# Table of contents

1 Foreword .............................................................................................................................. 5  
  1.1 Notes on the documentation ............................................................................................... 5  
  1.2 Safety instructions ............................................................................................................... 6  
  1.3 Notes on information security ............................................................................................. 7  

2 Overview ................................................................................................................................ 8

3 From version 2.x.xxxx.x .......................................................................................................... 10  
  3.1 Installation ........................................................................................................................... 10  
  3.2 Licenses ................................................................................................................................ 12  
  3.3 Setting up driver signing ....................................................................................................... 13  
  3.4 Quick start ............................................................................................................................ 15  
  3.5 TwinCAT Library in Simulink® ............................................................................................ 21  
  3.6 Overview of automatically generated files ......................................................................... 24  
  3.7 Parameterization of the code generation in Simulink .......................................................... 28  
     3.7.1 Creation of versioned drivers ....................................................................................... 29  
     3.7.2 Creating TMX archives ............................................................................................... 33  
     3.7.3 Bundling of several models in one TwinCAT driver ...................................................... 35  
     3.7.4 Configuration of the TMX file properties ...................................................................... 36  
     3.7.5 Working with callbacks ................................................................................................ 37  
  3.8 Application of modules in TwinCAT ................................................................................... 38  
     3.8.1 Working with the TcCOM module ................................................................................ 39  
     3.8.2 Working with the PLC library ..................................................................................... 55  
     3.8.3 Debugging ..................................................................................................................... 58  
  3.9 FAQ ...................................................................................................................................... 60  
     3.9.1 Build of a sample fails .................................................................................................... 60  
     3.9.2 Problems with the block diagram representation in TwinCAT XAE ......................... 61  
     3.9.3 Can I use TE1400 version 1.2.x and version 2.x at the same time? ......................... 61  
     3.9.4 What is the difference between "Build" and "Generate code"? ...................................... 62  
     3.9.5 I can't change the parameters of a module in TwinCAT ............................................... 62  
     3.9.6 Mapping is lost with Reload TMI/TMC ........................................................................ 62  
  3.10 Samples ............................................................................................................................. 63  
     3.10.1 TwinCAT Automation Interface: use in MATLAB® ................................................... 64  
     3.10.2 Integrating the block diagram controls ....................................................................... 66  

4 Up to version 1.2.xxxx.x ........................................................................................................ 69  
  4.1 Installation ........................................................................................................................... 69  
  4.2 Licenses ................................................................................................................................ 71  
  4.3 Quickstart ............................................................................................................................. 72  
  4.4 TwinCAT Library in Simulink® ........................................................................................... 75  
  4.5 Parameterization of the code generation in Simulink ............................................................ 78  
     4.5.1 Module generation (Tc Build) ...................................................................................... 79  
     4.5.2 Data exchange (Tc Interfaces) ..................................................................................... 83  
     4.5.3 External mode (Tc External Mode) .............................................................................. 87  
     4.5.4 Advanced settings (Tc Advanced) ............................................................................... 90  
  4.6 Application of modules in TwinCAT .................................................................................... 96
4.6.1 Parameterization of a module instance ................................................................. 96
4.6.2 Executing the generated module under TwinCAT .................................................. 98
4.6.3 Calling the generated module from a PLC project .................................................. 101
4.6.4 Using the ToFile block ......................................................................................... 106
4.6.5 Signal access via TwinCAT 3 Scope ......................................................................... 112
4.7 FAQ ............................................................................................................................. 113
4.7.1 Does code generation work even if I integrate S-Functions into my model? ............ 113
4.7.2 Why do FPU/SSE exceptions occur at runtime in the generated TwinCAT module, but not in the Simulink model? ...................................................................................... 113
4.7.3 After updating TwinCAT and/or TE1400 I get an error message for an existing model. 114
4.7.4 Why do the parameters of the TcCOM instance not always change after a "Reload TMC/ TMI" operation? ................................................................. 113
4.7.5 After a "Reload TMC/TMI" error "Source File <path> to deploy to target not found ..... 115
4.7.6 Why do I have a ClassID conflict when I start TwinCAT?........................................ 116
4.7.7 Why can the values transferred via ADS differ from values transferred via output mapping? .................................................................................................................. 116
4.7.8 Are there limitations with regard to executing modules in real-time? ...................... 116
4.7.9 Which files are created automatically during code generation and publishing? .... 117
4.7.10 How do I resolve data type conflicts in the PLC project?........................................ 118
4.7.11 Why are the parameters of the transfer function block in the TwinCAT display not identical to the display in Simulink? ................................................................. 119
4.7.12 Why does my code generation/publish process take so long? ............................... 119
4.8 Examples ..................................................................................................................... 120
4.8.1 TemperatureController_minimal ........................................................................... 120
4.8.2 Temperature Controller ......................................................................................... 126
4.8.3 SFunStaticLib .......................................................................................................... 134
4.8.4 SFunWrappedStaticLib ......................................................................................... 140
4.8.5 Module generation Callbacks ............................................................................... 145
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 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®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® 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:


with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="DANGER" /></td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="image" alt="WARNING" /></td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="image" alt="CAUTION" /></td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><img src="image" alt="NOTE" /></td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

**Tip or pointer**
This symbol indicates information that contributes to better understanding.
1.3 Notes on information security

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

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

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

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at [https://www.beckhoff.com/secinfo](https://www.beckhoff.com/secinfo).
Overview

TE1400 TwinCAT Target for Simulink®

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

Areas of application and application examples

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

- Rapid Control Prototyping
- Real-time simulation
- SiL (Software in the Loop) simulation
- HiL (Hardware in the Loop) simulation
- Model-based design
- Model-based monitoring

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

- **Example 1: Rapid Control Prototyping**
  During the simulation development stage in Simulink®, a controller is implemented as a Simulink® model, which is integrated into the simulation model of the control loop via *model referencing*. This enables the closed control loop to be designed and tested in a simulation (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® function blocks are used as inputs and outputs, they can be used in the higher-level Simulink® model as well as in the module generated later in TwinCAT.

- **Example 1a: Real-time simulation of a controlled system**
  The controlled system is also implemented as a Simulink® model, which is integrated into the model of the closed control loop via *Model Referencing*. The TcCOM module generated from this is used to perform a real-time simulation, in which a controller implemented in IEC61131-3, C++ or Simulink® can be tested.

- **Example 2: Real-time simulation of a machine/Virtual commissioning**
  A TcCOM module is generated from a machine model created in Simulink®. This can be used to test a PLC program in real-time, before the actual machine is connected (virtual commissioning). Depending on the configuration, SiL or HiL simulations can be performed in this way. See also Overview.

- **Example 2a: SiL simulation of plant components**
  According to VDI/VDE 3693 Part 1, Software in the loop (SiL) is defined as a stage following MiL simulation, in which the control code is available as series code. The series code can be executed in an emulated controller, for testing against a system simulation model. According to this definition, there are two options for a SiL simulation of systems (components) with TwinCAT:
    - The system model remains in Simulink® and uses ADS to communicate with the series code, which is executed in the TwinCAT runtime. See also TE1410 Interface for MATLAB Simulink.
    - The system model is also compiled into a TcCOM module and executed in real-time (see example 1a).

- **Example 2b: HiL simulation of system components**
  According to VDI/VDE 3693 Part 1, Hardware in the loop (HiL) is defined as an advanced testing stage, in which the actual target control code is tested on an actual controller against a system model. The latter is executed in a simulation tool, which acts as a bus device and therefore uses the actual communication networks of the automation system for communication with the actual controller. Based on this definition, the model of the system or the system components is converted to TcCOM modules and executed on a second Industrial PC, taking into account the real-time requirements. The
function Overview is used to configure this IPC such that it makes the mirrored process image available to the actual controller. In this way, it is possible to use the actual controller and the actual configuration to communicate with the "simulation IPC" in hard real-time.

- **Example 3: Model-based monitoring of system components**
  In many cases, measured variables are of interest that are not directly accessible or would result in excessive effort/costs. By using a physically representative model with measurable input variables, non-measurable variables can still be determined. An example is temperature measurement at locations that are inaccessible, such as the permanent magnet temperature in an electric motor. Based on a thermal model of the motor, the temperature can be estimated by means of secondary parameters such as electric current, rotational speed and cooling temperature.

**Further Information**

- **Technical short videos**
  - TwinCAT Target for Simulink

- **Product descriptions**
  - [https://www.beckhoff.com/TE1400](https://www.beckhoff.com/TE1400)

- **Customer application videos**
  - Overview of customer applications
  - Success Story Vintecc bv
  - Success Story Magway

- **Website for MATLAB® and Simulink® with TwinCAT 3**: [http://www.beckhoff.com/matlab](http://www.beckhoff.com/matlab)
3 From version 2.x.xxxx.x

- TE1400 Target for Simulink® versions lower than 1.2.xxxx.x support MATLAB R2010b to MATLAB R2019a.
- TE1400 Target for Simulink® versions higher than 2.x.xxxx.x support MATLAB R2019a and higher.
- The installations for both versions can be used in parallel on one engineering system.
- Compatibility of the created modules: see Mapping is lost with Reload TMI/TMC.[Page 62].

3.1 Installation

System requirements

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

On the engineering PC

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

On the runtime PC

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

Built objects can be easily forwarded

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

Installation

- Install one of the supported Visual Studio versions, if not already installed. Note the installation of the option Desktop development with C++.

1. Start TwinCAT 3 XAE or Full 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. TwinCAT XAE Shell or Visual Studio without C++ option), you first have to install a suitable Visual Studio version (install C++ option, if necessary). Then run TwinCAT 3 Setup to integrate TwinCAT 3 into the new (or modified) Visual Studio version.

2. If you do not have a MATLAB® installation on your system, install it. The order in which MATLAB® was installed is irrelevant.
3. Start **TwinCAT Tools for MATLAB® and Simulink®** Setup to install the TE1400.

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

4. Start MATLAB® as administrator and run
   
   `%TwinCAT3Dir%\Functions\TE14xx-ToolsForMatlabAndSimulink\SetupTE14xx.p` in MATLAB®.

   - A setup window opens. See the following section.

**Setting up the software**

**Run SetupTE14xx.p**

After executing the p-file, a dialog opens in which you can save general default settings that will then apply to the system. You can make the settings directly or make/change them at a later time.

![Select common module generation default settings](image)

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

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

**The following setting options are available in the dialog:**

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

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

There are the entries:

- **VendorName** "TE140x Module Vendor"
- **GroupName** "TE140x\MATLAB Modules" for MATLAB® and "TE140x\Simulink Modules" for Simulink®.
Under the tab **Build** you can store a default certificate for driver signing. For the procedure to create and use a driver signing certificate, see section **Setting up driver signing [13]**.

To change the default settings, you can access the dialog with `TwinCAT.ModuleGenerator.Settings.Edit` in the MATLAB® Console. Here you are also offered additional entries that you can store as default.

## 3.2 Licenses

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

### Engineering license

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

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

### Restrictions in the demo version

Without valid TE1400 license models are allowed with maximum:

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

The TC1320 or TC1220 licenses with included PLC license are required to start a TwinCAT configuration with a TwinCAT object generated from Simulink®. Without activated license, the module and consequently the TwinCAT system cannot be started.

TC1320 contains the license for executing TwinCAT C++ objects as well as objects created via the Target for Simulink® and via the Target for MATLAB®.

TC1220 adds a PLC license to the above list of TC1320.

It is possible to create a 7-day trial license for the runtime licenses, which allows initial tests without purchasing the license.

3.3 Setting up driver signing

Create an OEM certificate level 2

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

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

- General documentation on OEM certificates
- Application-related documentation for tmx driver signing

The most important facts in brief:

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

Use of an OEM level 2 certificate for driver signing

There are four possible variants for signing tmx drivers.

1. You can set a default certificate on an engineering PC, which is always used for TwinCAT C++, Target for MATLAB® and Target for Simulink®, unless you explicitly specify a different certificate.
2. You can set a default certificate on an engineering PC that is always used for Target for MATLAB® and Target for Simulink® unless you explicitly specify a different certificate.
3. You can explicitly name a certificate for each build operation.
4. You can build without a certificate and sign afterwards with the TcSignTool.

For Variant 1 use a Windows environment variable. Create a new environment variable at User > Variables with:

Variable: TcSignTwinCatCertName

Value: Name of the desired certificate
(Available certificates are located at TwinCAT\3.1\CustomConfig\Certificates).
For **variant 2** open the above **Common Settings dialog** with **TwinCAT.ModuleGenerator.Settings.Edit** and name the default certificate **build > Certificate name for TwinCAT signing**. This certificate is stored in your user directory as default and is used by all MATLAB® versions on your system as default.

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

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

**Target for MATLAB®: Property** `SignTwinCatCertName`

For **variant 4** you can use the TcSignTool. The TcSignTool is a command line program located in the path `C:\TwinCAT\3.x\sdk\Bin`. With `tcsigntool /?` or `tcsigntool sign /?` you get help how to use the tool concretely.

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

For **variants 1 to 3**, the associated password must be stored in the system in addition to specifying the certificate with the TcSignTool. For security reasons, the password should not be entered in the source code in the Simulink® model or in the MATLAB® code. With the TcSignTool you can store passwords belonging to your certificates encrypted in the registry of the Windows operating system.

The storage of the password is carried out with the following parameters:

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

The password is deleted with the following parameters:

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

The unencrypted password is stored under: `HKEY_CURRENT_USER\SOFTWARE\Beckhoff\TcSignTool\`

---

### Operating TcSignTool from MATLAB®

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

---

### Behavior of the TwinCAT runtime

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

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

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

---

### Registry file is only dependent on the OEM certificate

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

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

```
bcdedit /set testsigning yes
```
If you are using a countersigned OEM certificate, this step is not necessary.

3.4 Quick start

Starting with a simple Simulink® model

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

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

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

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

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

**Beginner video**

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

The configuration steps in Simulink®

1. Select a fixed-step solver. To do this, go to the **Configuration Parameters** of the model.
From version 2.x.xxxx.x

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

3. Save your changes in the Simulink® model.

Insert TcCOM in TwinCAT
5. Open TwinCAT (TwinCAT XAE or TwinCAT in a Visual Studio environment).
6. Instantiate a new TcCOM object.
7. Select the desired object.

8. Create a cyclic task.

9. Assign the created task to your TcCOM instance.

Note that the cycle time of the task and the SampleTime in Simulink® (here 5 ms) match.
From version 2.x.xxxx.x

10. Activate the configuration.

Configure and link TcCOM instance

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

In the Parameters (Init) area of the TcCOM instance, you can optionally configure the instance differently than specified when it was created from Simulink®.
For example, you can set the parameter $K_p$ to "52" here. The TcCOM module would then use this value as the startup value for this instance.

**Insert as PLC function block**

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

If these options have not been set, this can be done subsequently without using Simulink® and the TwinCAT Target for Simulink®. See Create PLC library [p. 56].

**Brief overview of action steps PLC library/function block**

- Create PLC in TwinCAT:

- Add PLC library:
• Select PLC library and view content:

• Use the function block from the library in the PLC:
3.5 TwinCAT Library in Simulink®

In Simulink®, TwinCAT-specific input and output function blocks can be used (not mandatory!). These input and output function blocks are then offered in TwinCAT as inputs and outputs. 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 (TC Module Input) and output (TC Module Output) function blocks provided by Beckhoff, you will benefit from the following additional functionalities, compared to the standard Simulink® input and output ports:

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

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. In accordance with the first property described, the TC Module Input in the subsystem is also included in the process image in TwinCAT.
The inputs and outputs of the Simulink® model should now additionally be mapped automatically to digital inputs and outputs via the properties of the TC Module Input and TC Module Output function blocks.

By opening the function block parameters all parameters of the common input and output ports appear. In addition, a tree item identifier can be specified, which is to be linked in TwinCAT.

By selecting the button Link a dialog opens. You can now load an existing TwinCAT project (Select Project) and browse the existing inputs or outputs. By selecting the input or output you want to link, the tree item identifier is automatically set and the appropriate data type is automatically entered in Simulink®.

Likewise, you can create a new project. Then select a memory path of the new TwinCAT project to be created, select the target to which you want to download the project and automatically scan the I/O tree of the target. All inputs and outputs of the target are displayed and can be selected.
If the Simulink® model described above is compiled into a TcCOM and integrated in a TwinCAT 3 solution, a mapping to the inputs and outputs selected in Simulink is automatically created. The automatically generated mappings are marked with a blue symbol to distinguish them from manual mapping, while manual mapping symbols appear white.

### 3.6 Overview of automatically generated files

When a build process is initiated, some files and folders are created automatically. Where the files are located, what can be done with them and what the files mean - this is described below.

What are the categories of automatically generated files?

- Source code is generated.
- Log files are generated.
- The TwinCAT objects, drivers (*.tmx) and description files (*.tmc, *.library, ...) are created.
Generated source code

All source files required for the build, i.e. for creating the TwinCAT objects, are combined by default in the current MATLAB® path in the folder `<SimulinkModelName>_tcgrt`. MathWorks generates the slprj folder in the MATLAB® path.

Central file for the source code is `<SimulinkModelName>.vcxproj`. 
This file can be used to create all TwinCAT objects. For example, from Simulink® only the code generation can be triggered without a build process and the build process can be executed on another system, e.g. a Build Server.

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

The generated log files are also summarized in the folder `<SimulinkModelName>_tcgrt`.

The log files created are the first place to look when debugging. If you request help from our support, please always send the following file with your request:

`<ModelName>_ModuleGenerationLog.txt`

Created TwinCAT objects

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

```
%TwinCATInstallDir% \3.1\Repository\<TE140x Module Vendor>\<ModelName>\<Version>\n```

This folder contains the tmc description file and the tmx drivers for the configured platforms as well as other description files.

If the order at `<ModelName>` level is copied to other PCs with TwinCAT XAE in the local engineering repositories, their users can use the created TwinCAT objects in their TwinCAT solutions.

Compare also Creating TMX archives [*33*].

Additional Notes

Description of the generated C++ files and binary files

Versioned C++ projects
3.7 Parameterization of the code generation in Simulink

Within Simulink®, a large number of settings can be made for the configuration of the TwinCAT objects to be generated (TcCOM and function blocks). To this end the tree structure under Code Generation is extended with the several entries (see entries starting with TC).

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

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

Use the MATLAB® Command Window to set the view:

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

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

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

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

Read the tooltips

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

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

Define versioning in Simulink®

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

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

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

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

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

If a version is to be specified in Simulink®, this can be done via "DrvFileVersion". Simply enter the target version in the input field.
In the Engineering Repository, a folder is created under the model name for each version created. Each version folder then contains the corresponding drivers and TwinCAT files. See also Overview of automatically generated files [27].

Use versioned models in TwinCAT XAE

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

As an example, two versions (0.0.0.4 and 0.0.0.5) of SimpleTempCtrl are available here:
Online Change of a TcCOM during TwinCAT Run

✓ To switch between different versions of a TcCOM during operation, the corresponding interface must be implemented.

1. To do this, select the checkbox for **Online change support** under the tab **TC TcCom General** in Simulink®.

**Online Change for PLC library**

If you use the function block in a versioned PLC library, you do not have to activate the checkbox **Online change support**. The Online Change process then runs via another PLC-specific mechanism.
In addition, created TcCOM data areas must be compatible with each other. If **Online change support** is activated, the last hierarchical level is more strictly differentiated in the Insert TcCom Object dialog. Only Online Change compatible TcCOM are combined.

The following shows that versions 0.0.0.1 and 0.0.0.3 or 0.0.0.4 and 0.0.0.5 are compatible for Online Change. However, not 0.0.0.3 to 0.0.0.4 or 0.0.0.2.
In order to better ensure the compatibility of the Data Areas, it is possible, for example, to keep the parameters, Block I/O, ContState and DWork of a model not in an internal Data Area, but as module parameters. This means that only the inputs and outputs as Data Area are relevant for the compatibility of the TcCOM versions.

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

3. Select a version of your choice from the drop-down menu at Online Version (only compatible versions are displayed).

4. Right-click on the line of the object and select Apply changed online object versions to activate the new version of the TcCOM.

Details can also be found in this TwinCAT C++ documentation.

3.7.2 Creating TMX archives

In order to be able to work with the created TwinCAT objects (TcCOM and PLC library) in TwinCAT XAE, they must be available in the repository folder on the local engineering PC and the PLC library must be installed in the local PLC Library Repository.

For example the SimpleTempCtrl in version 0.0.0.2 is located here:
Manual copying to engineering PCs is error-prone. It is therefore easier to create a so-called TMX archive. The TMX archive is an archive of a newly created project, for example the SimpleTempCtrl in version 0.0.0.2. Only the archive has to be copied to an engineering PC and executed. It is a self-extracting archive, which then automatically copies all files to the correct location.

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

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

You can then copy the TMX archive to any path on an engineering PC and execute it. This will copy the files in the archive to the correct location in your repository.

You can also use the Command prompt, for example, and use other options:
For example, the `<tmxarchive>.exe /plclib:install` command creates (as the *.tml file) and installs the PLC library on your local engineering PC.

### 3.7.3 Bundling of several models in one TwinCAT driver

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

**Advantages of bundling**

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

**Advantages of bundling in one driver:**
• The number of files in the repository directory is significantly reduced. This also means that fewer files have to be copied to other engineering systems in order to make a large number of modules available on engineering systems.

• The management of different versions is simplified, as interacting modules can be exchanged in a bundle, so that no version conflicts arise.

Procedure

1. Disable "Run the publish step after project generation" in Simulink®. This will abort after the code generation and the created C++ project will not be compiled.

2. When bundling multiple modules, simply use the generated folders `<modelname>_tcgrt`, which are placed in the current MATLAB® path.

3. Load, bundle and configure export configurations (`<modelname>_tcgrt`) in a project using ModuleGenerator.

4. Create an export project.

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

```matlab
% find the code artifacts in the existing code generation directories
controllerBuildDir = fullfile(pwd,'TempCtrl_tcgrt');
ctrlsystemBuildDir = fullfile(pwd,'TempCtrlSysPT2_tcbgrt');
% load existing export configurations
controllerCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(controllerBuildDir);
ctrlsystemCfg = TwinCAT.ModuleGenerator.ProjectExportConfig.Load(ctrlsystemBuildDir);
% create a new project export configuration
combinedCfg = TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',fullfile(pwd,'TempCtrlLib','TempCtrlLib.vcxproj'));
% add the loaded class export configurations to the new project configuration
combinedCfg.AddClassExportConfig(controllerCfg.ClassExportCfg{1});
combinedCfg.AddClassExportConfig(ctrlsystemCfg.ClassExportCfg{1});
% ...% additinal class export configurations can be added here, loaded from% - Simulink code generation directories (as described)% - MATLAB code generation directories (from MATLAB Coder with TE1401) in the same way% turn on generation and installation of the PLC library
combinedCfg.Project.GeneratePlcLibrary = true; % generate a PLC Lib true/false
combinedCfg.Project.InstallPlcLibrary = true; % install the PLC lib on local system true/false% instantiate and run the project exporter
TwinCAT.ModuleGenerator.ProjectExporter(combinedCfg);
```

Sample code in MATLAB®

Open the appropriate sample with: `TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules')`

### 3.7.4 Configuration of the TMX file properties

You can parameterize the entries in the TMX file (TwinCAT Module Executable) from Simulink®. To do this, switch to Advanced mode.

`TwinCAT.ModuleGenerator.Settings.Change('ConfigurationLevel', 'Advanced')`
Fig. 1: tmx_file_details

Relationship of TMX properties (left) to parameters in Simulink® (right)

File description -> Description
File Version -> DrvFileVersion
Product name -> ProductName
Product version -> DrvProductVersion
Copyright -> Copyright

Note: $<>$ describes placeholders. For example, DrvProductVersion is set to the value in DrvFileVersion which in turn gets the value from Version Source File.

3.7.5 Working with callbacks

There are three different callback functions:

- Pre code generation callback function: Callback before the model is converted to C++ code.
- Post code generation callback function: Callback after the model has been converted to C++ code.
- Post publish callback function: Callback after the created C++ project has been built for the configured platforms.

Enter the name of your created MATLAB function here to call it.
Your MATLAB function is passed the ProjectExporter object as a transfer parameter:

```matlab
function MyCallback(obj)
  ...
  return
```

The object contains the current configuration of the build in its properties.

```
ProjectExporter with properties:
ProjectGenerator: [1×1 TwinCAT.ModuleGenerator.ProjectGenerator]
Configuration: [1×1 TwinCAT.ModuleGenerator.ProjectExportConfig]
Project: [1×1 TwinCAT.ModuleGenerator.Project]
State: [1×1 struct]
ClassExporters: [{1×1 TwinCAT.ModuleGenerator.Simulink.ModelExporter}]
AdditionalExports: [1×1 containers.Map]
```

### 3.8 Application of modules in TwinCAT

TcCOM and function blocks created with the Target for Simulink® can be used seamlessly in TwinCAT XAE.

The only requirement for use on any TwinCAT XAE system is the use of TwinCAT XAE version 3.1.4024.7 and higher. MATLAB®, a full Visual Studio installation, etc. are not necessary, since you work with objects and description files already compiled for TwinCAT. Simply copy the Engineering Repository folder to the engineering system of your choice. Compare Overview of automatically generated files [P 27]. Always keep the folder structure:

```
%TwinCATInstallDir% \3.1\Repository\<TE140x Module Vendor>\<ModelName>\<Version>\.
```
3.8.1 Working with the TcCOM module

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

3. Select the desired object.

4. Create a cyclic task.
5. Assign the created task to your TcCOM instance.

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

6. Activate the configuration.

3.8.1.1 Parameterization of a module instance

Default, Startup, Online and Prepared parameters

Parameters can have different states/properties. These are defined below:

- **Default values** are the parameter values during code generation. They are invariably stored in the module description file (*.tmc) and enable the manufacturing settings to be restored after parameter changes.

- **Startup values** are stored in the TwinCAT project file and downloaded to the module instance as soon as TwinCAT starts the module instance. Startup values for the input process image can also be specified in Simulink® modules. This allows the
module to be started with non-zero input values, without the need for linking the inputs with other process images. Internal signals and output signals have no starting values, since they would, in any case, be overwritten in the first cycle.

- **Online values** are only available if the TcCOM 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.

### Parameter lists in TwinCAT XAE

![Diagram of a block diagram in TwinCAT XAE](image)

In general, TcCOM modules can be parameterized via the parameter list under the tabs **Parameter (Init)** and **Parameter (Online)** in the TwinCAT 3 development environment (XAE). Simulink® modules can also be parameterized via the block diagram, provided that the export of the block diagram was activated when the module was created.

Parameters that have a startup value can be set under Parameters (Init). The startup values can be changed in Config mode. Values can also be changed online here in TwinCAT Run Mode. For a better overview in Run Mode, select the checkbox **Show Online Values** in the lower area of the Parameters (Init) page.

Only parameters without a startup value are listed under Parameters (Online).

### Module- and model-specific parameters

The parameter list contains module- and model-specific parameters. Parameters of the module are properties that can be assigned to TwinCAT. Model-specific parameters are assigned to the model from Simulink®.

Module-specific parameters are e.g. ModuleCaller or StepSizeAdaptation.
Model-specific parameters are summarized in structures:

- `<ModelName>_U` => Inputs of the model
- `<ModelName>_Y` => Outputs of the model
- `<ModelName>_B` => Block I/O of the model (shown as a blue line in the TC3 BlockDiagram)
- `<ModelName>_P` => Modifiable parameters of the model
- `<ModelName>_X` => Continuous state variables of the model
- `<ModelName>_DW` => Discrete state variables of the model

In the TC3 BlockDiagram, module-specific parameters are only displayed in the parameter area (right side of the window) if the top level of the block diagram ("<root>") is selected.

Model-specific parameters are also completely visible only at the top level. If you are in a function block, only parameters assigned to this function block are displayed. Their values can be adjusted when the associated function 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:

![Parameter Window Example](image)

ModelParameters.Kp

- **Type:** LREAL
- **Default value:** 0
- **Startup value:** 50
- **Prepared value:** 50
- **Online value:** 50

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.

ADS 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 TcCOM Interface when creating the module.

### 3.8.1.2  Parameterization of several module instances

In the above section it was described that an instance of a TcCOM can be parameterized in TwinCAT, even deviating from the parameters in Simulink®. If several instances of a TcCOM are used in a TwinCAT Solution, different options exist with regard to the individual parameterization of the instances. To implement the following three options, you must use the Code interface packaging setting in Simulink®.
All instances should have the same parameters.
1. Set the parameter to “Reusable function”. This is the default value when selecting the target TwinCatGrt.tlc.
2. Create several instances of your TcCOM in TwinCAT.
3. Under Parameters (Init), configure the <ModelName>_P_Sharing parameter to define or inherit. Define specifies the parameterization of all dependent instances configured with inherit.

⇒ Only one instance with define may be configured.

⇒ It should be possible to parameterize each instance individually.
1. Set the parameter to “C++ class”.
2. Create several instances of your TcCOM in TwinCAT.

⇒ No <ModelName>_P_Sharing parameter is generated. Each instance can be parameterized individually.

⇒ Only one instance should be allowed in the project.
1. Set the parameter to “Nonreusable function”.
   ⇒ If you create several instances of your TcCOM in TwinCAT, you receive an error message when activating the solution.
   ⇒ Whether an instance of a TcCOM can be instantiated multiple times can be seen in the TC3 BlockDiagram.
2. To do this, go to the parameter area on the right. Under Block Identification a parameter "SingleInstance" is visible.

⇒ The value False means multi-instantiable. Accordingly, True means one instantiation.
3.8.1.3 Working with the block diagram in TwinCAT

3.8.1.3.1 Simulink®-TcCOM

If a TwinCAT object was created with the TwinCAT Target for Simulink® and the block diagram export was executed in the process, the block diagram of the Simulink® model can be displayed as a control in the TwinCAT XAE.

Using the block diagram

The block diagram export can be configured during generation of a TcCOM module from MATLAB® or Simulink®. If the export was enabled, the block diagram can be found in the TwinCAT development environment under the "Block Diagram" tab of the module instance.

Using shortcuts, drag & drop and a context menu you can navigate through the hierarchy of the TcCOM module, view parameter values, display signals values and obtain optional additional debug information.
Shortcut functions:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>Zoom to current size of the block diagram tab</td>
</tr>
<tr>
<td>Backspace</td>
<td>Switch to the next higher hierarchical level</td>
</tr>
<tr>
<td>ESC</td>
<td>Switch to the next higher hierarchical level</td>
</tr>
<tr>
<td>CTRL + &quot;+&quot;</td>
<td>Zoom in</td>
</tr>
<tr>
<td>CTRL + &quot;-&quot;</td>
<td>Zoom out</td>
</tr>
<tr>
<td>F5</td>
<td>Attach Debugger</td>
</tr>
<tr>
<td></td>
<td>(System- &gt; Real-Time -&gt; C++ Debugger -&gt; Enable C++ Debugger must be activated)</td>
</tr>
</tbody>
</table>

Context menu functions:

Display signal curves

For verification and troubleshooting it is often helpful to display signal curves. The block diagram offers the following options:

Display signal curves in the block diagram

The block diagram offers an option to display signal curves in a window. To this end, drag and drop a signal or block into a free area of the block diagram.
Create a scope in the block diagram

After the drop, a scope window opens in the block diagram.

Display the scope in the block diagram

The title bar of the scope window offers the following options:

- Close window
- Keep window in the foreground across all block diagram hierarchies
- Minimize window to the title bar
When creating a scope window in the block diagram for a Simulink® bus, all signals of the bus are directly displayed in the scope window.

The scope window in the block diagram can be used for a quick overview. For more detailed analyzes, it is advisable to analyze the signals in a TwinCAT Measurement project.

**Display signal curves in TwinCAT 3 Scope**

If the drop is not made to the block diagram control but to an Axis Group in a TwinCAT Measurement project, the signal is added there.

Add a signal in a TwinCAT 3 Scope

**Module parameterization in the block diagram**

To parameterize a TcCOM instance, the parameter window can be used directly in the block diagram. In addition, the Property table can be used, which can be expanded or collapsed on the right-hand edge of the block diagram. A basic distinction is made between different parameter values:

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

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

- **Default values** are the parameter values during code generation. They are invariably stored in the module description file and enable the manufacturing settings to be restored after parameter changes.

- **Startup values** are stored in the TwinCAT project file and downloaded to the module instance as soon as TwinCAT starts the module instance.
  
  Startup values for the input process image can also be specified in Simulink® modules. This allows the module to be started with non-zero input values, without the need for linking the inputs with other process images. Internal signals and output signals have no starting values, since they would, in any case, be overwritten in the first cycle.

- **Online values** are only available if the module was started on the target system. They show the current parameter value in the running module. This value can also be changed during runtime. Although in this case the corresponding input field has to be enabled via the context menu, in order to prevent accidental inputs.

- **Prepared values** can be specified whenever online values are available. They can be used to save various values, in order to write them consistently to the module. If prepared values have been specified, they are displayed in a table below the block diagram. The buttons to the right of the list can be used to download prepared values as online values and/or save them as starting value, or delete them.

**Parameterization in the block diagram**

Parameterizable blocks are marked with a yellow box in the block diagram.
Parameterizable block

Double-clicking on the block or a single click on the yellow box brings up a window with the parameters that can be changed.

**Control**

If a value is changed, it can be applied with the following keyboard commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + Enter</td>
<td>Set online value directly</td>
</tr>
<tr>
<td>SHIFT + Enter</td>
<td>Set startup value</td>
</tr>
<tr>
<td>Enter</td>
<td>Set prepared value</td>
</tr>
</tbody>
</table>

The icons in the title bar have the following functions:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Close window</td>
</tr>
<tr>
<td>🌋</td>
<td>Keep window in the foreground across all block diagram hierarchical levels</td>
</tr>
<tr>
<td>⚪️</td>
<td>Keep window open at the current block diagram hierarchical level</td>
</tr>
<tr>
<td>🟢</td>
<td>Minimize window to title bar</td>
</tr>
</tbody>
</table>

**Debug**

Different ways are available to find errors within a TcCOM module created with MATLAB®/Simulink®, or to analyze the behavior of the module within the overall architecture of the TwinCAT project.
Debugging in the block diagram

If the block diagram was exported during generation of the TcCOM module, it can be displayed in the TwinCAT development environment and used for debugging within the corresponding module instance, for example. To do so, the block diagram uses the Microsoft Visual Studio debugger, which can be linked with the TwinCAT runtime via the TwinCAT debugger port. Attach the debugger as described in the C++ section under Debugging.

Prerequisites for debugging within the block diagram are:

- The C/C++ source code of the TcCOM module must be present on the engineering systems, and the Visual Studio debugger must be able to find it. Ideally, debugging should take place on the system on which the code was generated. If the module was created on another system, the associated C/C++ source code can usually be made known by integrating the Visual Studio project into the TwinCAT C++ section. The file `<ModelName>.vcxproj` is located in the build directory, see Which files are created automatically during code generation and publishing? [1.1.17]
- 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) section under publish configuration. When publishing the module from the C++ section in TwinCAT, the debugger in the C++ node of the solution must be enabled; see C/C++ documentation, Debugging.
- During code generation, the options Export block diagram and Export block diagram debug information must be enabled in the coder settings under Tc Advanced.
- In the TwinCAT project, the debugger port must be enabled, as described in TwinCAT 3 C++ Enable C++ debugger.

Setting breakpoints in the block diagram

1. After attaching the debugger to the TwinCAT runtime, the possible breakpoints are assigned to the blocks in the block diagram and represented as points. Clicking on the desired breakpoint activates it, so that execution of the module instance is stopped next time the associated function block is executed. The color of the point provides information about the current state of the breakpoint:
   - Gray: breakpoint inactive
   - Red: breakpoint active. The program is stopped next time this function block is executed
   - Yellow dot in the middle: breakpoint hit. Program execution is currently stopped at this point
   - Blue dot in the middle: breakpoint hit (as yellow), but in a different instance of the module.
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 function block. In many cases, the functions of several blocks are consolidated in a code section or even a line in the underlying C++ code. This means that several blocks can share the same breakpoint. Therefore, activation of a breakpoint in the block diagram may also result in changes in the point display in other blocks.

**Evaluating exceptions**

If exceptions occur during processing of a TcCOM module, such as division by zero, the point at which the exception occurred can be shown in the block diagram. To this end, the TcCOM module must meet the above requirements, and the C++ debugger must be enabled in the TwinCAT project (TwinCAT 3 C++ Enable C++ debugger). After the debugger has been attached, which may be done before the exception has occurred or indeed after, the block that caused the exception is highlighted in the block diagram, provided the line of code responsible for the exception can be allocated to a block. The name of the function block is shown in red, and the function block itself is marked in bold.
Manual evaluation of exceptions without source code

Even if the module source code is not available on the engineering system or the C++ debugger was not activated, you can highlight the error location in the block diagram once an exception has occurred.

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

- A prerequisite for highlighting the error location within the block diagram is that debug information was generated (option Export block diagram debug information in the coder settings under Tc Advanced).
3. From the context menu of the block diagram select the entry **Provide exception data**:

4. In the dialog that opens, enter the source code file and line number provided in the error message:
5. The name of the function block associated with the line number is displayed in red, and the function block itself is marked in bold:

3.8.1.4 Connecting to the External mode

You can connect via External mode from your Simulink® environment to a running TcCOM object in the TwinCAT XAR.

- **External mode only for Nonreusable function**
  
  External mode can only be used for a TcCOM instance that was created with the setting "Code interface packaging: Nonreusable function".

- **Simulation time in Simulink® set to "inf"**
  
  Set the Simulink® simulation time to "inf". For operation in TwinCAT it makes no sense to stop the execution of the module after a defined time.

When creating the TcCOM from Simulink®, the "External mode support" must also be activated.
From version 2.x.xxxx.x

To connect, navigate in Simulink® to “External Mode Connect”, then select your TwinCAT target system and the corresponding TcCOM instance you want to connect to in the pop-up window.

3.8.1.5 Online Change

For a description of the Online Change for TcCOM, see: Creation of versioned drivers [29].

- Open sample in MATLAB® with: TwinCAT.ModuleGenerator.Samples.Start('OnlineChange_TemperatureController')
3.8.2 Working with the PLC library

Brief overview

- Create PLC project:

- Load PLC library:

- View content of the PLC library:
3.8.2.1 Create PLC library

In order to use the created function block in the PLC, the corresponding PLC library must be installed on your TwinCAT engineering system.

- You use the Target for Simulink® on the same PC on which you want to program your PLC...
  1. Create a PLC library and also install it directly on your local engineering system from Simulink®.
  2. Select the appropriate checkboxes at TC PLC Library:
From version 2.x.xxxx.x

After a successful build, the new version of the PLC library is directly available in the local TwinCAT XAE.

You want to use the PLC library on any TwinCAT XAE systems...

1. First, copy the created folder from the engineering repository of your build system to the engineering system on which you want to use the PLC library. Note that the folder structure is maintained.

2. Open TwinCAT after copying the files and select PLC > Library Repository.

3. In the newly opened window, select "Install" and navigate to the `<modelname>.library` to install it.

If no `<modelname>.library` file exists, you can also select and install `<modelname>.tml`. From this file the TwinCAT XAE can create the `<modelname>.library`.

### 3.8.2.2 Online Change of the PLC library

While TwinCAT is in run mode, you can exchange the PLC library version in TwinCAT XAE and load it into the running application via Online Change. This means that all function blocks in a PLC library can be updated without a TwinCAT restart.

**Step-by-step procedure:**

1. Create a first PLC library version with the TwinCAT Target for Simulink®.
2. Include this PLC library version in a PLC project.
3. Activate your TwinCAT configuration with the first PLC library version (e.g. version 0.0.0.1).
4. Adapt your Simulink® model and create a PLC library version (0.0.0.2) from it.
5. Select the newly created PLC library version in the PLC at References (you may have to install the new library on the XAE system).
6. Select Build > Build Solution to rebuild the project.
7. Select Login > Login with online change (more information in the PLC documentation).

3.8.3 Debugging

In addition to debugging via the External Mode [53] and via the Block Diagram in TwinCAT XAE [48], you can also use the C++ project created for debugging in the classic way.

Step-by-step procedure:

1. Make sure that your TwinCAT application has been activated with the C++ debugger enabled.

2. Open the C++ project created during code generation that belongs to the module you want to debug. The project can be found in the folder <SimulinkModelName>_tcgrt, which is created in the current MATLAB® path when you start the code generation process.

3. In this folder, search for the file <SimulinkModelName>.vcxproj. You can open the <SimulinkModelName>.vcxproj in Visual Studio alone or also add the vcxproj file in your TwinCAT Solution under C++ with "Add existing Item".
4. Select **Debug > Attach to Process** in the menu bar and select "TwinCAT XAE" as Connection Type and your desired target system under Connection target. Then select **Attach**.
5. Set breakpoints in your C++ code and step through your code as usual. Tip: when executing the code, the step function is used, which you can find in the folder Simulink > Sources > <SimulinkModelName>.cpp.

3.9 FAQ

3.9.1 Build of a sample fails

All samples supplied (list by TwinCAT.ModuleGenerator.Samples.List in the MATLAB® Command Window) have been checked by tests at Beckhoff Automation. If a build of a sample still does not run successfully, it is likely that something needs to be adjusted during setup on your engineering PC.
To test the platform toolset without the influence of MATLAB® please create a TwinCAT Versioned C++ project in TwinCAT (open TwinCAT in Visual Studio).

1. Right-click Add New Item on C++ Tree Item.
2. Then select TwinCAT Module Class with Cyclic Caller.
   - A C++ project appears in the TwinCAT Tree under C++.
3. Build the C++ project and view the Output Window in TwinCAT.

The output window should return "1 succeeded" for the build process. If this is not the case, check whether you have installed the Desktop development with C++ option in Visual Studio.

3.9.2 Problems with the block diagram representation in TwinCAT XAE

TcCOM modules created with the TwinCAT Target for Simulink® version 2.x.xxxx.x and higher require a TC3 BlockDiagram version 1.4.1419.0 and higher for correct display in TwinCAT XAE.

Where can I find the version of the TC3 BlockDiagram?

- Under Programs and Features in the Control Panel > Beckhoff TwinCAT 3 BlockDiagram.
- In the block diagram in TwinCAT XAE > right-click in the window > About TC3 BlockDiagram.

If your TwinCAT XAE installation contains an earlier version of the TC3 BlockDiagram:

- Can you install the TwinCAT Tools for MATLAB and Simulink setup. This includes a new TC3 BlockDiagram version.
- You can contact Beckhoff support to request a separate TC3 BlockDiagram setup.

3.9.3 Can I use TE1400 version 1.2.x and version 2.x at the same time?

Yes, this is possible. Simply install both products on your system and select the appropriate target to differentiate between the two versions.

TwinCatGrt.tlc for version 2.x and TwinCAT.tlc for 1.2.x
3.9.4 What is the difference between "Build" and "Generate code"?

In the Simulink® Coder App, you can choose between "Build" and "Generate Code":

- **Build** refers to the execution of the build process, which creates the corresponding binaries for specific TwinCAT platforms. From Simulink®, the corresponding binaries are then created one after the other for the platforms activated under TC Build, so that it can be decided afterwards for which target platform the compiled functions are to be used.

- **Generate Code** refers to the generation of the C++ code only. It is possible to execute this step after the build process if desired. To do so, uncheck the checkbox **Run the publish step after project generation** under TC Build.

3.9.5 I can't change the parameters of a module in TwinCAT

"Inlined" is set in the TwinCatGrt.tlc as the default value for the parameter **Default parameter behavior**. Change this to "Tunable" or configure which parameters should be marked as "Tunable" via the button **configure**.

3.9.6 Mapping is lost with Reload TMI/TMC

**Challenge:**
You have already TcCOM objects in your TwinCAT solution, which you have created with the Target for Simulink® version 1.2.xxxx.x. You now want to create a new TcCOM object with the Target for Simulink® version 2.x.x.x and replace the newly created TcCOM with Reload TMI/TMC File... in your existing TwinCAT solution.

**Solution:**
Under TC TcCOM Interface, set the "mapping between variable names and ADS symbol names" to "Classic" and use this to create the new TcCOM object.
This means that the mapping is retained if you now replace your old TcCOM object in TwinCAT with Reload TMI/TMC File....

### 3.10 Samples

Samples provided by Beckhoff Automation are installed on your system with the *TwinCAT Tools for MATLAB® and Simulink®* setup.

You can use the following command to display all available samples:

```plaintext
TwinCAT.ModuleGenerator.Samples.List
```
You can access the samples by clicking on the blue start link. To do this, the sample code is copied to your user directory so that you do not change the original sample. You can work with the copy of the sample accordingly and try it out.

Also available for displaying and starting individual samples:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT.ModuleGenerator.Samples.Show(SampleName)</td>
<td>Displays sample information.</td>
</tr>
<tr>
<td>TwinCAT.ModuleGenerator.Samples.Start(SampleName)</td>
<td>Starts the sample.</td>
</tr>
</tbody>
</table>

The argument SampleName is to be passed as a string, e.g.:

TwinCAT.ModuleGenerator.Samples.Start('SimpleTemperatureController')

Spaces in the SampleName are to be replaced with underline in the argument, e.g. "Combine Modules" -> TwinCAT.ModuleGenerator.Samples.Start('Combine_Modules').

### 3.10.1 TwinCAT Automation Interface: use in MATLAB®

**Short description of the Automation Interface**

TwinCAT XAE configurations can be automatically generated and edited via programming/script codes using the TwinCAT Automation Interface. The automation of a TwinCAT configuration is available thanks to so-called Automation Interfaces, which can be accessed via all COM-capable programming languages (e.g. C++ or .NET) and also via dynamic script languages such as Windows PowerShell, IronPython or even the (obsolete) Vbscript. Use from the MATLAB® environment is also possible.

Detailed documentation of the product can be found here: [TwinCAT Automation Interface](#).

**Use in MATLAB®**

The Automation Interface can be made visible in MATLAB® through the command NET.addAssembly. This will enable you to use the interfaces ([Automation Interface API](#)) described in the product documentation. You can also find many programming samples for use from C# and PowerShell ([Automation Interface Configuration](#)).

In order to simplify the entry from MATLAB® for you, you can find below a sample implementation for MATLAB® on the basis of a MATLAB® class, which you can use, modify and expand.
3.10.1.1 Sample: Tc3AutomationInterface

Overview

The sample code consists of two m-files:

- **Tc3AutomationInterface.m**: MATLAB® class that implements several frequently used methods.
- **Tc3AutomationInterfaceGuide.mlx**: MATLAB live script that calls the MATLAB® class as an example.

Call sample with MATLAB®

The TwinCAT Tool for MATLAB® and Simulink® Setup installs the sample on your system. Call the sample with the MATLAB® Command Window: `TwinCAT.ModuleGenerator.Samples.Start('AutomationInterface')`.

The MATLAB® script

The MATLAB® script provides a sample of how you can generate a TwinCAT solution, scan the EtherCAT master for I/Os, instantiate two TcCOM modules, link them and activate the project on a target.

In order to be able to run the script, the two TcCOMs used must be present in your publish directory `%TwinCATDir%\CustomConfig\Modules`. For this, download the Temperature Controller sample from the TE1400 | Target for MATLAB®/Simulink®. Then copy the file folder from the directory `\TE1400Sample_TemperatureController\PrecompiledTcComModules\Actual TwinCAT versions` into the publish directory.

Run the m-file `Tc3AutomationInterface_Testbench.m`. The latest Visual Studio instance available on your system is opened in the background and the TwinCAT solution is configured, saved and activated.

The MATLAB® class

The properties

All variables and interfaces belonging to the instance of the class are contained in the properties of the `Tc3AutomationInterface` class. Hence, several TwinCAT solutions can be built up in a MATLAB® script by generating an instance of the class for each solution. There are then no overlaps.

The constructor

```matlab
function this = Tc3AutomationInterface
```

The constructor loads all necessary assemblies and, if successful, sets the AssembliesLoaded property to TRUE. The loaded assemblies are:

- EnvDTE and EnvDTE80: libraries for the Visual Studio Core Automation. Necessary for the configuration of Visual Studio.
- TwinCAT.Ads: ADS library, e.g. for reading and changing the XAR state.
- System.Xml: library for parsing XML files.

Selected methods of the class

```matlab
function TcComObject = CreateTcCOM(this, Modelname)
```

Use the MATLAB® help functions in order to view the function and the parameters of the method.
CreateTcCOM creates a new instance of a TcCOM

TpComObject = CreateTcCOM(Modelname)
Instanciates the TcCOM with the specified name (Modelname).
Also a task with a matching cycle time is created and linked to the TcCOM-Object.

set properties: TcCOM

see also:
Beckhoff Infosys

A link to the Beckhoff Infosys is also offered with some methods. These refer to documentation examples from the TwinCAT Automation Interface documentation, so that you can directly view a comparison of the implementation in MATLAB®, C# and PowerShell. You can also find a link to the Beckhoff Infosys in the comment in some sections, allowing you to view the source of the information.

The CreateTcCOM method initially begins with the parsing of the <modelname>.tmc file, from which the ClassID, the task cycle time and the task priority are extracted with System.Xml. A corresponding TcCOM is then instantiated and one (or more) associated tasks generated with the Automation Interface. Finally, the task is/tasks are assigned to the TcCOM.

function ActivateOnDevice(this, AmsNetId)

TwinCAT ADS is used in order to query or change the current status of a TwinCAT runtime, e.g. config or run. In the ActivateOnDevice method the XAR is initially switched to the config mode with the specified AmsNetId and the current TwinCAT configuration is then activated and the system started. Pauses are entered between the individual steps, as this procedure may need a little time.

Static methods
Static methods are also available even without an instance of the class.

function vsVersions = GetInstalledVisualStudios

A function that detects and lists the Visual Studio installations available on the system via the Register Key entries is prepared here. The implementation is limited to VS 2010 to VS 2017.

Documents about this
- AutomationInterfaceMATLAB (Resources/zip/5776206091.zip)

3.10.2 Integrating the block diagram controls

The control that displays the block diagram in the TwinCAT XAE environment can also be integrated as a control in your own visualizations.

The following steps are required:
1. Create a new Windows Forms application.
2. Add TwinCAT.BlockDiagram.dll to the toolbox:
3. To do this, select the “Choose Items...” entry in the context menu.

4. Navigate to TwinCAT.Blockdiagram.dll, which can be found under $<\text{TwinCAT installation path}>\3.1\Components\TcBlockDiagram$. 

![Image showing the context menu options and the TwinCAT.Blockdiagram.dll file selection process.](Image)
5. Add a TcBlockdiagram control instance to the Windows Forms object using drag and drop.
4 Up to version 1.2.xxxx.x

TE1400 Target for Simulink® versions lower than 1.2.xxxx.x support MATLAB R2010b to MATLAB R2019a.
TE1400 Target for Simulink® versions higher than 2.x.xxxx.x support MATLAB R2019a and higher.

4.1 Installation

System requirements

Initially, the same requirements apply to the target for MATLAB®/Simulink® as for TwinCAT 3 C/C++. For a detailed description of the TwinCAT 3 C/C++ requirements, please refer to Chapter 4, "Requirements", of the TC3 C++ manual.

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

On the engineering PC

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

On the runtime PC

- IPC or Embedded CX PC with Microsoft operating system based on "Windows NT kernel" (Win XP, Win 7 and corresponding embedded versions, Win 10)
- TwinCAT 3 XAR
  - TwinCAT 3.0 only supports 32-bit operating systems on the target
  - TwinCAT 3.1 supports 32-bit and 64-bit operating systems. If the target is a x64 system, the created drivers must be signed. The TE1400 supports OS driver signing. See "x64: driver signing" in the 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® R2010b up to and including R2019a. As of and including R2019a, the use of TE1400 version 2.x.xxxx.x is recommended.
- Simulink Coder™ (in MATLAB® versions prior to R2011a: Real-Time Workshop®)
- MATLAB Coder™ (in MATLAB® versions prior to R2011a: part of the Real-Time Workshop®)
- Installing the TE1400 Target for MATLAB®/Simulink®

Setup instructions

- Install one of the supported Visual Studio versions, if not already installed. Pay attention to the installation of the C++ components.
  1. Start TwinCAT 3 Setup, if it does not already exist.

If a Visual Studio and a TwinCAT installation already exists but the Visual Studio version does not meet the requirements mentioned above (e.g. Visual Studio Shell or Visual Studio without Visual C++), you first have to install a suitable Visual Studio version (install Visual C++, if necessary). Then run TwinCAT 3 Setup to integrate TwinCAT 3 into the new (or modified) Visual Studio version.

2. If necessary, install the Microsoft Windows Driver Kit (see Installation "Microsoft Windows Driver Kit (WDK)" in the TwinCAT 3 C/C++ manual).
   The order in which the Windows Driver Kit was installed is irrelevant.

3. If you do not have a MATLAB® installation on your system, install it. The order in which MATLAB® was installed is irrelevant.

4. Start the setup TE1400-TargetForMatlabSimulink to install the TE1400.
   The TE1400 is installed in the TwinCAT folder, i.e. it is separate from the MATLAB® installation. A MATLAB® version that exists on the system can be linked to the TE1400 according to point 6.

5. Start MATLAB® as administrator and execute %TwinCAT3Dir%\Functions\TE1400-TargetForMatlabSimulink\SetupTwinCatTarget.p in MATLAB®.
   A setup window opens. See the following section.

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

**User Account Control**

If MATLAB® is executed in a system with activated User Account Control (UAC) without administrator rights, the MATLAB® path cannot be stored permanently. In this case, SetupTwinCatTarget.p must be executed every time MATLAB® is started, since otherwise some files required for generating TwinCAT modules cannot be found.
Driver signing for targets with x64 operating system

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

4.2 Licenses

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

**Required licenses for TE1400**

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

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

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

**Restrictions in the demo version**

The module generator has the following restrictions without a license.

Allowed are models with maximum:

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

Modules created with a demo license may only be used for non-commercial purposes!
TC1320/TC1220: TC3 [PLC /] C++ / MatSim (runtime license)

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

4.3 Quickstart

Configuration of the Simulink® model

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

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

Generating a TcCOM module from Simulink

Generation of the C++ code or the TcCOM module can be started with the Build button (or Generate code) in the lower section of the window for the code generator options. If the option Publish module is activated under TC Build (default setting), the build process for generating executable files starts immediately after the C++ code has been generated, and a TcCOM module is created. Otherwise, the module generator stops after the C++ code and the project file for Visual Studio™ has been generated. For further information please refer to Publish Module [79].
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**

<table>
<thead>
<tr>
<th>Object</th>
<th>Context</th>
<th>Data Area</th>
<th>Interfaces</th>
<th>Block Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context:</td>
<td>Depend On:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Need Call From Sync Mapping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Areas:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Pointer:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Task</th>
<th>Name</th>
<th>Priority</th>
<th>Cycle Time (µs)</th>
<th>ADS Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02000105</td>
<td>Task 1</td>
<td>5</td>
<td>5000</td>
<td>350</td>
</tr>
</tbody>
</table>
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".

**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.

### 4.4 TwinCAT Library in Simulink®

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

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

If you use the input and output function blocks provided by Beckhoff, you will benefit from the following additional functionalities, compared to the standard Simulink® input and output ports:
• You can also define signals and buses from subsystems directly as inputs or outputs for TcCOM, without first transferring the signals/buses from the subsystem to the top system.
• You can (not mandatory) store an automatic mapping to other TcCOM or I/Os in the function block parameters, so that the mapping is executed directly and automatically when the TcCOM is instantiated.

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.
4.5 Parameterization of the code generation in Simulink

Within MATLAB® Simulink® a wide range of configuration setting options for the TcCOM module to be generated are available. To this end the tree structure under Code Generation is extended with the entries Tc Build, Tc Interfaces, Tc External Mode and Tc Advanced. Many parameters can be modified in TwinCAT 3 at the module instance level; see Application of modules in TwinCAT [14], [96].
The corresponding setting options are described below.

**Toolips**
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).

### 4.5.1 Module generation (Tc Build)

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

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

**Publish directory**

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

**Application**

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

The Publish mechanism can be configured 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 un compilable, 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>TC3 Build</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large DataAreas</td>
<td>&lt; 4018</td>
<td>DataAreas &gt; 16 MB are not supported</td>
</tr>
<tr>
<td></td>
<td>&gt;= 4018</td>
<td>DataAreas &gt; 16 MB utilize the data areas of several DataArea IDs, using the &quot;OBJDATAAREA_SPAN_xxx&quot; macros.</td>
</tr>
<tr>
<td>Project subdirectory &quot;_ModuleInstall&quot;</td>
<td>&lt; 4018</td>
<td>During instantiation of a module that was a previously exported via &quot;Publish&quot;, only the TMC description is imported into the TwinCAT project. The module instance still refers to files within the Module generation (Tc Build) [79]. To load the TwinCAT project on other development computer, the Publish directories of the modules in use have to be copied manually into the corresponding directories of the other computers. Otherwise the project cannot be activated, and the block diagram is not displayed.</td>
</tr>
<tr>
<td></td>
<td>&gt;= 4018</td>
<td>During instantiation of an exported module, all associated files are copied to subdirectory &quot;_ModuleInstall&quot; 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 Module generation (Tc Build) [79] 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 &quot;Reload TMC&quot; and can still be re-activated on a target system.</td>
</tr>
</tbody>
</table>

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:

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

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

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

See also: Examples [120]

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;ENVIRONMENT VARIABLE&gt;</td>
<td>This placeholder is resolved at an early stage during code generation. The value is written into the generated C++ project. If the specified environment variable is not found, the code generation process terminates with a corresponding error message.</td>
</tr>
<tr>
<td>$(ENVIRONMENT VARIABLE)</td>
<td>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.</td>
</tr>
<tr>
<td>CertificateName</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 4.5.2 Data exchange (Tc Interfaces)

#### Configuration in Simulink

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

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Model inputs</td>
</tr>
<tr>
<td>Output</td>
<td>Model outputs</td>
</tr>
<tr>
<td>Parameter</td>
<td>Model-specific parameters: Parameters of Simulink blocks that can be &quot;set&quot;</td>
</tr>
<tr>
<td>BlockIO</td>
<td>Global output signals of Simulink blocks: Internal signals for which a &quot;test point&quot; was set or which were declared as global due to code optimizations of the code generator.</td>
</tr>
<tr>
<td>ContState</td>
<td>Continuous state variables</td>
</tr>
<tr>
<td>DWork</td>
<td>Time-discrete state variables</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP access (checkbox)</td>
<td>TRUE The module enables access to variables of this group.</td>
</tr>
<tr>
<td>ADS access</td>
<td>FALSE The module denies access to variables of this group.</td>
</tr>
<tr>
<td></td>
<td>Only relevant if &quot;GROUP access&quot;=TRUE</td>
</tr>
<tr>
<td>No ADS access</td>
<td>No ADS access</td>
</tr>
<tr>
<td>ReadOnly_NoSymbols</td>
<td>No ADS write access, ADS communication is only possible via the Index group and the Index Offset information</td>
</tr>
<tr>
<td>ReadWrite_NoSymbols</td>
<td>Full ADS access, ADS communication is only possible via the Index group and the Index Offset information</td>
</tr>
<tr>
<td>ReadOnly_CreateSymbols</td>
<td>No ADS write access, ADS symbol information is generated</td>
</tr>
</tbody>
</table>
### Parameter Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadWrite_CreateSymbols</td>
<td>Full ADS access,</td>
</tr>
<tr>
<td></td>
<td>ADS symbol information is generated</td>
</tr>
<tr>
<td>Process image</td>
<td>Only relevant if &quot;GROUP access&quot;=TRUE</td>
</tr>
<tr>
<td>No DataArea</td>
<td>Link to DataArea or I/O: no</td>
</tr>
<tr>
<td></td>
<td>Link to DataPointer: no</td>
</tr>
<tr>
<td>Standard DataArea</td>
<td>Link to DataArea or I/O: no</td>
</tr>
<tr>
<td></td>
<td>Link to DataPointer: yes</td>
</tr>
<tr>
<td>Input-Destination DataArea</td>
<td>Link to DataArea or I/O: yes</td>
</tr>
<tr>
<td></td>
<td>Link to DataPointer: yes</td>
</tr>
<tr>
<td>Output-Source DataArea</td>
<td>Link to DataArea or I/O: yes</td>
</tr>
<tr>
<td></td>
<td>Link to DataPointer: yes</td>
</tr>
<tr>
<td>Internal DataArea</td>
<td>Link to DataArea or I/O: no</td>
</tr>
<tr>
<td></td>
<td>Link to DataPointer: no</td>
</tr>
<tr>
<td>Retain DataArea</td>
<td>Enables linking to a &quot;retain handler&quot; (see retain data [84]) for</td>
</tr>
<tr>
<td></td>
<td>remanent data management.</td>
</tr>
</tbody>
</table>

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

![Configuration Parameters](image)

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.

### 4.5.2.1 Retain data

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

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

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

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

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

Using the retain handler with a PLC project

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

```plaintext
PROGRAM MAIN
VAR RETAIN
  l: UINT;
  k: UINT;
END_VAR
VAR
```
Corresponding symbols are created after a "Build". The assignment to the retain handler of the NOV-DP-RAM device is done in column \textbf{Retain Hdl}.

If self-defined data types (DUTs) are used as retain, the data types must be available in the TwinCAT type system. You can either use the option \textit{Convert to Global Type} or you can create structures directly as

\begin{verbatim}
STRUCT RETAIN
\end{verbatim}

However, the Retain Handler then handles all occurrences of the structure.

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

\subsection*{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 \textbf{Retain Hdl}. 

\begin{verbatim}
{attribute 'TcRetain':='1'}
m: UINT;
x: UINT;
END_VAR
\end{verbatim}
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".

4.5.3 External mode (Tc External Mode)

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

Configuration of the module generator

An External Mode connection is only possible if the generated module supports it. To this end External Mode must be activated in the settings for the Simulink Coder under TC External Mode before the module is generated:
In addition, there is a button for preconfiguring the "External Mode" connection. For information on configuring the "External Mode" connection see section "Establishing a connection". Further parameters under this tab are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow real-time execution commands via External Mode</td>
<td>Defines the default value of the module parameter [88] &quot;AllowExecutionCommands&quot;, which specifies whether the module should process start and stop commands from Simulink. <strong>Special behavior of this parameter:</strong> The module parameter &quot;AllowExecutionCommands&quot; 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.</td>
<td>FALSE</td>
</tr>
<tr>
<td>By default wait on External Mode start command</td>
<td>Default value of the module parameter [88] &quot;WaitForStartCommand&quot;</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activated</td>
<td><strong>ReadOnly.</strong> Determines whether the generated module supports the external mode.</td>
<td>Setting the module generator</td>
</tr>
<tr>
<td>AllowExecutionCommands</td>
<td>Only relevant if &quot;Activated&quot;=TRUE. <strong>ReadOnly</strong> 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.</td>
<td>Setting the module generator</td>
</tr>
<tr>
<td>TRUE</td>
<td>Enables Simulink to start and stop the module execution via the &quot;External Mode&quot; connection.</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>Start and stop commands are ignored in the module.</td>
<td></td>
</tr>
</tbody>
</table>
### 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" [p. 116].

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).

4.5.4 Advanced settings (Tc Advanced)

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

<table>
<thead>
<tr>
<th>&quot;Depend On&quot; setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Config</td>
<td>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 &quot;Call parameters&quot;</td>
</tr>
<tr>
<td>Parent Object</td>
<td>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.</td>
</tr>
<tr>
<td>Task Properties</td>
<td>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 &quot;System Configuration -&gt; Task Management&quot;.</td>
</tr>
</tbody>
</table>

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 [101]."

**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
  - Medium jitter in the reaction time

- **StateUpdateBeforeOutputUpdate**
  - Average reaction time
  - Maximum jitter in the reaction time, since it depends on the execution times for "state update" and "output update"

**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" [116].

**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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitoring execution times</td>
<td>The execution times of the TC module are calculated and made available as ADS variable for monitoring.</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE Calculation of execution times disabled.</td>
<td>FALSE</td>
</tr>
<tr>
<td>Export block diagram</td>
<td>The Simulink block diagram is exported and displayed in XAE under its &quot;Block Diagram&quot; tab once the module has been instantiated.</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE The Simulink block diagram is not exported, and the &quot;Block Diagram&quot; tab is not displayed in XAE.</td>
<td>FALSE</td>
</tr>
<tr>
<td>Resolve masked Subsystems</td>
<td>Only relevant if &quot;Export block diagram&quot;=TRUE.</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Masked subsystems are dissolved. All contents of masked subsystems are visible to users of the generated module in XAE.</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>Masked subsystems are not resolved. The contents of the masked subsystems are not visible to users of the generated module.</td>
<td></td>
</tr>
<tr>
<td>Access to VariableGroup, not referenced by any block</td>
<td>Variables of this group, which belong to a block within an unresolved subsystem, are assigned to the next higher, visible subsystem.</td>
<td></td>
</tr>
<tr>
<td>Hide in block diagram</td>
<td>Variables of this group, which cannot be allocated to a Simulink block, are not displayed in the block diagram.</td>
<td></td>
</tr>
<tr>
<td>Hide, No access</td>
<td>Variables of this group, which cannot be allocated to a Simulink block, are hidden and made inaccessible. This is only possible, if &quot;No DataArea&quot; was selected for the process image of this variable group (<a href="83">Data exchange (Tc Interfaces)</a>).</td>
<td></td>
</tr>
<tr>
<td>Export block diagram debug information</td>
<td>Debugging information generated for the block diagram enables allocation of row numbers in the generated code to displayed blocks. Required for Debug [48].</td>
<td>TRUE</td>
</tr>
<tr>
<td>Show parameter table in XAE</td>
<td>The &quot;Parameter (Init)&quot; tab is displayed in XAE and enables parameterization of the module via the parameter list.</td>
<td>TRUE</td>
</tr>
<tr>
<td>Use original input and output block names</td>
<td>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.</td>
<td>FALSE</td>
</tr>
<tr>
<td>Set testpoints at Simulink Scope signals before code generation</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Maximum number of visible array elements</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

### 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?](#118).

**Skip caller verification**

This option disables queries when calling a TcCOM from the PLC, see Calling the generated module from a PLC project [1.101]. 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 [1.101]."

**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}'

---

### 4.6 Application of modules in TwinCAT

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

#### 4.6.1 Parameterization of a module instance

**Parameter representation in XAE**

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

Module- and model-specific parameters

The parameter list contains module- and model-specific parameters. Examples of module-specific parameters are "Call Parameter" or "External Mode Parameter". 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 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:

**ModelParameters.Kp**

<table>
<thead>
<tr>
<th>Type</th>
<th>LREAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Startup value</td>
<td>50</td>
</tr>
<tr>
<td>Prepared value</td>
<td>50</td>
</tr>
<tr>
<td>Online value</td>
<td>50</td>
</tr>
</tbody>
</table>

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)](Data_exchange_(Tc_Interfaces)) [83]).

**“Default”, “Startup”, “Online” and “Prepared”**

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

- **Default values** are the parameter values during code generation. They are invariably stored in the module description file and enable the manufacturing settings to be restored after parameter changes.

- **Startup values** are stored in the TwinCAT project file and downloaded to the module instance as soon as TwinCAT starts the module instance. Startup values for the input process image can also be specified in Simulink® modules. This allows the module to be started with non-zero input values, without the need for linking the inputs with other process images. Internal signals and output signals have no starting values, since they would, in any case, be overwritten in the first cycle.

- **Online values** are only available if the module was started on the target system. They show the current parameter value in the running module. This value can also be changed during runtime. Although in this case the corresponding input field has to be enabled via the context menu, in order to prevent accidental inputs.

- **Prepared values** can be specified whenever online values are available. They can be used to save various values, in order to write them consistently to the module. If prepared values have been specified, they are displayed in a table below the block diagram. The buttons to the right of the list can be used to download prepared values as online values and/or save them as starting value, or delete them.

### 4.6.2 Executing the generated module under TwinCAT

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

**Context settings**

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

<table>
<thead>
<tr>
<th>&quot;Depend On&quot; setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Config</td>
<td>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 &quot;Call parameters&quot;.</td>
</tr>
<tr>
<td>Parent Object</td>
<td>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.</td>
</tr>
<tr>
<td>Task Properties</td>
<td>The tasks are automatically assigned to the module when the cycle time and 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 &quot;System Configuration -&gt; Task Management&quot;.</td>
</tr>
</tbody>
</table>

**Configuration in XAE**

Parameters that affect the behavior of Simulink module execution are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options / description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallBy</td>
<td>Task</td>
</tr>
<tr>
<td>Modules</td>
<td>The module automatically appends itself to the tasks specified in the context settings [98], when TwinCAT is switched to Run mode. The tasks call the module cyclically until TwinCAT is stopped.</td>
</tr>
<tr>
<td>Step size</td>
<td>RequireMatchingTaskCycleTime</td>
</tr>
<tr>
<td>The module expects the &quot;Fixed Step Size&quot; 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.</td>
<td></td>
</tr>
</tbody>
</table>
Up to version 1.2.xxxx.x

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options / description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseTaskCycleTime</td>
<td>The module enables cycle times, which differ from the &quot;Fixed Step Size&quot; specified in Simulink. In multitasking modules, all task cycle times must match the corresponding Simulink sample times.</td>
</tr>
<tr>
<td>UseModelStepSize</td>
<td>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.</td>
</tr>
<tr>
<td>ExecutionSequence</td>
<td>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 &quot;StateUpdateAfterOutputMapping&quot;. The differences between the different options are described under &quot;order of execution [93]&quot;.</td>
</tr>
<tr>
<td>IOAtTaskbeginn</td>
<td>Execution order:</td>
</tr>
<tr>
<td></td>
<td>Input mapping -&gt; Output mapping -&gt; State update -&gt; Output update -&gt; External mode processing -&gt; ADS access</td>
</tr>
<tr>
<td>StateupdateAfterOutputMapping</td>
<td>Execution order:</td>
</tr>
<tr>
<td></td>
<td>Input mapping -&gt; Output update -&gt; Output mapping -&gt; State update -&gt; External mode processing -&gt; ADS access</td>
</tr>
<tr>
<td>StateupdateBeforeOutputUpdate</td>
<td>Execution order:</td>
</tr>
<tr>
<td></td>
<td>Input mapping -&gt; State update -&gt; Output update -&gt; Output mapping -&gt; External mode processing -&gt; ADS access</td>
</tr>
</tbody>
</table>

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

- **Parameter (Init)**:

![Parameter (Init) table](image)

- **Block diagram**: 

![Block diagram](image)
Up to version 1.2.xxxx.x

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

4.6.3 Calling the generated module from a PLC project

If the call parameter "CallBy" was set to the value "Module", the assigned tasks do not call the module automatically. To call the generated TcCom module via another module instead, the interfaces of that module can be accessed. This can be done from a C++ module or 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 [79] 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No PLCopen XML file is generated</td>
</tr>
</tbody>
</table>
| **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: "ParameterName_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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitData</td>
<td>ST_&lt;ModuleName&gt;_InitData</td>
<td>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.</td>
</tr>
<tr>
<td>&lt;ParameterName&gt;_Startup</td>
<td>&lt;ParameterType&gt;</td>
<td>Available for all parameters, if the coder is configured accordingly. Enables transparent access to the corresponding element of the InitData structure (read/write).</td>
</tr>
<tr>
<td>&lt;ParameterName&gt;_Online</td>
<td>HRESULT</td>
<td>Available for all parameters, if the coder is configured accordingly. Reads or writes the online values of the corresponding module parameter.</td>
</tr>
</tbody>
</table>

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:

```st
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:

```st
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;
```
Generating and referencing a dynamic module instance:

If \(<\text{ObjectId}> = 0\), the FB attempts to generate an instance of the TcCom module dynamically:

\[
\text{fbDynamic : FB_<ModuleName>(oid:=0);}
\]

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:

\[
\text{PROGRAM MAIN}
\]

\[
\text{VAR}
\]

\[
\text{// declare function block}
\]

\[
\text{fbTempContr_Dyn : FB_TempContr(oid:= 0 );}
\]

\[
\text{// input process image variable}
\]

\[
\text{nInputTemperature AT%I*: INT;}
\]

\[
\text{// output process image variable}
\]

\[
\text{bHeaterOn AT%Q*: BOOL;}
\]

\[
\text{// reset error code and reinitialize Object}
\]

\[
\text{bReset: BOOL;}
\]

\[
\text{// initialization helpers}
\]

\[
\text{stInitData : ST_TempContr_InitData;}
\]

\[
\text{bInitialized : BOOL;}
\]

\[
\text{END_VAR}
\]

\[
\text{IF (NOT bInitialized) THEN}
\]

\[
\text{stInitData := fbTempContr_Dyn.InitData; // read default parameters}
\]

\[
\text{// adapt required parameters:}
\]

\[
\text{stInitData.ContextInfoArr_0_TaskOid := 16#02010020; // oid of the plc task}
\]

\[
\text{stInitData.ContextInfoArr_0_TaskCycleTimeNs := 10 * 1000000; // plc task cycle time in ns}
\]

\[
\text{stInitData.ContextInfoArr_0_TaskPriority := 20; // plc task priority}
\]

\[
\text{stInitData.CallBy := TctModuleCallByType.Module;}
\]

\[
\text{stInitData.StepSize := TctStepSizeType.UseTaskCycleTime;}
\]

\[
\text{// set init data, copied to module the next time it switches from INIT to PREOP:}
\]

\[
\text{fbTempContr_Dyn.InitData := stInitData;}
\]

\[
\text{bInitialized := TRUE;}
\]

\[
\text{ELSIF (fbTempContr_Dyn.State<TCOM_STATE.TCOM_STATE_OP) THEN}
\]

\[
\text{// try to change to OP mode}
\]

\[
\text{fbTempContr_Dyn.State := TCOM_STATE.TCOM_STATE_OP;}
\]

\[
\text{ELSIF (NOT fbTempContr_Dyn.Error) THEN}
\]

\[
\text{// set input values}
\]

\[
\text{fbTempContr_Dyn.stInput.FeedbackTemp := nInputTemperature;}
\]

\[
\text{// execute module code}
\]

\[
\text{fbTempContr_Dyn.Execute();}
\]

\[
\text{// get output values}
\]

\[
\text{bHeaterOn := fbTempContr_Dyn.stOutput.HeaterOn;}
\]

\[
\text{END_IF}
\]

\[
\text{IF (bReset) THEN}
\]

\[
\text{IF (fbTempContr_Dyn.Error) THEN}
\]

\[
\text{fbTempContr_Dyn.ResetHresult();}
\]

\[
\text{END_IF}
\]

\[
\text{fbTempContr_Dyn.State := TCOM_STATE.TCOM_STATE_INIT;}
\]

\[
\text{END_IF}
\]
Task context setting

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Task</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02000114</td>
<td>PlcTask</td>
</tr>
</tbody>
</table>

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).

Import of several PLCopen XML files: FB_TcMatSimObject

The generic function block FB_TcMatSimObject is identical for all modules generated with TE1400 (from V1.2.1217.0). 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Return data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>HRESULT</td>
<td>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</td>
</tr>
<tr>
<td>GetObjPara</td>
<td>HRESULT</td>
<td>Reads parameter values from the object via PID (Parameter ID = ADS Index Offset)</td>
</tr>
<tr>
<td>SetObjPara</td>
<td>HRESULT</td>
<td>Writes parameter values to the object via PID (Parameter ID = ADS Index Offset)</td>
</tr>
<tr>
<td>ResetHresult</td>
<td></td>
<td>Acknowledges error codes that have occurred during initialization of the FB or when calling &quot;Execute()&quot;.</td>
</tr>
<tr>
<td>SaveOnlineParametersForInit</td>
<td>HRESULT</td>
<td>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</td>
</tr>
<tr>
<td>SetObjState</td>
<td>HRESULT</td>
<td>Tries to bring the TCOM_STATE of the object to the required target state, step-by-step</td>
</tr>
<tr>
<td>AssignClassId</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>SetInitDataInfo</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Up to version 1.2.xxxx.x

<table>
<thead>
<tr>
<th>Method</th>
<th>Return data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>enables subsequent re-initialization of the object. In this case, the structure may also be initialized by calling “SaveOnlineParametersForInit”.</td>
</tr>
<tr>
<td>SetDataAreaInfo</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Public Properties**

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

### 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
```

#### 4.6.4 Using the ToFile block

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

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

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

**Configuration in the Simulink model**

A model with a sine and a cosine source serves as a simple example. Each signal is to be logged with a ToFile block.
To enable code generation for the ToFile blocks, the format must be set to **Array**.
The model is now ready for code generation.

### Configuration in TwinCAT

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

![Sink Block Parameters: Sine](image)

- **File name:** 
  
  `C:\TestResults\FileWriter\Sinus.mat`

- **Variable name:** 
  
  `YSinus`

- **Save format:** 
  
  `Array`

- **Decimation:** 
  
  `1`

- **Sample time (-1 for inherited):** 
  
  `-1`
Initially, both TcCOMs are instantiated. Both objects can be linked to a joint task or separate task. In order to establish a link between the two objects, the ObjectID of the TcExtendedFileWriter instance is communicated to the Simulink TcCOM.

The ObjectID can be found under the Object tab.
The ObjectID is then inserted under the "Parameters (Init)" tab of the Simulink TcCOM for the parameter 
ExtendedFileAccessOID:

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 TcExtendedFileWriter (the TcExtendedFileWriter instance does not have to have the same cycle time as the assigned Simulink TcCOM module). However, be aware that a write command may not yet be complete when the next cycle starts. If this is the case, the write command is suspended in this cycle. It is therefore a best case assessment.

The TwinCAT project can now be activated.

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

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

The closed files can be opened as usual in MATLAB:

The plot shows the expected sine wave:
4.6.5 Signal access via TwinCAT 3 Scope

TwinCAT 3 Scope enables access to all variable groups for which at least read ADS access was enabled, see Tc Interfaces [83] 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.
4.7 FAQ

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

S-Functions can be integrated into Simulink\textsuperscript{\textregistered} 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.[134].

**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.[140].

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

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

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

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

To debug an SSE exception in TwinCAT, please use the C++ debugger, see Debug.[48] 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.
### 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!

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

**Description of the situation:**

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

**Possible cause and solution:**

A folder named `<modelname>_tct` already exists in the Build directory, see Which files are created automatically during code generation and publishing? [p. 117]. 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.

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

**Observation:**

TwinCAT 3 contains an existing instance of a TcCOM object. As already explained, the model parameters, e.g. the parameters of a PID controller, can be modified in TwinCAT via the exported block diagram or via the Parameter (init) tab of the TcCOM object outside of Simulink. If you change your Simulink model in Simulink and create a new TcCOM object, you can, of course, update this via the call reload TMC/TMI by right-clicking on the corresponding TcCOM object in TwinCAT. In this case all links are preserved, as long as the process image remains unchanged.

A distinction is made between two different cases

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

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

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

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

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

If you use the TMC file from the Build directory, TwinCAT cannot find the corresponding driver and you get the error message shown in the heading when you start TwinCAT.
4.7.6 Why do I have a ClassID conflict when I start TwinCAT?

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

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

**Behavior for TwinCAT version < 4018:**
You will get an error message informing you that the class IDs do not match.

**Behavior for TwinCAT version ≥ 4018**
The driver from the _ModulInstall 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)](#).

**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.

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

**Transfer of the results for “minor time steps’**

Depending on the configured processing sequence [98] 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:

<table>
<thead>
<tr>
<th>Task cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input mapping</td>
</tr>
<tr>
<td>Output update</td>
</tr>
<tr>
<td>Output mapping</td>
</tr>
<tr>
<td>State update</td>
</tr>
<tr>
<td>External mode processing</td>
</tr>
<tr>
<td>ADS access</td>
</tr>
</tbody>
</table>

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 [87].

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

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

- **Direct file access:** No direct access to the file system of the IPC can be realized from the TwinCAT runtime. An exception is the Simulink® sink function block "To File". As described under Using the ToFile block [106], 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.

### 4.7.9 Which files are created automatically during code generation and publishing?

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

**Publish directory**: `%TwinCATDir%\CostumConfig\Modules\`

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

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ModelName&gt;.tmc</code></td>
<td>TwinCAT module class file</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;_ModuleInfo.xml</code></td>
<td>Block diagram information and summary of the engineering system versions (Matlab version, TC version, …)</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;_PlcOpenPOUs.xml</code></td>
<td>Optional file. Can be included for the call of TcCOM from the PLC, see Calling the generated module from a PLC project [101].</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;.sys</code></td>
<td>In the subdirectories TwinCAT RT (x64) and TwinCAT RT (x86). Real-time driver of the created module.</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;.pdb</code></td>
<td>In all subdirectories. Debug information file.</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;.dll</code></td>
<td>In the subdirectories TwinCAT UM (x64) and TwinCAT UM (x86). Driver for the user-mode runtime.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subfolder html</td>
<td></td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;_codegen_rpt.html</code></td>
<td>Summary of code generation and publishing process in html format.</td>
</tr>
<tr>
<td>*.cpp and *.h</td>
<td>Source code of automatic code generation</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;.vcxproj</code></td>
<td>Visual Studio project of automatic code generation. Can be included in the TwinCAT C++ node as an existing project and published from there.</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;_PublishLog.txt</code></td>
<td>Text file with Publish log.</td>
</tr>
<tr>
<td><code>&lt;ModelName&gt;_ModuleInfo.xml</code></td>
<td>Block diagram information and summary of the engineering system versions (Matlab version, TC version, …)</td>
</tr>
</tbody>
</table>
### 4.7.10 How do I resolve data type conflicts in the PLC project?

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

Especially after a module instance has been updated via "Reload TMC", several data types of the same name may exist in the type system, of which typically only the type related to the currently instantiated module class should be used. In some cases the user has 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:

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ModelName&gt;_PlcOpenPOUs.xml</code></td>
<td>Optional file. Can be included for the call of TcCOM from the PLC, see Calling the generated module from a PLC project [101].</td>
</tr>
</tbody>
</table>
4.7.11 Why are the parameters of the transfer function block in the TwinCAT display not identical to the display in Simulink?

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

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

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

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

```
# You can use the C++ project TcSmplTempCtrl.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)".
# Publishing 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.
4.8 Examples

Example models for generating TcCom modules:

<table>
<thead>
<tr>
<th>Example</th>
<th>Topics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemperatureController_minimal [120]</td>
<td>• Basic principles</td>
<td>A very simple temperature controller that covers the basics.</td>
</tr>
<tr>
<td>TemperatureController [126]</td>
<td>• Parameter access</td>
<td>A very simple temperature controller with PWM output. Provides a quick overview of how to use the module generator. Also uses Simulink BusObjects (structures) for an output and includes a test point, which affects the accessibility of internal signals via ADS. ExternalMode is also used in the example.</td>
</tr>
<tr>
<td>SFunStaticLib [134]</td>
<td>• SFunction</td>
<td>Generates TwinCAT modules from Simulink models with SFuntions that are provided by third parties without source code.</td>
</tr>
<tr>
<td>SFunWrappedStaticLib [140]</td>
<td>• SFunction</td>
<td>Generates TwinCAT modules from Simulink models with SFuntions, for which the source code is available, but is dependent on static libraries.</td>
</tr>
</tbody>
</table>

Examples for module generation callbacks [81]:

<table>
<thead>
<tr>
<th>Example</th>
<th>Topics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packaging module files into ZIP archives [145]</td>
<td>• PostPublish callback</td>
<td>This simple example illustrates the automatic archiving of generated module files.</td>
</tr>
<tr>
<td></td>
<td>• Archiving generated module files</td>
<td></td>
</tr>
</tbody>
</table>

4.8.1 TemperatureController_minimal

Description

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

Overview of project directory

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/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:

![Model Explorer](image1)

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**

![Simulink Code Generation](image2)

* 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.

![Block Diagram Status](image)

### 4.8.2 Temperature Controller

**Description**

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

- Parameter access [127]
- Using Bus Objects [129]
- Using Test Points [130]
- Using Referenced Models [132]
- Using External Mode [133]
- Generating TwinCAT modules from SubSystems [134]

**Overview of project directory**

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/zip/27021599304187787.zip contains all the files for this example:
**TctSmplTempCtrl.mdl**  
More advanced (but still very simple) temperature controller.

![Diagram of the TctSmplTempCtrl model](image)

**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.

**_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 $K_p$ and $T_n$ remain configurable, which means that only the Simulink block parameters that depend on these parameters are configurable. The parameter structure is reduced to these two elements.

For further information on **parameter inlining** see Simulink documentation.
Using bus objects

Simulink BusObjects enable access to TwinCAT modules generated in Simulink via structured symbols. This 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.
Using External Mode

The temperature controller model `TctSmpTempCtrl.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 [127]").

### Generating TwinCAT modules from subsystems

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

![Generating TwinCAT modules from subsystems](image)

### 4.8.3 SFunStaticLib

**Use cases**

Encapsulating own code within static libraries can be useful to

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

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

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

Model development side

Algorithm development side

- SimModel.mdl
  - SFunctionBlock

- Code generation and compilation
- SimModel.sys (TC RT X86)
- SimModel.sys (TC RT X64)
- ...

- SFunction.mexW32
- SFunction.mexW64

- Link

- Foo.h
- Foo.cpp
- SFunction.cpp
- ...

- Compile

- Sfunction_TC_RT_x86.lib
- Sfunction_TC_RT_x64.lib
- ...

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

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

Project directory overview

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/
zip/27021599304133259.zip contains all the files necessary to reproduce this example:

<table>
<thead>
<tr>
<th>TctSmplSFunctionStaticLib.mdl</th>
<th>contains the model that references the SFunction.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildLibsAndSFunction.m</td>
<td>contains an M-script that creates the static library for all currently available TwinCAT platforms and creates the SFunction.</td>
</tr>
</tbody>
</table>
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
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%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

#### Code Generation Settings

- **Use the same custom code settings as Simulation Target**

- **Include custom C code in generated:**

  - Source file
  - Header file
  - Initialize function
  - Terminate function

#### Include directories

- Include directories:

  - "C:\\TwinCAT3\\Functions\\TE1400-Target\\ForMatlabSimulink\\Libraries\\TwinCAT RT (x64)"
  - "C:\\TwinCAT3\\Functions\\TE1400-Target\\ForMatlabSimulink\\Libraries\\TwinCAT RT (x64)\\Release"
  - "C:\\TwinCAT3\\Functions\\TE1400-Target\\ForMatlabSimulink\\Libraries\\TwinCAT RT (x64)\\Debug"
  - "C:\\TwinCAT3\\Functions\\TE1400-Target\\ForMatlabSimulink\\Libraries\\TwinCAT UM (x64)"
  - "C:\\TwinCAT3\\Functions\\TE1400-Target\\ForMatlabSimulink\\Libraries\\TwinCAT UM (x64)\\Release"
  - "C:\\TwinCAT3\\Functions\\TE1400-Target\\ForMatlabSimulink\\Libraries\\TwinCAT UM (x64)\\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

4.8.4 SFunWrappedStaticLib

Use cases
Encapsulating own code within static libraries can be useful to
- speed up module generation, if the code contains algorithms, which are not changed frequently
- deliver TwinCAT Target compatible SFunction algorithms to customers, without the need to hand out
  the source code but only the compiled libraries

Description
The following example shows the configuration of the module generator when using Sfunctions that depend
on statically linked libraries. For this type of code integration, a suitable library must be available for all
TwinCAT platforms for which the module is to be created.
The following diagram illustrates the typical workflow when using third-party algorithms in a custom Simulink model:

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

https://infosys.beckhoff.com/content/1033/te1400_tc3_target_Matlab/Resources/zip/27021599304134923.zip contains all the files necessary to reproduce this example:

<table>
<thead>
<tr>
<th>TctSmplSFunWrappedStaticLib.mdl</th>
<th>contains the model that references the SFunction.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TctSample_SFUnLibWrapper.cpp</td>
<td>must be present on the target system. Contains the source code of the SFunction.</td>
</tr>
<tr>
<td>StaticLib.cpp</td>
<td>Simple example of source code of a static library.</td>
</tr>
<tr>
<td>BuildLibsAndSFunction.m</td>
<td>contains an M-script that creates the static library for all currently available TwinCAT platforms and creates the SFunction.</td>
</tr>
<tr>
<td>OpenLibProject.m</td>
<td>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.</td>
</tr>
<tr>
<td>Subdirectory LibProject</td>
<td>contains the static library.</td>
</tr>
<tr>
<td>Subdirectory BuildScripts</td>
<td>contains several M-scripts for &quot;BuildLibsAndSFunction.m&quot; and &quot;OpenLibProject.m&quot;.</td>
</tr>
<tr>
<td>_PrecompiledTcComModules</td>
<td>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!</td>
</tr>
<tr>
<td>_PreviousSimulinkVersions</td>
<td>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</td>
</tr>
</tbody>
</table>

Create SFunction and corresponding statically linked libraries

Build requirements

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

- **Windows Driver Kit** installed on your system and the environment variable `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:
Up to version 1.2.xxxx.x

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

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.

**Generating a TwinCAT module**

**Configuring a model**

The general settings for generating a TwinCAT module must be set, e.g. a fixed-step solver must be configured, and the system target file "TwinCAT.tlc" must be selected under the "General" tab in the model coder settings. For further information on the general configuration of the TwinCAT module generator see Quickstart [72].

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

4.8.5 Module generation Callbacks

Examples for module generation callbacks [81]:

<table>
<thead>
<tr>
<th>Example</th>
<th>Topics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packaging module files into ZIP archives [145]</td>
<td>• PostPublish callback&lt;br&gt;• Archiving generated module files</td>
<td>This simple example illustrates the automatic archiving of generated module files.</td>
</tr>
</tbody>
</table>

4.8.5.1 Packaging module files into ZIP archives

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

```plaintext
zip(fullfile('C:\MyGeneratedTcComModules',cgStruct.ModuleName),''',fullfile(getenv('TwinCat3Dir'),'CustomConfig','Modules',cgStruct.ModuleName))
```
More Information:
www.beckhoff.com/TE1400