# 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 Installation .......................................................................................................................................... 9  
   3.1 Setting up driver signing ............................................................................................................. 11  
4 Licenses ............................................................................................................................................... 13  
5 Quick start .......................................................................................................................................... 14  
6 Overview of automatically generated files ....................................................................................... 26  
7 Settings of the TwinCAT module generator ...................................................................................... 28  
   7.1 Creating TMX archives .............................................................................................................. 32  
8 Application of modules in TwinCAT ................................................................................................. 34  
   8.1 Working with the TcCOM module ............................................................................................... 34  
      8.1.1 MATLAB code representation .......................................................................................... 35  
   8.2 Working with the PLC library .................................................................................................... 39  
      8.2.1 Online change of the PLC library ..................................................................................... 43  
   8.3 Debugging .................................................................................................................................... 44  
9 FAQ ..................................................................................................................................................... 48  
   9.1 Build of a sample fails .................................................................................................................. 48  
10 Samples ............................................................................................................................................ 49  
   10.1 TwinCAT Automation Interface: Use in MATLAB® ............................................................ 49  
      10.1.1 Sample: Tc3AutomationInterface .................................................................................... 50
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>! 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>! 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>! 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>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.

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.
Overview

TE1401 TwinCAT Target for MATLAB®

With the TwinCAT 3 Target for MATLAB®, it is possible to make use of the functions developed in the MATLAB® script language in TwinCAT 3. The functions are automatically transcoded in C/C++ code with the aid of the MATLAB Coder™ and transformed into TwinCAT objects with the TwinCAT 3 Target for MATLAB®. These objects can be used seamlessly in the TwinCAT 3 Engineering, e.g. extended with PLC source code to make an overall project, debugged and linked with fieldbus devices. The automatically generated modules can be integrated in the TwinCAT solution as TcCOM objects on the one hand and as PLC function blocks on the other. The inserted modules are downloaded with the complete TwinCAT project into the TwinCAT 3 runtime, where they are executed within the real-time environment like all other objects. TwinCAT 3 Target for Simulink® supports targets with Windows 32-bit and 64-bit as well as TwinCAT/BSD.

Further Information

Technical short video

- TwinCAT Target for MATLAB

Product description

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

Web page for MATLAB® and Simulink® with TwinCAT 3

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

System requirements

In the following, a distinction is made between the engineering PC and the runtime PC. The following definition applies: on the engineering PC, MATLAB® functions are converted to TwinCAT objects by using the Target for MATLAB®. 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
  - MATLAB® and MATLAB Coder™ Toolbox
- 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.
     - The TwinCAT Target for MATLAB® is installed within the TwinCAT folder structure, i.e. it is separate from the MATLAB® installation.
  4. Start MATLAB® as administrator and run
     \%TwinCAT3\%\Functions\TE14xx-ToolsForMatlabAndSimulink\SetupTE14xx.p in MATLAB®.
     - A setup window opens. See the following section.

Setting up the software

Version 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 now or make/change them at a later time.

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

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

Setting options in the dialog are

VendorName, GroupName (MATLAB) and GroupName (Simulink):

These settings influence the hierarchy in which the generated TwinCAT objects are sorted. See diagram below. Here the entries VendorName "TE140x Module Vendor" and GroupName "TE140x|MATLAB Modules" are for MATLAB and "TE140x|Simulink Modules" for Simulink.

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 can be stored as default.
### 3.1 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

*C:\TwinCAT\3.1\Repository\TE140x Module Vendor\ModulName\0.0.0.1\TwinCAT RT (x64)\MyDriver.tmx*
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:

```
bcedit /set testsigning yes
```

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

Two licenses are required to use the full functionality of the TE1401 TwinCAT Target for MATLAB®. On the one hand, the TE1401 engineering license for creating TwinCAT objects from MATLAB® functions and, on the other hand, a runtime license for executing these objects during the TwinCAT runtime.

Engineering license

The license TE1401 Target for MATLAB® is required for the engineering system to create TcCOM and PLC function blocks from MATLAB®. 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 a valid TE1401 license, the following restrictions must be observed:

- All cpp and header files from MATLAB Coder™ must not exceed 50 kB in total.
- Function inputs and function outputs are limited to 5 variables.
- You cannot merge multiple MATLAB® functions into one PLC library.

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

Runtime license

The TC1320 or TC1220 licenses with included PLC license are required to start a TwinCAT configuration with a TwinCAT object generated from MATLAB®. 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.
Quick start

Starting with a simple MATLAB® function

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

1. First select simple samples, e.g. BaseStatistics - can also be called directly with TwinCAT.ModuleGenerator.Samples.Start('BaseStatistics').

Beginner video

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

Getting started with the Base Statistics Sample

✓ Opening the Base Statistics Sample opens a MATLAB Live Script, which contains documentation parts as well as sections with code for execution.

2. Execute the Code Sections by clicking on the respective Run buttons and work your way through the sample step by step.

The sample shows how you can use the Target for MATLAB® to convert two MATLAB® functions into two TcCOM objects and two function blocks, bundling the function blocks in a common PLC library. If you do not have a valid TE1401 license, you can activate the TrialLicense checkbox in the sample. This converts only one MATLAB® function into a TcCOM and a function block. The sample is then compliant with the demo terms [13] of the product.

Selection of components and paths

The General Preparation section of the sample states:

What should be the name of the created TwinCAT driver (tmx-file)?

Here Tc3_BaseStatistics is selected. This name is then used in the following places:
Quick start

- File path in the Engineering Repository:
  \%TwinCATInstallDir\13.1\Repository\<TE140x Module Vendor>\Tc3_BaseStatistics\<Version>\n
- Name of the created files *.tmx, *.tml, *.tmc and *.library
- Name of the created PLC library in TwinCAT, which then contains the two function blocks

Where should all source files be stored?

`buildDir` specifies where the MATLAB Coder™ and also the TwinCAT Target for MATLAB® should store all source files, log files and other meta information files. This folder contains all the information required to create the TwinCAT objects from here. In this case, a new folder `buildDir` is created in the current MATLAB® path.

Which MATLAB® functions should be made available in TwinCAT?

The MATLAB® functions are named here with the variables `module1` and `module2`; the two MATLAB® functions `BaseStatistics` and `BaseStatisticsIteravtive` (stored in subfolder M) are to be transferred to TwinCAT objects accordingly.

Each of these modules gets its own subfolder in `buildDir` which is named `cppDir1` and `cppDir2`. The C++ code is later generated into these subfolders by the MATLAB Coder™.

Creating a MATLAB Coder™ configuration

In the further course of the MATLAB® Live Script, a MATLAB Coder™ configuration is created. This section does not contain any TwinCAT-specific components, i.e. only the MATLAB Coder™ is used. For detailed MATLAB Coder™ documentation, see the MATLAB® documentation.
When creating the Coder configuration `cfg`, please note:

- The Embedded Coder is not supported.
- Only the generated code is needed.

The `codegen` command then receives the Coder configuration and the corresponding MATLAB® function to be translated. The argument "-d", `cppDir1` instructs the MATLAB Coder™ to place the C++ code in the `cppDir1` path.

Accordingly, after this step the generated C++ code of the function `BaseStatistic.m` and `BaseStatisticIterarive.m` are located in the folders `_BuildDir/ BaseStatistic` and `_BuildDir\ BaseStatisticIterarive`.

**Creating a Target for MATLAB® project export configuration**

The following code segments in the MATLAB® Live Script apply only to the Target for MATLAB® and are independent of the MATLAB Coder™ in that only C++ code already created by the MATLAB Coder™ will be used.

Optionally, you can extract the MATLAB® code description from the m-file and display it later in TwinCAT XAE, see MATLAB Code in TcCOM [35].

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

The m-file with the module1 function, i.e. BaseStatistics, must be located in the MATLAB® workspace. The information is then extracted from the `BaseStatistics.m` file and stored in the `cppDir1` folder.

In the next step, a project export configuration is created by the TwinCAT module generator with

TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',FullPathToVcxproj);
The fullpath to the new Visual Studio project to be created is specified as the argument. In this case ...
\_BuildDir\Tc3\_BaseStatistics.vcxproj. After the build, the naming of the Visual Studio project also defines
the naming of the created files *.tmx, *.tmc, *.tml and *.library. See Quick start [14].

Fig. 1:

For each MATLAB® function, an export configuration must be created with
TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig(). AddClassExportConfig adds this
export configuration to the project export configuration as a module.

```
exportConfig.AddClassExportConfig(TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig('MFile',module1,'BuildDir',cppDir1));
```

The path to the C++ code created by the MATLAB Coder™ and the name of the m-file with the
Corresponding MATLAB® function are passed as arguments to FunctionExportConfig(). For example,
module1 is then used to set that the TcCOM object to be created and the function block in the PLC library
are called BaseStatistics or FB_BaseStatistics.

The project export configuration will be further adapted in the following. This defines the platforms for which
a driver is to be built (here for Windows 32-bit, Windows 64-bit and TwinCAT/BSD 64-bit). It is also
configured that a PLC library is to be created and also installed on the local TwinCAT XAE.

For each module added to the project export configuration, properties can be set individually. Here it is
explicitly set that for the first added module both a PLC function block and a TcCOM are to be generated.

You can use `disp(exportConfiguration)` to display an overview of the entire configuration.
This gives you an overview of the values set (Value), the data type used (DataType), suggested values (Options) and a short description (Displayname).

With `TwinCAT.ModuleGenerator.ProjectExporter(exportConfig)` the build process of the configured platforms is triggered. This creates a folder on the local file system in the repository and stores the created drivers and description files.

The path description is:

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

You can copy the folder to any number of TwinCAT XAE systems, so that the modules are available there. Only the *.library must be installed in TwinCAT via the PLC library repository. Please note that the folder structure is not changed during copying.

**Use PLC library in TwinCAT 3**

- Starting from a new TwinCAT solution, create a PLC project:
  1. Perform the following menu steps.

---

<table>
<thead>
<tr>
<th>Project Path</th>
<th>Value</th>
<th>DataType</th>
<th>Options</th>
<th>Displayname</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Projectfullname</code></td>
<td><code>&quot;_BuildDir\BaseStatistics&quot;</code></td>
<td>&quot;string&quot;</td>
<td><strong>&quot;&quot;</strong></td>
<td>&quot;Twincat c++ project path&quot;</td>
</tr>
<tr>
<td><code>ProjectName</code></td>
<td><code>&quot;TMCS module project&quot;</code></td>
<td>&quot;string&quot;</td>
<td><strong>&quot;&quot;</strong></td>
<td>&quot;Twincat source file&quot;</td>
</tr>
<tr>
<td><code>ProjectVersion</code></td>
<td><code>&quot;TwinCAT 3.1&quot;</code></td>
<td>&quot;string&quot;</td>
<td><strong>&quot;&quot;</strong></td>
<td>&quot;Version part for increment&quot;</td>
</tr>
<tr>
<td><code>ProjectHaveControlFlags</code></td>
<td><code>&quot;true&quot;</code></td>
<td>&quot;bool&quot;</td>
<td><strong>&quot;true&quot;</strong></td>
<td>&quot;Applies and creates module&quot;</td>
</tr>
<tr>
<td><code>ProjectPublish</code></td>
<td><code>&quot;true&quot;</code></td>
<td>&quot;bool&quot;</td>
<td><strong>&quot;Release&quot;</strong></td>
<td>&quot;Release&quot;</td>
</tr>
<tr>
<td><code>ProjectPublishControl</code></td>
<td><code>&quot;true&quot;</code></td>
<td>&quot;bool&quot;</td>
<td><strong>&quot;&quot;</strong></td>
<td>&quot;Release: debug&quot;</td>
</tr>
</tbody>
</table>

---

This page contains information on how to use and configure the TwinCAT ModuleGenerator, including details on the build process, path descriptions, and how to copy the modules to TwinCAT XAE systems.

---

**Quick start**

1. Use PLC library in TwinCAT 3:
   - Starting from a new TwinCAT solution, create a PLC project:
     1. Perform the following menu steps.
4. Then add the newly created (and already installed) PLC library:
5. Get an overview of the data types and function blocks:
6. Insert instances of the function blocks into your PLC and use them in your application:

Using TcCOM objects in TwinCAT 3
7. Insert a new TcCOM object.
8. Select the corresponding TcCOM object:

9. Create a new cyclic task of type *TwinCAT Task.*
10. Assign the newly created task to the newly created TcCOM object. To do this, go to the instance of the TcCOM object and select the Context tab.

You can now activate the configuration. In order to connect the TcCOM object with other modules in your TwinCAT solution beforehand, you can use the process image to create mappings.

You can view the MATLAB® code on the Block Diagram tab and observe and scope values on the fly. See MATLAB code representation [35].
6 Overview of automatically generated files

If a build process is triggered via the TwinCAT module generator, 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 stored in the folder specified during initialization of the TwinCAT module generator. The location and name of the Visual Studio project to be generated are specified precisely here.

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

For example, the following graphic shows the FullPath as ...\BuildDir\Tc3_BaseStatistics.

The central file for the source code is `<ProjectName>.vcxproj`. This file can be used to create all TwinCAT objects. From MATLAB® you can, for example, trigger code generation only without a build process and run the build process on another system such as a build server. To do this, set `Project.Publish = false` in the TwinCAT module generator.

Generated log files

The generated log files are also collected in the folder mentioned above.

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

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

If the order 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.

Additional Notes

Description of the generated C++ files and binary files

Versioned C++ projects
7 Settings of the TwinCAT module generator

Adds a project export configuration

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

The full path and name of the Visual Studio project to be created is passed to the 'FullPath' property.

Returns an object of the class TwinCAT.ModuleGenerator.ProjectExportConfig.

Sample call:

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

Methods of the class TwinCAT.ModuleGenerator.ProjectExportConfig

AddClassExportConfig

Adds an export configuration to the project. To create an export configuration, see section Settings of the TwinCAT module generator [32]. The export configuration is appended under Properties in the cell array ClassExportCfg.

Sample call see Quick start [16].

Save

Creates a mat file and stores the project export configuration in it. Transfer argument is a path where the mat file is to be stored.

Sample call:

```matlab
exportConfig.Save(pwd)
```

Saves the project export configuration in the current path.

Load

Loads a saved project export configuration. The transfer argument is the path where the mat file with the saved configuration is located.

Sample call:

```matlab
exportConfig.Load(pwd)
```

Loads the project export configuration from the current path.

Edit

Opens a graphical configuration interface for configuring the project export configuration.

```matlab
disp
```

Gives an overview in the MATLAB® Command Window of the current project export configuration. See Quick start [16].

Sample call:

```matlab
exportConfig.disp alternatively disp(exportConfig)
### Properties of the class TwinCAT.ModuleGenerator.ProjectExportConfig

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Datatype</th>
<th>Options</th>
<th>DisplayItem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FullPath</strong></td>
<td>(&quot;_build{Project}_export{Project}));</td>
<td>&quot;string&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>VendorName</strong></td>
<td>(&quot;_build{Project}_export{Project})));</td>
<td>&quot;string&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IncrementVersion</strong></td>
<td></td>
<td>&quot;string&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DrvFileVersion</strong></td>
<td></td>
<td>&quot;string&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Publish</strong></td>
<td></td>
<td>&quot;bool&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PublishPlatformtoolset</strong></td>
<td></td>
<td>&quot;bool&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PublishTcRtx86</strong></td>
<td></td>
<td>&quot;bool&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PublishTcRtx64</strong></td>
<td></td>
<td>&quot;bool&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Publish</strong></td>
<td></td>
<td>&quot;bool&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Structure with entries for configuring the build properties, the PLC library and the possible callbacks.**

**FullPath to the TwinCAT C++ project to be created.**

**Vendor Name**

Name of the Vendor. The Vendor Name is used to structure the TwinCAT objects. The vendor is created as a folder in the path of the repository and is visible in the structure when the PLC library and the TcCOM object are inserted.

**IncrementVersion**

Possible values: "None", "Revision", "Build", "Minor", "Major"

Default: "Revision"

This setting influences at which of the four digits the version should be incremented. The basis is the last version available on the engineering system (see DrvFileVersion).

**DrvFileVersion**

Default: searches for the last version on the engineering system. If no existing version is found, it is started with 0.0.0.0.

Direct setting of a version: can be set directly as a string, e.g. "1.52.32.0". IncrementVersion will then not be executed.

**Publish**

If TRUE, the created TwinCAT C++ project is built for the configured platforms.

**PublishPlatformtoolset**

Configures the Visual Studio version to use. This can be specified precisely or set to Auto (default).

**PublishTcRtx86**

If TRUE, then XAR is built on a Windows 32-bit platform.

**PublishTcRtx64**

If TRUE, then XAR is built on a Windows 64-bit platform.
Settings of the TwinCAT module generator

Project.PublishTcOSx64

If TRUE, then XAR is built on a TwinCAT/BSD platform.

Project.SignTwinCatCertName

A TwinCAT OEM certificate for driver signing can be specified here (not mandatory). The password is to be entered into the Windows Registry of the user with the TcSignTool. Detail see Setting up driver signing.

Project.TmxArchive

Enter a path and file name here as a string in order to create a TMX archive. Example: Project.TmxArchive = "c:\archives\[Date]-[Time]-[LibName][LibVersion].exe" creates a self-extracting TMX archive under c:\archives. The placeholders are resolved in the module generator before the file is written.

Project.GeneratePlcLibrary

If TRUE, then a PLC library (*.library) is created in the repository folder for the project.

Project.InstallPlcLibrary

If TRUE, the created PLC library is installed on the local TwinCAT XAE.

Project.PreCodeGenerationCallbackFcn

A function can be called here as a string, which is called before the code generation, i.e. the creation of the TwinCAT C++ project. For example, an m-file MyCallback.m can be created in the workspace with the following content:

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

The PreCodeGenerationCallbackFcn property is then set to "MyCallback". By default, the ProjectExporter object is passed to the function, so that you have access to all data of the current project in the callback function.

Project.PostCodeGenerationCallbackFcn

A function can be called here as a string, which is called after code generation, i.e. creation of the TwinCAT C++ project. For example, an m-file MyCallback.m can be created in the workspace with the following content:

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

The PostCodeGenerationCallbackFcn property is then set to "MyCallback". By default, the ProjectExporter object is passed to the function, so that you have access to all data of the current project in the callback function.

Project.PostPublishCallbackFcn

A function can be called here as a string that is called after the compilation, i.e. creation of the TwinCAT objects. For example, an m-file MyCallback.m can be created in the workspace with the following content:

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

The PostPublishCallbackFcn property is then set to "MyCallback". By default, the ProjectExporter object is passed to the function, so that you have access to all data of the current project in the callback function.

Project.OemId and Project.OemLicenses
Settings of the TwinCAT module generator

Optionally, a generated TcCOM or a function block can be linked to an OEM license. This OEM license is checked when starting the object (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.

You can insert your 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.

Sample entry:

```matlab
exportConfig.Project.OemId = '{ABBAABBA-AFFE-AFFE-ABBAABBAABBA}';
exportConfig.Project.OemLicenses = '{11111111-0000-FEFE-CCCC-BBBBBBBBBBB}';
```

ClassExportCfg

Cell array of the added export configurations. Each export configuration, i.e. each converted MATLAB® function or each converted Simulink® model, can be configured individually.

TcCom.Generate

If TRUE, a TcCOM object is created, which can be used in the TwinCAT XAE.

TcCom.FpExceptionsForInit

Floating Point Exceptions within the TcCOM object during the init phase can be set.

Options: "CallerExceptions", "Exceptions", "NoExceptions"

NoException: Floating Point Exceptions are disabled.

Exceptions: Floating Point Exceptions are enabled.

CallerExceptions: The settings of the caller are adopted, e.g. the task, another TcCOM or the PLC.

TcCom.FpExceptionsForUpdate

Floating Point Exceptions within the TcCOM object during the update phase can be set.

Options: "CallerExceptions", "Exceptions", "NoExceptions"

NoException: Floating Point Exceptions are disabled.

Exceptions: Floating Point Exceptions are enabled.

CallerExceptions: The settings of the caller are adopted, e.g. the task, another TcCOM or the PLC.

TcCom.OnlineChange

If TRUE, then the TcCOM can be replaced by online change while TwinCAT XAR is running. See also Online Change for Target for Simulink®.

PlcFb.Generate

If TRUE, a function block is created in the PLC library, which can be used in the TwinCAT XAE.

PlcFb. FpExceptionsForInit

Floating Point Exceptions within the function block during the init phase can be set.

Options: "CallerExceptions", "Exceptions", "NoExceptions"

NoException: Floating Point Exceptions are disabled.
Settings of the TwinCAT module generator

Exceptions: Floating Point Exceptions are enabled.

CallerExceptions: The settings of the caller are adopted, e.g. the task, another TcCOM or the PLC.

PlcFB. FpExceptionsForUpdate

Floating Point Exceptions within the function block during the update phase can be set.

Options: "CallerExceptions", "Exceptions", "NoExceptions"

NoException: Floating Point Exceptions are disabled.

Exceptions: Floating Point Exceptions are enabled.

CallerExceptions: The settings of the caller are adopted, e.g. the task, another TcCOM or the PLC.

NOTE

Execution stop

Floating Point Exceptions are active by default in TwinCAT. Comparisons or other operations with NaN (Not a Number) can cause such an exception that leads to an execution stop and may possibly cause machine damage. It is urgently recommended to check the result for NaN before it is processed.

Creating and loading an export configuration

When using the TwinCAT Target for MATLAB®, the MATLAB Coder™ is first used to generate C++ sources. These C++ sources can then be combined into an export configuration in the TwinCAT module generator by:

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

The path to the C++ sources created by the MATLAB Coder™ is passed as properties with 'BuildDir' and the name of the MATLAB® function is passed with 'MFile'. If, for example, BaseStatistics is selected as the name, the TcCOM object will have this name and the function block in the PLC will be given the name FB_BaseStatistics.

If the TwinCAT Target for Simulink® is used, the approach is somewhat different. Start the build process from Simulink® with the "Run the publish step after project generation" option disabled. Then load the created <modelname>_tcgrt folder as follows in order to add the C++ sources of the Simulink® model to the project export configuration.

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

Creating TwinCAT objects with the Module Generator

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

TwinCAT.ModuleGenerator.ProjectExporter() triggers the build process for the platforms set in the Project property. The object of the class TwinCAT.ModuleGenerator.ProjectExportConfig is passed as argument. This creates a folder on the local file system in the repository and stores the created drivers and description files.

Sample call:

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

7.1 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.

To do this, use the Project property of the module generator:

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

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.
```
8 Application of modules in TwinCAT

8.1 Working with the TcCOM module

Using TcCOM objects in TwinCAT 3
1. Insert a new TcCOM object.

2. Select the corresponding TcCOM object:
3. Create a new cyclic task of type *TwinCAT Task*.

4. Assign the newly created task to the newly created TcCOM object. To do this, go to the instance of the TcCOM object and select the *Context* tab.

You can now activate the configuration. In order to connect the TcCOM object with other modules in your TwinCAT solution beforehand, you can use the process image to create mappings.

You can view the MATLAB® code on the Block Diagram tab and observe and scope values on the fly. See MATLAB code representation [35].

### 8.1.1 MATLAB code representation

#### 8.1.1.1 MATLAB®-TcCOM

If a TwinCAT object was created with the TwinCAT Target for MATLAB® and the MATLAB® code export was executed, the MATLAB® code of the MATLAB® function can be displayed as a control in TwinCAT XAE.
8.1.1.1 Operation of the block diagram window

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

On the top level you will find the created module in block representation. Select the gray arrow in the block to display the content.

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>Shortcut</td>
<td>Function</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------</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><em>(System- &gt; Real-Time -&gt; C++ Debugger -&gt; Enable C++ Debugger must be activated)</em></td>
</tr>
</tbody>
</table>

**Context menu functions:**

- Fit to view
- 100 %
- Zoom +
- Zoom -
- Hide online values
- Disable debugging
- Provide exception data
- Save block diagram to image

### 8.1.1.1.2 Display of signal curves

Selected variables can be retrieved in TwinCAT XAE via ADS. It is therefore possible to display them in a mini-scope within the block diagram, or with the TwinCAT Scope within a measurement project.

Variables that can be displayed in scope have a trailing black frame in the code display. In this frame, the values are displayed in blue during operation.

Drag&Drop a "blue variable" onto the block diagram window to open a Mini.Scope.
By dragging and dropping a "blue variable" onto the Axis Group of a chart in the TwinCAT Measurement project, the variables are added to the TwinCAT Scope.

Which variables are visible as "blue variables" in TwinCAT XAE?

- The input variables
- The output variables
- Persistent variables (MATLAB definition `persistent var1 ... varN`)
8.2 Working with the PLC library

Use PLC library in TwinCAT 3

✓ Starting from a new TwinCAT solution, create a PLC project:
1. Perform the following menu steps.
2. Then add the newly created (and already installed) PLC library:
3. Get an overview of the data types and function blocks:
4. Insert instances of the function blocks into your PLC and use them in your application:

8.2.1 Online change of the PLC library

While TwinCAT is in run mode, you can exchange the PLC library version in TwinCAT XAE and load it into the running application via Online Change. This means that all function blocks in a PLC library can be updated without a TwinCAT restart.
Step-by-step procedure:
1. Create a first PLC library version with the TwinCAT Target for MATLAB®.
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 MATLAB function(s) and create a PLC library version (0.0.0.2) from it.
5. Select the newly created PLC library version in TwinCAT - PLC - 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).

8.3 Debugging

The following step-by-step instructions apply equally to the use of TcCOM objects and function blocks created with Target for MATLAB®. The following shows the debugging for a PLC function block.

✓ Step-by-step procedure:
1. Make sure that your TwinCAT application has been activated with the C++ debugger enabled.

2. Open the TwinCAT C++ project created during code generation that belongs to the module you want to debug.
   - You specified the project location when initializing the project export configuration, see Overview of automatically generated files [26].
   - You can open the Visual Studio project directly, or add it to your TwinCAT solution under C++ with "Add existing Item".
3. In the MATLAB® folder in the Visual Studio solution, view the subfolders that bear the name of the MATLAB® function created.

   In the Sources sub-folder, you can find the executed code generated by MATLAB Coder™.

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.
Application of modules in TwinCAT
9 FAQ

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.
## 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:

```matlab
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 are

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

For displaying and starting individual samples. The argument `SampleName` is to be passed as a string, e.g.

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

### 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++, .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.

### 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.

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 `\TE1400\Sample_TemperatureController_PrecompiledTcComModules\Actual TwinCAT versions` into the publish directory.

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

#### The MATLAB® class

##### The properties

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

##### The constructor

```
function this = Tc3AutomationInterface
```

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

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

##### Selected methods of the class

```
function TcComObject = CreateTcCOM(this, Modelname)
```
Use the MATLAB® help functions in order to view the function and the parameters of the method.

```matlab
>> help Tc3_AI.CreateTcCOM
--- help for Tc3AutomationInterface/CreateTcCOM ---

CreateTcCOM creates a new instance of a TcCOM

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

set properties: TcCOM

see also:
Beckhoff Infosys
```

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

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

```matlab
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.

```matlab
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)
More Information:
www.beckhoff.com/TE1401