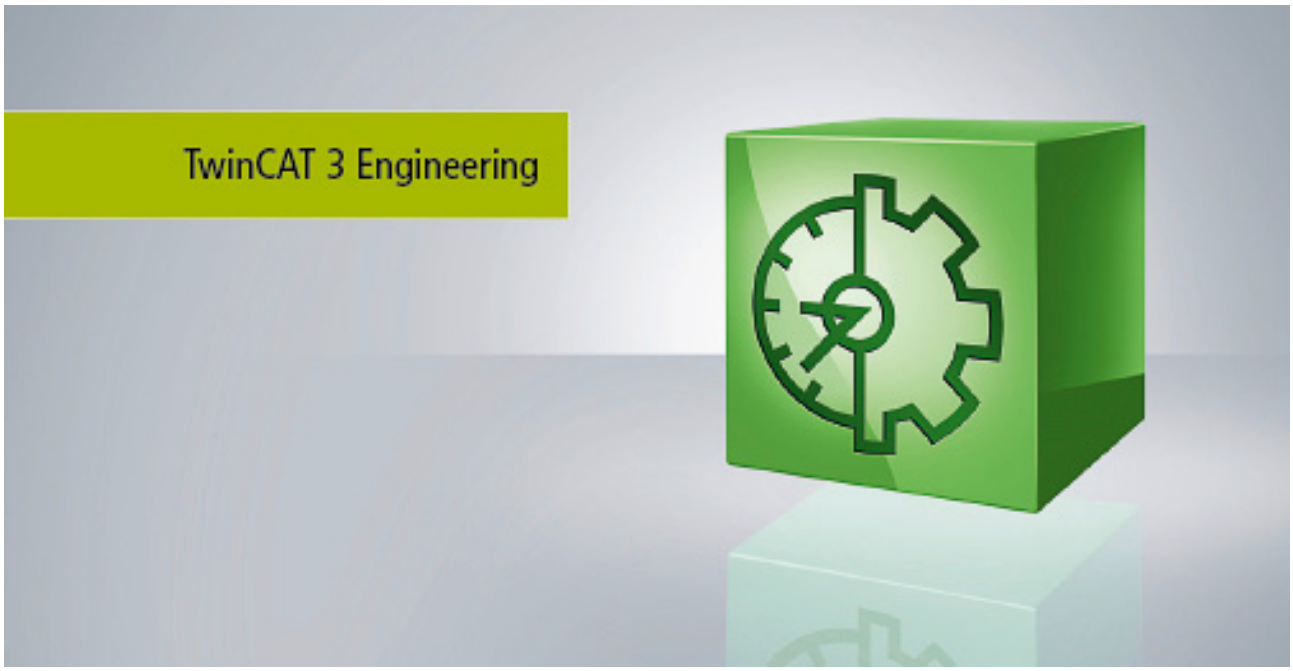


Manual

TC3 Target for Matlab®/Simulink®

TwinCAT 3

Version: 1.3
Date: 2018-09-11
Order No.: TE1400

BECKHOFF

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with the applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the 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 prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® and XTS® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, DE102004044764, DE102007017835
with corresponding applications or registrations in various other countries.

The TwinCAT Technology is covered, including but not limited to the following patent applications and patents:
EP0851348, US6167425 with corresponding applications or registrations in various other countries.

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

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations 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 engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

2 Overview

TE1400 TwinCAT Target for MATLAB®/Simulink®

The Simulink Coder® (previously called Real-Time Workshop®) for the MATLAB®/Simulink® environment contains a code generator, which can generate corresponding C/C++ code from the model in Simulink®. The TwinCAT Target for MATLAB®/Simulink® works on the basis of this code generator. If the code generator is configured appropriately with TwinCAT Target, a TwinCAT Object Model (TcCOM) is created with the input and output behavior of the Simulink model. This module class can be instantiated in the TwinCAT 3 development environment (TC3 XAE). The module instance can be reconfigured in the TC3 XAE, if required. Once the TC3 runtime has started, the module is executed in real-time and can therefore be integrated into an actual machine control system.

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 (software-in-the-loop) simulation
- HiL (hardware in the loop) simulation
- 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-loop control circuit to be designed and tested in a simulation (model-in-the-loop 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 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 TE1111 EtherCAT Simulation.
- **Example 2a: Software-in-the-loop simulation of system components.** According to VDI/VDE 3693 Part 1, software-in-the-loop is defined as a stage following an 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: Hardware-in-the-Loop simulation of system components.** According to VDI/VDE 3693 Part 1, hardware-in-the-loop 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 TE1111 EtherCAT Simulation 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. Such parameters can nevertheless be determined by using a physically representative model with input variables that are easier to determine. 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 velocity and cooling temperature.

Webinars concerning TE1400 and TE1410

Date	Topic	Referent
01.10.2013	TwinCAT 3 | Matlab®/Simulink®-integration: Introduction, application examples, TC3 Interface for Matlab®/Simulink® (german)	Dr. Knut Güttel

Overview of the Beckhoff webinars (german)

3 Installation

System requirements

The target for MATLAB®/Simulink® initially meets the same requirements 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 TC3 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 or 2015 Professional, Premium, Ultimate or Community Edition
  - Installation under Windows 7 always with right-click **run as admin…**
  - For Visual Studio 2015 check the Visual C++ checkbox during installation
- Microsoft "Windows Driver Kit" version 7.1.0
  - It is sufficient to install the "Build Environments".
  - Set the environment variable (variable name WINDDK7, variable value <Installation directory> e.g. C:\WinDDDK\7600.16385.1)
- TwinCAT 3 XAE

On the runtime PC

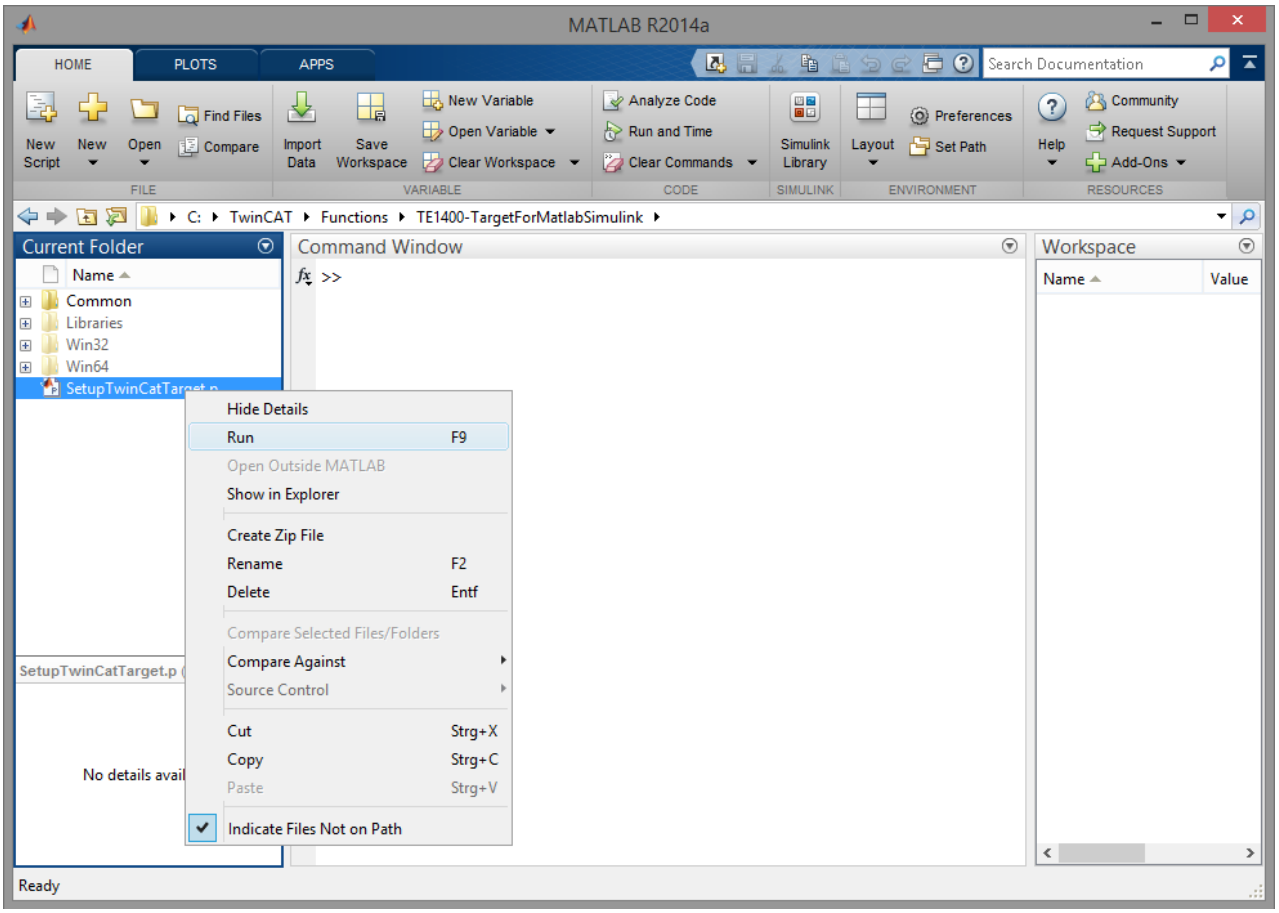
- IPC or Embedded CX PC with Microsoft operating system based on Windows NT kernel (Win XP, Win 7 and corresponding embedded versions)
- TwinCAT 3 XAR
  - TwinCAT 3.0 only supports 32-bit operating systems on the target
  - TwinCAT 3.1 supports 32-bit and 64-bit operating systems. If the target is a x64 system, the created drivers must be signed. See "x64: driver signing" in the TC3 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® R2010a or later version. R2010b or later version is recommended, since R2010a requires a patch to support the Microsoft VC++ 2010 compiler.
- Simulink coder® (in MATLAB® versions prior to R2011a: Real-Time Workshop®)
- MATLAB Coder® (in MATLAB® versions prior to R2011a: part of Real-Time Workshop®)
- Installing of the TE1400 Target for MATLAB®/Simulink®

Setup instructions

1. Install one of the supported Visual Studio versions, if not already installed.
2. 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 (reinstall Visual C++, if necessary). Then run TwinCAT 3 Setup to integrate TwinCAT 3 into the new (or modified) Visual Studio version.
3. 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.
4. If you do not have a MATLAB installation on your system, install it.
   The order in which MATLAB was installed is irrelevant.
5. Then start the TE1400 TargetForMatlabSimulink Setup 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.
6. Start MATLAB as administrator and execute %TwinCAT3Dir%..\Functions\TE1400-TargetForMatlabSimulink\SetupTwinCatTarget.p in MATLAB.



- 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.

User Account Control

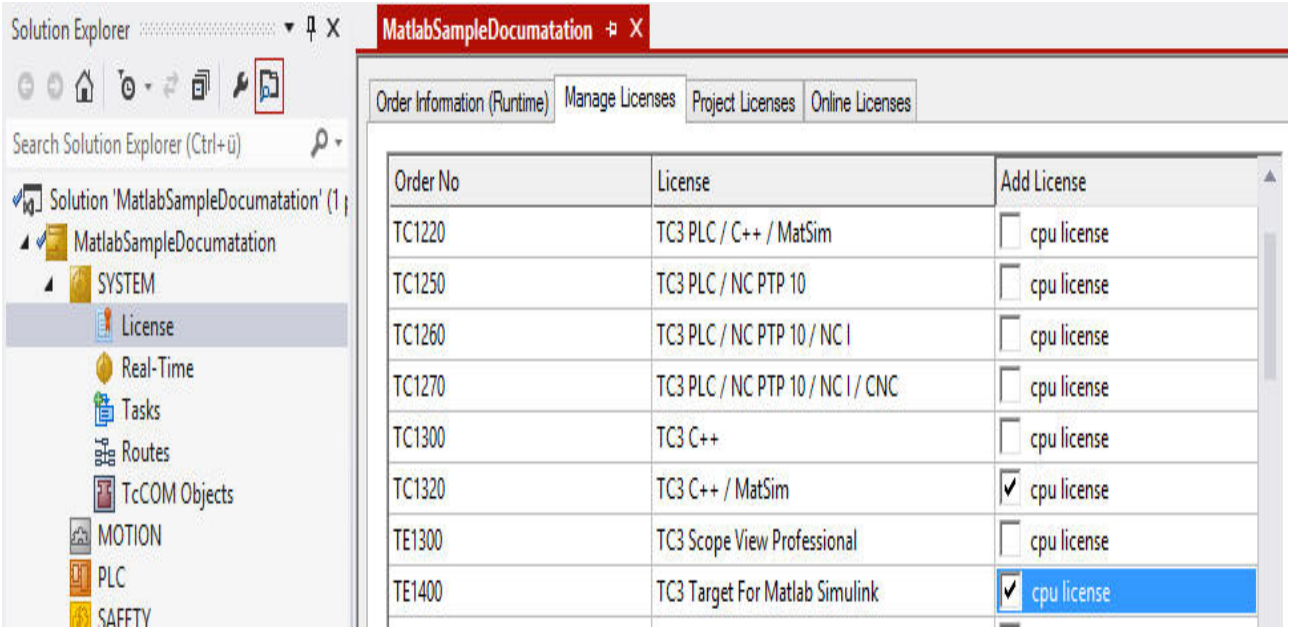
If MATLAB is executed in a system with activated User Account Control (UAC) without administrator authority, the MATLAB path cannot be stored permanently. In this case, SetupTwinCatTarget.p must be executed every time MATLAB is started, since otherwise some files required for generating TwinCAT modules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

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 x64: driver signing.

4 Licenses

Two licenses are required, if you want to use the full functionality of the TE1400 Target for MATLAB®/ Simulink®. You can activate the licenses as described in "Ordering and activation of TwinCAT 3 standard licenses”



required licenses for the 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 demo mode

You can run the module generator without any license with the following limitations relating to the complexity of the source Simulink model. Allowed are models with a maximum of:

- 100 blocks
- 5 input signals
- 5 output signals

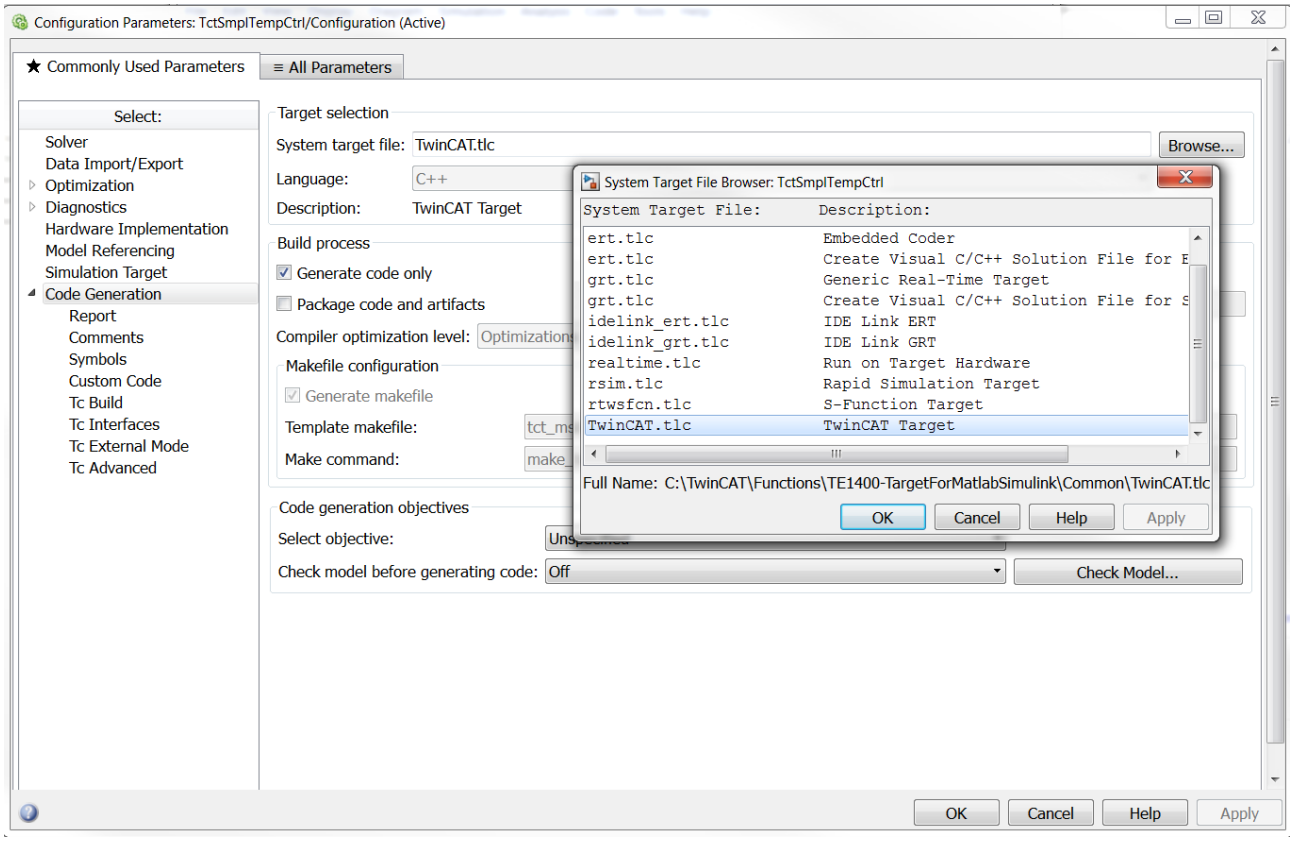
Modules created without the TE1400 license are only allowed to be used for non-commercial purposes!

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.

5 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.

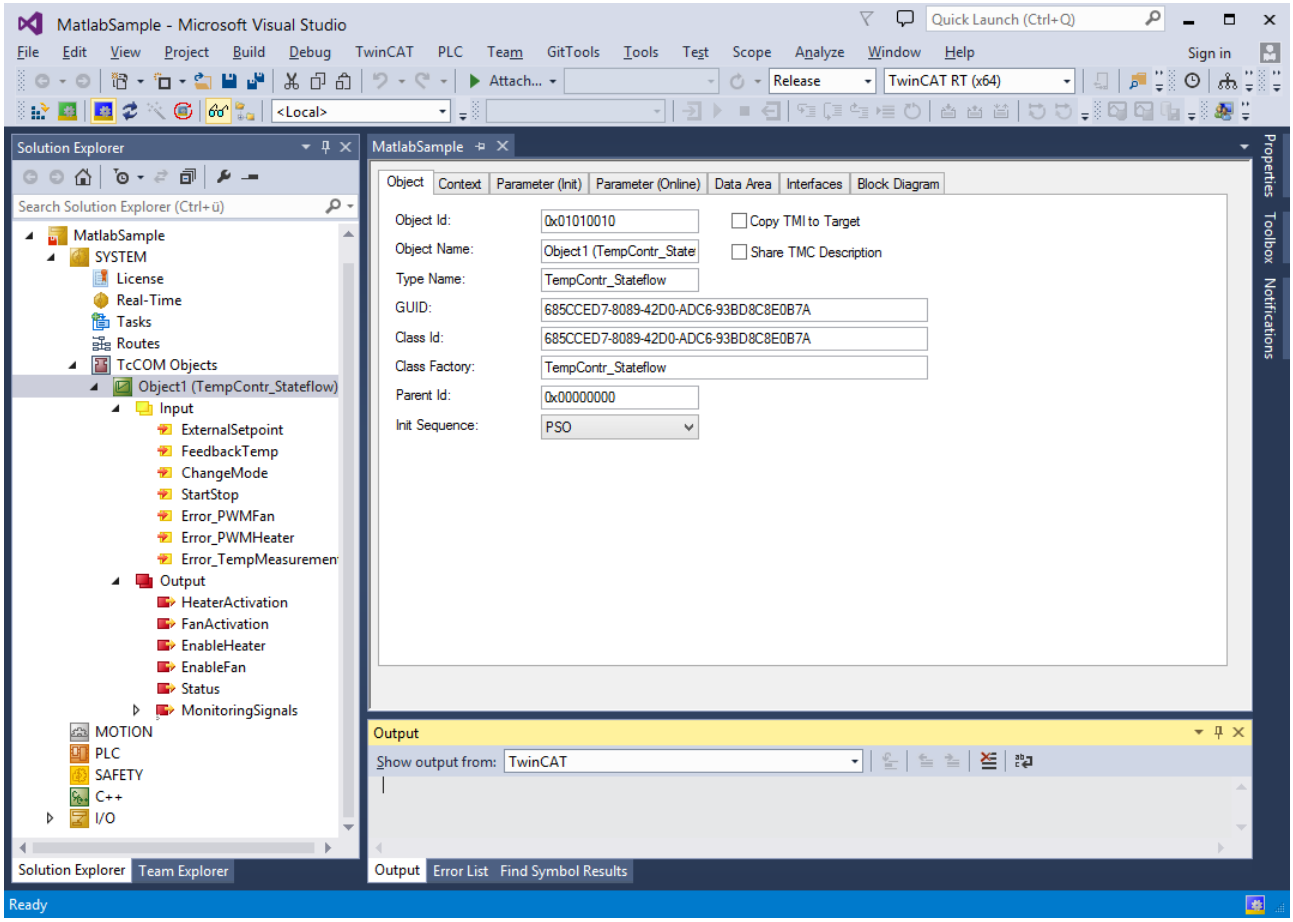
Generating a TcCOM module from Simulink

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 [▶ 20].

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.

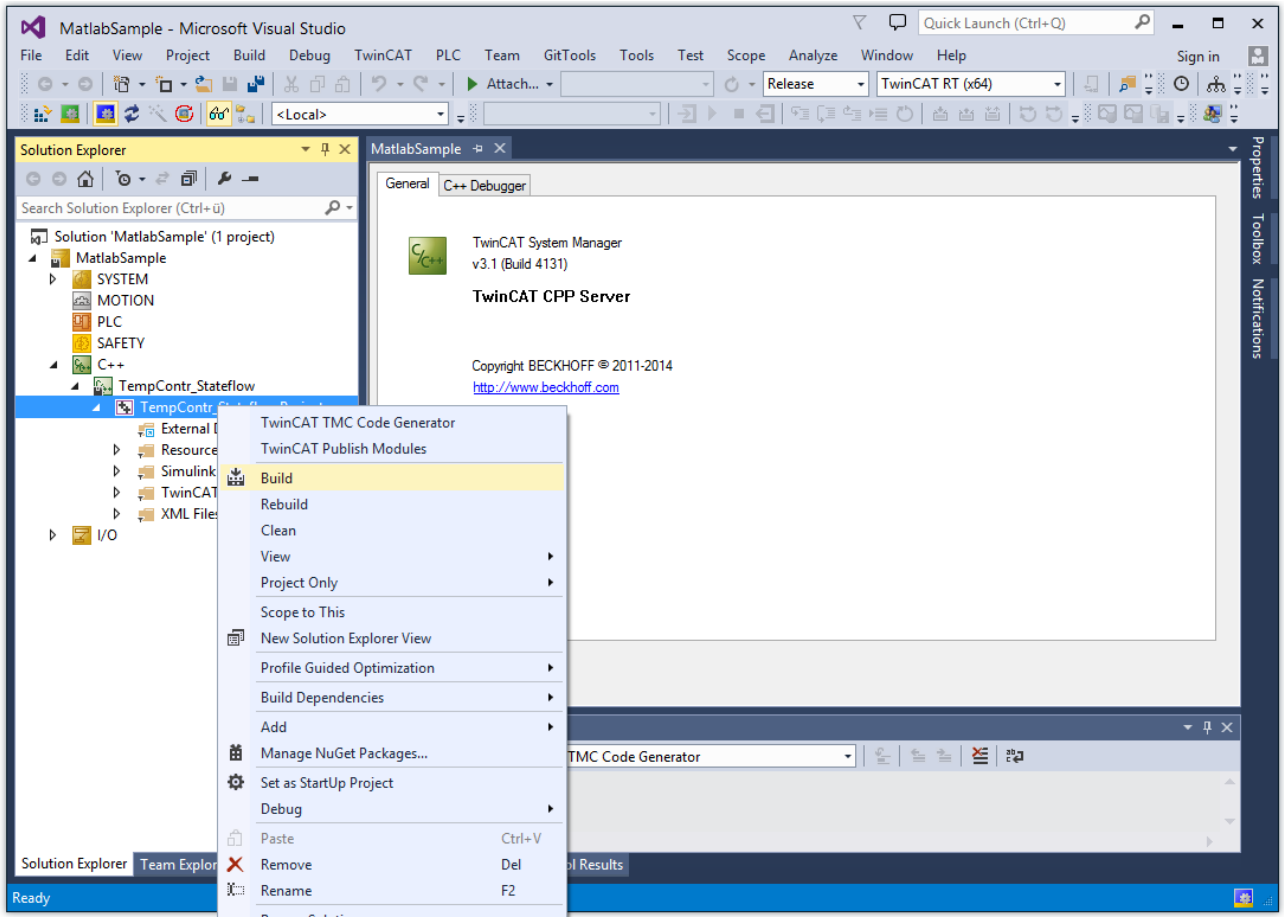


Instances of the generated module can be integrated in a TwinCAT3 project any number of times. TcCOM objects are usually appended to the node **TcCOM Objects** via the **Add New Item** context menu. A selection list of the modules that are available on the system can be obtained via this option. The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

Compiling the code without "Publish"

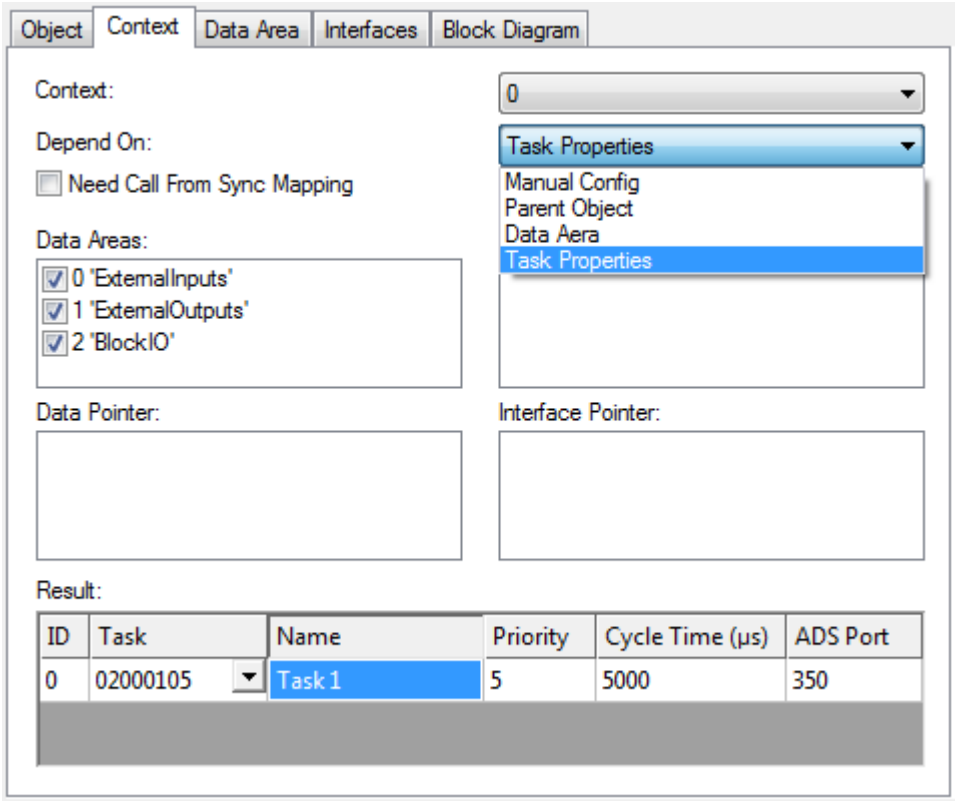
If the option **Publish Module** was disabled prior to the module generation, the C/C++ code pertaining to the module still has to be compiled, before it can be executed.

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



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 [▶ 40]".

Data exchange with other modules or fieldbus devices

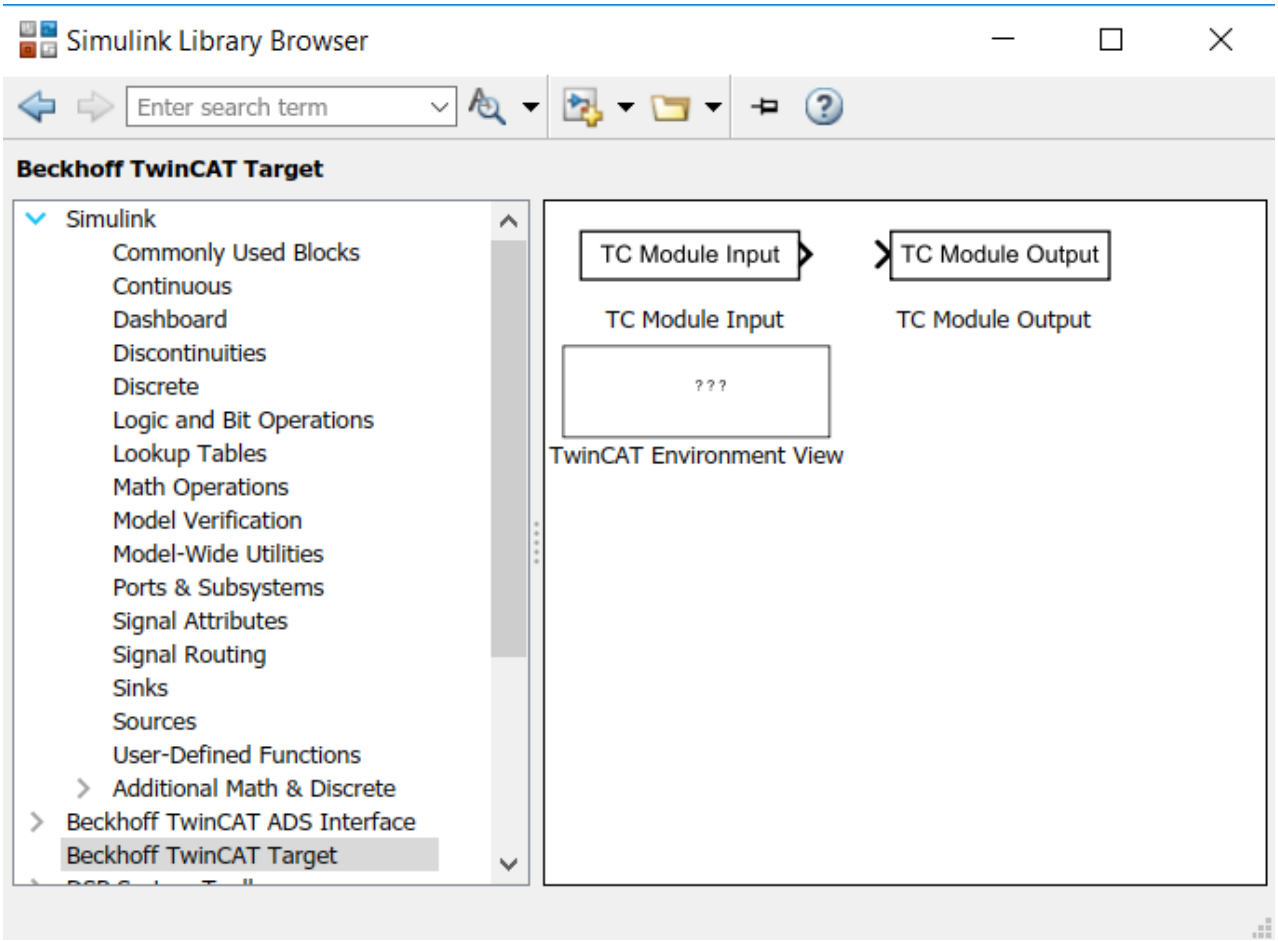
The process images of the module inputs and outputs can be expanded below the module instance node in the TwinCAT development environment. Here you will find all ports that have been defined in the Simulink model with the aid of the function blocks **In1** and **Out1** (components of the standard Simulink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

6 TwinCAT library in Simulink

In Simulink®, TwinCAT-specific input and output function blocks can (but do not have to) be used to define the signals/busses 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:

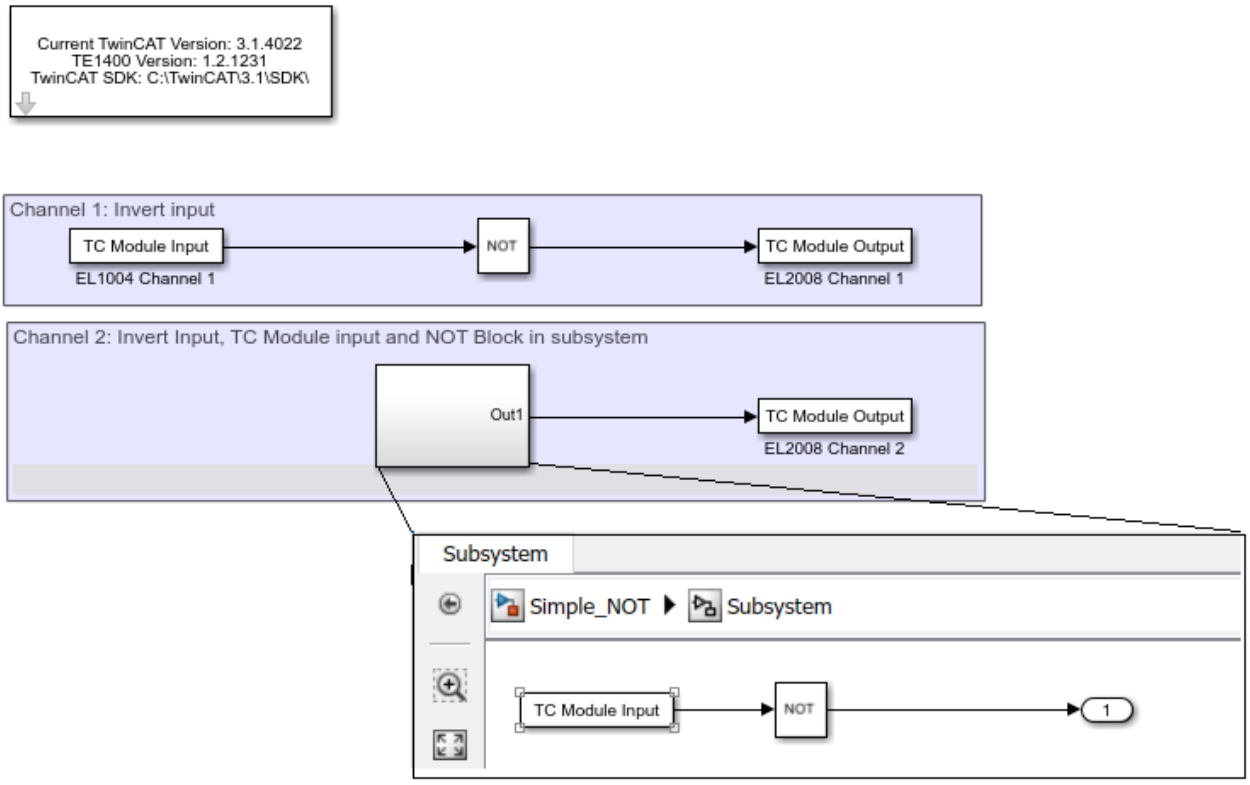
- You can also define signals and busses from subsystems directly as inputs or outputs for TcCOM, without first transferring the signals/busses from the subsystem to the top system.
- You can (but do not have to) store an automatic mapping to other TcCOM or I/Os in the function block parameters, so that mapping is executed directly when the TcCOM is instantiated.

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.

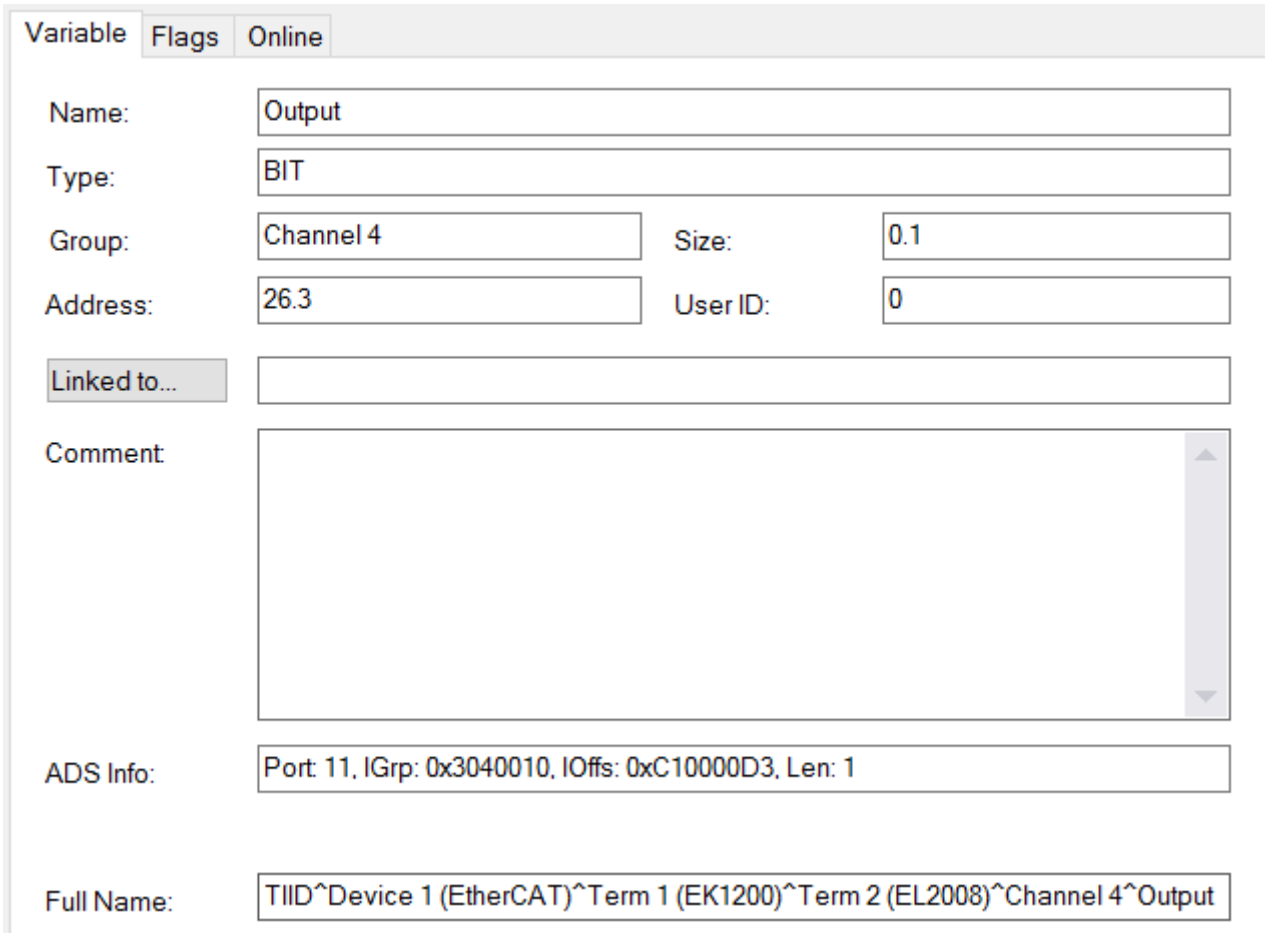
In addition to TwinCAT-specific input and output function blocks, a TwinCAT Environment View Block is also provided. This can be used in the Simulink environment to simply display TwinCAT and TE1400 versions on the system.

Example

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

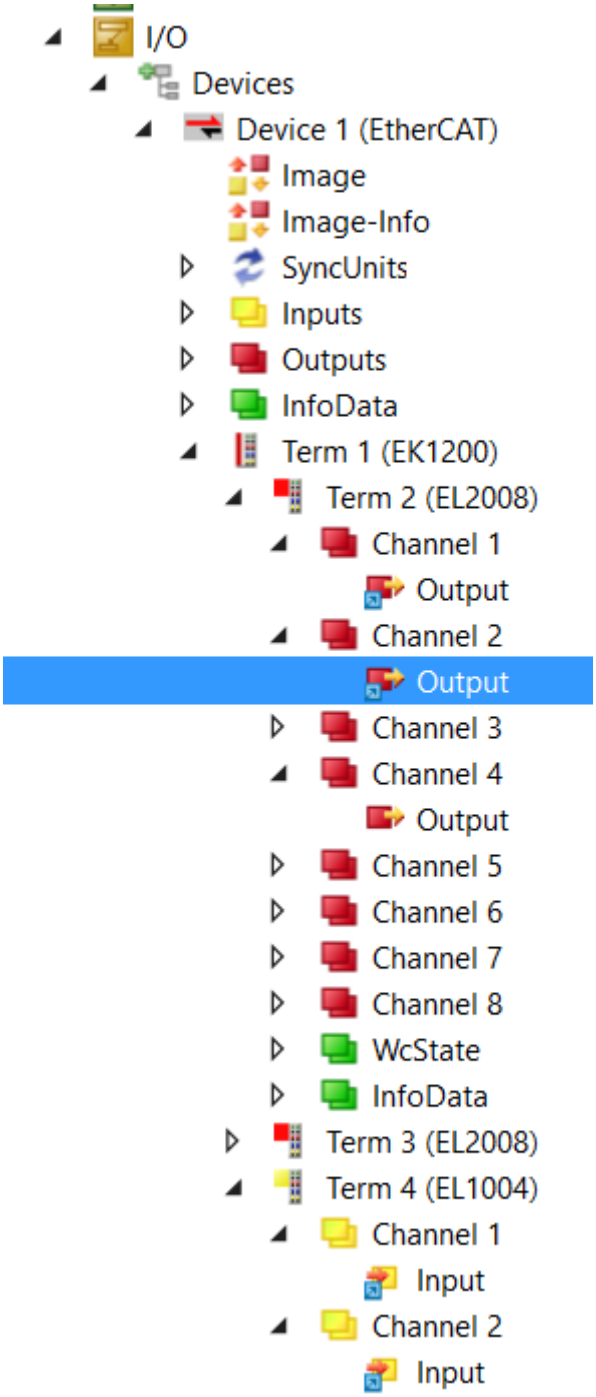


The inputs and outputs of the model are automatically mapped to digital inputs and outputs via the properties of the TC Modules Input and Output. The necessary tree items can be found in TwinCAT 3 by selecting the desired input or output and then copying the string in the **Variable** tab under **Full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



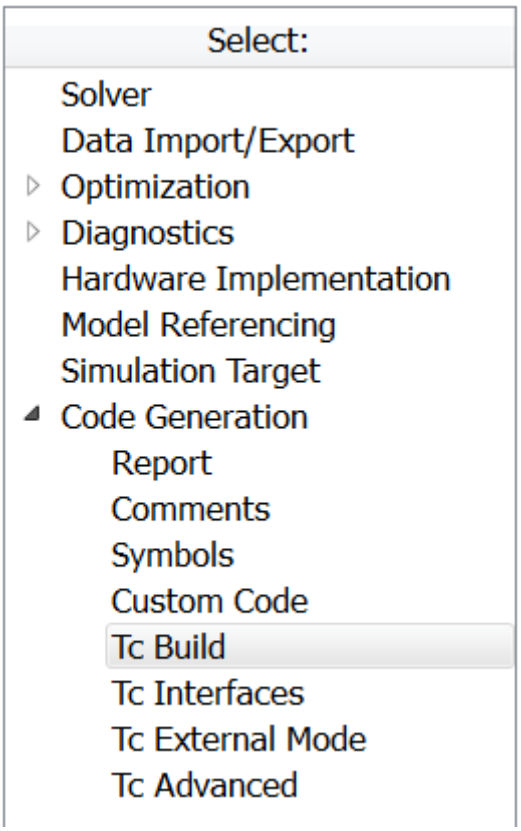
A list of shortcuts for quick access can be found in the documentation of the Automation Interface > API > ITcSysManager > ITcSysmanager::LookupTreeItem.

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.


I/O
Devices
Device 1 (EtherCAT)
Image
Image-Info
SyncUnits
Inputs
Outputs
InfoData
Term 1 (EK1200)
Term 2 (EL2008)
Channel 1
Output
Channel 2
Output
Channel 3
Channel 4
Output
Channel 5
Channel 6
Channel 7
Channel 8
WcState
InfoData
Term 3 (EL2008)
Term 4 (EL1004)
Channel 1
Input
Channel 2
Input

7 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.

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).

7.1 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

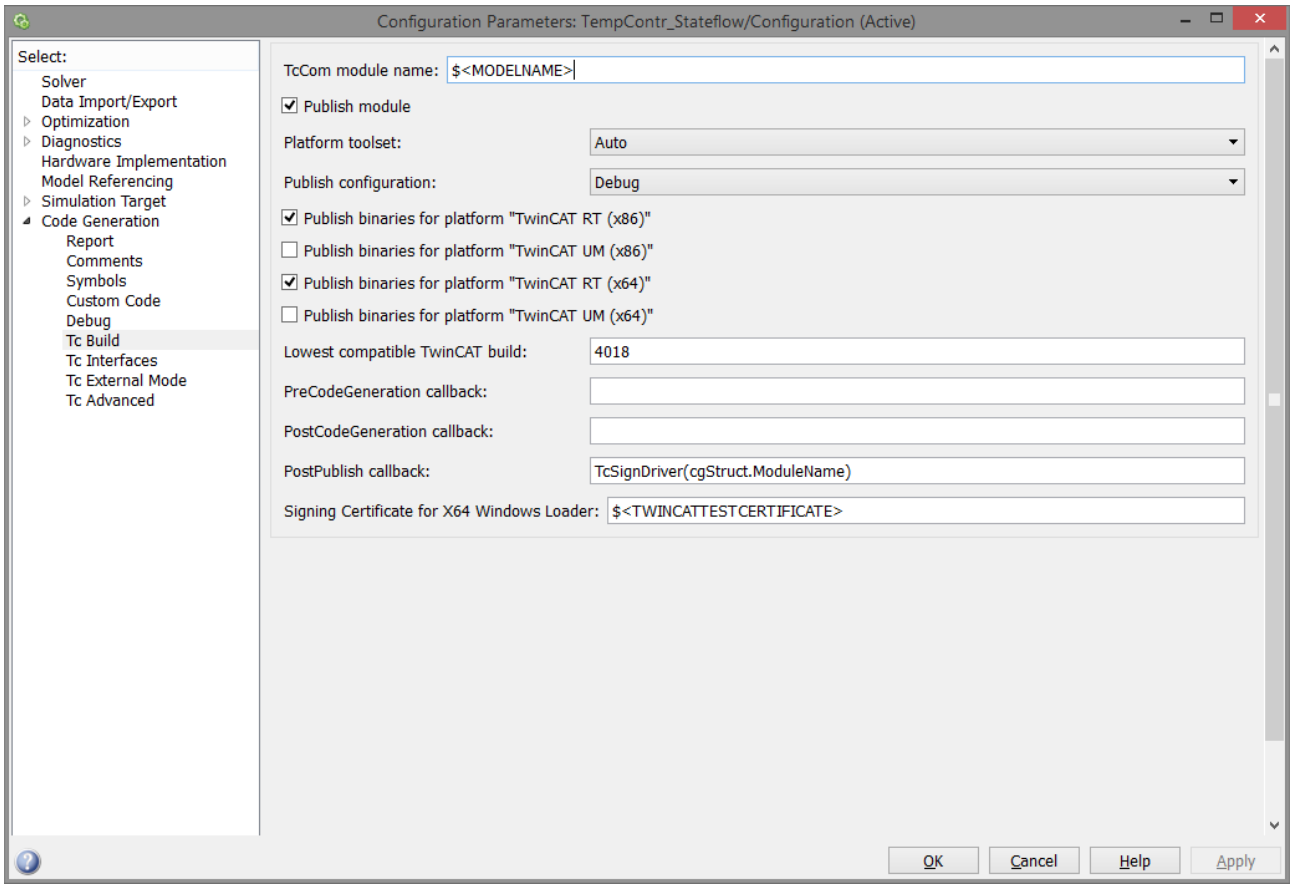
The files relating to exported modules are copied to the directory *%TwinCat3Dir%CustomConfig\Modules \<MODULENAME>*. To instantiate the module on another development PC, this folder can be copied to the appropriate directory on the other computer.

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 under **Tc Build**: (Export options for TwinCAT modules)



Publish module:

- **deactivated**: 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.
- **activated**: "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.

"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:

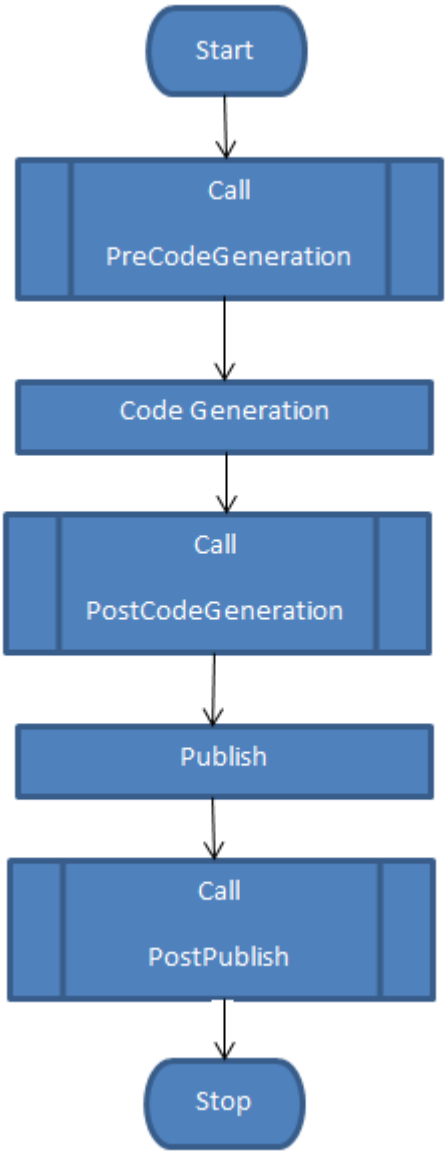
Select **Debug** here in order to enable debugging of the exported block diagram in TwinCAT 3. If no debugging is required, e.g. in a release version, **Release** can be selected here.

- **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 uncompilable, 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 a previously exported via "Publish", only the TMC description is imported into the TwinCAT project. The module instance still refers to files within the Publish directory [▶ 20]. To load the TwinCAT project on other development computer, the Publish directories of the modules in use have to be copied manually into to 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 Publish directory [▶ 20] 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: Examples for : [▶ 68]

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 "x64: 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.

7.2 Data exchange (Tc Interfaces)

Configuration in Simulink

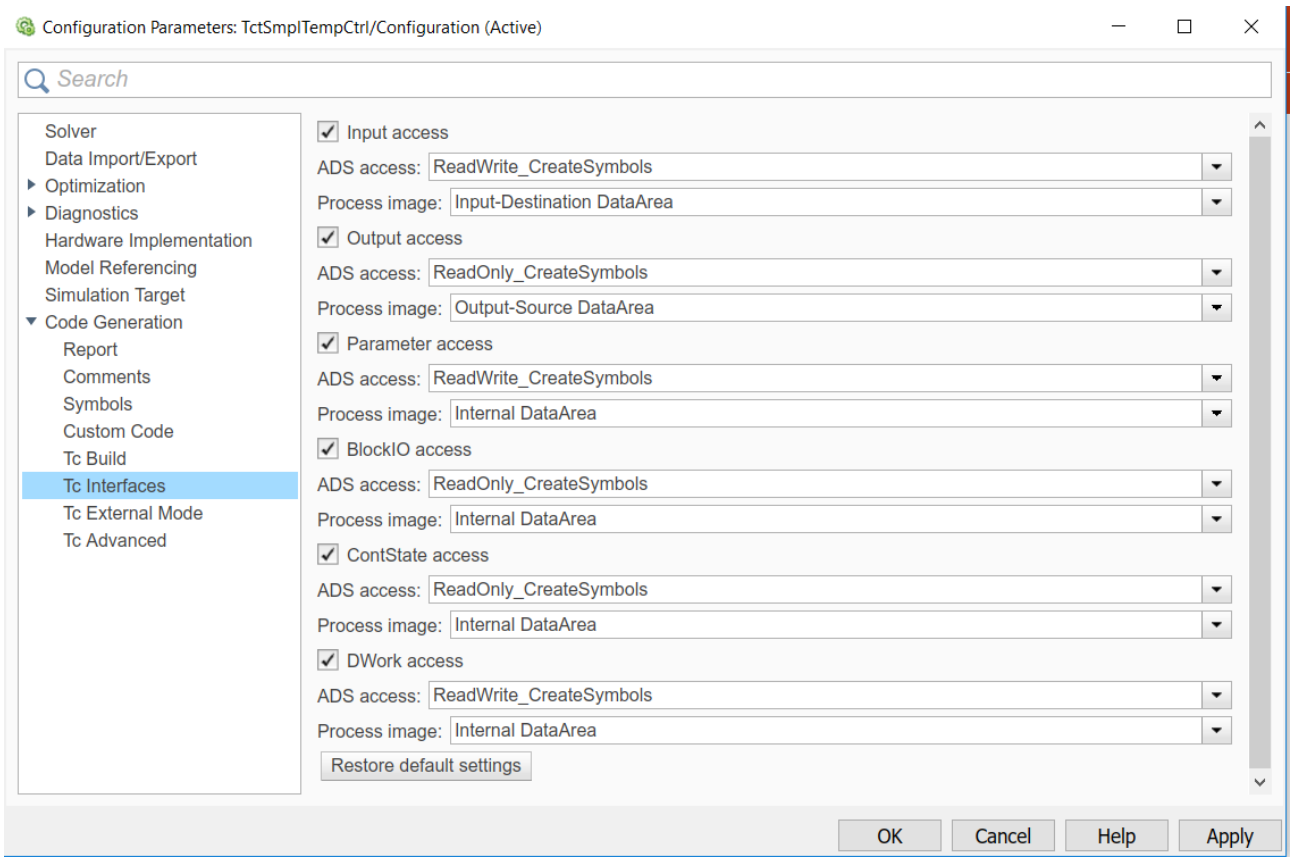
Depending on the Simulink model, there are several groups of internal variables in addition to the input and output variables. ADS access and the process image type can be configured as required. These settings affect how the variables are linked with other process images in the TwinCAT development environment, 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	
GROUP 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 "*GROUP* 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
	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 [▶ 26]) 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.

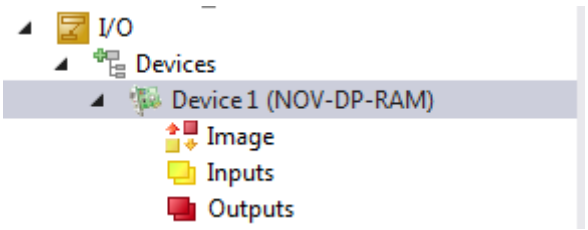
7.2.1 Retain data

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 following section describes the retain handler, which stores data and makes them available again, and the application of the different TwinCAT 3 programming languages.

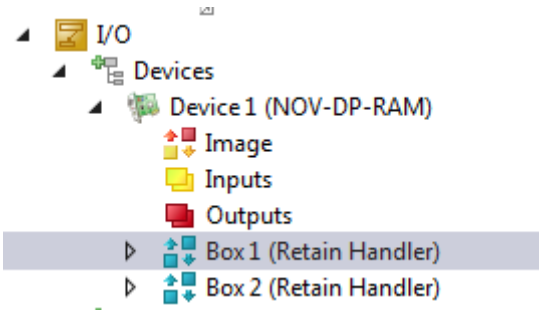
Configuring a retain device

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.

A NOV-RAM DP device is created in the IO section of the solution for this purpose.

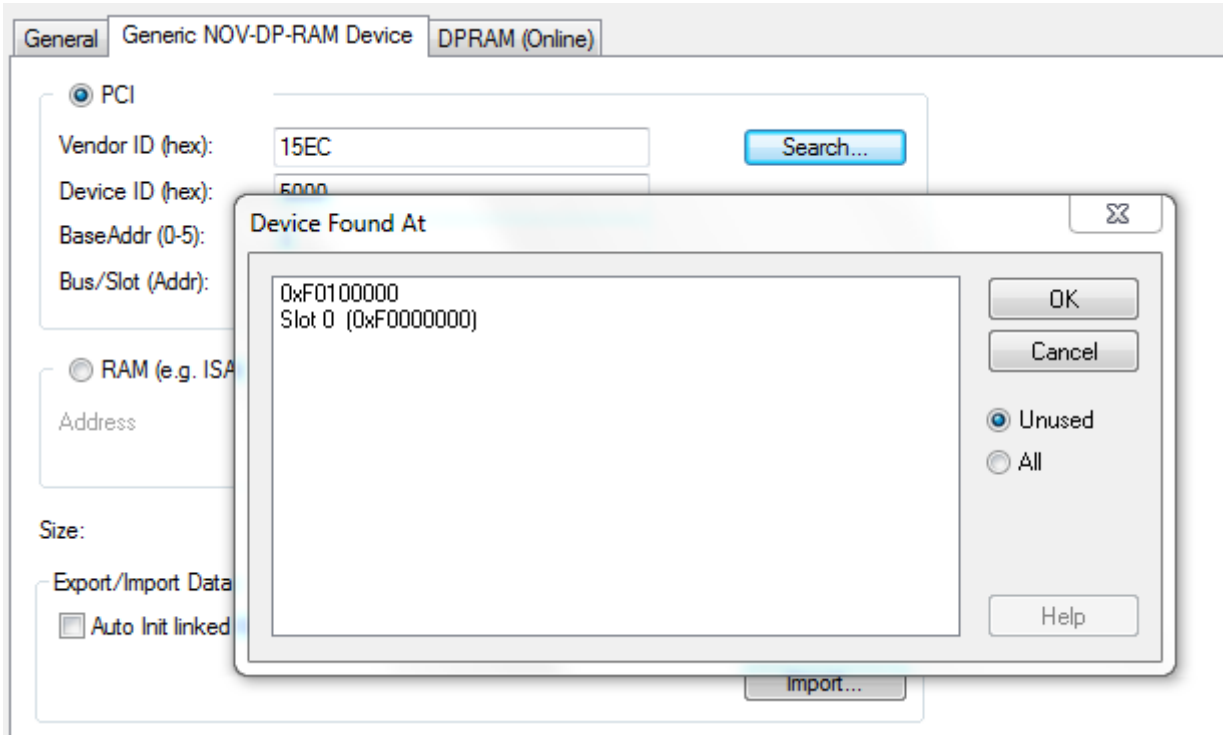


One or several retain handlers can be created under this device



Storage location: NOVRAM

The NOV-DP-RAM device must be configured. The range to be used can be defined in the "Generic NOV-DP-RAM device" tab via "Search…".



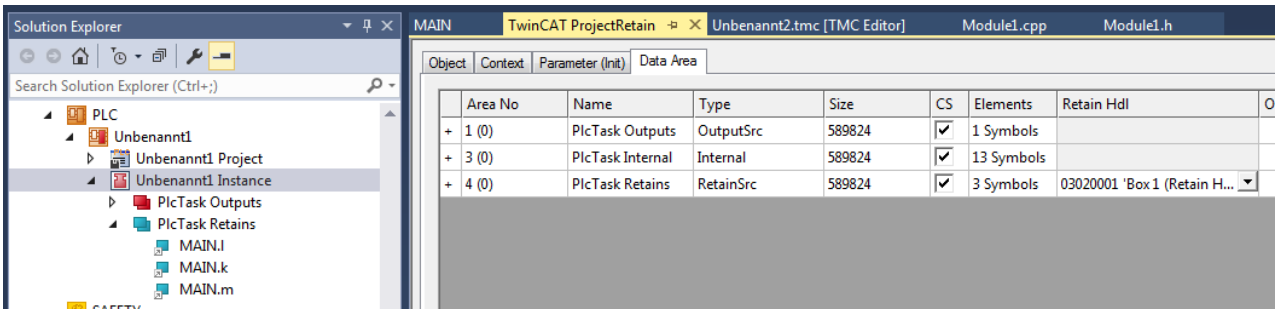
An additional retain directory for the symbols is created in the TwinCAT boot directory.

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 AT %Q* : UINT;
END_VAR
VAR
    {attribute 'TcRetain':='1'}
    m: UINT;
    x: UINT;
END_VAR
```

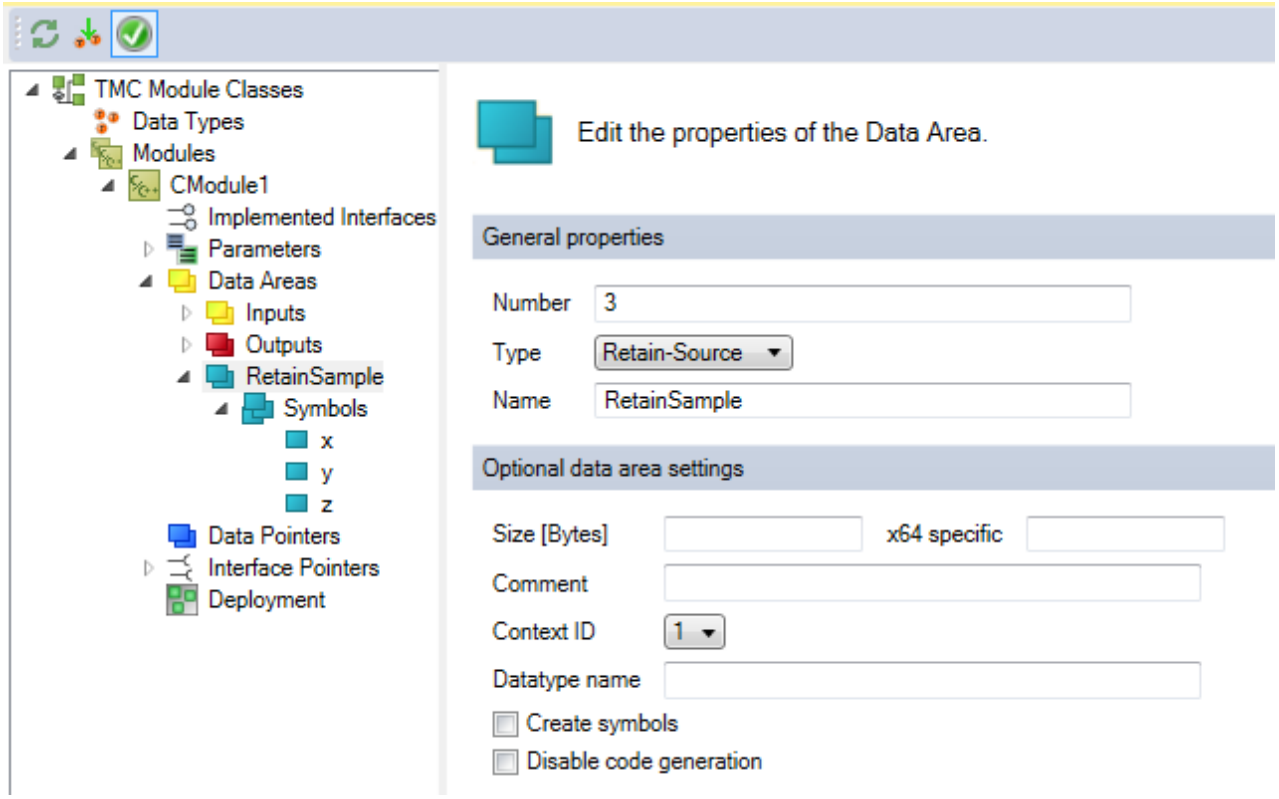
Corresponding symbols are created after a "Build".
The assignment to the retain handler of the NOV-DP-RAM device is done in column "Retain Hdl".



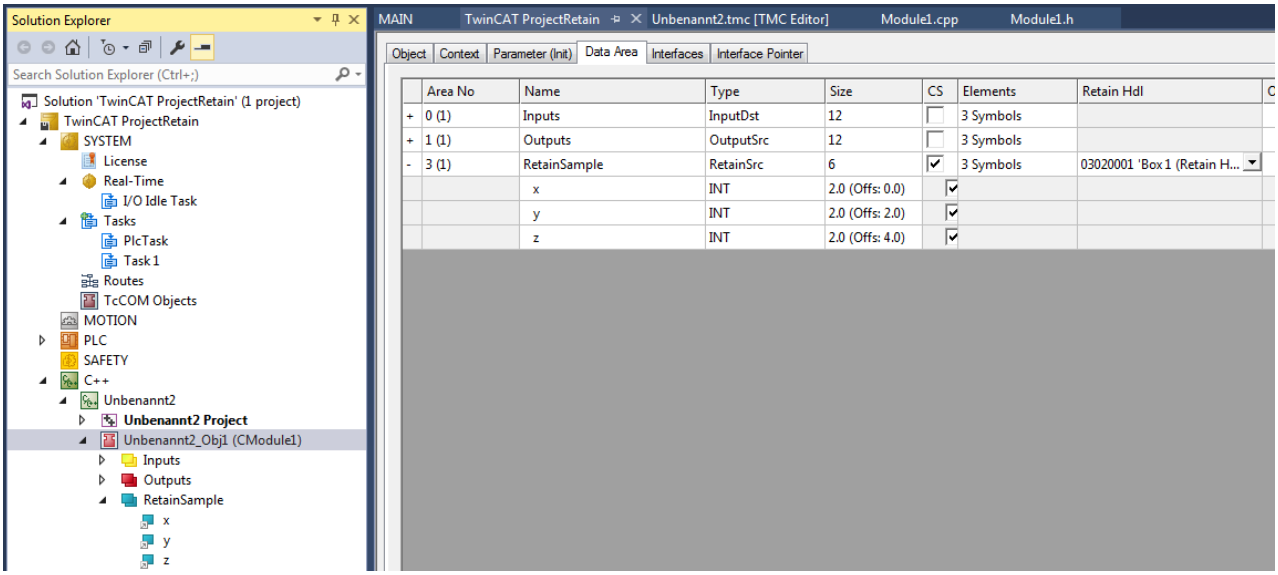
If self-generated data types (DUTs) are used as retains, the data types must exist in the TwinCAT type system. You can either use the option "Convert to Global Type" or you can create structures directly as `STRUCT RETAIN`, which will handle all occurrences of the structure via the retain handler.

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.

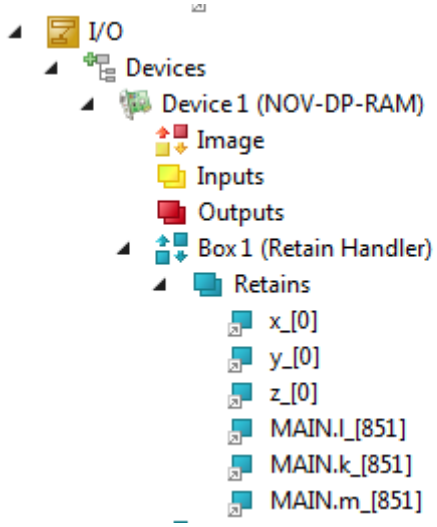


At the instances of the C++ module, a retain handler of the NOV-DP-RAM device to be used for this data area is defined in column "Retain Hdl".



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".

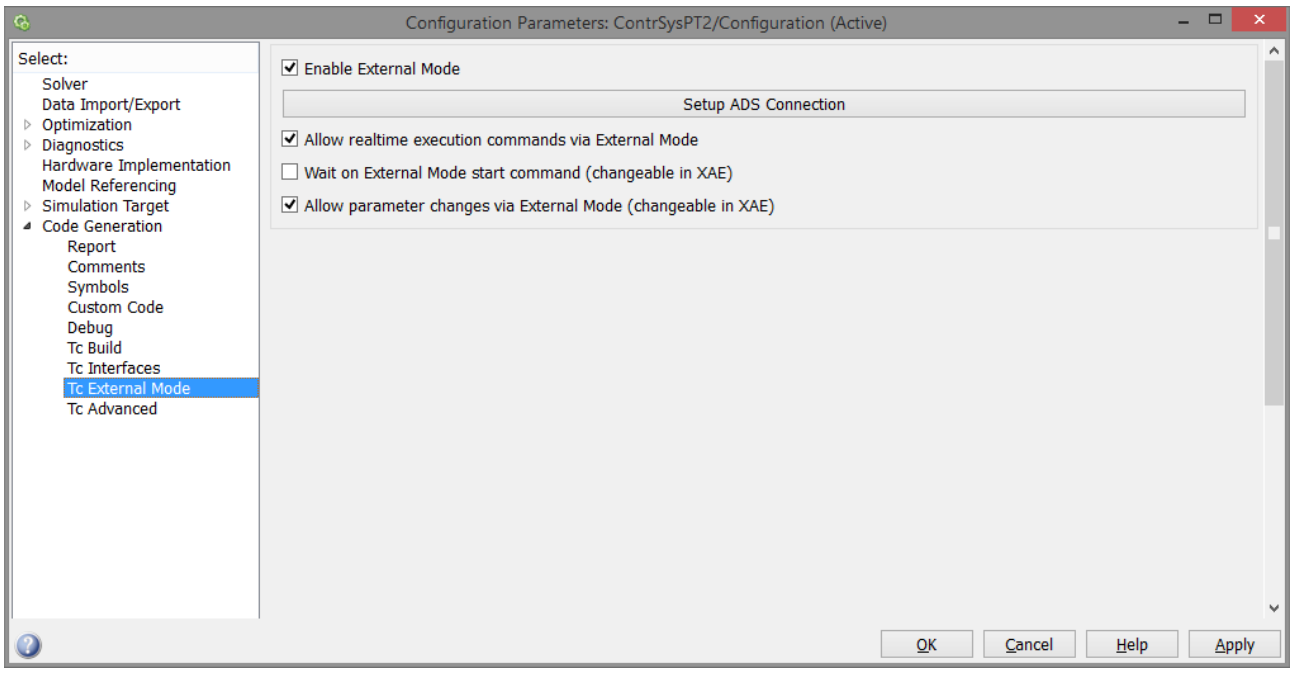


7.3 External mode (Tc External Mode)

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 module parameter [▶ 30] "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 module parameter [▶ 30] "WaitForStartCommand"	FALSE

Module parameter

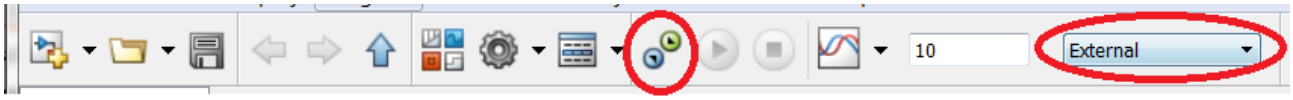
For configuring the behavior in **external mode** (on the XAE side) the parameter External Mode is defined as a structure in generated modules, which contains the following elements:

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.	
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)	

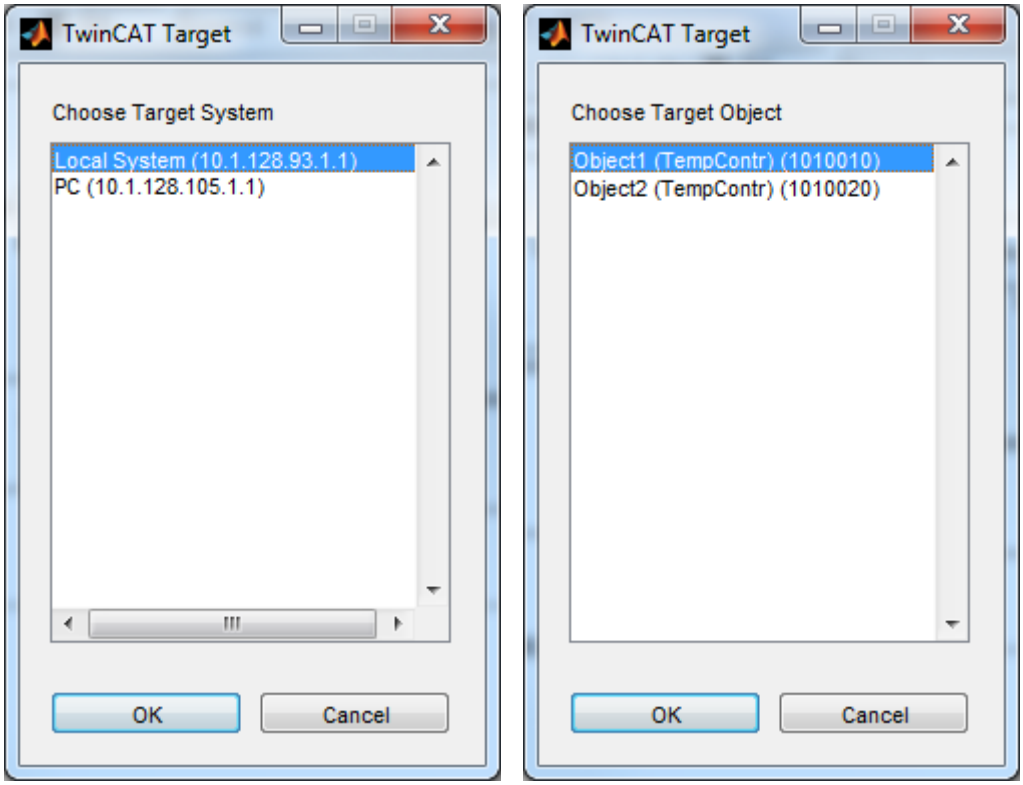
For further information on adapting the module parameters in XAE see section "Parameterization of the generated module".

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:



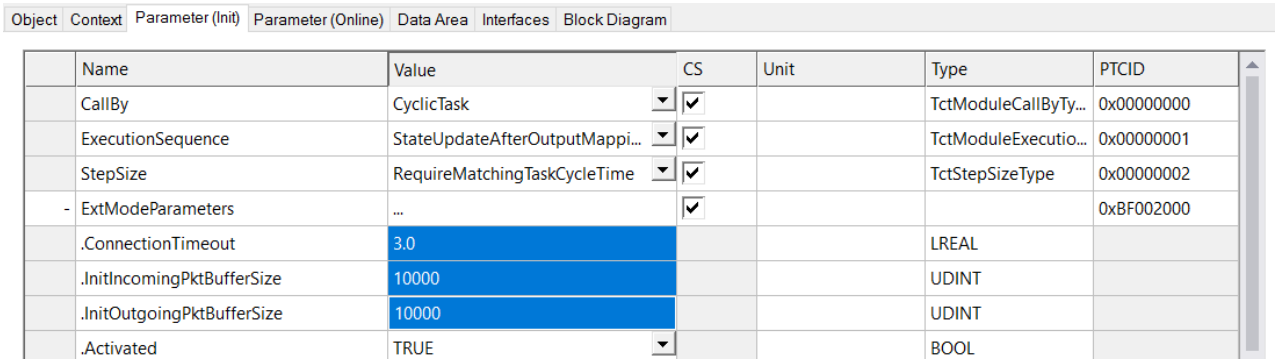
The first dialog box shows a list of target systems, the second box shows a list of the available module instances on the selected target system. In the following dialog the user can specify whether the new connection data should be stored. Once the connection data have been stored, the connection is 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" [▶ 63].

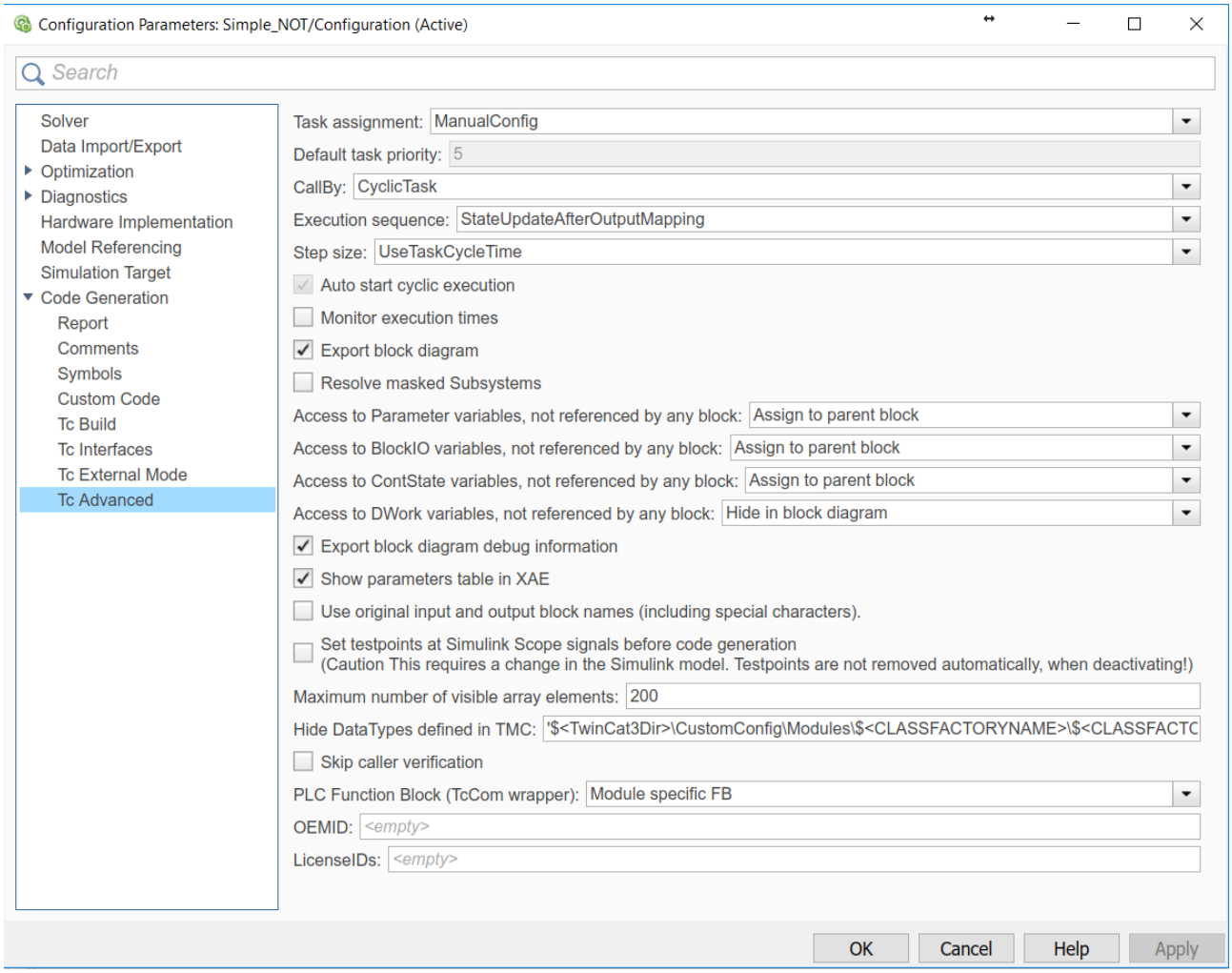
Parameterization in TwinCAT

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).



7.4 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:

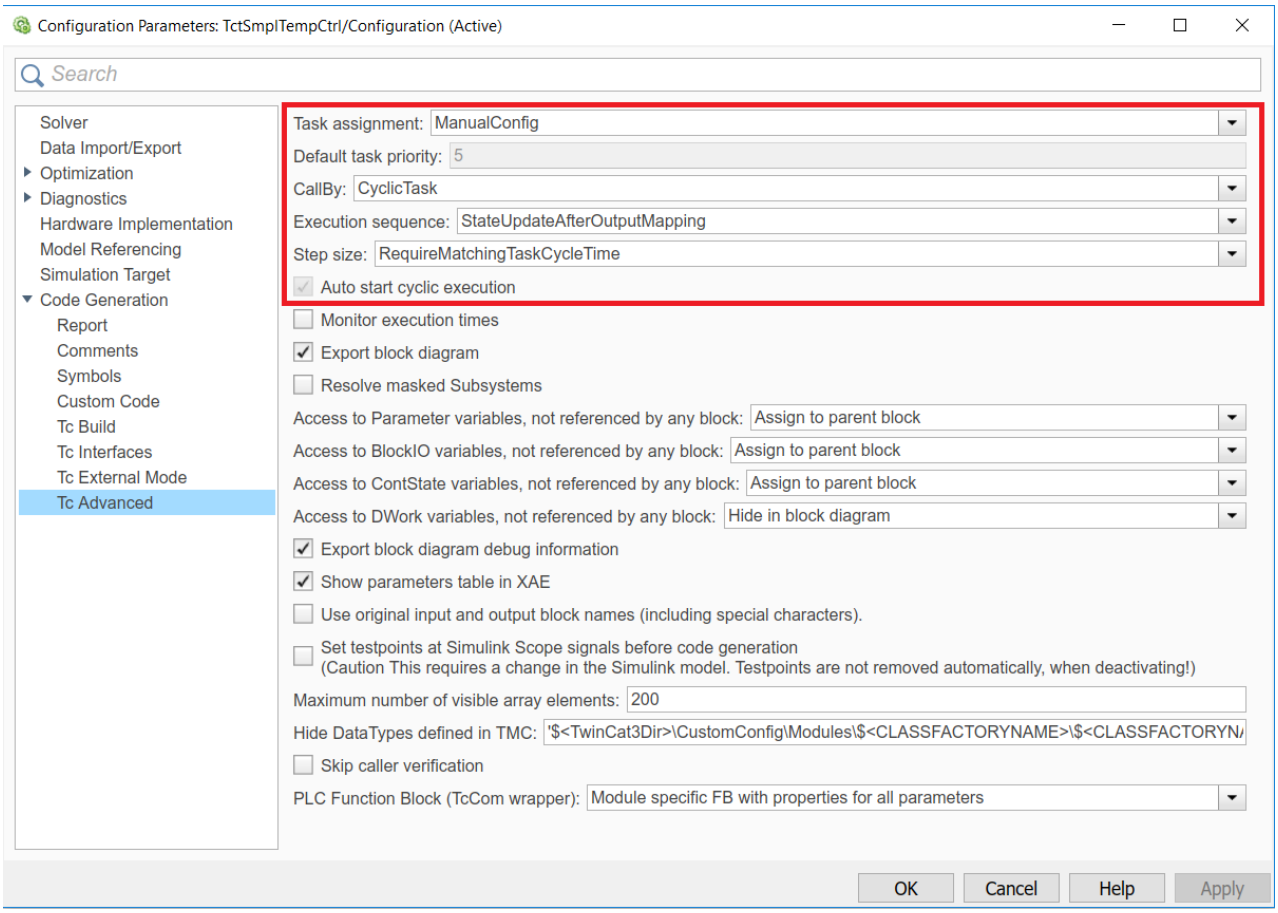


Execution behavior of the generated module

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.

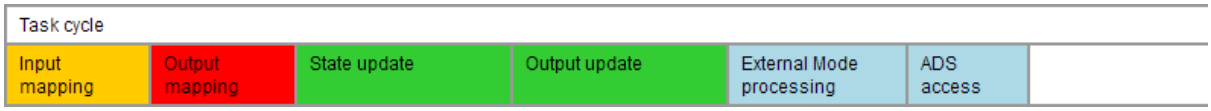
Call from another TwinCAT module

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

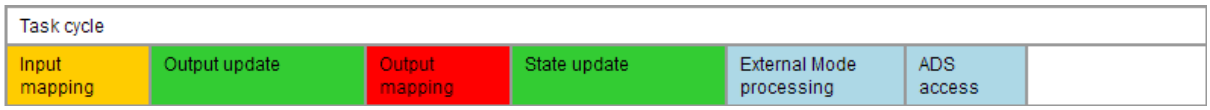
In modules that were created with TE1400 from version 1.1, an execution order can be specified, in order to optimize the jitter and the response times for the respective application. Older versions use always the order "StateUpdateAfterOutputMapping". The following table illustrates the advantages and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

- IoAtTaskBegin



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

- StateUpdateAfterOutputMapping

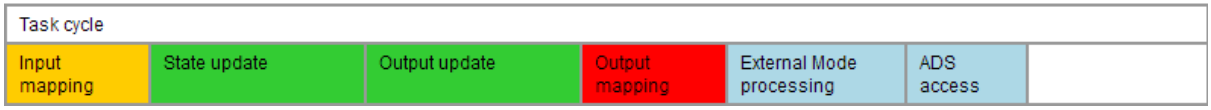


+ Shortest response time
o Medium jitter in the reaction time

Results from "Minor Time Steps" are transferred via ADS

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" [▶ 63].

- **StateUpdateBeforeOutputUpdate**



o Average reaction time
- Maximum jitter in the reaction time, since it depends on the 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.

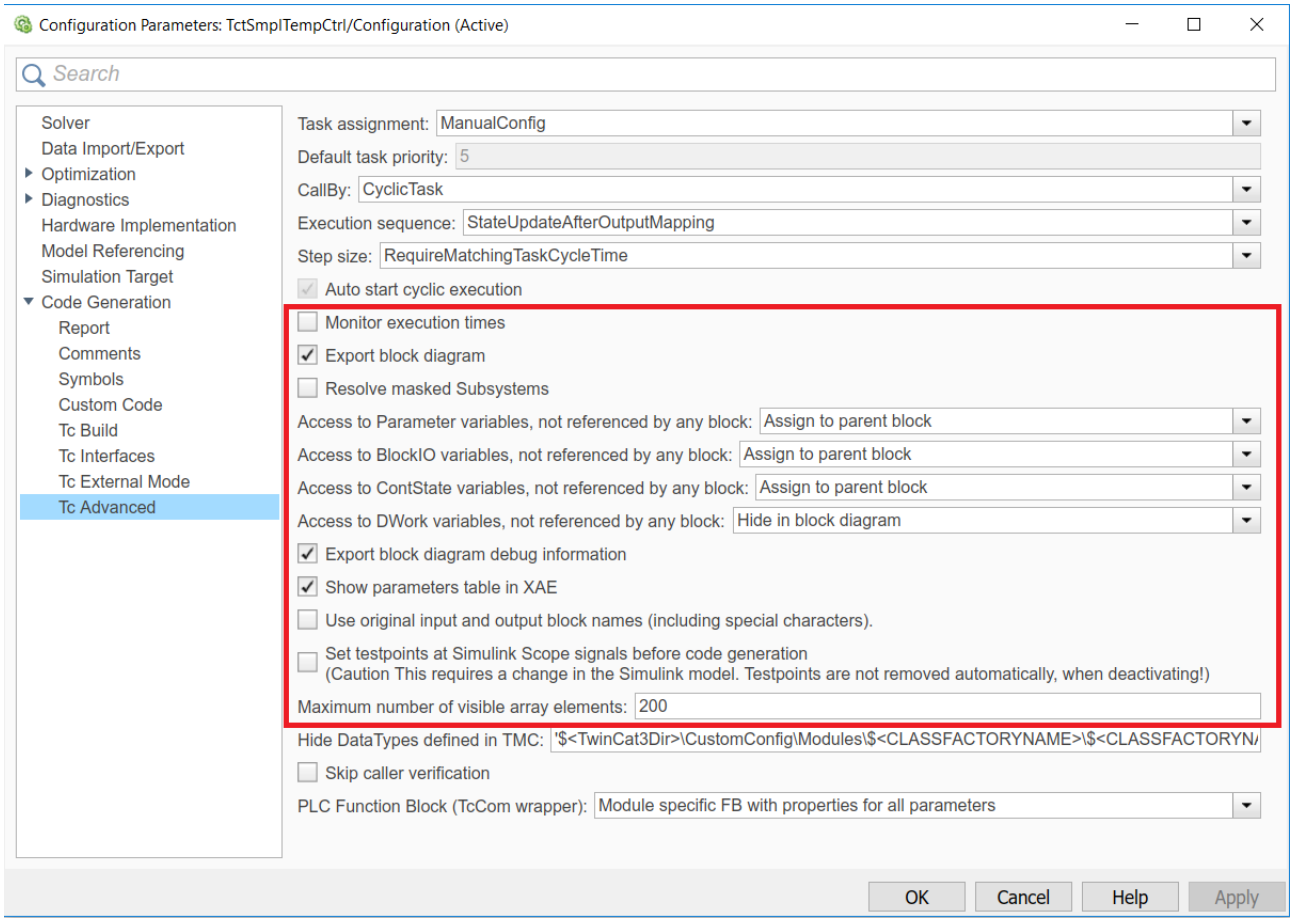
Auto start cyclic execution

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:



Parameter	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
	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 *VariableGroup*, 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 [▶ 24]).	
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 [▶ 55].	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? [▶ 65].

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}'

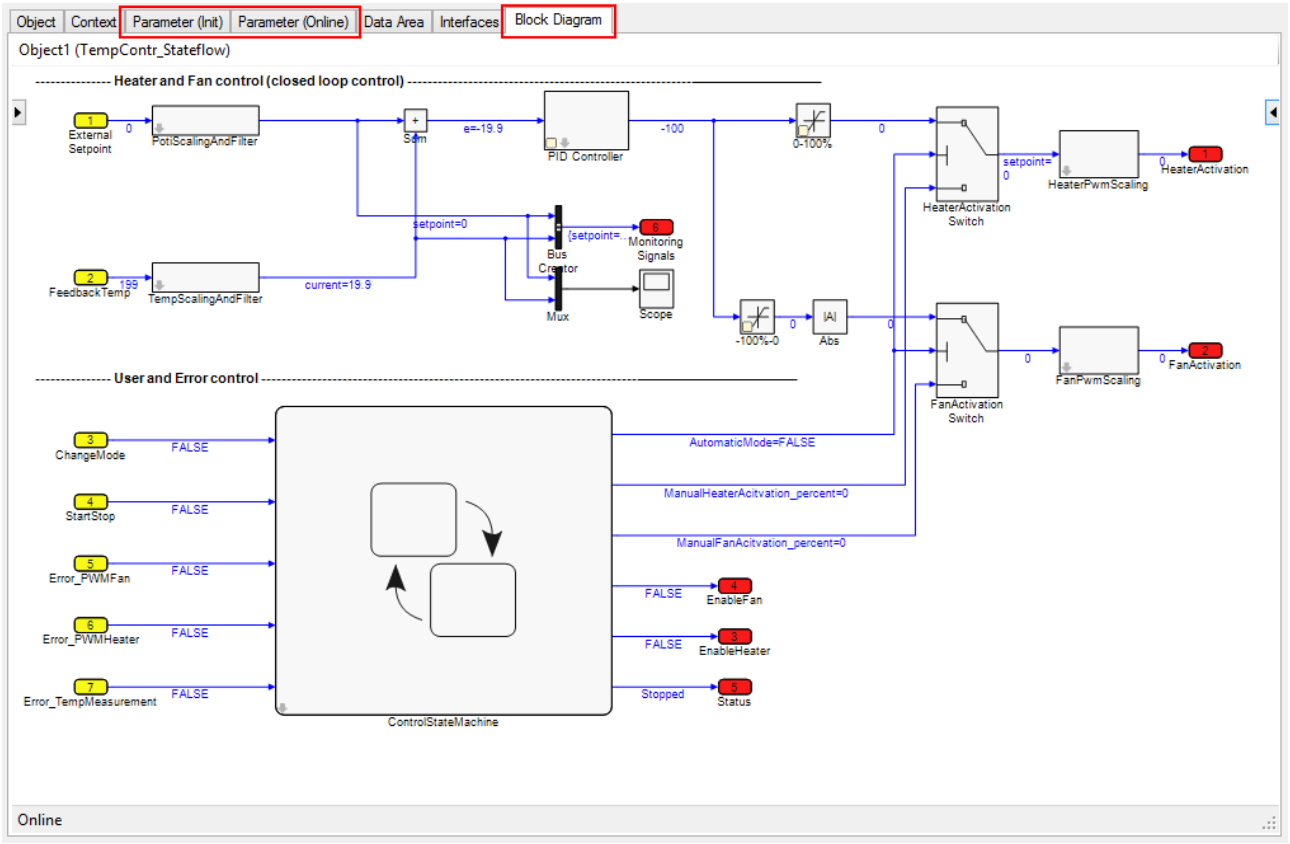
8 Application of modules in TwinCAT

The data of modules exported from Simulink are stored in directory *%TwinCat3Dir\CustomConfig\Modules \<MODULENAME>\%*, and from were 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.

8.1 Parameterization of a module instance

Parameter representation in XAE

The block diagram in the Browser Parameters tab:

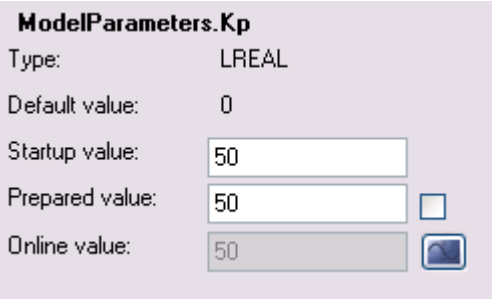


In general, TcCOM modules can be parameterized via the parameter list under the tab **Parameter (Init)** in the TwinCAT 3 development environment (XAE). Simulink modules can also be parameterized via the block diagram, if block diagram export is enabled in the Simulink coder settings under **Tc Advanced**.

Module- and model-specific parameters

The parameter list contains module- and model-specific parameters. Examples of module-specific parameters are "Call Parameter [▶ 40]" or "External Mode Parameter [▶ 29]". 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) [▶ 24]).

"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:

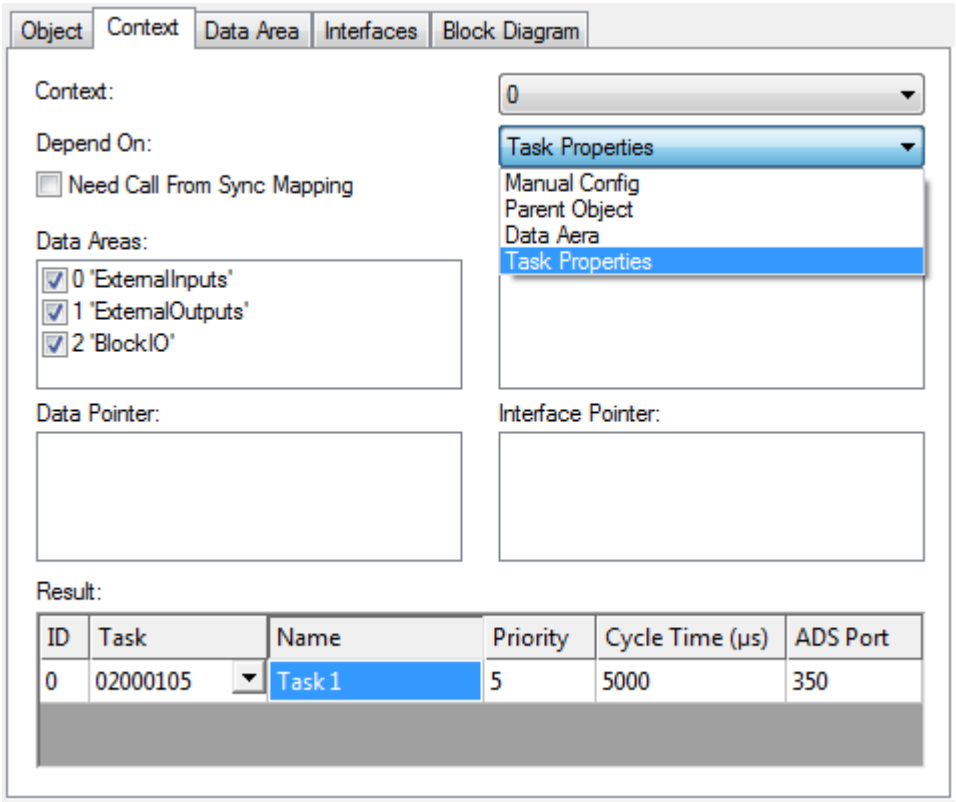
- **Default values** are the parameter values during code generation. They are invariably stored in the module description file and enable the manufacturing settings to be restored after parameter changes
- **Startup values** are stored in the TwinCAT project file and downloaded into the module instance when TwinCAT starts the module instance.
  In Simulink modules, it is also possible to specify starting values for the input process image. 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.

8.2 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:



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".

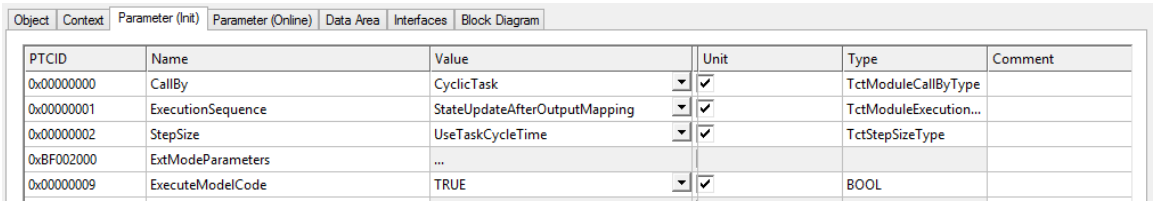
Configuration in XAE

Parameters that affect the behavior of Simulink module execution are:

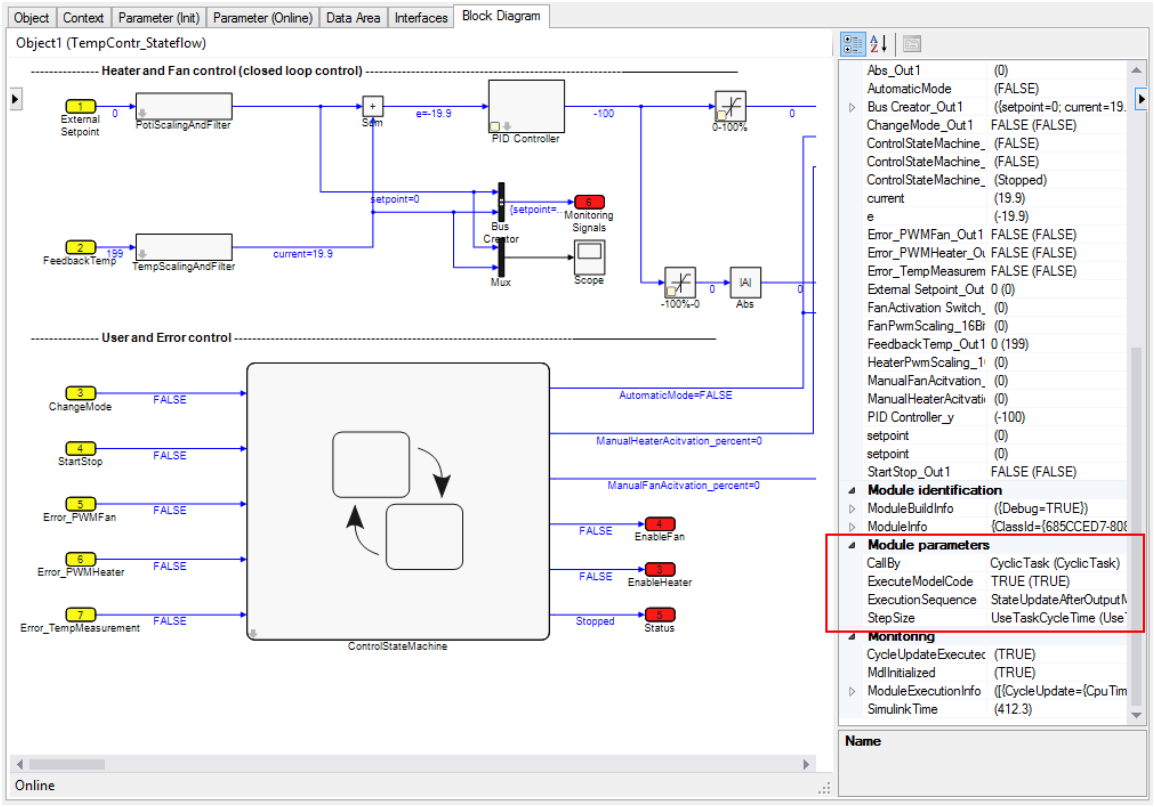
Parameter	Options / description	
CallBy	Task	The module automatically appends itself to the tasks specified in the context settings [▶ 40], when TwinCAT is switched to Run mode. The tasks call the module cyclically until TwinCAT is stopped.
	Modules	The module is not called directly by the assigned tasks, 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 [▶ 35]".	
	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
	StateupdateBeforeOutpunkUpdate	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)** :



- **Block diagram :**



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

8.3 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 from the TwinCAT PLC, as shown below. 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 [▶ 20] 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.

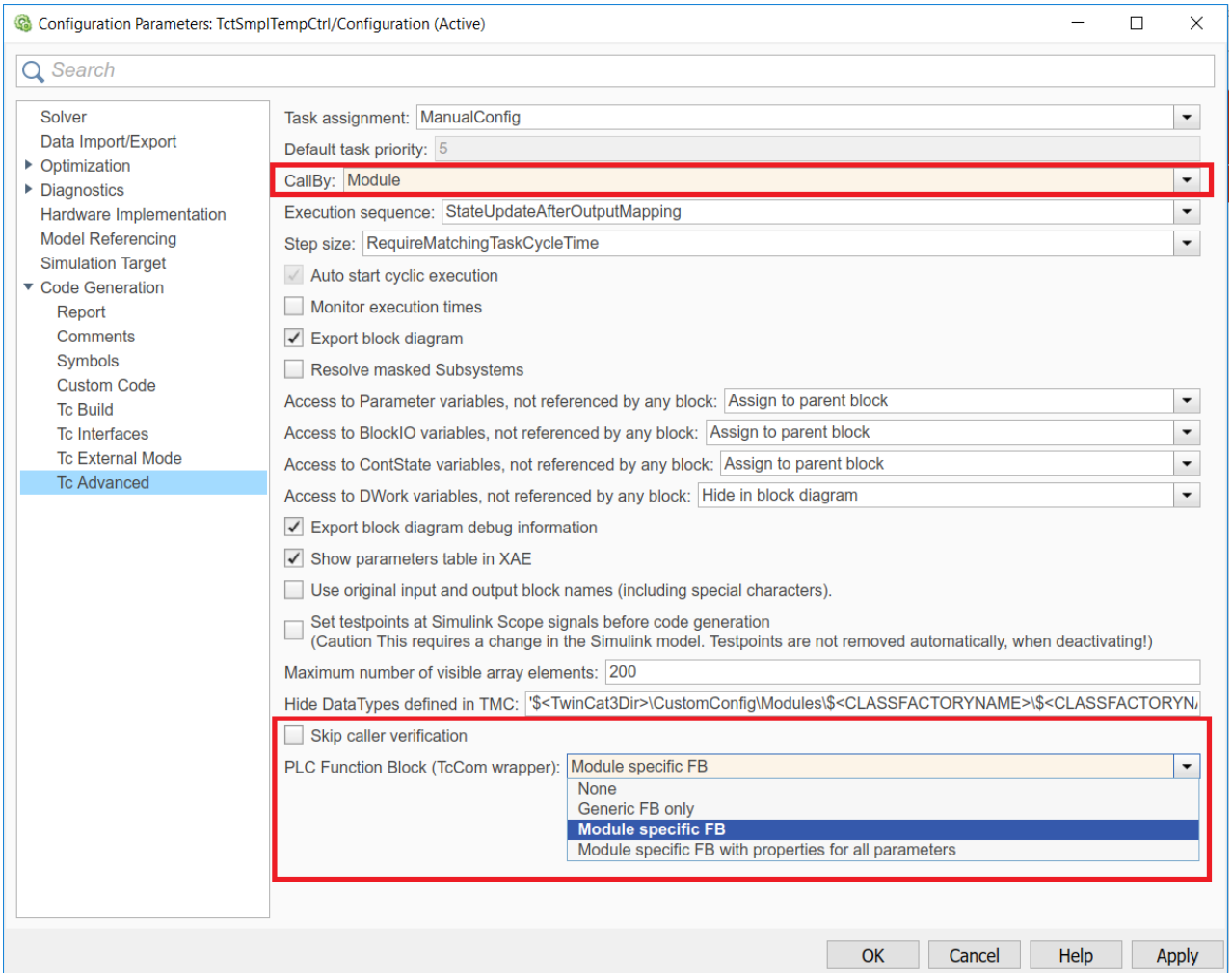
The following descriptions apply from version 1.2.1216.0 of TE1400!!

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: "*PrameterName*_Startup" and "*ParameterName*_Online".

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 node **System > TcCOM Objects**.

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 [▶ 40] 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 Advanced settings (Tc Advanced) [▶ 38].

Import of several PLCopen XML files: 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 pInitData, 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, in order 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 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 data area 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
```

8.4 Using the ToFile block

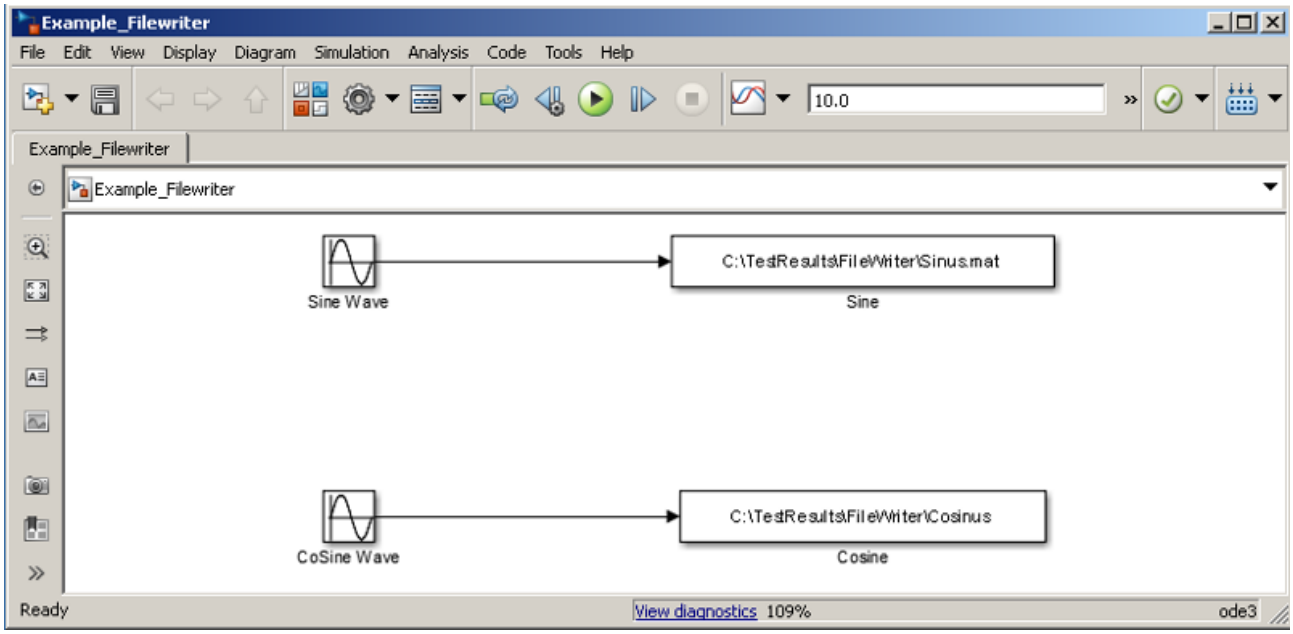
The ToFile block from the Simulink default library can be used to log signals in a MAT file. From within a created TcCOM, this block can still be used from the TwinCAT runtime.

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).

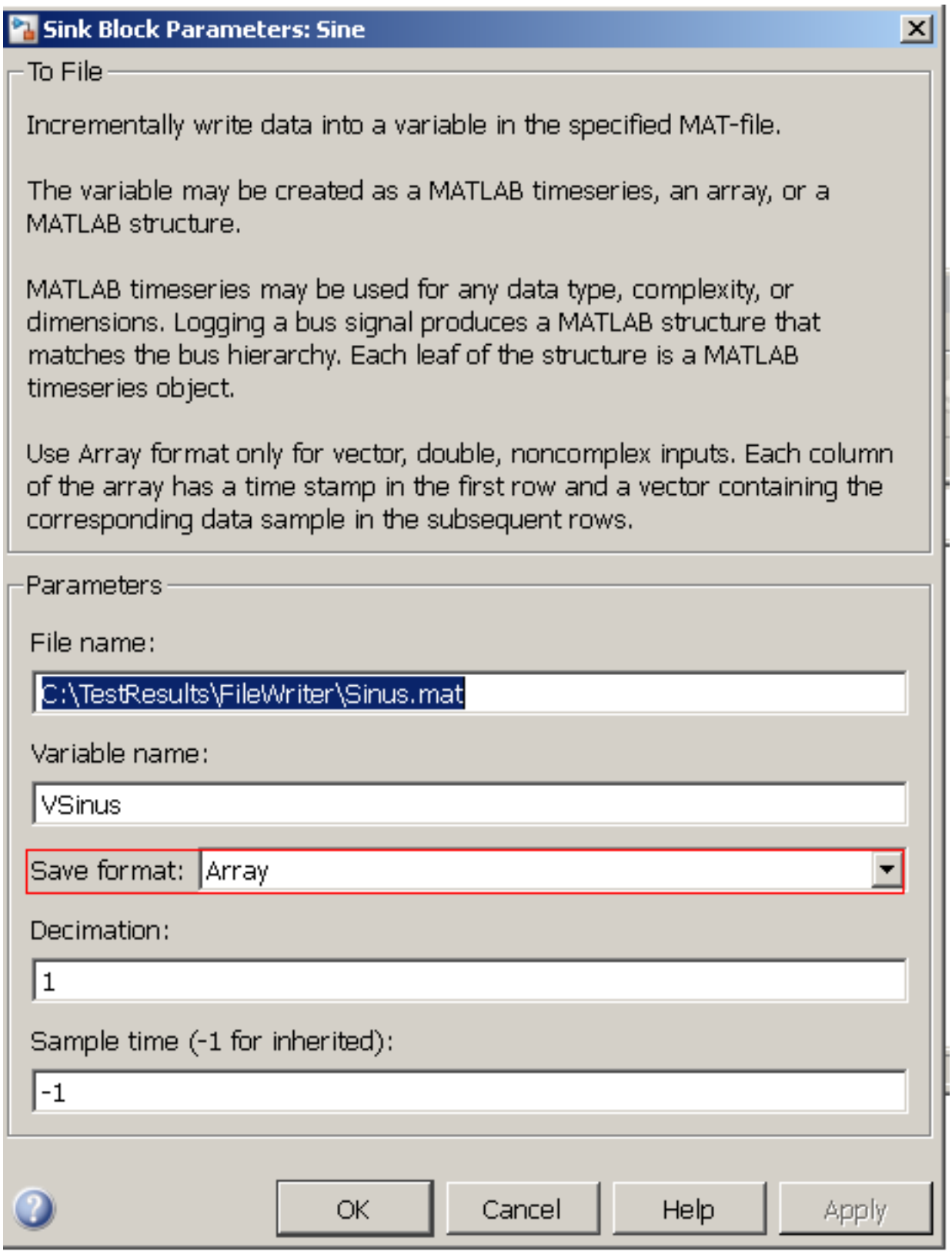
The settings in Simulink and TwinCAT are described step by step below, based on an example.

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.



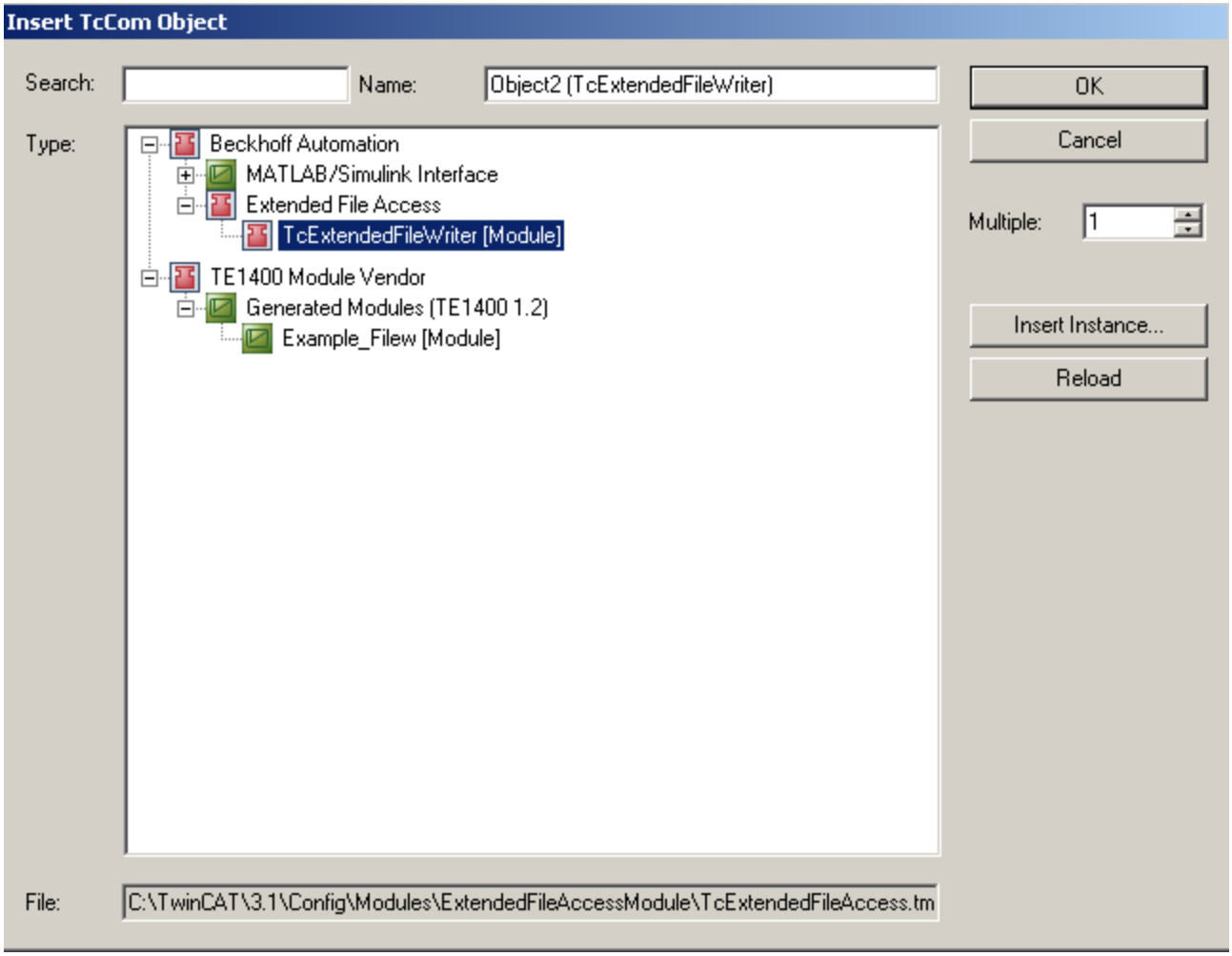
To enable code generation for the ToFile blocks, the format must be set to **Array**:



The model is now ready for code generation.

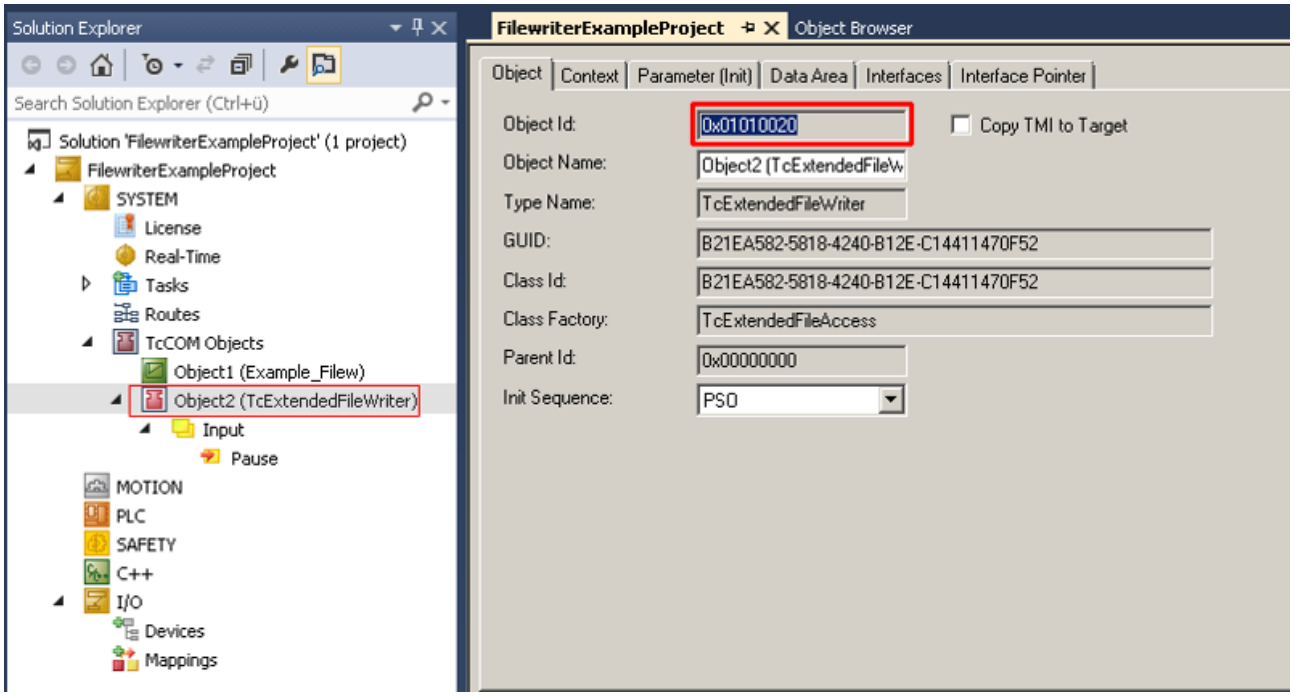
Configuration in TwinCAT

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*:

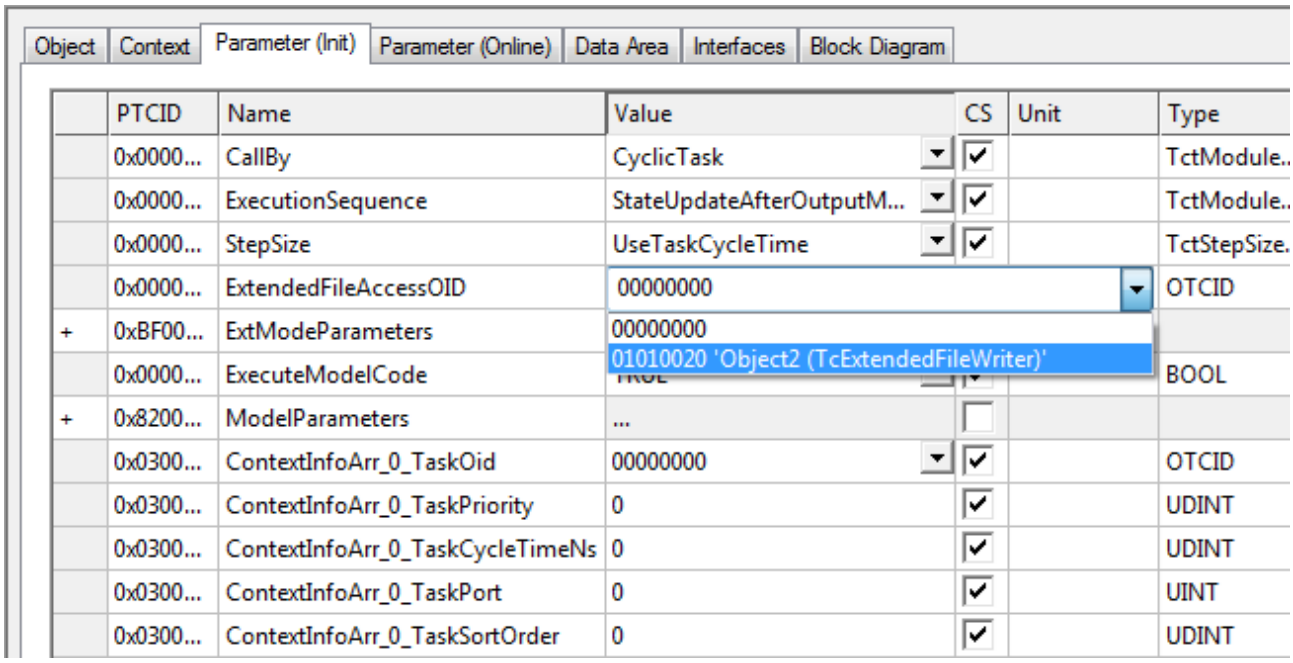


Initially, both TcCOMs are instantiated. Both objects can be linked to a joint task or separate task. In order to establish a link between the two objects, the object ID of the *TcExtendedFilewriter* instance is communicated to the Simulink TcCOM.

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



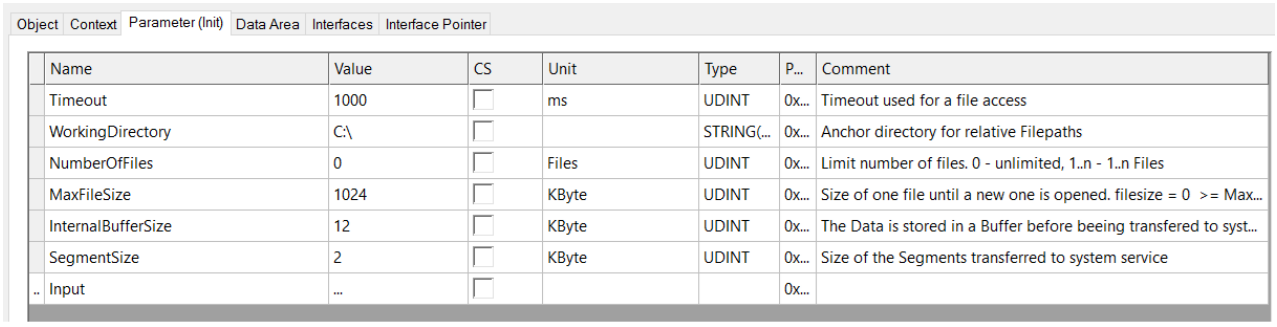
The ObjectID is then inserted under the "Parameters (Init)" tab of the Simulink TcCOM for the parameter **ExtendedFileAccessOID**:



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.



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:

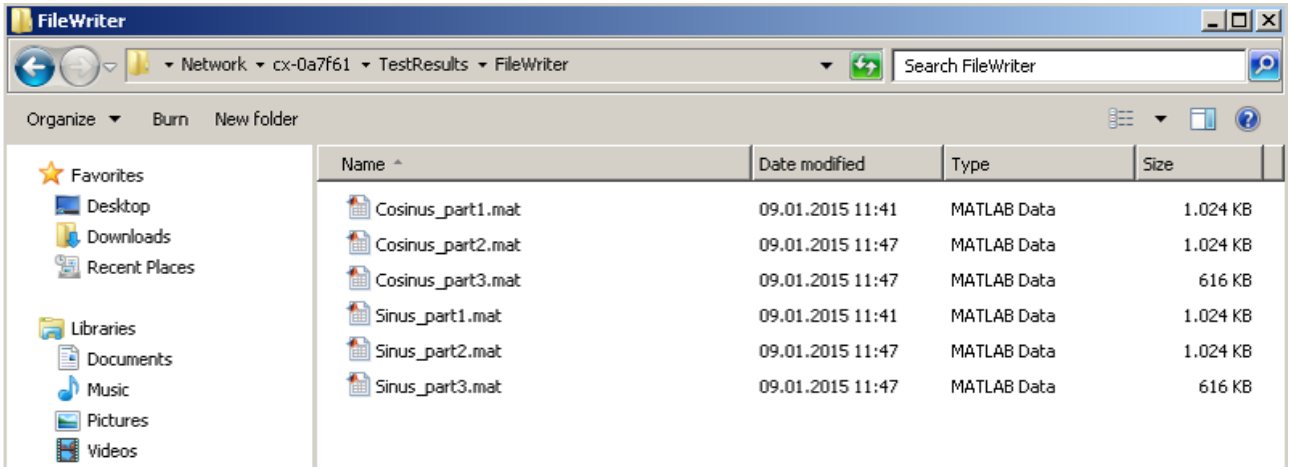
A buffer with the size InternalBufferSize is created on the TwinCAT side, from which the data is then written.

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 TcExtendetFileWriter (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.

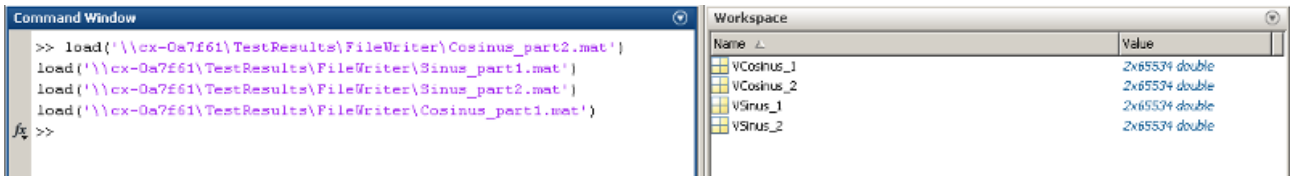
The TwinCAT project can now be activated.

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):

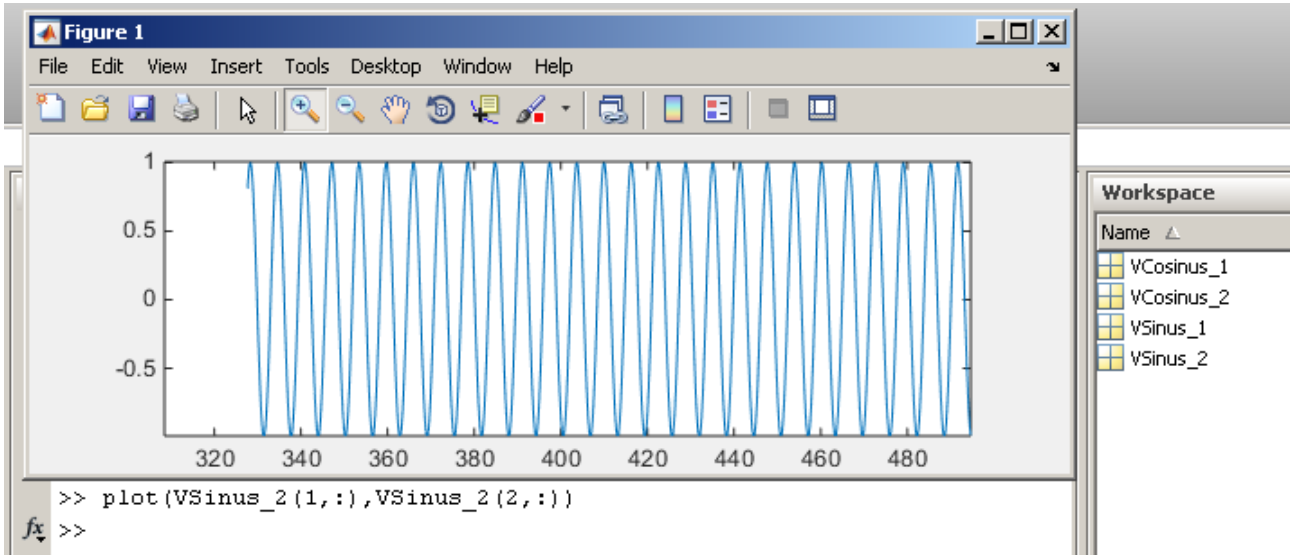


The TcExtendedFilewriter object has a Pause input to prevent continuous writing. If the input is set to TRUE, the file currently in use for write access is closed, and all further incoming data is discarded. If the input is set to FALSE again, a new file for logging incoming data is opened.

The closed files can be opened as usual in MATLAB:

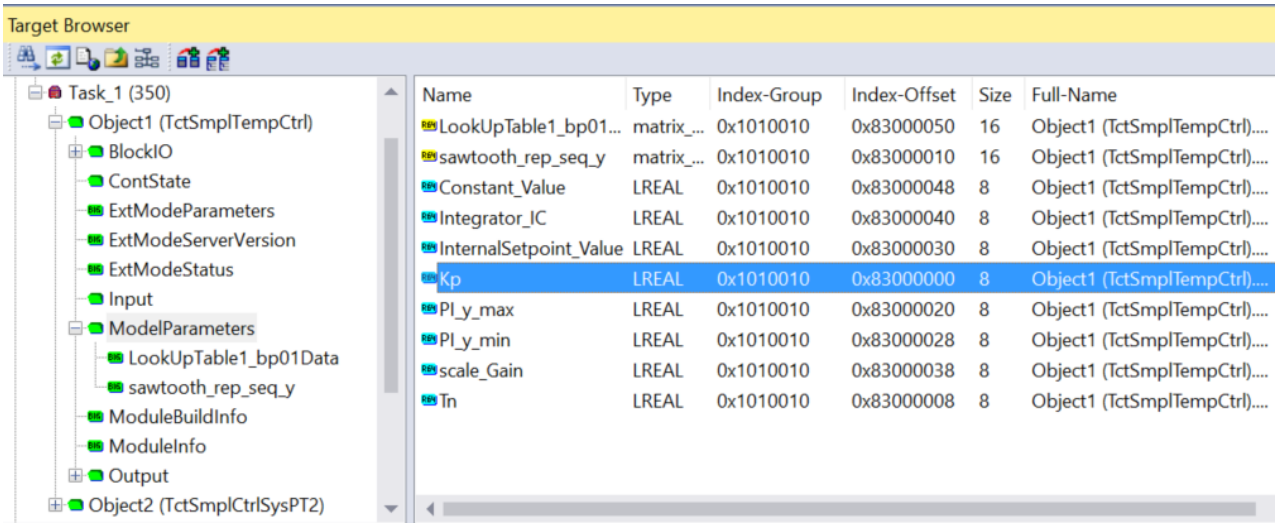


The plot shows the expected sine wave:

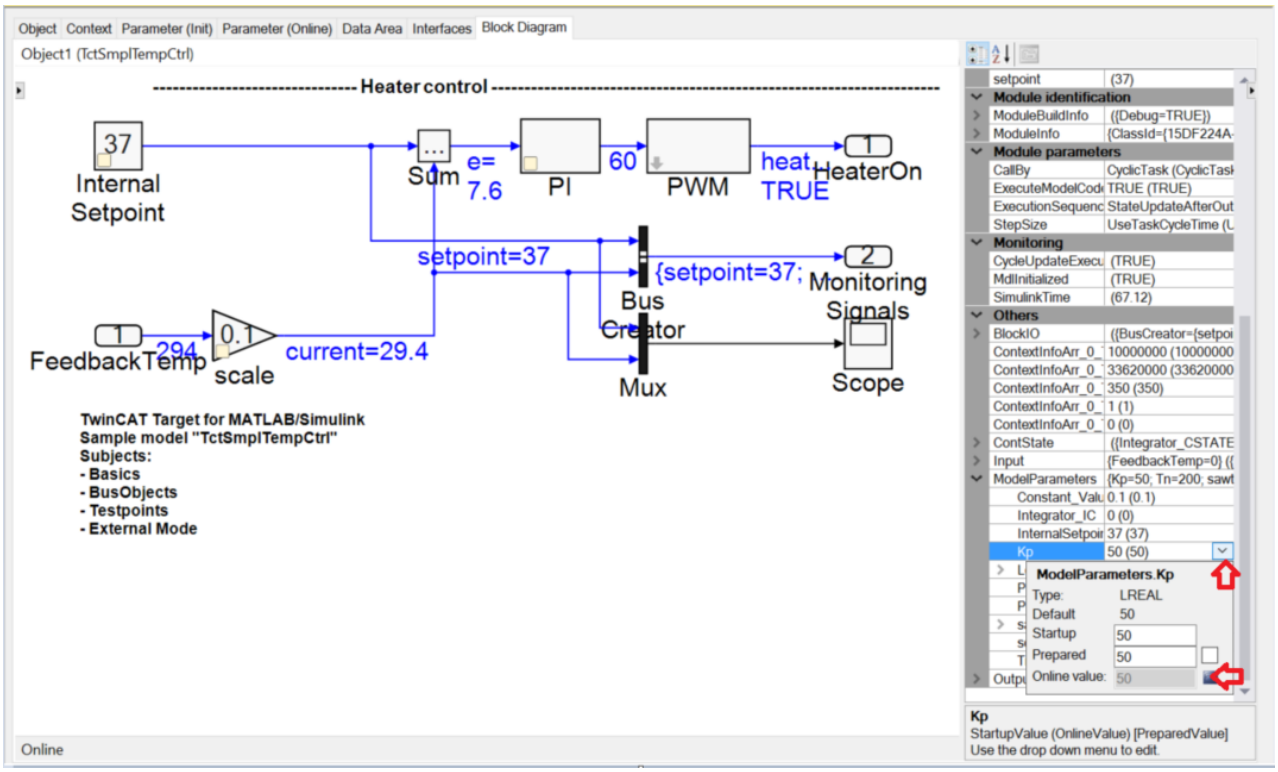


8.5 Signal access via TwinCAT 3 Scope

TwinCAT 3 Scope enables access to all variable groups for which at least read ADS access was enabled, see Tc Interfaces [▶ 24] 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.

9 Debugging

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

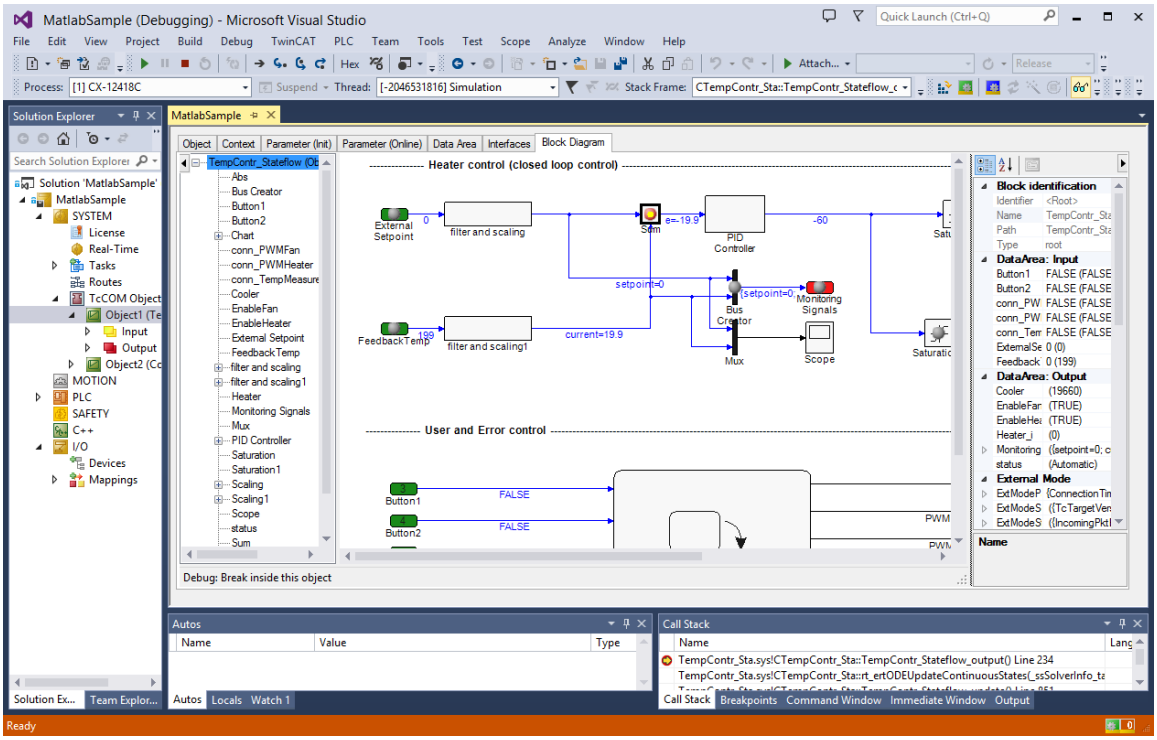
If the block diagram was exported during generation of the TcCOM module, it can be displayed in the TwinCAT development environment and used for debugging within the corresponding module instance, for example. To do so, the block diagram uses the Microsoft Visual Studio debugger, which can be linked 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:

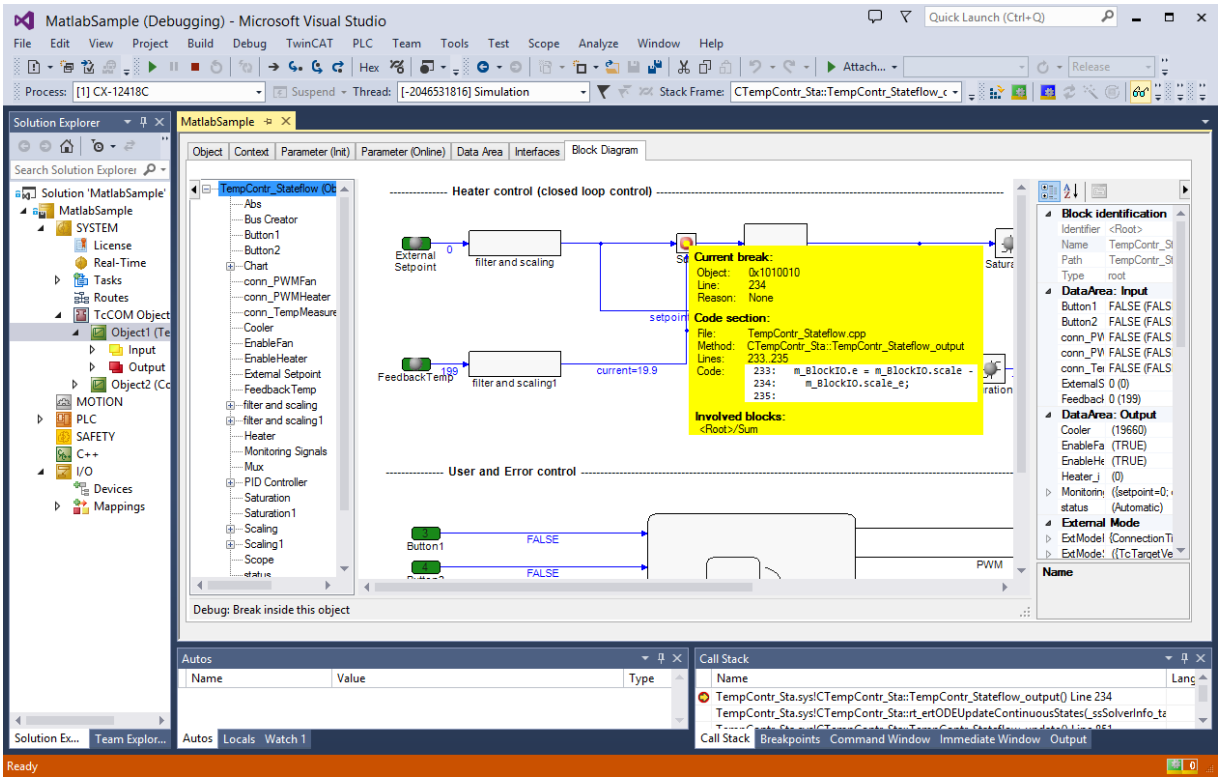
- The C/C++ source code of the TcCOM module must be present on the engineering systems, and the Visual Studio debugger must be able to find it. Ideally, debugging should take place on the system on which the code was generated. If the module was created on another system, the associated C/C++ source 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? [▶ 64].
- 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) [▶ 20] 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 block is executed. The color of the point provides information about the current state of the breakpoint:
    - Grey: Breakpoint inactive
    - Red: Breakpoint active. The program is stopped next time this block is executed
    - Yellow point in the middle: Breakpoint hit. Program execution is currently stopped at this point
    - Blue point in the middle: Breakpoint hit (as yellow), but in a different instance of the module.



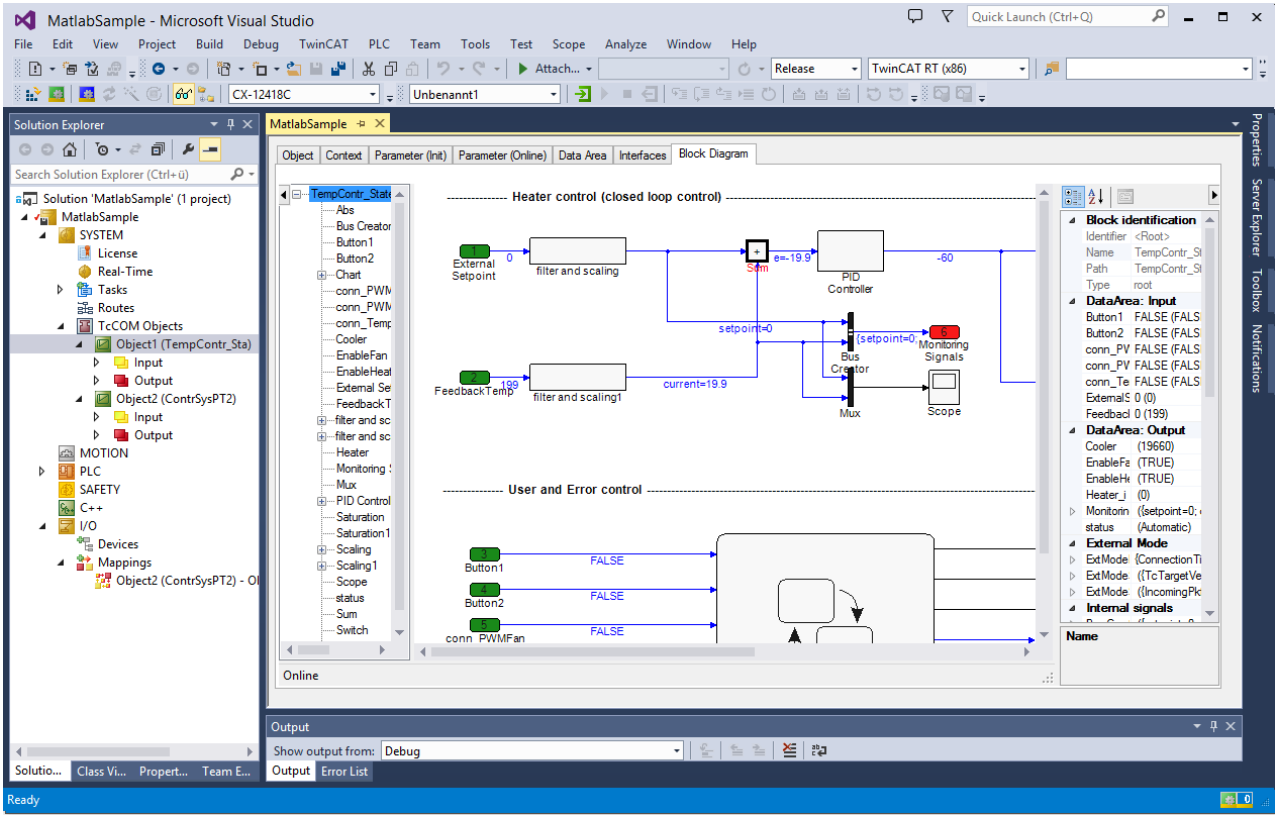
2. Additional information, such as the corresponding C++ code section, can be found in the tooltip for the breakpoint:



Breakpoints are not always assigned to a single 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 block is shown in red, and the block itself is marked in bold.



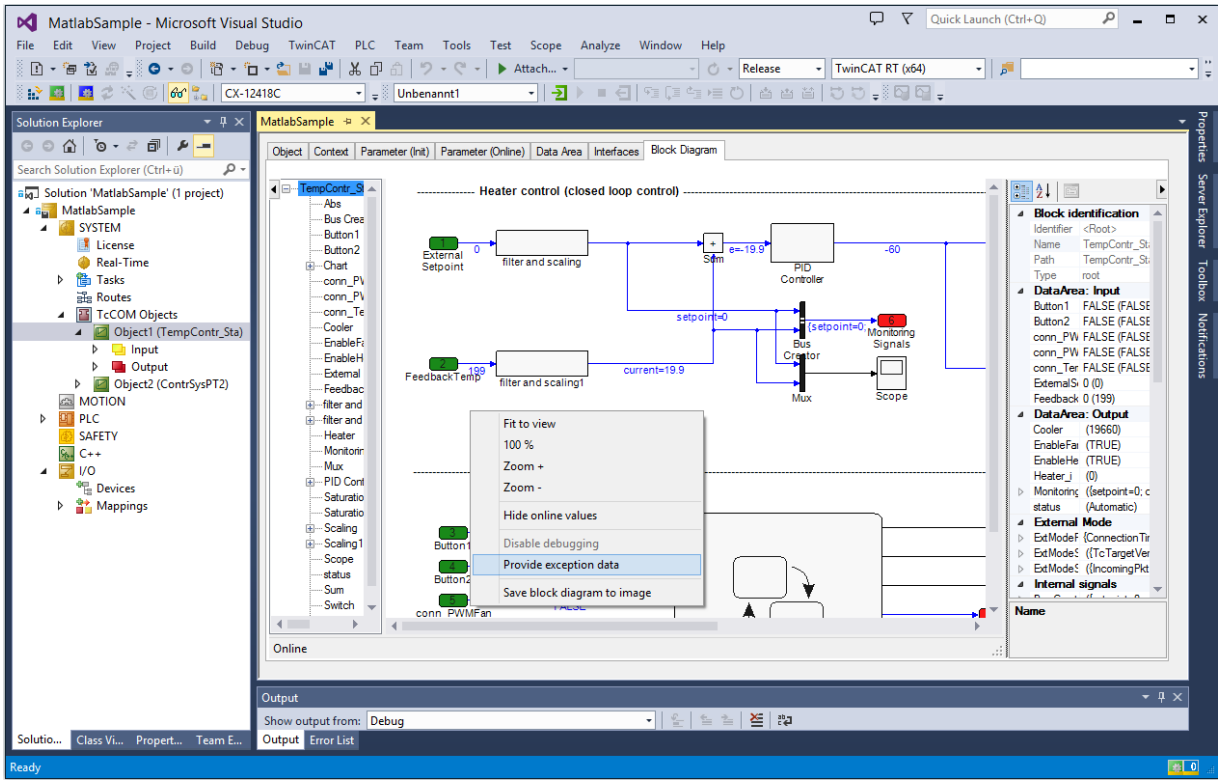
Manual evaluation of exceptions without source code

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.

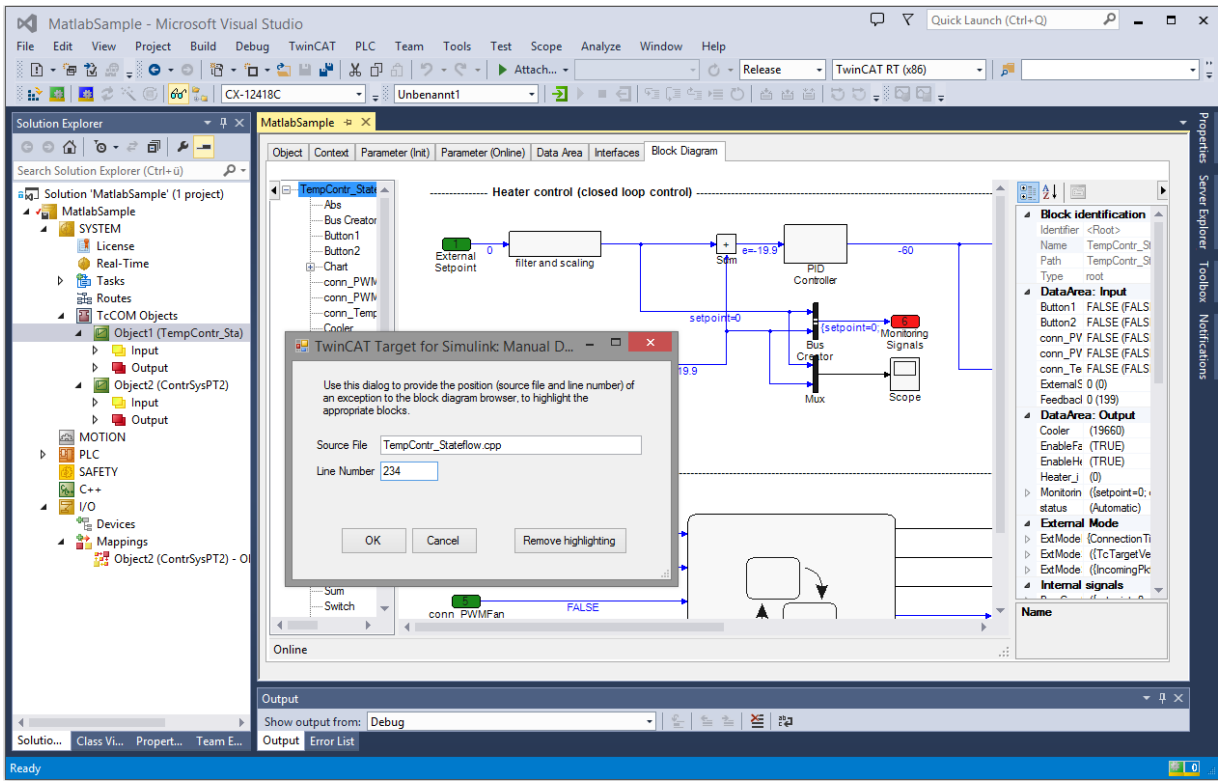
Typically, an error message will always be generated when an error occurs, indicating the source file and the line in the source code. In many cases, this information can be used to allocate an exception to a block in the block diagram. To do this, you can proceed as follows:

✓ 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**).

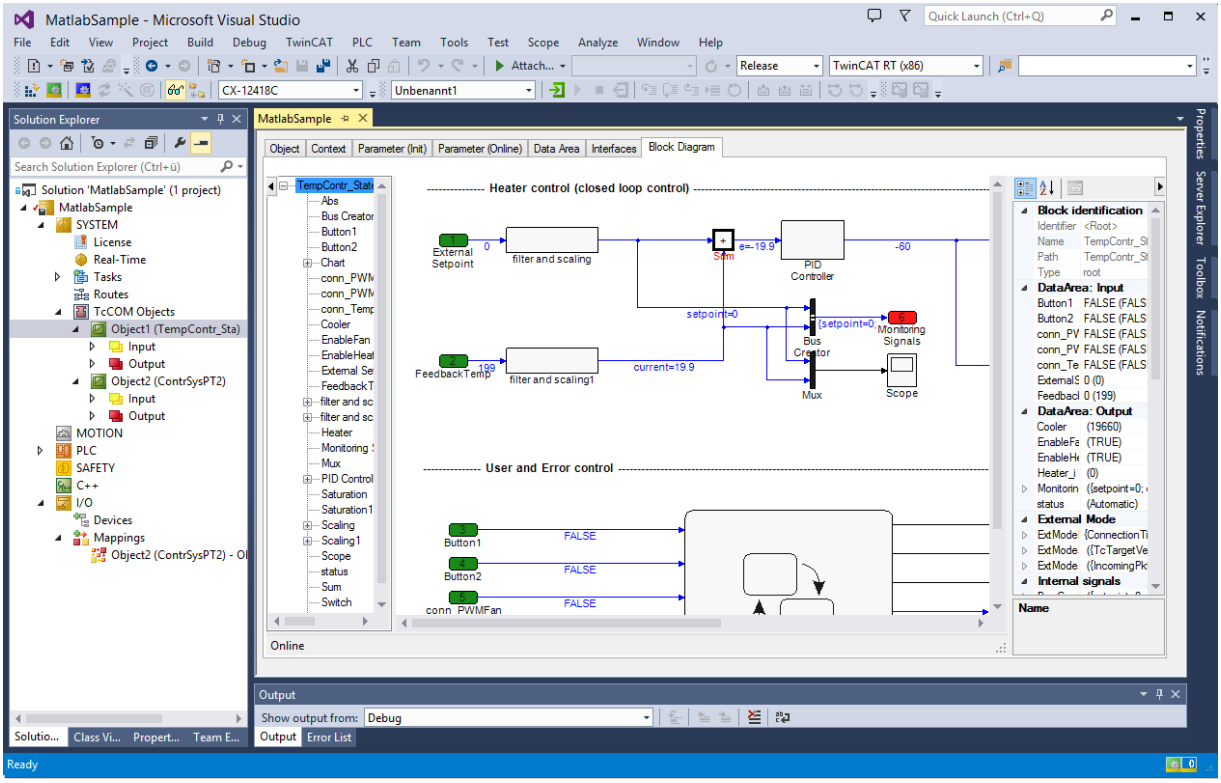
3. From the context menu of the block diagram select the entry **Provide exception data**:



4. In the dialog that opens, enter the source code file and line number provided in the error message:



5. The name of the block associated with the line number is displayed in red, and the block itself is marked in bold:



10 FAQ

10.1 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. For information on how to create a TLC file for an S-Function, see the MathWorks documentation: 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 [▶ 83].

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 [▶ 89].

10.2 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 Debugging [▶ 55] 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.

NOTE

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!

10.3 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? [▶ 64]. 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.

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

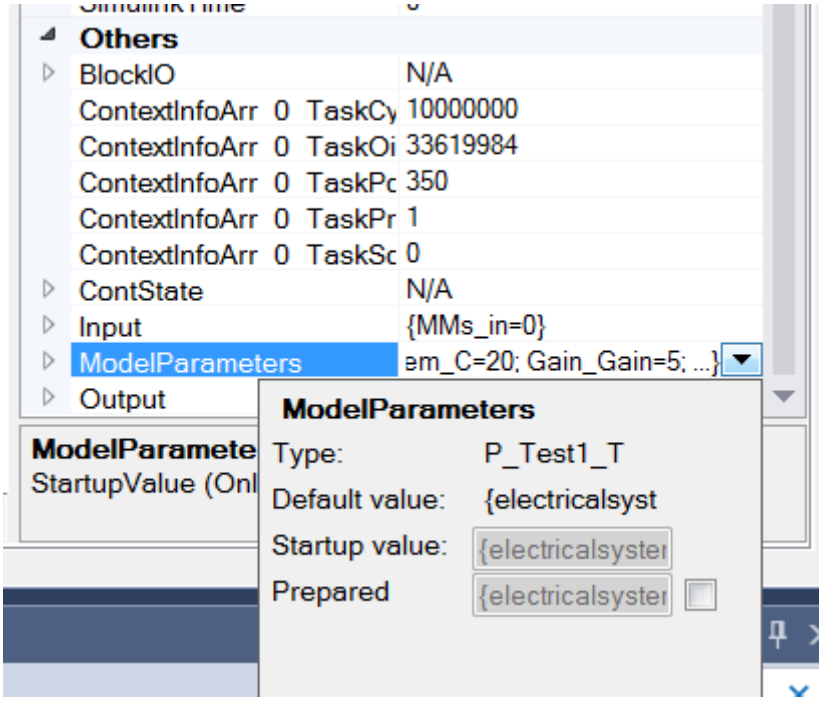
Observation:

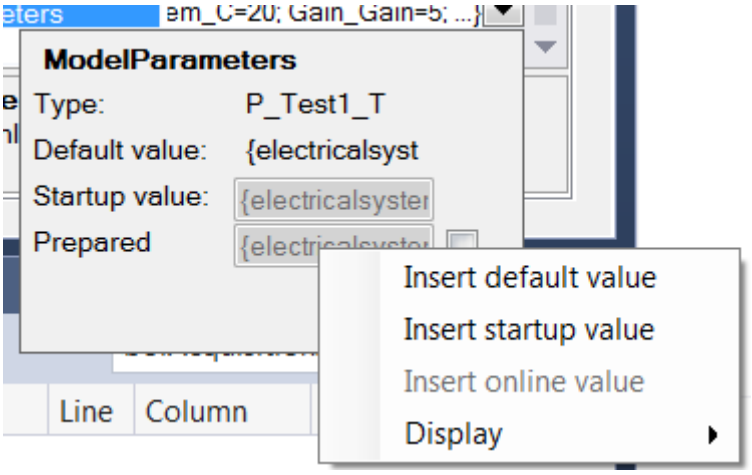
TwinCAT 3 contains an existing instance of a TcCOM object.
As already explained, the model parameters, e.g. the parameters of a PID controller, can be modified in TwinCAT via the exported block diagram or via the **Parameter (init)** tab of the TcCOM object outside of Simulink. If you change your Simulink 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.

10.5 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.

10.6 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 _ModulIInstall 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) [▶ 20].

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.

10.7 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 [▶ 40] 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 [▶ 29].

10.8 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 [▶ 49], 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.

10.9 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 <ModelName>_codegen_rpt.html	Summary of code generation and publishing process in html format.
*.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 *existing project* 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, …)
<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].

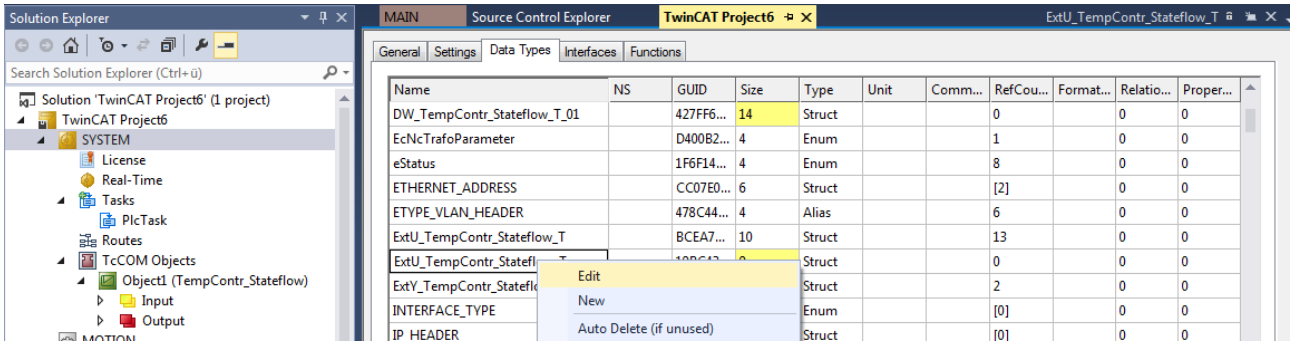
The files stored in the Build directory are suitable for transfer to other engineering systems, just like the files in the Publish directory. On the corresponding engineering systems, the publish process must then be performed manually via the C++ area in TwinCAT. In addition to the resources for the 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.

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

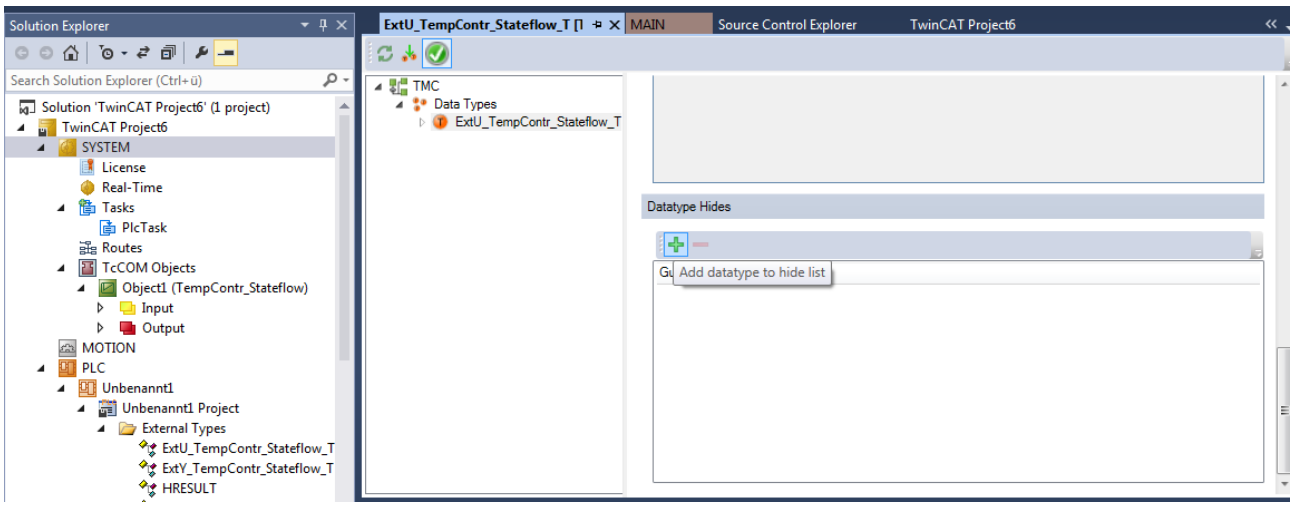
If inputs, outputs, parameters or state variables of a Simulink model are changed, the corresponding data types in the TwinCAT module generated from it also change. After the update, the data types have the same name but a different GUID. The type system of the TwinCAT development environment (XAE) can 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 the 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:



10.11 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.

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

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.

11 Samples

Example models for generating TcCom modules:

Example	Topics	Description
TemperatureController_minimal [▶ 68]	• Basic principles	A very simple temperature controller that covers the basics.
TemperatureController [▶ 74]	• 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 vie ADS. ExternalMode is also used in the example.
SFunStaticLib [▶ 83]	• SFunction • Static library	Generates TwinCAT modules from Simulink models with SFunctions that are provided by third parties without source code.
SFunWrappedStaticLib [▶ 89]	• 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 [▶ 22]:

Example	Topics	Description
Packaging module files into ZIP archives [▶ 94]	• PostPublish callback • Archiving generated module files	This simple example illustrates the automatic archiving of generated module files.

11.1 TemperatureController_minimal

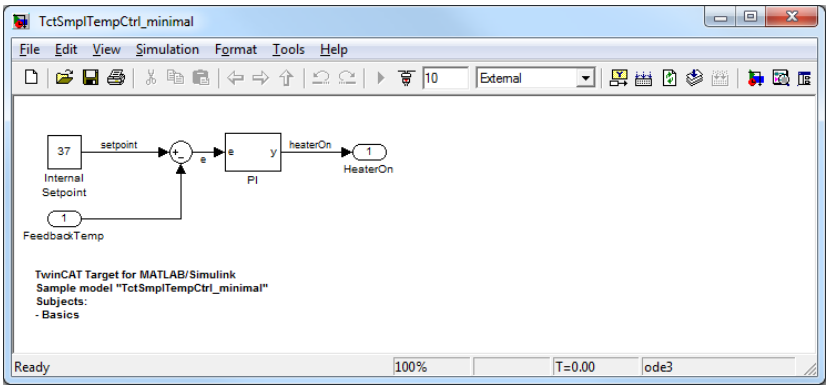
Description

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

Overview of project directory

TE1400Sample_TemperatureController_minimal (Resources/zip/27021599304189451.zip) contains all the files required for reproducing this example:

TctSmplMinTempCtrl.mdl Simulink much of a simple PI temperature controller.



TctSmplTempCtrlParameters.mat Contains all the necessary model parameters.

TctSmplMinCtrlSysPT2.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.

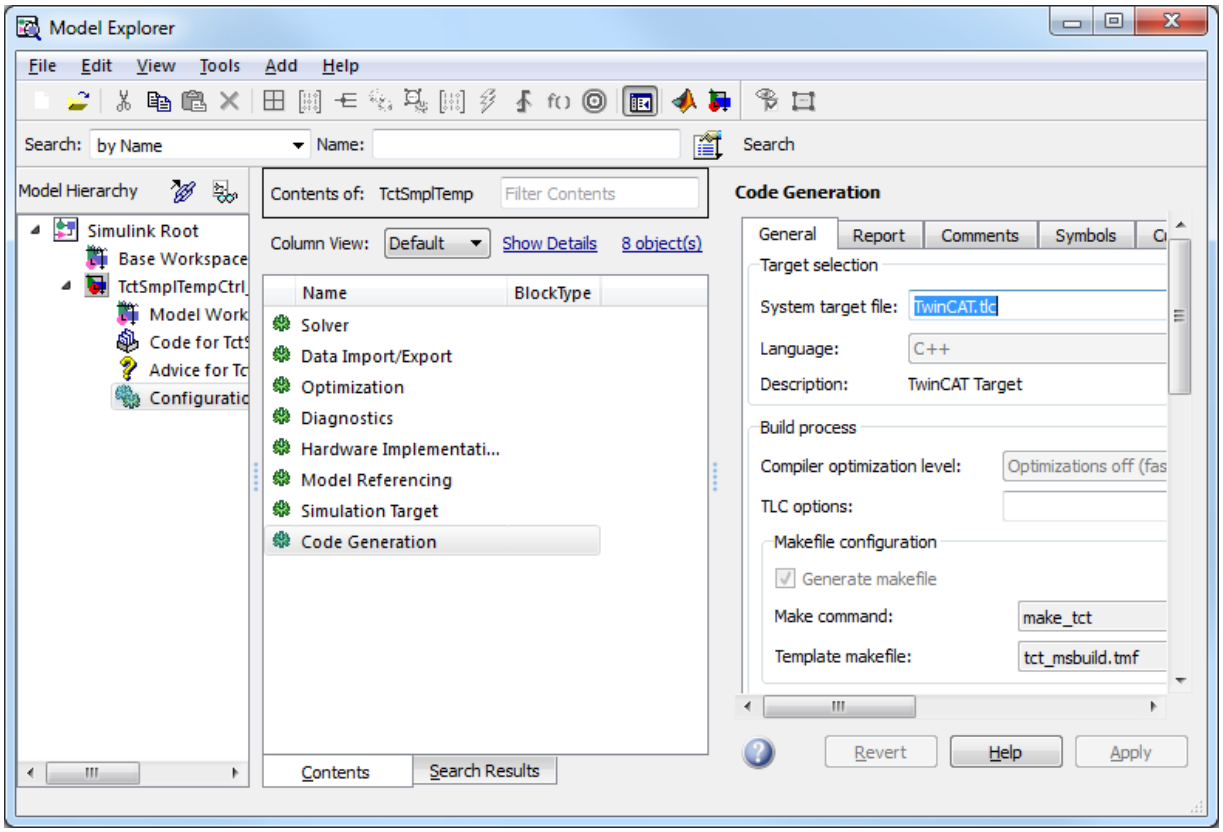
Info: To start the module on an x64 target system, the system must be switched to test mode!

_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.

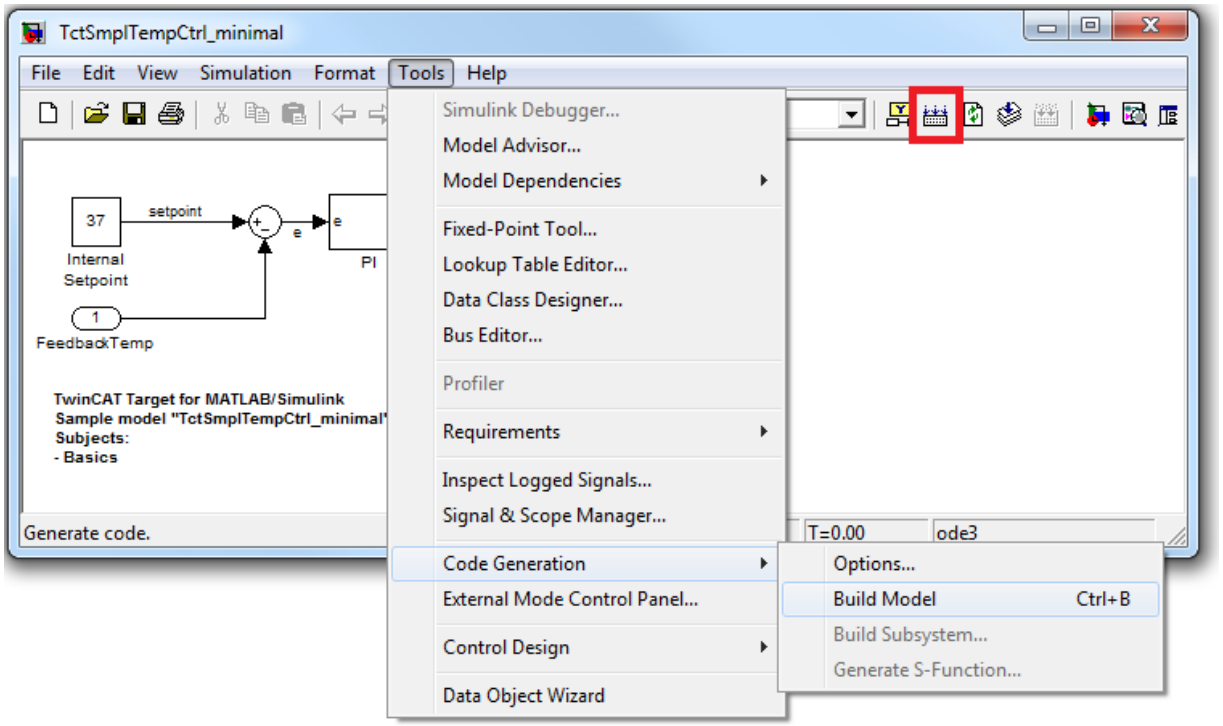
Generating a TwinCAT module

1. Open *TctSmplMinTempCtrl.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:



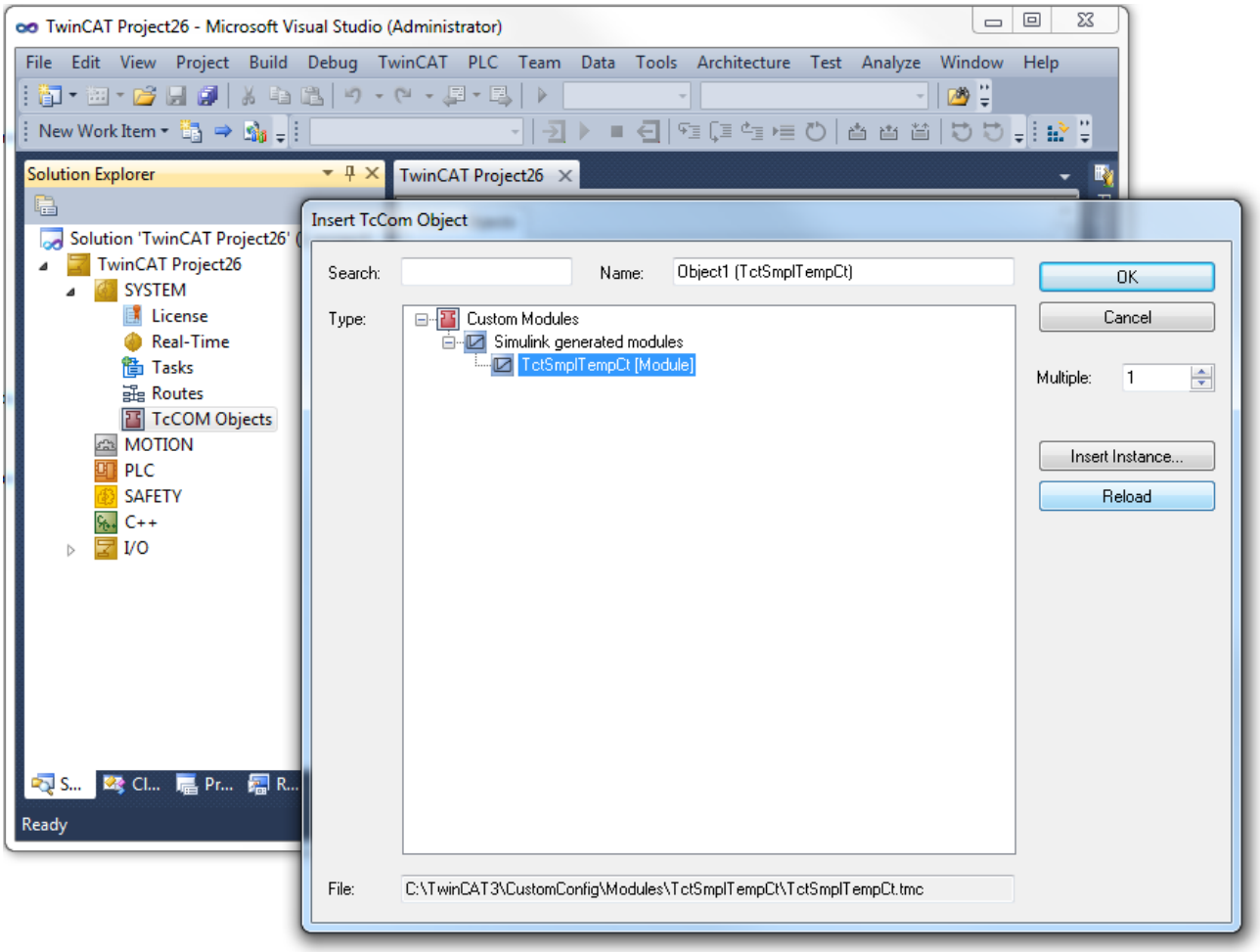
4. Close **Model Explorer**
5. Start code generation via the Simulink menu item **Tools->Code Generation-> Build Model** or via the toolbar icon **Incremental build**



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

Using the generated TwinCAT module

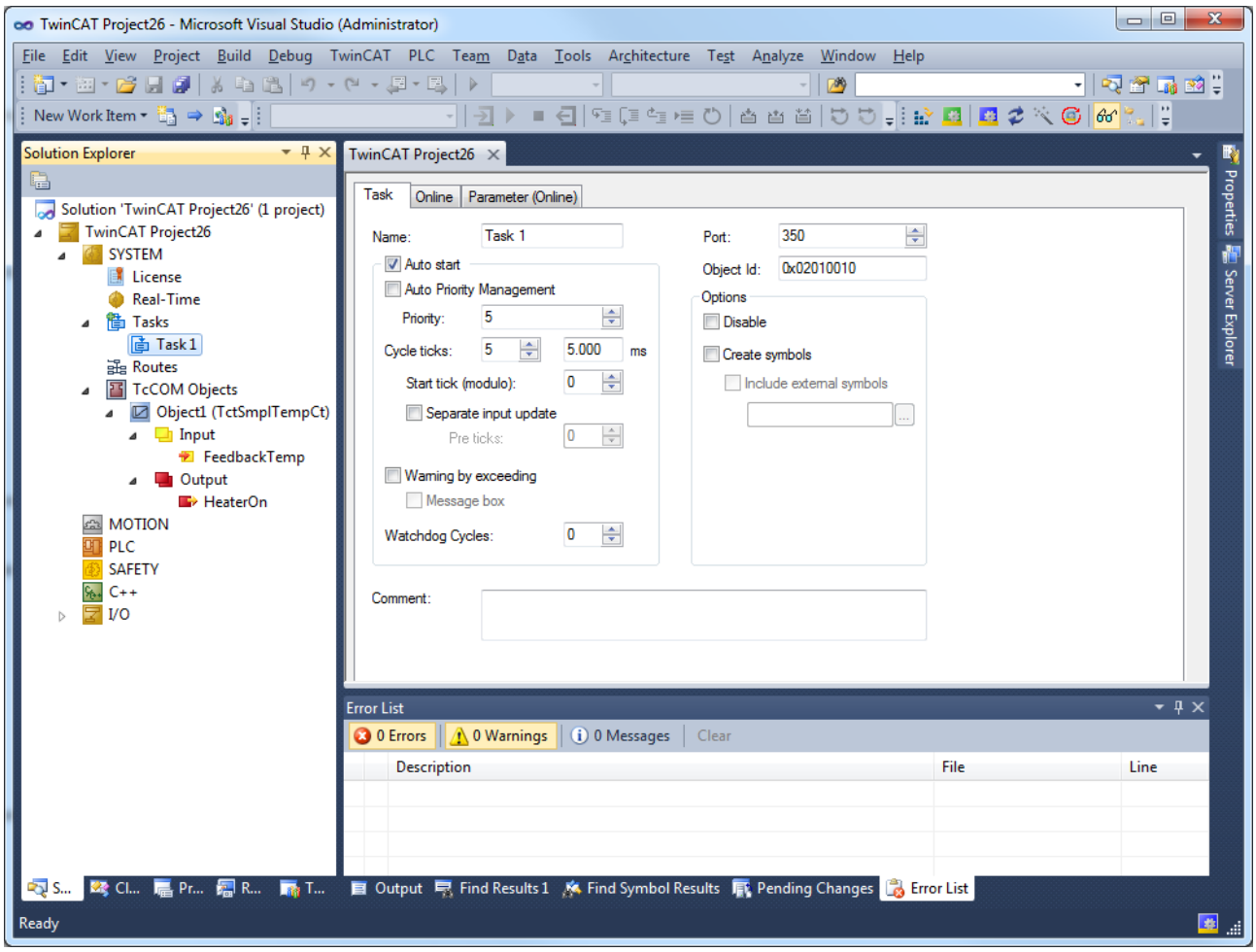
Open the TwinCAT development environment and create a new TwinCAT project. Expand node **System** in the **Solution Explorer**. Select the menu item **Add new item** in the context menu of node *TcCOM Objects*. The following dialog is displayed:



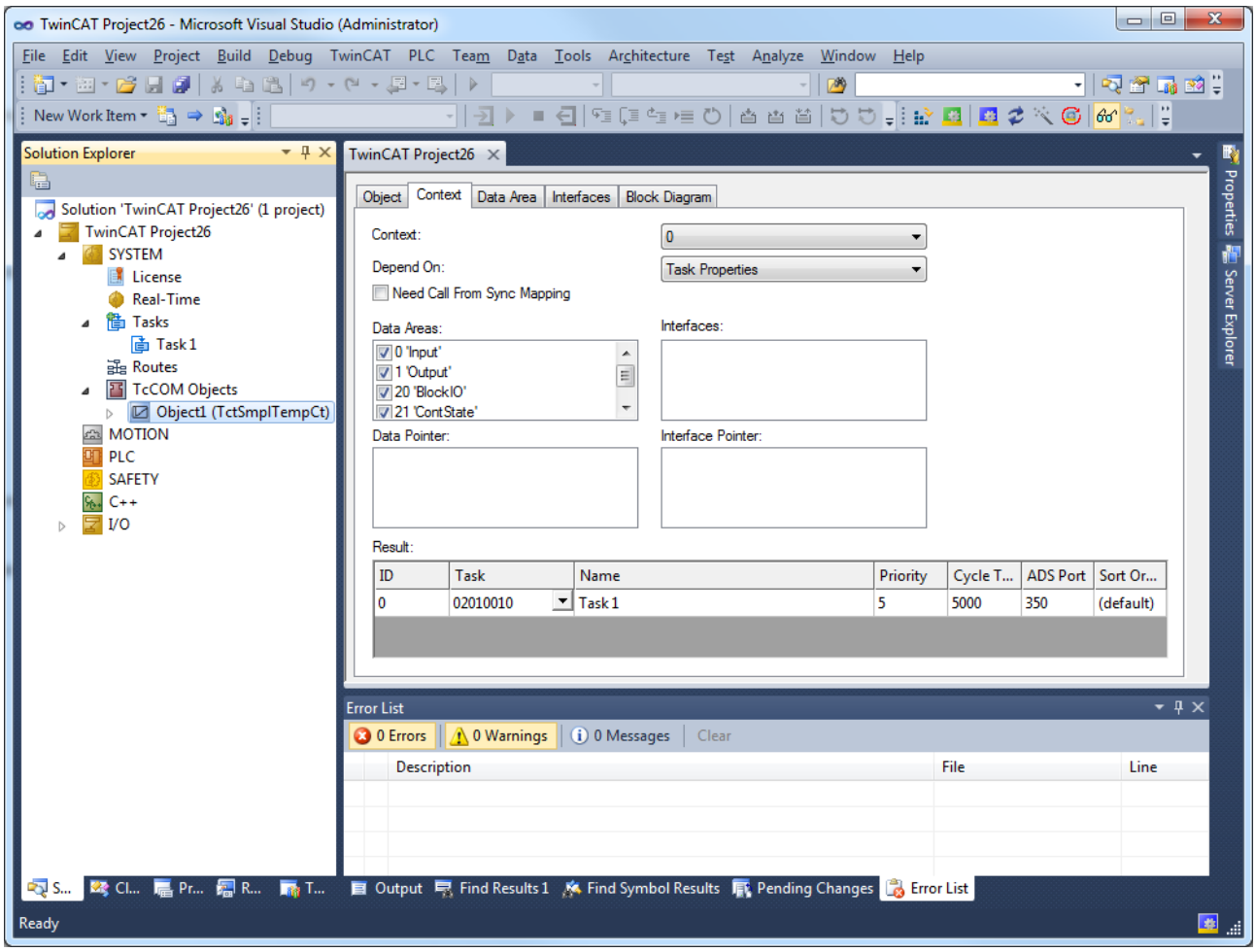
Select the generated module from the group **Custom Modules -> Simulink generated modules**. If XAE was started before the end of the code generation, first press the **Reload** button.

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



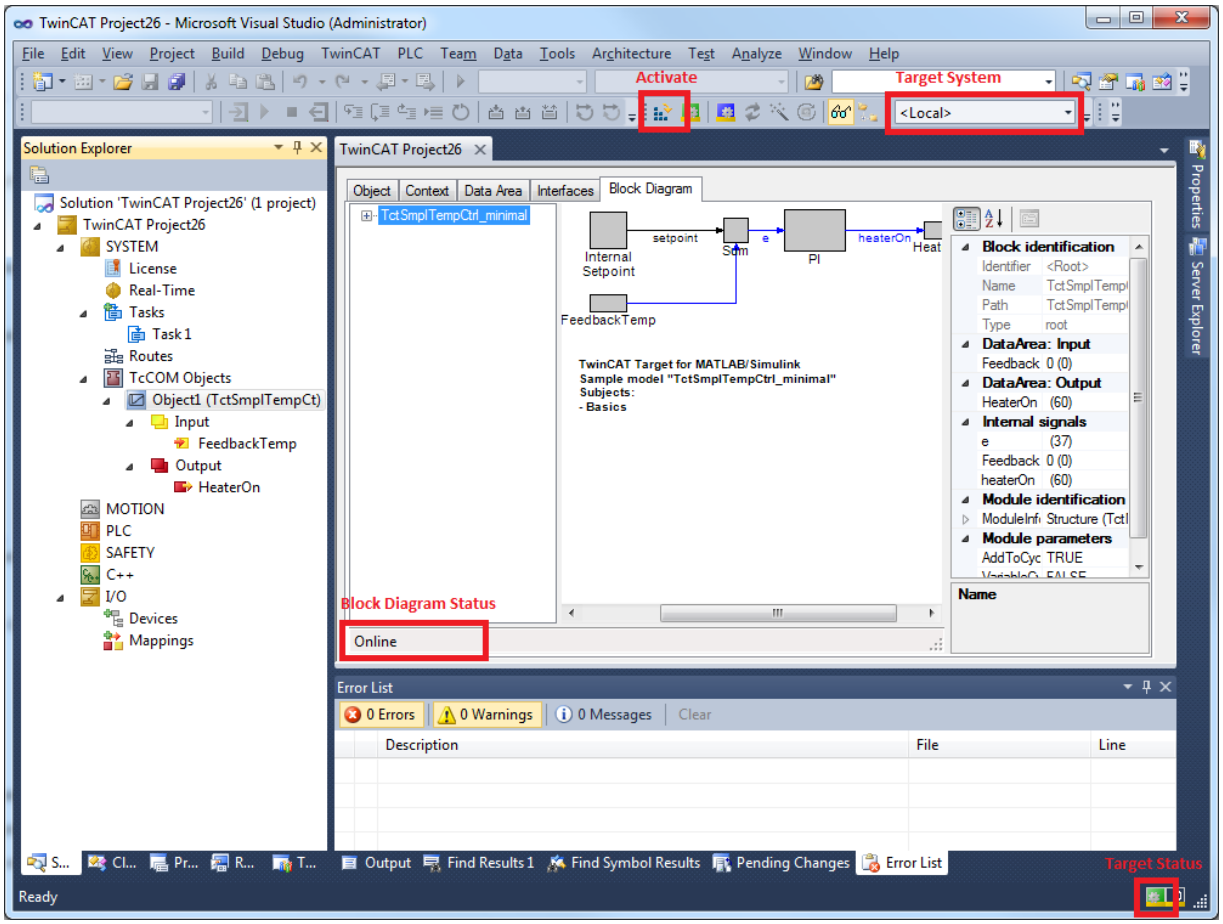
The module (with its default settings) should then have been configured automatically for attaching to this task. To verify this, select the object node **Object1 (TctSmplTempCt)** and open the **Context** tab. The **Result** table should contain the object ID and the object name of the task, as shown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



11.2 Temperature Controller

Description

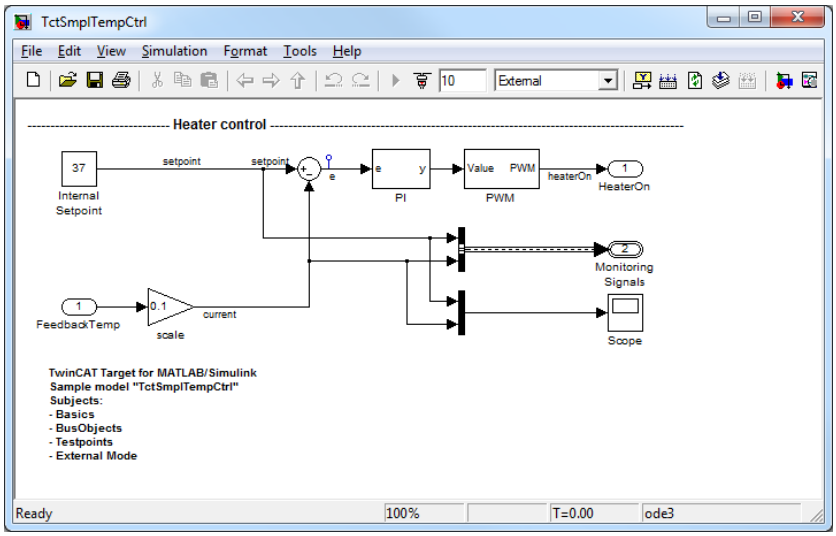
The following example extends the basics, shown in example "TemperatureController_minimal" by the following subjects:

- Parameter access [▶ 75]
- Using Bus Objects [▶ 77]
- Using Test Points [▶ 78]
- Using Referenced Models [▶ 80]
- Using External Mode [▶ 81]
- Generating TwinCAT modules from SubSystems [▶ 82]

Overview of project directory

TE1400Sample_TemperatureController (Resources/zip/27021599304187787.zip) contains all the files for this example:

TctSmplTempCtrl.mdl More advanced (but still very simple) temperature controller.



TctSmplCtrlSysPT2.mdl Simple PT2 model for the controlled system.

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

TctSmplTempCtrlParameters.mat Contains all the necessary model parameters.

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

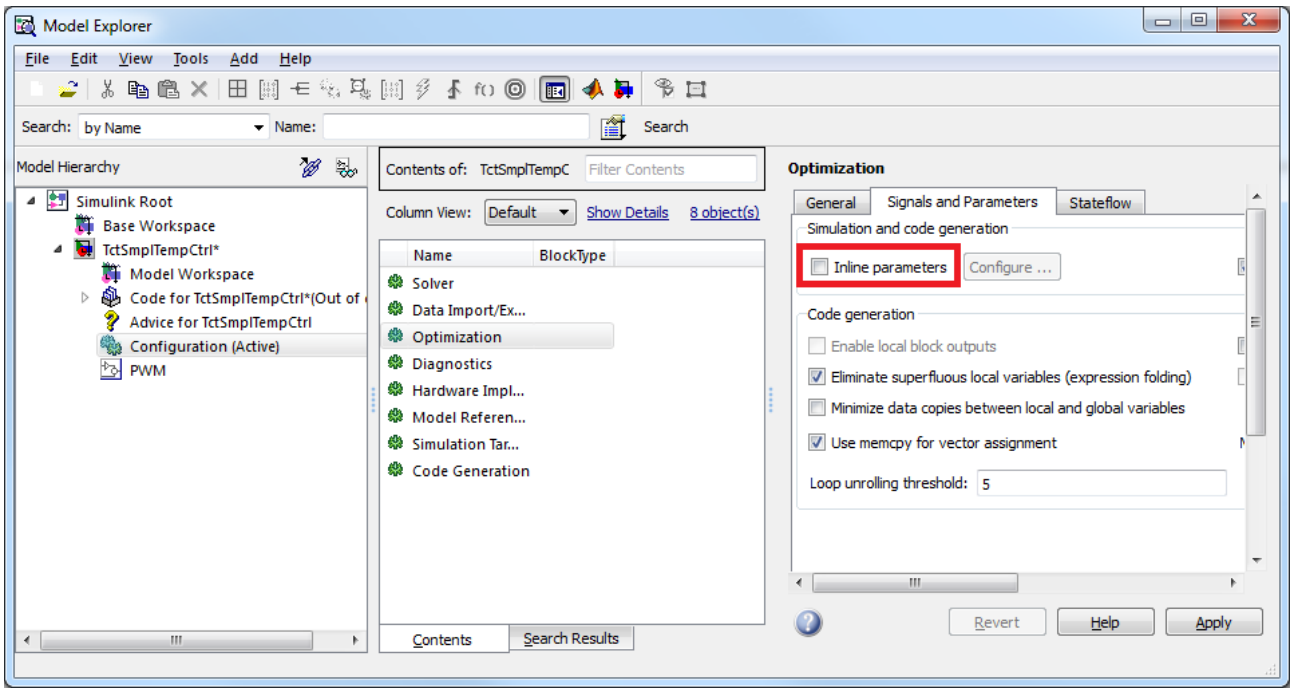
_PrecompiledTcComModules This subdirectory contains readily compiled TwinCAT modules that were generated from the enclosed Simulink models. They enable the integration of a module in TwinCAT to be tested, without having to generate the module first. They can be used in situations where a MATLAB license 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!

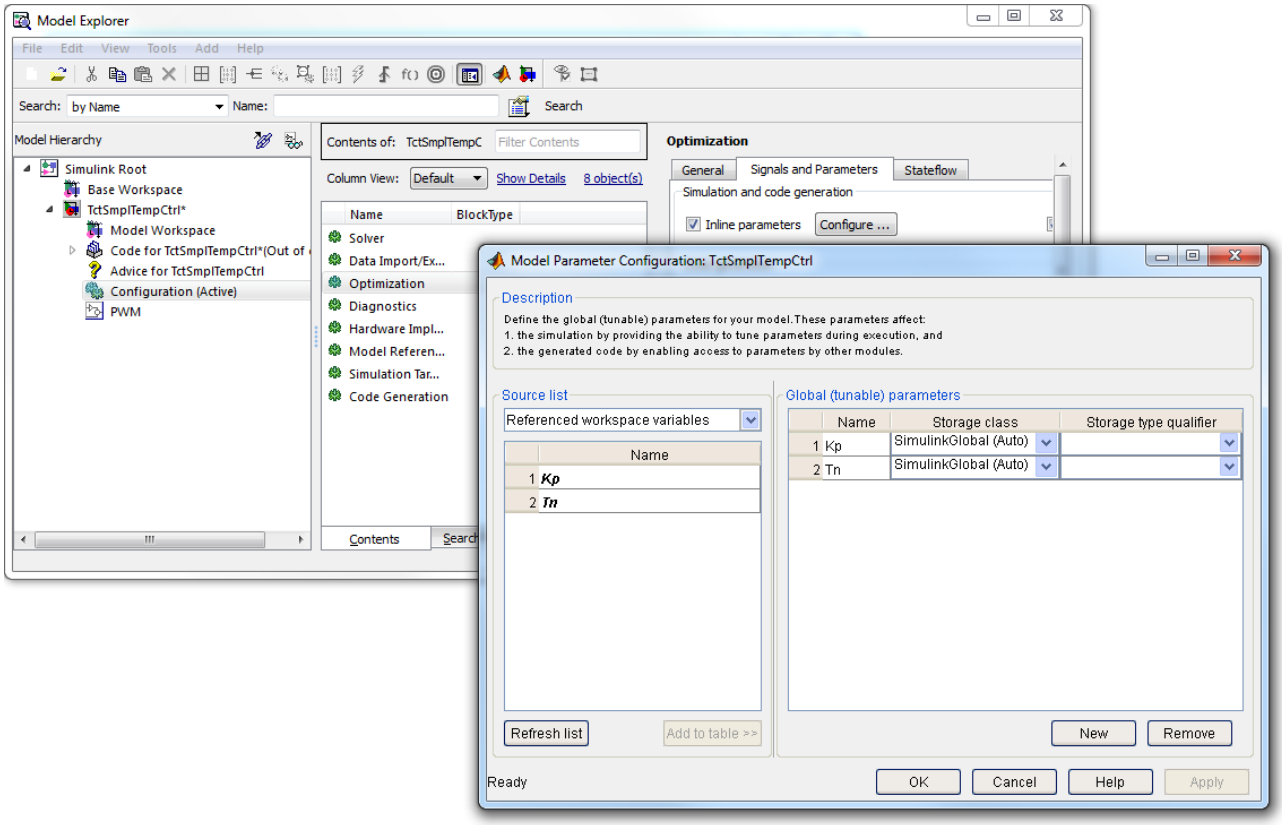
_PreviousSimulinkVersions The MDL files described above are stored in the file format of the current Simulink version. This subdirectory contains the models in the file format of elder Simulink versions.

Parameter access

TctSmplTempCtrl.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 workspace that are to remain configurable in the generated module:

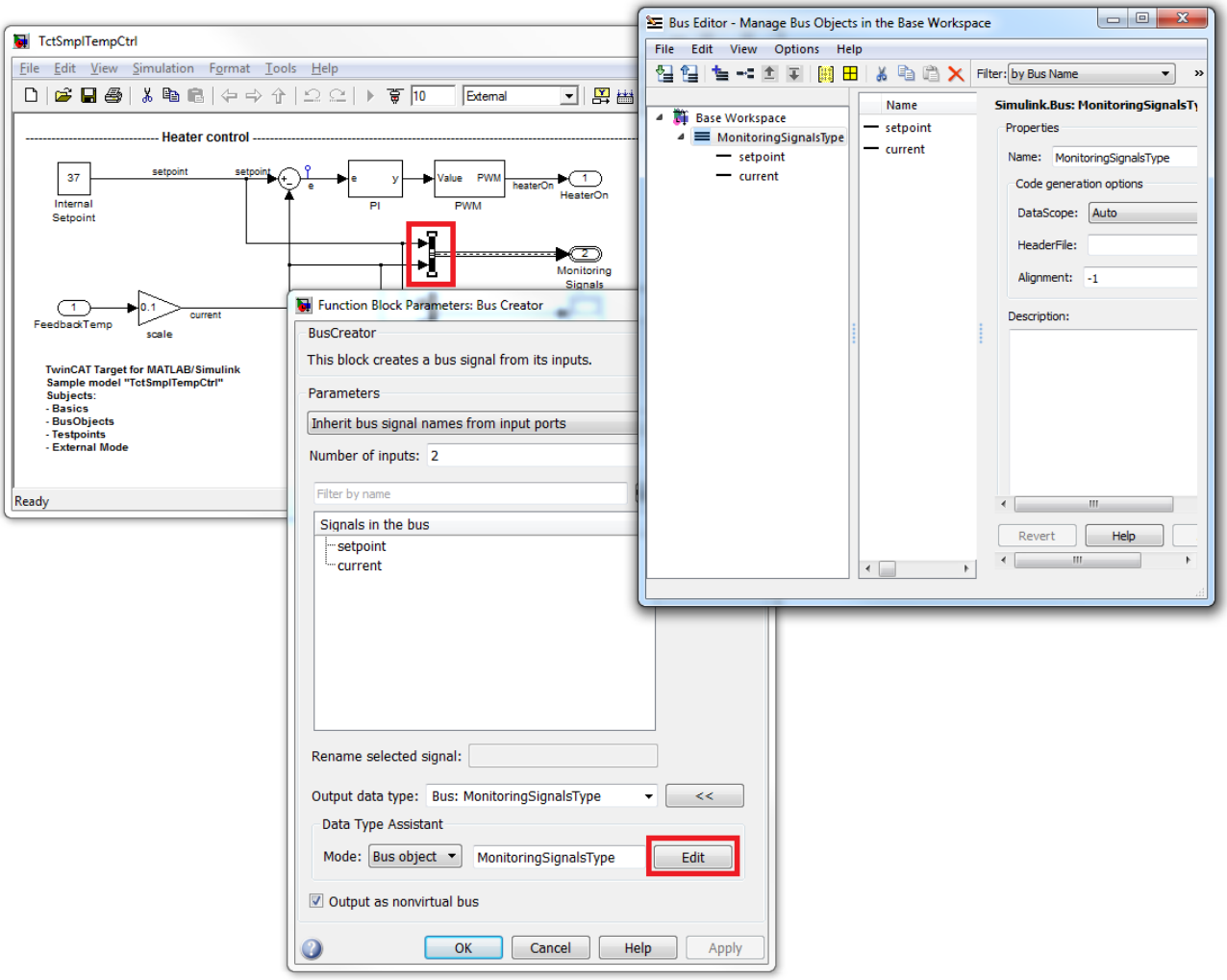


In the example shown, only the workspace variables *Kp* and *Tn* remain configurable, which means that only the Simlulink block parameters that depend on these parameters are configurable. The parameter structure is reduced to these two elements.

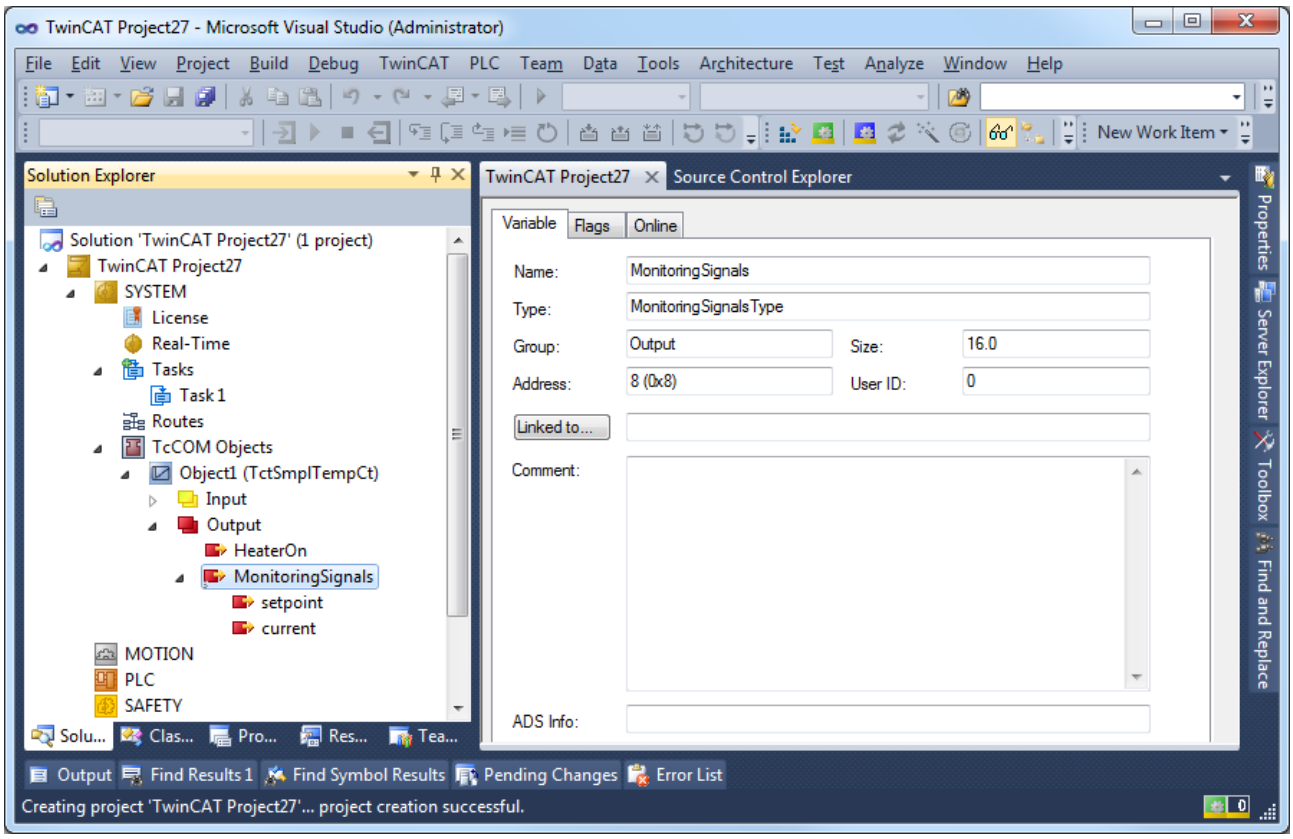
For further information on *parameter inlining* see
Simulink documentation.

Using bus objects

Simulink BusObjects enable access to TwinCAT modules generated in Simulink via structured symbols. This example contains a predefined BusObject called *MonitoringSignalsType*. It is an output structure, i.e. it assigns the received signals to a PLC module.
To start configuring a BusObject, double-click 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.

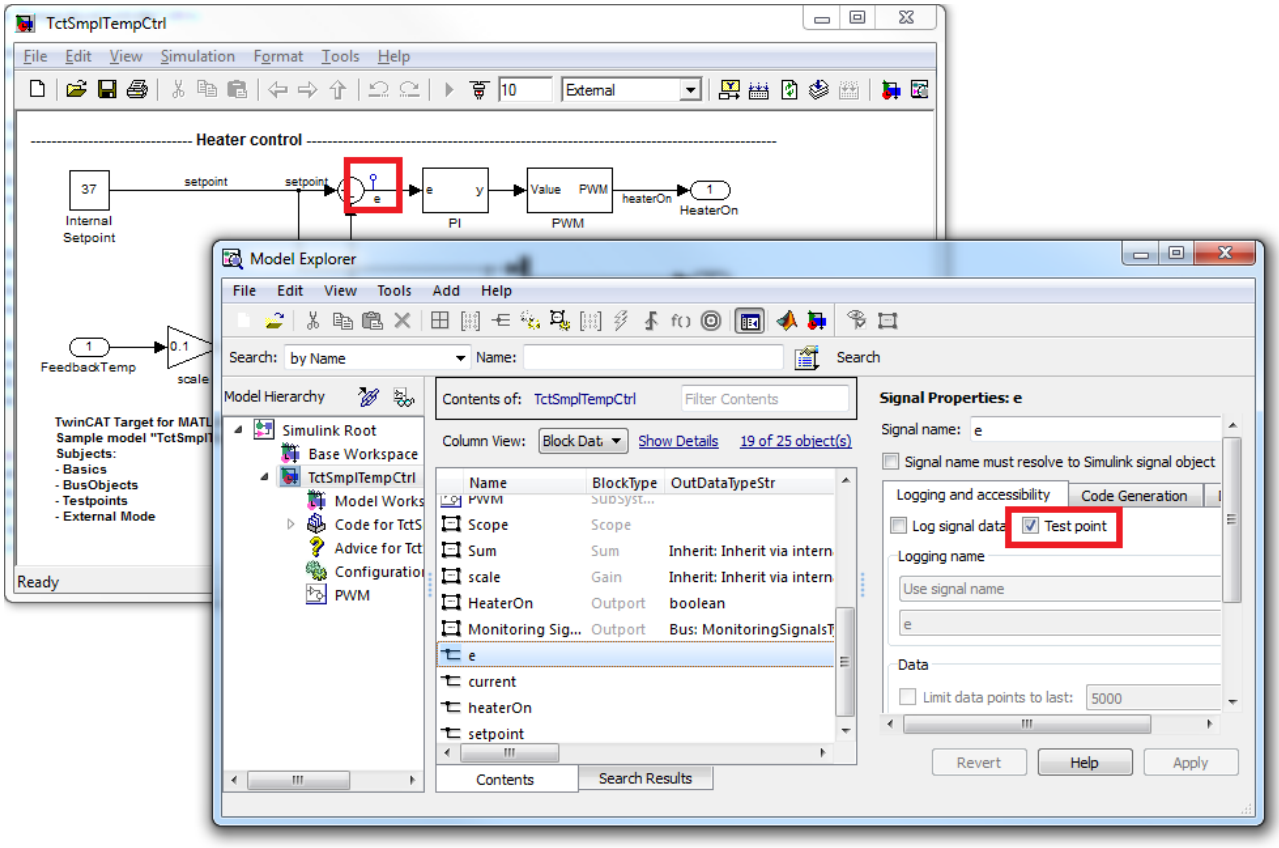


During instantiation of the generated module in a TwinCAT project, the specified *BusObject* is imported into the TwinCAT project as a global TwinCAT data type. This data type is used by the generated module itself for displaying the output structure, although it can also be used by other modules, such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

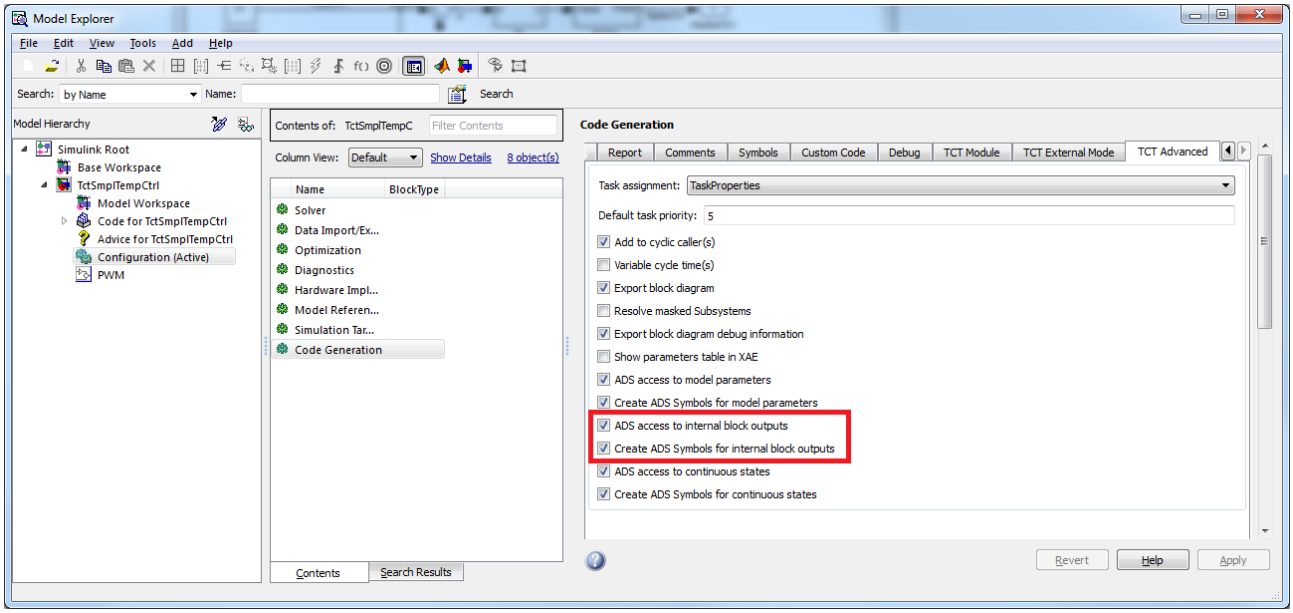


Using test points

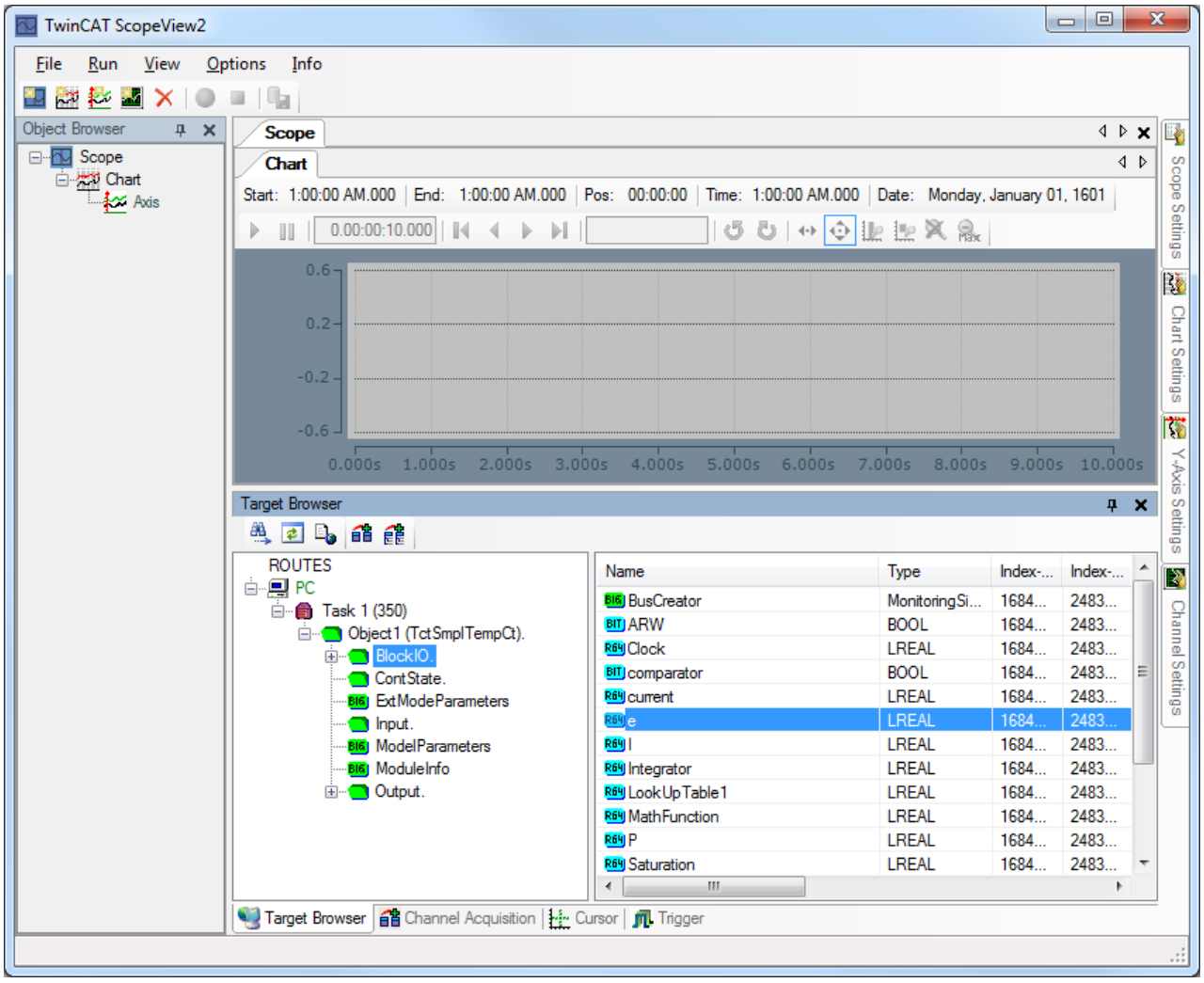
In Simulink you can specify test points on signals for monitoring by Simulink "Floating Scope", for example. If the TwinCAT Target module generator is used, signals with such test points are invariably declared as member variable for the generated TwinCAT module. This enables ADS access to the signal. For further information on test points see Simulink documentation.
In this example, the **Model Explorer** is used to define a test point for the control deviation *e*:



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

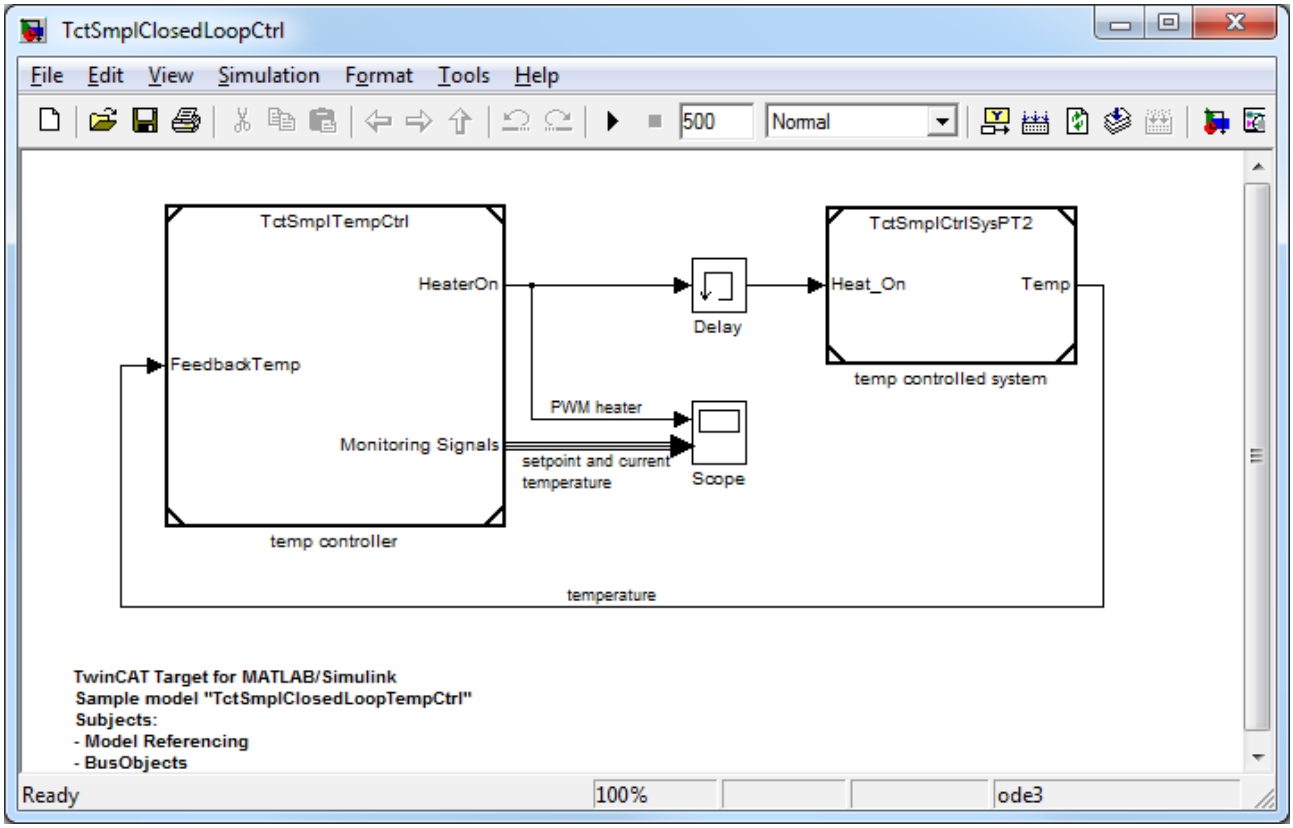


In this way you can use TwinCAT Scope View to access the signal with test points and some other block output variables when the generated TwinCAT module is executed.



Using referenced models

Open the model *TctSmplClosedLoopCtrl.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:

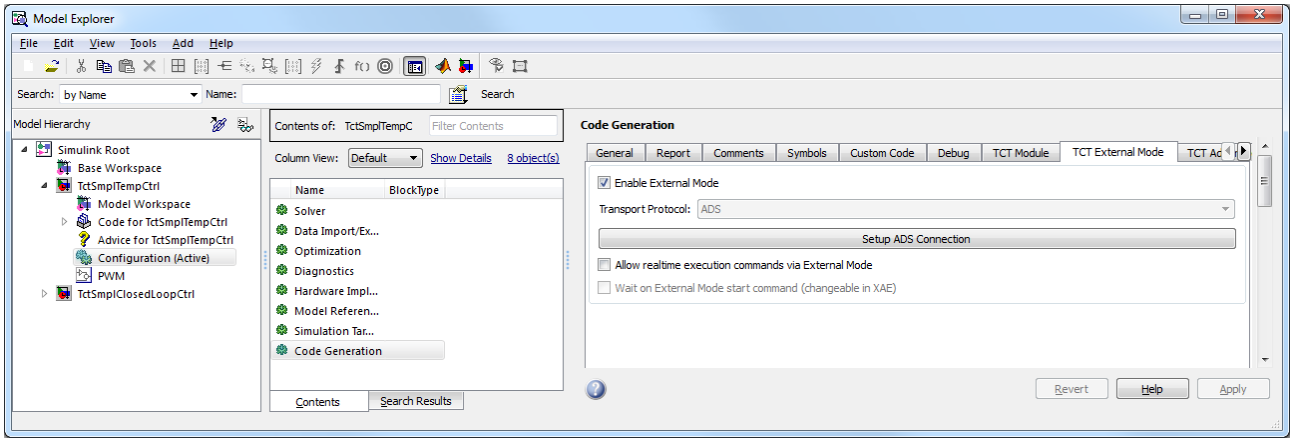
A model can be referenced several times in one or several higher-level models. In this way, the models can be divided into reusable functional units, similar to text programming languages, where the code is structured into functions or methods. This improves the readability of complex models.
The generated 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:

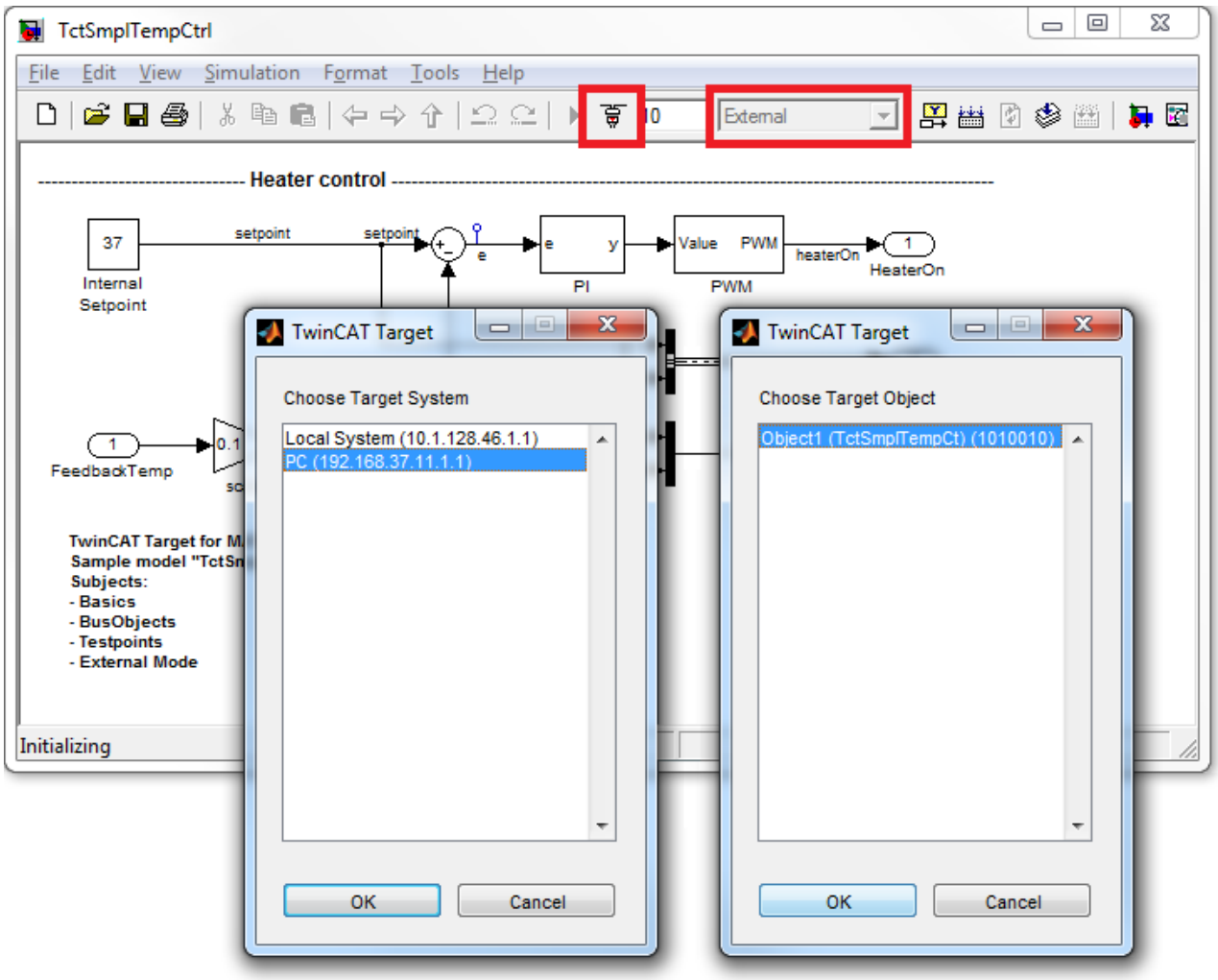
The control loop model of this example can only be compiled into a TwinCAT module with a valid TwinCAT Target license (TE1400). Otherwise, this model exceeds the limits for unlicensed models.

Using External Mode

The temperature controller model *TctSmplTempCtrl.mdl* was preconfigured to allow ExternalMode connections:



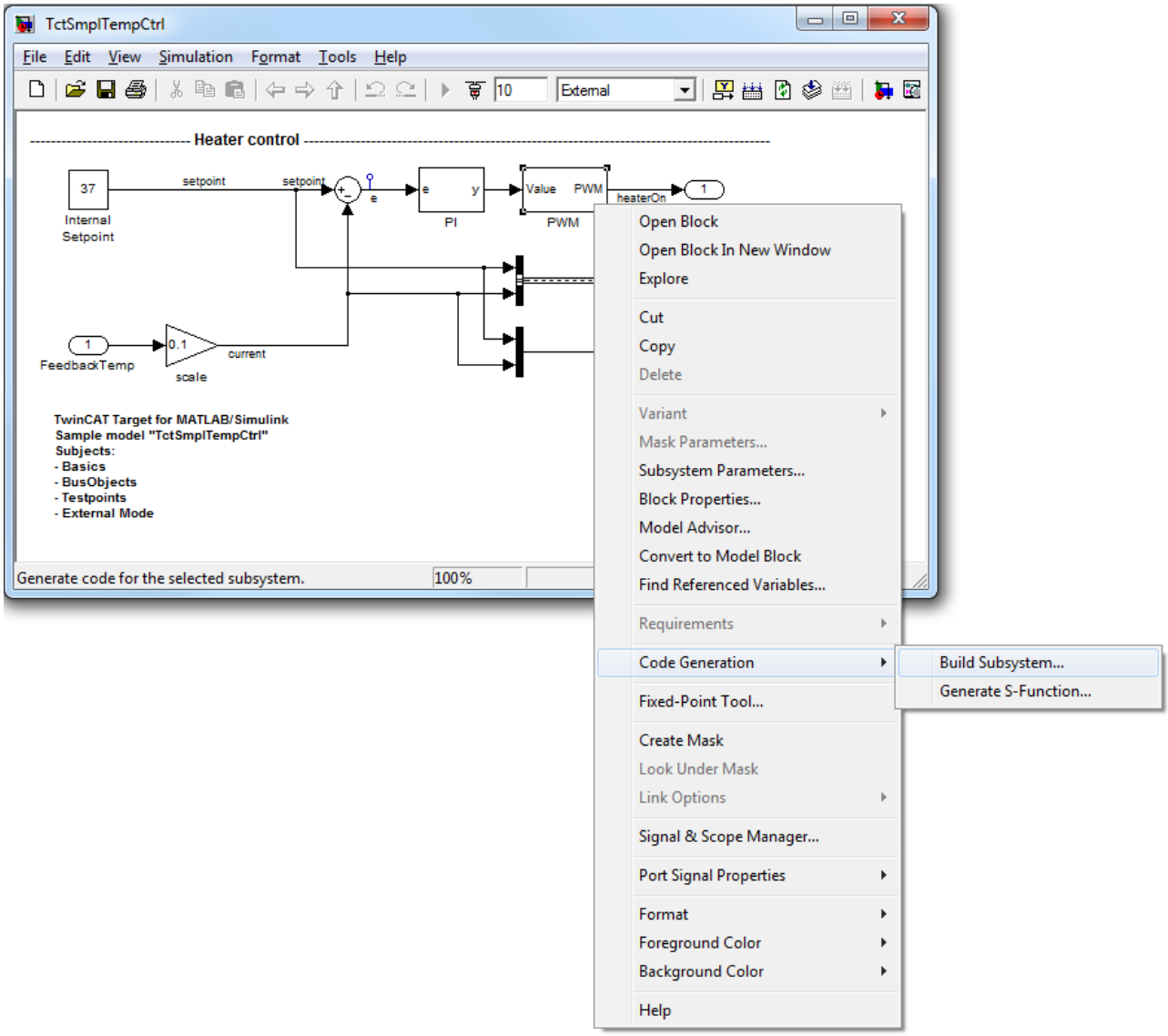
Due to this configurations, you can connect to the generated temperature controller via ExternalMode, using the **Connect to Target** icon in the Simulink toolbar. Of course the module has to be generated and started on a TwinCAT system before and an ADS route has to be configured between your engineering system and the appropriate target system. Your will see some dialogs, helping to navigate to the desired module instance:



Now you can use the **Scope** block in Simulink to monitor the real time signals of the generated and now connected TwinCAT module. Also you can change e. g. the value of the **Internal Setpoint** block. When the parameter change is confirmed, it is directly downloaded to the target module. However this works only for tunable parameters and only if the model parameters are not inlined (see "Parameter access [▶ 75]").

Generating TwinCAT modules from subsystems

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



11.3 SFunStaticLib

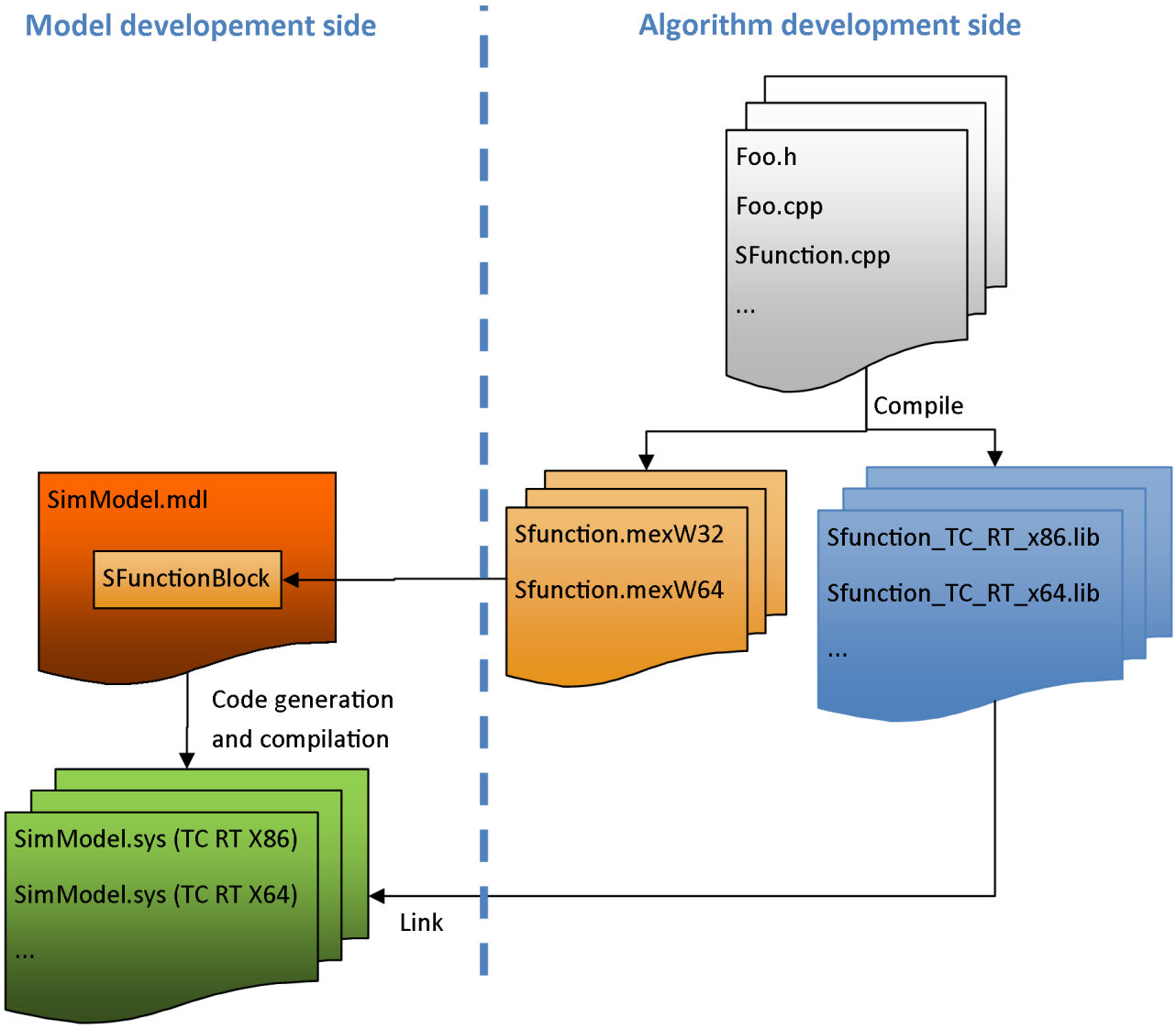
Use cases

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

TE1400Sample_SFunStaticLib (Resources/zip/27021599304133259.zip) contains all the files necessary to reproduce this example:

TctSmplSFunStaticLib.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".
_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 additionally contains the models in the file format elder Simulink versions.

Build SFunction and appropriate static link libraries

Build requirements

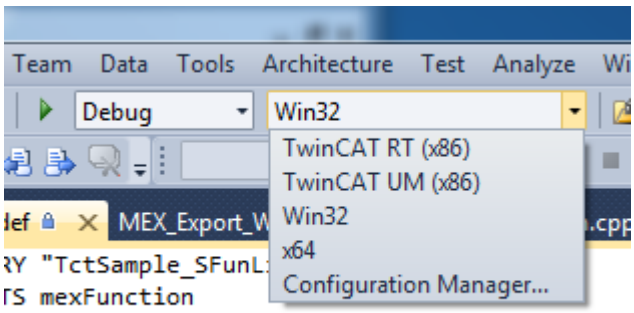
It is recommended to have TwinCAT 3 installed on your system to build the binaries, but not required. Requirements are:

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.

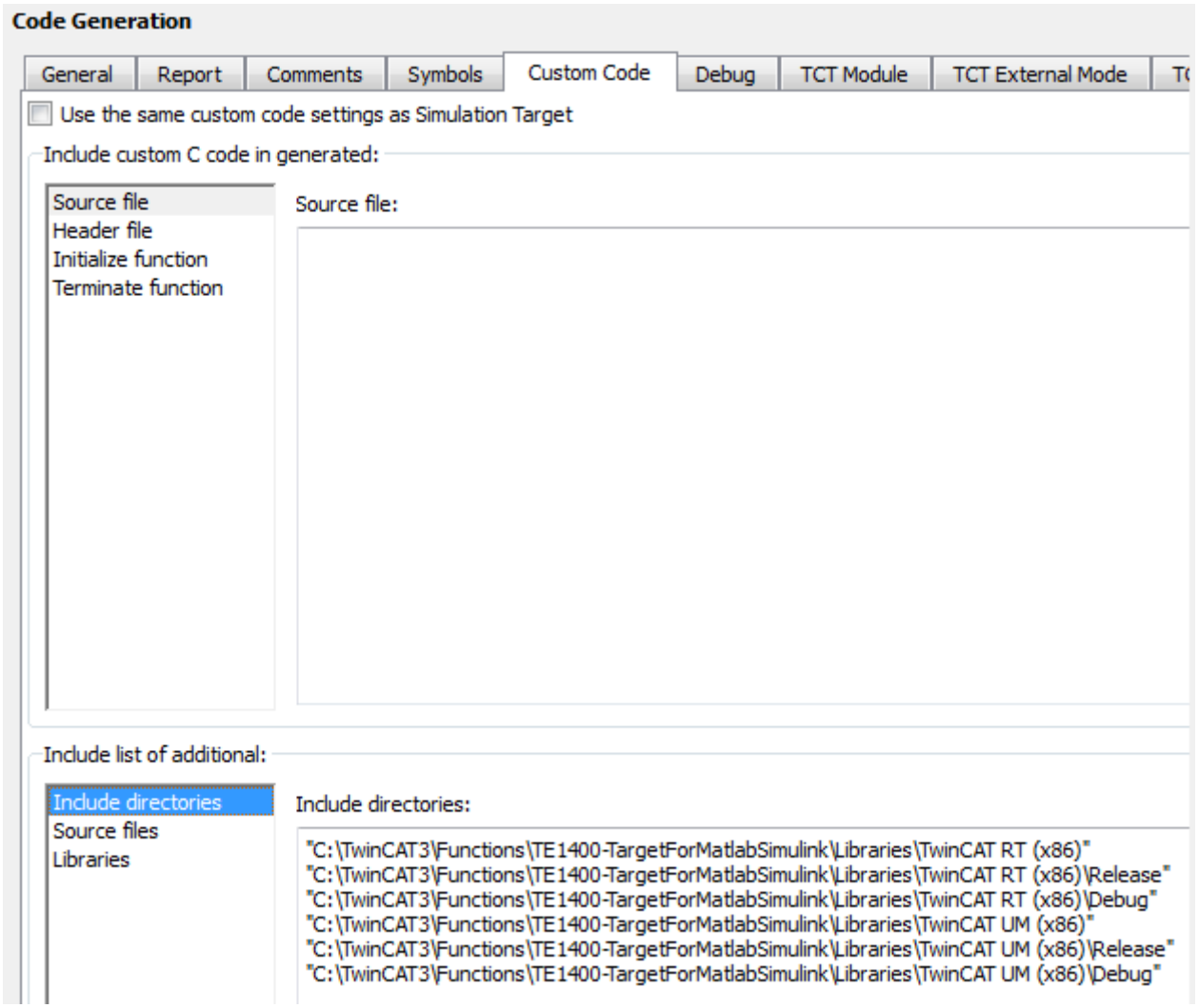
Run simulation

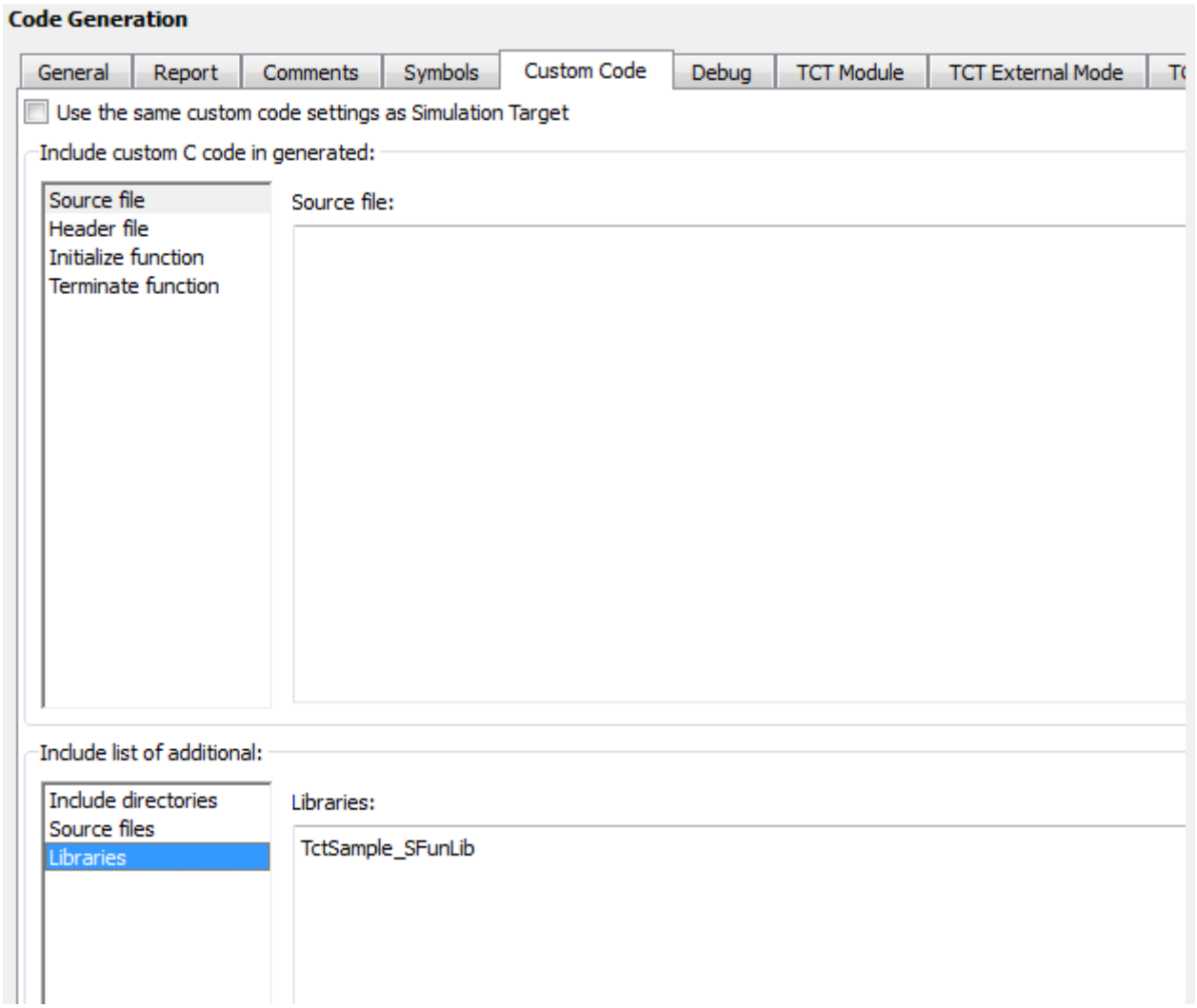
To check if everything works, open "TctSmplSFunWrappedStaticLib.mdl" and start simulation. If the simulation starts without error messages, everything is prepared as needed.

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
Use the same custom code settings as Simulation Target
Include custom C code in generated:
Source file
Header file
Initialize function
Terminate function
Source file:
Include list of additional:
Include directories
Source files
Libraries
Include directories:
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Release"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT RT (x86)\Debug"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Release"
"C:\TwinCAT3\Functions\TE1400-TargetForMatlabSimulink\Libraries\TwinCAT UM (x86)\Debug"



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*

11.4 SFunWrappedStaticLib

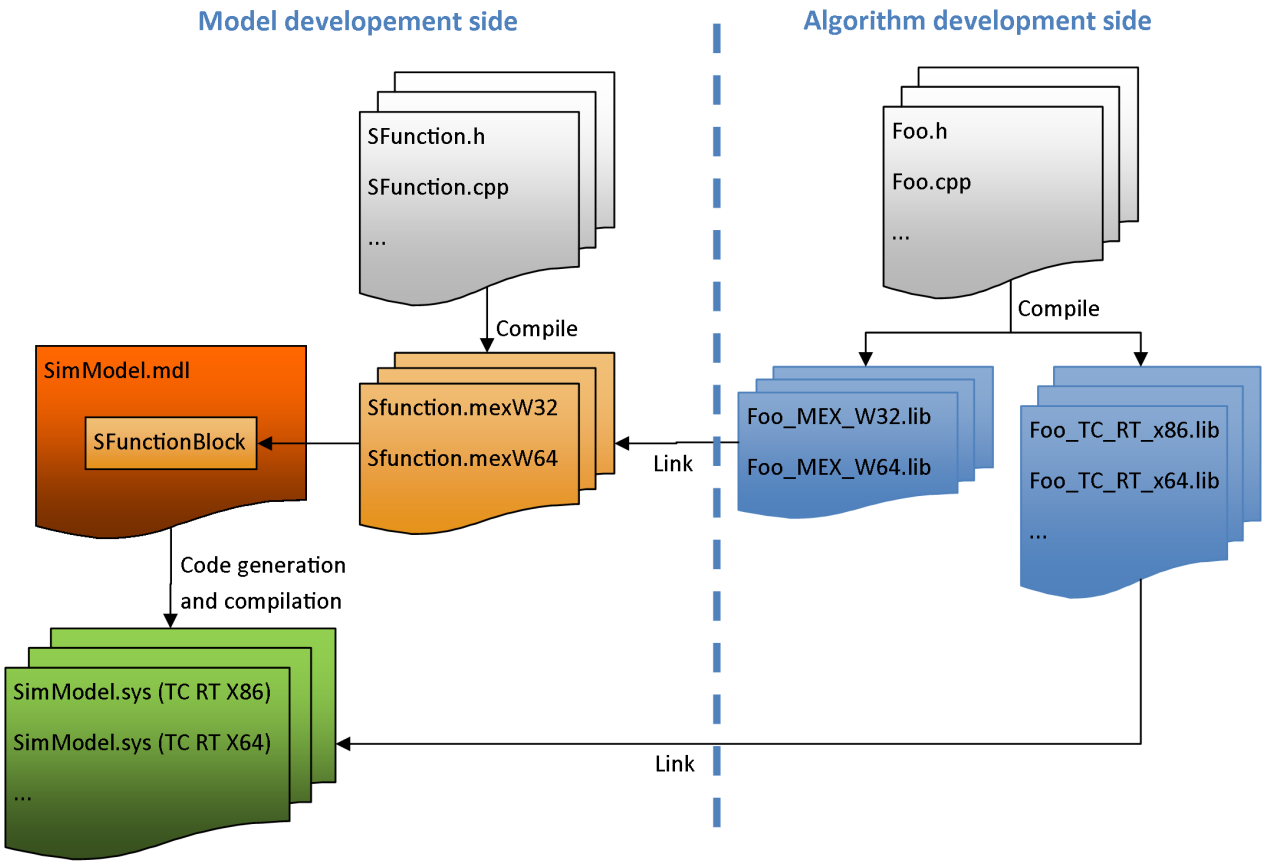
Use cases

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 how to configure the module generator, when using SFunctions, which depend on static link libraries. Prerequisite to this kind of code integration is, that the appropriate library is available for all TwinCAT platforms, the module shall be build for.
The following graphic illustrates a typical workflow for using third party algorithms inside an own Simulink model:



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

- create dependent libraries
- deliver such libraries (e. g. to a customer)
- use such libraries in an own model

Overview of project directory

TE1400Sample_SFunWrappedStaticLib (Resources/zip/27021599304134923.zip) contains all the files necessary to reproduce this example:

TctSmplSFunWrappedStaticLib.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

Build SFunction and appropriate static link libraries

Build requirements

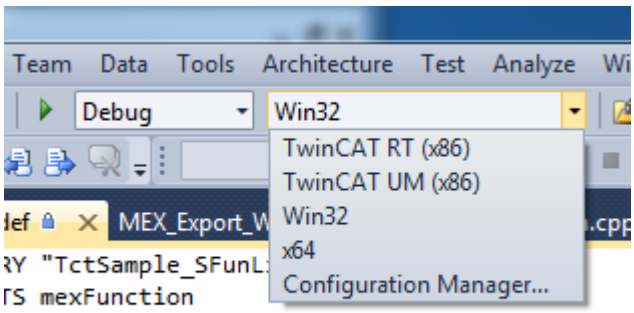
It is recommended to have TwinCAT 3 installed on your system to build the binaries, but not required. Requirements are:

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 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

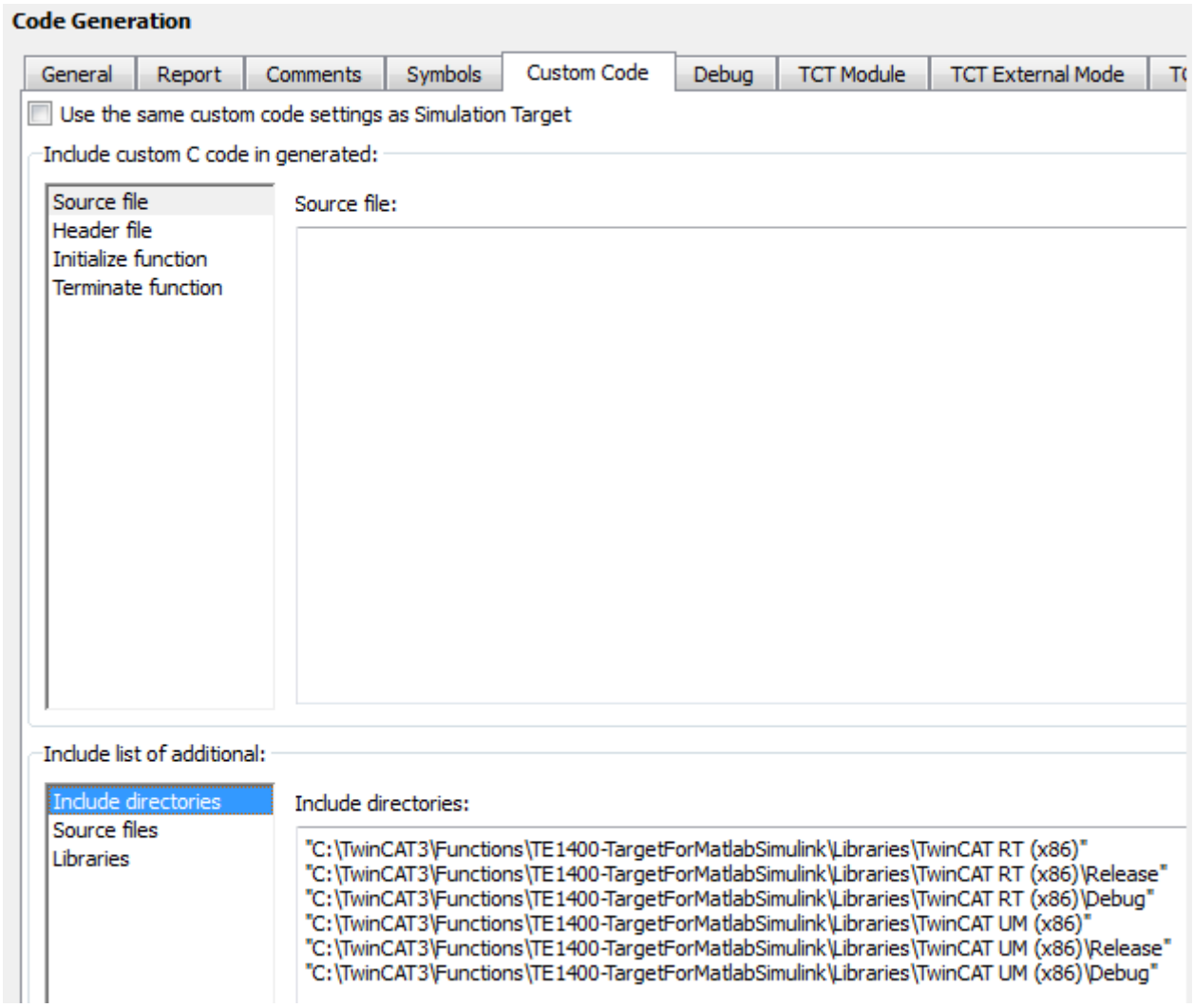
Before the SFunction can be used inside the Simulink model, it has to be built, too. Of course this can be done manually, as in any other SFunction. However, the MEX compiler has to be advised to link the static library to the SFunction.
When executing *BuildLibsAndSFunction.m*, this is also done automatically. Afterwards the file "SFunStaticLib.mexw32" should be located inside your working folder.
To check if everything works, open "TctSmplSFunWrappedStaticLib.mdl" and start simulation. If the simulation starts without error messages, everything is prepared as needed.

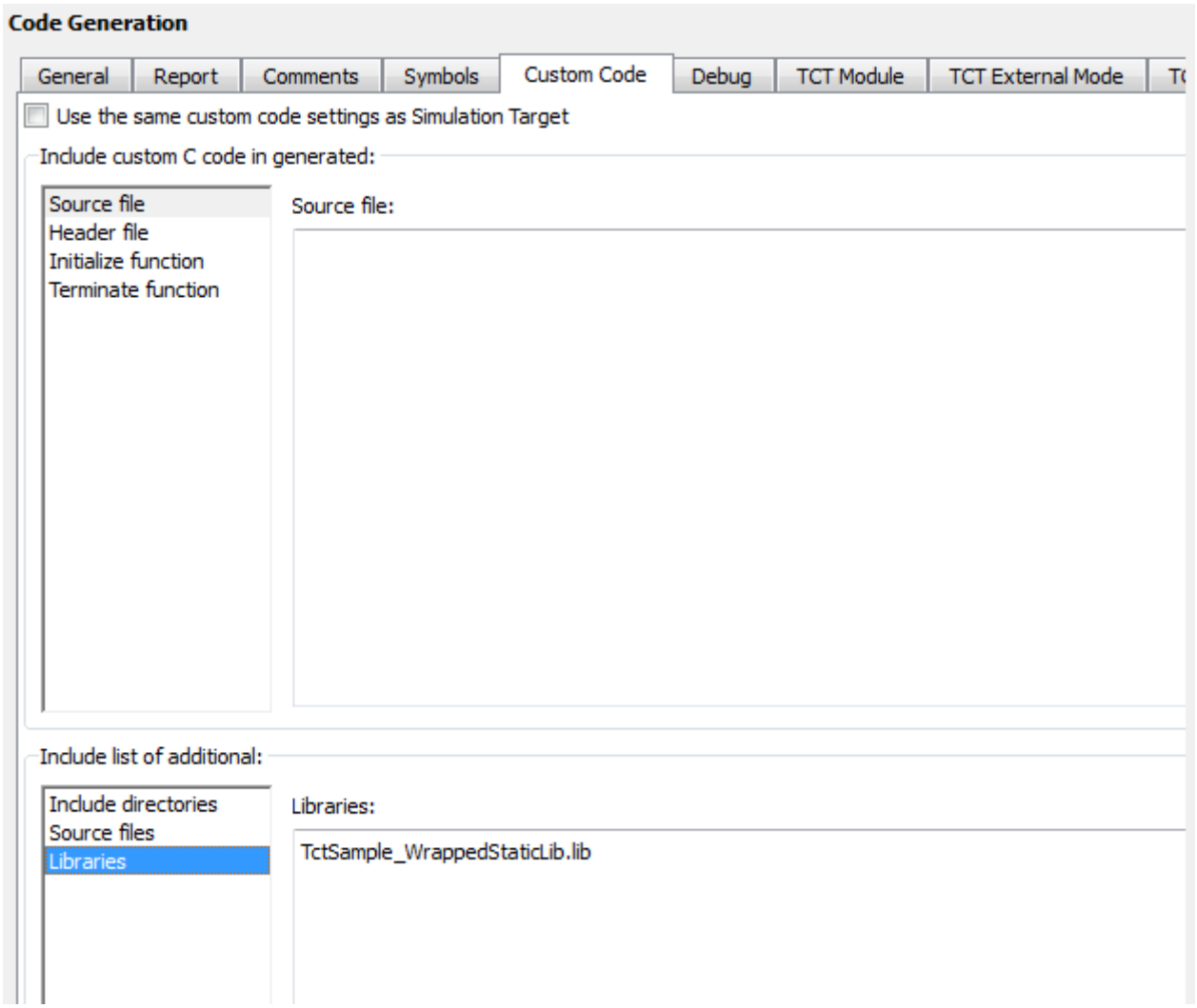
Generate TwinCAT module

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 [▶ 12].
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.





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*

11.5 Module generation Callbacks

Examples for module generation callbacks [▶ 22]:

Example	Topics	Description
Packaging module files into ZIP archives [▶ 94]	• PostPublish callback • Archiving generated module files	This simple example illustrates the automatic archiving of generated module files.

11.5.1 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('Twin-
Cat3Dir'),'CustomConfig','Modules',cgStruct.ModuleName))
```