot</i> .
Maximum file byte size	Maximum size (in bytes) of the .mat file. When this size is reached, the current file is closed and a new one is started.	0 -> Maximum due to file format
Maximum file count	Maximum number of .mat files to write.	0 -> Infinite files. If the maximum is reached, old files are overwritten, starting with <i>_part0.mat</i> .
Pause writing files	Pauses writing	Parameters on the TcCOM
Write simulation time with data	Writes per date a structure with 2 fields, "time" and "data".	
Expose Pause as block input	Creates an input via which the TwinCAT File Writer can be paused.	

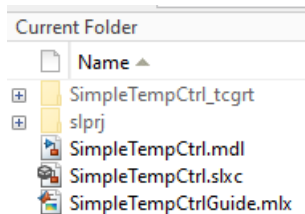
4.6 Overview of automatically generated files

When a build process is initiated, some files and folders are created automatically. "Where are the files located?", "What can be done with them?" and "What do the files mean?" These questions are answered below.

What are the categories of automatically generated files?

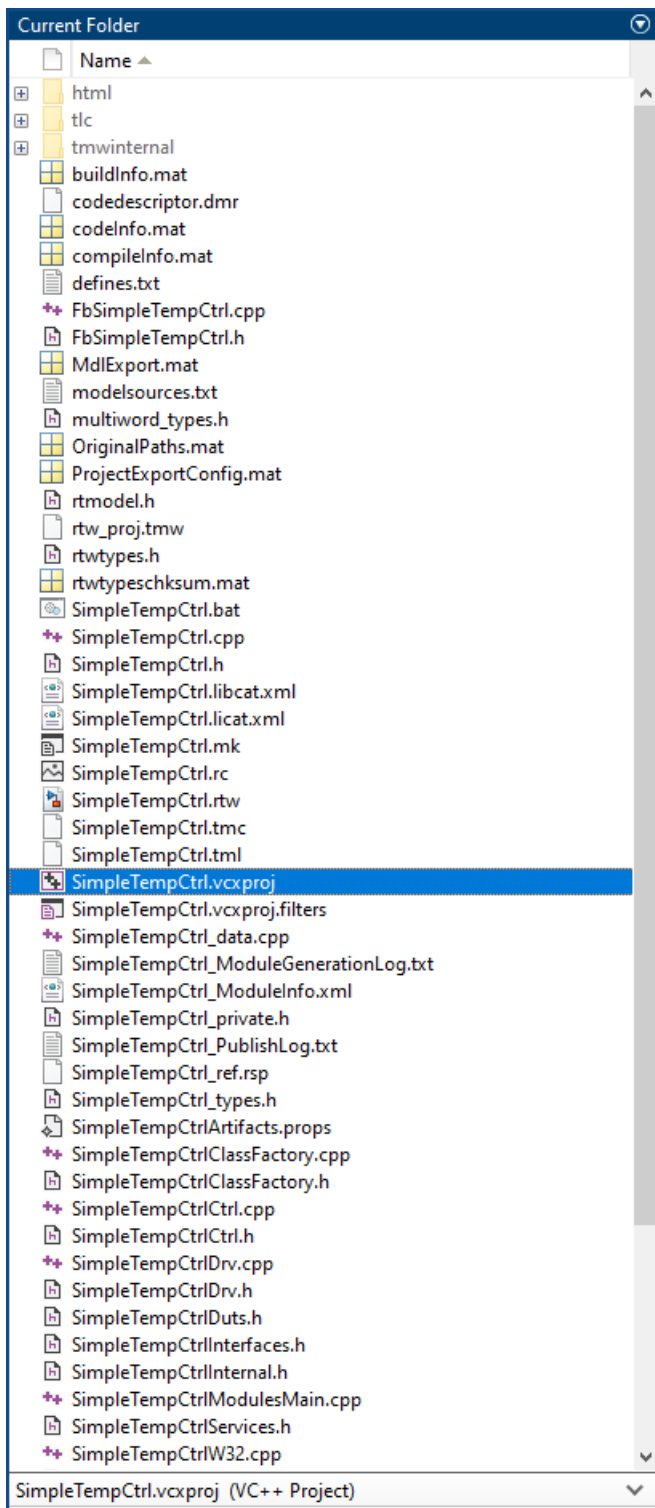
- [Source code is generated \[► 117\]](#).
- [Log files are generated \[► 119\]](#).
- [The TwinCAT objects, drivers \(*.tmx\) and description files \(*.tmc, *.library, ...\) are created \[► 120\]](#).

All files created by the TwinCAT Target are summarized in the current MATLAB® path in the folder *<SimulinkModelName>_tcgrt*. The folder is located next to the slprj folder generated by MathWorks®.



Generated source code

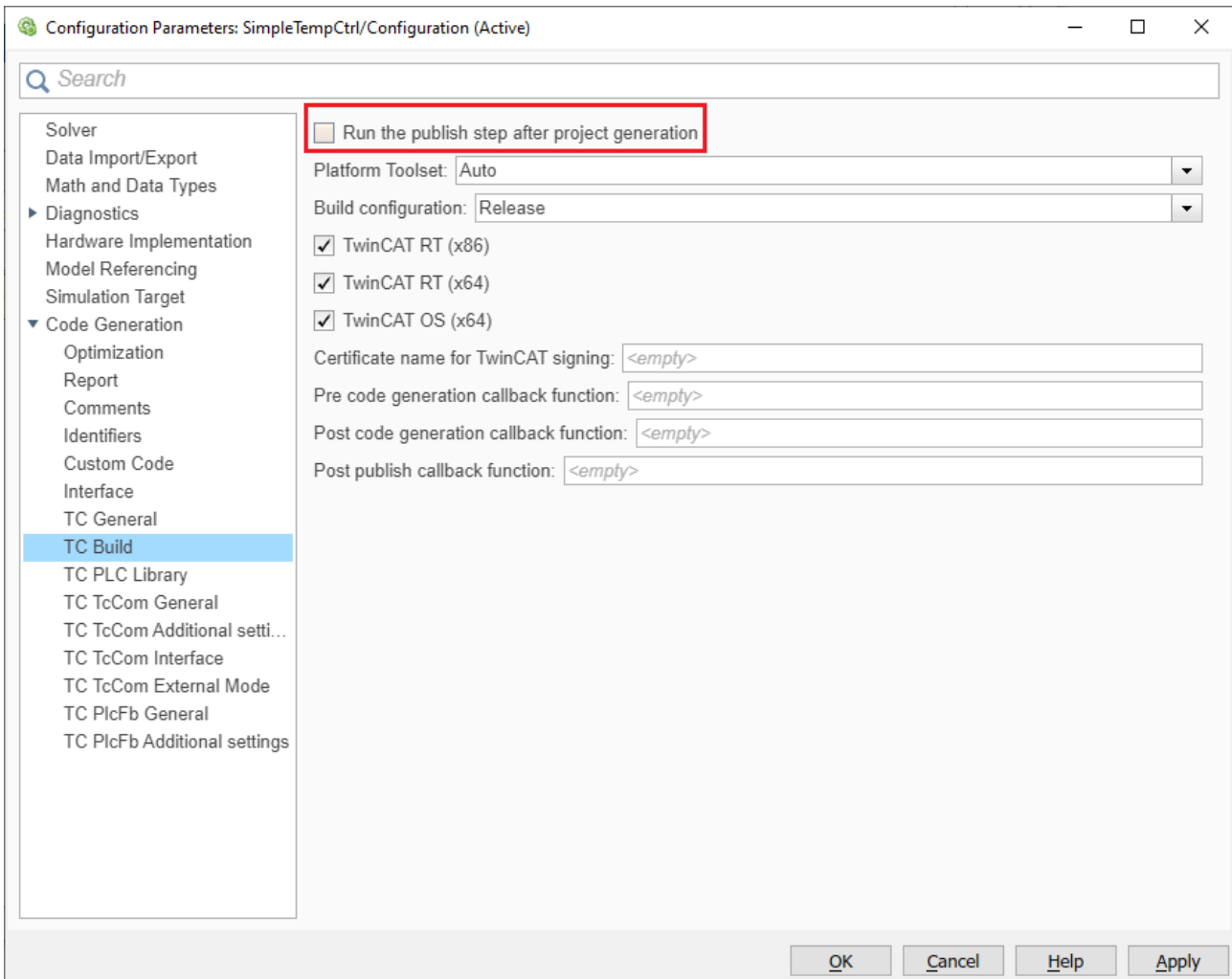
The central file for the source code is *<SimulinkModelName>.vcxproj* and is located in the *<SimulinkModelName>_tcgrt* folder. The file opens a TwinCAT C++ project, which can be used to inspect the generated source code, to subsequently build TwinCAT objects or also for [debugging in TwinCAT \[► 214\]](#).



With regard to the build option from the generated TwinCAT C++ project, it is worth knowing that you can switch off the publish step, i.e. building for the configured platforms, in Simulink®.



You can achieve code generation without build in Simulink® by deselecting *Run the publish step after project generation*. The *publish step* contains the *build* of the TwinCAT objects for the selected platforms (TwinCAT RT x86, x64 ...).

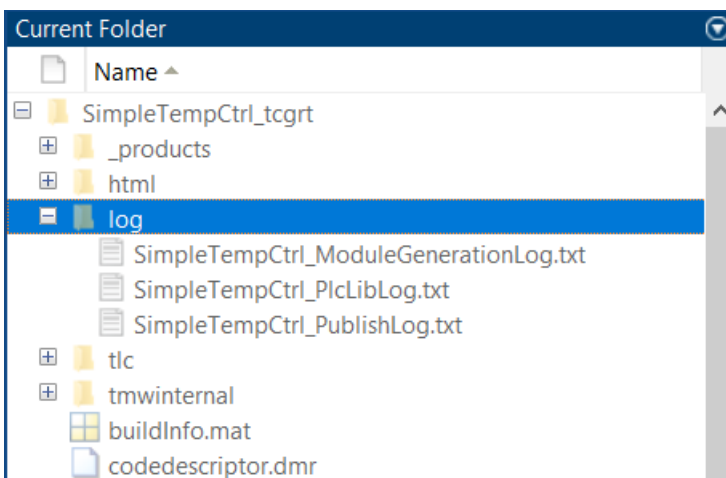


Further useful information:

- Pack-and-go with the Target for Simulink®
- Continuous Integration

Generated log files

The generated log files are summarized in the subfolder *log*. The log files created are the first place to look when debugging.



This folder contains up to three log files. The main point of contact is the summary of all logs in the file *<ModelName>_ModuleGenerationLog.txt*.

i Beckhoff Support requires log file

If you require assistance from our support team, please send at least the following file from the log folder: `<ModelName>_ModuleGenerationLog.txt`

The structure of the ModuleGenerationLog is divided into several segments that represent different steps of the TwinCAT Target. The steps are displayed with `###` in the log.

Sample:

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

If no warnings or errors occur in the steps, no entries are usually made for the steps performed. In some cases, explicit reference is made to created files, such as the vcxproj file created in the sample above. Reference is also made to other log files for more detailed information, see the following sample:

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

If warnings or errors occur, these are displayed in the ModuleGenerationLog. The detailed logs, on the other hand, contain all the outputs of the step performed. For example, warnings can be seen here in the signature verification area of the created tmx files:

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

Created TwinCAT objects

After a successful *build*, the binary files and description files created, which can be re-used in TwinCAT XAE, are stored in the so-called *Engineering Repository*, i.e. on the engineering PC at:

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


> TwinCAT > 3.1 > Repository > TE140x Module Vendor > SimpleTempCtrl > 2.0.1.24 >

Name	Date modified	Type	Size
deploy	10/18/2023 2:49 PM	File folder	
TwinCAT OS (x64)	10/18/2023 2:49 PM	File folder	
TwinCAT RT (x64)	10/18/2023 2:49 PM	File folder	
TwinCAT RT (x86)	10/18/2023 2:49 PM	File folder	
SimpleTempCtrl.library	10/18/2023 2:49 PM	LIBRARY File	141 KB
SimpleTempCtrl.tmc	10/18/2023 2:49 PM	TMC File	36 KB
SimpleTempCtrl.tml	10/18/2023 2:49 PM	TML File	118 KB

- <ModelName>.tmc: Description file of the TcCOM object (object class)
- <ModelName>.tml: Library file: Can be used like .library file.
- <ModelName>.library: TwinCAT PLC library file
- Deploy\<ModelName>_ModuleInfo.xml: Module information, e.g. block diagram
- <Platform>\<ModelName>.tmx: Driver file

Distribution of TwinCAT objects to other XAE systems: If the folder at <ModelName> level is copied to other PCs with TwinCAT XAE in the local *Engineering Repositories*, their users can use the created TwinCAT objects in their TwinCAT Solutions.

Compare also [Sharing created TwinCAT objects \[► 135\]](#).

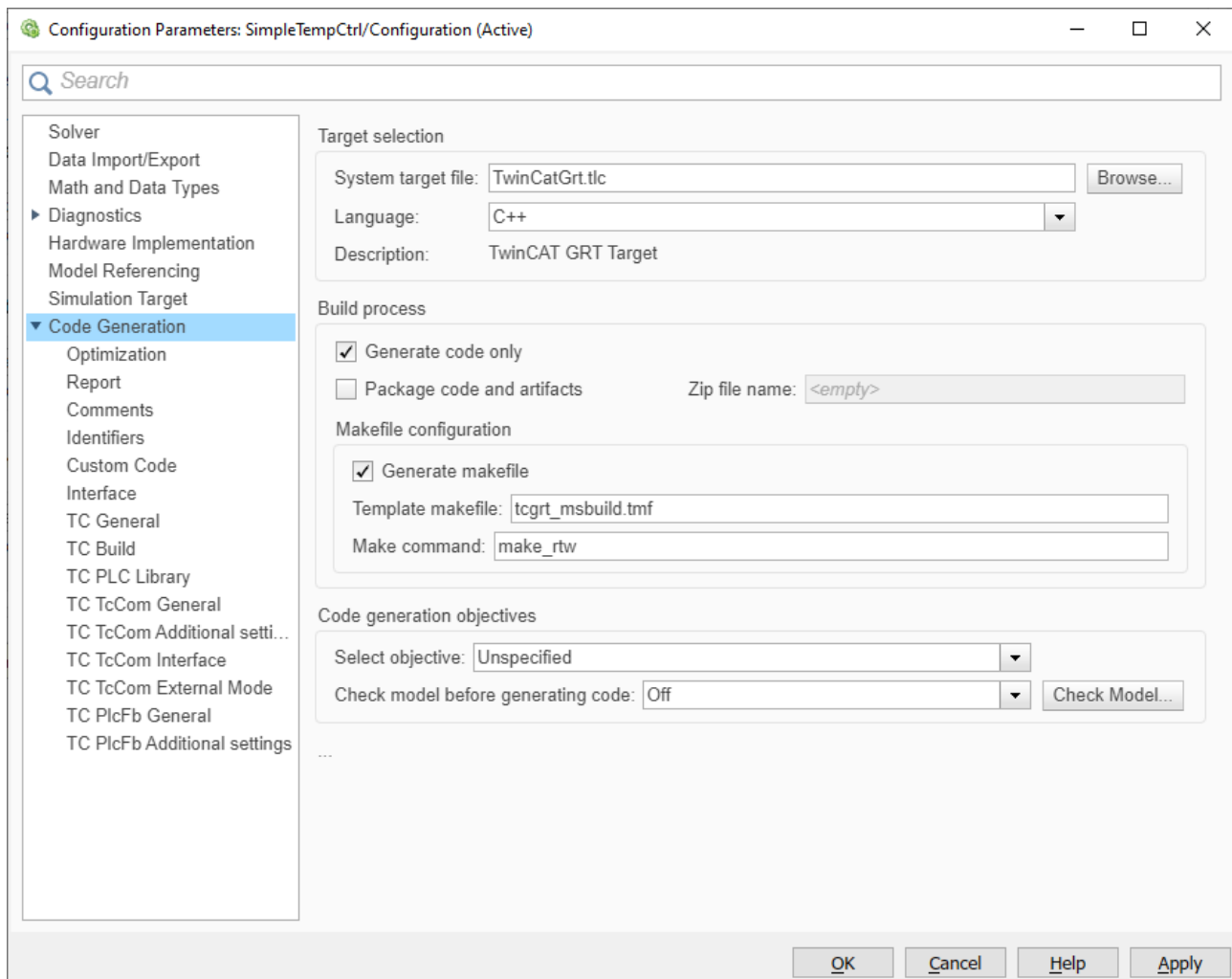
Additional Notes

[Description of the generated C++ files and binary files](#)

[Versioned C++ projects](#)

4.7 Parameterization of the code generation in Simulink®

Within Simulink® a large number of settings can be made for the configuration of the TwinCAT objects to be generated (TcCOM and function blocks). For this purpose the tree structure under Code Generation is extended by several entries (see entries starting with TC) as soon as you have selected the **TwinCatGrt.tlc** as System **target file**.



Configure View

Due to the wide range of configuration options, it is possible to switch the view. After installation, the configuration level is set to *Standard*, which displays only the most frequently used parameters. You can also increase the configuration level to *Advanced* to be able to make significantly more settings.

Use the MATLAB® Command Window to set the view:

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

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

The setting made is initially only temporary. To save them, use the Save command:

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

If you run the code generation and build process on different systems, make sure that the configuration level is identical on both systems.

Overview of the configuration parameters

An overview of all configuration parameters can be found in the documentation in section [Overview table of all configuration parameters \[▶ 124\]](#).



Read the tooltips

Hover with the cursor over the text fields of the dialog boxes to bring up a detailed description of the option as a tooltip.

Change the software setup

To change the default settings of the TwinCAT target, you can access a dialog in the MATLAB® Console as follows:

TwinCAT.ModuleGenerator.Settings.Edit

Here you are offered various entries that you can store as default values.

Accept changes

1. Enter the new default settings in the dialog box.
 2. Confirm with the **Save** button.
 3. Restart MATLAB®.
- ⇒ The changes have been adopted.

Change configurations after build

Many settings selected in the *Configuration parameters* can be changed again in TwinCAT 3 at the level of the TcCOM instances, so that, for example, it is defined for the class of a model that it is to be called via a cyclic task, but the individual instance can also be configured subsequently for calling from the PLC.

4.7.1 Overview table of all configuration parameters

An overview of all configuration parameters is given below. The configuration parameters are visible or invisible depending on the set configuration level in the UI of Simulink®. If you configure the module generator via an m-file [► 131], the configuration level has no influence.

A brief description is given for each parameter and in some cases reference is made to other sections of the documentation for more detailed information. The categorization is based on the representation in the UI. In the UI, the display name is used. If you use the module generator via an m-file, the column "Name" is crucial.

Category	Name	Displayname	Default	Description
TC General	Generate	Generate TwinCAT C++ Project	TRUE	<u>Generate a TwinCAT C++ project. If unset, only code artifacts will be generated which can get used to generate C++ projects later</u> [► 131].
	FullPath	TwinCAT C++ Project Path		Full path to the generated VCXPROJ file (e. g. "C:\Temp\MyGeneratedProject.vcxproj")
	LowestCompatibleTcBuild	Lowest compatible TwinCAT version (build number)	\$(TwinCAT:Version:BUILD)	The lowest TwinCAT build number the generated C++ project and its modules and POU's are to be compatible with.
	ClassFactoryName	Class factory name	\$(Project:Name)	Name of the generated C++-Project, Name of the TcCOM classfactory and tmx-file name
	ProductName	Product name	\$(ModuleGenerator:ProductId) \$(ModuleGenerator:Version:MAJOR.MINOR)	Product name, used e.g. for the <u>module driver description and the module TMC description</u> [► 159].
	Copyright	Copyright notice	Copyright \$(VendorName) \$(LocalDateTime:%Y)	<u>Copyright notice of the generated module driver file</u> [► 159]
	Description	Driver description	TwinCAT executable file, generated by TwinCAT \$(ModuleGenerator:ProductId)	<u>Driver description</u> [► 159]
	VendorName	Vendor name	TE140x Module Vendor	Module vendor name, used as the <u>company name of the generated executables in the repository</u> [► 120] and <u>the major module group as shown in the TwinCAT XAE module dialog</u> [► 90].
	VersionSrc	Version source file	\$(LatestTMFile)	<u>Path to an existing TMC, TML or XML file containing the previous version value</u> [► 137].
	IncrementVersion	Version part for increment	Revision	<u>The part of the version number that is to be incremented</u> [► 137].
	DrvFileVersion	Driver file version	\$(VersionFromFile)	<u>Executable file version and library version.</u> [► 137]

Category	Name	Displayname	Default	Description
	DrvProductVersion	Driver product version	\$(DrvFileVersion)	Product version [▶ 137]
	CodeGenPlaceholders	Code generation placeholders		Define custom placeholders
	UseDataExchangeModules	Load DataExchangeModules	0	Manually set DataExchangeModule dependency (currently no need to set manually)
	MaxVisibleArrayElements	Maximum number of visible array elements	200U	Specifies the maximum number of array elements to be displayed in the TwinCAT XAE. In the TwinCAT XAE, larger arrays cannot get expanded and linked to by its individual items
	CreateUniqueEnumItemNames	Create unique item names for enumeration types	1	Create unique item names for enumeration types.
	DataTypeTmcFiles	Data type import TMC files		TMC file(s) containing additional type definitions for code generation
TC Build	PreferToolArchitectureX64	Prefer X64 build tools	TRUE	Prefer X64 compiler and linker. Useful for complex source files, where X86 tools may run out of heap space.
	Verbosity	Codegeneration and build verbosity	Normal	Verbosity level of code generation and build output messages. Silent and Detailed are other possible values.
	Publish	Run the publish step after project generation	TRUE	Start the build procedure after code generation for all selected platforms. The generated module binaries and module description files will get copied to the "publish folder". Published modules are automatically located by the XAE and can get instantiated in all TwinCAT 3 projects. If unset, the module generation process will be stopped after code generation. To instantiate in a TwinCAT3 project, the generated C++ project needs to be inserted and built from.
	PublishPlatformtoolset	Platform Toolset	Auto	Choose Platform Toolset to build binaries.
	PublishConfiguration	Build configuration	Release	Build configuration to build binaries.
	PublishTcRTx86	TwinCAT RT (x86)	TRUE	Publish binaries for platform 'TwinCAT RT (x86).'
	PublishTcRTx64	TwinCAT RT (x64)	TRUE	Publish binaries for platform 'TwinCAT RT (x64).'
	PublishTcOSx64	TwinCAT OS (x64)	TRUE	Publish binaries for platform 'TwinCAT OS (x64)' (e.g. TwinCAT/BSD)
	ForceRebuildForPublish	Always rebuild all source files on publish	FALSE	Always rebuild all source files on publish

Category	Name	Displayname	Default	Description
	SignTwinCatCertificateName	Certificate name for TwinCAT signing		Certificate name for TwinCAT signing with OEM Certificate level 2. [► 93]
	TmxInstall	Install TMX	TRUE	Install all generated TwinCAT Objects on local XAE (fill local Engineering Repository [► 120]).
	TmxArchive	TMX Archive		Name of an optional archive containing all files required to use the generated TwinCAT Objects on another TwinCAT development system. [► 135]
	MsBuildPublishProperties	MsBuild publish properties		Set additional MsBuild publish properties.
	MsBuildProjProperties	MsBuild project properties		Set additional MsBuild project properties.
	PreCodeGenerationCallbackFunction	Pre code generation callback function		The defined MATLAB® function is called before code generation [► 182].
	PostCodeGenerationCallbackFunction	Post code generation callback function		The defined MATLAB® function is called after code generation [► 182].
	PostPublishCallbackFunction	Post publish callback function		The defined MATLAB® function is called after publish [► 182].
TC PLC library	LibCatPath	PLC library category description file	\$(ProjectDir)\\$(Name>.libcat.xml	Path to the PLC library category description file
	LibraryCategories	PLC library categories	\$(VendorName>	Define PLC library category hierarchy. Default only one hierarchy level = vendor. List separated with possible: <MainCategory> <SubCategory1> ...
	GeneratePlcLibrary	Generate a PLC library	FALSE	Generate a PLC library with POU's. [► 206] Define containing POU's with parameter TcComWrapperFb and PlcFb>General>Generate.
	InstallPlcLibrary	Install the generated PLC library	FALSE	Install the generated PLC library for use in the local TwinCAT XAE/ PLC [► 206].
	PlcTypePrefixes	Type Prefixes		Define custom type prefixes
	PlcVarPrefixes	Variable Prefixes	`PVOID=p \\ BOOL=b \\ BOOL32=b \\ DATE=d \\ TIME_OF_DATE=td \\ TIME=t \\ LTIME=t \\ GUID=n`	Define custom variable prefixes.
TC License	OemId	ID of OEM		ID of OEM. Required for OEM Licence checks [► 156]

Category	Name	Displayname	Default	Description
	OemLicenses	IDs of OEM Licenses		IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 156]
TC TcCom General	Generate	Generate TcCOM Module (TwinCAT Module Class)	TRUE	Generate a TcCOM module class for the model.
	OnlineChange	Online change support	FALSE	Allow to switch between different TcCOM module versions without switching TwinCAT runtime to config mode [▶ 141].
	ModuleProperties	TMC Properties		Additional properties added to the module description in the TMC file: Name1=Value1 Name2=Value2 ...
	GroupName	GroupName	TE140x Simulink Modules	Minor module group name in the TwinCAT XAE module dialog
	GroupDisplayName	GroupDisplayName	\$(GroupName)	Minor module group description in the TwinCAT XAE module dialog
	GroupIcon	GroupIcon	\$(TE140x:Icon)	Optional module group icon in the TwinCAT XAE module dialog
	ModuleIcon	ModuleIcon	\$(TE140x:Icon)	Optional module icon in the TwinCAT XAE module dialog
	InitExceptionHandler	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress or handle floating point exceptions during initialization [▶ 220].
	UpdateExceptionHandler	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress or handle floating point exceptions during cyclic execution [▶ 220].
	AdditionalIncludeFiles	Additional include files		Additional files required to be included after rtwtypes.h
TC TcCom License	OemLicenses	IDs of OEM License	\$(Project:OemLicenses)	IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 156]
TC TcCom Wrapper	TcComWrapperFb	TcCom Wrapper FB	FALSE	Generate a PLC Functionblock simplifying the interaction between a PLC and an instance of the generated TcCOM module [▶ 209]
	TcComWrapperFbProperties	TcCom Wrapper FB properties	FALSE	Generate properties for accessible data in the referenced TcCOM object [▶ 209]
	TcComWrapperFbPropertyMonitoring	TcCom Wrapper FB property monitoring	NoMonitoring	NoMonitoring: Online values of properties are not monitored in the PLC online view, CyclicUpdate: Update property values in the PLC online view cyclically, ExecutionUpdate: Update

Category	Name	Displayname	Default	Description
				<u>property values in the PLC online view when the property getter or setter is called</u> [► 209]
TC TcCom Additional settings	ModuleCaller	Default module caller	CyclicTask	CyclicTask: Call module via TwinCAT Task. Module: Call module from another TwinCAT module (see e.g. TcCOM-Wrapper-FB).
	CallerVerification	Verify caller	Default	Verify the caller context to prevent concurrent execution of the model code and corresponding DataArea mappings. Skip verification to reduce the execution time.
	StepSizeAdaptation	Default StepSize adaptation mode	RequireMatchingTaskCycleTime	Configure how to handle differences between the default model step size(s) and the cycle time of the assigned task(s).
	ExecutionSequence	Default execution sequence	UpdateBeforeOutputMapping	Configure the execution order of input mapping, model code execution and output mapping.
	ExecuteModelCode	Execute model code after startup	TRUE	Start cyclic execution of the model code after startup by default. If FALSE, Module Parameter Execute needs to be set to TRUE to start execution of code.
	BlockDiagramExport	Export BlockDiagram	TRUE	<u>Export graphical block diagram information for monitoring and optional debugging on the generated TwinCAT module in TwinCAT XAE</u> [► 193]
	ResolveMaskedSubsystems	Resolve Masked Subsystems	FALSE	Resolve masked subsystems in the block diagram
	ExtendSignalResolution	Extended resolution of signals in block diagram	FALSE	<u>Intensified search for assignments of variables and block diagram signals (blue signals). This option increases the build time.</u> [► 235]
	BlockDiagramVariableAccess	Access to VariableGroup not referenced by any block	AssignToParent	Variables from a block within an unresolved subsystem are either assigned to the next higher visible block or hidden in the block diagram.
	BlockDiagramDebugInfoExport	Export BlockDiagram debug info	TRUE	<u>Export additional information required to debug the module using the block diagram</u> [► 197].
TC TcCom Interfaces	ExecutionInfoOutput	Create ExecutionInfo output	FALSE	<u>Create additional output DataAreas containing execution and exception information</u> [► 220].
	MonitorExecutionTime	Monitor execution time	FALSE	Calculate and expose the execution time of the module as an ADS variable for monitoring purposes.

Category	Name	Displayname	Default	Description
	InputDataAccess	Input: Data Access	Input Destination DataArea	Defines how the input variables are exposed in TwinCAT [▶ 143].
	InputCreateSymbols	Input: Create ADS Symbols	TRUE	Create ADS symbol information for the input variables [▶ 143]
	InputInitValues	Input: Initial values	FALSE	Create module parameters for the input variables to allow definition of initial values [▶ 143]
	InputProperties	Input: TMC Properties		Additional properties added to the Input symbol description in the TMC file. [▶ 166]
	OutputDataAccess	Output: Data Access	Output Source DataArea	Defines how the output variables are exposed in TwinCAT [▶ 143].
	OutputCreateSymbols	Output: Create ADS Symbols	TRUE	Create ADS symbol information for the output variables [▶ 143].
	OutputProperties	Output: TMC Properties		Additional properties added to the Output symbol description in the TMC file. [▶ 166]
	ParametersDataAccess	Parameters: Data Access	Internal DataArea	Defines how the model parameter variables are exposed in TwinCAT [▶ 143]
	ParametersCreateSymbols	Parameters: Create ADS Symbols	TRUE	Create ADS symbol information for the model parameter variables [▶ 143].
	ParametersInitValues	Parameters: Initial values	TRUE	Create module parameters for the model parameter variables to allow definition of initial values [▶ 143].
	ParametersProperties	Parameters: TMC Properties		Additional properties added to the Parameters symbol description in the TMC file. [▶ 166]
	BlockIODataAccess	BlockIO: Data Access	Internal DataArea	Defines how the BlockIO variables are exposed in TwinCAT [▶ 143]
	BlockIOCreateSymbols	BlockIO: Create ADS Symbols	TRUE	Create ADS symbol information for the BlockIO variables [▶ 143].
	BlockIOProperties	BlockIO: TMC Properties		Additional properties added to the BlockIO symbol description in the TMC file. [▶ 166]
	ContStateDataAccess	ContState: Data Access	Internal DataArea	Defines how the continuous state variables are in TwinCAT [▶ 143]
	ContStateCreateSymbols	ContState: Create ADS Symbols	TRUE	Create ADS symbol information for the continuous state variables [▶ 143].
	ContStateProperties	ContState: TMC Properties		Additional properties added to the ContState symbol description in the TMC file. [▶ 166]
	DWorkDataAccess	DWork: Data Access	Internal DataArea	Defines how the DWork variables are exposed in TwinCAT [▶ 143]

Category	Name	Displayname	Default	Description
	DWorkCreateSymbols	DWork: Create ADS Symbols	TRUE	Create ADS symbol information for the DWork variables [▶ 143].
	DWorkProperties	DWork: TMC Properties		Additional properties added to the DWork symbol description in the TMC file. [▶ 166]
	DataStoreDataAccess	DataStore: Data Access	None	Defines how the DataStore variables are exposed in TwinCAT [▶ 143]
	DataStoreCreateSymbols	DataStore: Create ADS Symbols	TRUE	Create ADS symbol information for the DataStore variables [▶ 143].
	DataStoreReadOnly	DataStore: Read Only	FALSE	Restrict ADS access to be read only for the DataStore variables [▶ 143].
	DataStoreProperties	DataStore: TMC Properties		Additional properties added to the DataStore symbol description in the TMC file. [▶ 166]
	SymbolProperties	Additional TMC Symbol Properties		Additional properties added to specific symbol descriptions in the TMC file. [▶ 166]
	VariableSymbolMapping	Mapping between variable names and ADS symbol names	Identical	Defines the TwinCAT symbol names for the generated C/C++ variables. 'Identical': Symbol name equals variable name, 'Classic': Use symbol names known from TE1400 Release 1.2.x.x [▶ 143]
TC TcCom External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [▶ 217].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode [▶ 217] connection before starting model code execution.
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [▶ 217].
TC PlcFb General	Generate	Generate TwinCAT PLC Function Block	TRUE	Generate a PLC-FB for the model [▶ 213].
	InitExceptionHandling	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during initialization [▶ 220].
	UpdateExceptionHandling	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during cyclic execution [▶ 220].
TC PlcFb License	OemLicenses	IDs of OEM License	\$(Project:OemLicenses)	IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [▶ 156]

Category	Name	Displayname	Default	Description
TC PlcFb Additional settings	MonitorExecutionTime	Monitor ExecutionTime	FALSE	Calculate and expose the execution times of TwinCAT modules as an ADS variable for monitoring purposes.
PlcFb->Interface	InputAttributes	Input variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
	OutputAttributes	Output variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
TC PlcFb External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	<u>Allow to start and stop model code execution via External Mode [► 217].</u>
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	<u>Wait for External Mode connection before starting model code execution [► 217].</u>
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	<u>Allow to change parameter online values via External Mode [► 217].</u>

4.7.2 Parameterization of the code generation via an m-file

There are two ways to parameterize the code generation via an m-file (or mlx-file):

- via model-specific Simulink® parameters with `set_param`
- via an instance of the module generator `TwinCAT.ModuleGenerator`

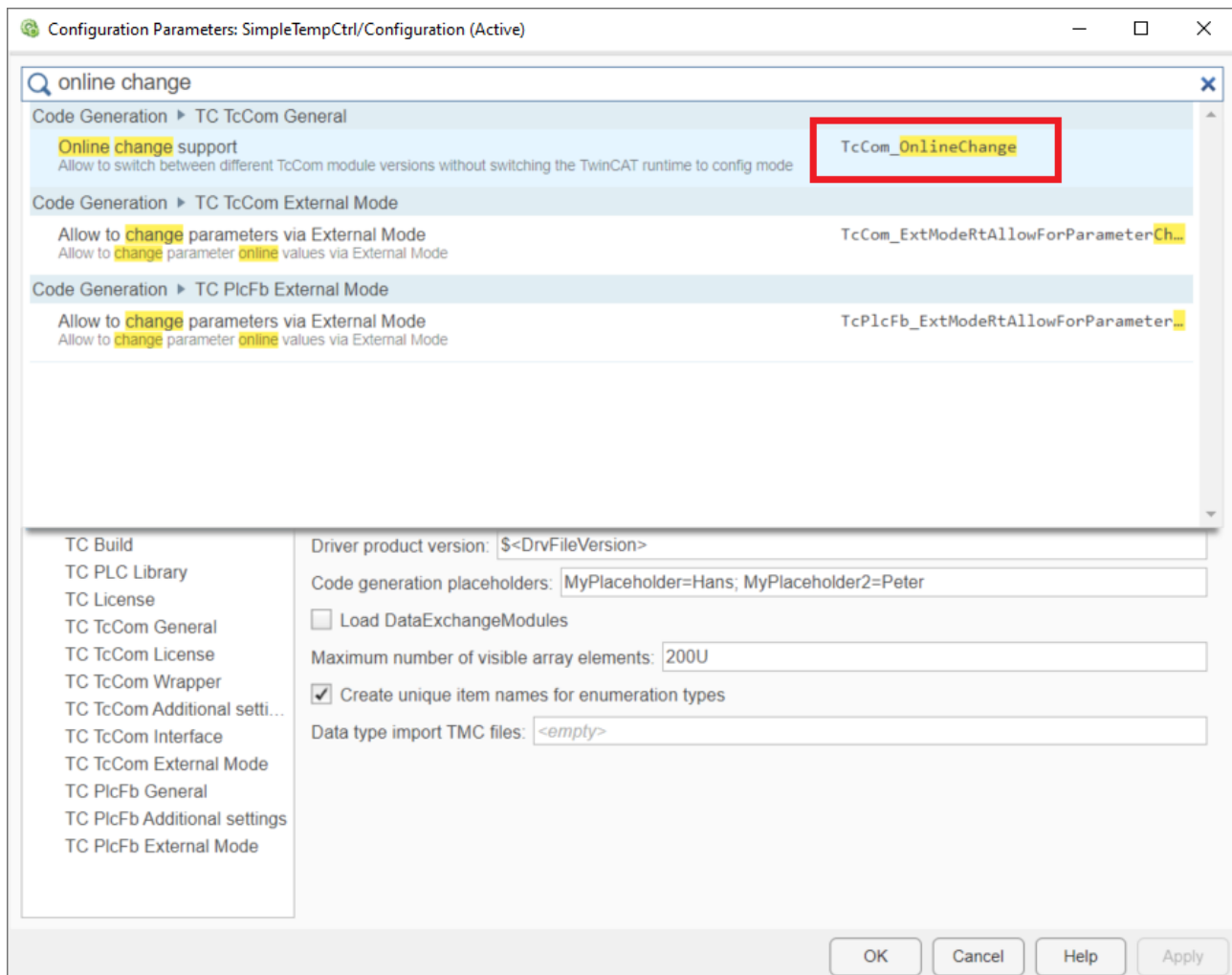
Configuration via Simulink® parameters

With `set_param` you can assign specific parameters to an object in Simulink®. If you configure the module generator with `set_param`, it will be stored accordingly in the Simulink® model.

To structure the configuration parameters [► 124] no namespace can be used here, therefore a prefix is set to the configuration parameters. The parameter name ("Name" column in the configuration parameter [► 124] table) is preceded by `Project_`, `TcCom_` or `TcPlcFb_` depending on the level.

- `Project_` prefix: For all parameters that are grouped in the presentation of the configuration parameters in the tabs TC General, TC Build, TC PLC Library and TC License.
- `TcCom_` prefix: For all parameters grouped in the presentation of the configuration parameters in the tabs TC TcCom.
- `TcPlcFb_` prefix: For all parameters grouped in the presentation of the configuration parameters in the tabs TC PlcFb.

You can also find out the exact parameter name by using the search function in the Simulink® model.



Sample

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

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

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

% build the model
slbuild(controller);

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

Configuration via the module generator

Parameter settings like in the sample above can also be made via the module generator. Thus, the settings do not remain in the Simulink® model, but in the instance of the module generator.

For this purpose, only the SystemTargetFile is defined for the Simulink® model and the parameter TcProject_Generate is switched off. This will only generate *Code Artifacts*, but no TwinCAT C++ project will be derived and accordingly not compiled. The code artifacts are stored in the current MATLAB® path in the folder <ModelName>_tcgrt.

i Export of the block diagram

At the level of the Simulink® model, you should also decide whether you want to export the block diagram. In the following steps you will work on the *Code Artifacts* and no longer on the Simulink® model.

You can specify the *Code Artifacts* folder afterwards to load a ProjectExport configuration to the TwinCAT.Modulgenerator. Here you can then make your settings and compile the project.

Sample

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

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

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

% create code artifacts
slbuild(controller);

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

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

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

% show complete configuration in MATABL Command Window
controllerCfg

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

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

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

Application examples

The separation of the code generation process into the steps (1) creation of the *Code Artifacts* and (2) configuration of the module generator and creation of the TwinCAT objects, can be helpful in different scenarios:

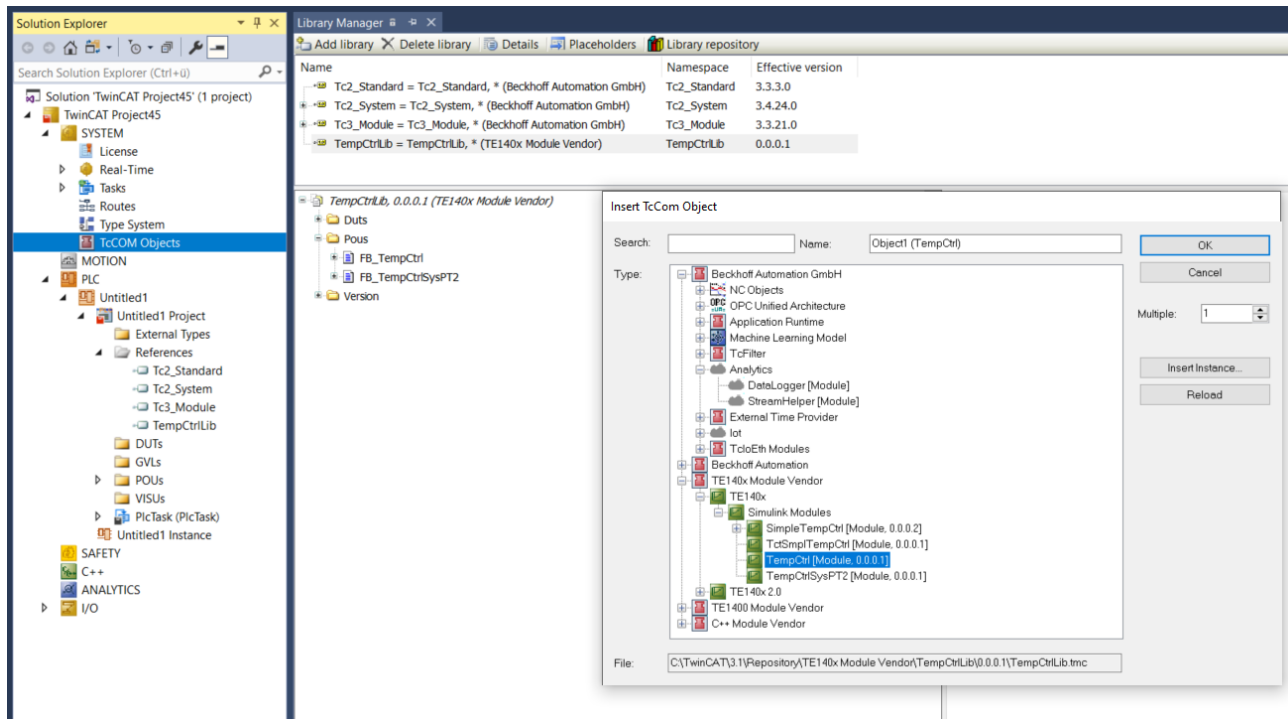
- You do not want to save the module generator settings in each Simulink® model.
- You want to merge several Simulink® models into one project. [▶ 133] This combines all models in one driver and in one PLC library. Sample:
TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules')
- You are working with a Build Server or with a CI/CD system. Sample:
TwinCAT.ModuleGenerator.Samples.Start('Continuous Integration')

4.7.3 Bundling of several models in one TwinCAT driver

Automatically generated code from Simulink® models and MATLAB® functions can be bundled into a single C++ project. After the build process, all bundled objects are then available in one driver.

Advantages of bundling

When a PLC library is created, all created objects are then listed as a function block (FB) in this library. Although only one driver and one tmc file are created, all modules can still be instantiated individually at **System > TcCOM**, i.e. from the user's point of view in TwinCAT XAE nothing changes with regard to the use of the TcCOM objects. Using the PLC library increases the clarity.



Advantages of bundling in one driver:

- The number of files in the repository directory is significantly reduced. This also means that fewer files have to be copied to other engineering systems in order to make a large number of modules available on engineering systems.
- The management of different versions is simplified, as interacting modules can be exchanged in a bundle, so that no version conflicts arise.

NOTICE

Simulink Coder™ does not support namespaces

If data types or functions with the same naming are defined in multiple models, the build process fails because the definitions are in the same namespace.

Procedure

1. Disable "Run the publish step after project generation" in Simulink®. This will abort after the code generation and the created C++ project will not be compiled.
2. When bundling multiple modules, simply use the generated folders `<modelName>_tcgrt`, which are placed in the current MATLAB® path.
3. Load, bundle and configure export configurations (`<modelName>_tcgrt`) in a project using ModuleGenerator.
4. Create an export project.

In the following, this is exemplified by the bundling of 2 export configurations:

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

```

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

Sample code in MATLAB®



Open the appropriate sample with:

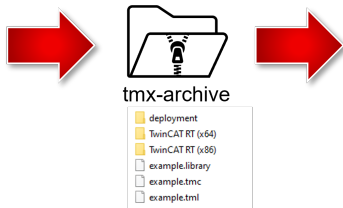
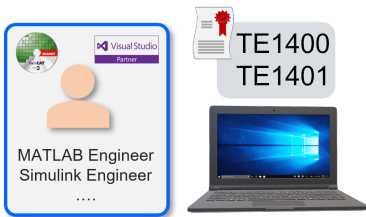
```
TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules')
```

4.7.4 Sharing created TwinCAT objects

Often the colleagues who create TwinCAT objects (TcCOM and/or PLC library) from Simulink® or MATLAB® are not the ones who implement the created modules into a TwinCAT configuration and create the machine code.

In order to be able to work with the created TwinCAT objects in TwinCAT XAE, they must be available in the repository folder on the local engineering PC and the PLC library must be installed in the local PLC Library Repository. Manual copying to engineering PCs is error-prone. It is therefore strongly recommended to create a so-called *TMX archive* and share it with colleagues who want to use the created modules.

Code generation and build



Use build models in TwinCAT Solution

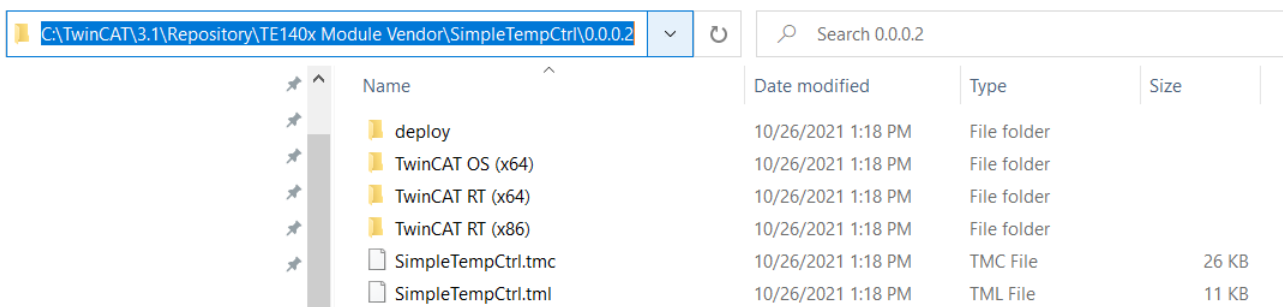


What is a TMX archive?

The TMX archive is an archive of all necessary files, which are required for the use of TcCOM and PLC libraries in a TwinCAT Engineering. The archive contains all compiled drivers, description files (e.g. for the block diagram in TwinCAT), the tmc file for TcCOM or the tml or library file for the PLC.

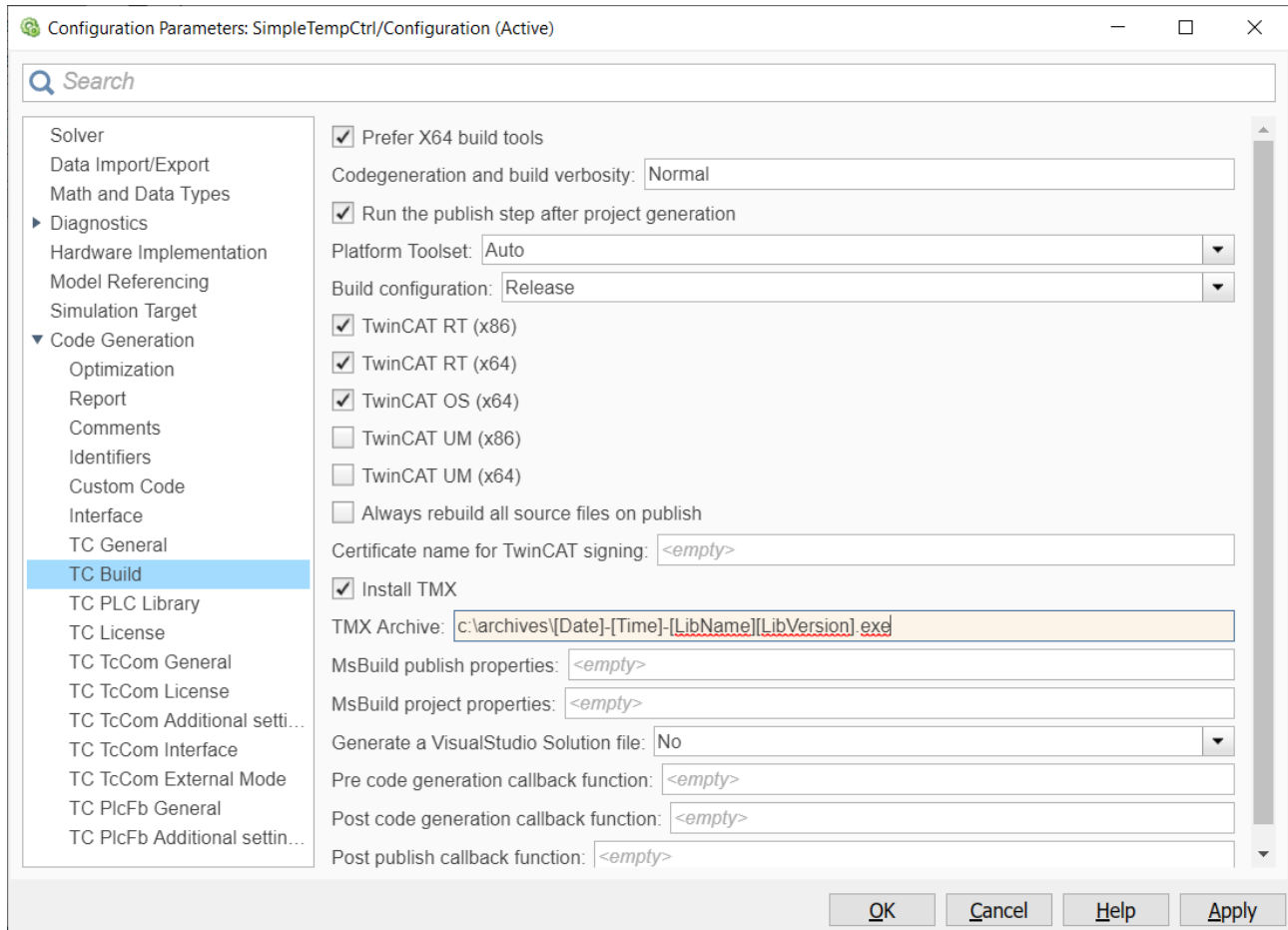
A TMX archive only needs to be copied to any path on an engineering PC and executed. It is a self-extracting archive, which then automatically copies all files to the correct location.

The files of the model *SimpleTempCtrl* in version 0.0.0.2 from the Vendor TE140x Module Vendor must be located at this position, for example:



How do I create a TMX archive?

You can specify the path and name of the TMX archive under TC Build to have it created with the next build.



You can also use placeholders for the path and name as shown in the sample above. Result of this setting is e.g. a TMX archive *2021-11-04-172921-SimpleTempCtrl0.0.0.3.exe* (new build, therefore revision incremented).

How do I use a TMX archive?

You can then copy the TMX archive to any path on an engineering PC and execute it. This will copy the files in the archive to the correct location in your repository. (self-extracting zip). If you want to use not only the TcCOM modules in the archive on the engineering PC, but also the PLC library, this must still be installed via the PLC - Library Repository.

Alternatively, you can also use the **Command Prompt**. Call the help of the **TmxPackageInstaller** via:

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



```

Command Prompt
C:\models>2021-11-04-172921-SimpleTempCtrl0.0.0.3.exe /?
TmxPackageInstaller (Version 0.5.0.0)
*****
*** Adds TwinCAT Modules or versioned TwinCAT Module Libraries to the TwinCAT installation or ***
*** creates a setup package containing TwinCAT Modules or versioned TwinCAT Module Libraries ***
*****
2021-11-04-172921-SimpleTempCtrl0.0.0.3[.exe]  [/?|/help] [/noWindow] [/list] [/noprompt[:o|:s]] [/create:<PackageName>[.exe|.zip]] [/plcLib:skip|create|install] [<
path 1> [... <path N>]]

  /?, /help ..... view options
  /console ..... run in console mode
  /noprompt ..... suppress user prompt
    :o ..... overwrite existing files (default)
    :s ..... skip existing files
  /list ..... list all contained TwinCAT Module packages without installation

  /create:<name>.exe ... create a new setup executable <name>.exe containing the archives or directories (<path 1>...<path N>)
  /create:<name>.zip ... create a new ZIP archive <name>.zip containing the archives or directories (<path 1>...<path N>)

  /plcLib ..... create a PLC library from a containing TML file and optionally install it
    :skip ..... don't create a PLC library
    :create ..... create a PLC library, but don't install it
    :install ..... create and install a PLC library
    .. combined with '/create': specifies the default behaviour for the created setup executable
    .. used without '/create': forces the behaviour for the extracted setup executable or archive

  <path 1>...<path N> .. combined with '/create': archive or directory paths which will be added to the new package
    .. used without '/create': archive (.zip or .exe) paths to install (in addition to packages possibly contained in this executable itself)

  /log:<logfile> ..... create a log file

C:\models>

```

For example, to unpack a TMX archive and install the PLC library it contains in the local TwinCAT Engineering, execute the following command in the command line:

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

What software do I need on the TwinCAT Engineering PC?

On the engineering PC on which you want to use the already compiled TwinCAT objects to implement them in a TwinCAT configuration, you only need a TwinCAT XAE installation.



You do not need a full Visual Studio on this engineering PC (the XAE shell of the TwinCAT XAE setup is sufficient) nor do you need a MATLAB® installation.

In some cases it is necessary to install the so-called *DataExchange modules* on the engineering PC, so that you can use TwinCAT objects created on another system.

These cases are:

- The created module was created with the "External Mode" option.
- The created module uses the [TwinCAT File Writer \[▶ 116\]](#) or MAT file logging.
- The created module contains function blocks from TE1410 TwinCAT Interface for MATLAB®/Simulink®.

In all other cases the created TwinCAT objects have no dependency to the DataExchange modules.

The **DataExchange modules** setup is part of the TE14xx toolForMatlabAndSimulink setup and is installed by default.

The DataExchange modules setup is also copied to the following folder, so that the employee who has installed the TE14xx-ToolsForMatlabAndSimulink setup can distribute the DataExchange modules setup to the relevant colleagues.

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

Continuous Integration with the Target for Simulink® or Target for MATLAB®

Tips for setting up a CI/CD pipeline are compiled in the following sample:

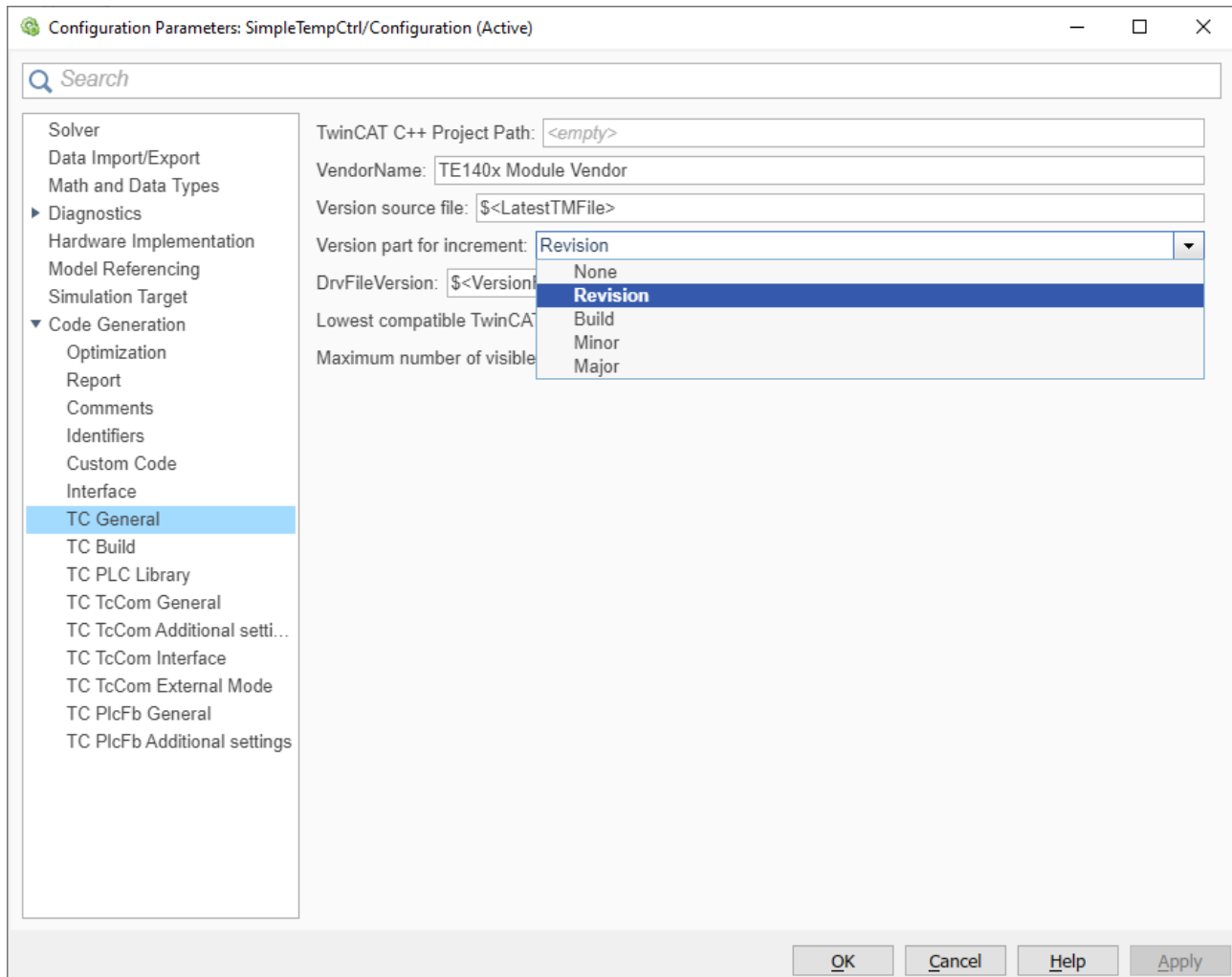
```
TwinCAT.ModuleGenerator.Samples.Start('Continuous Integration')
```

4.7.5 Creation of versioned drivers

Each object created from Simulink® contains version information. Accordingly, you can build several versions of a Simulink® model and instantiate the created modules version-selectively in TwinCAT.

Define revision control in Simulink®

Before creating a PLC function block or a TcCOM object, you can define the version of the TcCOM and the created PLC library under TC General with the entries "Version source file" and "Version part for increment".



Automatic version increments

The basic version on which a version update is to be created is specified via "Version source file". In the standard case \$<LatestTMFile> is specified there. This searches for the last available version of the model on the local engineering PC and then uses this as the basis for the version increment.

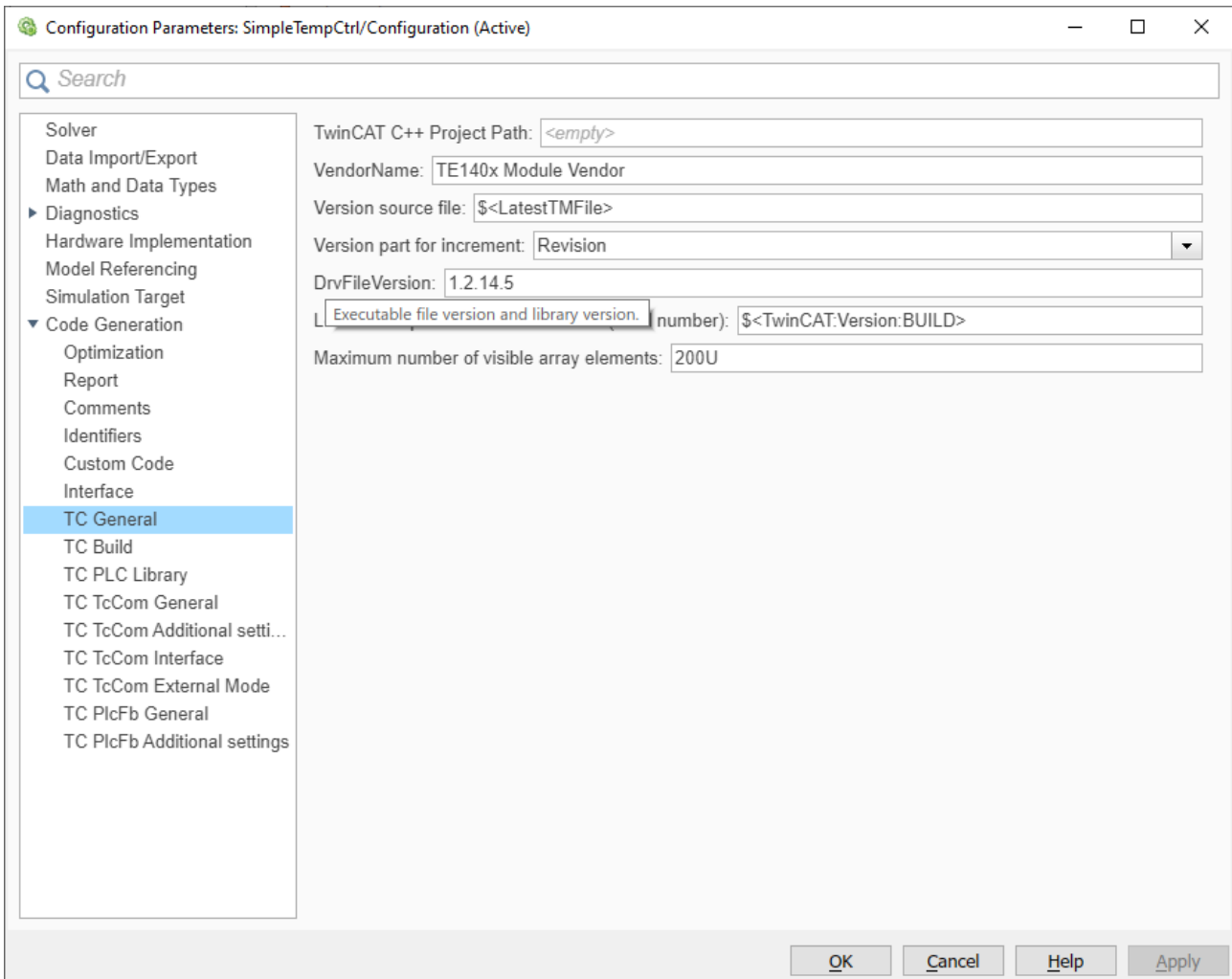
The version number consists of four digits, e.g. 1.0.3.2 or 2.12.123.14. Each digit can be incremented separately according to the scheme: <Major>.<Minor>.<Build>.<Revision>

For example, if the last version of a model named "MyModelXY" on the engineering PC is found to be 1.2.12.4 and the increment is set to "Revision", a version 1.2.12.5 is created.

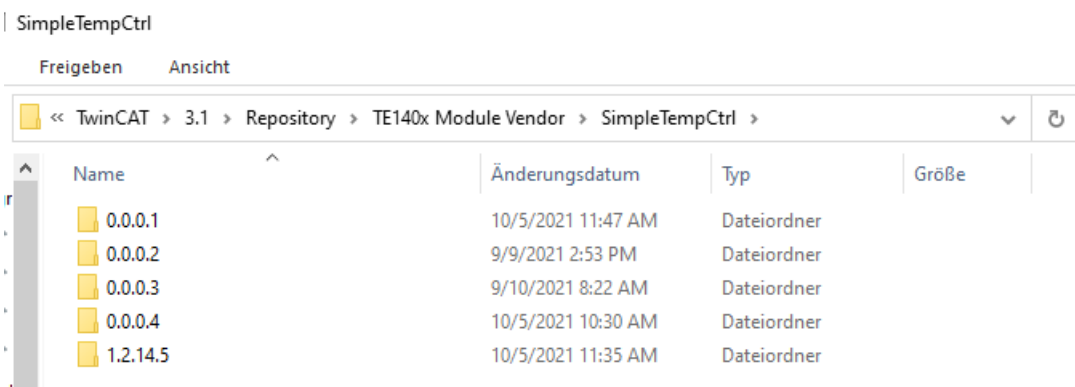
If "None" is selected, no version update takes place and the last version on the engineering PC is overwritten.

Default of a fixed version number

If a version is to be specified in Simulink®, this can be done via **DrvFileVersion**. Simply enter the target version in the input field.



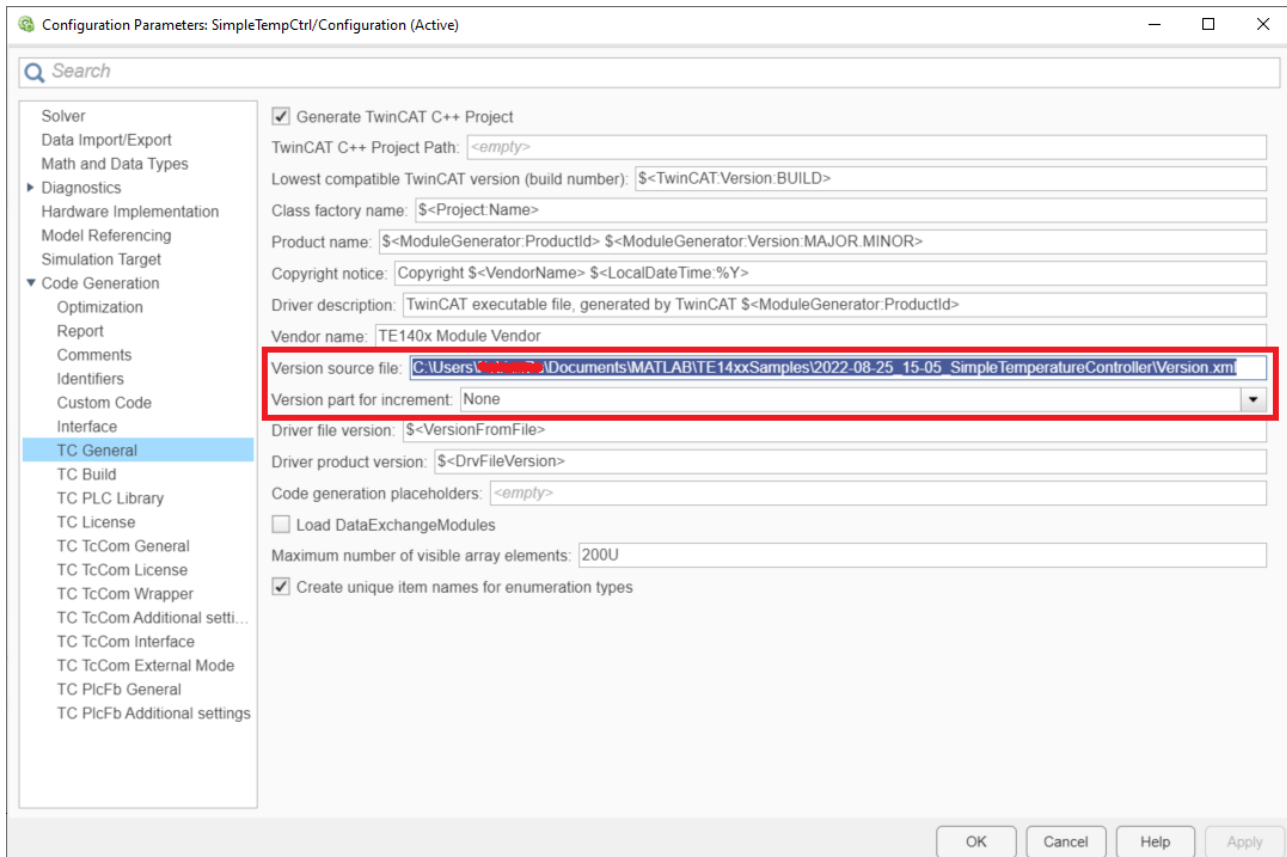
In the Engineering Repository, a folder is created under the model name for each version created. Each version folder then contains the corresponding drivers and TwinCAT files. See also [TwinCAT objects \[► 120\]](#).



Presetting of a version number via an external file

You can also use an external file to specify a version. For example, this is a common method when using build agents at *Continuous Integration*.

For configuration in Simulink® set TC General **Version part for increment** to "None" and specify the full path to your version file at **Version source file**.



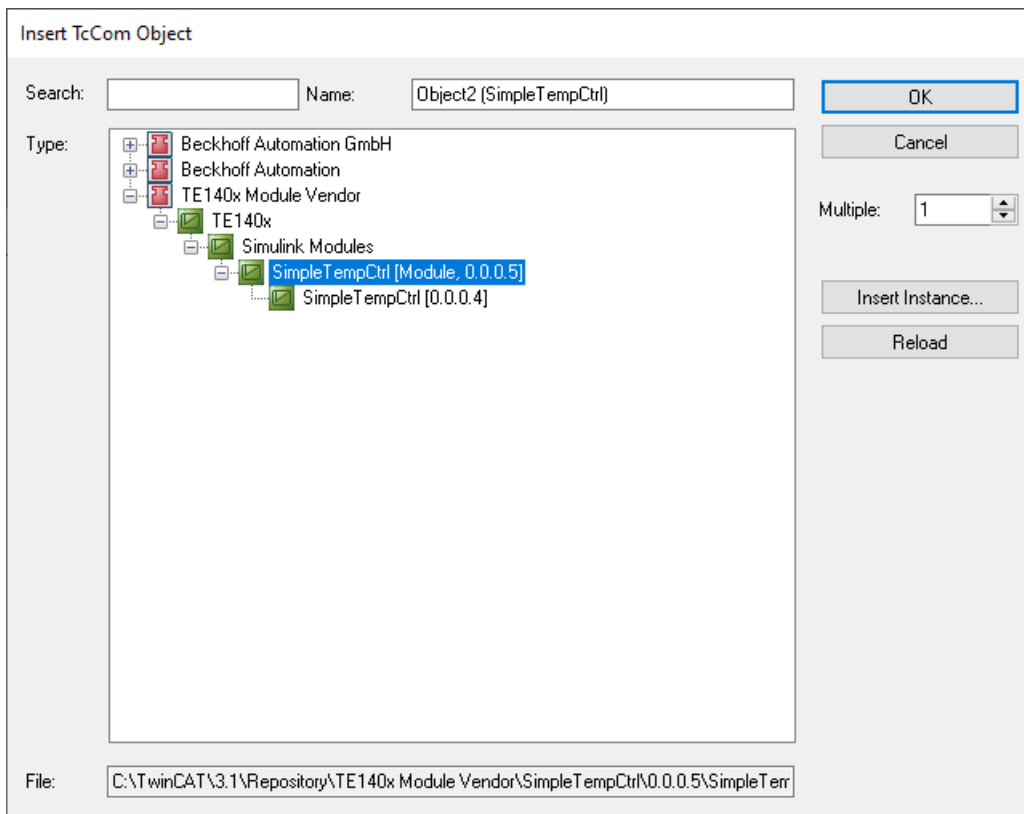
Structure of the external file:

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

Use versioned models in TwinCAT XAE

All models available in the Engineering Repository can also be instantiated in any available version. To do this, navigate through the tree as usual to find the TcCOM of your choice. In the last hierarchical level, you can now also select the version of the TcCOM.

As an example, two versions (0.0.0.4 and 0.0.0.5) of SimpleTempCtrl are available here:

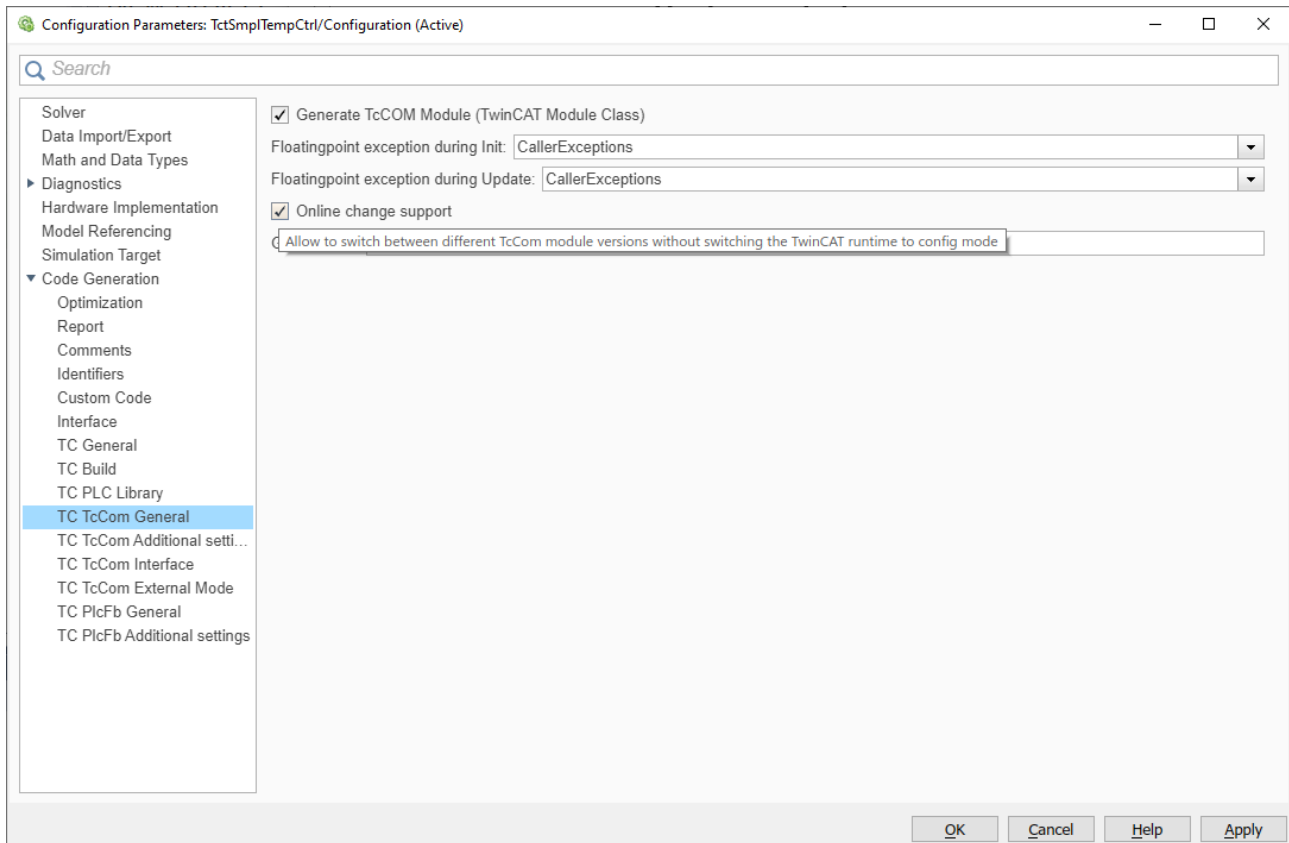


Online Change of a TcCOM during TwinCAT Run

- ✓ To switch between different versions of a TcCOM during operation, the corresponding interface must be implemented.
- 1. To do this, select the checkbox for **Online change support** under the tab **TC TcCom General** in Simulink®.

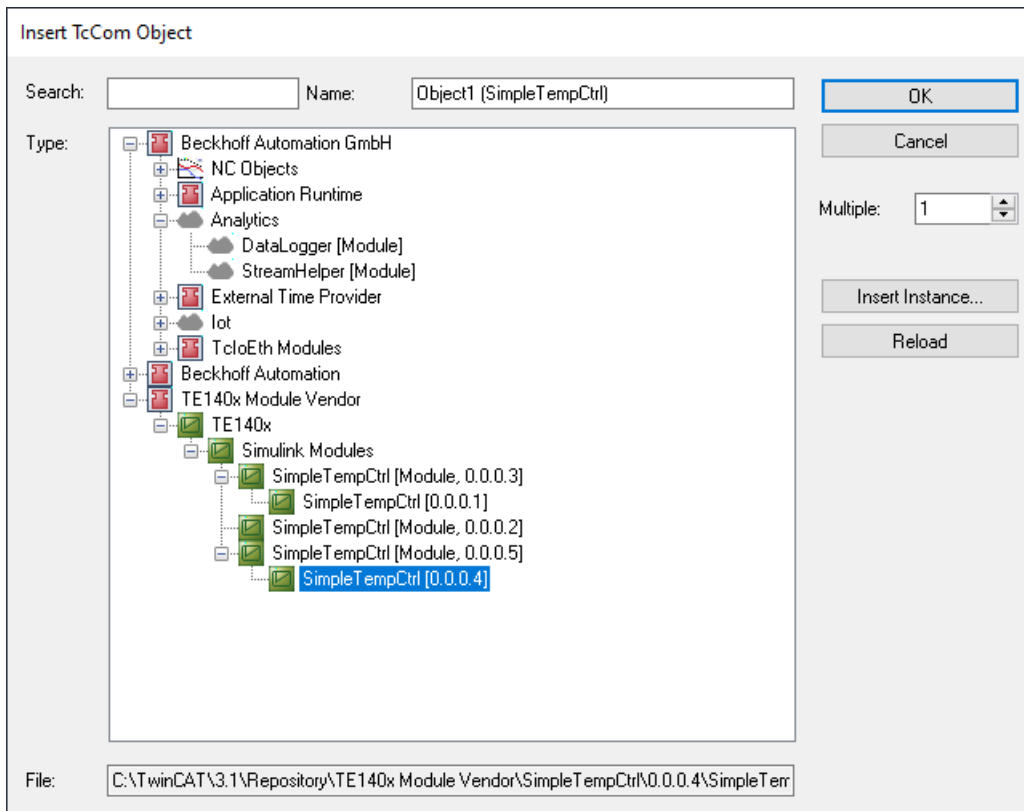
● **Online Change for PLC function block**

i If you use the function block (PLC-FB) in a versioned PLC library, you do not have to check the checkbox **Online change support**. The Online Change process then runs via the PLC-specific mechanism.

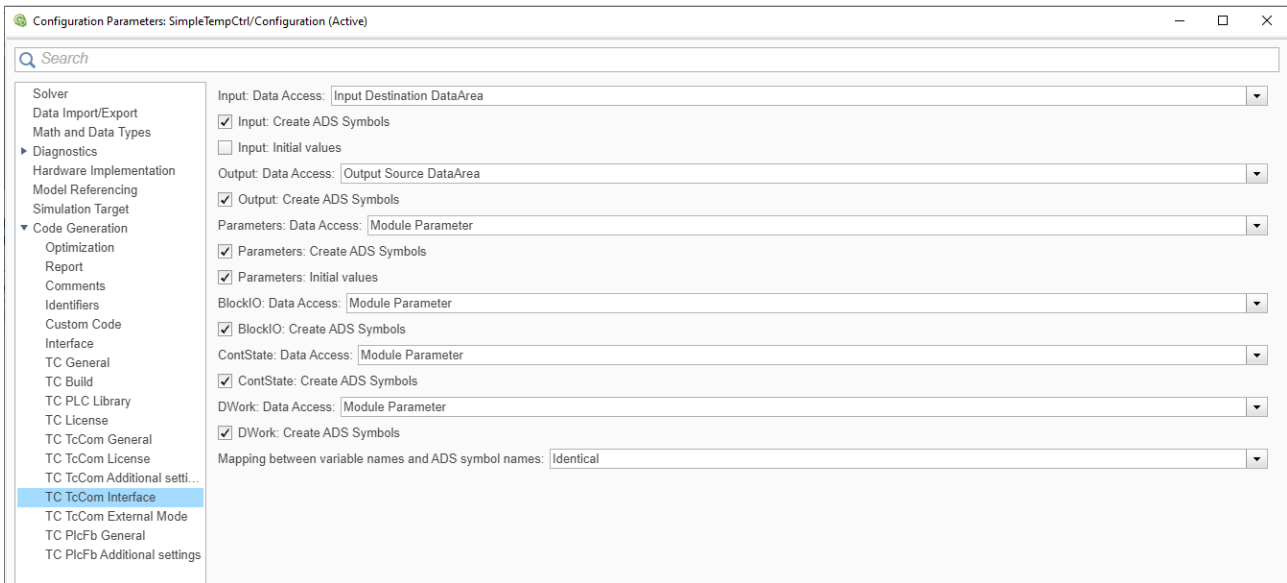


In addition, created TcCOM data areas must be compatible with each other. If **Online change support** is activated, the last hierarchical level is more strictly differentiated in the Insert TcCom Object dialog. Only Online Change compatible TcCOM are combined.

The following shows that versions 0.0.0.1 and 0.0.0.3 or 0.0.0.4 and 0.0.0.5 are compatible for Online Change. However, not 0.0.0.3 to 0.0.0.4 or 0.0.0.2.



In order to better ensure the compatibility of the Data Areas, it is possible, for example, to keep the parameters, Block I/O, ContState and DWork of a model not in an internal Data Area, but as module parameters. This means that only the inputs and outputs as Data Area are relevant for the compatibility of the TcCOM versions.



- To perform the Online Change in TwinCAT XAE, use the **tree item TcCOM Modules** and navigate to the **Online Changable Objects** tab.



- Select a version of your choice from the drop-down menu at **Online Version** (only compatible versions are displayed).
 - Right-click on the line of the object and select **Apply changed online object versions** to activate the new version of the TcCOM.
- ⇒ Details can also be found in this [TwinCAT C++ documentation](#).

4.7.6 Configuration of data access to data of a TcCOM object

In the *TC TcCom Interface* area, you configure the way in which data of certain variable groups can be accessed. ADS access and the process image type can be configured as required. These settings affect how the variables in a group are linked with other process images in the TwinCAT development environment, and how they can exchange data.

Variable groups

Depending on the Simulink® model, there are several groups of internal variables in addition to the input and output variables.

The following groups can be configured:

Group	Description	Naming of the DataArea (default)	Naming of the DataArea ("classic" option)
Input	Model inputs	<ModelName>_U	Input
Output	Model outputs	<ModelName>_Y	Output

Group	Description	Naming of the DataArea (default)	Naming of the DataArea ("classic" option)
BlockIO	Global output signals of Simulink® blocks: internal signals for which a "test point" has been defined.	<ModelName>_B	BlockIO
Parameters	Model-specific parameters: Parameters of Simulink® blocks that are <i>tunable</i> .	<ModelName>_P	Model parameters
ContState	Continuous state variables	<ModelName>_X	ContStates
DWork	Time-discrete state variables	<ModelName>_DW	DWork
DataStore	Data Store Memory	<ModelName>_DW_<DataStoreName>	<ModelName>_DW_<DataStoreName>

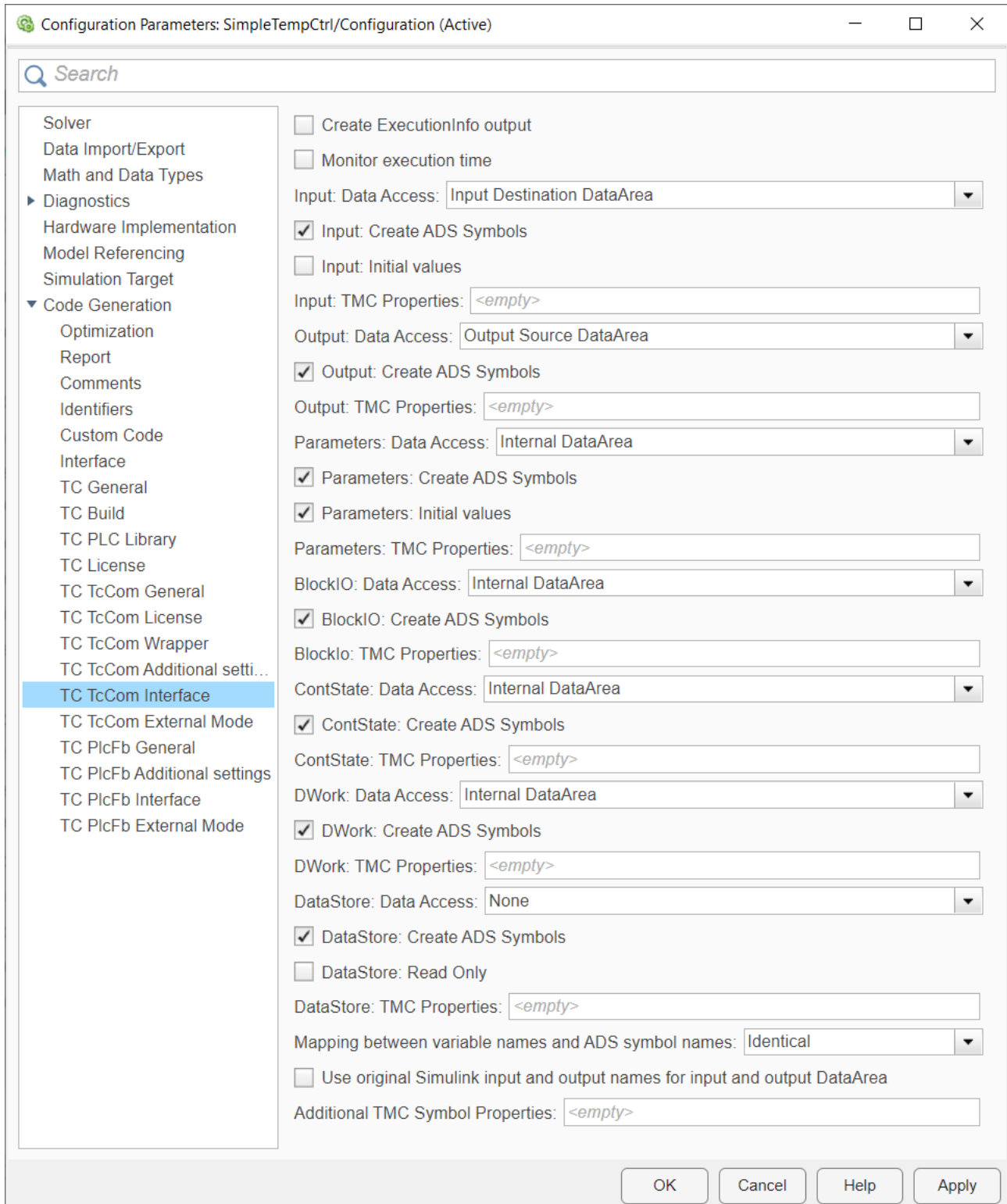
Under TC TcCom Interface, the option **Mapping between variable names and ADS symbols** can be used to influence the naming of the DataAreas. By default, they are given the same name as the associated C++ variable. This is specified by the Simulink Coder™. With the "Classic" setting, the names are abbreviated to that of the variable group, as known from earlier TE1400 versions 1.2.x. The inputs are then combined, for example, in the DataArea "Input" and not in "<ModelName>_U".

● TcCOM with multiple task contexts



If a TcCOM is created that has more than one task context (see [Multitask, Concurrent Execution and OpenMP \[▶ 159\]](#)), the DataAreas are automatically separated. There are several inputs or Dwork DataAreas, for example.

Configuration interface in Simulink®

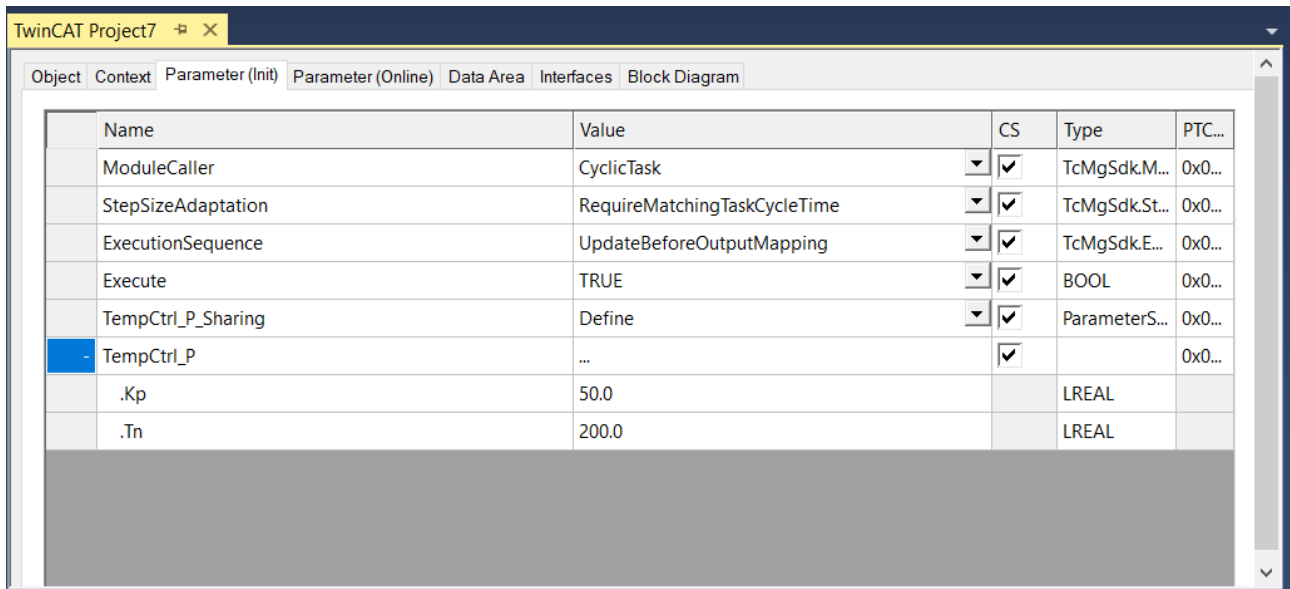


The *Data Access* can be defined individually for each variable group named above. Selection options for DataAreas and module parameters are summarized in the following table:

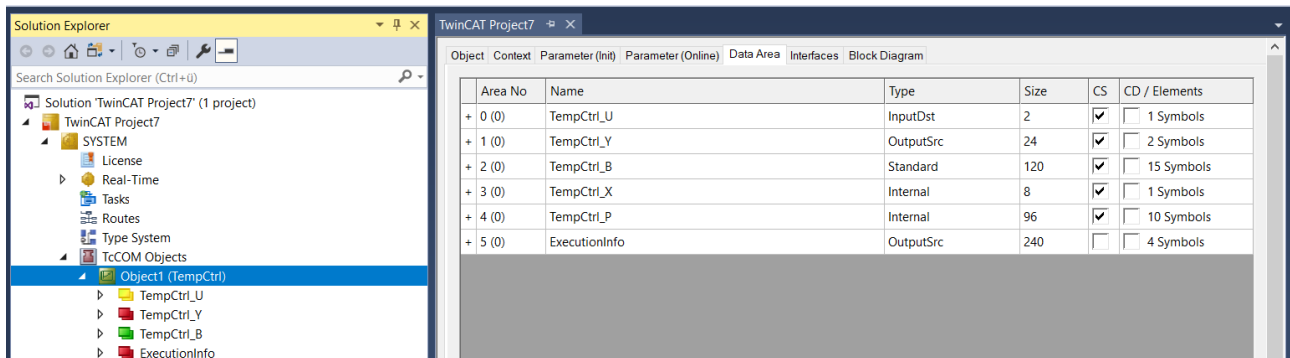
Data Access	Access via ADS	Access via mapping	Access via Data Pointer
None	No	No	No
Module parameters	Yes	No	No
Input Destination DataArea	Yes	Yes (with Output Source)	Yes
Output Source DataArea	Yes	Yes (with Input Destination)	Yes

Data Access	Access via ADS	Access via mapping	Access via Data Pointer
Internal DataArea	Yes	No	No
Standard DataArea	Yes	No	Yes
Retain Source DataArea	Yes	Yes (with Retain Handler)	No

Unlike DataAreas, **Module parameters** are not intended for cyclic (process) data exchange with other modules or I/Os. They are generally read or written asynchronously, e.g. via ADS. The parameters on the **Parameters (Init)** tab have an initial value that can be configured in the project and is written when the TcCOM instance is started. The parameters on **Parameter (Online)** on the other hand, have no configurable initial value and are typically used to monitor internal states.



DataAreas can be found on the TcCOM instance on the **DataArea** tab. The exact type of the DataArea is also displayed here. Internal DataAreas are not created as process data image in the development environment.



Properties of the parameterization

The **access via ADS** is usually **ReadOnly**. Only the Inputs and Parameters groups can also be written via ADS.

Independently of this, it is possible to set for each group whether ADS symbol information (**Create ADS Symbols**) should be created for the respective group. If no ADS symbol information is available, the data can only be accessed by ADS via IndexGroup and IndexOffset.

Via the option **"Input: Initial values"** you get an entry under "Parameter (Init)" in addition to the DataArea (for the cyclic data exchange). This way initial values for inputs can be realized. These values apply if the corresponding inputs have not been linked. Without these parameters, unlinked input values are always 0.

Via the option **"Parameter: Initial values"**, the parameters of the Simulink® model are also created as module parameters (structure <ModelName>_P) in addition to the DataArea setting.

● Special case: DataArea for model parameters

i If the [code interface is set to Reusable function \[▶ 191\]](#) (default), then no DataArea is created for model parameters, since the parameters are kept global for this case and thus shared between multiple instances. If "Parameter: Initial values" is selected, the parameter "<ModelParameterName>_Sharing" is created in addition to the entry for the model parameters at **Parameter (Init)**. This parameter can only be set to "Define" for an instance of this TcCOM module class.

All further instances must use "Inherit". Since all instances share the parameters in this case, they all work with the parameter set in the "Definer". The model parameter settings of the other instances are ignored.

See also [Parameterization of several module instances \[▶ 191\]](#).

Monitor Execution Time: Creates a module parameter at **Parameter (Online)** and in the block diagram (right side). The execution time of the TcCOM can be read via this parameter. The measurement is updated with each call of the TcCOM, so that, for example, the execution time can be tracked precisely for each call via the TwinCAT Scope.

Create ExecutionInfo output: Creates an additional Output Source DataArea with information about the module call, see [Exception handling \[▶ 220\]](#).

DataStores: Variables from a Data Store Memory Block are stored in DWork by default. You can create an additional addressing of these variables when you create a DataArea for DataStores. No DataArea is created by default. For an example, see [Shared memory between TcCOM instances \[▶ 150\]](#).

Further Descriptions:

- Structure of a TcCOM: [Description of the TcCOM properties](#)
- [Properties of DataAreas](#)
- Using a RetainDataArea with the NOV-RAM: [RetainDataArea](#)
- You can use DataPointer e.g. via the TC Module Input and TC Module Output blocks, see [Shared memory between TcCOM instances \[▶ 150\]](#).

4.7.6.1 Best practice: access to TcCOM data

Introduction

The section [Configuration of data access to data of a TcCOM object \[▶ 143\]](#) described how to set data access to a TcCOM in TC TcCom Interfaces in the configuration parameters of a Simulink® model. In the following, **advantages and disadvantages** and also **application scenarios** as well as **useful hints** are given as to when which type of data access to a TcCOM may be useful.

Access via ADS

Properties

- On-demand data exchange
- Asynchronous communication
- Threadsafe
- Time of reading or writing not determined

Use Cases

- Read or write access from outside the TwinCAT runtime, e.g. from an HMI. Read and write model parameters or read internal signals.
- Access from inside the TwinCAT runtime to the TcCOM, if the writing software module (other TcCOM or PLC) is not called in the same task context. Thus, the data exchange is thread-safe. It can be used when a software module supplies parameter sets to different software modules in various contexts.

Notes

- If several parameters are to be changed or read simultaneously, an ADS sum command should be used. Otherwise it cannot be guaranteed that separate ADS read or write commands are all processed consistently in one task cycle.
- These properties also apply if the data access via the network is to be made via OPC UA and not via ADS.

Access via mapping

Properties

- Cyclic data exchange
- Threadsafe
- Time of data exchange determined

Use Cases

- Cyclic data exchange between different software modules, especially for data that changes every task cycle.

Notes

- Even values that have not been changed are copied in each task cycle. Therefore, it is advisable to keep the input and output mapping DataAreas small and limit them to the essentials. Large Simulink® bus structures as input where only a few elements change cyclically should be avoided here.
- The time of the data exchange is determined and can be specified in Simulink® under Configuration Parameters -> TC TcCom Additional settings -> Default execution sequence.
- Defining model parameters as Input Destination DataArea only makes sense if the parameters change cyclically (in which case it should first be checked whether the parameter cannot be better mapped as a model input) or the number of parameters is so small that the overhead of cyclic copying does not interfere.

Access via Data Pointer

Properties

- On-demand data exchange
- Not threadsafe
- Time of reading or writing determined (shared memory area)
- Local copy of the data in each instance
- Flexible linking of DataArea and Data Pointer in XAE

Use Cases

- Shared memory area of several TcCOM, for example for variables in the DataStore. TcCOM instances can share a common memory area with each other via Data Pointer data.



Data Pointer example

An example can be found here: [Shared memory between TcCOM instances \[▶ 150\]](#).

Exported Global/Imported external

Properties

- On-demand data exchange
- Not threadsafe
- Time of reading or writing determined (shared memory area)
- No local copies of the data in each instance
- Only modules, bundled in the same driver [▶ 133], can access the global variable.

Use Cases

- Shared memory area for several TcCOM. Also suitable for large variables.

● Example for Exported Global/Imported external



An example can be found here: [Shared memory between TcCOM instances \[▶ 150\]](#).

Special case: Interaction via the TcCOM Wrapper FB

The TcCOM Wrapper FB (see [Applying the TcCOM Wrapper FB \[▶ 209\]](#)) simplifies the interaction between PLC and TcCOM. This FB primarily enables, with little programming effort, the cyclic execution of the linked TcCOM object from the PLC code, including the exchange of input and output data. However, it also allows simple acyclic access to the object's parameters and, if required, flexible access to the DataAreas as well.

The wrapper provides the ITcADI interface (see [ADI Interface \[▶ 213\]](#)) on the one hand and (optionally) properties on the function block (see [FB properties \[▶ 212\]](#)) on the other. The TcCOM Wrapper FB should be called in the same context as the TcCOM.

Function block properties

- Access to module parameters only
- No access to DataAreas. Note: By default, model parameters are created both as DataArea and as module parameters.
- Simple access to module parameters by name
- Not threadsafe

ITcADI interface

- Efficient and flexible access to all DataAreas (also subareas of a DataArea)
- Access only via DataArea number and ByteOffset
- No type information (type cast is to be realized by the user)
- The pointer must be fetched and released every task cycle
- No access to module parameters
- Not threadsafe

Changes persist

Above, it was described how data can be changed in a TcCOM instance at TwinCAT runtime. If TwinCAT is restarted, these changes will be lost unless further recovery measures have been taken. The default behavior of a TcCOM is that when a TcCOM instance is started, the parameterization is performed according to its **Startup Values**.

In addition, remanent variables can be defined that retain their value beyond the usual program runtime. Remanent variables can be declared as **RETAIN** variables or even more strictly as **PERSISTENT** variables.

More information about remanent data (Persistent and Retain) in the TwinCAT PLC: [Link](#)

In the following three ways are described in which values can be restored after restarting TwinCAT.

1) Startup values

See [Parameterization of a module instance \[▶ 185\]](#) for differentiation between Online, Prepared, Default and Startup Values.

● TwinCAT 3 XAE necessary



Startup values are changed in the TwinCAT configuration, i.e. in the engineering. Changes can be entered here. However, the change must be compiled and downloaded to the runtime system.

If online values are changed during runtime, you can read the current online values via ADS, for example. This can be done, for example, with the TE1410 TwinCAT Interface for MATLAB®/Simulink®. The read values can then be entered as new startup values in the TwinCAT configuration.

Use the Automation Interface example for this purpose:

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

In the example, a live script is used to show different functions. The section *Read and write parameter online values via ADS* describes how to read and write online values from a TcCOM. The section *Read and change parameter startup value* describes how to read and also write startup values.

The new startup values are entered in the TwinCAT configuration on the engineering system. To make the changes available on the runtime system, you must activate the configuration on the system. You can also realize this with the Automation Interface (`sysManager.ActivateConfiguration`). You can either restart the system directly via Automation Interface (`sysManager.StartRestartTwinCAT`), or you can let the system run without restarting. At the next start of TwinCAT, the startup values are set and the TcCOM starts up with the new startup values.

2) Retain Data

At [TC TcCom Interfaces \[► 143\]](#) you can create the model parameters (and also other groups) as DataArea of type Retain Source DataArea. You can then create a retain handler in the TwinCAT configuration and connect it to the DataArea. For details see [NOV-RAM and Retain Handler](#).

● NOV RAM necessary

I To work with the Retain Handler, you need an IPC/CX with built-in NOV-RAM. The use of an EtherCAT Terminal with NOV-RAM (e.g. EL6080) is not supported.

3) Persistent Data

You can access persistent data in the TwinCAT 3 PLC without TwinCAT XAE and without NOV-RAM.

● TwinCAT 3 PLC runtime required

I To use persistent data, you must declare variables as persistent in the PLC.

Persistent data can only be created in the PLC. Therefore, use the PLC to define the data you want to persist. Work with the [TcCOM Wrapper FB \[► 209\]](#), for example, to read data from the TcCOM and persist them via the persistent PLC variable. Create a state machine for starting the TcCOM object after a TwinCAT restart in order to write the persisted data back into the TcCOM object before it is called again in the code.

4.7.7 Shared memory between TcCOM instances

In some applications it may be advantageous for TcCOM instances to share a memory area, so that certain structures/variables are defined once in an object and all other objects reference this memory location.

To achieve this, you can follow two different paths in TwinCAT:

- [Linking TcCOM instances in TwinCAT with data pointers \[► 150\]](#)
- [Use of global variables \(Exported Global/Imported External\) \[► 154\]](#)

In the remainder of this chapter, both paths will be described.

While with data pointers you are more flexible in linking the data, when using global variables you have the advantage that the structure/variable exists only once in memory. For data pointers, a local copy of the data is held in each TcCOM instance.

NOTICE

Data transmission not threadsafe

Use data pointers and global variables with caution. The data exchange is not threadsafe. Therefore, we strongly recommend to run all involved TcCOM instances in the same task context.

Linking TcCOM instances in TwinCAT with data pointers

The following describes how you can specifically use a shared memory area between TcCOM instances via data pointer.

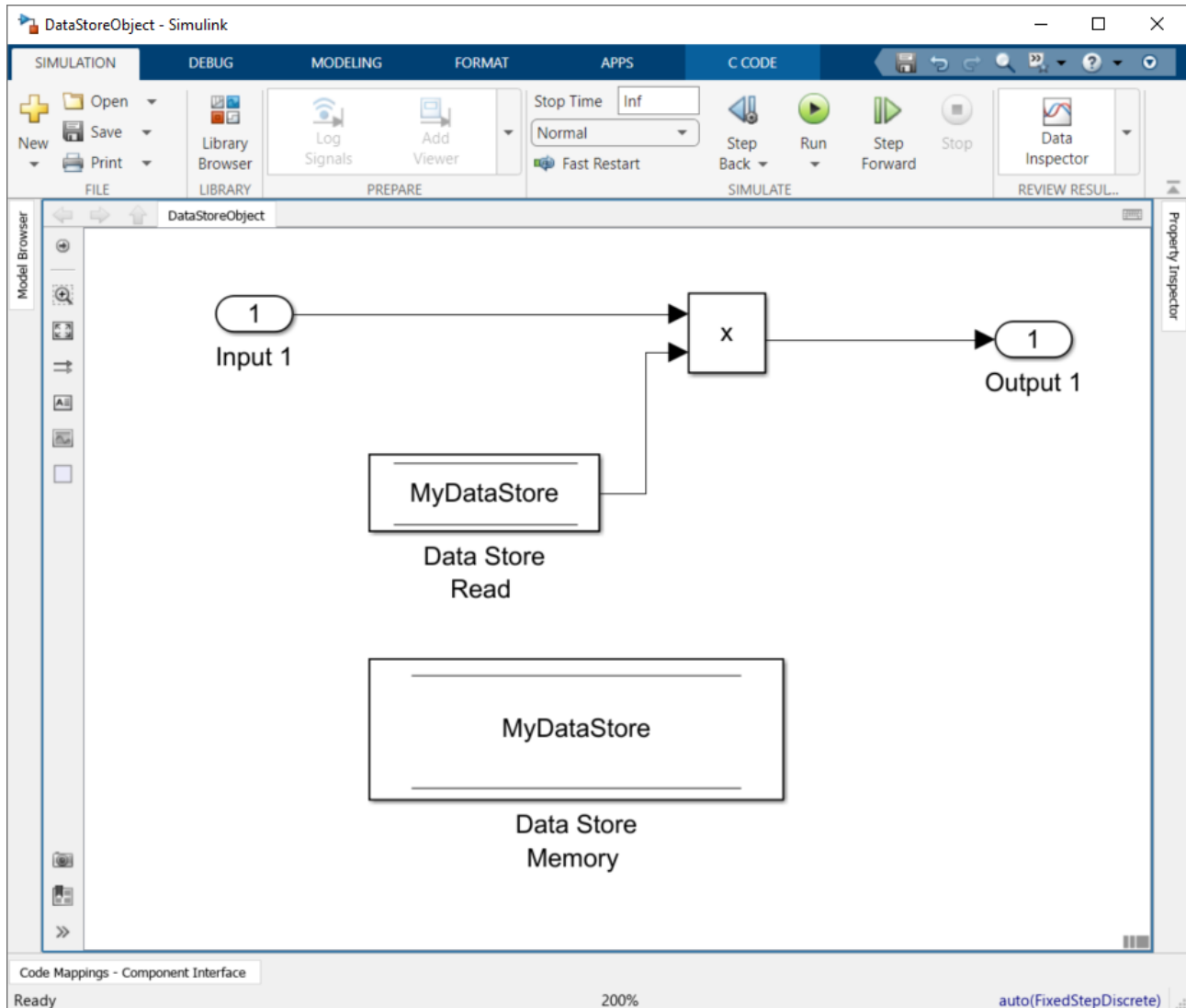
As described in section [Configuration of data access to data of a TcCOM object \[▶ 143\]](#), you can make DataAreas accessible via DataPointer. Input Destination DataArea, Output Source DataArea and Standard DataArea are accessible via DataPointer. In principle, you can create any variable group, model parameter, DWork, BlockIO, etc. as a standard DataArea and thus make it accessible as Data Pointer.

Objective

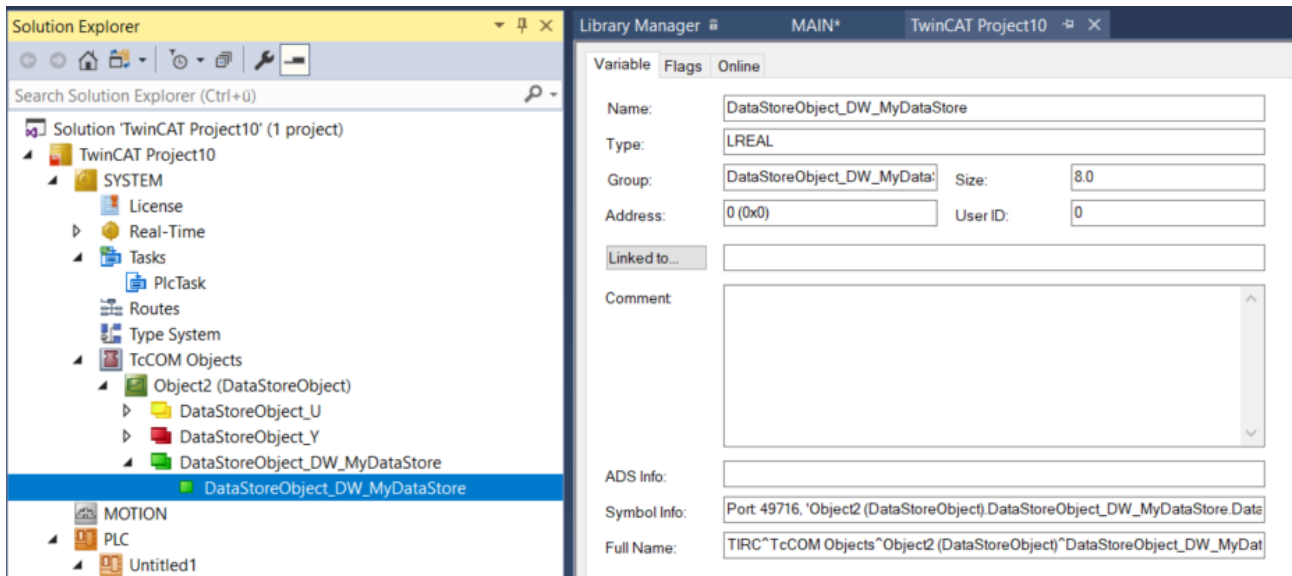
In the following, we will show you how to make a specific subrange of variables accessible via DataPointer and not the entire variable group range at once. The basic concept is based on the use of data store memory blocks in Simulink®.

Model with DataStore

A Data Store Memory block is created in a model "DataStoreObject". In this sample with only one variable of type double. This variable is read with a Data Store Read, multiplied by an input and set to an output.



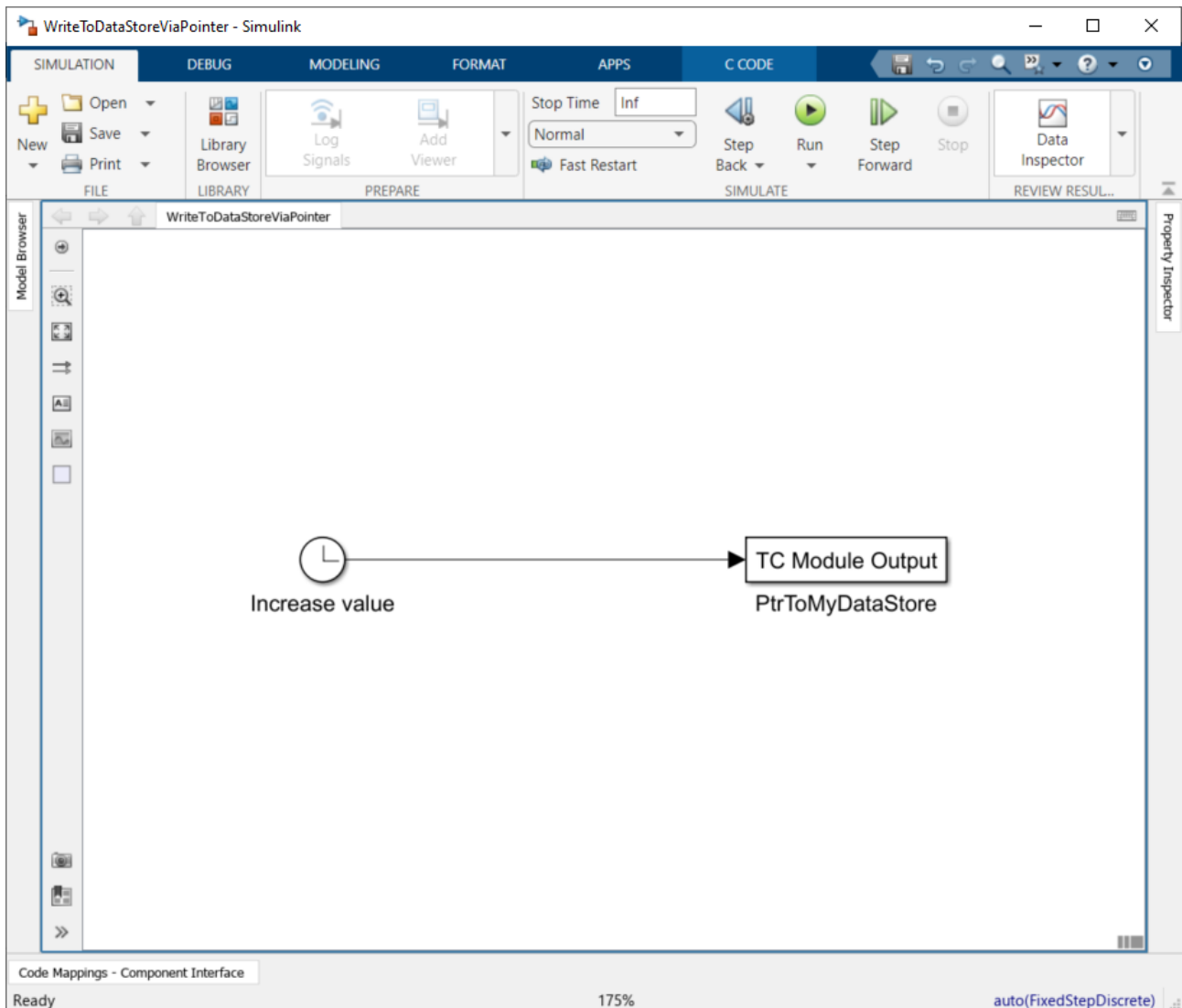
At TC TcCom Interface, the option *DataStore: Data Access* is changed from None to *Standard DataArea* and the Simulink® model is compiled into a TcCOM. If the object is instantiated in TwinCAT, the following representation is obtained:



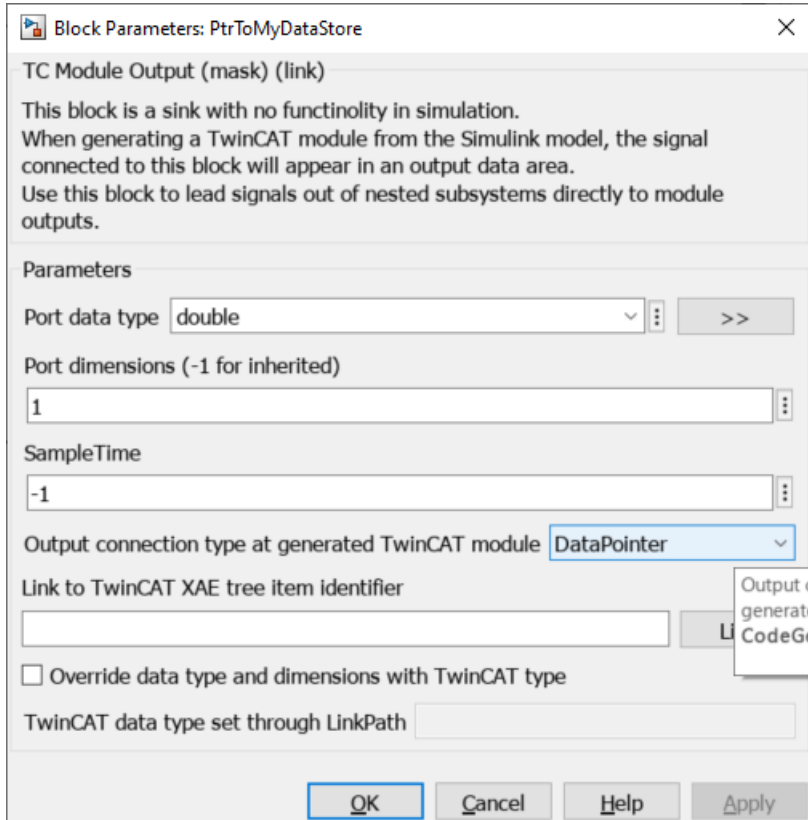
The DataStore DataArea is now displayed in the process image and contains a variable of type LREAL, which can be accessed via DataPointer.

Model with DataPointer

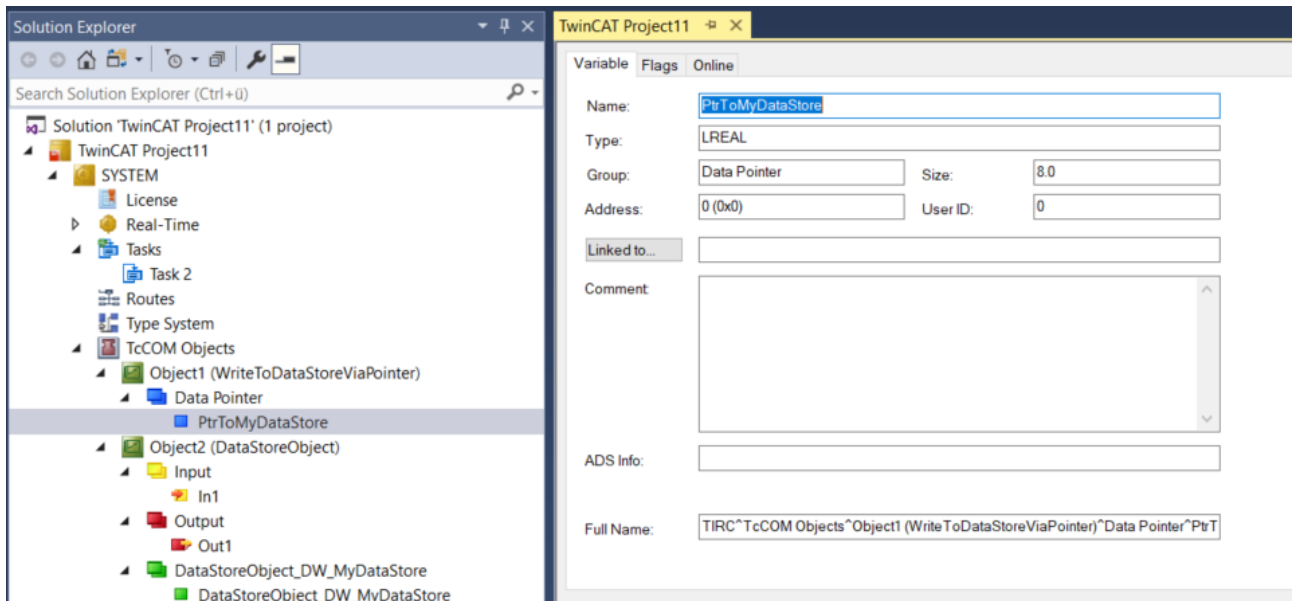
In a second Simulink® model, a double value is generated by a clock and this is set to the TC Module Output [▶ 109].



In the configuration of the TC Module Output, the connection type is set to DataPointer and the data type is set to double.



If this model is compiled into a TcCOM object, the following representation is obtained in TwinCAT:



The name of the TC Module Output "PtrToMyDataStore" of type LREAL is now displayed under Data Pointer. This can now be linked by double-clicking on the DataStoreObject_DW_MyDataStore.

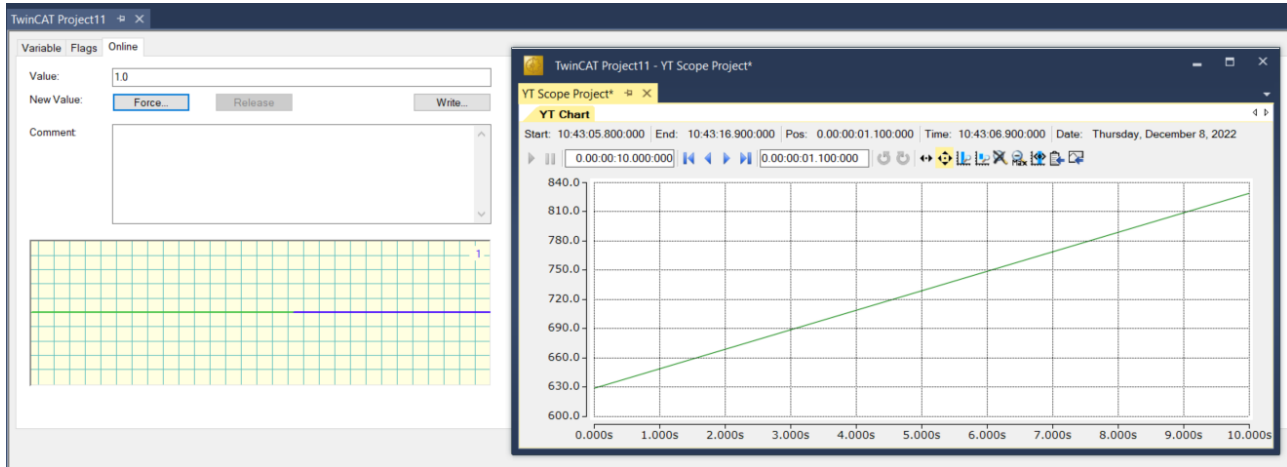
Both TcCOM instances should be called in the same task because the communication is not threadsafe.



If different tasks are used, the user must ensure that data is consistent and is read or written at appropriate times.

The behavior of the sample here is shown in the graphic below.

If the input In1 of the DataStore model is manually set to 1 and the output Out1 is observed via the TwinCAT Scope, an increasing straight line is seen. This means that the increasing value from the clock is written to the DataStore of the DataStoreObject-TcCOM via pointer.



● Read access via TC Module Input

I Read access to a DataArea can be realized via DataPointer using the TC Module Input.

● Local data copy in each module

I The Simulink Coder™ generates the C/C++ code in such a way that local variables are created for the TC Module Input and TC Module Output and thus each module instance contains a local copy of the data. Accordingly, the memory requirements of the project increase with each instance.

Use of global variables (Exported Global/Imported External)

The following describes how to create a global variable in TcCOM using the Storage Classes in Simulink® and reference it in other TcCOM.

Objective

A Simulink® parameter GlobalParams is created, which contains a structure of 3 elements. The goal is to define this structure via a TcCOM object and to instantiate further TcCOM objects that access this structure via shared memory area, so that changes in the defining TcCOM arrive directly in all connected TcCOM instances.

Modeling in Simulink®

Two Simulink® models are created:

1. "DataDefiner" model
2. "DataUser" model

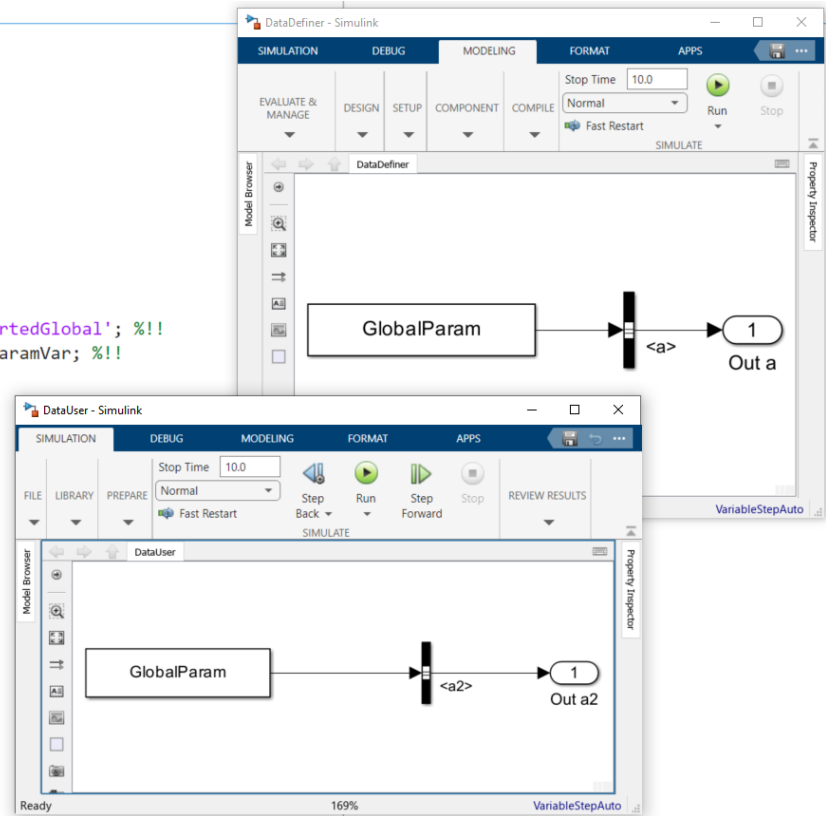
Furthermore, a Simulink® bus is defined and created as a Simulink® parameter named GlobalParam. The DataDefiner model determines the content of GlobalParam at the end, while the DataUser models only read the content.

```

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

% global variable name
globalParamVar = 'GlobalParam';

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



Note that the Storage Class definition of GlobalParam is set differently for the DataDefiner and DataUser below. For the DataDefiner, the Storage Class is set to **Exported Global**, while for the DataUser, the Storage Class is set to **Imported Extern**.

The following script bundles the two described Simulink® models into one driver, cf. [Bundling of several models in one TwinCAT driver](#) [► 133].

```

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

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

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

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

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

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

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

```

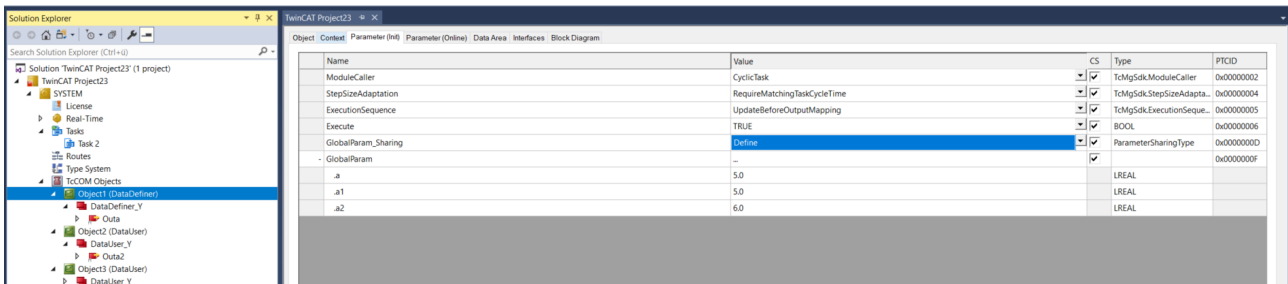
% add the class export configuration to the "global" project export configuration
mdlProjCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(codeDirectories(i));
clsCfg = mdlProjCfg.ClassExportCfg{1};
projCfg.AddClassExportConfig(clsCfg)
end

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

```

Configuration in TwinCAT XAE

- ✓ Create **one** instance of the DataDefiner in the TwinCAT XAE (more are not allowed!). On the other hand, you can create as many DataUser instances as you like.
- 1. Set **Parameter (Init) > Parameter GlobalParam_sharing** for the DataDefiner to "Define".
- 2. Set **Parameter (Init) > Parameter GlobalParam_sharing** for all DataUsers to "Inherit".
- 3. Create a task.
- 4. Assign this task to all created instances.



- 5. Activate the project.
- ⇒ Change the values of GlobalParam in the DataDefiner, e.g. via the Block Diagram, and observe the direct effect on the DataUser instances.

4.7.8 Creating a module with OEM license query

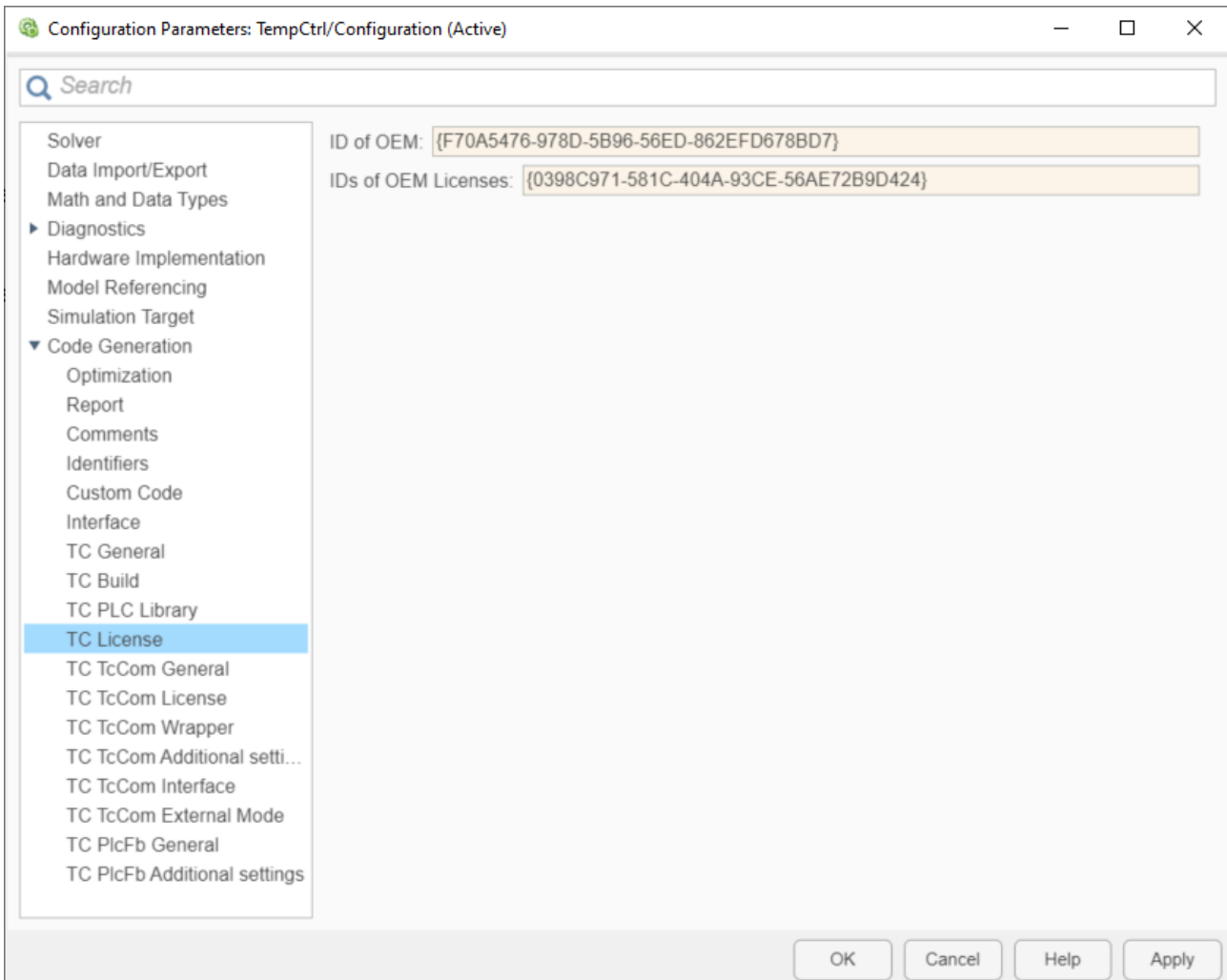
Why couple an own license to a module?

If a TwinCAT object, in addition to the TwinCAT licenses, is also bound to an OEM license, a binding of this TwinCAT object to a hardware can thereby be realized, so that the application is protected against cloning. In addition, functionalities of an application can be licensed to end customers via this route.

For more information, see [Software Protection / Own OEM Licenses](#).

Configuration in Simulink®

- Switch to the Advanced configuration level:
`TwinCAT.ModuleGenerator.Settings.Change('ConfigurationLevel', 'Advanced')`
- Enter your OEM ID and the OEM license(s) you are requesting:



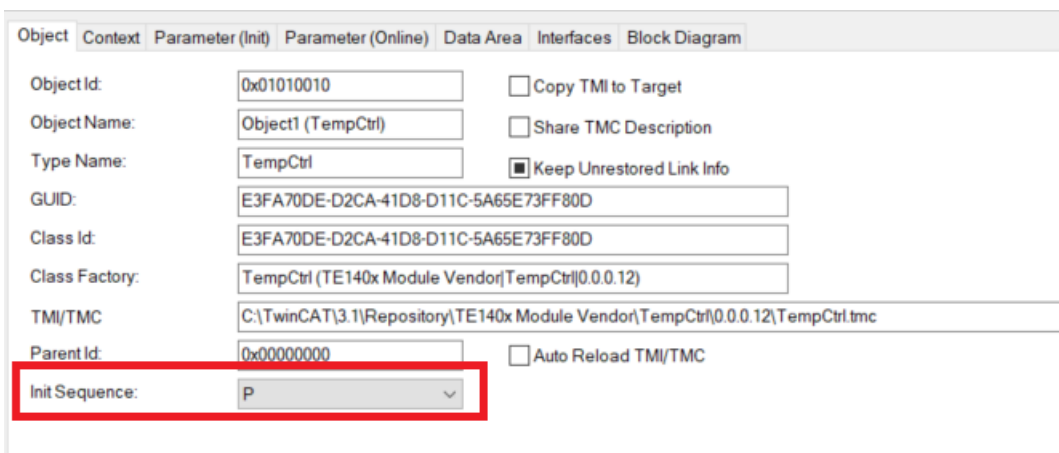
If the module is created with the above settings and instantiated in TwinCAT, a valid OEM license must be present in addition to a valid TwinCAT license (TC1220, TC1320) so that TwinCAT can be activated.

To note for license dongles

The following should be noted when using the OEM license for the *target system* on a dongle:

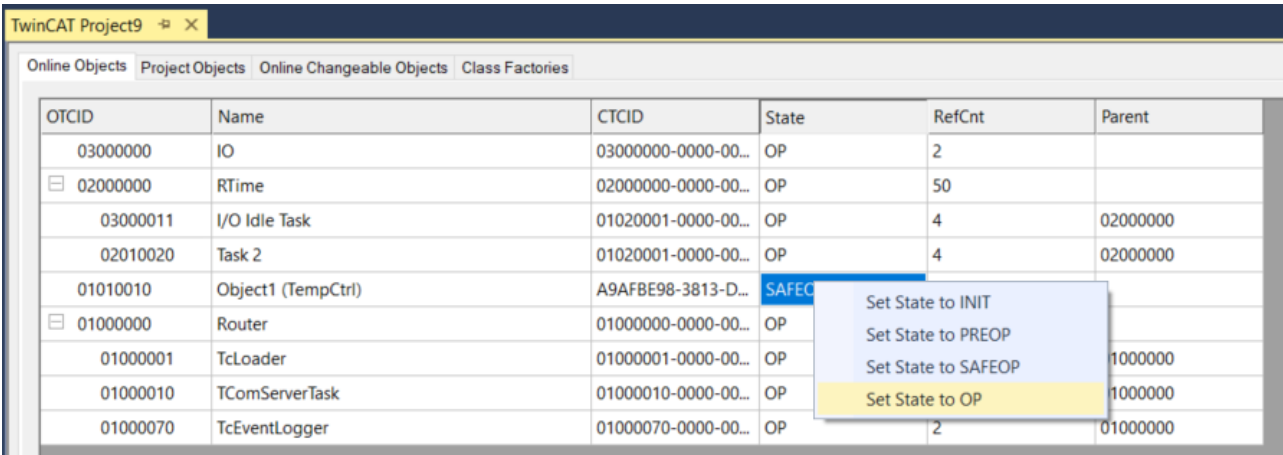
✓ **Do you use an instance of a TcCOM?**

1. Set the Init Sequence on the object instance to P.



2. Note that you cannot use active mappings in this case. It is advised to use the TcCOM wrapper FB or to call the module from TwinCAT C++.
3. Activate the configuration.

- After TwinCAT is in run mode, switch the TcCOM object to OP state, e.g. via the XAE (see graphic below), via the TcCOM wrapper FB or via ADS.



⇒ The license will be checked and accepted (if valid) when booting into the OP-State.

✓ **Do you use the TcCOM wrapper FB from the created PLC library and reference a static TcCOM instance?**

- Set the Init Sequence on the object instance to P (see above).
- Use the TcCOM wrapper FB to switch the referenced TcCOM to OP.

⇒ The license will be checked and accepted (if valid) when booting into the OP-State.

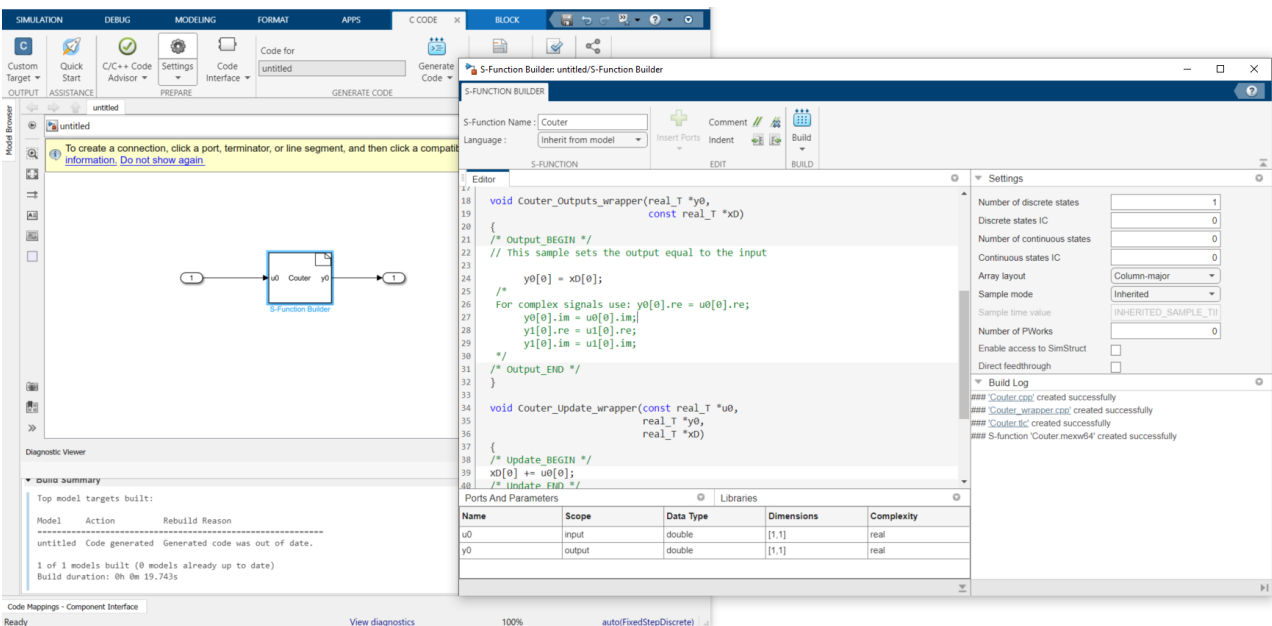
✓ **When do you not need to pay attention to anything else?**

- If you use the PLC-FB from the created PLC library.
- If you create a TcCOM dynamically with the TcCOM wrapper FB.

4.7.9 Integration of own C/C++ code

The TwinCAT Target for Simulink® also supports the integration of own C/C++ code in Simulink®. MathWorks® offers several possibilities for this, for example the way via the S Function Builder.

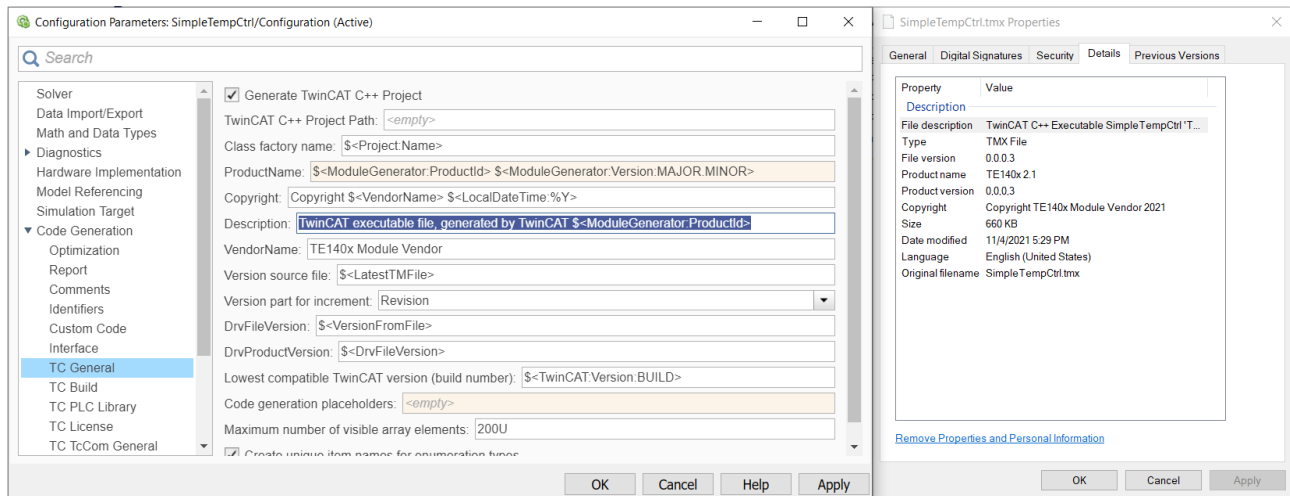
Note that you set the language to "Inherit from model". You can also include libraries as long as they are platform independent and available as source code. For example, the inclusion of a precompiled DLL is not possible.



4.7.10 Configuration of the TMX file properties

You can parameterize the entries in the TMX file (TwinCAT Module Executable) from Simulink®. To do this, switch to Advanced mode:

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



Relationship of TMX properties (left) to parameters in Simulink® (right)

File description -> Description

File Version -> DrvFileVersion

Product name -> ProductName

Product version -> DrvProductVersion

Copyright -> Copyright

Note: \$< > describes placeholders [▶ 172]. For example, DrvProductVersion is set to the value in DrvFileVersion which in turn gets the value from Version Source File.

4.7.11 Multitask, Concurrent Execution and OpenMP

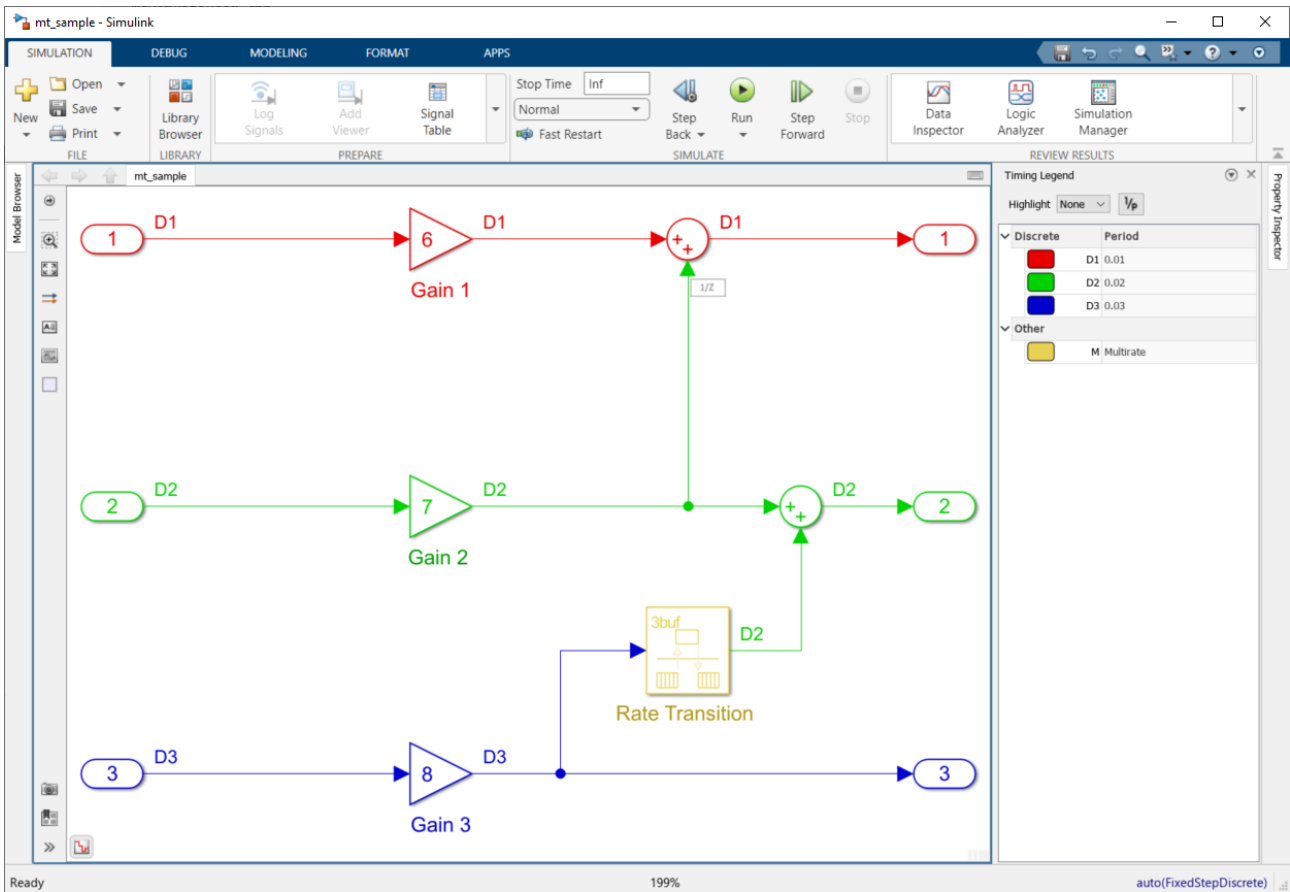
In Simulink® you can configure your models to run on Multicore Target systems. Further details can be found in the [MathWorks® documentation](#). Beckhoff targets usually offer a multi-core architecture, which can be used efficiently with TwinCAT 3. This is also possible with the TwinCAT Target for Simulink® as shown below.

A distinction is made in this description between Multitask, Concurrent Execution and OpenMP.

- With [Multitask](#) [▶ 161], a TcCOM object is created which has several tasks available. **All** tasks must run on the same core. It is not parallelized.
- With [Concurrent Execution](#) [▶ 163], a TcCOM object is also created with multiple tasks that can be distributed on different cores. Calculations can actually be executed in parallel.
- With [OpenMP](#) [▶ 164], a TcCOM object is created with a task context. In addition, multiple JobTasks distributed on different cores can execute the code fragments generated as OpenMP code in parallel.

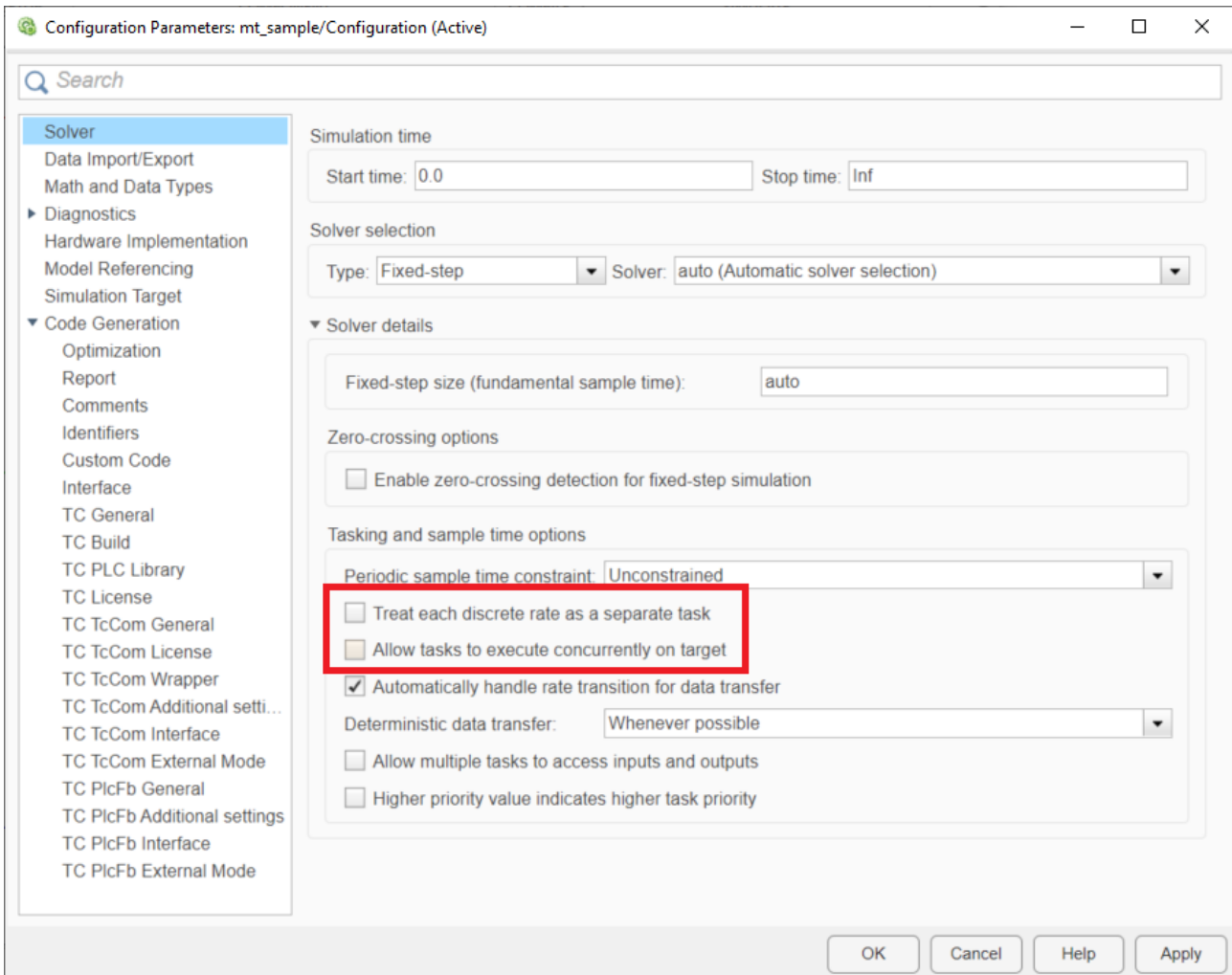
Multitask and Concurrent Execution

The following multirate system in Simulink® is considered for the descriptions of the options *Multitask* and *Concurrent Execution*. The model has an explicit and an implicit *rate transition*.



Go to **Configuration Parameters** and select **Solver**. Here you can choose between:

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



Treat each discrete rate as separate task: Multitask

If a TcCOM object is created with the *Treat each discrete rate as separate task* option enabled, you will get an object to which you can assign multiple task contexts. 3 tasks in this case.

Object Context Parameter (Init) Parameter (Online) Data Area Interfaces Block Diagram

Context: 0 'Context0 (10 ms)'

Depend On: Manual Config

Need Call From Sync Mapping

Data Areas:

- 0 'mt_sample_U1'
- 1 'mt_sample_U2'
- 2 'mt_sample_U3'
- 3 'mt_sample_Y1'

Interfaces:

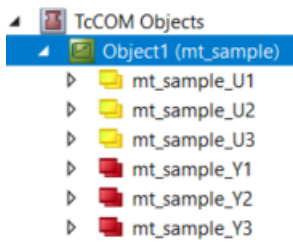
Data Pointer:

Interface Pointer:

Result

ID	Task	Name	Priority	Cycle Time ...	Task Port
0	02010020	Task 2	1	10000	350
1	02010030	Task 3	2	20000	351
2	02010040	Task 4	3	30000	352

The inputs, outputs and all other DataAreas are divided into the different contexts, so there are 3 Input DataAreas and 3 Output DataAreas in this case.



In this case, the cyclic tasks must all be placed on the same core. There is no parallel processing of the tasks.

The advantage over a TcCOM with only one task interface is that now not all calculations have to be completed within the fastest task cycle time (see Scheduling). If the above Simulink model were created with default setting without *Treat each discrete rate as separate task*, only one task with 10 ms (fastest task) would be linkable. This means that all calculations must be completed within this time. By distributing to multiple tasks on the same core, this rule is disabled because tasks can interrupt each other (see Priorities).

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

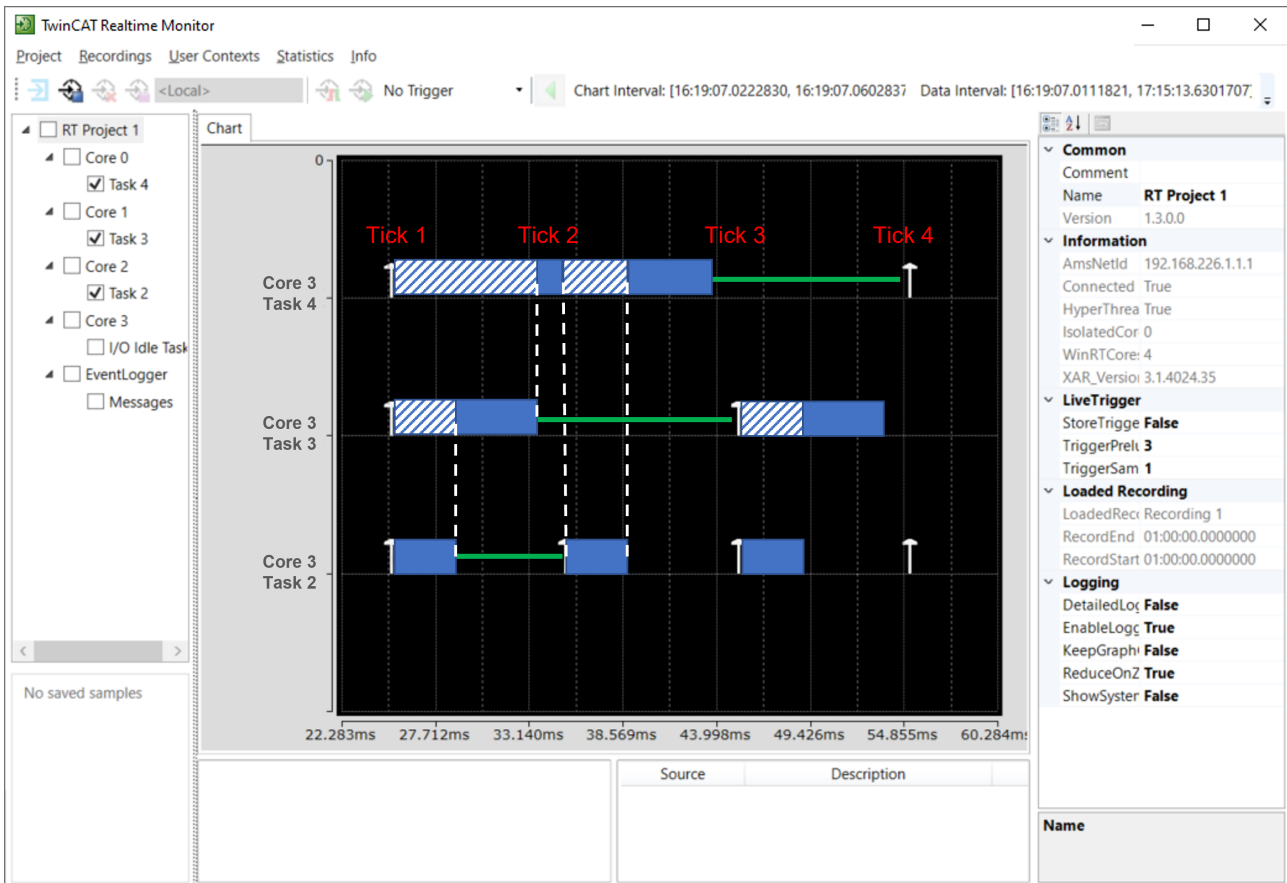
Properties:

- No function block is supported in the PLC.
- The TwinCAT Usermode Runtime is not supported.
- All tasks are assigned to the same core.
- The fastest task must be assigned the highest priority (smallest priority value). The second fastest task the second highest priority and so on.

Scheduling Details:

The graphic below describes an example of how the computing times can be distributed. The hatched areas indicate that a task may not work during this time due to a higher priority task. The full blue area indicates that the task is working. Note that the surfaces have only been subsequently overlaid on the real-time monitor image to aid comprehension and are not real images.

- Task 2, Task 3, and Task 4 are executed sequentially on the same core in Tick 1. The execution of Task 2 and Task 3 runs without interruption. The execution of Task 4 is interrupted by the higher priority Task 2 in the transition to Tick 2.
- Task 2 is executed first in Tick 2. The execution of Task 4 is resumed after Task 2 is completed.
- Task 2 starts again and Task 3 follows in Tick 3.



If cycle time overruns occur and scheduling cannot be adhered to, the execution of the respective task context is skipped until all relevant contexts are in the appropriate state. In the TcCOM object this behavior can be observed via the online parameter *SkippedExecutionCount*.

Allow tasks to execute concurrently on target: Concurrent Execution

If a TcCOM object is created with the *Allow tasks to execute concurrently on target* option enabled, you will get an object to which you can assign multiple task contexts. In this case, as in the example above, 3 tasks.

Again, the DataAreas are separated into the different contexts. The difference to the multitask object is that you can now distribute the tasks to different cores so that the processing is actually parallelized.

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

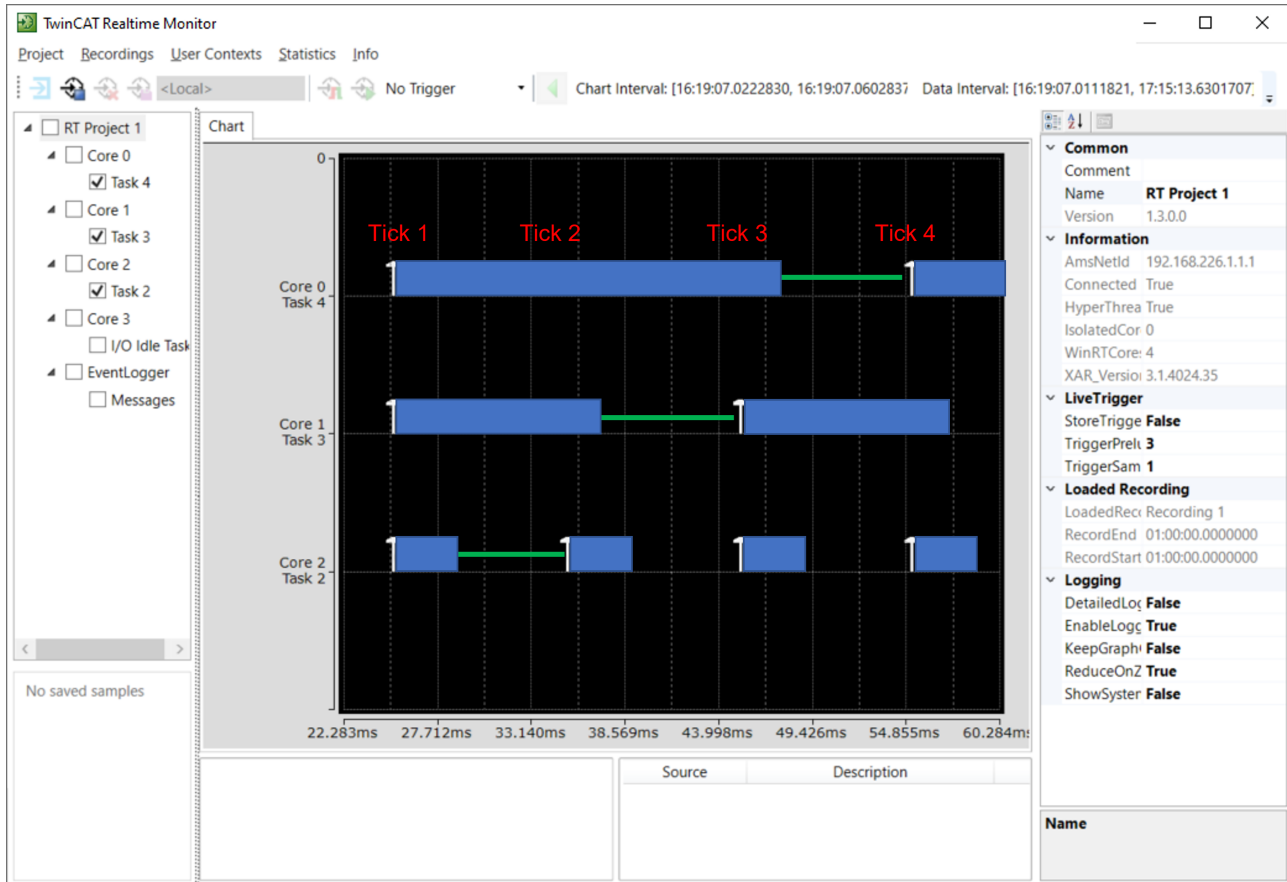
Properties:

- No function block is supported in the PLC.
- The TwinCAT Usermode Runtime is supported.
- Tasks can be assigned to different cores.
- The fastest task must be assigned the highest priority (smallest priority value). The second fastest task the second highest priority and so on.

Scheduling Details:

The graphic below describes an example of how the computing times can be distributed. The full blue area indicates that the task is working. Note that the surfaces have only been subsequently overlaid on the real-time monitor image to aid comprehension and are not real images.

- Task 2, Task 3 and Task 4 are executed in parallel on different cores in Tick 1. The execution of Task 1 must be completed by the start of Tick 2.
- Task 2 is executed again in tick 2. Task 3 and Task 4 may continue to work. The execution of Task 2 and Task 3 must be completed by the start of Tick 3.
- Task 2 and task 3 are executed again in tick 3. Task 4 may continue to work. The execution of Task 2 and Task 4 must be completed by the start of Tick 4.



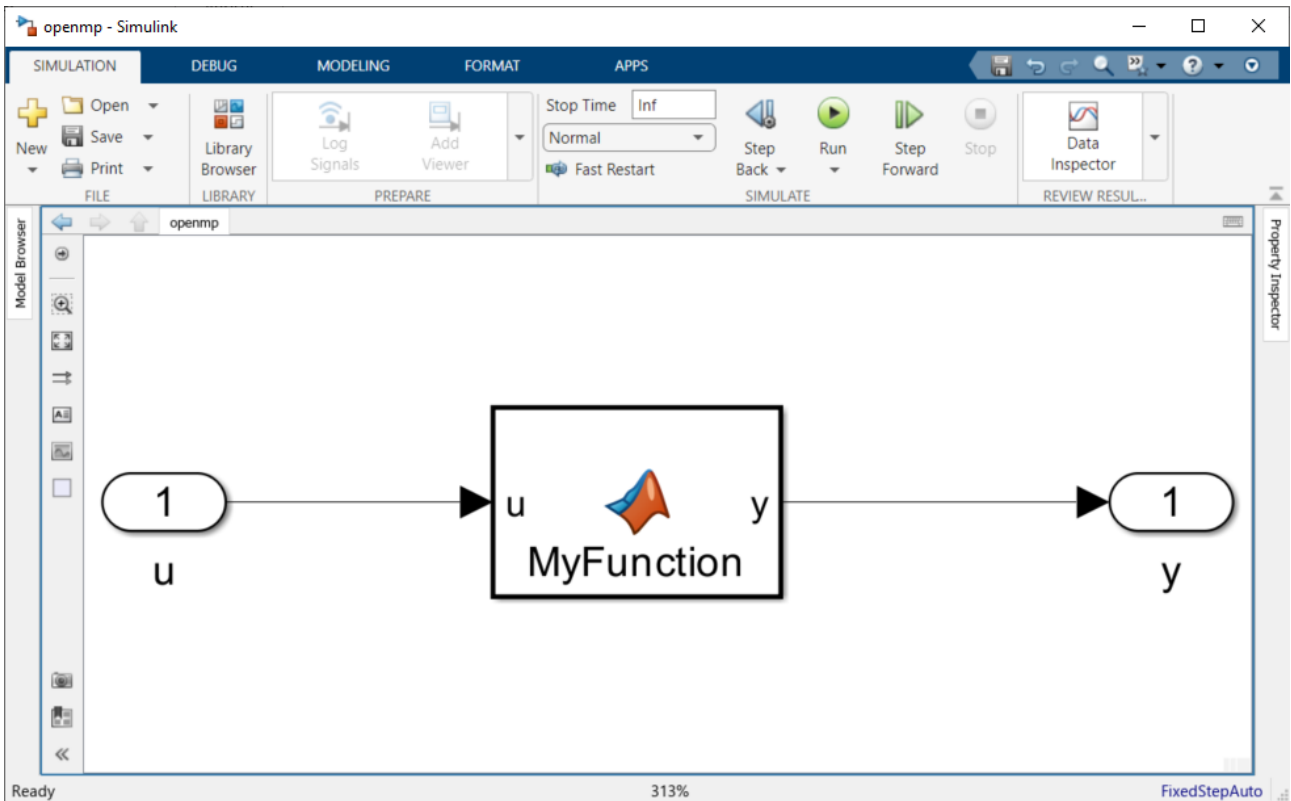
If cycle time overruns occur and scheduling cannot be adhered to, the execution of the respective task context is skipped until all relevant contexts are in the appropriate state. In the TcCOM object this behavior can be observed via the online parameter `SkippedExecutionCount`.

OpenMP

The Simulink Coder™ or the MATLAB Coder™ can generate openMP code. Please refer to the [MathWorks® documentation](#) for the exact cases in which this happens.

The following is an example using a MATLAB® Function in Simulink®. A MATLAB® example can be found in conjunction with the TE1401 TwinCAT Target for MATLAB® in the examples:

```
TwinCAT.ModuleGenerator.Samples.Start('Code parallelization with OpenMP').
```



The parfor command is used to parallelize the FOR loop in the MATLAB® function. In this case, the number of parallel workers is limited to 4.

```
function y = MyFunction(u) %#codegen
A = ones(20,50);
t = 42;

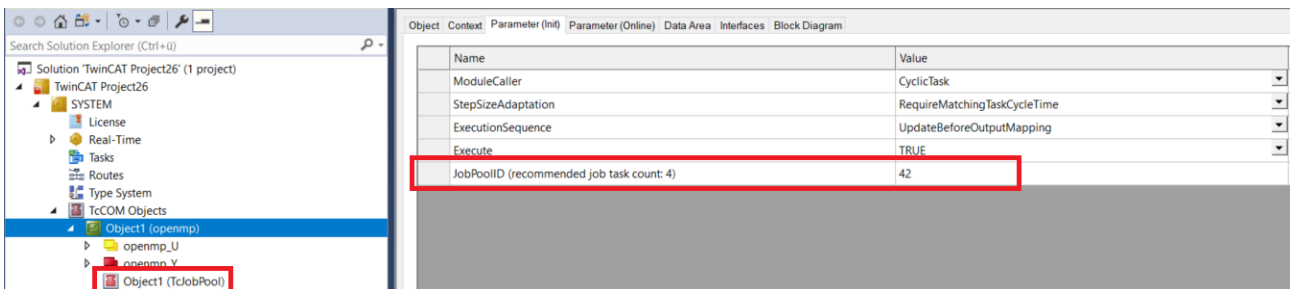
parfor (i = 1:10,4)
    A(i,1) = A(i,1) + t;
end

y = A(1,4) + u;
```

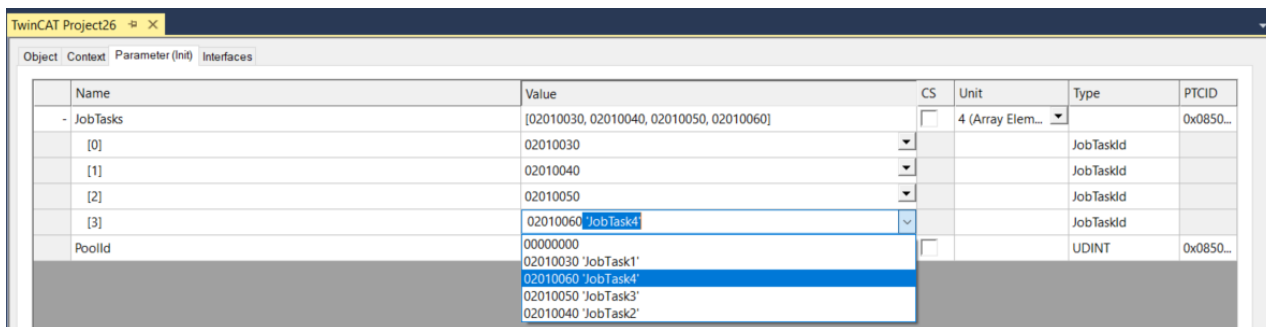
No special settings regarding openMP have to be made for the TwinCAT target. You generate your TwinCAT objects as usual. The Simulink Coder compiles this code into openMP code, so that the C/C++ code is parallelized accordingly. The Embedded Coder is not required for this feature.

In the TwinCAT XAE you can now instantiate the created TcCOM or the PLC-FB and configure it accordingly. As usual, the object instance offers only a cyclic task interface under the Context tab. A Task 2 with 200 ms cycle time is created and assigned to the object in this example.

There is a parameter JobPoolID under Parameter (Init). Here, as far as known from the C/C++ code, it is also shown how many workers can work in parallel. A JobPool is an organization unit for JobTasks, which can be created in the Tasks node.



Accordingly, an object of type TcJobPool must be added under TcCOM Objects with "Add new item". Under Parameter (Init) on the TcJobPool object, the JobPoolId is to be entered and a group of JobTasks is to be referenced. First define how many JobTasks the pool should combine and then select the JobTasks with the drop-down menu.



Under System > Realtime you can distribute JobTasks to different cores.

Object	RT-Core	Base Time (ms)	Cycle Time (ms)	Cycle Ticks	Priority
Task 2	Default (4)	1 ms	200 ms	200	1
JobTask1	Core 0	1 ms	(none)	0	2
JobTask2	Core 1	1 ms	(none)	0	3
JobTask3	Core 2	1 ms	(none)	0	4
JobTask4	Core 3	1 ms	(none)	0	5
I/O Idle Task	Default (4)	1 ms	1 ms	1	11

Execution in the configuration shown above then takes place as follows. Task 2 is executed on core 4 and cyclically drives the openmp object. The code fragments generated as openMP code can then outsource tasks to the configured JobTasks via the JobPool. When the JobTasks have finished their calculations, all partial results are bundled again and Task 2 on core 4 executes the code to the end.

4.7.12 Symbol Properties and Attribute Pragmas

What are properties and attributes?

For TcCOM objects, you can assign **properties** to all definitions, e.g. DataAreas, DataTypes, SubItems, etc. A property is defined as a name-value pair. Any additional information can be included.

Attributes are usually used in the PLC in the declaration part and can also bind any additional information to a variable, for example. Please refer to the [PLC documentation](#) for a list of PLC attributes.

Many TwinCAT functions use attributes and properties. Examples are:

- [TwinCAT OPC-UA](#)
 - {attribute 'OPC.UA.DA' := '1'}
 - {attribute 'OPC.UA.DA.Access' := '1'}
- [Analytics Logger](#)
 - {attribute 'TcAnalytics'}
- [JSON Library Tc3 JsonXml](#)
- ...

Attributes and properties can also be defined by the user and used for own applications.

How do I use symbol properties and attributes together with the TwinCAT target?

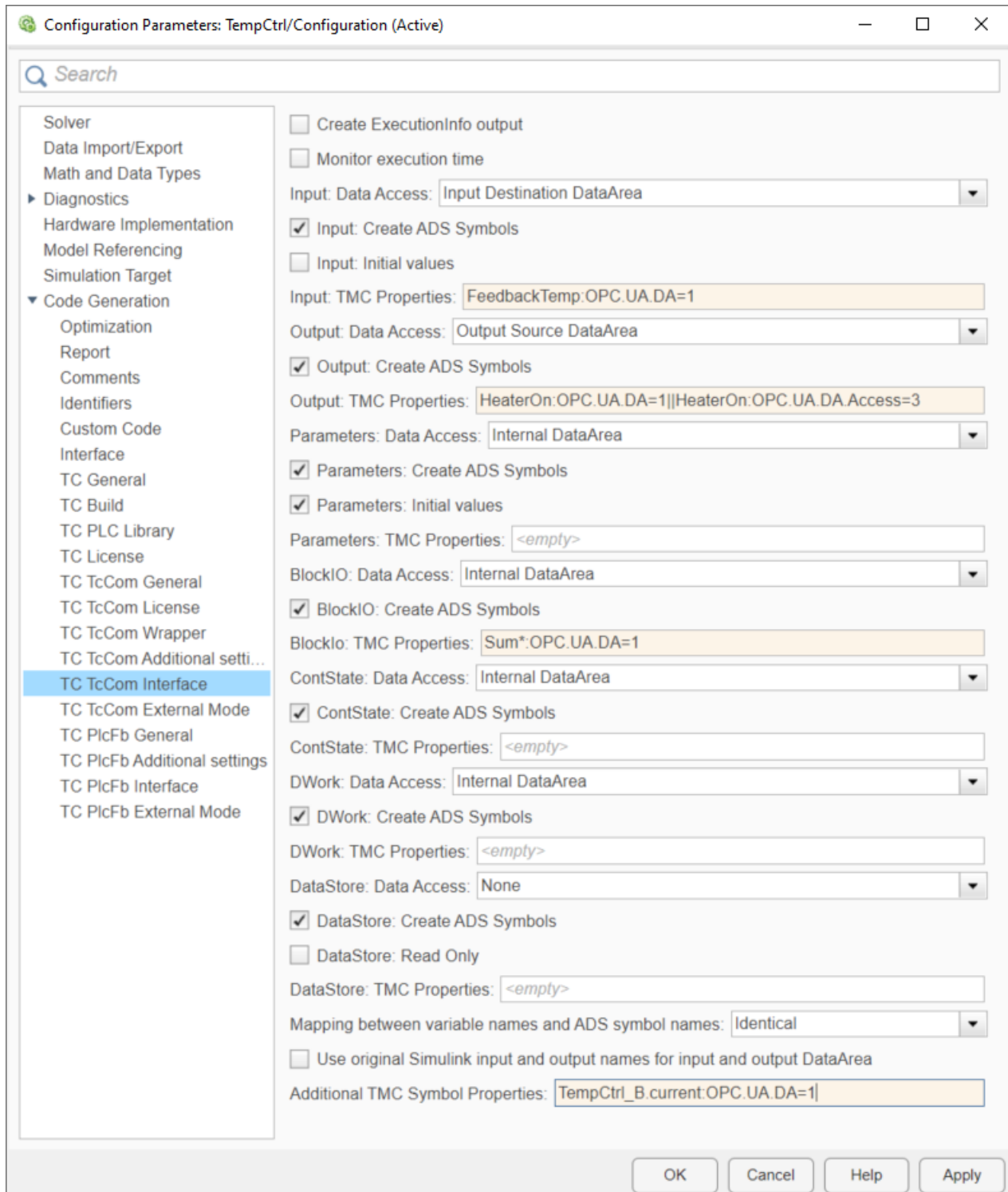
It is possible to assign properties to ADS symbols with the TwinCAT target. ADS symbols are to be understood in the sense of variables.

(TcCOM) TMC Properties

In Simulink® you can freely assign Properties as a string in the Configuration Parameters under TC TcCom Interface. Properties can be defined for the DataArea:

- Input: TMC Properties
- Output: TMC Properties
- Parameters: TMC Properties
- BlockIO: TMC Properties
- ContState: TMC Properties
- DWork: TMC Properties
- DataStore: TMC Properties

In addition, you can use the *Additional TMC Symbol Properties* field to assign properties independently of the DataArea.



The following notation is used for DataArea specific TMC properties:

```
SymbolName1:PropertyName=Value1
```

Sample: FeedbackTemp:OPC.UA.DA=1

If several properties are to be assigned, they must be separated with |:

```
SymbolName1:PropertyName=Value1|SymbolName2:PropertyName2=Value2
```

Sample: HeaterOn:OPC.UA.DA=1|HeaterOn:OPC.UA.DA.Access=3

wildcards can be used from MATLAB R2020b:

```
*:PropertyName=Value1
```

The * character is used as a wildcard and in this sample assigns the PropertyName1 with Value1 to all symbols in the DataArea. Substrings can also be combined with wildcards. The following sample assigns the specified property to all symbols that begin with Sum.

```
Sum*:OPC.UA.DA=1
```

The same structure must be used in the Additional TMC Symbol Properties field. However, the DataArea must be added for addressing.

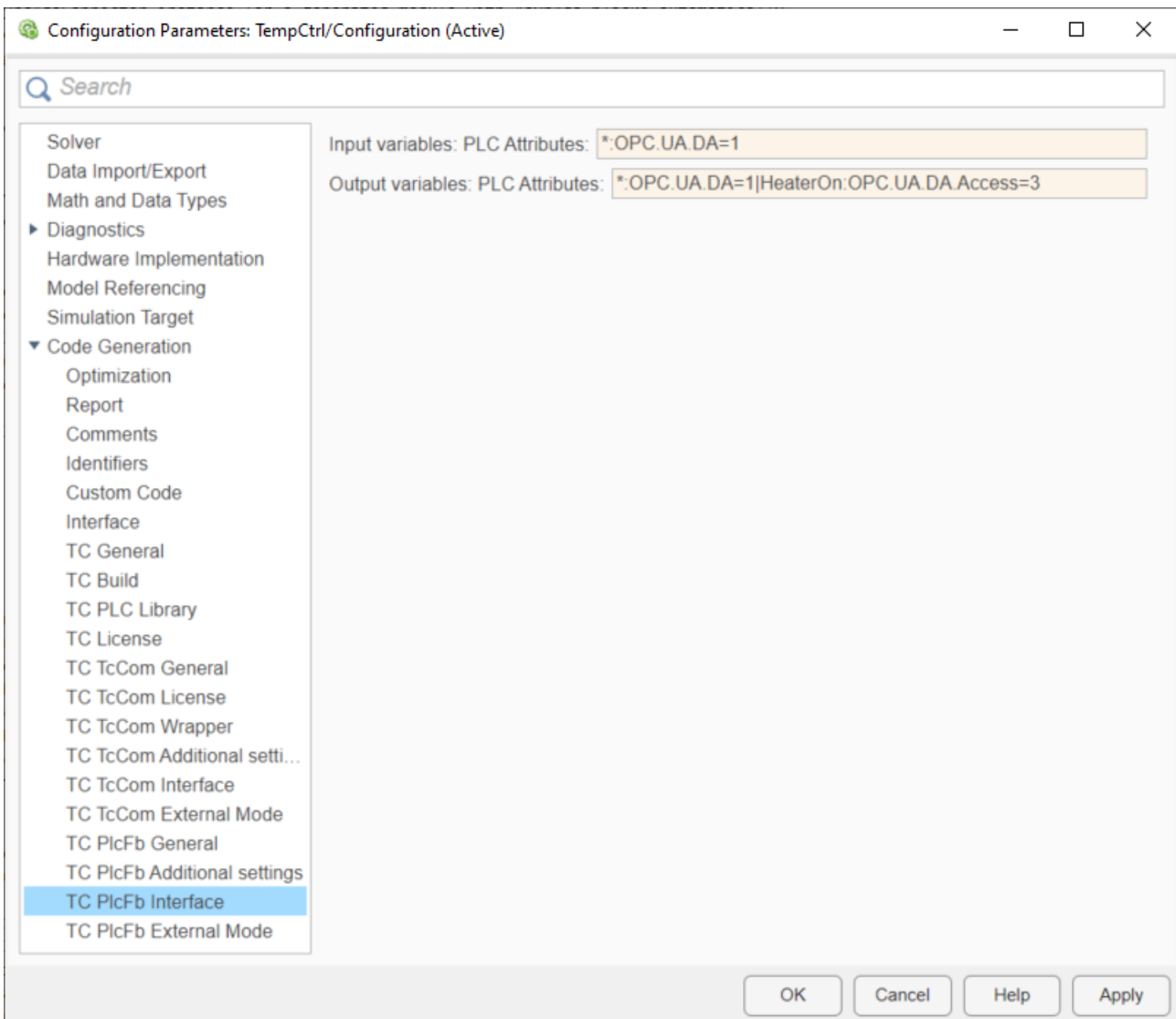
```
DataAreaName1.SymbolName1:PropertyName=Value1
```

Sample: TempCtrl_B.current:OPC.UA.DA=1

PLC Attributes

The same procedure as for TcCOM applies to the [PLC-FB \[► 213\]](#), but there only possible for the inputs and outputs. When using the [TcCOM-Wrapper-FB \[► 209\]](#) a TcCOM is referenced, so that you make the configuration according to TMC Properties.

The configuration for the PLC-FB is done under TC PlcFb Interface.



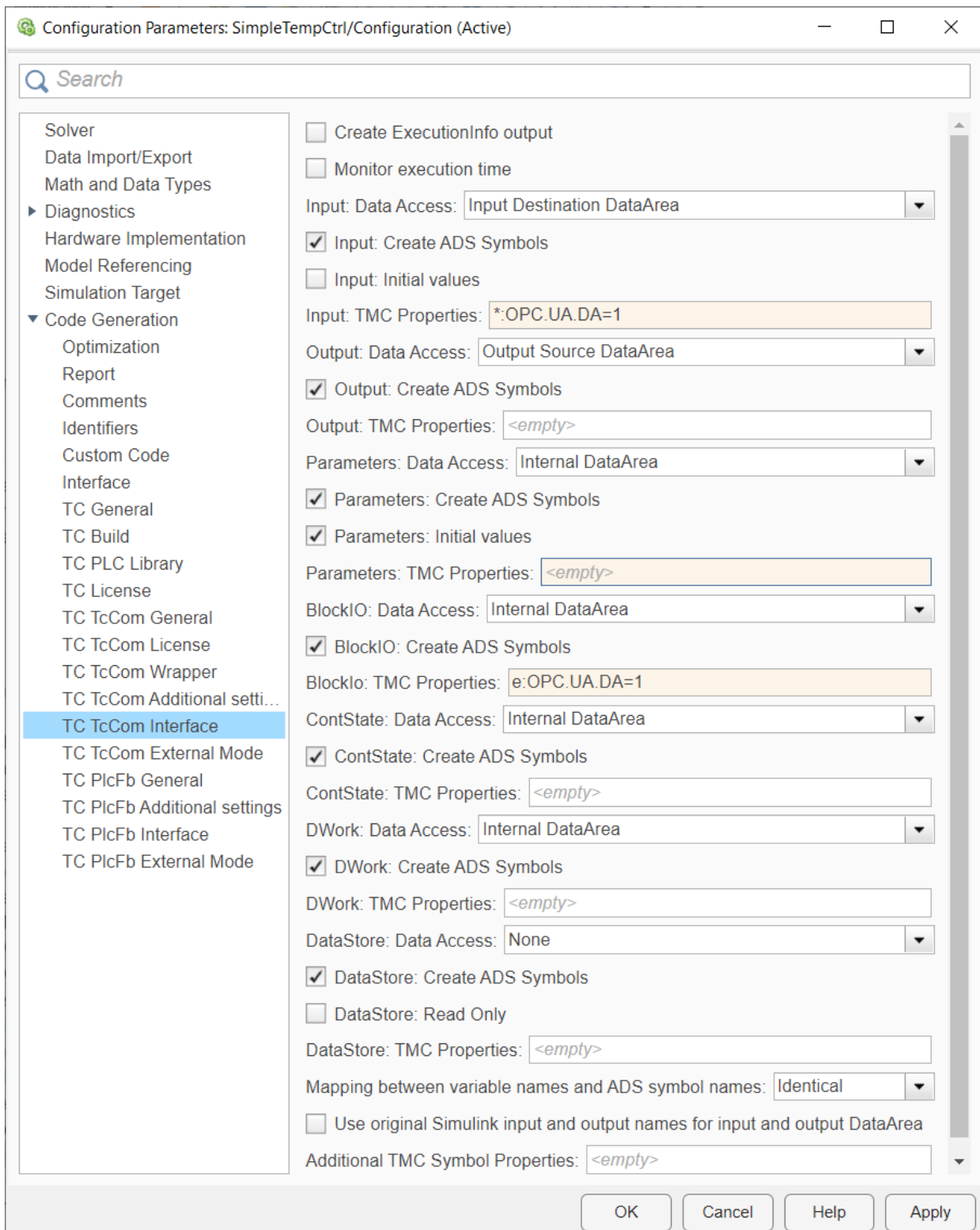
i Sample code in MATLAB®

Open the appropriate sample with: `TwinCAT.ModuleGenerator.Samples.Start('Simulink TMC Symbol Properties')`

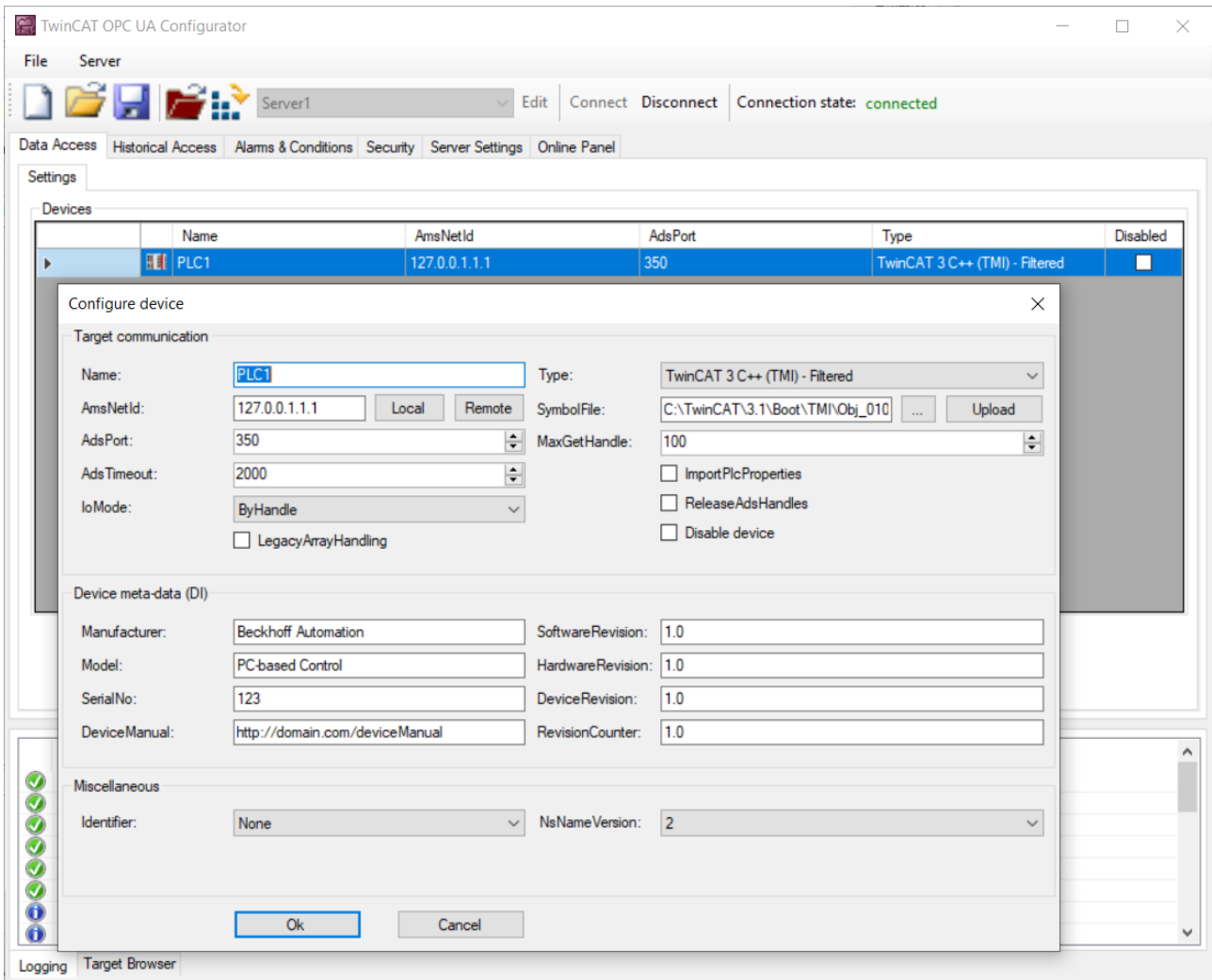
What are properties and attributes used for?

An example is the use of attributes in the PLC or Symbol Properties for TcCOM in connection with **TwinCAT 3 OPC-UA**. See the [OPC-UA list of attributes](#).

The following example shows how all input variables of the Input DataArea and only the signal e from the BlockIO DataArea are provided with the OPC-UA data access property. The wildcard * is used for the input DataArea.



You have to select TwinCAT 3 C++ (TMI) - Filtered on the right side under Type in the OPC-UA configurator, so that only the symbols with the corresponding property are displayed in the server. If you want to see all symbols in the server, simply select TwinCAT 3 C++ (TMI) - All at this point. Then all symbols will be displayed, regardless of the properties. Also select the TMI file of the TcCOM on the TwinCAT Target Device as SymbolFile (Boot\TMI Folder).



Connect to the OPC-UA Server with a client to inspect the namespace. Here the OPC-UA Sample Client of TwinCAT 3 was used. You can see that only those symbols are displayed in the server that have been explicitly assigned the OPC.UA.DA=1 property.

The screenshot shows the TwinCAT OPC UA Sample Client interface. At the top, the server is set to `opc.tcp://localhost:4840` and the endpoint is `opc.tcp://FabianBa-Nb01:4840 [Sign:Basic256Sha256:Binary]`. The interface is divided into three main sections: a tree browser on the left, a watchlist table in the center, and an attributes panel on the right. The watchlist table displays data for three variables: SetpointTemp, FeedbackTemp, and e. The attributes panel shows details for the selected variable 'e', including its namespace index, identifier type, and current value.

ID	Name	Value	TimeStamp	StatusCode
ns=4;s=OI	SetpointTemp	0	3/16/2023 3:18:41 PM	Good
ns=4;s=OI	FeedbackTemp	0	3/16/2023 3:18:41 PM	Good
ns=4;s=OI	e	0	3/16/2023 3:18:41 PM	Good

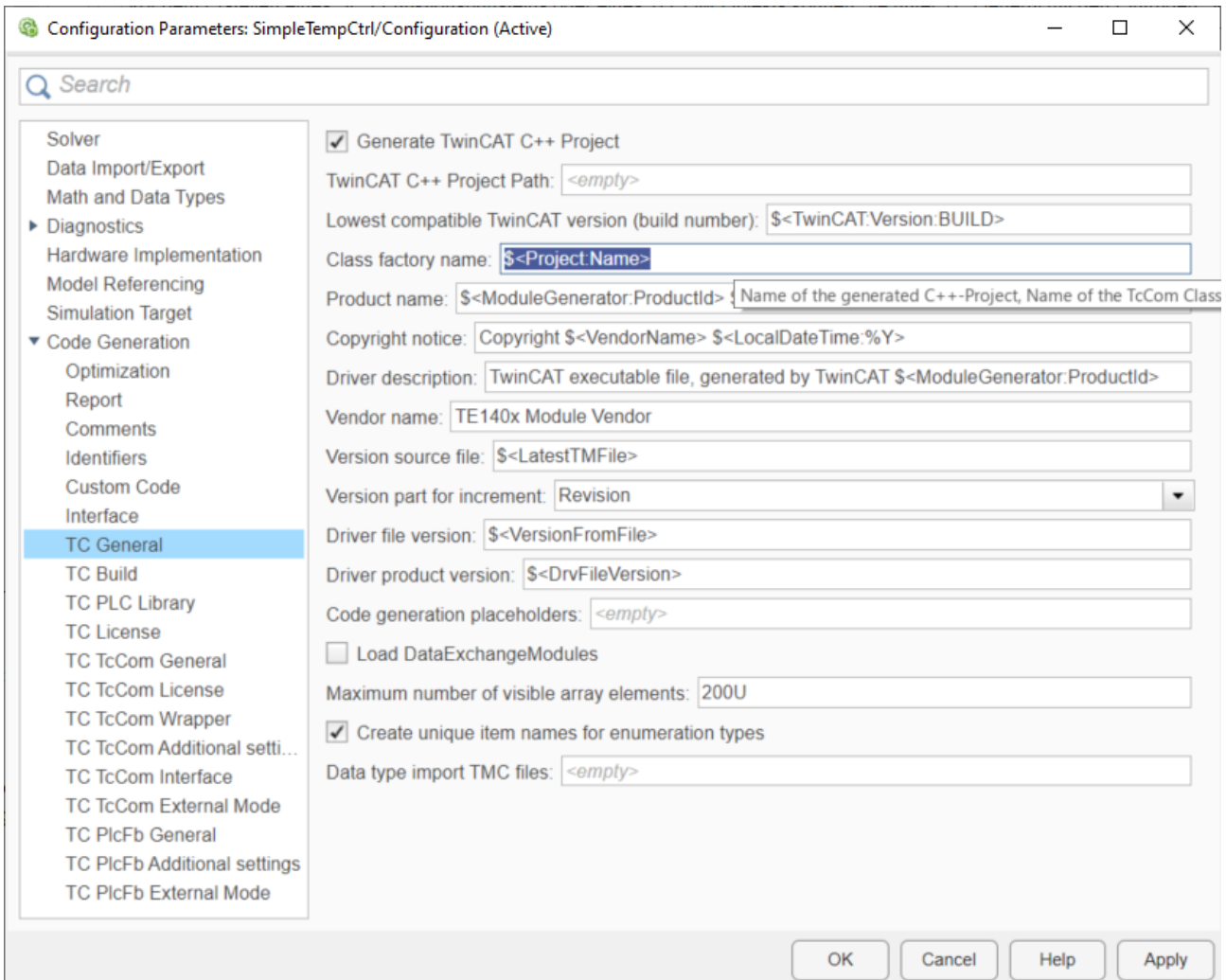
Attribute	Value
NodeId	NodeId
NamespaceIndex	4
IdentifierType	String
Identifier	Object1 (SimpleTempC)
NodeClass	Variable
BrowseName	4:e
DisplayName	e
Description	
WriteMask	0
UserWriteMask	0
Values	
SourceTimestamp	16.03.2023 02:16:01 PM
ServerTimestamp	16.03.2023 02:16:01 PM
SourcePicoSeconds	0
ServerPicoSeconds	0
Value	0
Data Type	
NamespaceIndex	0
IdentifierType	Numeric

Log messages in the bottom panel:

- 3:18:40 PM - Received data: Variable - e.
- 3:18:39 PM - Received data: Variable - FeedbackTemp.
- 3:18:26 PM - Received data: Variable - SetpointTemp.
- 3:16:04 PM - Browse treenode e.
- 3:16:01 PM - Browse treenode FeedbackTemp.

4.7.13 Available placeholders

Placeholders are used in the Target for Simulink® configuration to reduce configuration effort and increase clarity. Placeholders are specified in the configuration with `$<PlaceholderName>`.



What are placeholders?

Placeholders can be used to specify the value of a configuration parameter abstractly as a variable. Specific placeholders exist at the level of the target (module generator), the project and the modules (TcCOM and PLC FB). Furthermore, the configuration parameter itself is also a placeholder, so you can reuse it.

Example:

In the above graphic, the driver product version is specified with the placeholder `$(DrvFileVersion)`. This placeholder points to the driver file version entry, which in turn is occupied by the placeholder `$(VersionFromFile)`. The placeholder `$(VersionFromFile)` assumes the version value, where the source of the version value is defined with the version source file parameter.

Why do we need placeholders?

Placeholders allow us to define a parameter value and reuse it in many places in the configuration, or to set them in a chain dependency in relation to each other (see example above).

Overview of available placeholders

Placeholders from the group of configuration parameters

Category	Name	Displayname	Default	Description
TC General	Generate	Generate TwinCAT C++ Project	TRUE	Generate a TwinCAT C++ project. If unset, only code artifacts will be generated which can get used to generate C++ projects later [▶ 131].

Category	Name	Displayname	Default	Description
	FullPath	TwinCAT C++ Project Path		Full path to the generated VCXPROJ file (e. g. "C:\Temp\MyGeneratedProject.vcxproj")
	LowestCompatibleTcBuild	Lowest compatible TwinCAT version (build number)	\$(TwinCAT:Version:BUILD)	The lowest TwinCAT build number the generated C++ project and its modules and POU's are to be compatible with.
	ClassFactoryName	Class factory name	\$(Project:Name)	Name of the generated C++-Project, Name of the TcCOM classfactory and tmx-file name
	ProductName	Product name	\$(ModuleGenerator:ProductId) \$(ModuleGenerator:Version:MAJOR.MINOR)	Product name, used e.g. for the module driver description and the module TMC description [► 159].
	Copyright	Copyright notice	Copyright \$(VendorName) \$(LocalDateTime:%Y)	Copyright notice of the generated module driver file [► 159]
	Description	Driver description	TwinCAT executable file, generated by TwinCAT \$(ModuleGenerator:ProductId)	Driver description [► 159]
	VendorName	Vendor name	TE140x Module Vendor	Module vendor name, used as the company name of the generated executables in the repository [► 120] and the major module group as shown in the TwinCAT XAE module dialog [► 90].
	VersionSrc	Version source file	\$(LatestTMFile)	Path to an existing TMC, TML or XML file containing the previous version value [► 137].
	IncrementVersion	Version part for increment	Revision	The part of the version number that is to be incremented [► 137].
	DrvFileVersion	Driver file version	\$(VersionFromFile)	Executable file version and library version. [► 137]
	DrvProductVersion	Driver product version	\$(DrvFileVersion)	Product version [► 137]
	CodeGenPlaceholders	Code generation placeholders		Define custom placeholders
	UseDataExchangeModules	Load DataExchangeModules	0	Manually set DataExchangeModule dependency (currently no need to set manually)
	MaxVisibleArrayElements	Maximum number of visible array elements	200U	Specifies the maximum number of array elements to be displayed in the TwinCAT XAE. In the TwinCAT XAE, larger arrays cannot get expanded and linked to by its individual items

Category	Name	Displayname	Default	Description
	CreateUniqueEnumItemNames	Create unique item names for enumeration types	1	Create unique item names for enumeration types.
	DataTypeTmcFiles	Data type import TMC files		TMC file(s) containing additional type definitions for code generation
TC Build	PreferToolArchitectureX64	Prefer X64 build tools	TRUE	Prefer X64 compiler and linker. Useful for complex source files, where X86 tools may run out of heap space.
	Verbosity	Codegeneration and build verbosity	Normal	Verbosity level of code generation and build output messages. Silent and Detailed are other possible values.
	Publish	Run the publish step after project generation	TRUE	Start the build procedure after code generation for all selected platforms. The generated module binaries and module description files will get copied to the "publish folder". Published modules are automatically located by the XAE and can get instantiated in all TwinCAT 3 projects. If unset, the module generation process will be stopped after code generation. To instantiate in a TwinCAT3 project, the generated C++ project needs to be inserted and built from.
	PublishPlatformtoolset	Platform Toolset	Auto	Choose Platform Toolset to build binaries.
	PublishConfiguration	Build configuration	Release	Build configuration to build binaries.
	PublishTcRTx86	TwinCAT RT (x86)	TRUE	Publish binaries for platform 'TwinCAT RT (x86).'
	PublishTcRTx64	TwinCAT RT (x64)	TRUE	Publish binaries for platform 'TwinCAT RT (x64).'
	PublishTcOSx64	TwinCAT OS (x64)	TRUE	Publish binaries for platform 'TwinCAT OS (x64)' (e.g. TwinCAT/BSD)
	ForceRebuildForPublish	Always rebuild all source files on publish	FALSE	Always rebuild all source files on publish
	SignTwinCatCertName	Certificate name for TwinCAT signing		Certificate name for TwinCAT signing with OEM Certificate level 2. [▶ 93]
	TmxInstall	Install TMX	TRUE	Install all generated TwinCAT Objects on local XAE (fill local Engineering Repository [▶ 120]).
	TmxArchive	TMX Archive		Name of an optional archive containing all files required to use the generated TwinCAT Objects on another TwinCAT development system. [▶ 135]
	MsBuildPublishProperties	MsBuild publish properties		Set additional MsBuild publish properties.
	MsBuildProjProperties	MsBuild project properties		Set additional MsBuild project properties.

Category	Name	Displayname	Default	Description
	PreCodeGenerationCallbackFcn	Pre code generation callback function		The defined MATLAB® function is called before code generation [► 182].
	PostCodeGenerationCallbackFcn	Post code generation callback function		The defined MATLAB® function is called after code generation [► 182].
	PostPublishCallbackFcn	Post publish callback function		The defined MATLAB® function is called after publish [► 182].
TC PLC library	LibCatPath	PLC library category description file	\$(ProjectDir)\\$(Name).libcat.xml	Path to the PLC library category description file
	LibraryCategories	PLC library categories	\$(VendorName>	Define PLC library category hierarchy. Default only one hierarchy level = vendor. List separated with possible: <MainCategory> <SubCategory1> ...
	GeneratePlcLibrary	Generate a PLC library	FALSE	Generate a PLC library with POU's. [► 206] Define containing POU's with parameter TcComWrapperFb and PlcFb>General>Generate.
	InstallPlcLibrary	Install the generated PLC library	FALSE	Install the generated PLC library for use in the local TwinCAT XAE/PLC [► 206].
	PlcTypePrefixes	Type Prefixes		Define custom type prefixes
	PlcVarPrefixes	Variable Prefixes	`PVOID=p \ BOOL=b \ BOOL32=b \ DATE=d \ TIME_OF_DATE=td \ TIME=t \ LTIME=t \ GUID=n`	Define custom variable prefixes.
TC License	OemId	ID of OEM		ID of OEM. Required for OEM Licence checks [► 156]
	OemLicenses	IDs of OEM Licenses		IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [► 156]
TC TcCom General	Generate	Generate TcCOM Module (TwinCAT Module Class)	TRUE	Generate a TcCOM module class for the model.
	OnlineChange	Online change support	FALSE	Allow to switch between different TcCOM module versions without switching TwinCAT runtime to config mode [► 141].
	ModuleProperties	TMC Properties		Additional properties added to the module description in the TMC file: Name1=Value1 Name2=Value2 ...
	GroupName	GroupName	TE140x Simulink Modules	Minor module group name in the TwinCAT XAE module dialog

Category	Name	Displayname	Default	Description
	GroupDisplayName	GroupDisplayName	\$(GroupName)	Minor module group description in the TwinCAT XAE module dialog
	GroupIcon	GroupIcon	\$(TE140x:Icon)	Optional module group icon in the TwinCAT XAE module dialog
	ModuleIcon	ModuleIcon	\$(TE140x:Icon)	Optional module icon in the TwinCAT XAE module dialog
	InitExceptionHandler	Floating point exception handling during initialization	CallerExceptions	<u>Configures how to throw, suppress or handle floating point exceptions during initialization</u> [▶ 220].
	UpdateExceptionHandler	Floating point exception handling during update	CallerExceptions	<u>Configures how to throw, suppress or handle floating point exceptions during cyclic execution</u> [▶ 220].
	AdditionalIncludeFiles	Additional include files		Additional files required to be included after rtwtypes.h
TC TcCom License	OemLicenses	IDs of OEM License	\$(Project:OemLicenses)	<u>IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}"</u> [▶ 156]
TC TcCom Wrapper	TcComWrapperFb	TcCom Wrapper FB	FALSE	<u>Generate a PLC Functionblock simplifying the interaction between a PLC and an instance of the generated TcCOM module</u> [▶ 209]
	TcComWrapperFbProperties	TcCom Wrapper FB properties	FALSE	<u>Generate properties for accessible data in the referenced TcCOM object</u> [▶ 209]
	TcComWrapperFbPropertyMonitoring	TcCom Wrapper FB property monitoring	NoMonitoring	<u>NoMonitoring: Online values of properties are not monitored in the PLC online view, CyclicUpdate: Update property values in the PLC online view cyclically, ExecutionUpdate: Update property values in the PLC online view when the property getter or setter is called</u> [▶ 209]
TC TcCom Additional settings	ModuleCaller	Default module caller	CyclicTask	<u>CyclicTask: Call module via TwinCAT Task. Module: Call module from another TwinCAT module (see e.g. TcCOM-Wrapper-FB).</u>
	CallerVerification	Verify caller	Default	<u>Verify the caller context to prevent concurrent execution of the model code and corresponding DataArea mappings. Skip verification to reduce the execution time.</u>
	StepSizeAdaptation	Default StepSize adaptation mode	RequireMatchingTaskCycleTime	<u>Configure how to handle differences between the default model step size(s) and the cycle time of the assigned task(s).</u>

Category	Name	Displayname	Default	Description
	ExecutionSequence	Default execution sequence	UpdateBeforeOutputMapping	Configure the execution order of input mapping, model code execution and output mapping.
	ExecuteModelCode	Execute model code after startup	TRUE	Start cyclic execution of the model code after startup by default. If FALSE, Module Parameter Execute needs to be set to TRUE to start execution of code.
	BlockDiagramExport	Export BlockDiagram	TRUE	<u>Export graphical block diagram information for monitoring and optional debugging on the generated TwinCAT module in TwinCAT XAE [► 193]</u>
	ResolveMaskedSubsystems	Resolve Masked Subsystems	FALSE	Resolve masked subsystems in the block diagram
	ExtendSignalResolution	Extended resolution of signals in block diagram	FALSE	<u>Intensified search for assignments of variables and block diagram signals (blue signals). This option increases the build time. [► 235]</u>
	BlockDiagramVariableAccess	Access to VariableGroup not referenced by any block	AssignToParent	Variables from a block within an unresolved subsystem are either assigned to the next higher visible block or hidden in the block diagram.
	BlockDiagramDebugInfoExport	Export BlockDiagram debug info	TRUE	<u>Export additional information required to debug the module using the block diagram [► 197].</u>
TC TcCom Interfaces	ExecutionInfoOutput	Create ExecutionInfo output	FALSE	<u>Create additional output DataAreas containing execution and exception information [► 220].</u>
	MonitorExecutionTime	Monitor execution time	FALSE	Calculate and expose the execution time of the module as an ADS variable for monitoring purposes.
	InputDataAccesses	Input: Data Access	Input Destination DataArea	<u>Defines how the input variables are exposed in TwinCAT [► 143].</u>
	InputCreateSymbols	Input: Create ADS Symbols	TRUE	<u>Create ADS symbol information for the input variables [► 143]</u>
	InputInitValues	Input: Initial values	FALSE	<u>Create module parameters for the input variables to allow definition of initial values [► 143]</u>
	InputProperties	Input: TMC Properties		<u>Additional properties added to the Input symbol description in the TMC file. [► 166]</u>
	OutputDataAccess	Output: Data Access	Output Source DataArea	<u>Defines how the output variables are exposed in TwinCAT [► 143].</u>
	OutputCreateSymbols	Output: Create ADS Symbols	TRUE	<u>Create ADS symbol information for the output variables [► 143].</u>

Category	Name	Displayname	Default	Description
	OutputProperties	Output: TMC Properties		Additional properties added to the Output symbol description in the TMC file. [► 166]
	ParametersDataAccess	Parameters: Data Access	Internal DataArea	Defines how the model parameter variables are exposed in TwinCAT [► 143]
	ParametersCreateSymbols	Parameters: Create ADS Symbols	TRUE	Create ADS symbol information for the model parameter variables [► 143].
	ParametersInitValues	Parameters: Initial values	TRUE	Create module parameters for the model parameter variables to allow definition of initial values [► 143].
	ParametersProperties	Parameters: TMC Properties		Additional properties added to the Parameters symbol description in the TMC file. [► 166]
	BlockIoDataAccess	BlockIO: Data Access	Internal DataArea	Defines how the BlockIO variables are exposed in TwinCAT [► 143]
	BlockIoCreateSymbols	BlockIO: Create ADS Symbols	TRUE	Create ADS symbol information for the BlockIO variables [► 143].
	BlockIoProperties	BlockIO: TMC Properties		Additional properties added to the BlockIO symbol description in the TMC file. [► 166]
	ContStateDataAccess	ContState: Data Access	Internal DataArea	Defines how the continuous state variables are in TwinCAT [► 143]
	ContStateCreateSymbols	ContState: Create ADS Symbols	TRUE	Create ADS symbol information for the continuous state variables [► 143].
	ContStateProperties	ContState: TMC Properties		Additional properties added to the ContState symbol description in the TMC file. [► 166]
	DWorkDataAccess	DWork: Data Access	Internal DataArea	Defines how the DWork variables are exposed in TwinCAT [► 143]
	DWorkCreateSymbols	DWork: Create ADS Symbols	TRUE	Create ADS symbol information for the DWork variables [► 143].
	DWorkProperties	DWork: TMC Properties		Additional properties added to the DWork symbol description in the TMC file. [► 166]
	DataStoreDataAccess	DataStore: Data Access	None	Defines how the DataStore variables are exposed in TwinCAT [► 143]
	DataStoreCreateSymbols	DataStore: Create ADS Symbols	TRUE	Create ADS symbol information for the DataStore variables [► 143].
	DataStoreReadOnly	DataStore: Read Only	FALSE	Restrict ADS access to be read only for the DataStore variables [► 143].

Category	Name	Displayname	Default	Description
	DataStoreProperties	DataStore: TMC Properties		Additional properties added to the DataStore symbol description in the TMC file. [► 166]
	SymbolProperties	Additional TMC Symbol Properties		Additional properties added to specific symbol descriptions in the TMC file. [► 166]
	VariableSymbolMapping	Mapping between variable names and ADS symbol names	Identical	Defines the TwinCAT symbol names for the generated C/C++ variables. 'Identical': Symbol name equals variable name, 'Classic': Use symbol names known from TE1400 Release 1.2.x.x [► 143]
TC TcCom External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [► 217].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode [► 217] connection before starting model code execution.
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [► 217].
TC PlcFb General	Generate	Generate TwinCAT PLC Function Block	TRUE	Generate a PLC-FB for the model [► 213].
	InitExceptionHandler	Floating point exception handling during initialization	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during initialization [► 220].
	UpdateExceptionHandler	Floating point exception handling during update	CallerExceptions	Configures how to throw, suppress, or handle floating point exceptions during cyclic execution [► 220].
TC PlcFb License	OemLicenses	IDs of OEM License	\$(Project:OemLicenses)	IDs of OEM Licenses. Multiple IDs may be inserted as a comma separated list. "{GUID},{GUID}" [► 156]
TC PlcFb Additional settings	MonitorExecutionTime	Monitor ExecutionTime	FALSE	Calculate and expose the execution times of TwinCAT modules as an ADS variable for monitoring purposes.
PlcFb->Interface	InputAttributes	Input variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
	OutputAttributes	Output variables: PLC Attributes		Additional attributes added to the PLC FB Input variables.
TC PlcFb External Mode	ExtModeRtAllowExecutionCommands	Allow RealTime execution commands via External Mode	FALSE	Allow to start and stop model code execution via External Mode [► 217].
	ExtModeRtWaitForStart	Wait for RealTime execution start command via External Mode	FALSE	Wait for External Mode connection before starting model code execution [► 217].

Category	Name	Displayname	Default	Description
	ExtModeRtAllowForParameterChange	Allow to change parameters via External Mode	FALSE	Allow to change parameter online values via External Mode [► 217].

Placeholders at target level (module generator)

Placeholders from this group can be used at target/project and module level.

Placeholder name	Description
ModuleGenerator:ProductName	Product name of the module generator
ModuleGenerator:Version	Version of the module generator
TwinCAT:Version	Version of local TwinCAT installation
UsablePlatformToolsets	Available and supported platform toolsets
LocalDateTime:Format	Actual local time as string where Format must be defined like the format string for std::put_time (e.g. '%Y-%m-%d')
UtcDateTime:Format	Actual UTC time as string where Format must be defined like the format string for std::put_time (e.g. '%Y-%m-%d')
EnvironmentVarName	Any environment variable defined for the system, the current user or the current process (MATLAB®).

Placeholders at project level

Placeholders from this group can be used at project and module level.

Placeholder name	Description
Project:Name	Name of the project file (without directory and extension)
Project:Dir	Project file directory
Project:Ext	Project file extension
Project:Path	Full project file path
Project:Guid	Project GUID
Project:LibraryID	LibraryID of the generated repository driver
Project:VendorName	Company name part of the LibraryID
Project:DriverName	Driver name part of the LibraryID
Project:DrvFileVersion	Version part of the LibraryID
Project:LatestTMFile	Path to an existing corresponding TML or TMC file with the highest library version (searching project directory and repository)
Project:LatestTMFile:Repository	Path to an existing corresponding TML or TMC file with the highest library version (searching only repository)
Project:LatestTMFile:ProjectDir	Path to an existing corresponding TML or TMC file with the highest library version (searching only project directory)
Project:VersionFromFile	Version read from file defined by "VersionSrc"

Placeholders at module level (TcCOM and PLC FB)

Placeholders from this group can only be used at module level.

Placeholder name	Description
Module:Name	Name of the TcCom module or PLC FB
Module:ClsId	Class ID of the TcCom module (TcCom only)
Module:ContextCount	Number of task contexts

Placeholder name	Description
Module:ClassName	Name of the TcCom module
Module:CppClassFileName	Name of the corresponding .h and .cpp files
Module:ModelName	Name of the corresponding Simulink Model (TE1400 only)
Module:MFileName	Name of the corresponding M-File (TE1401 only)
Module:FileFilterName	Visual Studio project filter name

4.7.14 Working with callbacks

There are three different callback functions:

- **Pre code generation callback function:** Callback before the model is converted to C++ code.
- **Post code generation callback function:** Callback after the model has been converted to C++ code.
- **Post publish callback function:** Callback after the created C++ project has been built for the configured platforms.

Enter the name of your created MATLAB® function here to call it.

Configuration Parameters: SimpleTempCtrl/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
▶ Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
▼ Code Generation
Optimization
Report
Comments
Identifiers
Custom Code
Interface
TC General
TC Build
TC PLC Library
TC License
TC TcCom General
TC TcCom License
TC TcCom Additional setti...
TC TcCom Interface
TC TcCom External Mode
TC PlcFb General
TC PlcFb Additional settin...

Prefer X64 build tools
Codegeneration and build verbosity: Normal
 Run the publish step after project generation
Platform Toolset: Auto
Build configuration: Release
 TwinCAT RT (x86)
 TwinCAT RT (x64)
 TwinCAT OS (x64)
 TwinCAT UM (x86)
 TwinCAT UM (x64)
 Always rebuild all source files on publish
Certificate name for TwinCAT signing: <empty>
 Install TMX
TMX Archive: <empty>
MsBuild publish properties: <empty>
MsBuild project properties: <empty>
Generate a VisualStudio Solution file: No
Pre code generation callback function: <empty>
Post code generation callback function: <empty>
Post publish callback function: MyCallback

OK Cancel Help Apply

Your MATLAB® function is passed the ProjectExporter object as a transfer parameter:

```
function MyCallback(obj)
...
return
```

The object contains the current configuration of the build in its properties.

```
ProjectExporter with properties:
ProjectGenerator: [1x1 TwinCAT.ModuleGenerator.ProjectGenerator]
Configuration: [1x1 TwinCAT.ModuleGenerator.ProjectExportConfig]
```

```
Project: [1x1 TwinCAT.ModuleGenerator.Project]
State: [1x1 struct]
ClassExporters: {[1x1 TwinCAT.ModuleGenerator.Simulink.ModelExporter]}
AdditionalExports: [1x1 containers.Map]
```

i Sample code in MATLAB®

Open the appropriate sample with:
`TwinCAT.ModuleGenerator.Samples.Start('Callbacks')`

4.8 Application of modules in TwinCAT

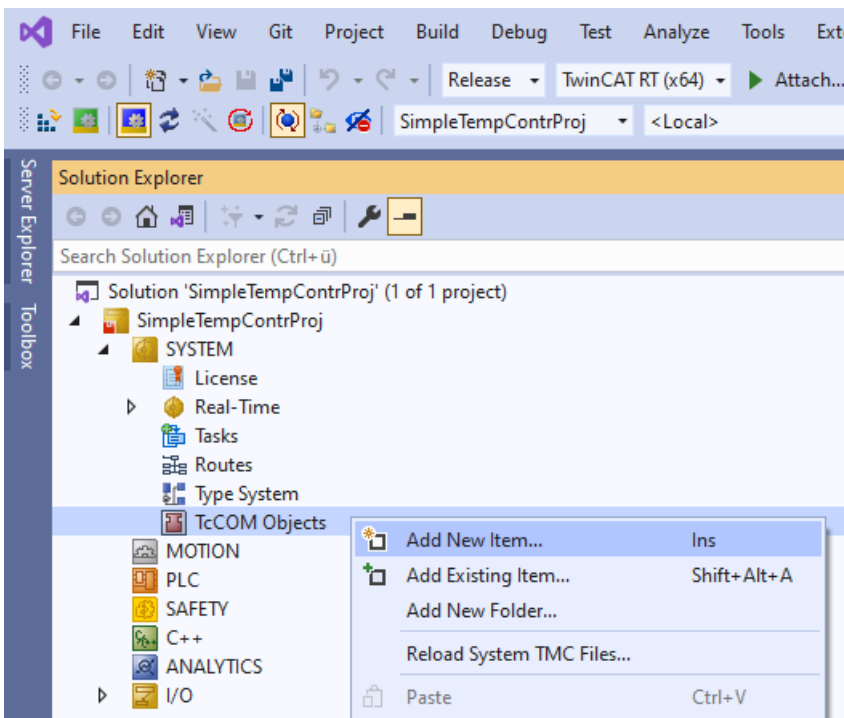
TcCOM and function blocks created with the Target for Simulink® can be used seamlessly in TwinCAT XAE.

The only requirement for use on any TwinCAT XAE system is the use of TwinCAT XAE version 3.1.4024.7 and higher. MATLAB®, a full Visual Studio installation, etc. are not necessary, since you work with objects and description files already compiled for TwinCAT. Simply copy the Engineering Repository folder to the engineering system of your choice. Compare [TwinCAT objects](#) [▶ 120]. Always keep the folder structure: `%TwinCATInstallDir%\3.1\Repository\<TE140x Module Vendor>\<ModelName>\<Version>\.`

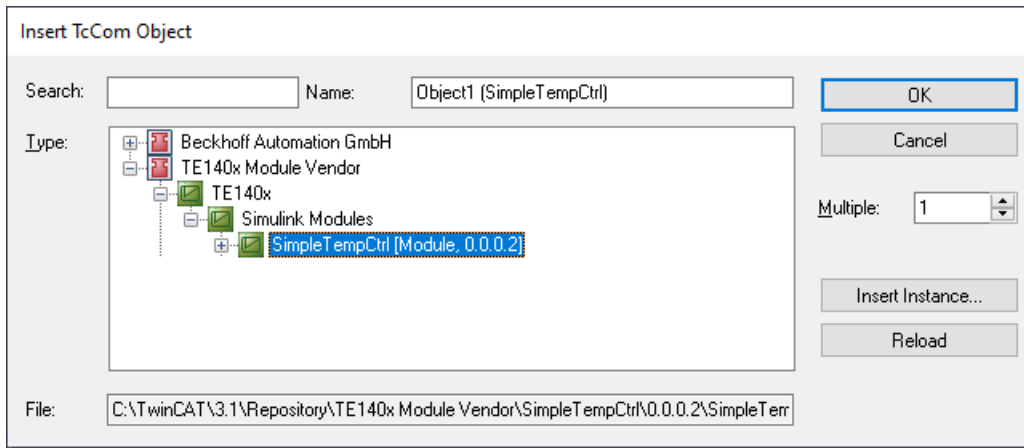
4.8.1 Working with the TcCOM module

Insert TcCOM in TwinCAT

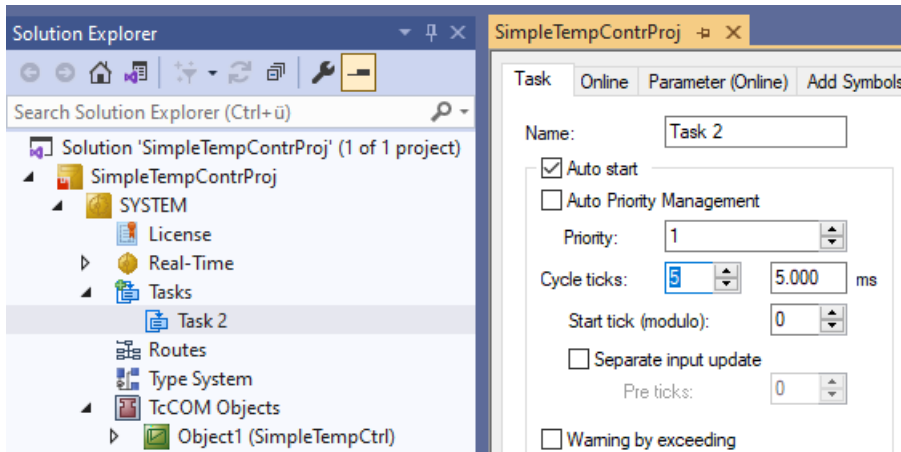
1. Open TwinCAT (TwinCAT XAE or TwinCAT in a Visual Studio environment).
2. Instantiate a new TcCOM object.



3. Select the desired object.

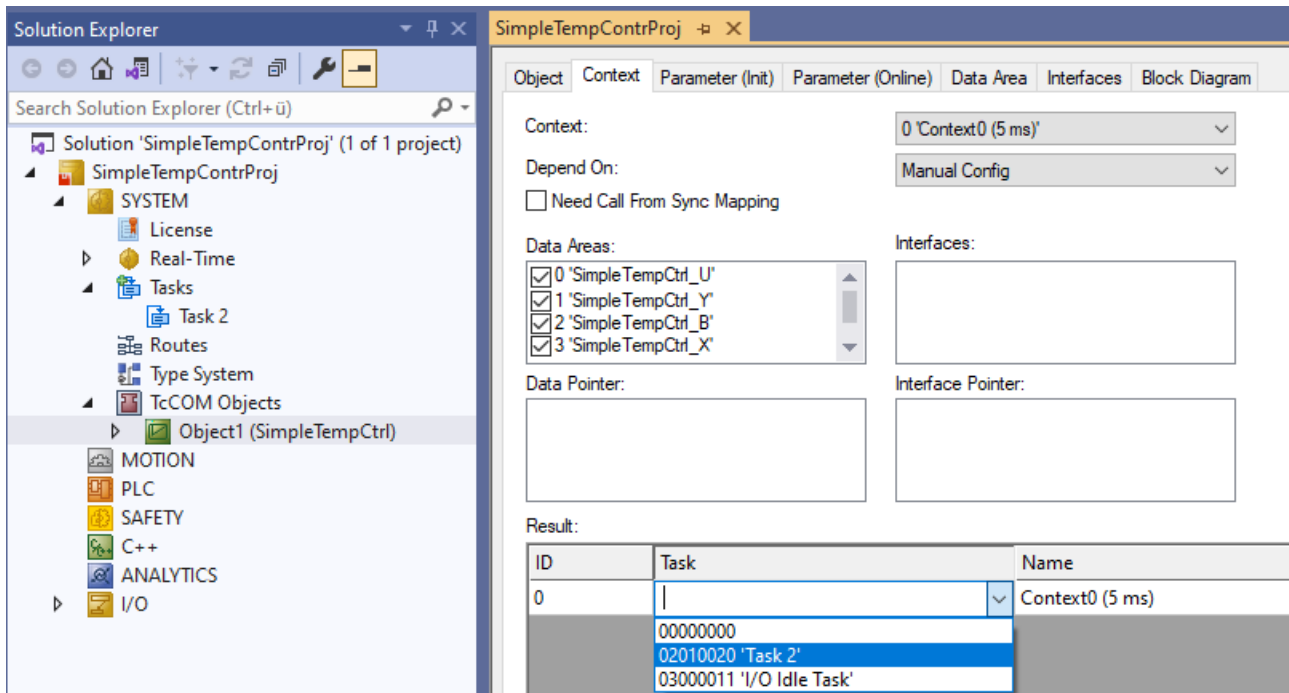


4. Create a cyclic task.

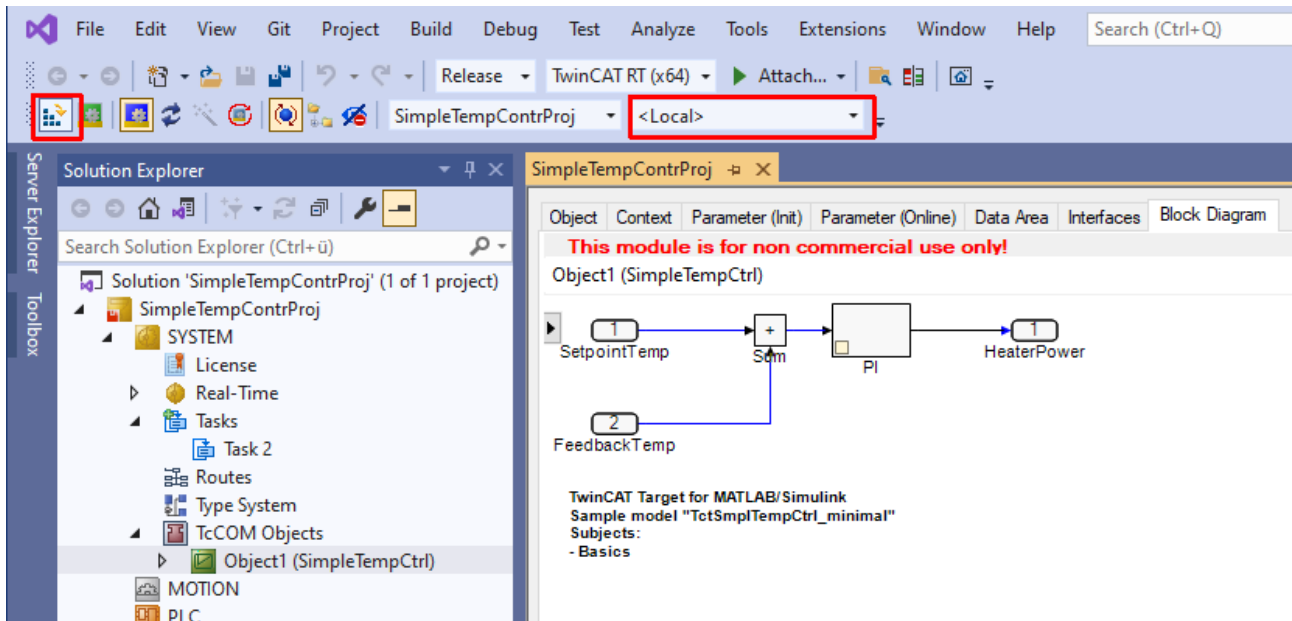


5. Assign the created task to your TcCOM instance.

Note that the cycle time of the task and the SampleTime in Simulink® (here 5 ms) match.

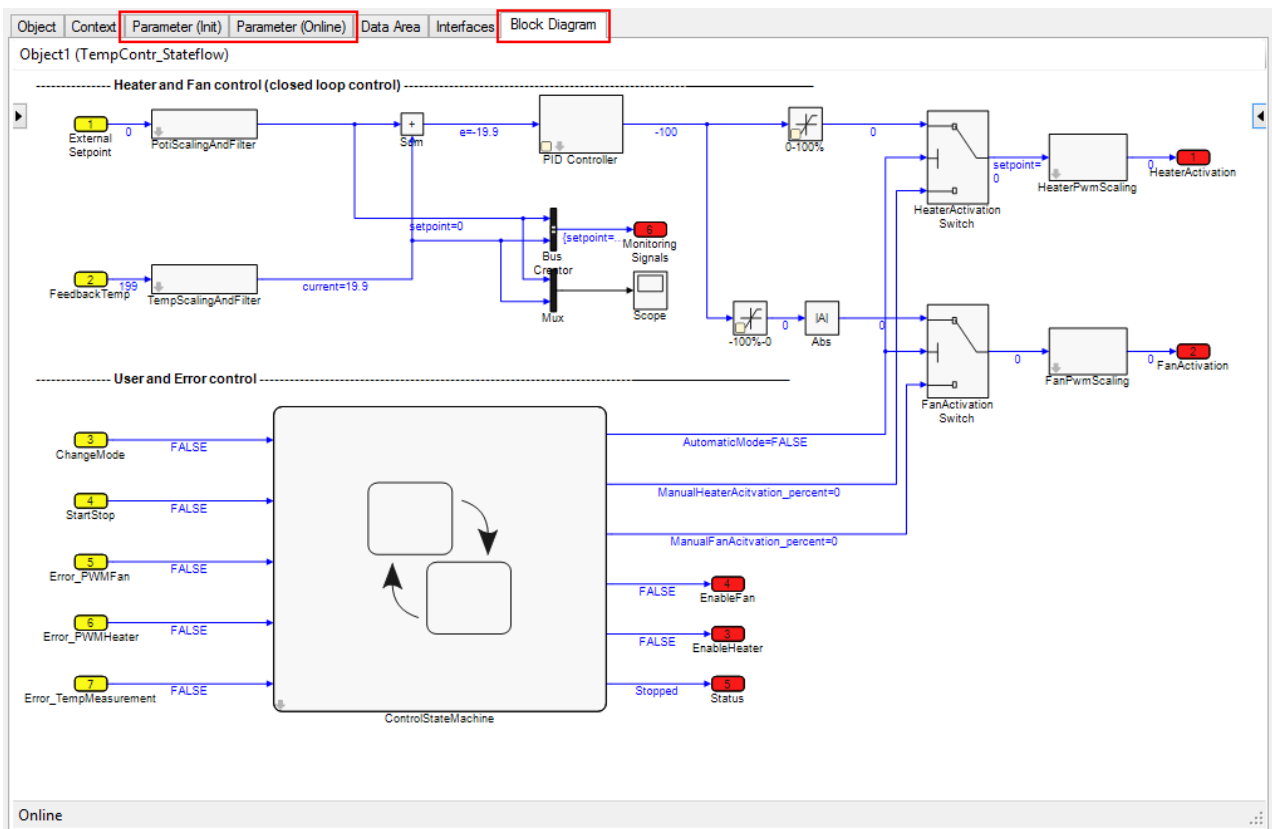


6. Activate the configuration.



4.8.1.1 Parameterization of a module instance

Given is an instance of a TcCOM, in which tabs can parameters be found?



You can view and change parameters in Parameter (Init), Parameter (Online) and around Block Diagram. In Data Area you can see the created DataAreas and their contents (parameter name and data type), but you cannot manually change any values here via the XAE.

See also [Best practice: access to TcCOM data \[147\]](#) for ways to access data on a TcCOM.

Default, Startup, Online and Prepared values

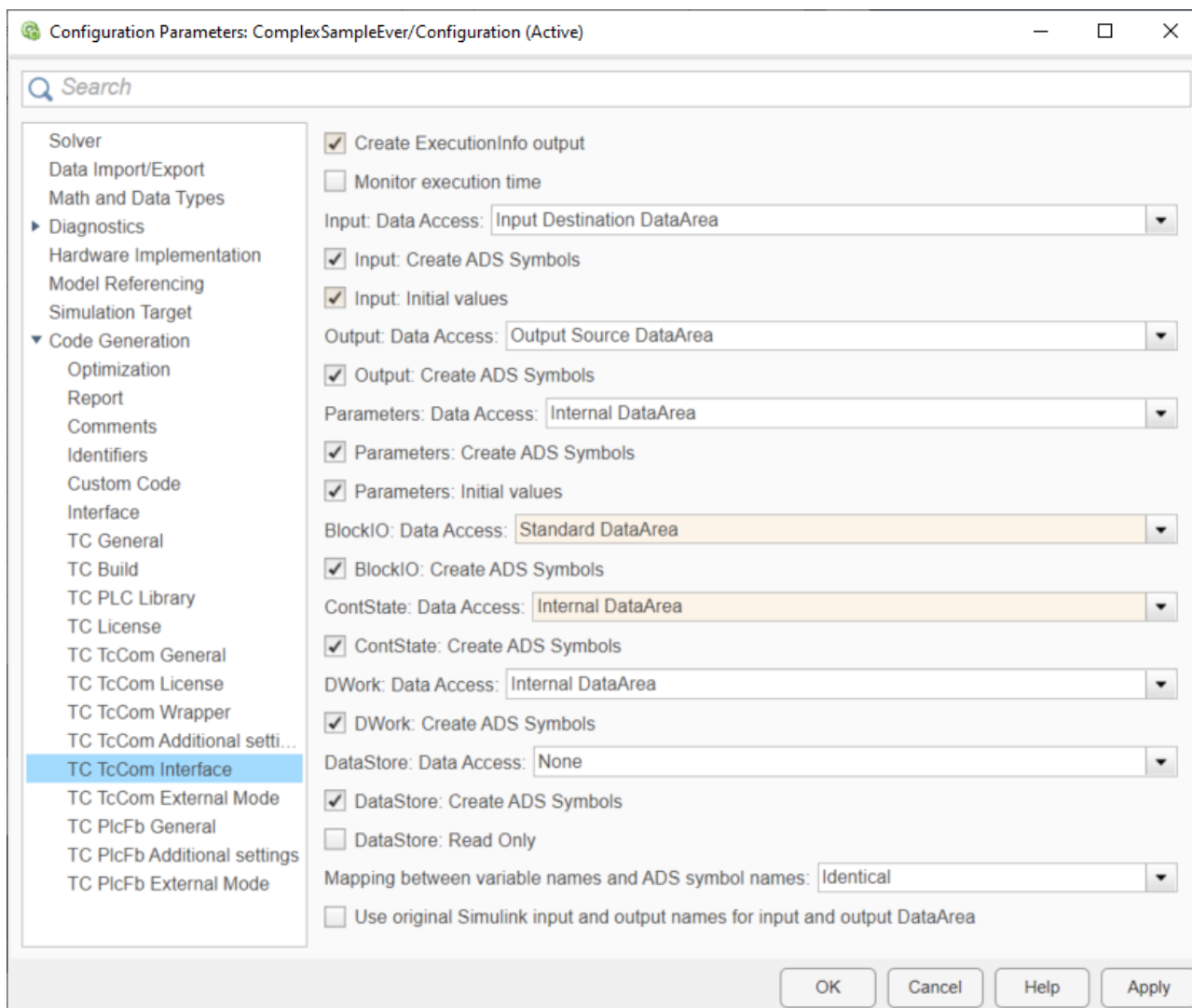
Parameters can have different states/properties. These are defined below:

- **Default values** are the parameter values during code generation (as they were set in Simulink®). They are stored unchangeably in the module description file (*.tmc), i.e. the description of the module class.
- **Startup values** are stored in the TwinCAT project file and written to the module instance. The startup values are located accordingly on the target system and define the values at the start of the module instance.
- **Online values** are only available if the TcCOM instance was started on the target system. They show the current parameter value in the running module. This value can also be changed during runtime.
- **Prepared values** can be specified whenever online values are available. With its help, several parameter values can be changed and written to the module at the same time.

Parameterization of a module instance

Example setting in TC TcCom Interface

To explain the parameterization of a module instance, a module with the following properties is assumed.



Properties of the TcCOM based on these settings:

- The structure of the model parameters <ModelName>_P is created as module parameters, because Parameters: Initial Values is enabled. As DataArea the model parameters are accessible if Code Interface packaging is not set to *Reusable function*.

i Enter model parameters as tunable

The Simulink Coder™ model parameters are set as "inlined" as the default behavior. This means that the object memory used is smaller and the generated code is optimized in terms of runtime, but often only a few parameters can be changed at runtime with this setting.

Use the *Configuration Parameter* under *Optimization > Default parameter behavior "Tunable"* so that you can parameterize your model at runtime.

- BlockIO is created as a standard DataArea and is therefore visible in the process image of the TcCOM and can be linked via DataPointer.
- Create ExecutionInfo output is enabled, accordingly there is another DataArea of type Output Source with the execution information of the instance.
- Create ADS Symbols is enabled on all DataAreas, so all parameters in the DataAreas are accessible by ADS symbol name.
- Module parameters are created for the inputs of the model, because Input: Initial Values is enabled.

For a description of the setting options, see [Configuration of data access to data of a TcCOM object \[▶ 143\]](#).

Possible parameter settings in Parameter (Init)

Module parameters can be found in Parameter (Init), which do not change cyclically but can be changed acyclically. No online values can be seen in Config mode, or when the TcCOM is not started.

Name	Value	CS	Type	PTCID
ModuleCaller	CyclicTask	✓	TcMgSdk.ModuleCaller	0x00000002
StepSizeAdaptation	RequireMatchingTaskCycleTime	✓	TcMgSdk.StepSizeAdapta...	0x00000004
ExecutionSequence	UpdateBeforeOutputMapping	✓	TcMgSdk.ExecutionSeque...	0x00000005
Execute	TRUE	✓	BOOL	0x00000006
- TempCtrl_U	...	✓		0x80000000
.FeedbackTemp	5		INT	
- TempCtrl_P	...	✓		0x84000000
.Kp	53.0		LREAL	
.Tn	200.0		LREAL	
.sawtooth_rep_seq_y[0]	0.0		LREAL	
.sawtooth_rep_seq_y[1]	60.0		LREAL	
.Pl_y_max	60.0		LREAL	
.Pl_y_min	0.0		LREAL	
.InternalSetpoint_Value	37.0		LREAL	
.scale_Gain	0.1		LREAL	
.Integrator_IC	0.0		LREAL	
.Constant_Value	0.1		LREAL	
.LookupTable1_bp01Data[0]	0.0		LREAL	
.LookupTable1_bp01Data[1]	0.1		LREAL	

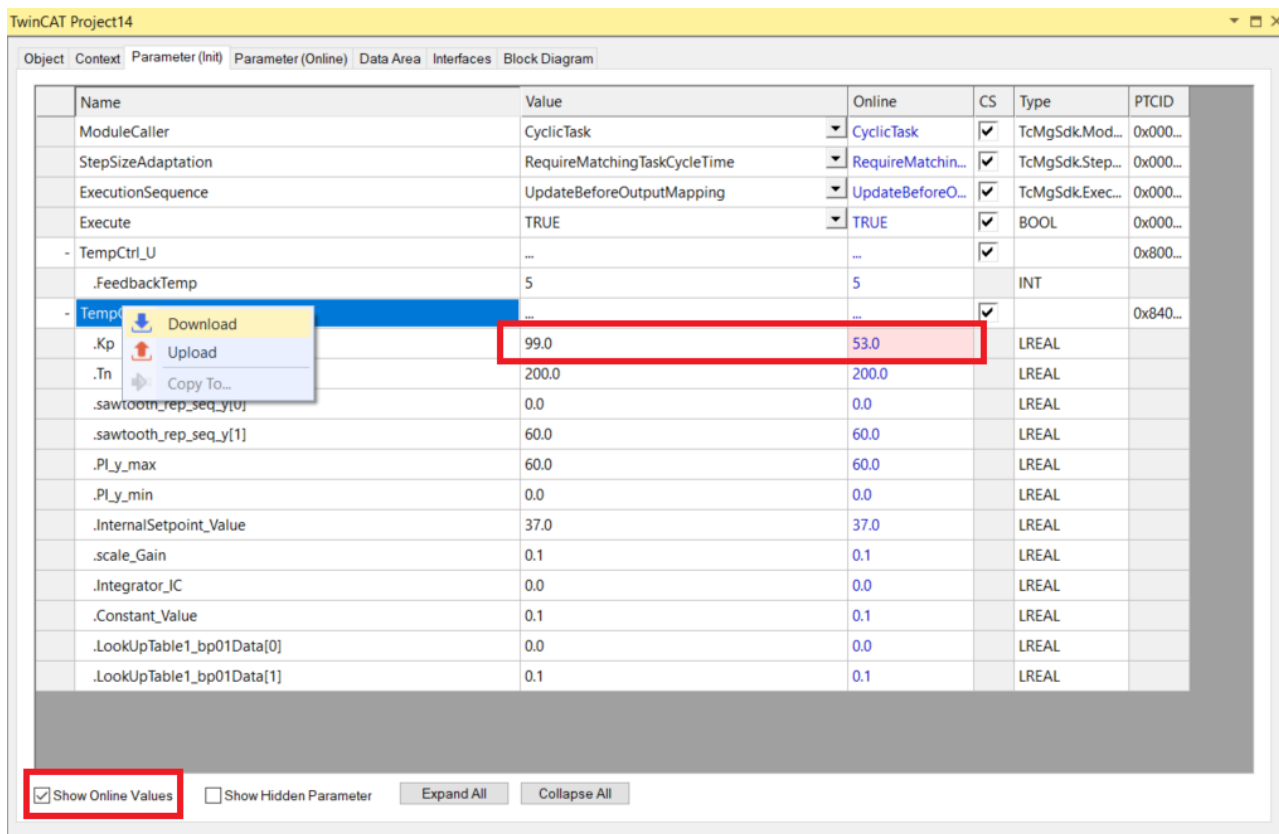
Define Startup Values

Startup Values can be set in the **Parameter (Init)** representation or in the **Block Diagram** representation (see in the following chapter). To do this, select the entry in the **Value** column that you want to adjust on the module instance and enter a value. The value set in this way is only available in the project file in the XAE at this time. Activate Configuration loads the set value (with the overall project) to the target system.

Settings such as the *ModuleCaller*, *StepSizeAdaption* or *ExecutionSequence* cannot be changed at runtime of the module, but they can only be defined as startup values. Other module parameters such as *Execute* or the model parameters can also be changed online.

Change values online

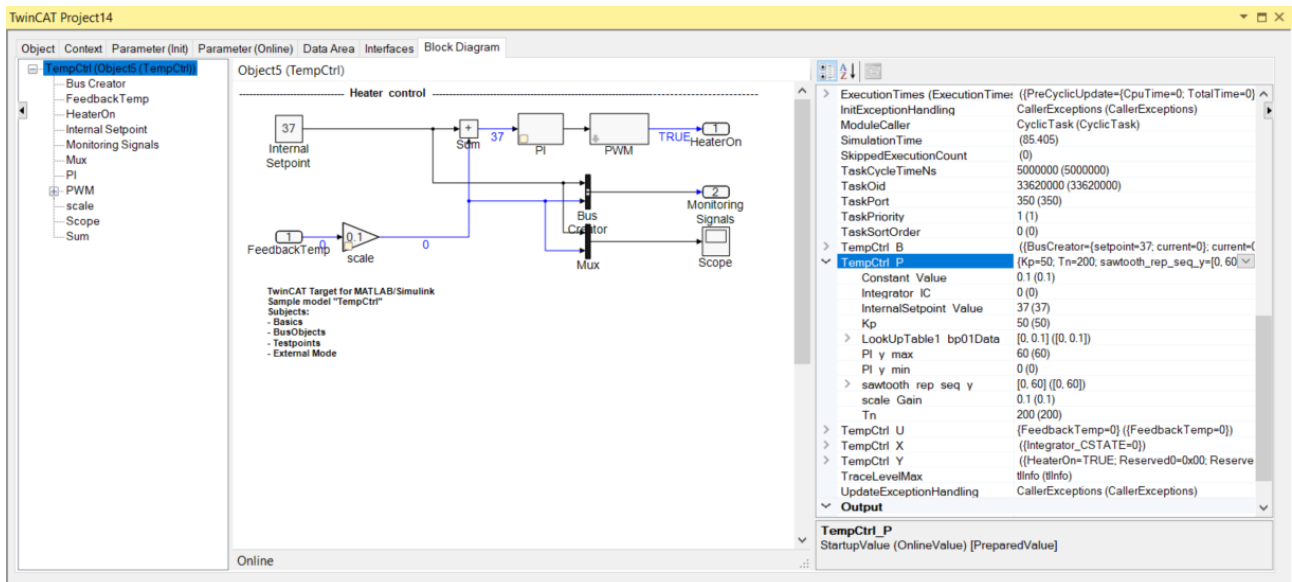
If the TcCOM module is active, you can use **Show Online Values** to make the current values of the instance visible. You can enter a new value in the **Value** column, making it a Prepared Value. If the Prepared Value is not yet loaded on the target system, the field with deviation between Value and Online appears with red marking. Once all changes have been made, you can right-click at the structure level and use "Download" to set the prepared values in the target system. Note that this does not change the startup value of the instance on the target system. To do this, you must first activate the current configuration on the target system. Likewise, you can use "Upload" to set the current Online Values as Startup Values in the project. Again, the change is not active on the runtime system until the project is compiled and downloaded, see [Best practice: access to TcCOM data \[► 147\]](#).



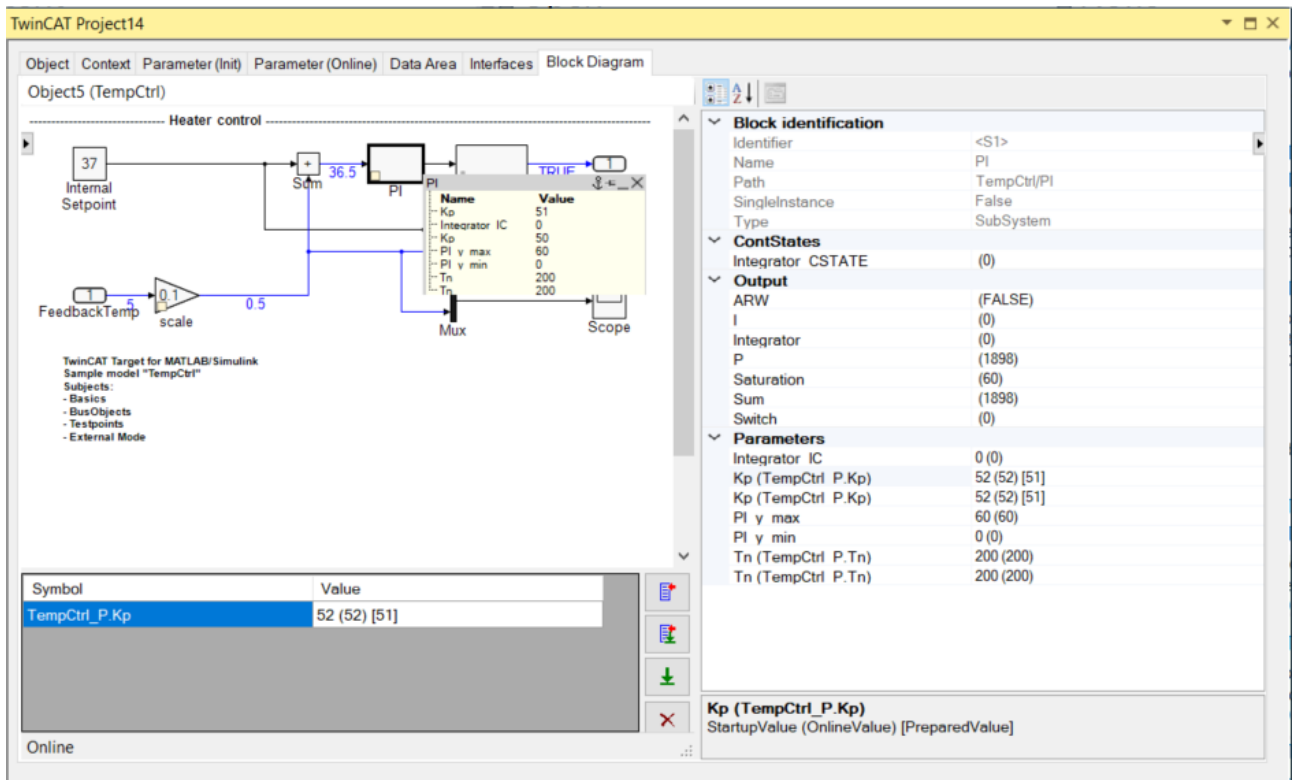
Possible parameter settings in Block Diagram

All parameters in the parameter range (right side of the window) are only displayed in the TwinCAT 3 block diagram if you are in the top level of the block diagram (" \langle root \rangle "). If you are in a subsystem or if you have selected a block, only the parameters of the active block are displayed.

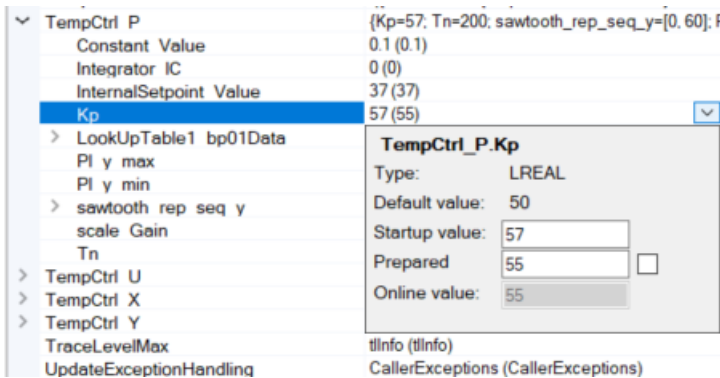
In the block diagram you have the possibility to read and write online values as well as to change Startup Values.



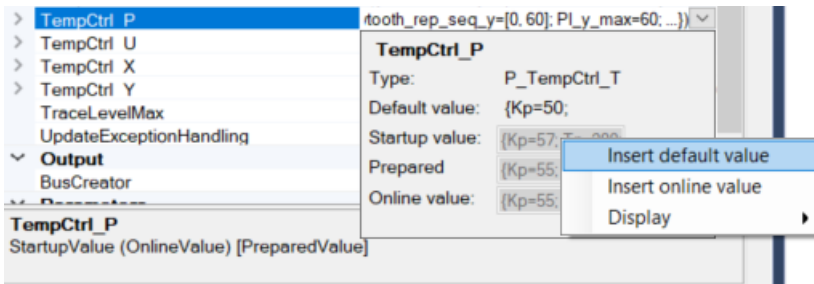
Select a block with a yellow dot in the lower left corner. If you click directly on the yellow dot, a context menu opens in which you can see the online values of the block and also change them. If you change a value, it will be entered in the Prepared list. When you have made all the changes, you can write the prepared values to the target system. You can select in the Prepared list whether you want to set the prepared values as Online and/or Startup values.



You can also go over the parameter list in the right area of the block diagram and change values here. Find the parameter you want to change and click the down arrow on the right side of the list. Here you can make changes in the editable fields. If you move the mouse pointer over the parameter name (TempCtrl_P.Kp in the following diagram), the ADS address information is displayed. By right-clicking on the parameter name, you can copy the parameter's ADS address information to the clipboard.



You can also change entire structures. Select the <ModelName>_P structure, for example, i.e. the structure that contains all model parameters, and select the downward pointing arrow here. A context menu appears by right-clicking on **Startup Values**, for example. Here you can set all current online values of the structure as Startup Values, or you can reset the startup list to the default values.



Interaction with DataAreas

You cannot change DataAreas via the XAE. You can only see the DataArea type (see Type column) and deduce which interaction options are possible with the DataArea, see [Configuration of data access to data of a TcCOM object](#) [▶ 143].

Via the CS column you can see whether ADS symbol names are to be generated for this DataArea. CS is editable by the user. If CS is enabled, you can use the Target Browser, for example, to search for the ADS symbols and include them in a scope configuration. If CS is disabled, you can only access the DataArea data via Index Group and Index Offset. Index Group is the Object ID of the instance (see Object tab) and Index Offset is displayed in the CD / Elements column.

Area No	Name	Type	Size	CS	CD / Elements
+ 0 (0)	TempCtrl_U	InputDst	2	<input checked="" type="checkbox"/>	<input type="checkbox"/> 1 Symbols
+ 1 (0)	TempCtrl_Y	OutputSrc	24	<input checked="" type="checkbox"/>	<input type="checkbox"/> 2 Symbols
+ 2 (0)	TempCtrl_B	Standard	120	<input checked="" type="checkbox"/>	<input type="checkbox"/> 15 Symbols
+ 3 (0)	TempCtrl_X	Internal	8	<input checked="" type="checkbox"/>	<input type="checkbox"/> 1 Symbols
- 4 (0)	TempCtrl_P	Internal	96	<input checked="" type="checkbox"/>	<input type="checkbox"/> 10 Symbols
	Kp	LREAL	8.0 (Offs: 0.0)	<input checked="" type="checkbox"/>	0x84000000
	Tn	LREAL	8.0 (Offs: 8.0)	<input checked="" type="checkbox"/>	0x84000008
	sawtooth_rep_seq_y	matrix_2_real_T	16.0 (Offs: 16.0)	<input checked="" type="checkbox"/>	0x84000010
	PI_y_max	LREAL	8.0 (Offs: 32.0)	<input checked="" type="checkbox"/>	0x84000020
	PI_y_min	LREAL	8.0 (Offs: 40.0)	<input checked="" type="checkbox"/>	0x84000028
	InternalSetpoint_Value	LREAL	8.0 (Offs: 48.0)	<input checked="" type="checkbox"/>	0x84000030
	scale_Gain	LREAL	8.0 (Offs: 56.0)	<input checked="" type="checkbox"/>	0x84000038
	Integrator_IC	LREAL	8.0 (Offs: 64.0)	<input checked="" type="checkbox"/>	0x84000040
	Constant_Value	LREAL	8.0 (Offs: 72.0)	<input checked="" type="checkbox"/>	0x84000048
	LookUpTable1_bp01Data	matrix_2_real_T	16.0 (Offs: 80.0)	<input checked="" type="checkbox"/>	0x84000050
+ 5 (0)	ExecutionInfo	OutputSrc	240	<input type="checkbox"/>	<input type="checkbox"/> 4 Symbols

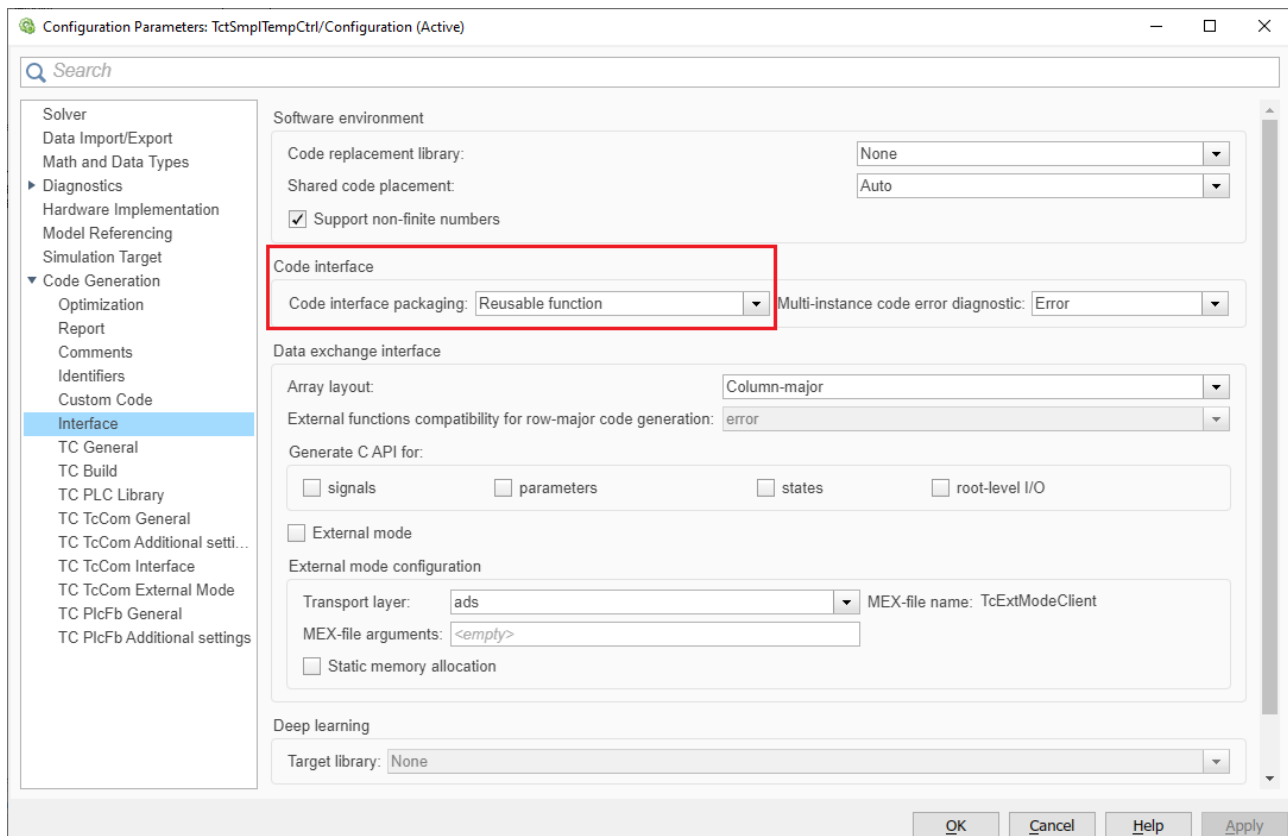
For more information on interacting with DataAreas, see [Configuration of data access to data of a TcCOM object \[► 143\]](#) and [Best practice: access to TcCOM data \[► 147\]](#).

4.8.1.2 Parameterization of several module instances

In the above section it was described that an instance of a TcCOM can be parameterized in TwinCAT, even deviating from the parameters in Simulink®.

If several instances of a TcCOM are used in a TwinCAT Solution, different options exist with regard to the individual parameterization of the instances. To implement the following three options, you must use the **Code interface packaging** setting in Simulink®.

Make sure that the setting *Default parameter behavior* is set to *Tunable* under *Optimization*.



✓ **All instances should have the same parameters.**

1. Set the parameter to "Reusable function". This is the default value when selecting the target *TwinCatGrt.tlc*.
2. Create several instances of your TcCOM in TwinCAT.
3. Under Parameters (Init), configure the <ModelName>_P_Sharing parameter to *define* or *inherit*. Define specifies the parameterization of all dependent instances configured with *inherit*.

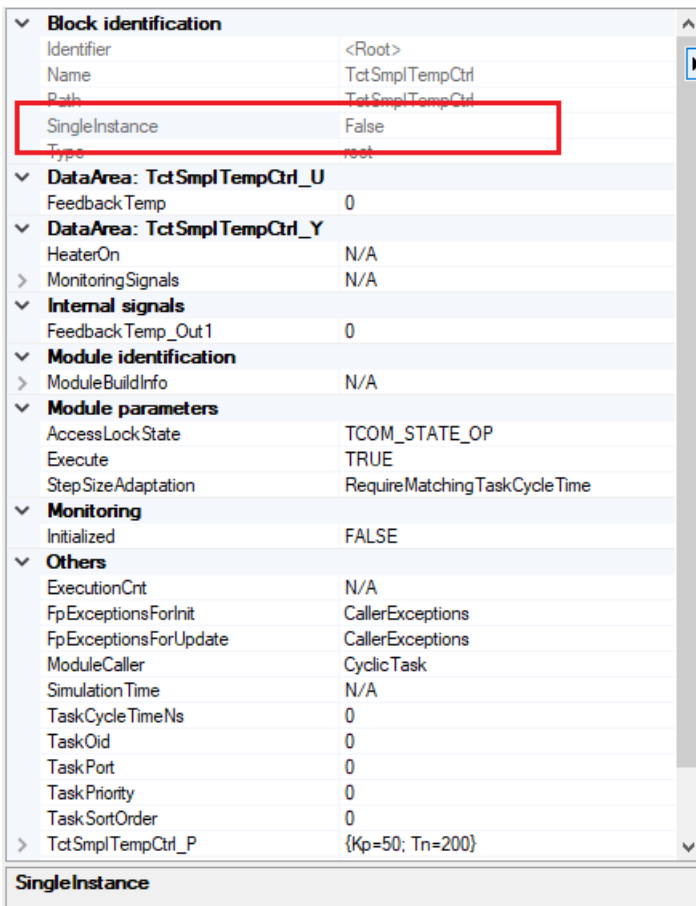
⇒ Only one instance with *define* may be configured.

✓ **It should be possible to parameterize each instance individually.**

1. Set the parameter to "C++ class".
 2. Create several instances of your TcCOM in TwinCAT.
- ⇒ No <ModelName>_P_Sharing parameter is generated. Each instance can be parameterized individually.

✓ **Only one instance should be allowed in the project.**

1. Set the parameter to "Nonreusable function".
 - ⇒ If you create several instances of your TcCOM in TwinCAT, you receive an error message when activating the solution.
 - ⇒ Whether an instance of a TcCOM can be instantiated multiple times can be seen in the TC3 BlockDiagram.
2. To do this, go to the parameter range on the right. Under Block Identification a parameter "SingleInstance" is visible.
 - ⇒ The value False means multi-instantiable. Accordingly, True means one instantiation.



● Settings also apply to the use of the PLC function blocks

I The setting of the *Code Interface Packaging* has the same meaning for the use of the TcCOM as well as for the use of the PLC function blocks.

● Open sample for parameterization

I In MATLAB®, open the Multi Instance sample: `TwinCAT.ModuleGenerator.Samples.Start('Multi_Instance')`

4.8.1.3 Working with the block diagram in TwinCAT

4.8.1.3.1 Simulink®-TcCOM

If a TwinCAT object was created with the TwinCAT Target for Simulink® and the block diagram export was executed in the process, the block diagram of the Simulink® model can be displayed as a control in the TwinCAT XAE.

4.8.1.3.1.1 Using the block diagram

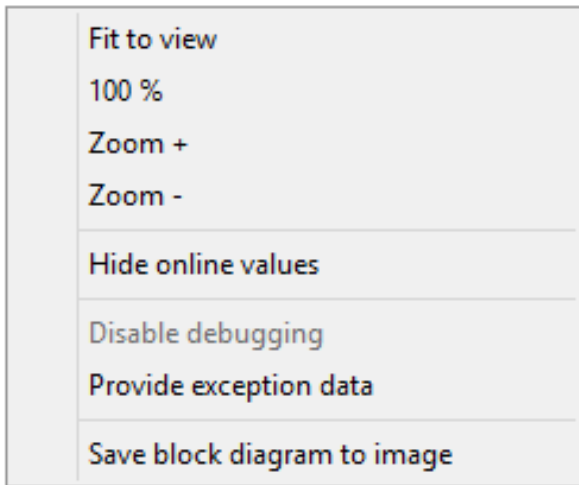
The block diagram export can be configured during generation of a TcCOM module from MATLAB® or Simulink®. If the export was enabled, the block diagram can be found in the TwinCAT development environment under the "Block Diagram" tab of the module instance.

Using shortcuts, drag & drop and a context menu you can navigate through the hierarchy of the TcCOM module, view parameter values, display signals values and obtain optional additional debug information.

Shortcut functions:

Shortcut	Function
Space	Zoom to current size of the block diagram tab

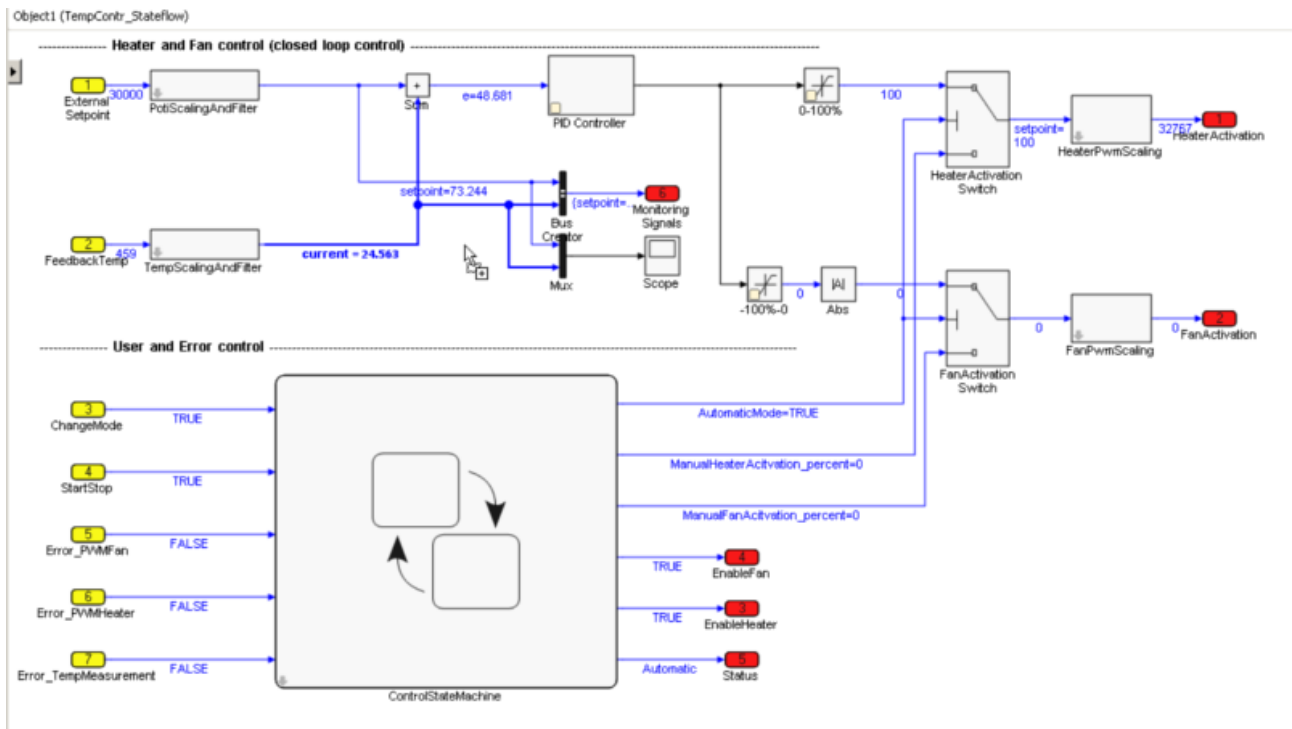
Shortcut	Function
Backspace	Switch to the next higher hierarchical level
ESC	Switch to the next higher hierarchical level
CTRL + "+"	Zoom in
CTRL + "-"	Zoom out
F5	Attach Debugger (System- > Real-Time -> C++ Debugger -> Enable C++ Debugger must be activated)

Context menu functions:**4.8.1.3.1.2 Display signal curves**

For verification and troubleshooting it is often helpful to display signal curves. The block diagram offers the following options:

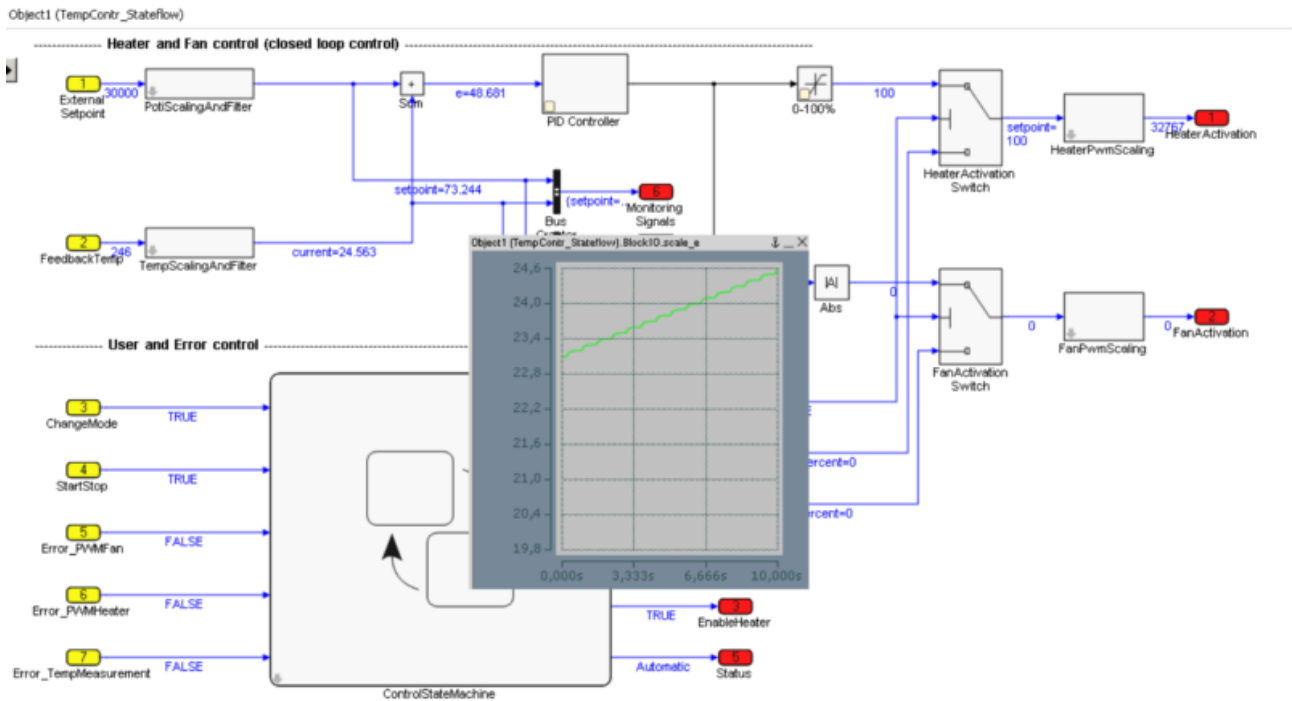
Display signal curves in the block diagram

The block diagram offers an option to display signal curves in a window. To this end, drag and drop a signal or block into a free area of the block diagram.



Create a scope in the block diagram

After the drop, a scope window opens in the block diagram.



Display the scope in the block diagram

The title bar of the scope window offers the following options:

	Close window
	Keep window in the foreground across all block diagram hierarchies
	Minimize window to the title bar



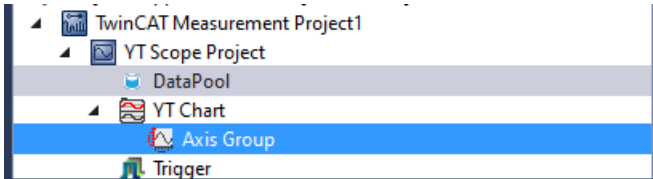
Displaying the scope in the [block diagram control](#) [▶ 242] requires a Scope View Professional (TE1300) license. No Scope View Professional license is required in TwinCAT XAE.

When creating a scope window in the block diagram for a Simulink® bus, all signals of the bus are directly displayed in the scope window.

The scope window in the block diagram can be used for a quick overview. For more detailed analyzes, it is advisable to analyze the signals in a TwinCAT Measurement project.

Display signal curves in TwinCAT 3 Scope

If the drop is not made to the block diagram control but to an Axis Group in a TwinCAT Measurement project, the signal is added there.



Add a signal in a TwinCAT 3 Scope

4.8.1.3.1.3 Module parameterization in the block diagram

To parameterize a TcCOM instance, the **parameter window** can be used directly in the block diagram. In addition, the Property table can be used, which can be expanded or collapsed on the right-hand edge of the block diagram. A basic distinction is made between different parameter values:

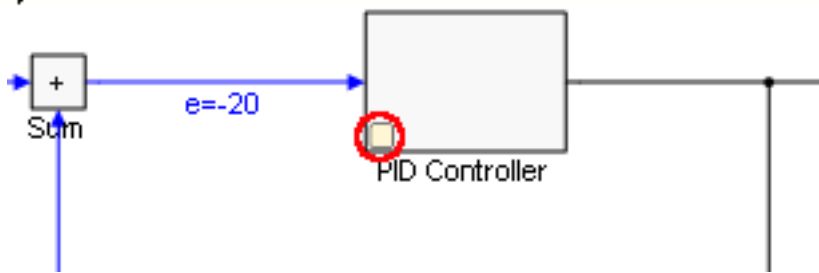
"Default", "Startup", "Online" and "Prepared"

The following value types can be found in the drop-down menu of the Property table of the block diagram:

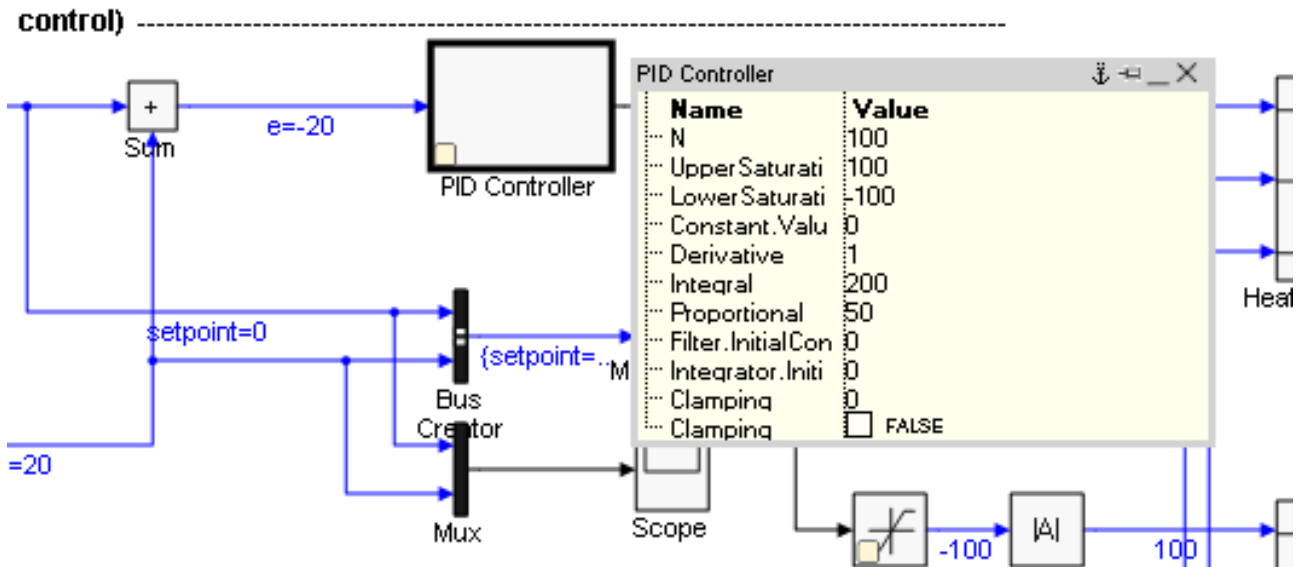
- **Default values** are the parameter values during code generation. They are invariably stored in the module description file and enable the manufacturing settings to be restored after parameter changes.
- **Startup values** are stored in the TwinCAT project file and downloaded to the module instance as soon as TwinCAT starts the module instance.
Startup values for the input process image can also be specified in Simulink® modules. This allows the module to be started with non-zero input values, without the need for linking the inputs with other process images. Internal signals and output signals have no starting values, since they would, in any case, be overwritten in the first cycle.
- **Online values** are only available if the module was started on the target system. They show the current parameter value in the running module. This value can also be changed during runtime. Although in this case the corresponding input field has to be enabled via the context menu, in order to prevent accidental inputs.
- **Prepared values** can be specified whenever online values are available. They can be used to save various values, in order to write them consistently to the module. If prepared values have been specified, they are displayed in a table below the block diagram. The buttons to the right of the list can be used to download prepared values as online values and/or save them as starting value, or delete them.

Parameterization in the block diagram

Parameterizable blocks are marked with a yellow box in the block diagram.



Double-clicking on the block or a single click on the yellow box brings up a window with the parameters that can be changed.



If a value is changed, it can be applied with the following keyboard commands:

CTRL + Enter	Set online value directly
SHIFT + Enter	Set startup value
Enter	Set prepared value

The icons in the title bar have the following functions:

	Close window
	Keep window in the foreground across all block diagram hierarchical levels
	Keep window open at the current block diagram hierarchical level
	Minimize window to title bar

4.8.1.3.1.4 Debug

Different ways are available to find errors within a TcCOM module created with MATLAB®/Simulink®, or to analyze the behavior of the module within the overall architecture of the TwinCAT project.

Debugging in the block diagram

If the block diagram was exported during generation of the TcCOM module, it can be displayed in the TwinCAT development environment and used for debugging within the corresponding module instance, for example. To do so, the block diagram uses the Microsoft Visual Studio debugger, which can be linked with the TwinCAT runtime via the TwinCAT debugger port. Attach the debugger as described in the C++ section under Debugging.

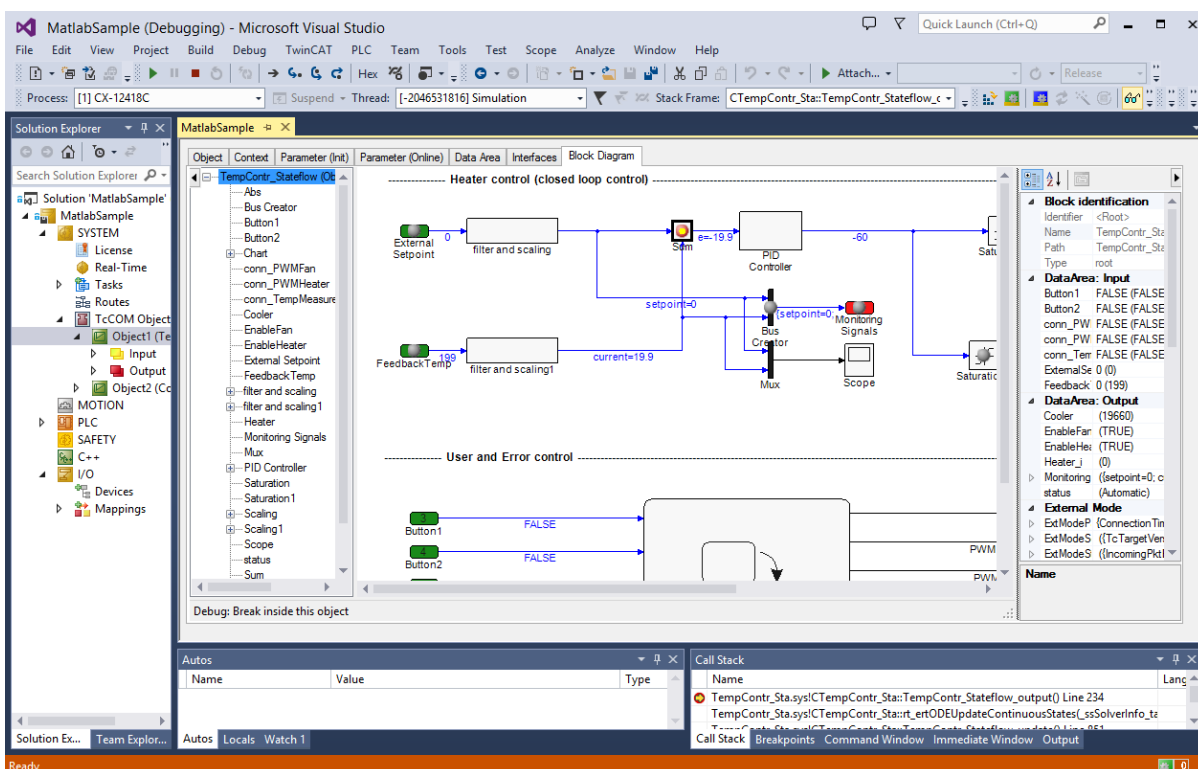
Prerequisites for debugging within the block diagram are:

- The C/C++ source code of the TcCOM module must be present on the engineering systems, and the Visual Studio debugger must be able to find it. Ideally, debugging should take place on the system on which the code was generated. If the module was created on another system, the associated C/C++ source code can usually be made known by integrating the Visual Studio project into the TwinCAT C++ section. The file <ModelName>.vcxproj is located in the build directory, see [Which files are created automatically during code generation and publishing?](#) [▶ 59]
- The module must have been created with the **Debug** configuration. When publishing takes place directly after the code generation, select the **Debug** setting in the [Module generation \(Tc Build\)](#) [▶ 22] section under **publish configuration**. When publishing the module from the C++ section in TwinCAT, the debugger in the C++ node of the solution must be enabled; see C/C++ documentation, Debugging.
- During code generation, the options **Export block diagram** and **Export block diagram debug information** must be enabled in the coder settings under **Tc Advanced**.
- In the TwinCAT project, the debugger port must be enabled, as described in: TwinCAT 3 C++ Enable C++ debugger.

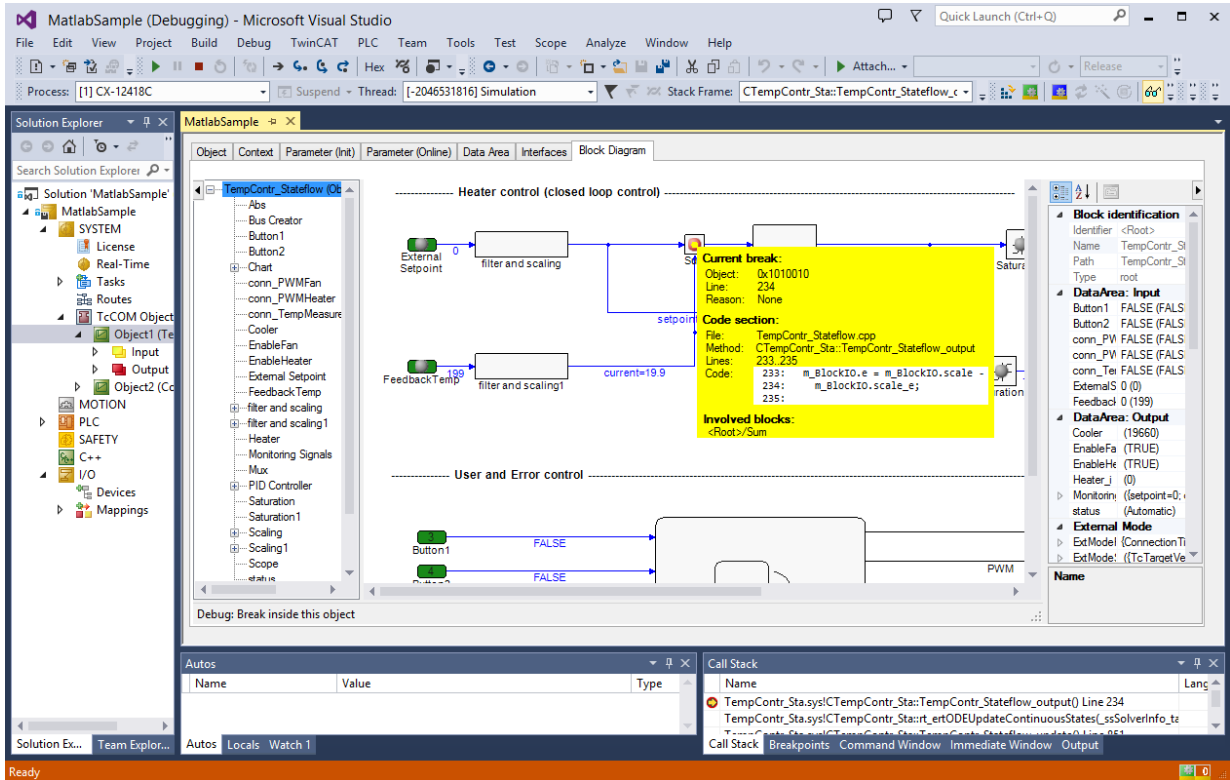
Setting breakpoints in the block diagram

1. After attaching the debugger to the TwinCAT runtime, the possible breakpoints are assigned to the blocks in the block diagram and represented as points. Clicking on the desired breakpoint activates it, so that execution of the module instance is stopped next time the associated function block is executed. The color of the point provides information about the current state of the breakpoint:

- Gray: breakpoint inactive
- Red: breakpoint active. The program is stopped next time this function block is executed
- Yellow dot in the middle: breakpoint hit. Program execution is currently stopped at this point
- Blue dot in the middle: breakpoint hit (as yellow), but in a different instance of the module.



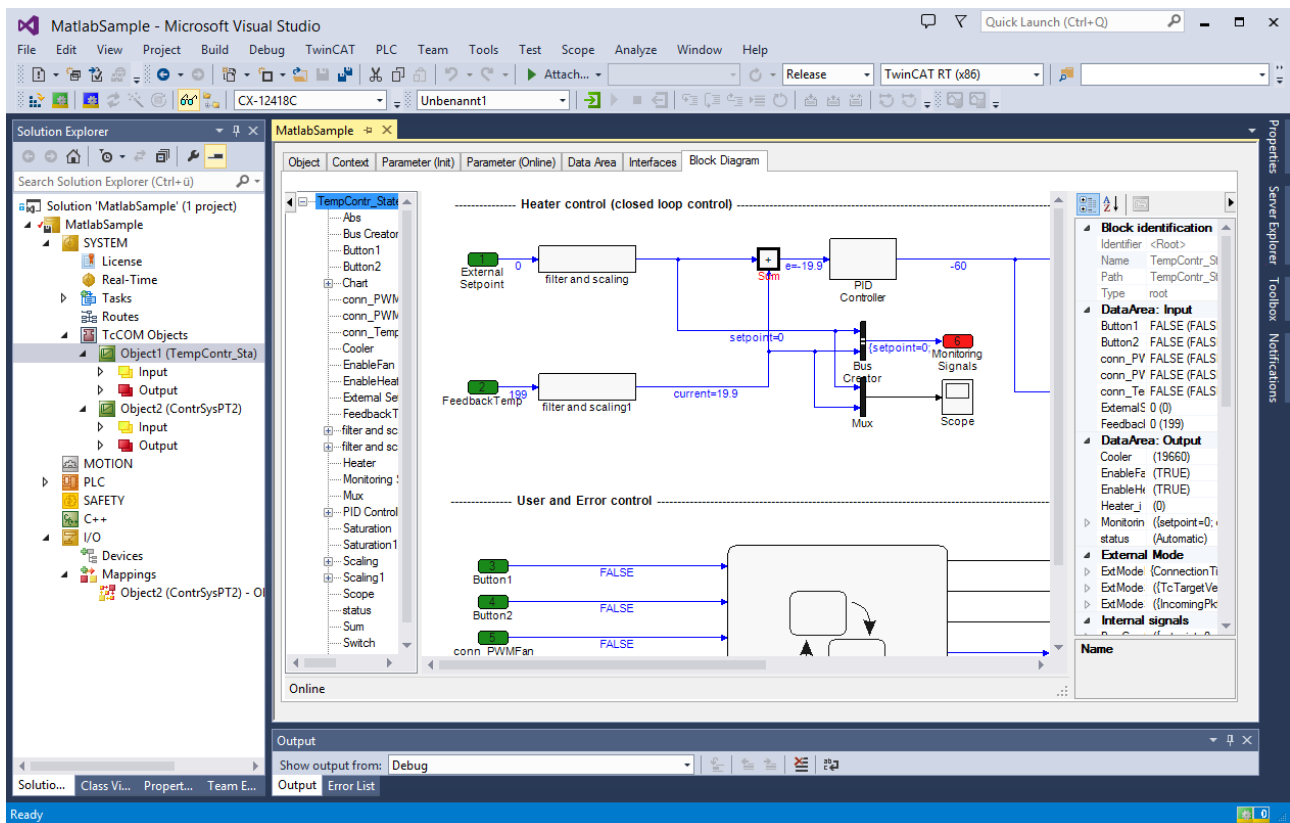
- Additional information, such as the corresponding C++ code section, can be found in the tooltip for the breakpoint:



i Breakpoints are not always assigned to a single function block. In many cases, the functions of several blocks are consolidated in a code section or even a line in the underlying C++ code. This means that several blocks can share the same breakpoint. Therefore, activation of a breakpoint in the block diagram may also result in changes in the point display in other blocks.

Evaluating exceptions

If exceptions occur during processing of a TcCOM module, such as division by zero, the point at which the exception occurred can be shown in the block diagram. To this end, the TcCOM module must meet the above requirements, and the C++ debugger must be enabled in the TwinCAT project (TwinCAT 3 C++ Enable C++ debugger). After the debugger has been attached, which may be done before the exception has occurred or indeed after, the block that caused the exception is highlighted in the block diagram, provided the line of code responsible for the exception can be allocated to a block. The name of the function block is shown in red, and the function block itself is marked in bold.



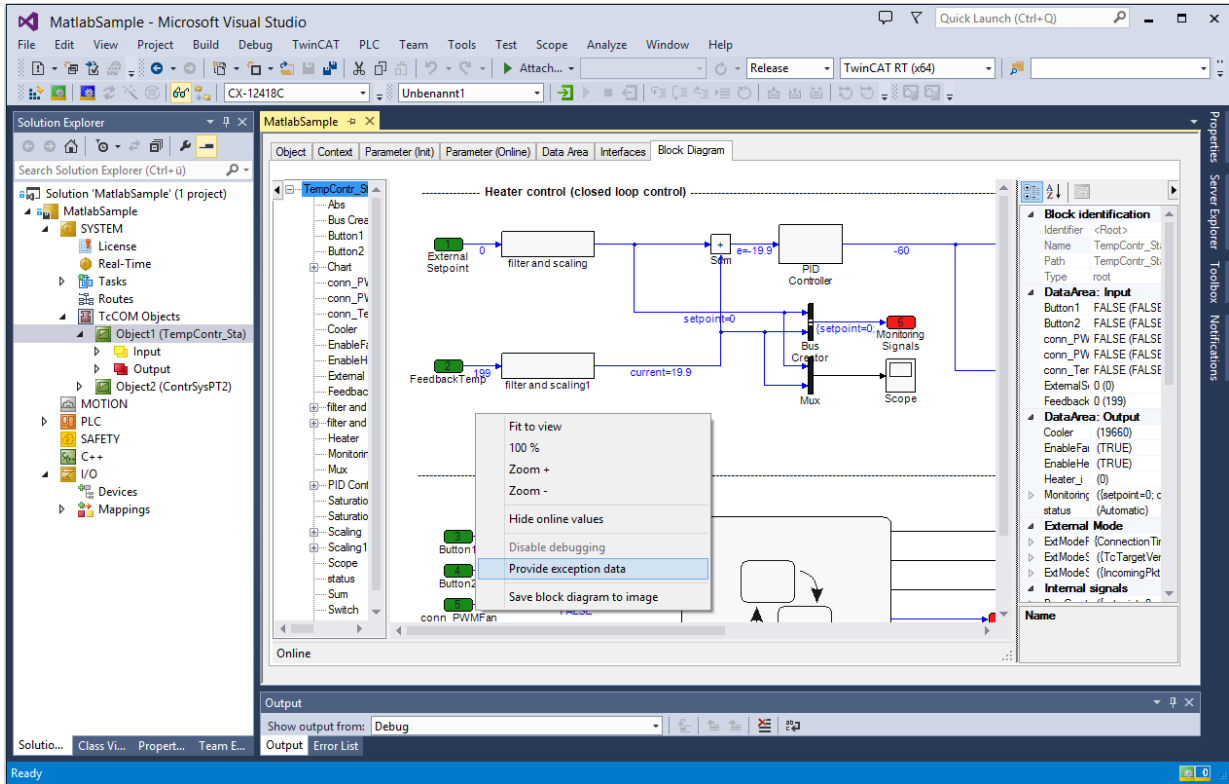
Manual evaluation of exceptions without source code

Even if the module source code is not available on the engineering system or the C++ debugger was not activated, you can highlight the error location in the block diagram once an exception has occurred.

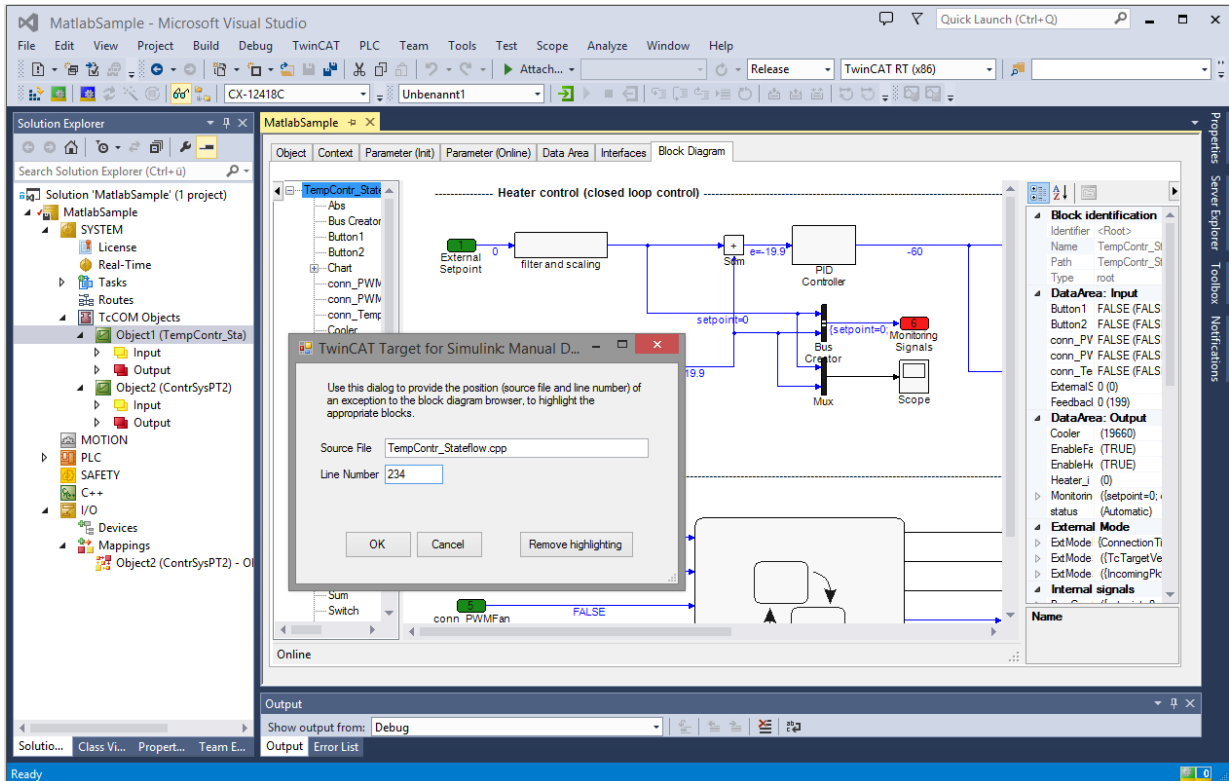
Typically, an error message will always be generated when an error occurs, indicating the source file and the line in the source code. In many cases, this information can be used to allocate an exception to a block in the block diagram. To do this, you can proceed as follows:

- ✓ A prerequisite for highlighting the error location within the block diagram is that debug information was generated (option **Export block diagram debug information** in the coder settings under **Tc Advanced**).

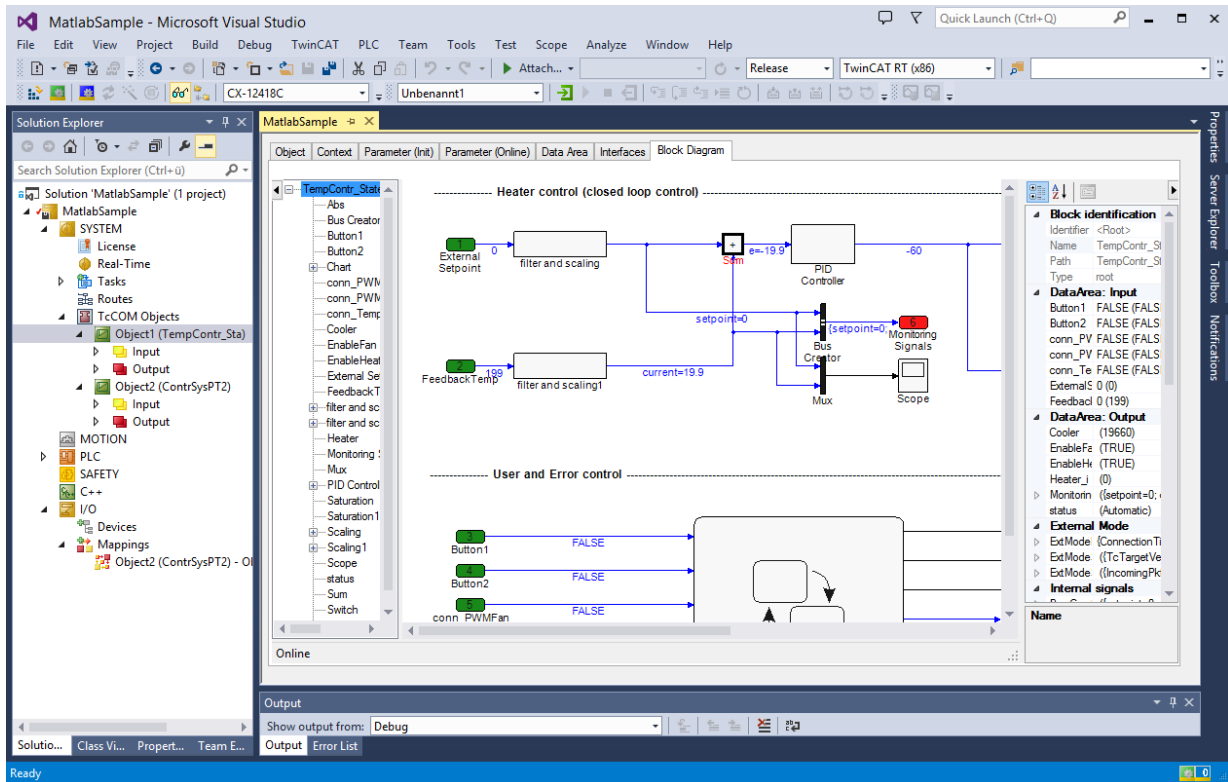
3. From the context menu of the block diagram select the entry **Provide exception data**:



4. In the dialog that opens, enter the source code file and line number provided in the error message:



5. The name of the function block associated with the line number is displayed in red, and the function block itself is marked in bold:



4.8.1.4 Online change of TcCOM at runtime

With the "Online Change" you can exchange TcCOM objects at runtime, i.e. without TwinCAT stop, on a runtime PC.

For a description of the Online Change for TcCOM, see: [Creation of versioned drivers \[► 137\]](#).

i Open sample in MATLAB®

Open a sample in MATLAB® with:

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

4.8.1.5 To File Block and MAT-file logging

You can configure your Simulink® models to generate MAT files on the file system of the runtime PC in the form of a TcCOM object in TwinCAT runtime.

i Observe write permissions on the runtime PC

Note the write permissions on the path you want to write to.

MAT-file logging

Configuration from Simulink®:

- Enable MAT-file logging at **Code Generation > Interface > MAT-file logging**, see [MathWorks® documentation](#).
- Enable **Code Generation > TC General > Load DataExchangeModules**.

If a TcCOM object is created with these settings and activated in a TwinCAT configuration on a runtime system, model signals are saved in a MAT file according to the properties specified by MathWorks®.

The MAT file is created on the file system of the runtime PC in the TwinCAT boot directory.

To File Block

Configuration in Simulink®:

- Enable MAT-file logging at **Code Generation > Interface > MAT-file logging**.
- Enable **Code Generation > TC General > Load DataExchangeModules**.
- In the **ToFile Block** specify the fullpath, e.g. *C:\Logs\MyLog.mat*. If you only specify the file name the MAT-file will be created in the TwinCAT boot directory.
- Select in the **Block Parameters** of the **To File Block** *Save format: Array*.

If a TcCOM object is created with these settings and activated in a TwinCAT configuration on a runtime system, a MAT file is created at the configured position.

- The MAT-file is filled with new data at runtime and grows in its required memory size accordingly over time.
- The terminate method to complete the MAT file is performed in the Transition Preop Init. To do this, either move the TcCOM object to the Init state or set the TwinCAT Runtime to Config mode.

NOTICE

Sufficient storage space

Note that you must keep enough storage space on the target system to avoid unpredictable behavior of the runtime PC.

TwinCAT File Writer

You can precisely control data logging at runtime with the TwinCAT File Writer. The TwinCAT File Writer can terminate file packages of defined size and generate a specified set of files on the runtime system. Thus, there is no danger of reaching the limit of the target system memory.

- Maximum size of .mat files adjustable
- Maximum number of .mat files adjustable
- Optionally pause writing via a TcCOM module parameter
- Does not support all data types

Documentation of the block can be found [here](#) [► 116].

4.8.1.6 Calling the TcCOM from the PLC

Please go to section [Applying the TcCOM Wrapper FB](#) [► 209].

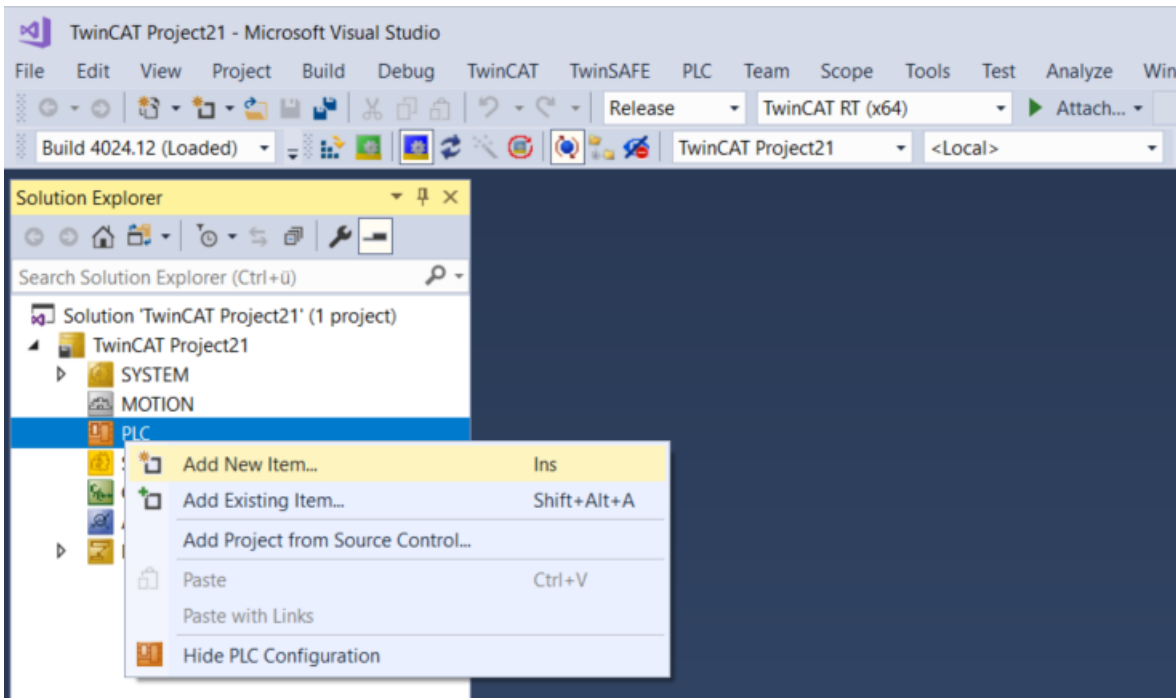
4.8.2 Working with the PLC library

In addition to the TcCOM object, a PLC library can also be created. This PLC library contains **two** different **types** of function blocks:

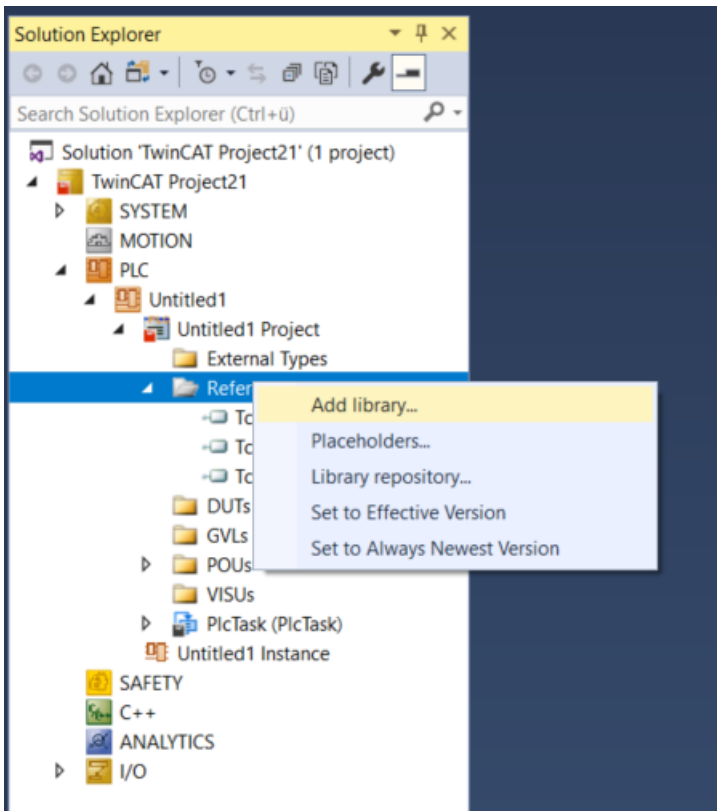
- The function block *FB_<modelname>* in the folder *POU* contains the full functionality of the Simulink® model, i.e. the source code is directly anchored in the FB. This FB is called **PLC-FB** in the following.
- The subfolder *POU/TcCOM Wrapper* contains FBs that are wrappers for a TcCOM object, i.e. the functionality is not directly in the FB but is outsourced to an instance of a TcCOM. The wrappers are called **TcCOM-Wrapper-FB** in the following.

Brief overview

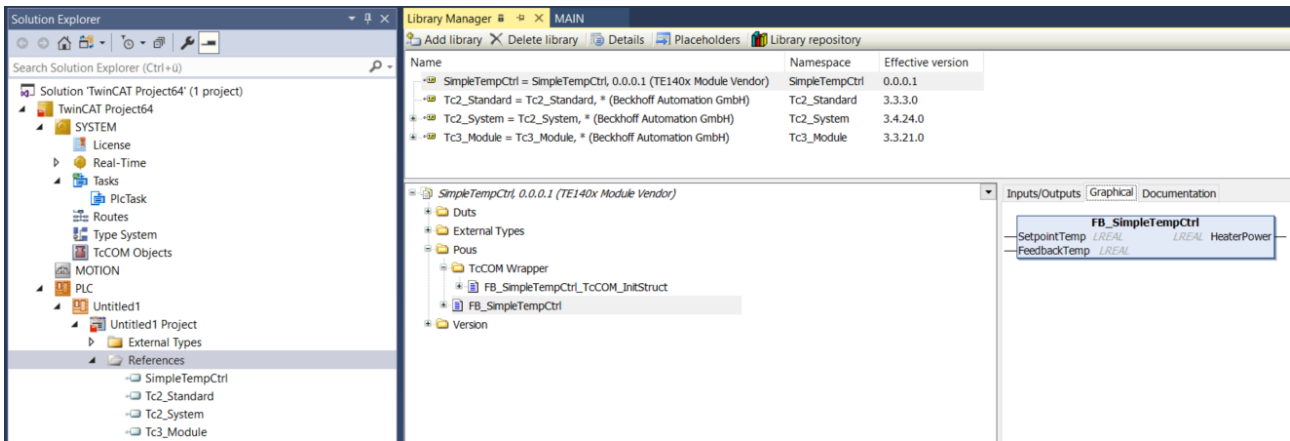
- Create PLC project:



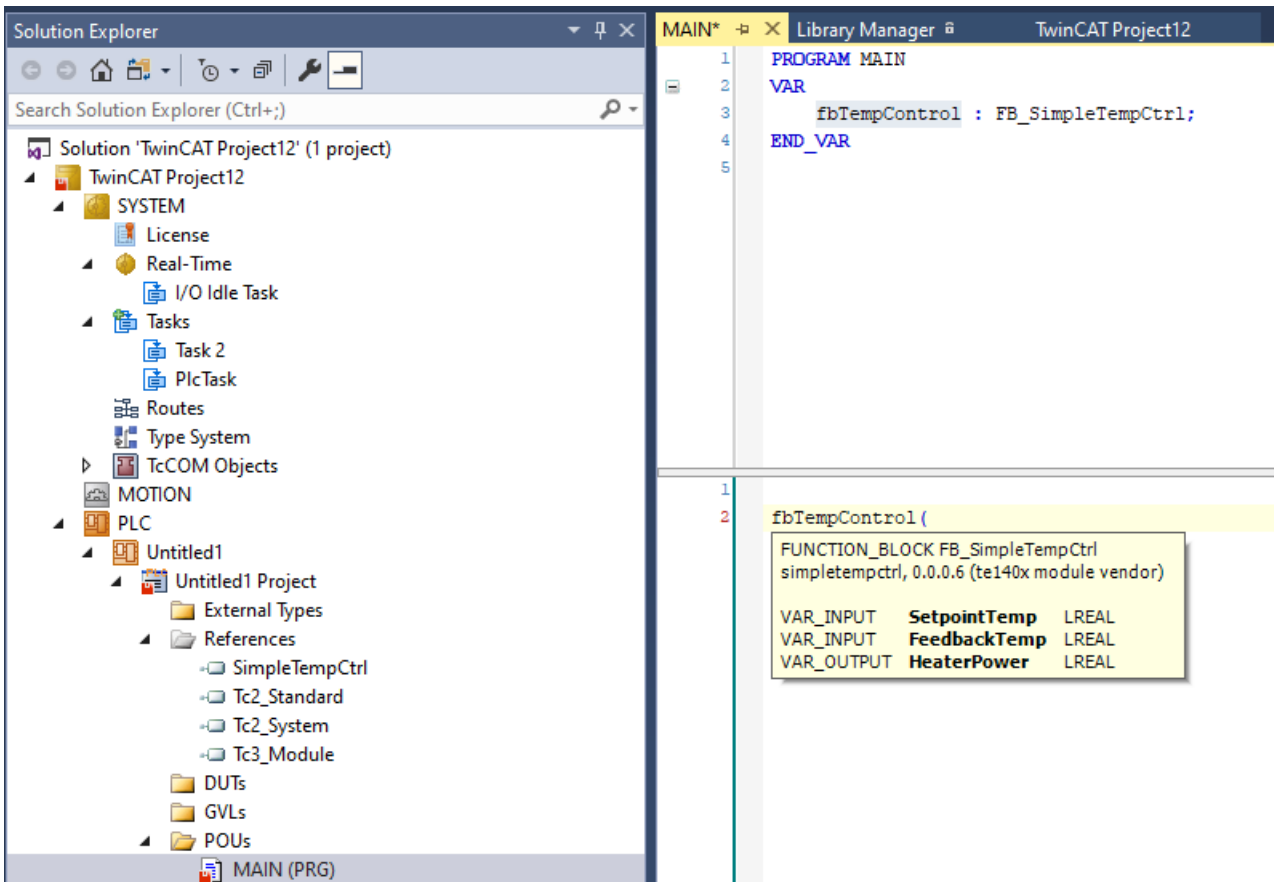
- Load PLC library:



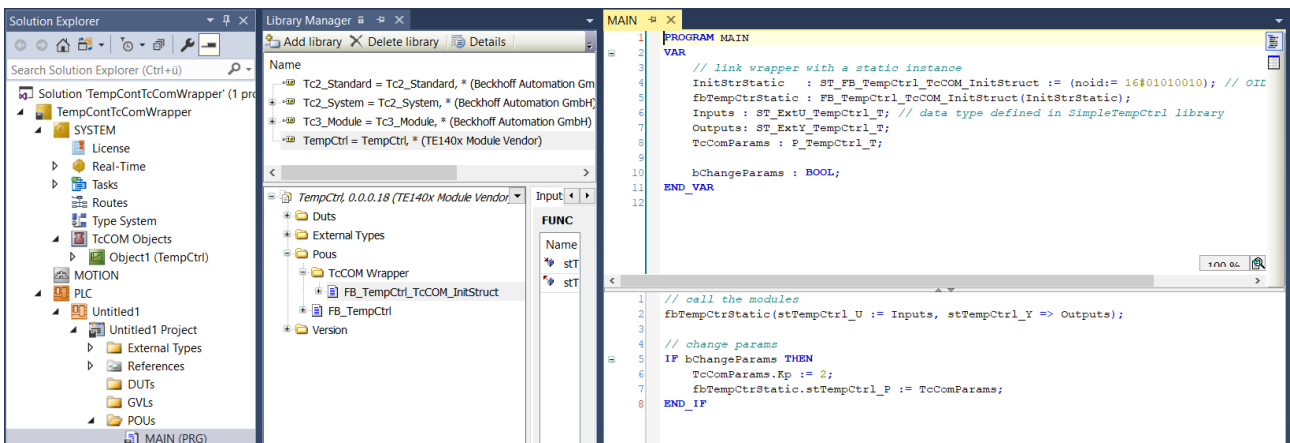
- View content of the PLC library:



- Instantiate function block `FB_<modelName>` (PLC-FB [▶ 213]) and use in PLC code:



- Alternatively: use `TcCOM Wrapper FB [▶ 209]` and use it in the PLC code:



4.8.2.1 Create and install PLC library

Configure the content of the PLC library

As described at [Working with the PLC library \[► 203\]](#), the PLC library can contain different function blocks.

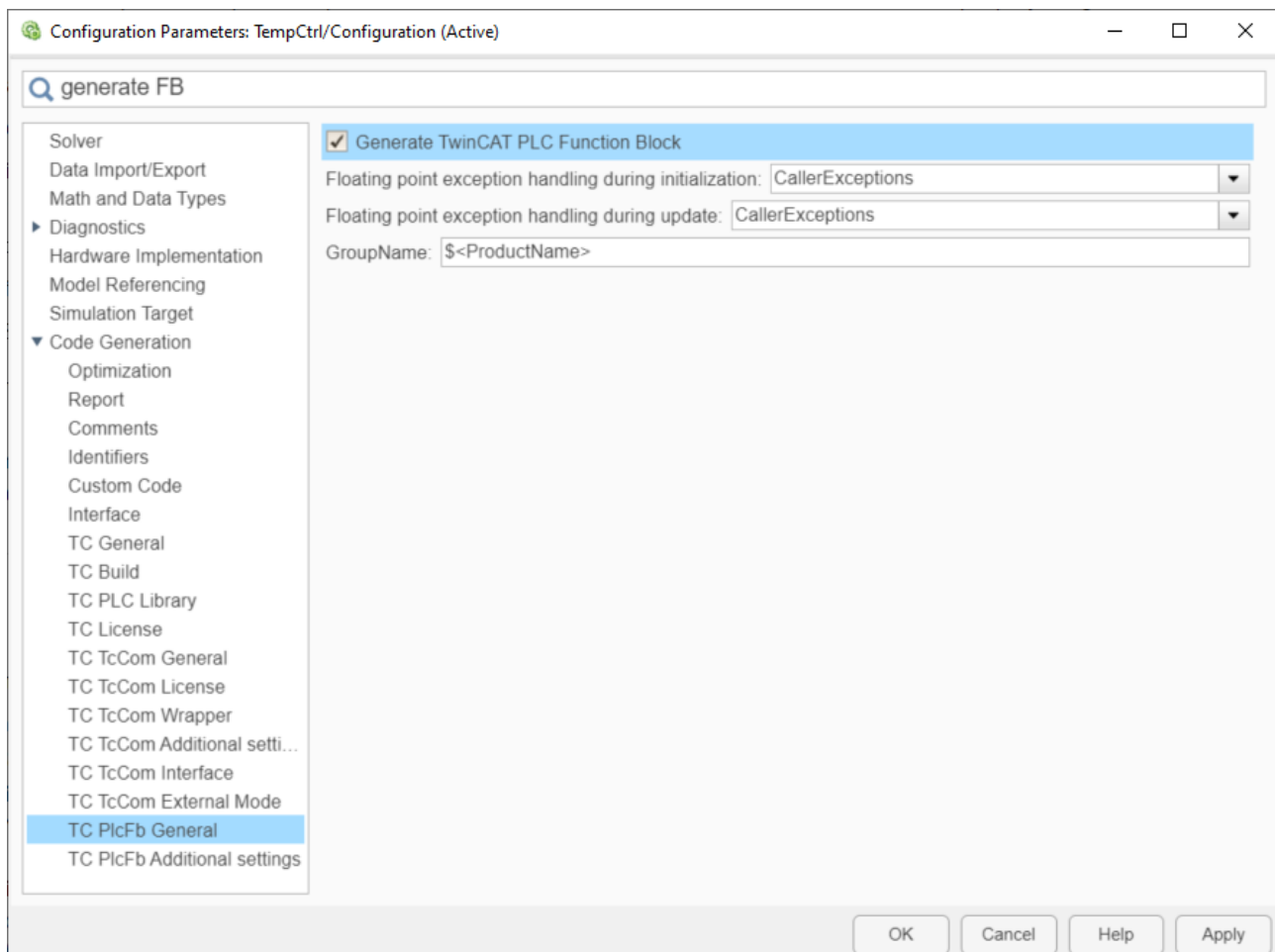
- The function block *FB_<modelName>* in the folder *POU* contains the full functionality of the Simulink® model, i.e. the source code is directly anchored in the FB. This FB is called **PLC-FB** in the following.
- The subfolder *POU/TcCOM Wrapper* contains FBs that are wrappers for a TcCOM object, i.e. the functionality is not directly in the FB but is outsourced to an instance of a TcCOM. The wrappers are called **TcCOM-Wrapper-FB** in the following.

In the following, you will learn how to configure the two function blocks and how to install the PLC library created.

4.8.2.1.1 Create and configure the PLC-FB

Navigate to **Configuration Parameters > Code Generation > TC PlcFb General**.

Check the checkbox *Generate TwinCAT PLC Function Block* here. In the standard configuration, the checkbox is checked.

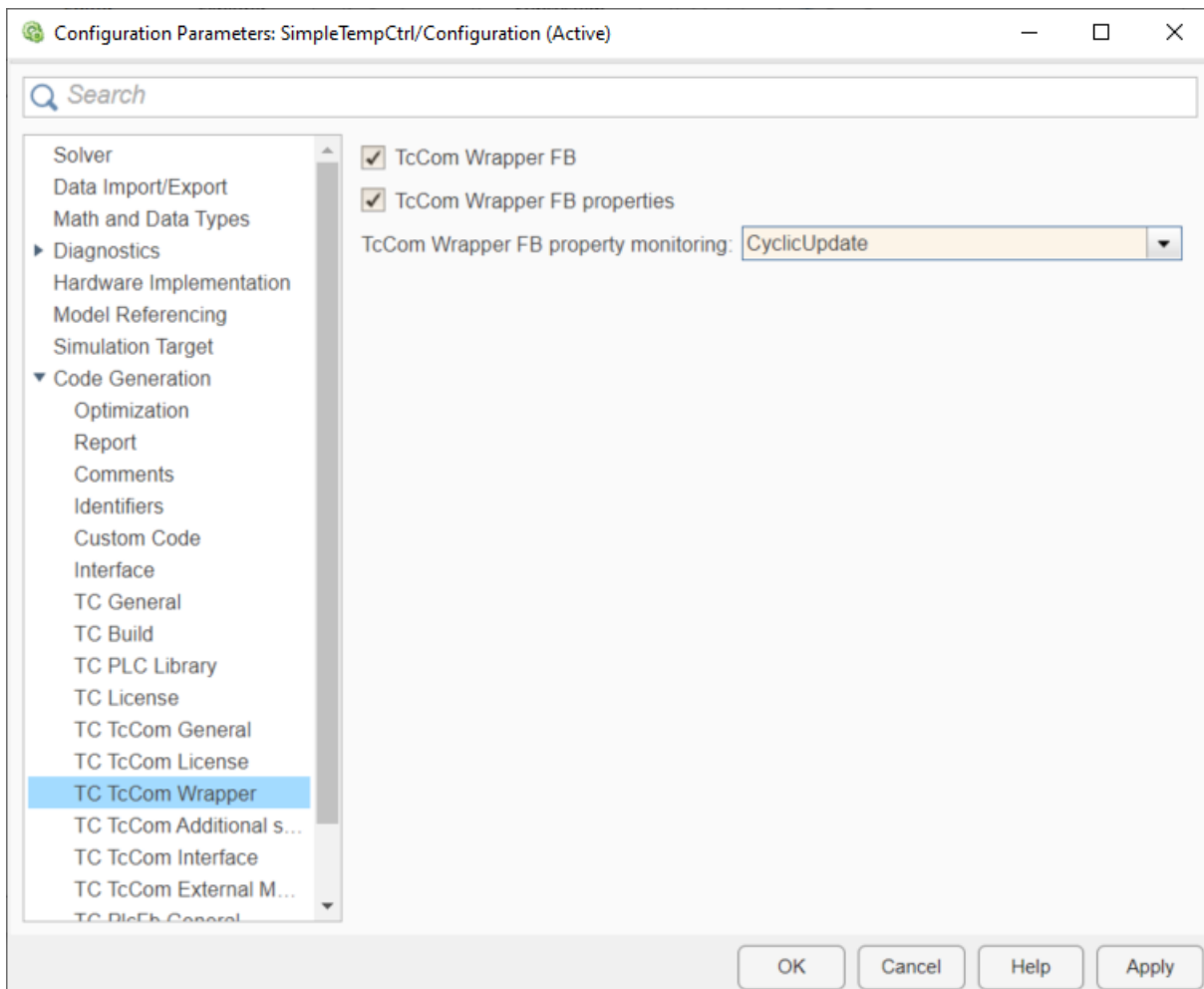


The further settings concern the handling of floating point exceptions for the PLC-FB. These must be made separately to the settings for the TcCOM object, see [Exception handling \[► 220\]](#).

4.8.2.1.2 Create and configure the TcCOM-Wrapper-FB

Navigate to **Configuration Parameters > Code Generation > TC TcCom Wrapper**.

Check the checkbox *TcCom Wrapper FB* here. The checkbox is unchecked in the standard configuration.



You can use the checkbox *TcCom Wrapper FB properties* to configure whether the module parameters on the FB are to be created as properties. See [Configuration of data access to data of a TcCOM object \[► 143\]](#), especially *Parameter: Initial values*.

You can learn how to apply the generated wrapper here: [Applying the TcCOM Wrapper FB \[► 209\]](#).

Sample

By setting the *Parameter: Initial Values* under Tc TcCom Interfaces, the model parameters are created as module parameters (switched on by default). Now create the "TcCom Wrapper FB" with the option "TcCom Wrapper FB properties". Set the property monitoring to "CyclicUpdate" to see the value change of the property directly in the online view.

Then you can access the module parameters as follows, for example:

```
PROGRAM MAIN
VAR
    // dynamic instance: create TcCOM from PLC
    InitStrDyn : ST_FB TempCtrl_TcCOM_InitStruct_InitStruct := (
        TaskOid:= 16#02010030,           // take TaskOID of PlcTask
        eModuleCaller:= ModuleCaller.Module ); // set module caller to "call by module"
    fbTempCtrDyn : FB_TempCtrl_TcCOM_InitStruct(InitStrDyn);

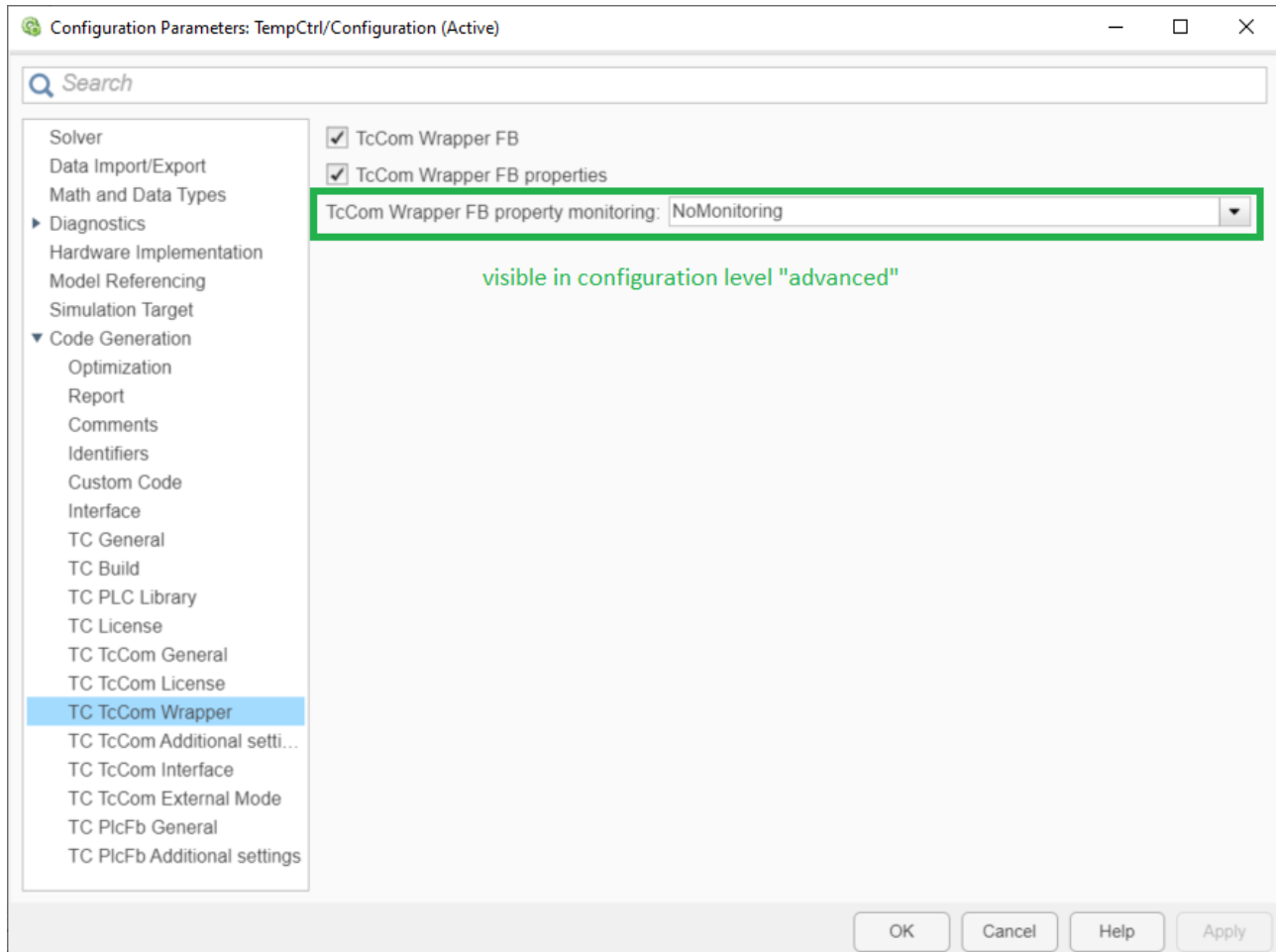
    Outputs      : ExtY_TempCtrl_T; // input
    Inputs       : ExtU_TempCtrl_T; // output
    Parameters   : P_TempCtrl_T;   // parameter

    bChange: BOOL;
END_VAR

fbTempCtrDyn(TempCtrl_U := Inputs, TempCtrl_Y => Outputs);

IF bChange THEN
```

```
Parameters.Kp := 10;
fbTempCtrlDyn.TempCtrl_P := Parameters;
END_IF
```



In the "advanced" configuration level, the monitoring attribute of the properties can also be specified. In the default case, "No Monitoring" is set, i.e. no attribute is set.

Setting in Simulink®	Attribute on property
ExecutionUpdate	{attribute 'monitoring' := 'variable'}
CyclicUpdate	{attribute 'monitoring' := 'call'}

Monitoring attributes influence the visibility of the attribute values in the online view, i.e. when you have logged into the PLC and want to monitor the current values of the properties on the FB.

- No Monitoring: the values are not visible in the online view.
- Cyclic Update: the values of the properties are updated and displayed cyclically.
In the logged-in state in the PLC additional code is executed.
- Execution Update: the values of the properties are only updated in the online view if getter/setter methods for the properties are called in the execution code. **This quickly leads to irritation and is only relevant in rare cases.**

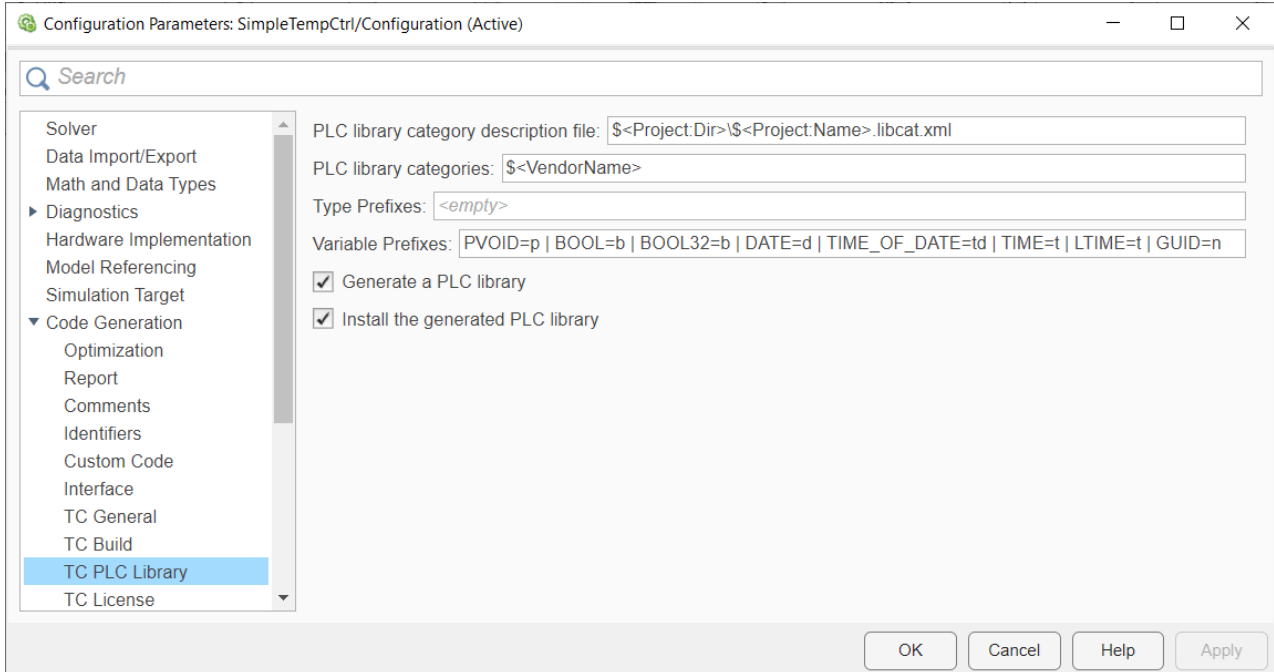
i No TcCOM Wrapper for Online Change-capable modules

If the TcCOM is Online Change-capable, no TcCOM-Wrapper-FB is generated, since the version of the PLC library and the version of the TcCOM object must always match, which cannot be guaranteed for the Online Change of the TcCOM.

4.8.2.1.3 Configuring and installing the PLC library

Configuring the type and variable prefixes

You can configure your individual type and variable prefixes to be used in the created PLC library at **Code Generation > TC PLC Library**. By default, the variable prefixes are generated according to this convention (see [Identifiers for variables and instances](#)).



Enter the desired prefixes as *pipe-separated list*.

Installing the PLC library

As described above, you can configure which function blocks (PLC-FB [[▶ 206](#)] and/or TcCOM-Wrapper-FB [[▶ 206](#)]) should be included in your PLC library. In order to use the created function blocks in the PLC, the corresponding PLC library must be installed on your TwinCAT engineering system.

Situation 1:

- ✓ You use the Target for Simulink® on the same PC on which you want to program your PLC.
- 1. Create a PLC library and install it directly on your local engineering system from Simulink®.
- 2. To do this, select the appropriate checkboxes at **TC PLC Library**:
 - ⇒ After a successful build, the new version of the PLC library is directly available in the local TwinCAT XAE.

Situation 2:

- ✓ You want to use the PLC library on any TwinCAT XAE systems.
- 1. Create a [TMX archive](#) [[▶ 135](#)].
- 2. Copy the [TMX archive](#) [[▶ 135](#)] to a TwinCAT engineering system of your choice.
- 3. Install the PLC library when unpacking the [TMX archive](#) [[▶ 135](#)].
 - ⇒ The library contained in the archive is available in TwinCAT XAE.

4.8.2.2 Applying the TcCOM Wrapper FB

There are two ways to call a TcCOM object from the PLC:

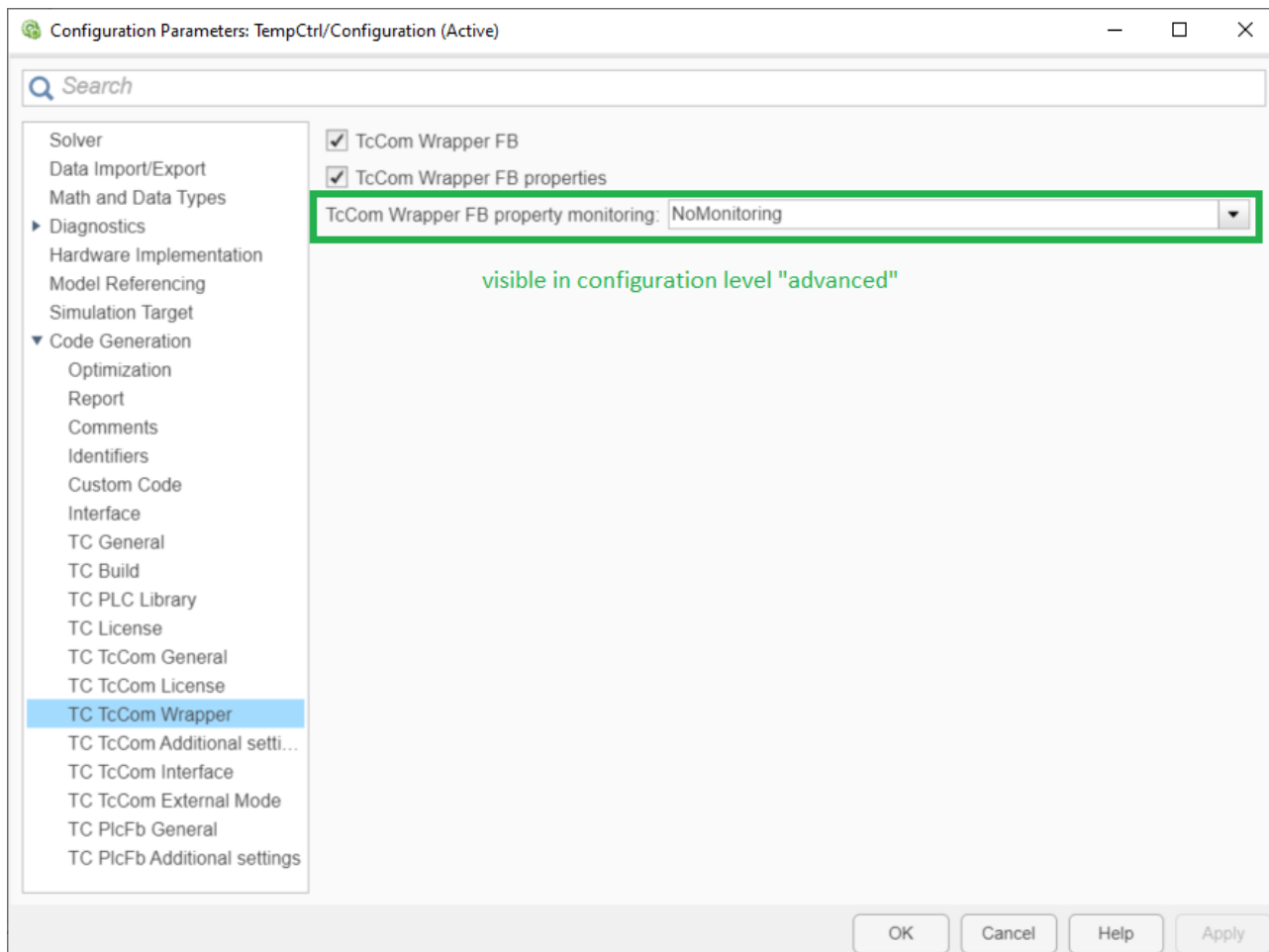
1. Referencing a static object instance

2. Dynamic instantiation of an object from the PLC

⇒ For both ways the TcCOM Wrapper function block from the generated PLC library is used.

Configuration of the TcCOM Wrapper function block in Simulink®

Navigate to **Configuration Parameters > Code Generation > TC TcCom Wrapper**. Check the **TcCom Wrapper FB** checkbox here. The checkbox is unchecked in the standard configuration. You can use the **TcCom Wrapper FB properties** checkbox to configure whether the model parameters on the function block are to be created as properties.



In the "advanced" configuration level, the monitoring attribute of the properties can also be specified. In the default case, "No Monitoring" is set, i.e. no attribute is set.

Setting in Simulink®	Attribute on property
ExecutionUpdate	{attribute 'monitoring' := 'variable'}
CyclicUpdate	{attribute 'monitoring' := 'call'}

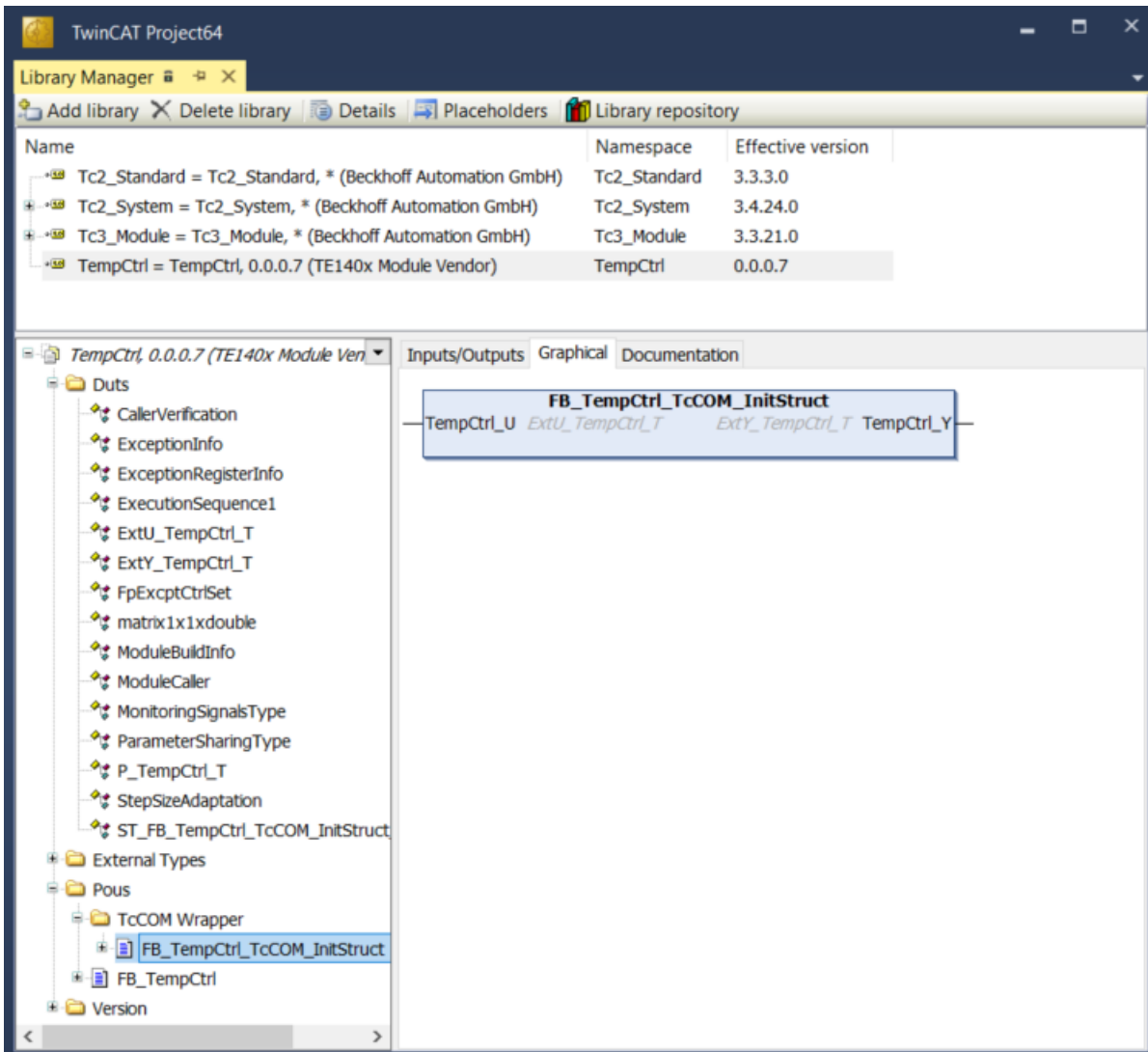
Monitoring attributes influence the visibility of the attribute values in the online view, i.e. when you have logged into the PLC and want to monitor the current values of the properties on the FB.

- No Monitoring: the values are not visible in the online view.
- Cyclic Update: the values of the properties are updated and displayed cyclically.
In the logged-in state in the PLC additional code is executed.
- Execution Update: the values of the properties are only updated in the online view if getter/setter methods for the properties are called in the execution code. **This quickly leads to irritation and is only relevant in rare cases.**

See also [Create and install PLC library \[► 206\]](#) or [Create the TcCOM-Wrapper-FB](#) for further details.

Create instance of the TcCOM Wrapper function block

1. Create a PLC project.
2. Add the desired library under **References**.



⇒ Under **Pous/TcCOM Wrapper** you get a function block that you can instantiate in the PLC. In addition, necessary data types are created in the *Duts* folder.

Version 1: referencing a static module instance

The function block can be used to access module instances previously created in the XAE, e.g. under **System > TcCOM Objects**. For this static case, the object ID of the corresponding module instance must be transferred during declaration of the function block instance. See red area in graphic below.



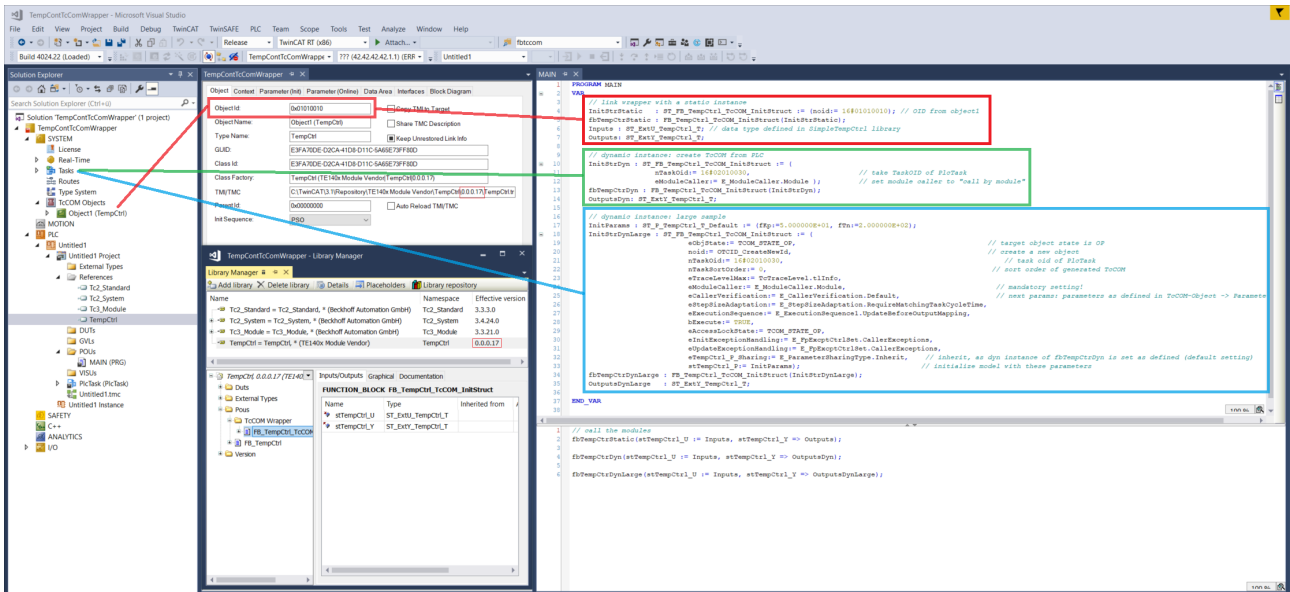
- The instance of the TcCOM object and the calling PLC must run in the same task.
- On the instance of the TcCOM object, make sure that under Parameter (Init) the entry *ModuleCaller* is set to *Module* and not to *CyclicTask*.
- In this case, the required memory for the TcCOM is obtained from the *non paged pool* of the system.

Version 2: dynamic instantiation and referencing from the PLC

The function block can also be used in such a way that a TcCOM object is generated from the PLC and linked to the wrapper. In the graphic below, this is the green area as a minimum example and the blue area with extended parameterization.

1

- The TaskOid of the PLC task must be used to specify the real-time task in which the wrapper is called.
- The ModuleCaller must also be set to *Module* here (via the Init structure).
- In this case, the required memory for the TcCOM is obtained from the router memory.



1

Graphic in the web browser view

Right-click on the image and select "open image in new tab" to better read the image.

The source code for the graph shown above is available in MATLAB® via the Command Window:

```
TwinCAT.ModuleGenerator.Samples.Start('TcComWrapper TemperatureController')
```

Working with the properties of the TcCOM Wrapper FB

Properties on the FB provide an easy way to interact with module parameters of a TcCOM, see also [Best Practice: access to TcCOM data](#) [▶ 147].

Sample

By setting the *Parameter: Initial Values* under Tc TcCom Interfaces, the model parameters are created as module parameters (switched on by default). Now create the "TcCom Wrapper FB" with the option "TcCom Wrapper FB properties". Set the property monitoring to "CyclicUpdate" to see the value change of the property directly in the online view.

Then you can access the module parameters as follows, for example:

```
PROGRAM MAIN
VAR
    // dynamic instance: create TcCOM from PLC
    InitStrDyn : ST_FB TempCtrl_TcCOM_InitStruct_InitStruct := (
        TaskOid:= 16#02010030, // take TaskOID of PlcTask
        eModuleCaller:= ModuleCaller.Module ); // set module caller to "call by module"
    fbTempCtrDyn : FB_TempCtrl_TcCOM_InitStruct(InitStrDyn);

    Outputs : ExtY_TempCtrl_T; // input
    Inputs : ExtU_TempCtrl_T; // output
    Parameters : P_TempCtrl_T; // parameter

    bChange: BOOL;
END_VAR

fbTempCtrDyn(TempCtrl_U := Inputs, TempCtrl_Y => Outputs);

IF bChange THEN
```

```
Parameters.Kp := 10;
fbTempCtrDyn.TempCtrl_P := Parameters;
END_IF
```

Working with the ADI Interface

⚠ WARNING

Unrestricted read and write access
 You get a pointer to the memory area of a DataArea via the ITc_ADI interface. Accordingly, you can read and write there without any restrictions.

The following is a sample of how to access the DataArea of the BlockIO in read-only mode:

```
stInitTemp : ST_Funktionsblock_SimpleTempCtrl_TcCOM_InitStruct := (nOid := 16#01010010);
FunktionsblockTempCtr : Funktionsblock_SimpleTempCtrl_TcCOM_InitStruct(stInitTemp);
stTempCtr_BlockIO : ST_B_SimpleTempCtrl_T;
pStTempCtr_BlockIO : POINTER TO ST_B_SimpleTempCtrl_T;
hr : HRESULT;

(* read data are via ADI Interface *)
// get a pointer to Data Area
hr := FunktionsblockTempCtr.ipADI.GetImagePtr(size := SIZEOF(stTempCtr_BlockIO), offs := 0, adi_x := 2, ppData := ADR(pStTempCtr_BlockIO));
IF hr = 0 THEN
    // copy data to a local variable
    MEMCPY(ADR(stTempCtr_BlockIO), pStTempCtr_BlockIO, SIZEOF(stTempCtr_BlockIO));
    // always release the pointer!
    FunktionsblockTempCtr.ipADI.ReleaseImagePtr(pData := pStTempCtr_BlockIO);
END_IF;
```

Area No	Name	Type	Size	CS	CD / Elements
+ 0 (0)	SimpleTempCtrl_U	InputDst	16	<input checked="" type="checkbox"/>	<input type="checkbox"/> 2 Symbols
+ 1 (0)	SimpleTempCtrl_Y	OutputSrc	8	<input checked="" type="checkbox"/>	<input type="checkbox"/> 1 Symbols
- 2 (0)	SimpleTempCtrl_B	Internal	64	<input checked="" type="checkbox"/>	<input type="checkbox"/> 8 Symbols
	e	LREAL	8.0 (Offs: 0.0)	<input checked="" type="checkbox"/>	0x82000000
	P	LREAL	8.0 (Offs: 8.0)	<input checked="" type="checkbox"/>	0x82000008
	Integrator	LREAL	8.0 (Offs: 16.0)	<input checked="" type="checkbox"/>	0x82000010
	Sum	LREAL	8.0 (Offs: 24.0)	<input checked="" type="checkbox"/>	0x82000018
	Saturation	LREAL	8.0 (Offs: 32.0)	<input checked="" type="checkbox"/>	0x82000020
	Switch	LREAL	8.0 (Offs: 40.0)	<input checked="" type="checkbox"/>	0x82000028
	I	LREAL	8.0 (Offs: 48.0)	<input checked="" type="checkbox"/>	0x82000030
	ARW	BOOL	1.0 (Offs: 56.0)	<input checked="" type="checkbox"/>	0x82000038
+ 3 (0)	SimpleTempCtrl_X	Internal	8	<input checked="" type="checkbox"/>	<input type="checkbox"/> 1 Symbols
+ 4 (0)	ExecutionInfo	OutputSrc	240	<input type="checkbox"/>	<input type="checkbox"/> 4 Symbols

Among other things, `adi_x` and `offs` are passed to the `GetImagePtr` method. These determine the DataArea itself, in this case DataArea number 2 (`SimpleTempCtrl_B`), and the data area in the area to be read/written, in this case without offset and the total size of the DataArea (i.e. the entire DataArea).

When writing, the source and destination must be swapped accordingly at `MEMCPY` in the above sample.

For further instructions on how to interact with the TcCOM, refer to [Best Practice: access to TcCOM data \[► 147\]](#).

4.8.2.3 Using the PLC function block (PLC-FB)

The application of the PLC function block (PLC-FB, `FB_<modelname>`) in the PLC library is focused on its simple application. This is also accompanied by a few restrictions compared to the TcCOM (or the PLC Wrapper FB).

Application

Create one or more instances of the PLC-FB from the created PLC library. When writing the source code, the IntelliSense is available so that you can conveniently see the expected inputs and outputs.

```

MAIN* X
1 PROGRAM MAIN
2 VAR
3   fbTemp : FB_SimpleTempCtrl;
4 END_VAR
5
1 fbTemp (
  FUNCTION_BLOCK FB_SimpleTempCtrl
  simpletempctrl, 1.3.1301.267 (te140x module vendor)
  VAR_INPUT   fSetpointTemp  LREAL
  VAR_INPUT   fFeedbackTemp  LREAL
  VAR_OUTPUT  fHeaterPower    LREAL

```

If bus objects have been used as inputs or outputs in Simulink®, these data types are automatically defined as structures in the PLC library.

Restrictions

The PLC-FB does not allow access to the model parameters via properties at the function block. However, there are two ways to change the model parameters:

- About the [External mode](#) [▶ 217]
- When using [reusable code](#) [▶ 191], a TcCOM instance can define the parameters. All instances of the PLC-FB are automatically set to "inherit" and adopt the parameters of the default TcCOM instance.

Also, the PLC-FB does not contain a [block diagram](#) [▶ 193] in TwinCAT XAE. Debugging of the function block is done via the TwinCAT C++ Debugger, as described [here](#) [▶ 214] or via External mode.

4.8.2.3.1 Online Change of the PLC library

While TwinCAT is in run mode, you can exchange the PLC library version in TwinCAT XAE and load it into the running application via Online Change. This means that all function blocks in a PLC library can be updated without a TwinCAT restart.

Step-by-step procedure:

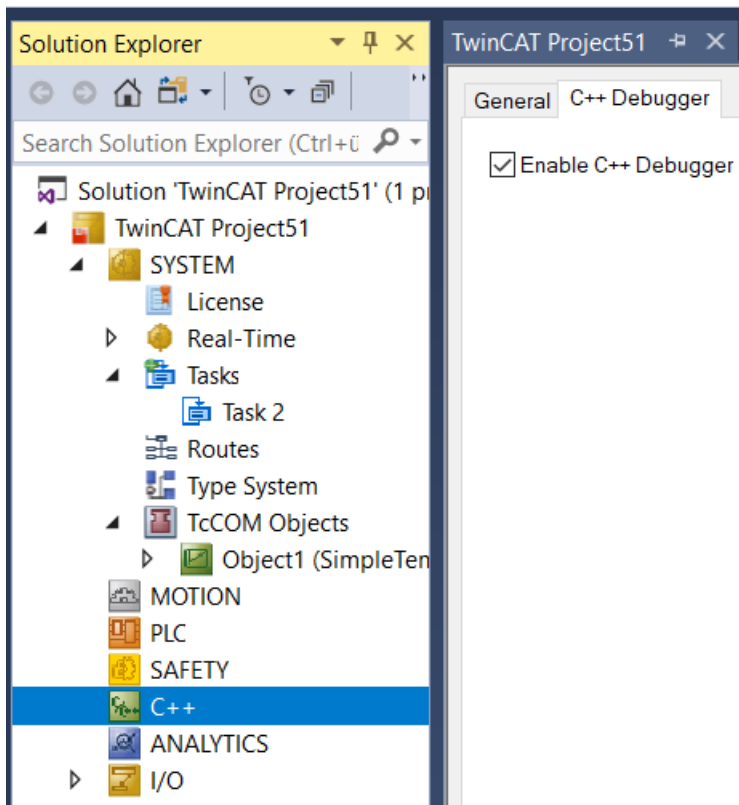
1. Create a first PLC library version with the TwinCAT Target for Simulink®.
2. Include this PLC library version in a PLC project.
3. Activate your TwinCAT configuration with the first PLC library version (e.g. version 0.0.0.1).
4. Adapt your Simulink® model and create a PLC library version (0.0.0.2) from it.
5. Select the newly created PLC library version in the PLC at **References** (you may have to install the new library on the XAE system).
6. Select **Build > Build Solution** to rebuild the project.
7. Select **Login > Login with online change** (more information in the [PLC documentation](#)).

4.8.3 Debugging

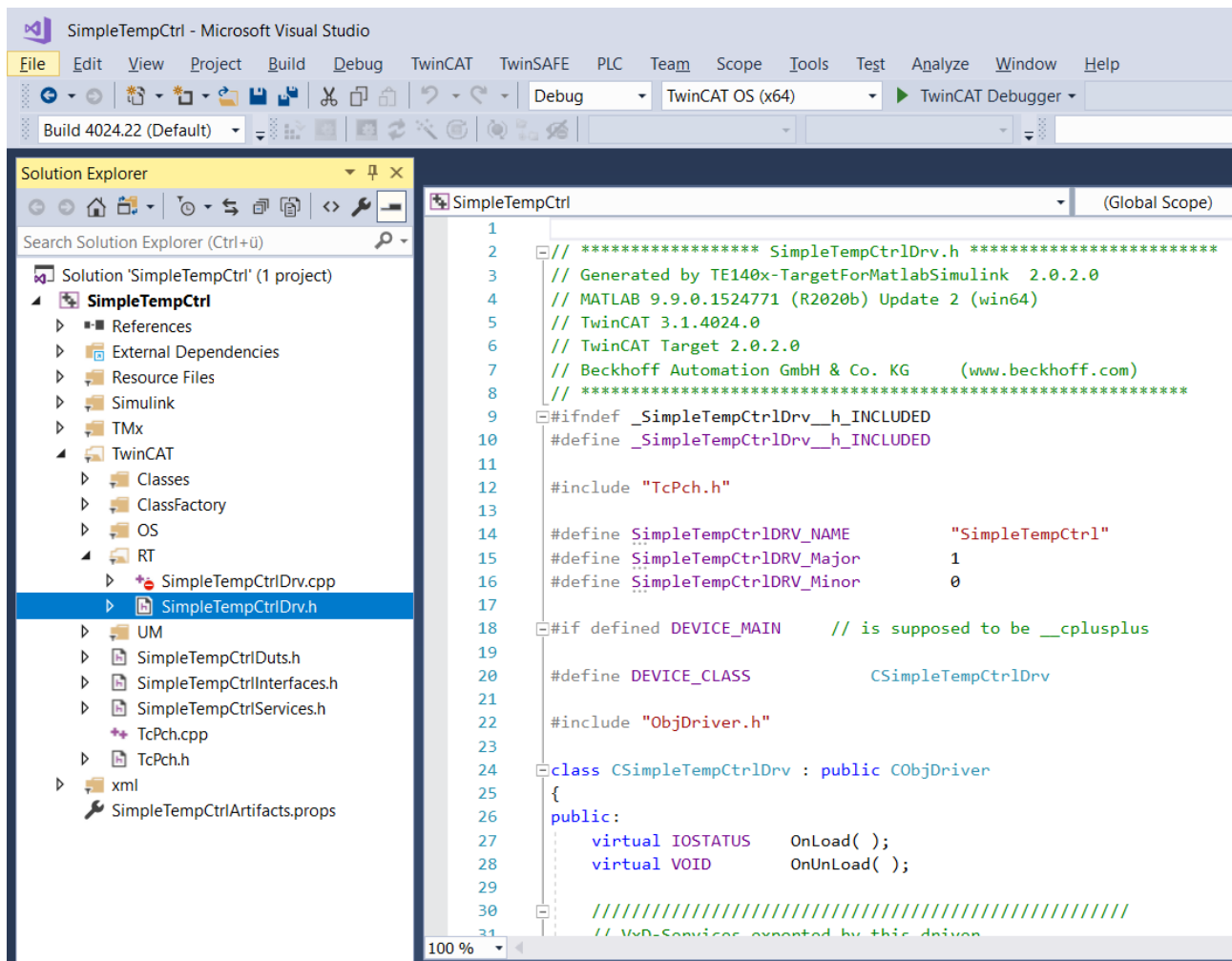
In addition to debugging via the [External Mode](#) [▶ 217] and via the [Block Diagram](#) in TwinCAT XAE [▶ 197], you can also use the C++ project created for debugging in the classic way.

Step-by-step procedure:

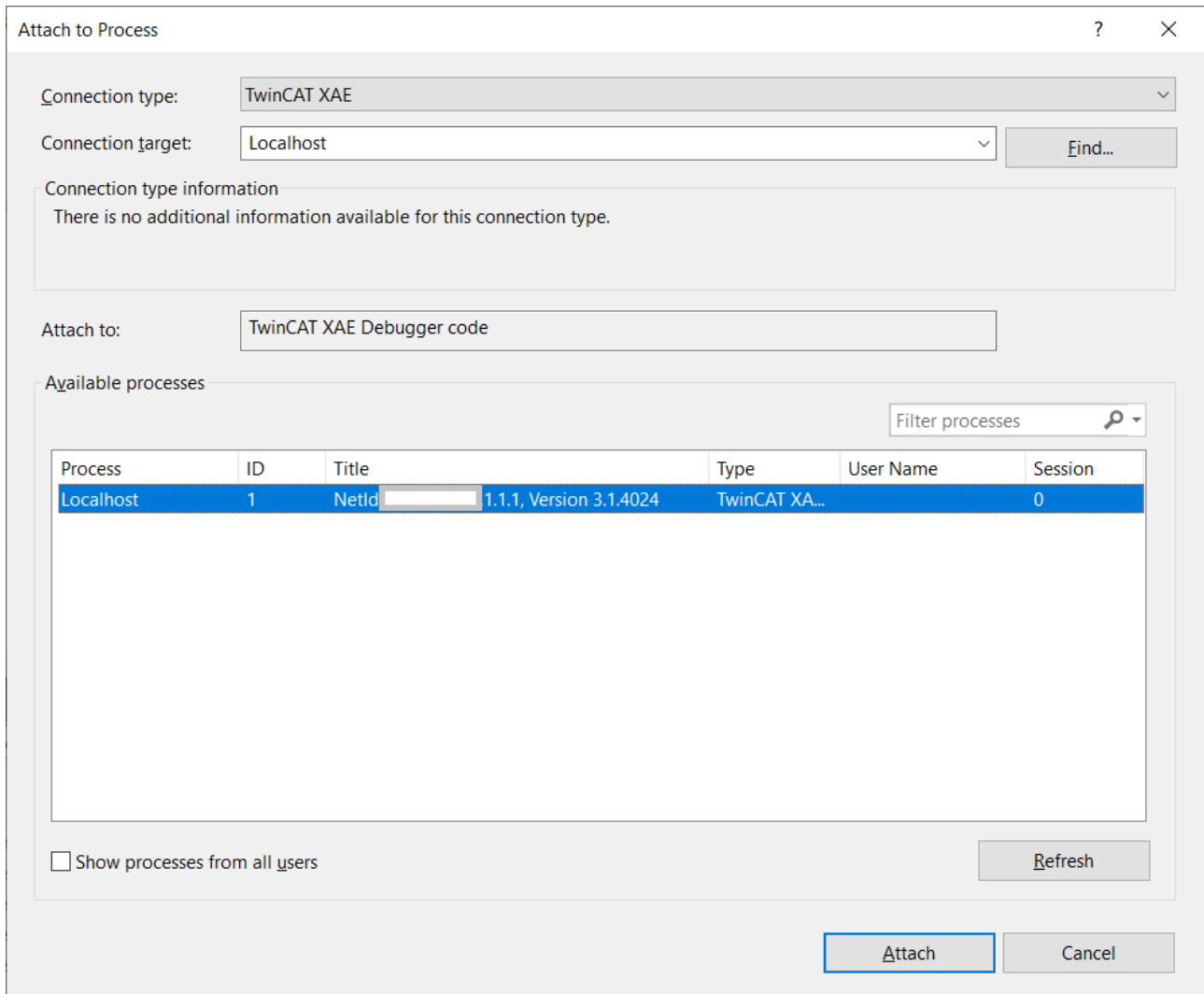
1. Make sure that your TwinCAT application has been activated with the C++ debugger enabled.



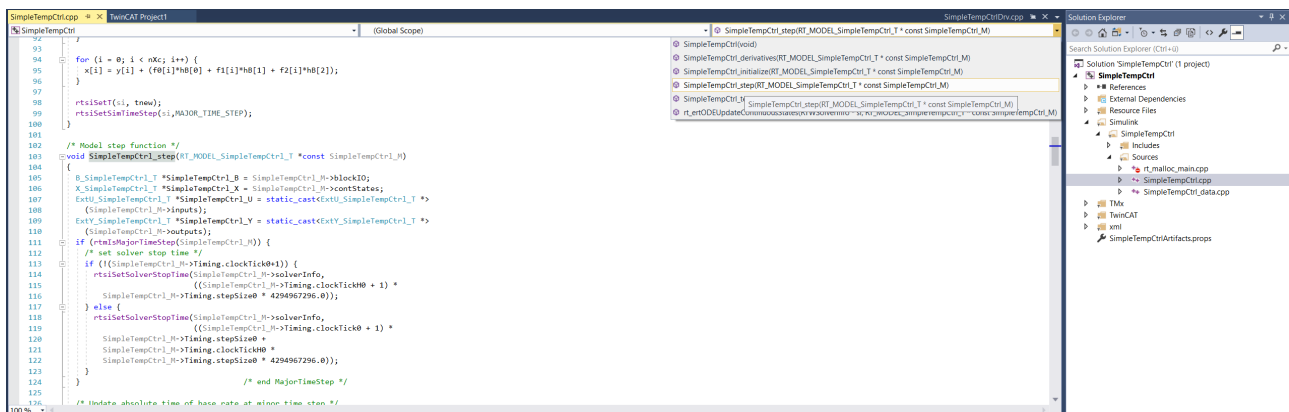
2. Open the C++ project created during code generation that belongs to the module you want to debug. The project can be found in the folder `<SimulinkModelName>_tcgrt`, which is created in the current MATLAB® path when you start the code generation process.
3. In this folder, search for the file `<SimulinkModelName>.vcxproj`. You can open the `<SimulinkModelName>.vcxproj` in Visual Studio alone or also add the `vcxproj` file in your TwinCAT Solution under C++ with "Add existing Item".



4. Select **Debug > Attach to Process** in the menu bar and select "TwinCAT XAE" as Connection Type and your desired target system under Connection target. Then select **Attach**.



- Set breakpoints in your C++ code and step through your code as usual. Tip: when executing the code, the step function is used, which you can find in the folder **Simulink > Sources > <SimulinkModelName>.cpp**.



4.8.4 Connecting to External mode

You can connect from your Simulink® environment to a running TcCOM object or an instance of the PLC function block in the TwinCAT XAR via External mode.

● Restriction on code interface packaging

i The code interface packaging defines the behavior for multiple instances of a created class in TwinCAT [▶ 191]. If the External mode is to be used, the setting C++ Class is not allowed.

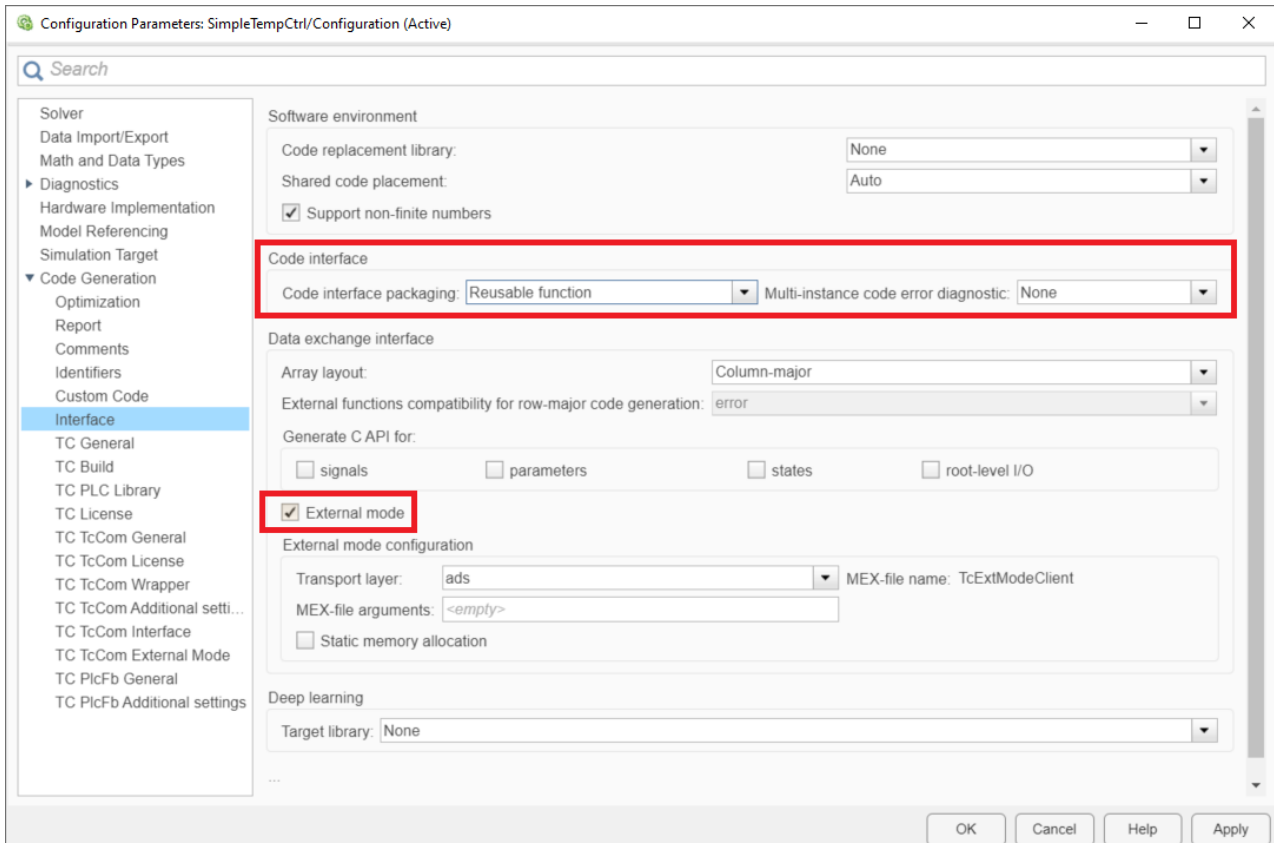
i Simulation time in Simulink® set to "inf"

Set the Simulink® simulation time to "inf". For operation in TwinCAT, it makes no sense to stop the execution of the module after a defined time.

✓ Code generation settings in Simulink®

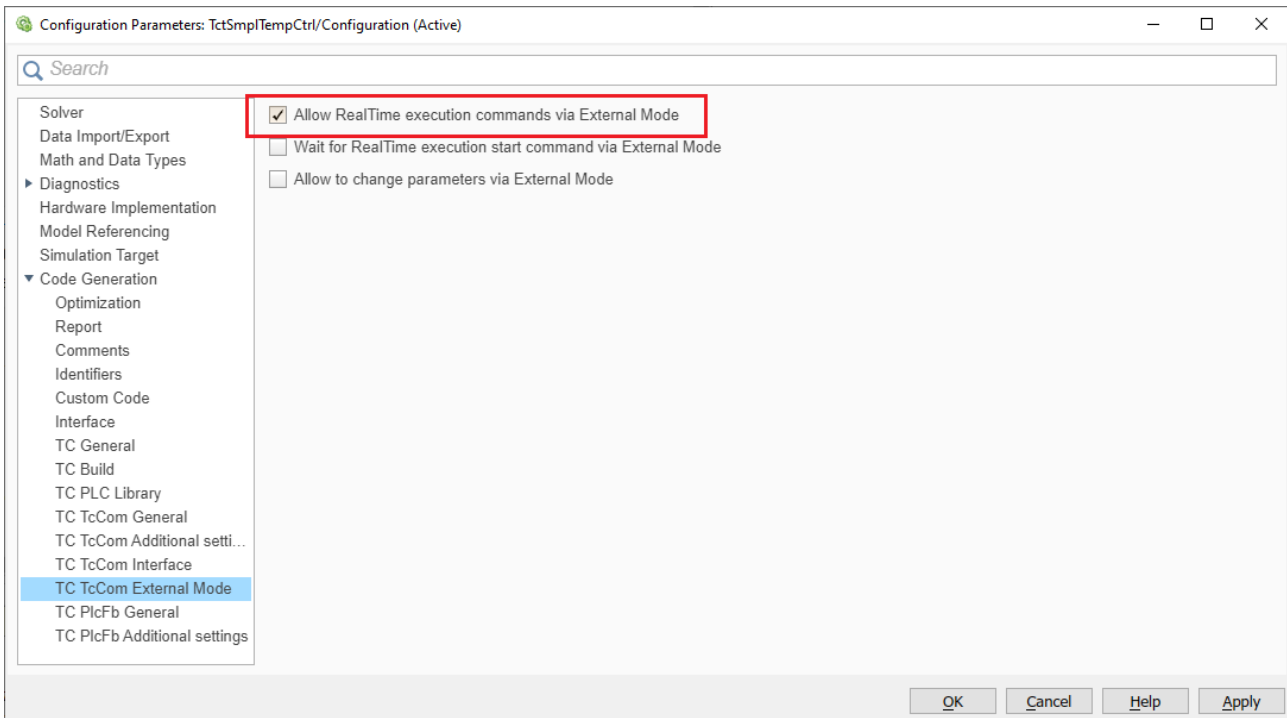
1. Under **Code Generation > Interface**, set the **External mode** parameter.

- Note the following for **Code interface packaging**:
 - Nonreusable function: allowed
 - Reusable function: allowed, set multi-instance code error diagnostic to **None**
 - C++ Class: not allowed

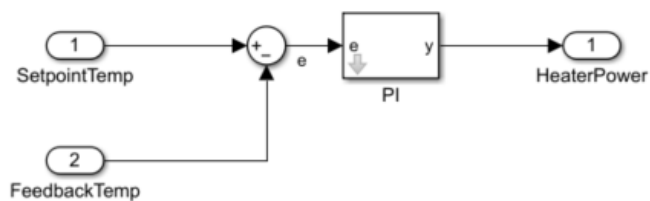
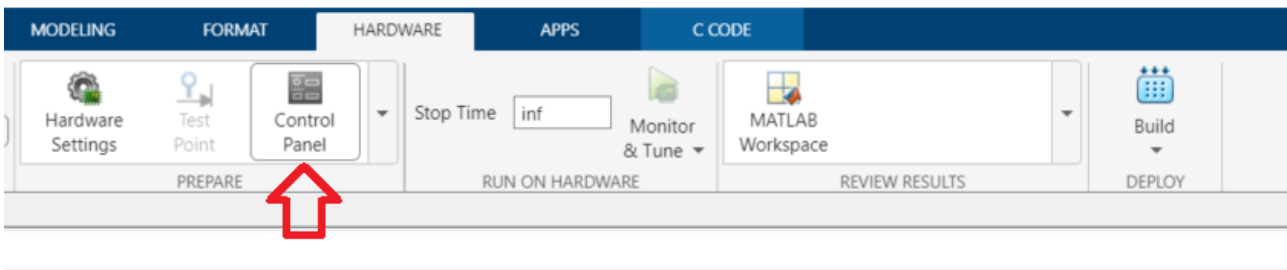


2. Define the permissions of the External mode (separately adjustable for TcCOM and PlcFb).

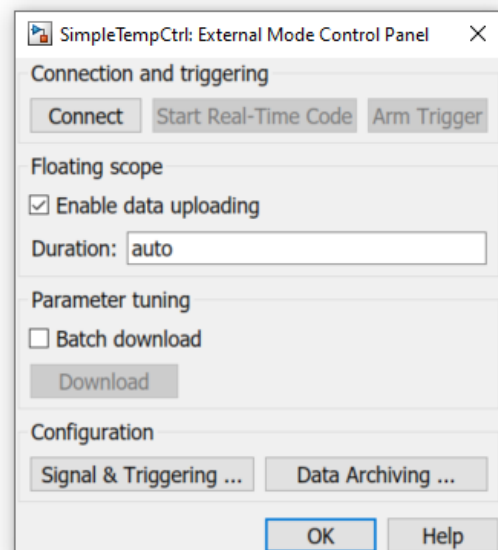
⇒ Shown in the following screenshot as an example for **TcCOM**.



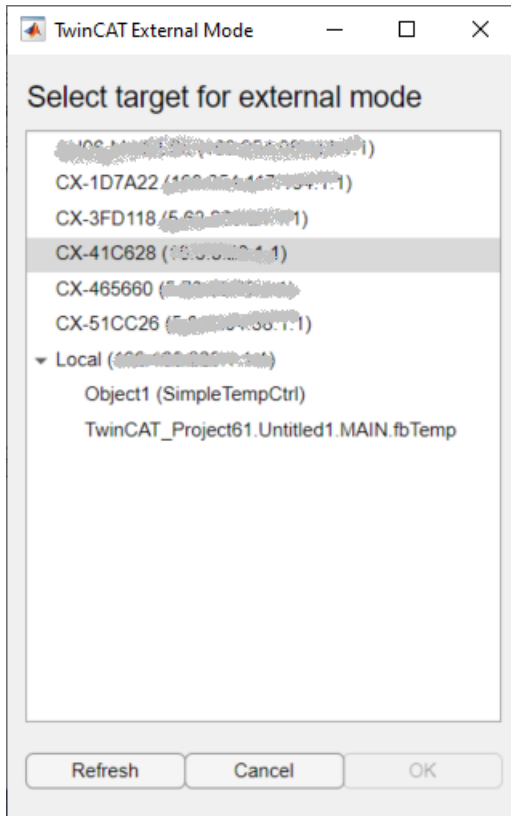
- ✓ Connect to a runtime object with the External mode
- 3. Open the External mode **control panel**.
- 4. Select **Connect**.



TwinCAT Target for MATLAB/Simulink
 Sample model "SimpleTempCtrl"
 Subjects:
 - Basics



5. Select the connected target and object instance.



⇒ After selecting **OK**, you will be connected to the object. The **Connect** button on the External mode control panel has changed to **Disconnect** and you can see the simulation time transferred from the target in Simulink®.

As can be seen in the above graphic for selecting the object in a target, the External mode is available for TcCOM instances as well as for PLC-FB instances.

i Bidirectional ADS route required

For the External mode, a bidirectional ADS route is necessary. Unidirectional routes cause a timeout in communication.

4.8.5 Exception handling

When processing the C++ code autogenerated from MATLAB® or Simulink® in TwinCAT, floating point exceptions can occur at runtime, for example if an unexpected value is passed into a function during programming. The handling of such exceptions is described below.

What is a floating point exception?

A floating point exception occurs when an arithmetically not exactly executable operation is instructed in the floating point unit of the CPU. IEEE 754 defines these cases: *inexact*, *underflow*, *overflow*, *divide-by-zero*, *invalid-operation*. If one of these cases occurs, a status flag is set, which indicates that the arithmetic operation cannot be executed exactly. It is further defined that each arithmetic operation must return a result – one that in the majority of cases leads to the possibility of ignoring the exception.

For example, a division by zero results in +inf or -inf. If a value is divided by inf in the further code, this results in zero, so that no consequential problems are to be expected. However, if inf is multiplied or other arithmetic operations are performed with inf, these are *invalid operations*, whose result is represented as a Not-a-Number (NaN).

How does the TwinCAT Runtime react in case of exceptions?

● TwinCAT C++ Debugger not active

I The following explanations only apply if the C++ debugger is **not** activated on the TwinCAT runtime system. When the C++ debugger is enabled, exceptions are caught by the debugger and can be handled, see [Debugging](#) [► 214].

Default behavior

Default setting in TwinCAT is that at "divide-by-zero" and "invalid-operation" the execution of the program is stopped and TwinCAT issues an error message.

Task setting: Floating Point Exceptions

This default setting can be changed on the level of each TwinCAT task. If the checkbox "Floating Point Exception" is unchecked, an exception **does not** lead to a TwinCAT stop and **no** error message is issued. This setting is then valid for all objects that are called by this task. As a consequence, care must be taken in the application that NaN and inf values are handled accordingly in the program code.

Check for NaN and Inf

If, for example, a NaN is passed on via mapping to a TwinCAT object that has activated floating point exceptions, an arithmetic operation with NaN naturally leads to an exception in this object and subsequently to a TwinCAT stop. Therefore, NaN or inf must be checked directly after mapping. In the PLC, corresponding functions are available in the [Tc2 Utilities](#) library, e.g. [LrealsNaN](#).

Try-Catch statement

Another way to handle exceptions is to embed them in a try-catch statement. In the PLC the instructions [TRY](#), [CATCH](#), [FINALLY](#), [ENDTRY](#) are available for this purpose. If floating point exceptions are enabled on the calling task and an exception occurs within the Try-Catch, it is caught in the Catch branch and can be handled. Accordingly, no variables are set to inf or NaN in this approach. However, it is also important to note that the code in the Try branch is run through only up to the point of the exception and then a jump is made to the Catch branch. In the application code, it should be noted that internal states in the Try branch may not be consistent.

Dump Files

From TwinCAT 3.1.4024.22 (XAR), dump files can be created at runtime in case of exceptions in the TcCOM object.

Specification of the behavior in the event of exceptions on object level

In addition to the option to influence behavior in the event of exceptions at task level, the behavior can also be specified at TwinCAT object level, i.e. the generated TcCOM or the generated PLC function block ([PLC-FB](#) [► 213]).

On the object level, a wealth of possibilities can be realized with the TwinCAT Target for Simulink®. Basically, however, all the options presented below are based on the above principles.

Definition of the object behavior in case of occurring exceptions

A total of 9 different settings are available.

- **CallerExceptions** (default): Exceptions are triggered as configured at the calling task.
- **ThrowExceptions**: Exceptions in the TwinCAT object are triggered in any case, regardless of how the task is configured.
 - An exception causes a TwinCAT error message and a TwinCAT stop
- **SuppressExceptions**: Exceptions are not triggered, regardless of how the task is configured.
 - An exception does not cause a TwinCAT stop.
 - Outputs or internal states can be NaN or inf.
- **LogExceptions**: Exceptions are triggered, but do not lead to a TwinCAT stop.

- An exception does not cause a TwinCAT stop.
- Outputs or internal states can be NaN or inf.
- The ExecutionInfo output is filled with information about an exception in the current cycle. If several exceptions occur in one cycle, only the first exception is displayed at the output. When the TwinCAT object is called again, the information is reset.
- **LogAndHold:** Exceptions are triggered. The execution of the TwinCAT object is stopped.
 - An exception does not cause a TwinCAT stop.
 - Outputs or internal states can be NaN or inf.
 - The ExecutionInfo output is filled with information about an exception in the current cycle. If several exceptions occur in one cycle, only the first exception is displayed at the output. When the TwinCAT object is called again, the information is reset.
 - The execution of the TwinCAT object is stopped after an exception occurs. TwinCAT itself remains in run mode. Restart execution: [ReleaseObjectStop](#) [► 226].
- **LogAndCatch:** Exceptions are caught with try-catch in the TwinCAT object. The execution of the TwinCAT object is stopped.
 - An exception does not cause a TwinCAT stop.
 - Outputs or internal states **cannot** contain NaN or inf.
 - The ExecutionInfo output is filled with information about an exception in the current cycle.
 - The execution of the code ends at the point of the exception. From there, the program jumps to the catch junction, i.e. internal states can be inconsistent.
 - The execution of the TwinCAT object is stopped after an exception occurs. TwinCAT itself remains in run mode. Restart execution: [ReleaseObjectStop](#) [► 226].
- **LogAndDump, LogHoldAndDump and LogCatchAndDump**
 - Behavior like LogExceptions
 - Additionally a dump file is stored on the runtime system in the TwinCAT folder *Boot*. For more on dump files, see [here](#) [► 230].

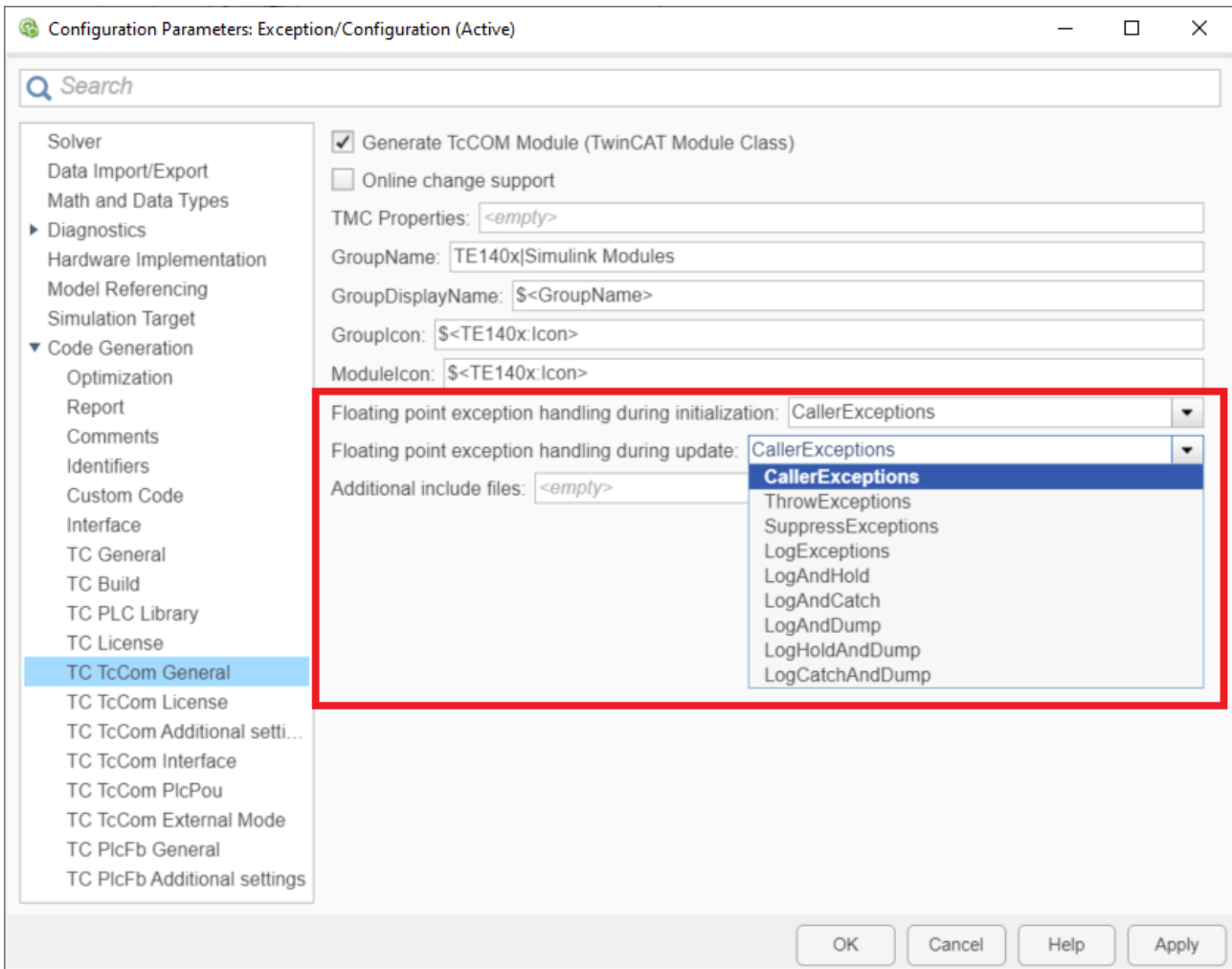


Version recommendation for 64-bit target systems

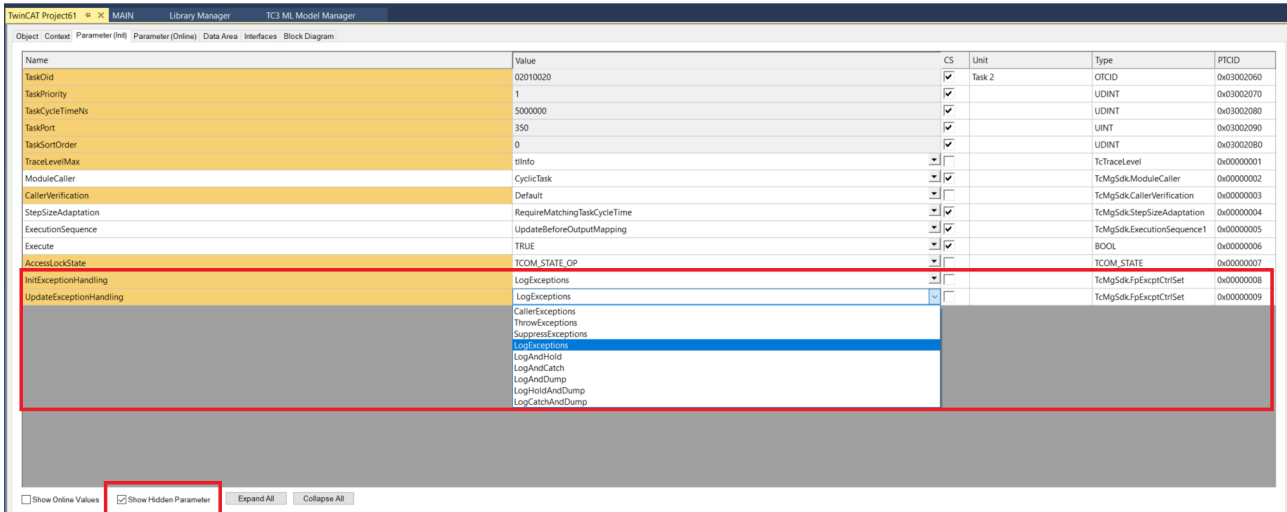
When using the "LogExceptions", "LogAndHold", "LogAndDump", "LogHoldAndDump" settings, it is recommended to use XAR version of at least 3.1.4024.35 and TE1400 version of at least 2.4.2.0.

<< Setting for TcCOM >>

You can define the behavior of a TcCOM object when exceptions occur under TC TcCOM General in the Code Generation Settings in Simulink®. The behavior must be defined separately for the initialization phase of the TcCOM and for the runtime phase (update phase).



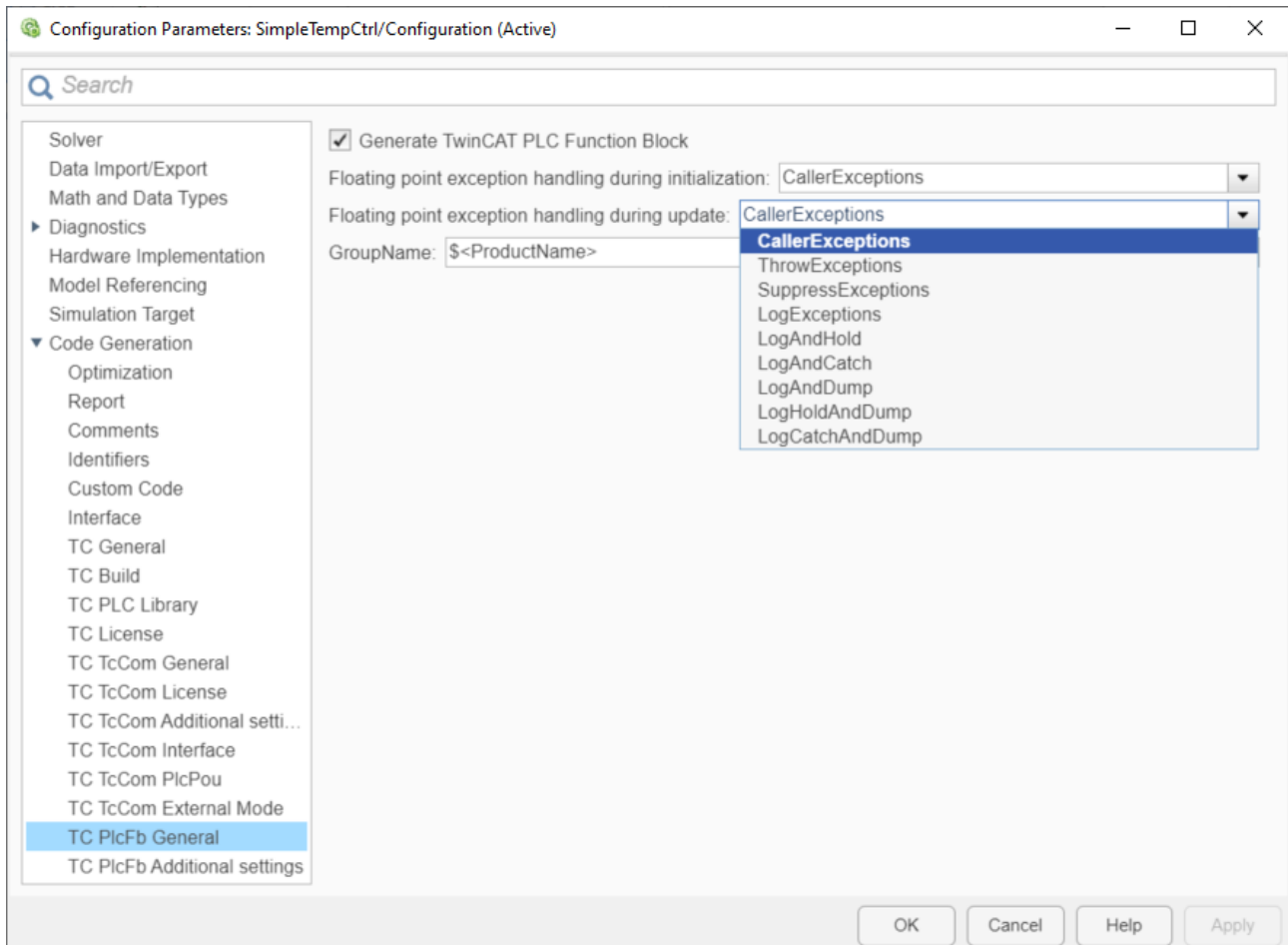
If you are working with an already compiled TcCOM in TwinCAT, you can also change the settings on the object instance afterwards. To do this, use the Parameters (init) tab and select **Show hidden Parameters**.



<< Setting for PLC-FB >>

The settings for the PLC-FB (PLC function block FB_<ModelName> in the PLC library) must be made independently of the settings for the TcCOM object at **TC PlcFb General**. A subsequent adaptation of the exception options when using the function block in TwinCAT is not provided.

Note that the other PLC function block FB_<ModelName>_TcCOM is a wrapper for a TcCOM object and therefore the exception settings from the TcCOM area are valid when it is used.

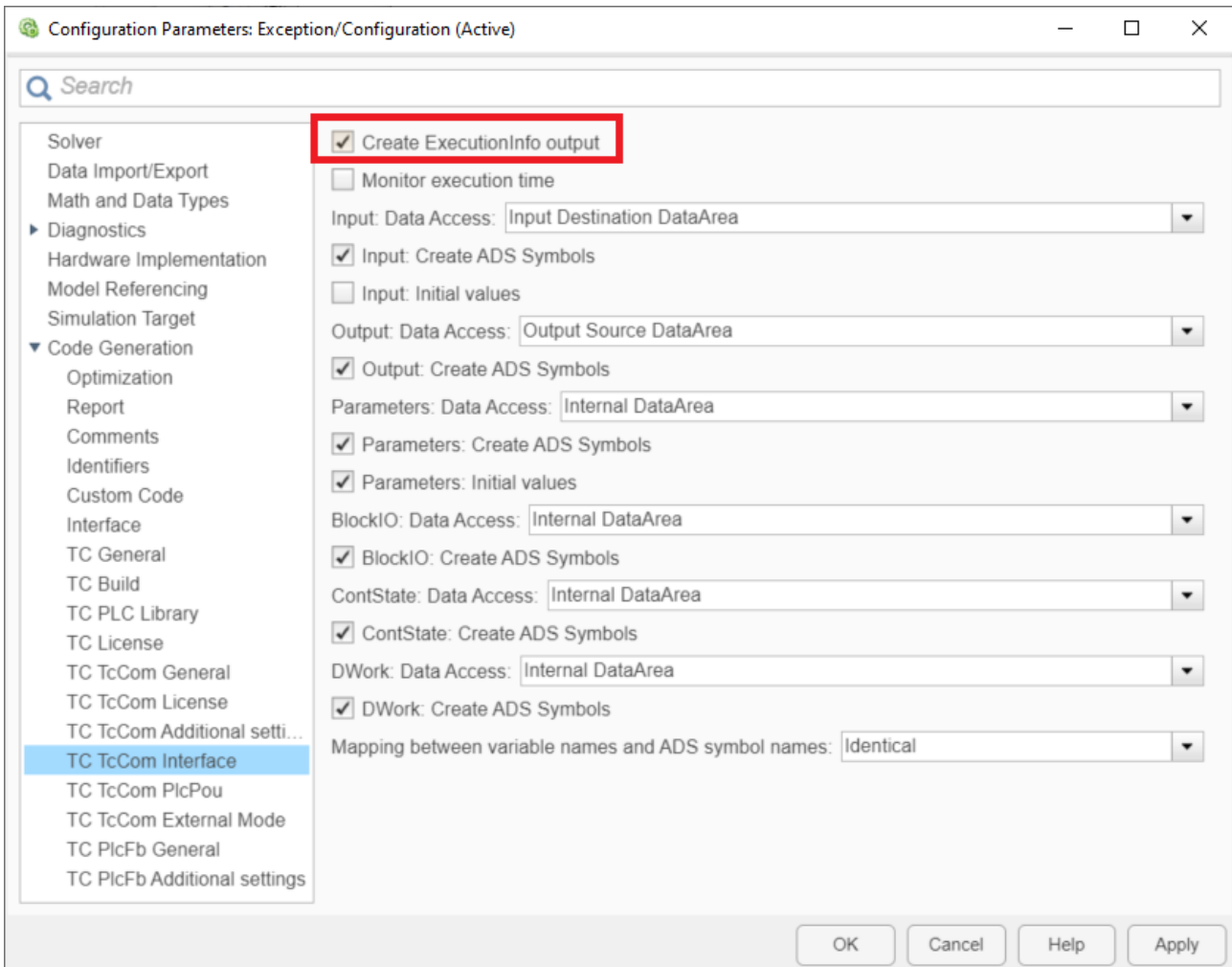


Optional ExecutionInfo Output

If exceptions are handled at object level, it makes sense to make corresponding information about occurred exceptions accessible at the object output. This output can be used to query whether an exception has occurred, what kind of exception it was, whether a dump file has been written, etc.

<< Setting for TcCOM >>

You can activate an additional "ExecutionInfo" output for the TcCOM object via the TC TcCom Interface entry.



The ExecutionInfo output is a structure with the following entries:

ExecutionInfo structure

Entry	Data type	Meaning
CycleCount	ULINT	Current cycle count (independent of an exception)
ExceptionCount	ULINT	Number of exceptions that have occurred so far
ActException	TcMgSdk.ExceptionInfo	More detailed explanation of the current exception (only first exception in the current cycle)

TcMgSdk.ExceptionInfo

Entry	Data type	Meaning
ExceptionCode	DINT	Exception code
TmxName	STRING(127)	Name of the tmx driver that threw the exception.
TmxVersion	ARRAY[0..3] OF UDINT	Version of the tmx driver that threw the exception.
InstructionAddr	UDINT	Relative address in memory; location where the exception occurred.
ReturnAddr	ARRAY[0..3] OF UDINT	Return addresses

Entry	Data type	Meaning
DumpCreated	BOOLEAN	TRUE if a dump file was created for the exception.

With the *InstructionAddr* it is possible to judge if the exception with the given *ExceptionCode* always occurs at the same place in the source code. If the *InstructionAddr* is the same for repeating exceptions, it always occurs at the same point in the code. Via *ReturnAddr* you can see where the calls came from that led to the location of the exception. So you can judge if the call that leads to the exception always takes the same call path. If the code is called from outside the Tmx driver, there is a 0 in *ReturnAddr*.

Exception code	Meaning
0xC000008E	Divide by zero
0xC000008F	Inexact result
0xC0000090	Invalid operation

If you use the [TcCOM Wrapper FB \[► 209\]](#), the ExecutionInfo structure is available at the function block. Note that according to the TwinCAT programming conventions the entries carry prefixes corresponding to the data type.

<< Setting for PLC-FB >>

The PLC-FB always contains `nExceptionCount` and `stActiveException` as properties according to the above definition. I.e. no checkbox has to be set separately to get these properties. The only parameter that is not available in comparison to the TcCOM is the cycle count, since this can be implemented very easily in the PLC itself if required.

Handle execution stop of a TwinCAT object

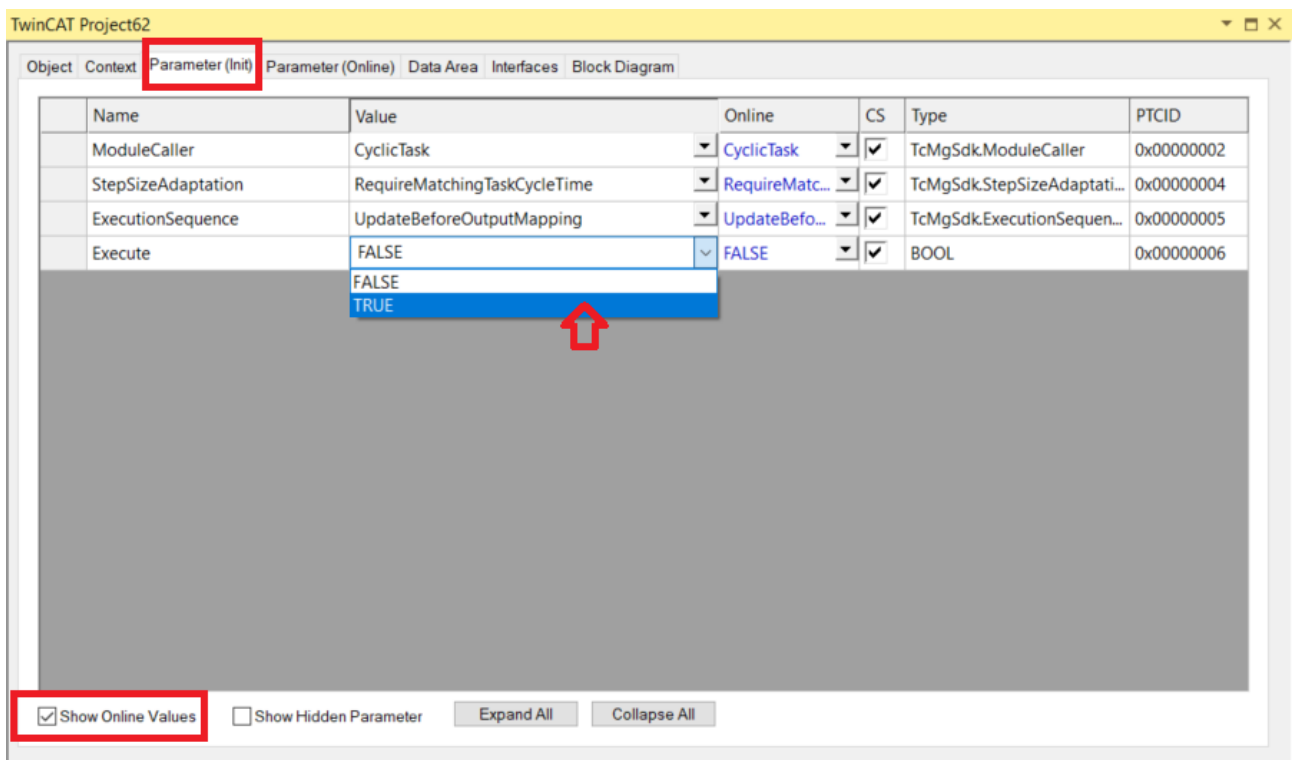
LogAndHold and LogHoldAndDump

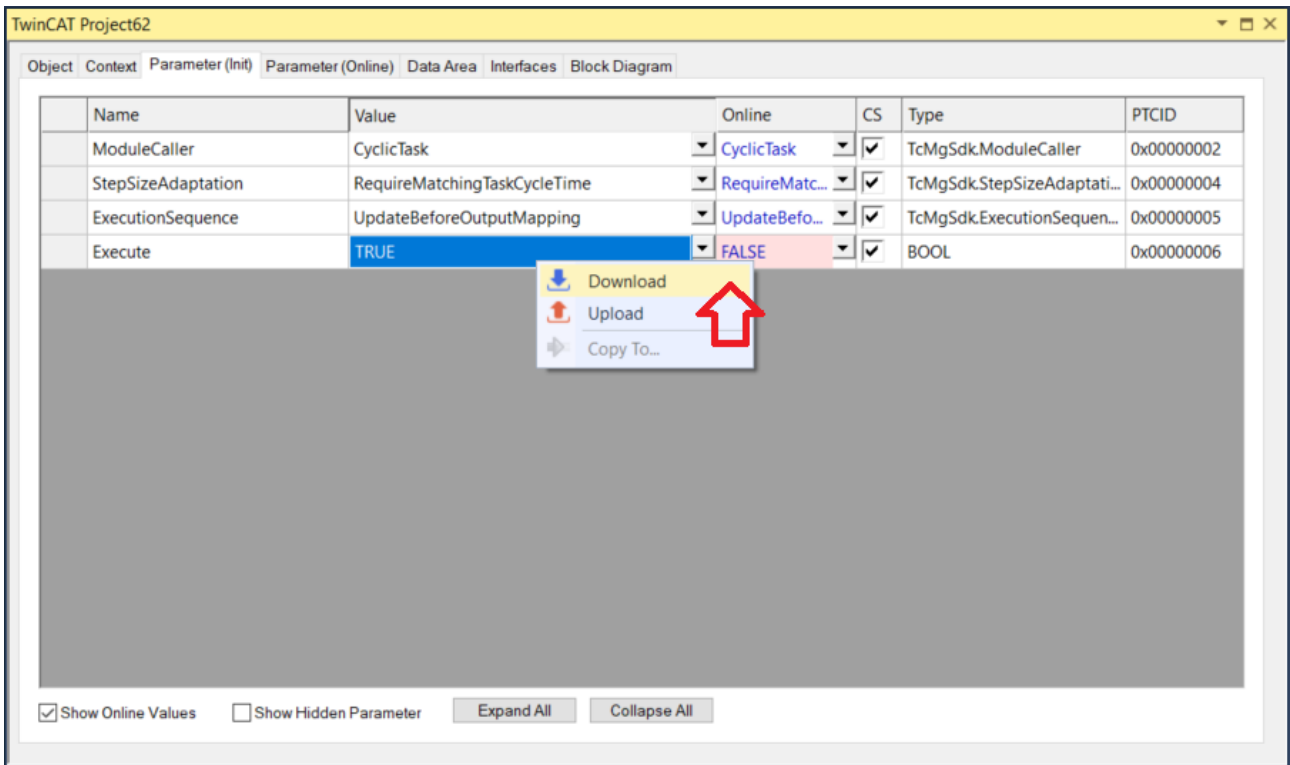
In the event of an exception, execution of the code in the TcCOM object or PLC function block (PLC-FB) concerned is stopped by setting the Execute parameter to FALSE.

<< Setting for TcCOM >>

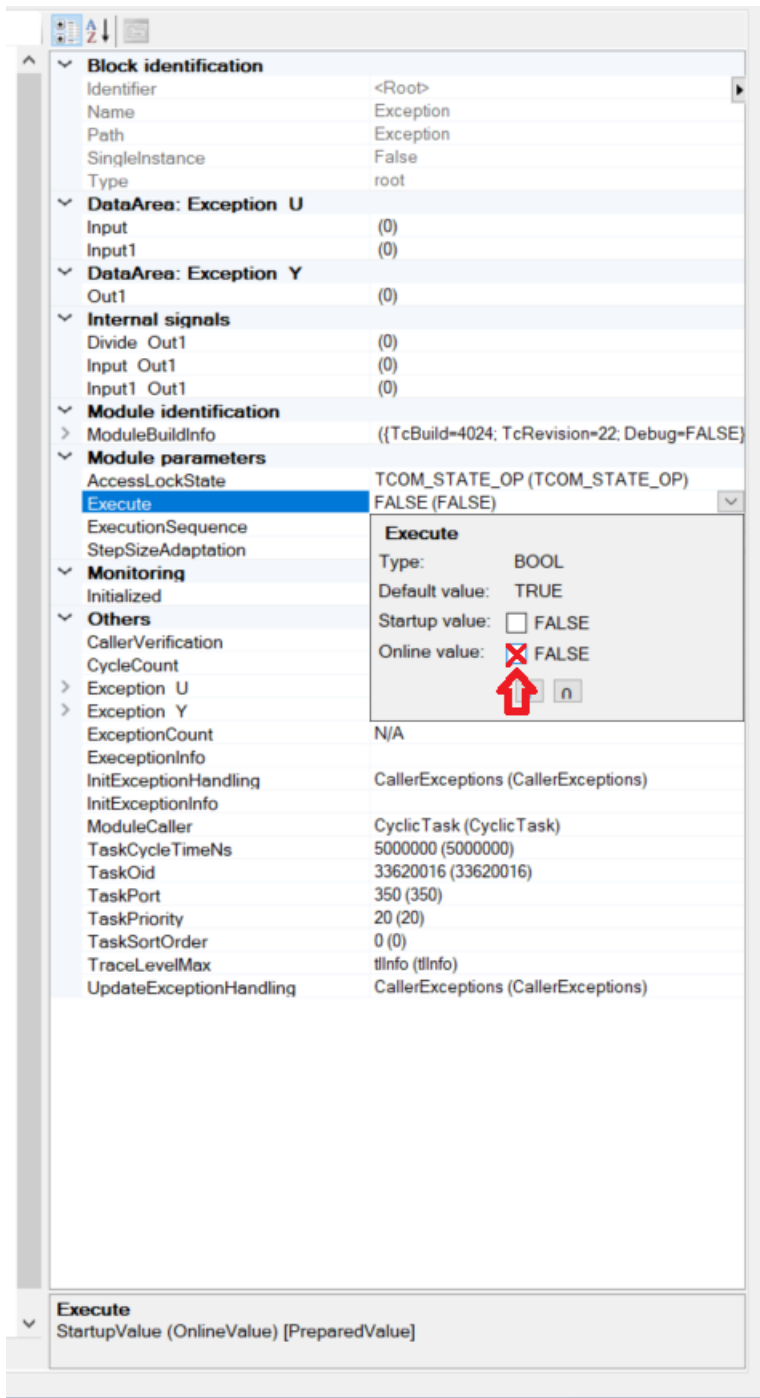
The parameter Execute can be read or written from the XAE and via ADS.

In the XAE, you can display and change the online values of the TcCOM object under Parameters (Init).

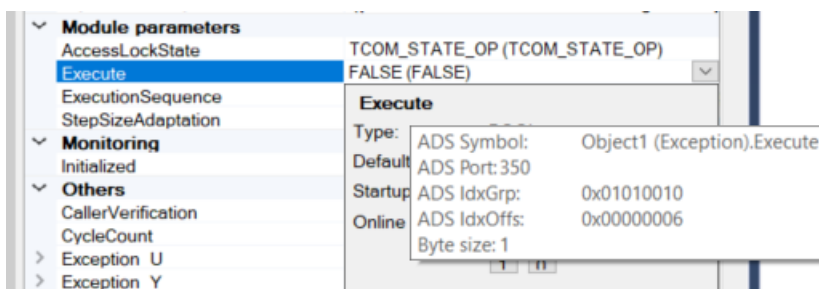




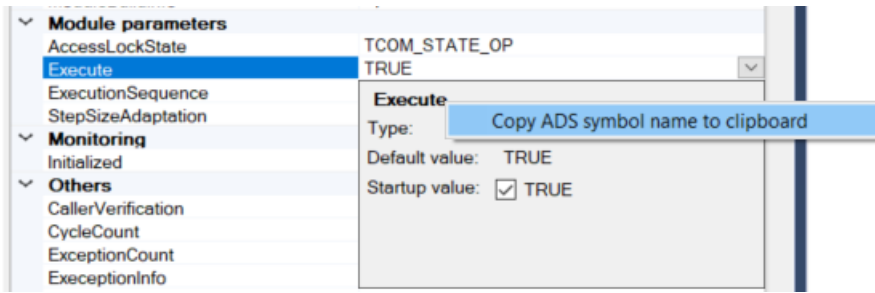
In the block diagram the parameter is offered to you under *Module parameters*.



If you move the mouse over the Execute name in the change dialog, you will be shown the ADS address of the parameter, as with all other parameters. This allows you to set the parameter also by ADS.



By right-clicking on the name **Execute** you can also save the ADS symbol information to the clipboard. This also applies to all other parameters.



If you use the TcCOM Wrapper FB, you can change the Execute parameter by writing to the bExecute property.

<< **Setting for PLC-FB** >>

The Execute parameter is available on the FB as property bExecute with read and write rights. Use this property to restart the execution of the FB.

LogAndCatch and LogCatchAndDump

In addition to the parameter *Execute*, the online parameter *Initialized* also changes to FALSE in the case of LogAndCatch and LogCatchAndDump. The module must be reinitialized before the module can perform a calculation again. This is necessary because internal states, due to the termination of code execution at the point of the exception, can be inconsistent.

<< **Setting for TcCOM** >>

Reinitialization can only be performed by returning the TcCOM object to the "Init" state and moving it to OP again. At runtime, only TcCOM objects that have no mappings can be shut down, otherwise active mappings would block the shutdown. A new initialization is only possible in the case of active mappings on the TcCOM by restarting the entire TwinCAT runtime. It is therefore recommended to use the TcCOM Wrapper FB [► 209]. This can be used to call the TcCOM from the PLC and does not require any mappings to access its inputs and outputs. Accordingly, the TcCOM object can also be reinitialized during runtime.

● **Property settings for the TcCOM Wrapper FB**

I In the following sample code properties, e.g. bExecute, are read. Create the TcCOM Wrapper FB with TcCom Wrapper FB properties set and with the "CyclicUpdate" option for the properties so that the code below matches the wrapper.

```
PROGRAM MAIN
VAR
  stInitTemp : ST_FB_SimpleTempCtrl_TcCOM_InitStruct := (nObj := 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_STATE_OP THEN

  // call the module
  fbTempCtr(stSimpleTempCtrl_U := Inputs, stSimpleTempCtrl_Y => Outputs, stExecutionInfo => ExecutionOut);

  // handle exceptions
  IF ExecutionOut.ActException.ExceptionCode <> 0 THEN
    // collect exception information
    (* ..... *)

    // reinit TcCOM
    fbTempCtr.Reinit(stReInit := stInitTemp);
  END_IF
END_IF
```

Note that the ReInit method is executed synchronously, i.e. depending on the cycle time and the time required to reinitialize, cycle overruns may occur.

<< Setting for PLC-FB >>

You can use the property `bInitialized` on the FB to check whether the stored module is not (no longer) initialized. You have read-only access here. Reinitialization is currently not possible via a method on the FB. The PLC runtime, alternatively the entire TwinCAT runtime, must be restarted.

Dump files

Writing the dump file may take a few cycles. It is best to use a separate task for the TcCOM object or the PLC-FB in question that does not block any important tasks.

Dump files are only written with a TwinCAT XAR version $\geq 3.1.4024.22$, otherwise you get a corresponding warning.

In the case of *LogAndDump* the execution of the code is continued cyclically after the occurrence of an exception, accordingly exceptions can occur cyclically which could lead to persistent cycle timeouts. Therefore, the online value of the parameter *UpdateExceptionHandling* is set to *LogExceptions* after the dump file has been written, i.e. the writing of dump files is deactivated, but can subsequently be switched on again, e.g. by ADS or intervention via the XAE under parameter (Init).

The created dump file is stored on the runtime PC in the boot folder and can be copied from there to another PC for analysis. If you use a TwinCAT version lower than 3.1.4024.x you can open the dump files with [WinDbg](#) and start your analysis.

4.8.6 Using Realtime Monitor time stamps

MATLAB® commands, such as `tic` and `toc`, are popular ways to analyze the performance of code sections in MATLAB®. These commands are not usable in this form during TwinCAT runtime.

For this purpose, TwinCAT provides the TwinCAT Realtime Monitor, which evaluates time stamps in the source code and displays them for analysis. Setting Realtime Monitor time stamps is supported in MATLAB® code, i.e. the time stamps are set in MATLAB® and can be evaluated by the Realtime Monitor after code generation and instantiation in TwinCAT. Running time stamps in MATLAB® results in output to the MATLAB® console.

Class: `TwinCAT.ModuleGenerator.Realtime.LogMark`

Methods: Start, Stop and Mark

MATLAB® documentation: `doc("TwinCAT.ModuleGenerator.Realtime.LogMark")`

i Example in MATLAB®

```
TwinCAT.ModuleGenerator.Samples.Start("BaseStatisticsLogMark")
```

Use of the time stamps is limited to MATLAB® code and requires appropriate embedding in MATLAB® function blocks for use in Simulink®.

4.9 FAQ

4.9.1 Change model parameters at runtime

Can I change model parameters during runtime in TwinCAT?

Yes, please note the following settings:

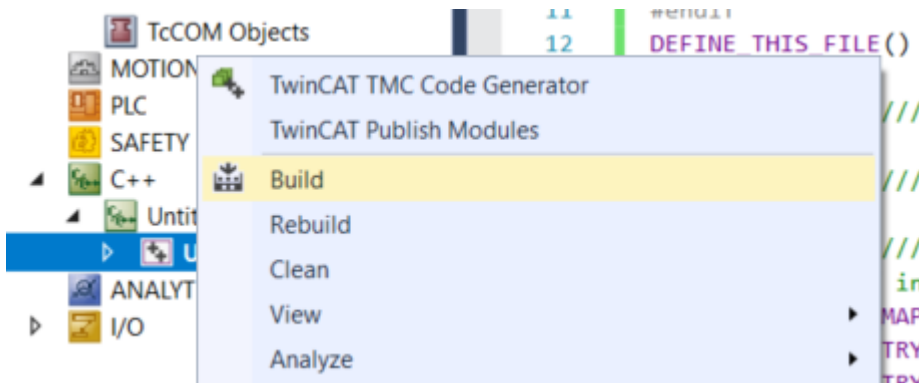
- Optimization > Default parameter behavior: Tunable
If the parameter is set in this way, model parameters can be set at runtime. See also [Parameterization of a module instance](#) [► 186].

- Interface > Code interface packaging
You have the options "Nonreusable function", "Reusable function" and "C++ Class" here. The settings affect whether you can instantiate multiple instances of a TcCOM in TwinCAT and whether you can then also set their model parameters individually or dependent on each other. See also [Parameterization of several module instances](#) [▶ 191].

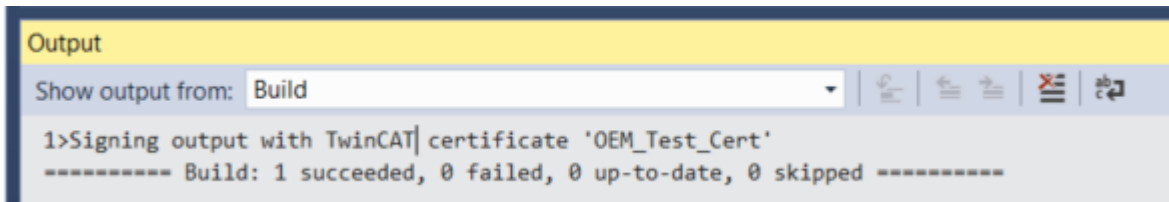
4.9.2 Build of a sample fails

All samples supplied (list by `TwinCAT.ModuleGenerator.Samples.List` in the MATLAB® Command Window) have been checked by tests at Beckhoff Automation. If a build of a sample still does not run successfully, it is likely that something needs to be adjusted during setup on your engineering PC.

- ✓ To test the platform toolset without the influence of MATLAB® please create a TwinCAT Versioned C++ project in TwinCAT (open TwinCAT in Visual Studio) .
1. Right-click **Add New Item** on C++ Tree Item.
 2. Then select **TwinCAT Module Class with Cyclic Caller**.
⇒ A C++ project appears in the TwinCAT Tree under C++.
 3. Build the C++ project and view the Output Window in TwinCAT.



- ⇒ The output window should return "1 succeeded" for the build process. If this is not the case, check whether you have installed the **Desktop development with C++** option in Visual Studio.



4.9.3 Problems with the block diagram representation in TwinCAT XAE

TcCOM modules created with the TwinCAT Target for Simulink® version 2.x.xxxx.x and higher require a TC3 BlockDiagram version 1.4.1419.0 and higher for correct display in TwinCAT XAE.

Where can I find the version of the TC3 BlockDiagram?

- Under Programs and Features in the **Control Panel > Beckhoff TwinCAT 3 BlockDiagram**.
- In the block diagram in TwinCAT XAE > right-click in the window > **About TC3 BlockDiagram**.

If your TwinCAT XAE installation contains an earlier version of the TC3 BlockDiagram:

- Can you install the *TwinCAT Tools for MATLAB and Simulink* setup. This includes a new TC3 BlockDiagram version.
- You can contact Beckhoff support to request a separate TC3 BlockDiagram setup.

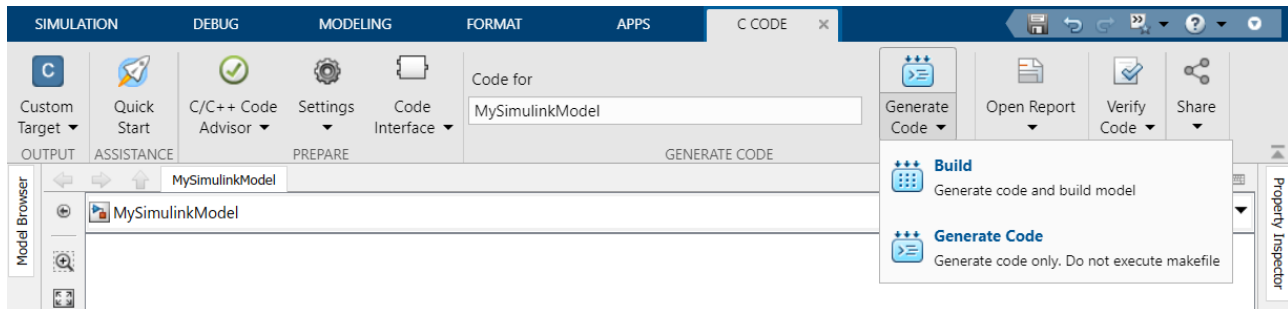
4.9.4 Can I use TE1400 version 1.2.x and version 2.x at the same time?

Yes, this is possible. Simply install both products on your system and select the appropriate target to differentiate between the two versions.

TwinCatGrT.tlc for version 2.x and *TwinCAT.tlc* for 1.2.x

4.9.5 What is the difference between "Build" and "Generate code"?

In the Simulink Coder™ App, you can choose between "Build" and "Generate Code":



If you have set *TwinCatGrT.tlc* as target, both options have the same function, because the *TwinCatGrT.tlc* does not run through the makefile of MathWorks®.

If you do not want to run the build process, but only generate the C++ code, uncheck the checkbox **Run the publish step after project generation** under TC Build.

What is the difference between "Build" and "Publish"?

"Publish" refers to the successive execution of the build process for specific TwinCAT platforms. From Simulink®, the corresponding binaries are then created one after the other for the platforms activated under TC Build, so that it can be decided afterwards for which target platform the compiled functions are to be used.

4.9.6 I can't change the parameters of a module in TwinCAT

"Inlined" is set in the *TwinCarGrT.tlc* as the default value for the parameter **Default parameter behavior**. Change this to "Tunable" or configure which parameters should be marked as "Tunable" via the button **configure..**

4.9.7 Mapping is lost with Reload TMI/TMC

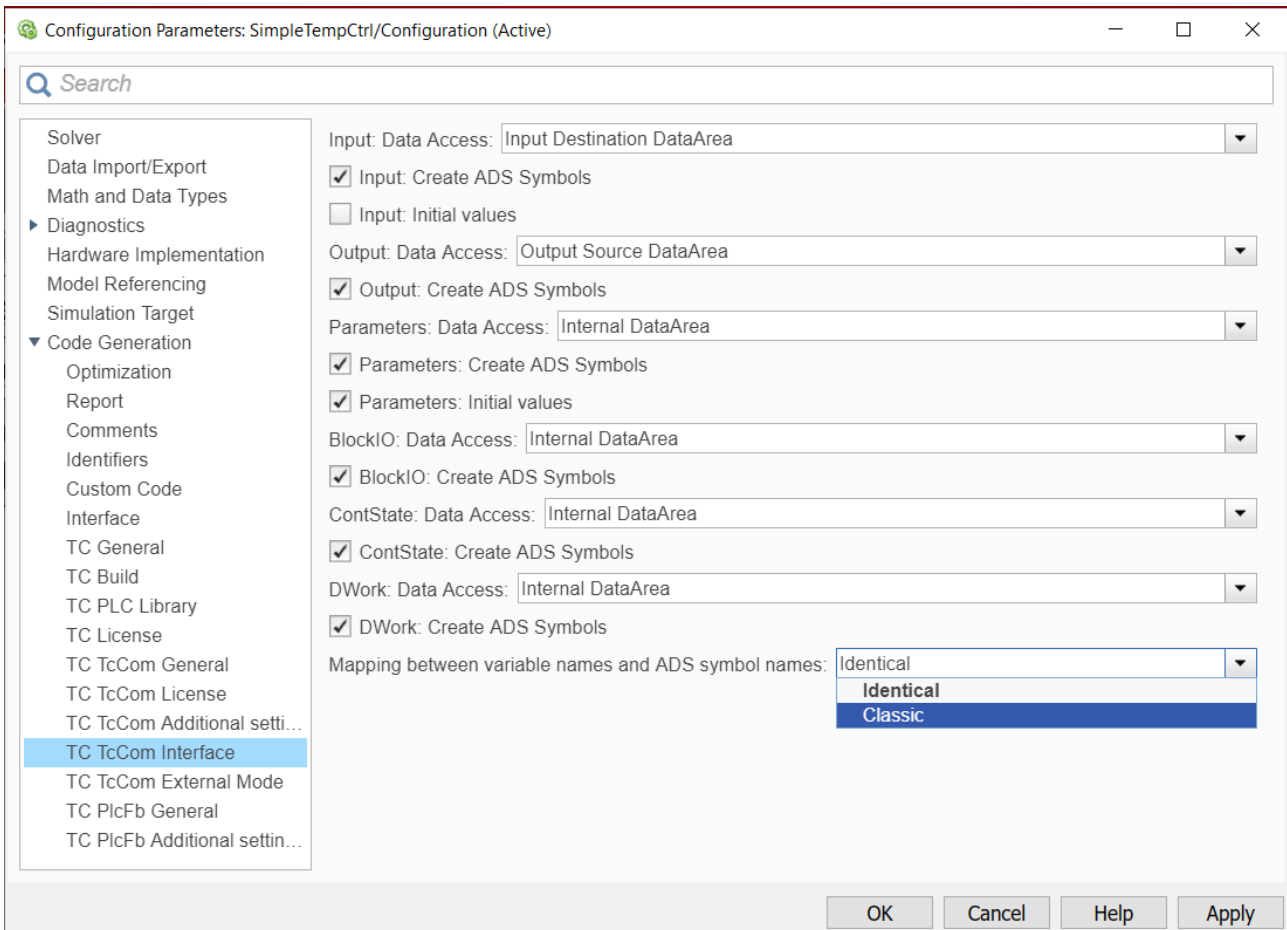
Challenge:

You have already TcCOM objects in your TwinCAT solution, which you have created with the Target for Simulink® version 1.2.xxxx.x. You now want to create a new TcCOM object with the Target for Simulink® version 2.x.x.x and replace the newly created TcCOM with Reload TMI/TMC File... in your existing TwinCAT solution.

In the default settings, you lose the mapping information by doing this.

Solution:

Under TC TcCOM Interface, set the "mapping between variable names and ADS symbol names" to "Classic" and use this to create the new TcCOM object.



This means that the mapping is retained if you now replace your old TcCOM object in TwinCAT with Reload TMI/TMC File....

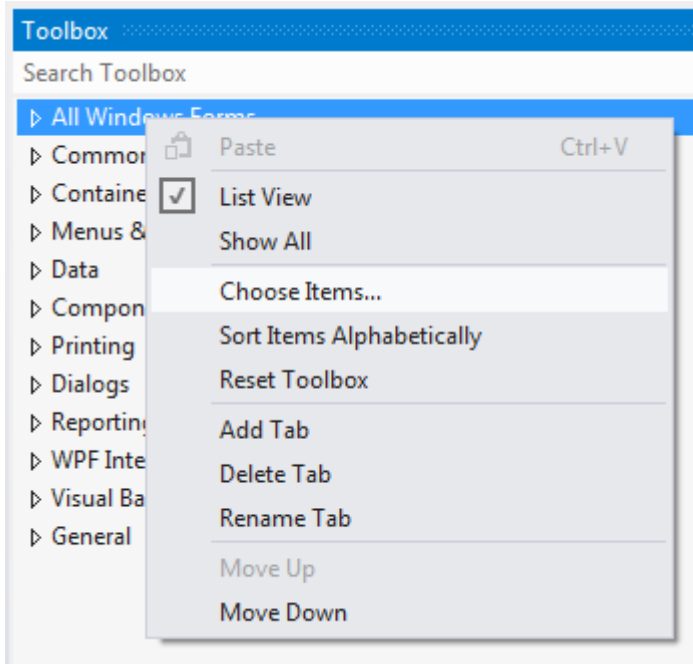
4.9.8 Integrating the block diagram controls in .NET

The option displaying the block diagram in the TwinCAT XAE environment can also be integrated in separate visualizations.

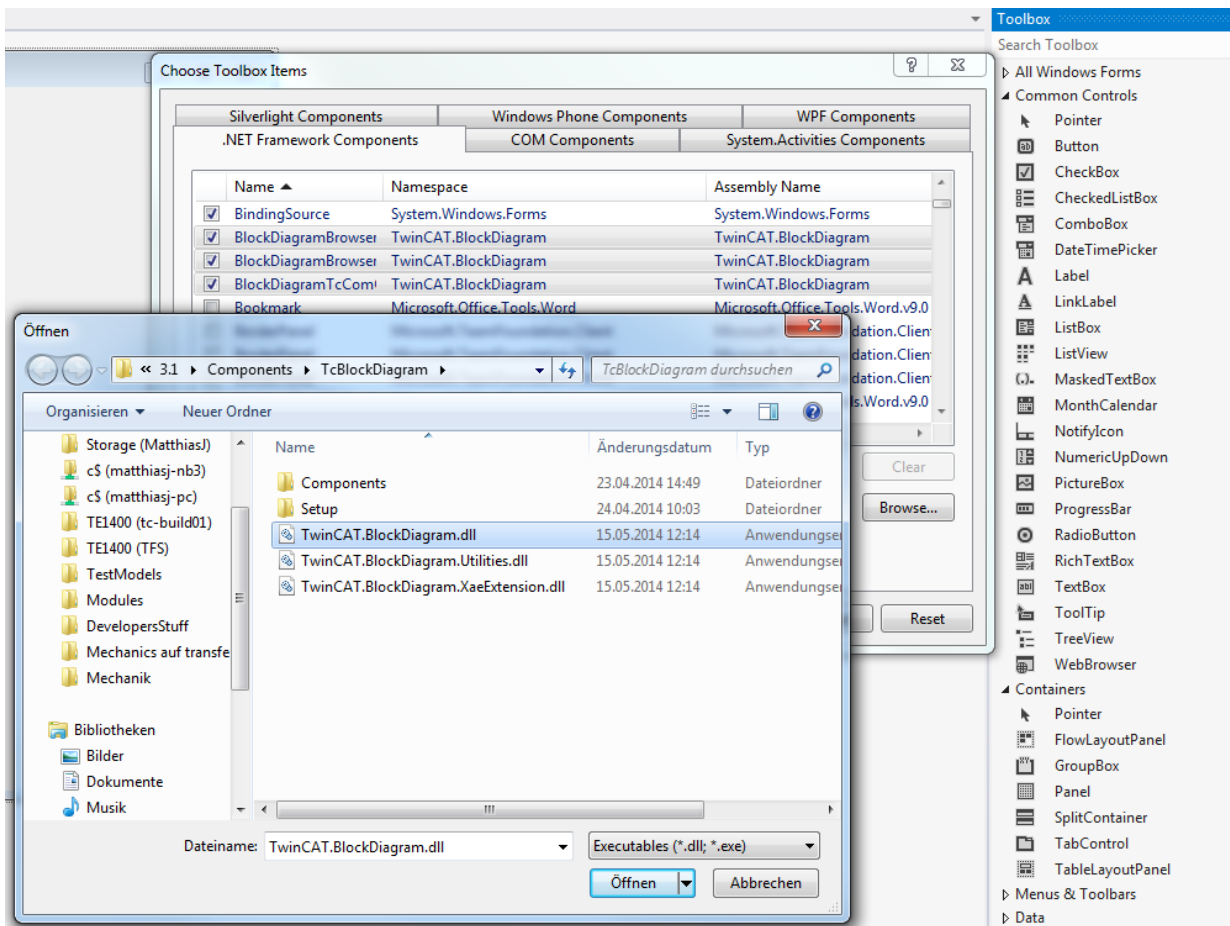
A sample program can be downloaded here: https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/11697311755/.zip

- ✓ The following steps are required:
 1. Create a new Windows Forms application.
 2. Add TwinCAT.BlockDiagram.dll to the toolbox.

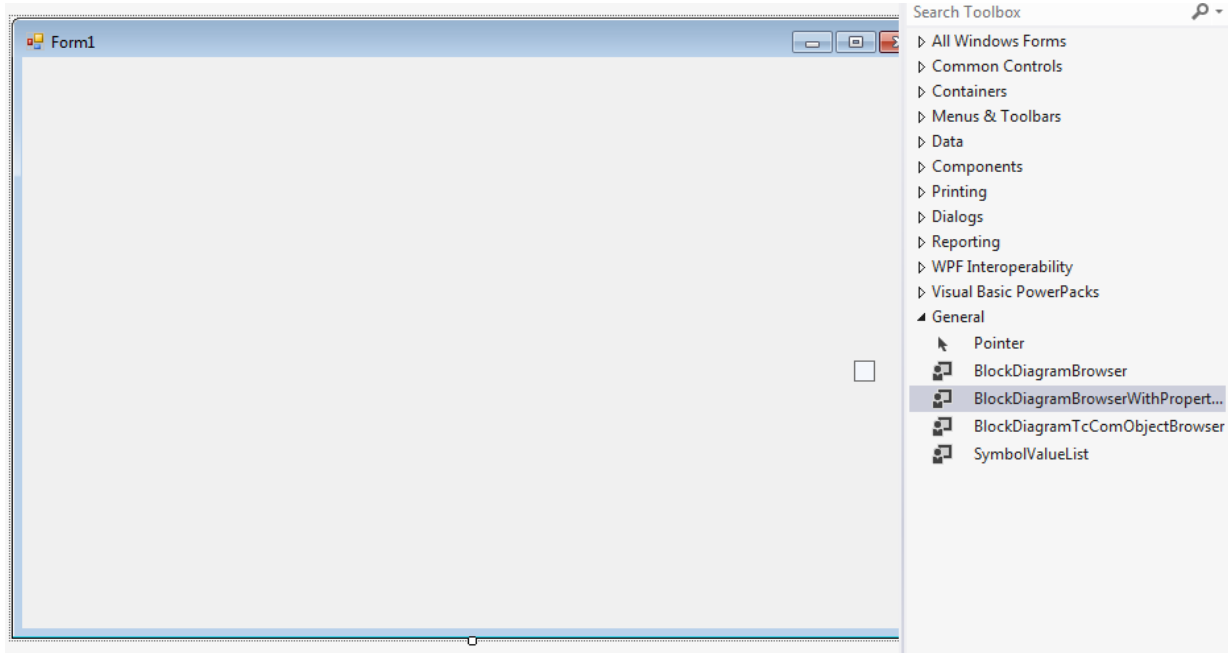
3. To do this, select the **Choose Items...** entry in the context menu.



4. Navigate to `TwinCAT.Blockdiagram.dll`, which can be found under *<TwinCAT installation path>\3.1\Components\TcBlockDiagram*.



5. Add a TcBlockdiagram control instance to the Windows Forms object using drag and drop (BlockDiagramBrowser or BlockDiagramTcComObjectbrowser).



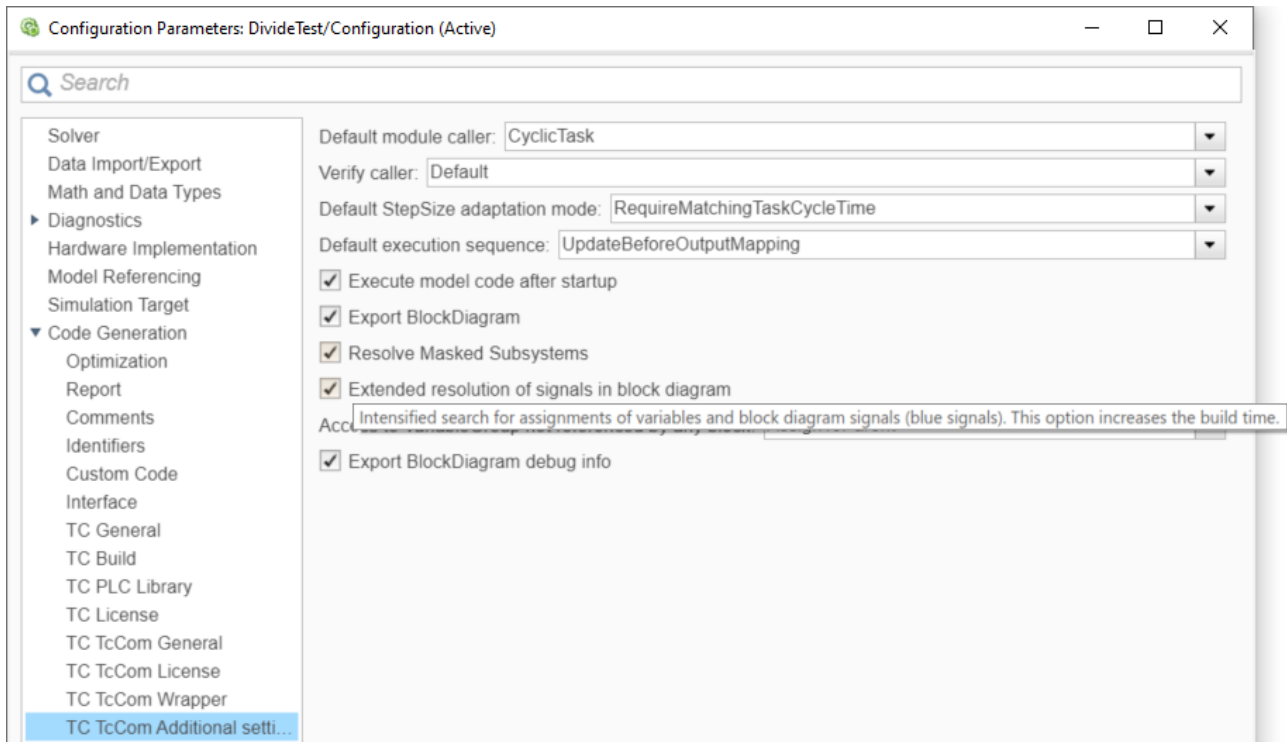
4.9.9 Observable signals in the TwinCAT block diagram

Which settings influence the number of "blue signals" in the TwinCAT block diagram?

Basically the TwinCAT Target for Simulink® needs a variable in the C/C++ code, which can be assigned to the signal in the block diagram. If a signal in the block diagram is not marked in blue, it means that either no variable exists (because it was dropped during code generation due to optimization of the code) or the variable could not be assigned.

To suppress omission of variables by code optimization, you can use **Test Points** in Simulink®: see *Configure Signals as Test Points* in the Simulink® documentation.

At **TC TcCom Additional settings** you can also set the entry **Extended resolution of signals in block diagram**. If this entry is active, an intensified search for assignments of variables and signals is performed. Note that this search takes time, so exporting the block diagram will take longer.



4.9.10 Using Simulink® Strings

Simulink® strings are explicitly allowed and can be used with the TwinCAT Target for Simulink®.

Restriction

Depending on the MATLAB® release version, code interface packaging setting and the set C/C++ standard, the Simulink Coder™ compiles a Simulink® string into the `std::string` data type. If this Simulink® string is then used as a model *input* or model *output*, please note that these entries cannot be linked to other objects in TwinCAT by mapping. Mapping in TwinCAT assumes a static data type size, which is not the case with `std::string`.

Handling in TwinCAT

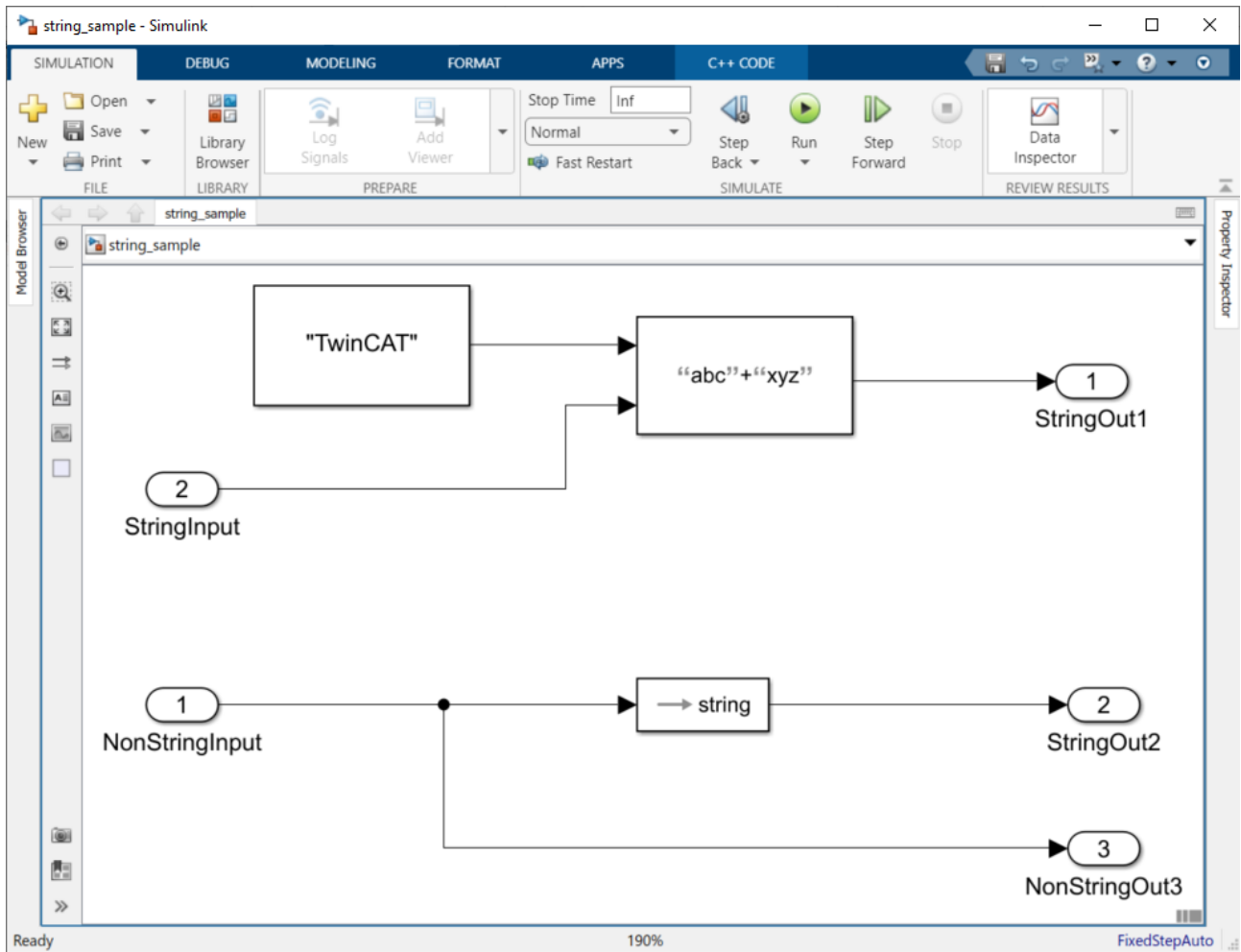
So that you can still use the inputs and outputs, it is recommended that the PLC-FB (no mappings necessary) or the TcCOM Wrapper FB are used. For the standard inputs and outputs of the FBs, the Simulink® string entries are not displayed in either case. These are to be set separately via getter and setter methods on the FB.

i Simulink® bus with Simulink® strings: restricted use

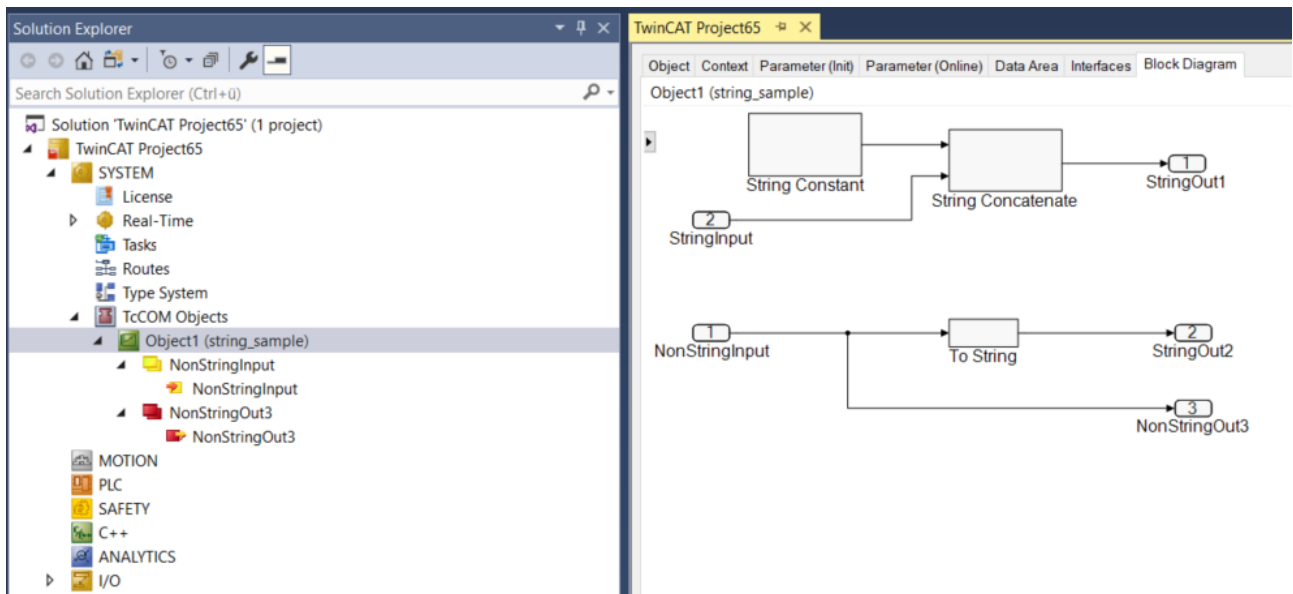
The following situation is currently not supported: a Simulink® string cannot be used in a Simulink® bus that serves as input or output of the model if it is mapped as `std::string` by the Simulink Coder™.

Example

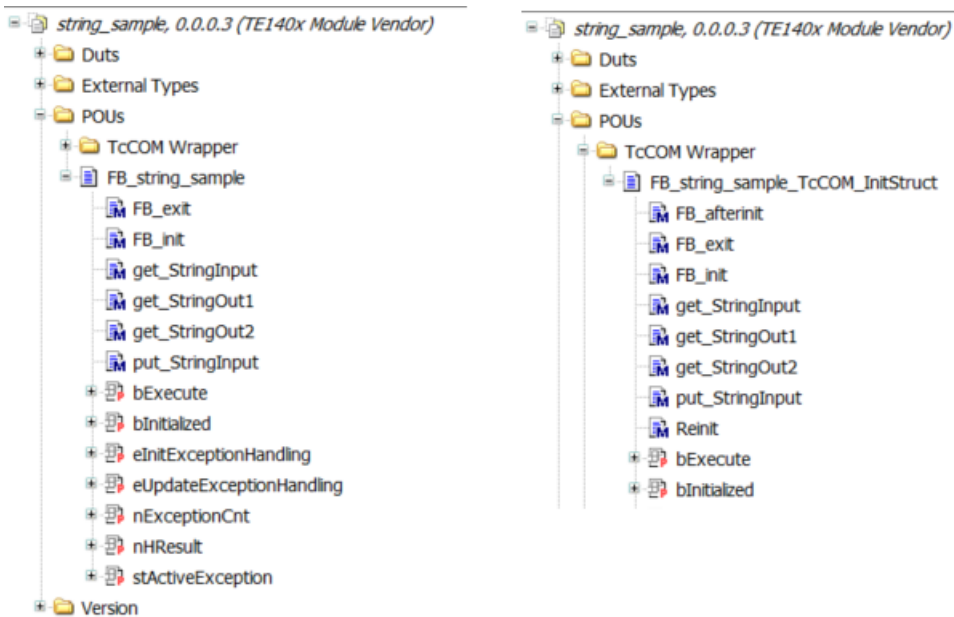
Let's take the following Simulink® model with a mixture of string and non-string inputs and outputs.



When this model is compiled with MATLAB R2022a and "C++ Class" code interface packaging, the data type `std::string` is generated by the Simulink Coder™. In the following figure, it can be seen that the string inputs and outputs are **not** present in the process image in TwinCAT. Only the non-string inputs and outputs are present in the process image.



To be able to write and read the strings, either the TcCOM Wrapper FB or the PLC FB must be used. Corresponding getter and setter methods are automatically created at the function blocks.



Sample code using the PLC-FB:

```

VAR
    fbStringSample : FB_string_sample;

    myStringIn    : T_MaxString;
    myStringOut1  : T_MaxString;
    myStringOut2  : T_MaxString;
    nSize         : ULINT;
    nSize2        : ULINT;
    fIn           : LREAL;
    fOut          : LREAL;
END_VAR

// put sting input
fbStringSample.put_StringInput(c_str := ADR(myStringIn));

// call function
fbStringSample(fNonStringInput := fIn, fNonStringOut3 => fOut);

// get string outputs
nSize := SIZEOF(myStringOut1);
fbStringSample.get_StringOut1(c_str := ADR(myStringOut1), size := nSize); // size in VAR IN OUT!

nSize2 := SIZEOF(myStringOut2);
fbStringSample.get_StringOut2(c_str := ADR(myStringOut2), size := nSize2);

```

4.9.11 Are there limitations with regard to executing modules in real-time?

Not all access operations that are possible in Simulink® under non-real-time conditions can be performed in the TwinCAT real-time environment. Known limitations are described below:

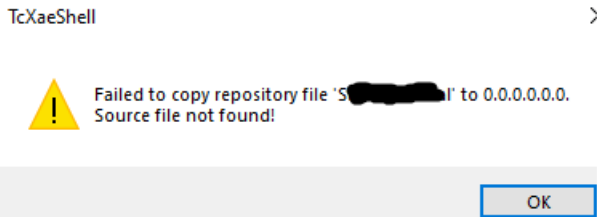
- **Direct file access:** No direct access to the file system of the IPC can be realized from the TwinCAT runtime. For writing .mat files please use the TwinCAT File Writer block instead of the ToFile block of Simulink®.
- **Direct hardware access:** Direct access to devices/interfaces requires a corresponding driver, e.g. RS232, USB, network card, ...
It is not possible to access the device drivers of the operating system from the real-time context. At present it is therefore not easily possible to establish an RS232 communication for non-real-time operation with the Instrument Controller Toolbox™ and then use this directly in the TwinCAT runtime. However, TwinCAT offers a wide range of communication options for linking external devices, see [TwinCAT 3 Connectivity TF6xxx](#).

- **Access to the operating system API:** The API of the operating system cannot be used directly from the TwinCAT runtime. An example is the integration of *windows.h* in C/C++ code. This is integrated by the Simulink Coder™ if the FFTW implementation of the FFT block from the DSP Systems Toolbox™ is used (but not with the Radix 2 implementation), for example.
- **Precompiled libraries:** It is possible that during code generation by the Simulink Coder™ no platform-independent C/C++ code is generated, but precompiled libraries are included. In these cases, no real-time execution in TwinCAT is possible.

4.9.12 Message: Failed to copy repository

After activating the TwinCAT Configuration the following error message appears in TwinCAT XAE?

Failed to copy repository file ...



The cause is usually that no or not all files, especially the TMX drivers, were found in your engineering repository. The TwinCAT XAE tries to copy the driver to the XAR system but does not find the correct file (check if you have a driver for the target platform of the selected XAR system in the engineering repository).

Consequential error in the Error dialog in XAE is for example:

'TCOM Server' (10): Error loading repository driver 'C:\TwinCAT\3.1\Boot\Repository\<model vendor>\<model name>\<version>\<tmx-name>' - hr = 0xc0000225

Since the TMX file could not be copied to the XAR system, it could not be loaded. Another consequential error is a link error "Could not link external function".

✓ Information and remedy for this error pattern:

1. What is the Engineering Repository and where can I find it?
⇒ See [automatically created files \[▶ 120\]](#).
2. How do I ensure that when copying compiled models to other XAE systems, all necessary files are always transferred and the folder structure remains correct?
⇒ See [TMX archive \[▶ 135\]](#).

4.10 Samples

Samples provided by Beckhoff Automation are installed on your system with the *TwinCAT Tools for MATLAB® and Simulink®* setup.

You can use the following command to display all available samples:

```
TwinCAT.ModuleGenerator.Samples.List
```

```

Command Window
New to MATLAB? See resources for Getting Started.
>> TwinCAT.ModuleGenerator.Samples.List

TE140x Samples:

SimpleTemperatureController:
  Products: TE1400
  Topics: Basic principles
  Level: 1
  Description: A very simple temperature controller that covers the basics.
  Start

BaseStatistics:
  Products: TE1401
  Topics: Basic principles
  Level: 1
  Description: A MATLAB code generation sample that covers the basics.
  Start

TemperatureController:
  Products: TE1400
  Topics: Parameter access, Bus objects, Test points, Referenced models, External Mode, Generating TwinCAT modules from subsystems
  Level: 2
  Description: A simple temperature controller with PWM output. A second model simulates the dynamic of the controlled system. This mod
  Start

FileAccess:
  Products: TE1401
  Topics: Basic principles
  Level: 10
  Description: A MATLAB code generation sample that covers file access capabilities.
  Start

fx >>

```

You can access the samples by clicking on the blue start link. To do this, the sample code is copied to your user directory so that you do not change the original sample. You can work with the copy of the sample accordingly and try it out.

Also available for displaying and starting individual samples:

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

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

The argument SampleName is to be passed as a string, e.g.:

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

Spaces in the SampleName are to be replaced with underline in the argument, e.g. "Combine Modules" -> TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules').

4.10.1 TwinCAT Automation Interface: use in MATLAB®

Short description of the Automation Interface

TwinCAT XAE configurations can be automatically generated and edited via programming/script codes using the TwinCAT Automation Interface. The automation of a TwinCAT configuration is available thanks to so-called Automation Interfaces, which can be accessed via all COM-capable programming languages (e.g. C++ or .NET) and also via dynamic script languages such as Windows PowerShell, IronPython or even the (obsolete) Vbscript. Use from the MATLAB® environment is also possible.

Detailed documentation of the product can be found here: [TwinCAT Automation Interface](#)

Use in MATLAB®

The Automation Interface can be made visible in MATLAB® through the command `NET.addAssembly`. This will enable you to use the interfaces ([Automation Interface API](#)) described in the product documentation. You can also find many programming samples for use from C# and PowerShell ([Automation Interface Configuration](#)).

In order to simplify the entry from MATLAB® for you, you can find below a sample implementation for MATLAB® on the basis of a MATLAB® class, which you can use, modify and expand.

4.10.1.1 Sample: Tc3AutomationInterface

Overview

The sample code consists of two m-files:

- *Tc3AutomationInterface.m*: MATLAB® class that implements several frequently used methods.
- *Tc3AutomationInterfaceGuide.mlx*: MATLAB live script that calls the MATLAB® class as an example.

● Call sample with MATLAB®

i The TwinCAT Tool for MATLAB® and Simulink® Setup installs the sample on your system. Call the sample with the MATLAB® Command Window:

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

The MATLAB® script

The MATLAB® script provides a sample of how you can generate a TwinCAT solution, scan the EtherCAT master for I/Os, instantiate two TcCOM modules, link them and activate the project on a target.

In order to be able to run the script, the two TcCOMs used must be present in your *publish directory* `%TwinCATDir%\CustomConfig\Modules\`. For this, download the Temperature Controller sample from the TE1400 | Target for MATLAB®/Simulink®. Then copy the file folder from the directory `\TE1400Sample_TemperatureController_PrecompiledTcComModules\Actual TwinCAT versions\` into the *publish directory*.

Run the m-file *Tc3AutomationInterface_Testbench.m*. The latest Visual Studio instance available on your system is opened in the background and the TwinCAT solution is configured, saved and activated.

The MATLAB® class

The properties

All variables and interfaces belonging to the instance of the class are contained in the properties of the *Tc3AutomationInterface* class. Hence, several TwinCAT solutions can be built up in a MATLAB® script by generating an instance of the class for each solution. There are then no overlaps.

The constructor

```
function this = Tc3AutomationInterface
```

The constructor loads all necessary assemblies and, if successful, sets the `AssembliesLoaded` property to `TRUE`. The loaded assemblies are:

- `EnvDTE` and `EnvDTE80`: libraries for the Visual Studio Core Automation. Necessary for the configuration of Visual Studio.
- `TCatSysManagerLib`: TwinCAT Automation Interface library for the configuration of a TwinCAT solution in Visual Studio.
- `TwinCAT.Ads`: ADS library, e.g. for reading and changing the XAR state.
- `System.Xml`: library for parsing XML files.

Selected methods of the class

```
function TcComObject = CreateTcCOM(this, Modelname)
```

Use the MATLAB® help functions in order to view the function and the parameters of the method.

```
>> 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
```

A link to the Beckhoff Infosys is also offered with some methods. These refer to documentation examples from the TwinCAT Automation Interface documentation, so that you can directly view a comparison of the implementation in MATLAB®, C# and PowerShell. You can also find a link to the Beckhoff Infosys in the comment in some sections, allowing you to view the source of the information.

The CreateTcCOM method initially begins with the parsing of the *<modelname>.tmc* file, from which the ClassID, the task cycle time and the task priority are extracted with *System.Xml*. A corresponding TcCOM is then instantiated and one (or more) associated tasks generated with the Automation Interface. Finally, the task is/tasks are assigned to the TcCOM.

```
function ActivateOnDevice(this, AmsNetId)
```

TwinCAT ADS is used in order to query or change the current status of a TwinCAT runtime, e.g. config or run. In the ActivateOnDevice method the XAR is initially switched to the config mode with the specified AmsNetId and the current TwinCAT configuration is then activated and the system started. Pauses are entered between the individual steps, as this procedure may need a little time.


Static methods

Static methods are also available even without an instance of the class.

```
function vsVersions = GetInstalledVisualStudios
```

A function that detects and lists the Visual Studio installations available on the system via the Register Key entries is prepared here. The implementation is limited to VS 2010 to VS 2017.

Documents about this

 AutomationInterfaceMATLAB (Resources/zip/5776206091.zip)

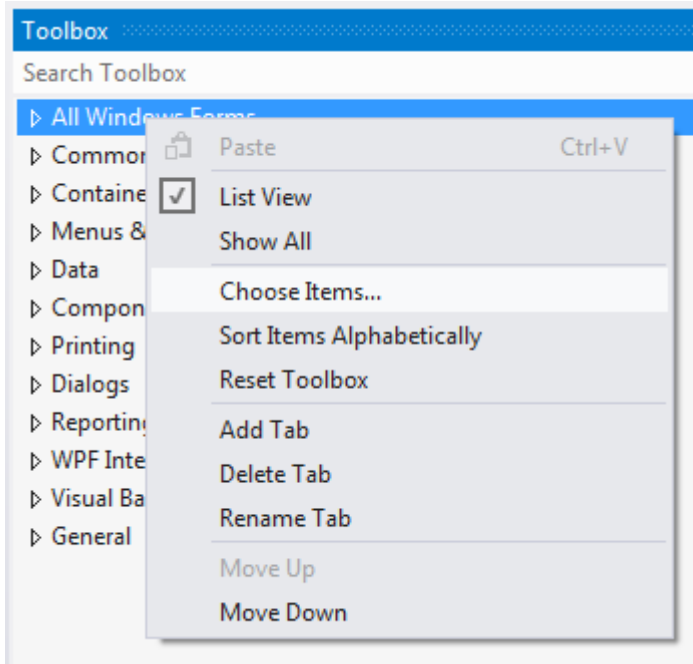
4.10.2 Integrating the block diagram controls

The control that displays the block diagram in the TwinCAT XAE environment can also be integrated as a control in your own visualizations.

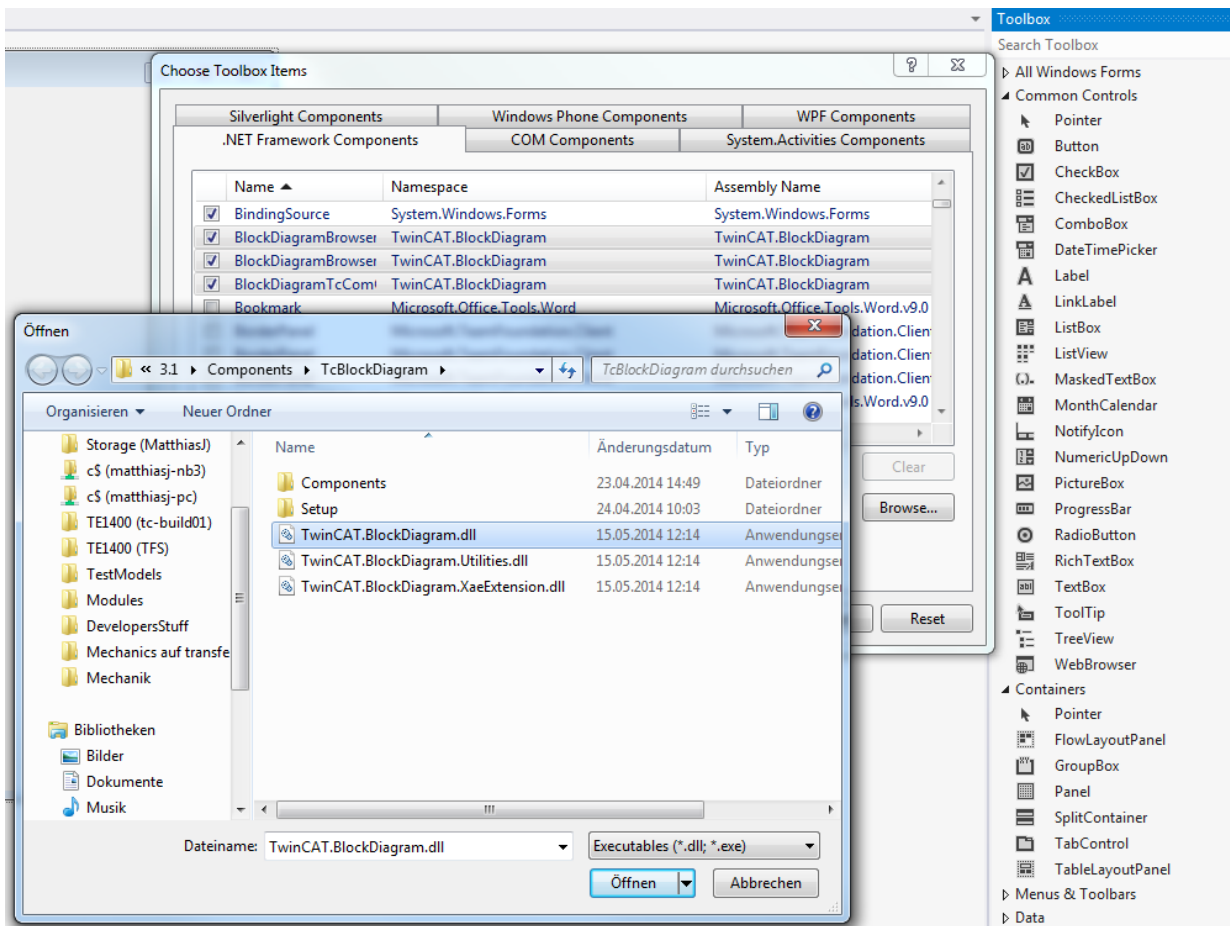
The following steps are required:

1. Create a new Windows Forms application.
2. Add TwinCAT.BlockDiagram.dll to the toolbox:

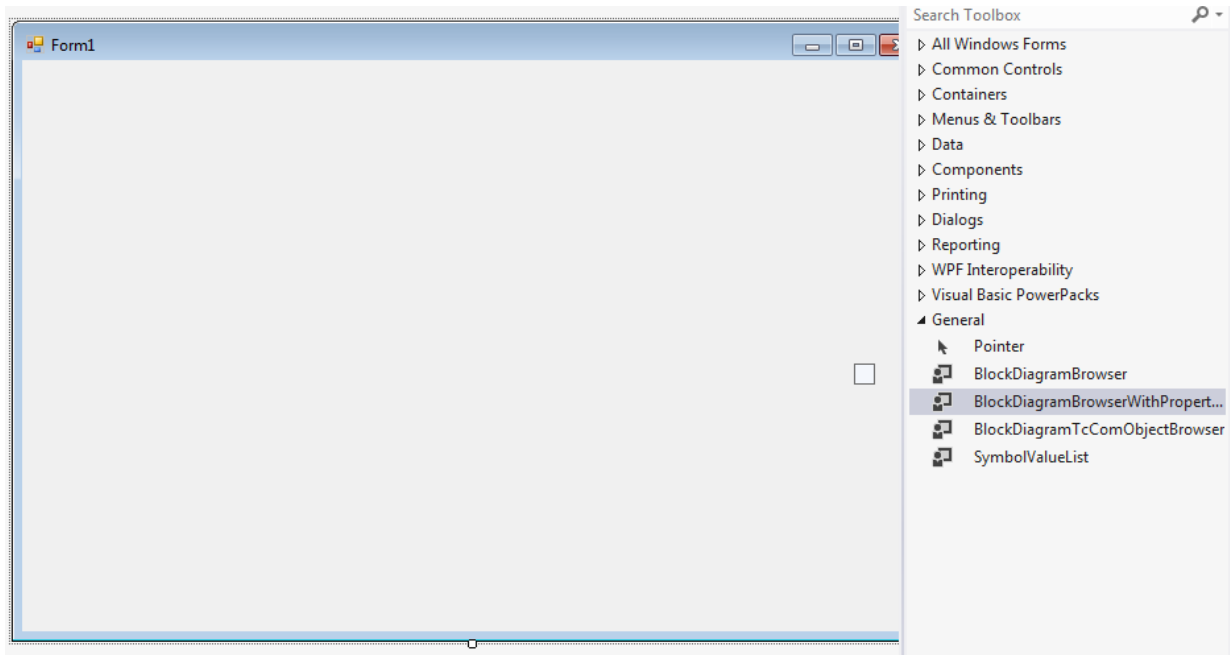
3. To do this, select the "Choose Items..." entry in the context menu.



4. Navigate to TwinCAT.Blockdiagram.dll, which can be found under <TwinCAT installation path>\3.1\Components\TcBlockDiagram.

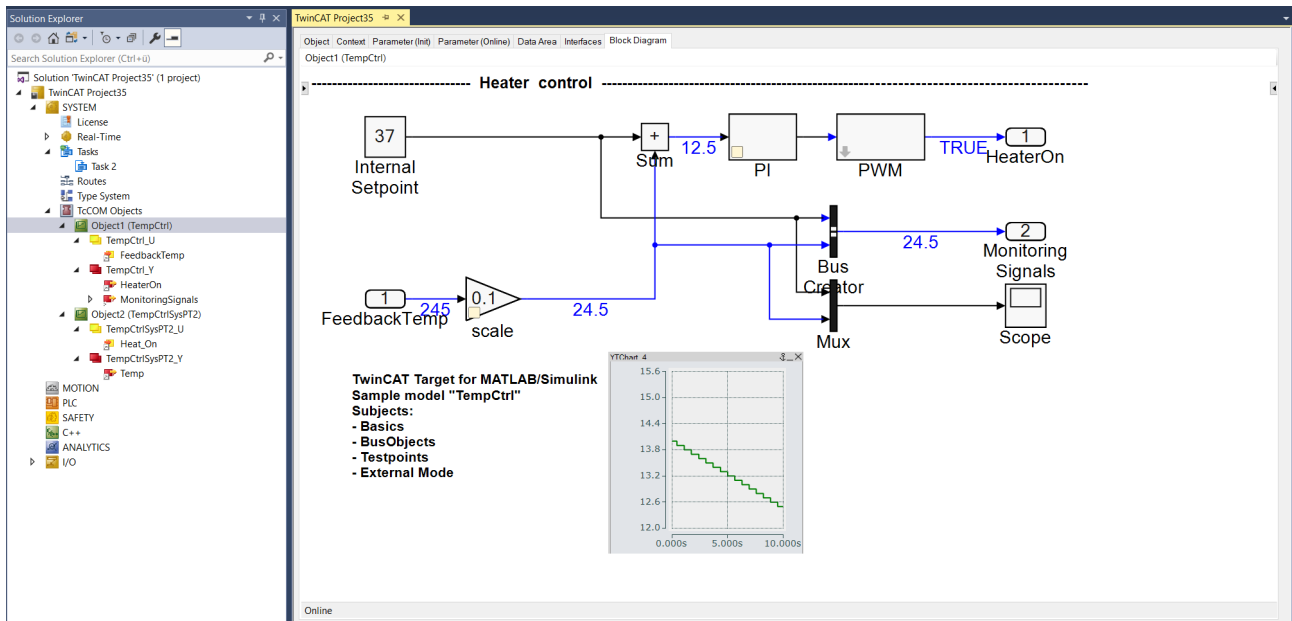


5. Add a TcBlockdiagram control instance to the Windows Forms object using drag and drop.



4.10.3 Try out created TwinCAT objects yourself

- ✓ You don't have MATLAB® or simply want to try out compiled TwinCAT objects with the TwinCAT Target for Simulink®?
Then follow the description below.
1. https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/14632780043/.zip
 2. Unzip the ZIP and execute the tmx archive.
 - ⇒ The zip contains two [tmx-archive \[▶_135\]](#) as executables.
 3. Execute these.
 - ⇒ This unpacks the TwinCAT objects into the correct TwinCAT path (Engineering Repository).
 4. Open XAE and create instances.
 - ⇒ You can now create new TcCOM objects in TwinCAT 3. You can find the two objects under "TE140x Module Vendor" - TE140x - Simulink Modules.
 5. Create the instances and assign a TwinCAT task.
 6. Activate the configuration.
 7. Include the certificate used for signing in the trust list of the target.
 - ⇒ The target system must load the tmx files used. These bear a signature, created with an OEM certificate. The OEM certificate used is probably not yet on the target system in the white list.
 8. [Add the certificate to the white list accordingly \[▶_95\]](#).
 9. Activate the configuration.
 - ⇒ After activating the configuration, you can observe the behavior of the objects and change parameters in the [block diagram \[▶_193\]](#).



More Information:
www.beckhoff.com/te1400

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

