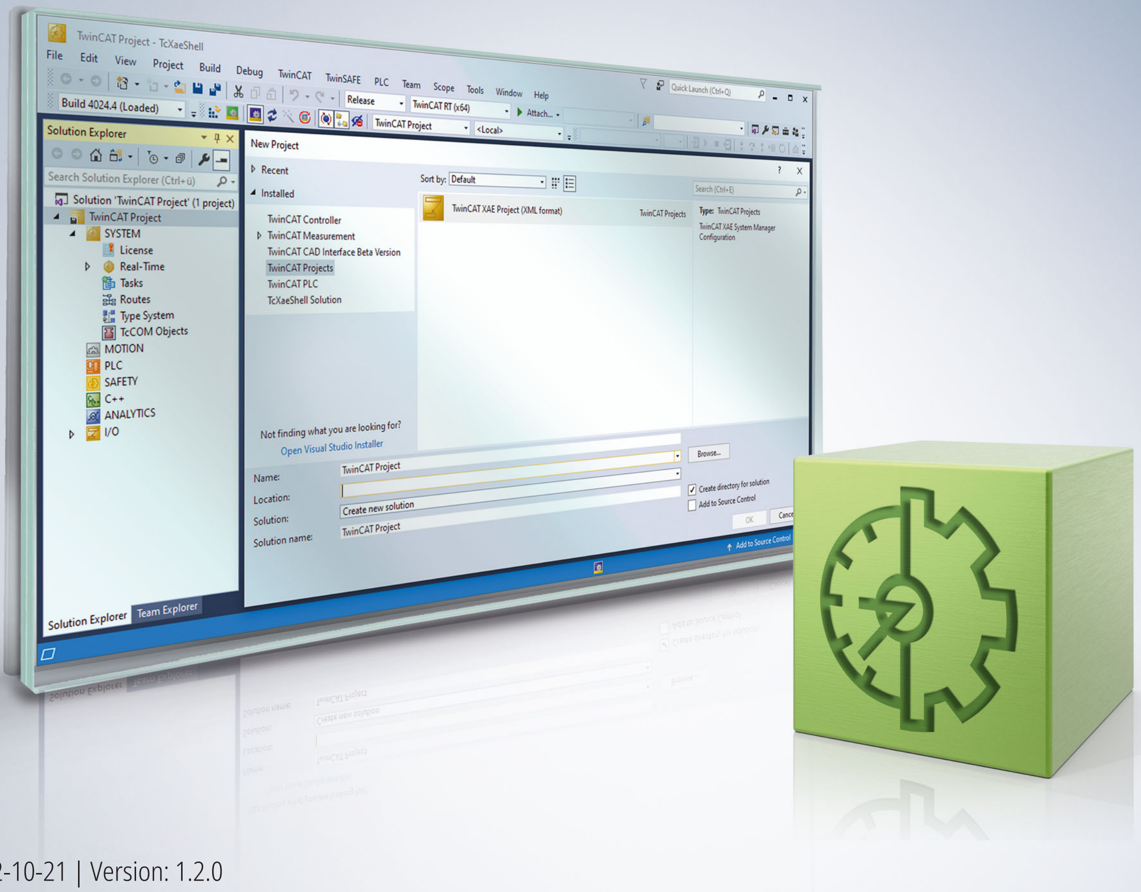
BECKHOFF New Automation Technology

Manual | EN

TE1401

TwinCAT 3 | Target for MATLAB®



2022-10-21 | Version: 1.2.0

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with 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:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

ℹ **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.

2 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++*.

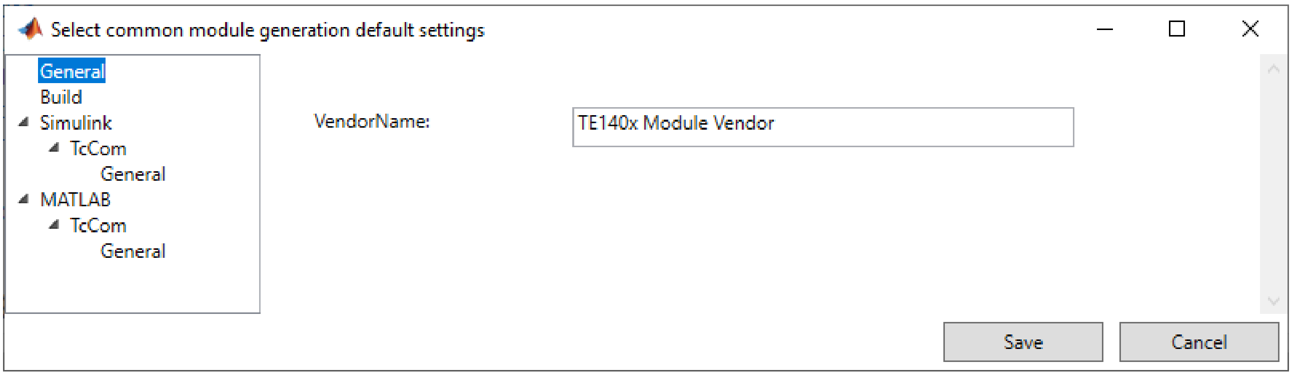
1. Start **TwinCAT 3 XAE or Full Setup**, if it does not already exist.
   If a Visual Studio and a TwinCAT installation already exists but the Visual Studio version does not meet the requirements mentioned above (e.g. TwinCAT XAE Shell or Visual Studio without C++ option), you first have to install 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.
3. Start **TwinCAT Tools for MATLAB® and Simulink®** Setup to install TE1401 TwinCAT Target for MATLAB®.
   ⇨ 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 *%TwinCAT3Dir%.. \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 directly or make/change them at a later time.



Select common module generation default settings

- General
- Build
- Simulink
  - TcCom
    - General
- MATLAB
  - TcCom
    - General

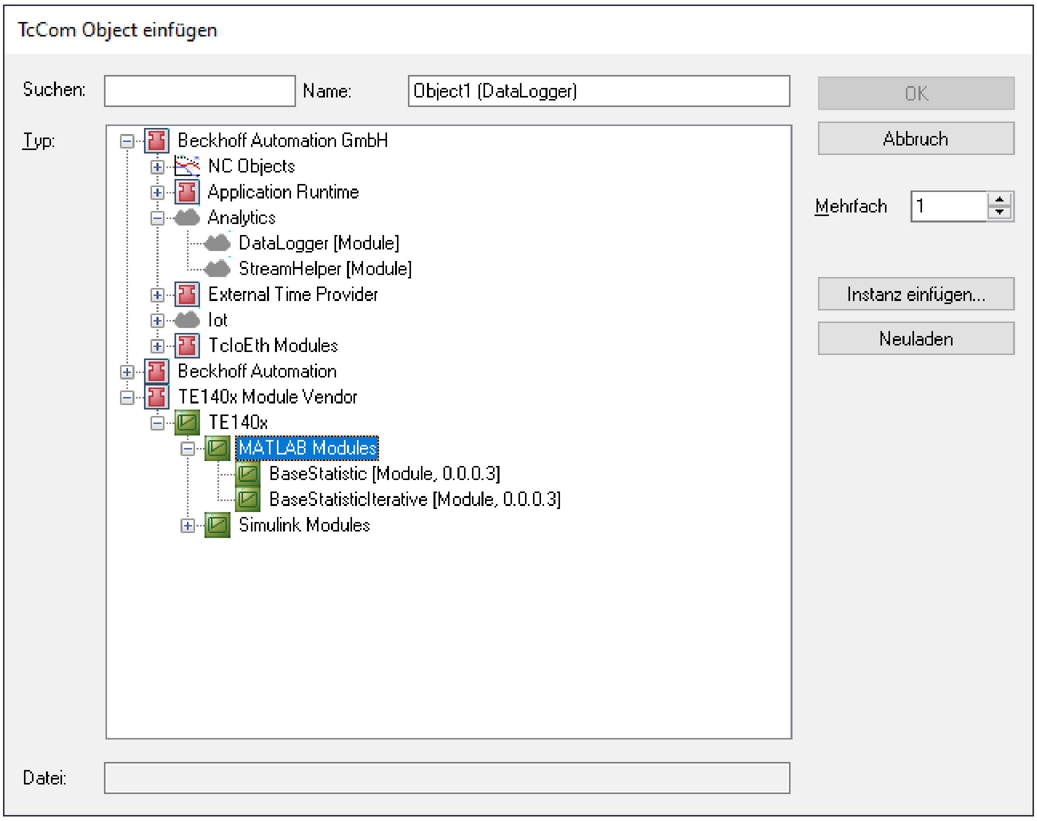
VendorName: TE140x Module Vendor

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®.



TcCom Object einfügen

Suchen:
Name: Object1 (DataLogger)

Typ:

- Beckhoff Automation GmbH
  - NC Objects
  - Application Runtime
  - Analytics
    - DataLogger [Module]
    - StreamHelper [Module]
  - External Time Provider
  - Iot
  - TcIoEth Modules
- Beckhoff Automation
- TE140x Module Vendor
  - TE140x
    - MATLAB Modules
      - BaseStatistic [Module, 0.0.0.3]
      - BaseStatisticIterative [Module, 0.0.0.3]
    - Simulink Modules

Mehrfach 1

Datei:

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.

3.1 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\`

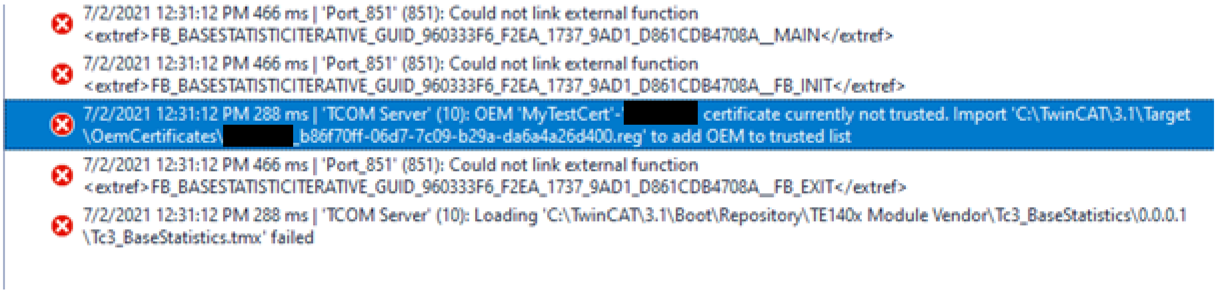
Operating TcSignTool from MATLAB®

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).



```
7/2/2021 12:31:12 PM 466 ms | 'Port_851' (851): Could not link external function
<extref>FB_BASESTATISTICITERATIVE_GUID_960333F6_F2EA_1737_9AD1_D861CDB4708A__MAIN</extref>
7/2/2021 12:31:12 PM 466 ms | 'Port_851' (851): Could not link external function
<extref>FB_BASESTATISTICITERATIVE_GUID_960333F6_F2EA_1737_9AD1_D861CDB4708A__FB_INIT</extref>
7/2/2021 12:31:12 PM 288 ms | 'TCOM Server' (10): OEM 'MyTestCert'-[redacted] certificate currently not trusted. Import 'C:\TwinCAT\3.1\Target
\OemCertificates\[redacted]_b86f70ff-06d7-7c09-b29a-da6a4a26d400.reg' to add OEM to trusted list
7/2/2021 12:31:12 PM 466 ms | 'Port_851' (851): Could not link external function
<extref>FB_BASESTATISTICITERATIVE_GUID_960333F6_F2EA_1737_9AD1_D861CDB4708A__FB_EXIT</extref>
7/2/2021 12:31:12 PM 288 ms | 'TCOM Server' (10): Loading 'C:\TwinCAT\3.1\Boot\Repository\TE140x Module Vendor\Tc3_BaseStatistics\0.0.0.1
\Tc3_BaseStatistics.tmx' failed
```

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

Registry file is only dependent on the OEM certificate

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.

3.1.1 Signed TwinCAT user certificates for delivery without test mode

System requirements

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

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

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

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

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

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

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

Validity of the TwinCAT 3 user certificate

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

What happens if the certificate has expired?

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

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

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

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

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

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

3.1.1.1 Request TwinCAT 3 user certificate

Overview of the ordering and validation process

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

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

Overview of the ordering and validation process

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

1. Contact your Beckhoff sales contact and announce the request of a TwinCAT 3 OEM certificate. Order "TC0007" or "TC0008".
2. Important: as the inquirer, please provide your contact details as the delivery address (= contact name and email address) and the area of use of the certificate (company name, address).
3. The contact details provided in the order will be verified and you (the inquirer named in the delivery address) will be contacted by Beckhoff Sales.
4. When requesting a new OEM certificate, Creation of the Certificate Request file for TC0008 [▶ 14].
5. Determine the "File Fingerprint" of the OEM certificate file using TwinCAT Engineering (see Determining the file fingerprint of the OEM certificate file [▶ 17]). Please inform the Beckhoff sales contact of this File Fingerprint as part of your contact data verification. The transmission of the File Fingerprint must be done by a different communication channel than the one used for sending the OEM certificate request file.
6. Now send the "OEM certificate file" to the Beckhoff sales contact.
7. After signing the certificate file at the Beckhoff headquarters, you will receive it by e-mail from your contact person.

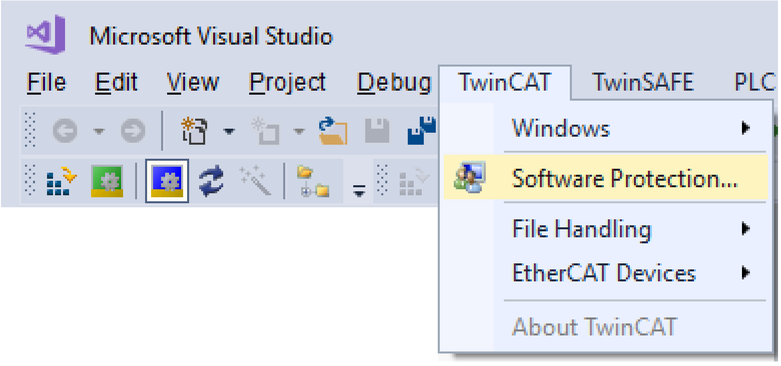
Please note that it may take a few days to validate your contact details and issue the certificate.

3.1.1.2 Creation of the Certificate Request file for TC0008

System requirements

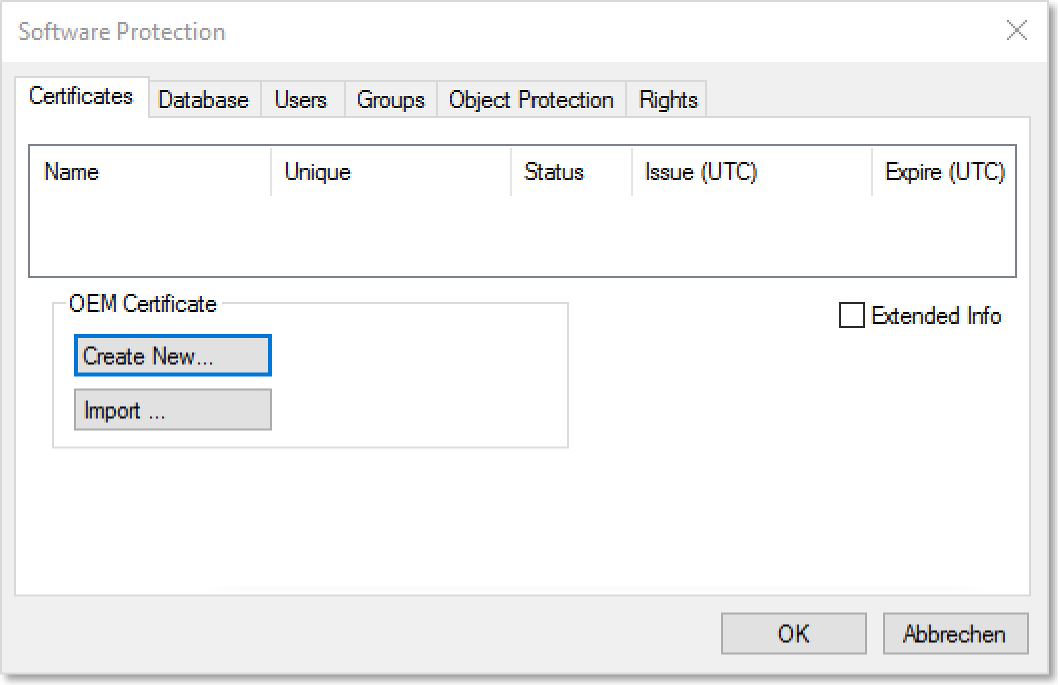
- Min. TwinCAT 3.1 Build 4024
- Min. Windows 10 or TwinCAT/BSD (on the target system)

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



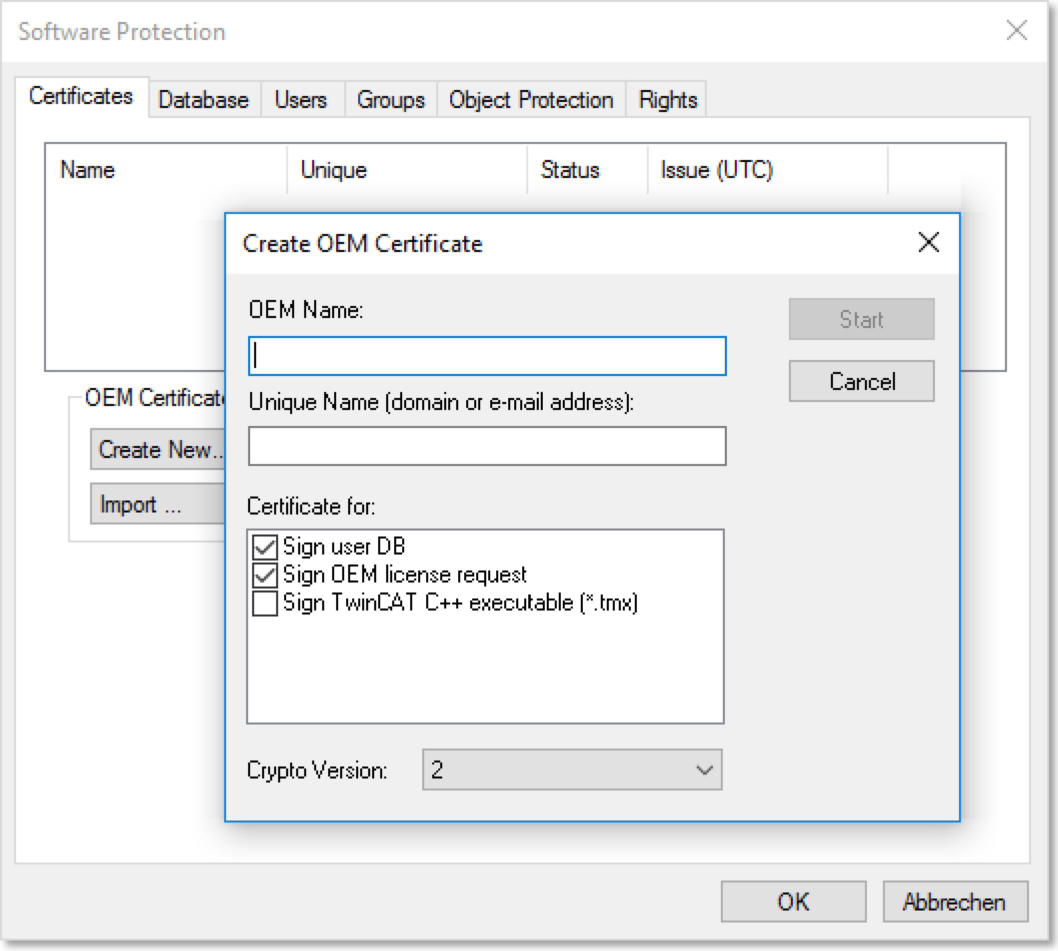
In the window that opens, select the **Certificates** tab.

Click **Create New….**:



□ Extended Info

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



OEM Name:

Unique Name (domain or e-mail address):

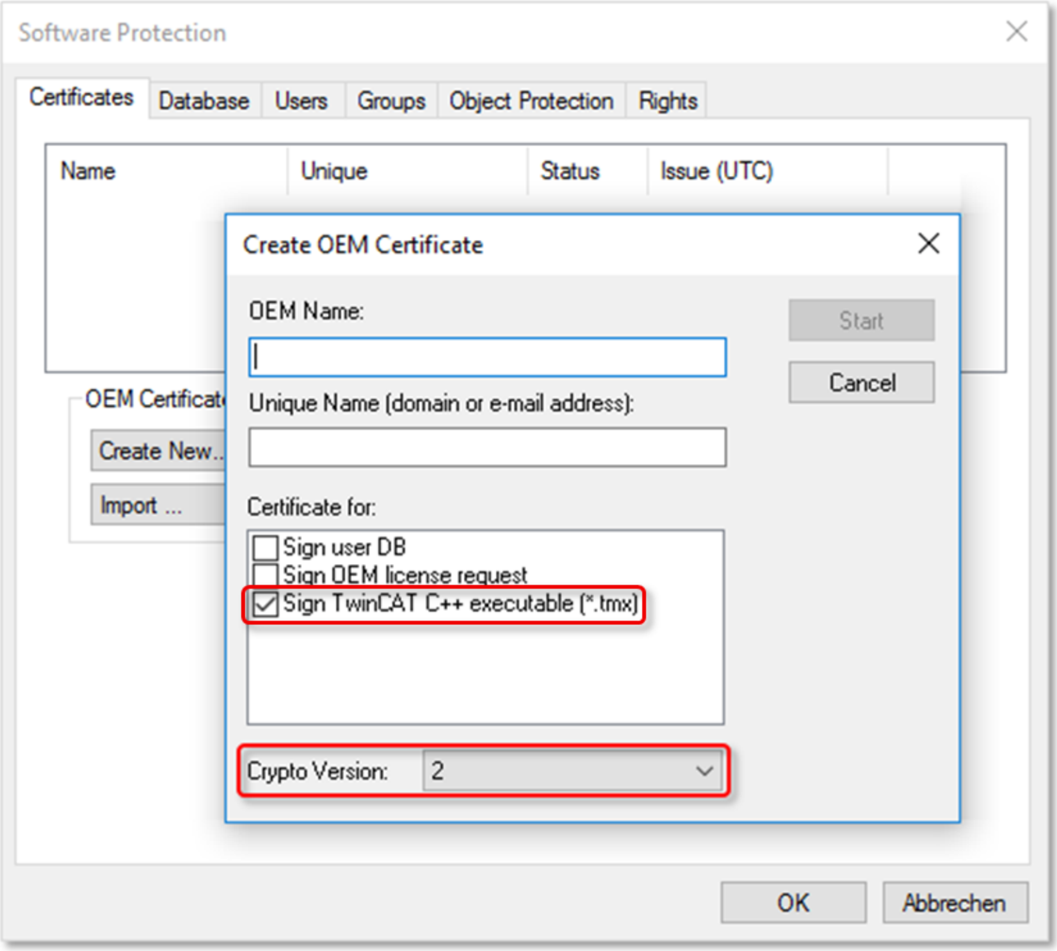
Certificate for:

☑ Sign user DB
☑ Sign OEM license request
□ Sign TwinCAT C++ executable (*.tmx)

Crypto Version: 2

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

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



OEM Name:

Unique Name (domain or e-mail address):

Certificate for:

☐ Sign user DB
☐ Sign OEM license request
☑ Sign TwinCAT C++ executable (*.tmx)

Crypto Version: 2

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

- Make sure that Crypto version 2 (for the encrypted content of the certificate content) is set. (standard setting)

2. Once you have entered the data, click **Start** and select a directory to save the file. **Note:** You can simply accept the suggested directory "c:\twincat\3.1\customconfig\certificates". You need the newly created file in this directory in order to be able to read out the file fingerprint for this file [▶ 17] in a subsequent step.

   ⇨ A dialog for selecting a password for the OEM Private Key opens.

3. Issue a password for the OEM Private Key**.**

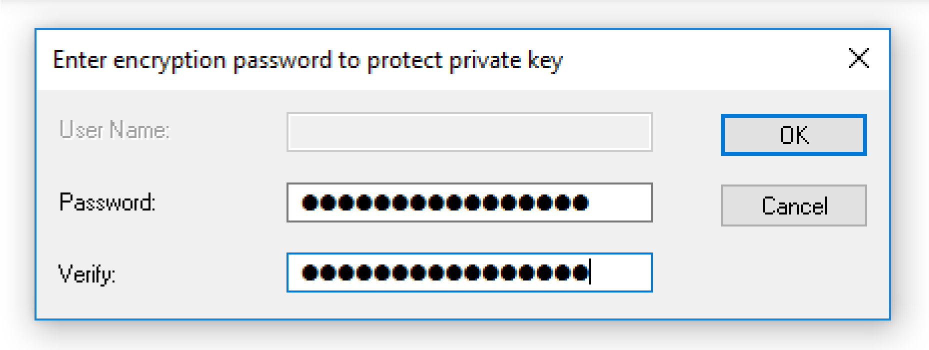
Important: Password security!

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

Password cannot be restored if lost

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

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



Enter encryption password to protect private key

User Name:

Password: ●●●●●●●●●●●●●●●●●

Verify: ●●●●●●●●●●●●●●●●●

⇨ The file is saved.

The "Certificate Request File" generated in this way must now be signed by the Beckhoff certificate section in order to be valid. The procedure is described in chapter Requesting a certificate [▶ 13].

3.1.1.3 Determining the file fingerprint of the OEM certificate file

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

System requirements

This functionality requires TwinCAT 3.1 build 4024 of higher.

Note for "OEM Certificate Request File"

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

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

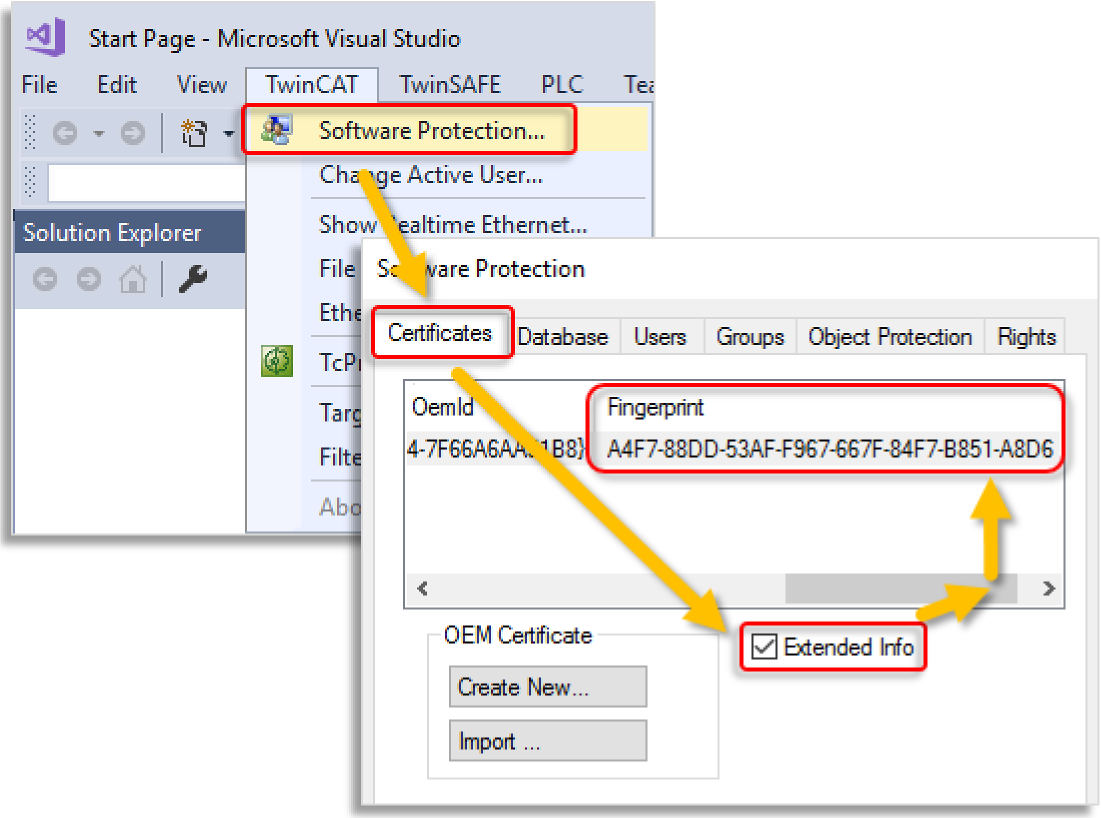
For this function it is necessary that the OEM certificate file is located in this directory: "c:\twincat\3.1\customconfig\certificates".

Notes:

- This directory contains your OEM certificate, if you already have a certificate and want to renew it.
- If you did not change the suggested directory when creating the "OEM Certificate Request File", the file is already in this directory.

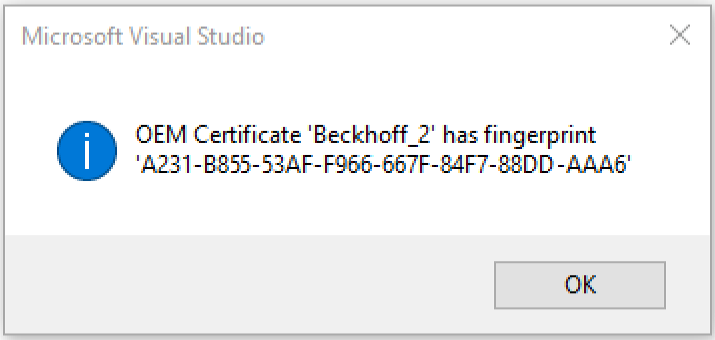
Procedure:

1. Call up the TwinCAT 3 Software Protection configurator



2. Select the "Certificates" tab
3. Check the "Extended Info" box
4. In the window scroll to the right until you see the Fingerprint column

Note: As an alternative to points 3 + 4, you can simply double-click the certificate line. The file fingerprint is then displayed in a pop-up window:



OEM Certificate 'Beckhoff_2' has fingerprint 'A231-B855-53AF-F966-667F-84F7-88DD-AAA6'

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

3.1.1.4 Saving the signed TwinCAT user certificate

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

System requirements

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

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

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

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

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

4 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.

5 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

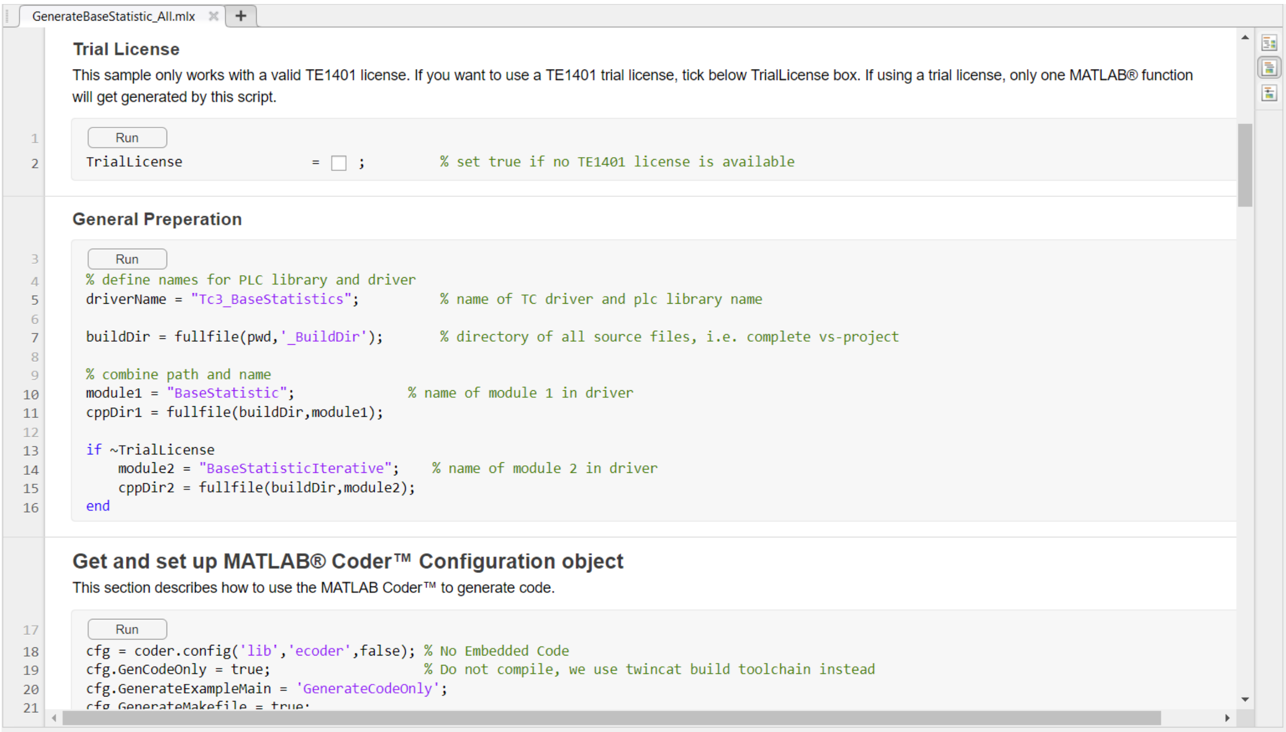
The following video (only available in English) can also be used as an introduction: TwinCAT Target for MATLAB®.

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.
3. 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.



Trial License

This sample only works with a valid TE1401 license. If you want to use a TE1401 trial license, tick below TrialLicense box. If using a trial license, only one MATLAB® function will get generated by this script.

```
TrialLicense             = □ ;        % set true if no TE1401 license is available
```

General Preperation

```
% define names for PLC library and driver
driverName = "Tc3_BaseStatistics";         % name of TC driver and plc library name

buildDir = fullfile(pwd,'_BuildDir');      % directory of all source files, i.e. complete vs-project

% combine path and name
module1 = "BaseStatistic";             % name of module 1 in driver
cppDir1 = fullfile(buildDir,module1);

if ~TrialLicense
    module2 = "BaseStatisticIterative";    % name of module 2 in driver
    cppDir2 = fullfile(buildDir,module2);
end
```

Get and set up MATLAB® Coder™ Configuration object

This section describes how to use the MATLAB Coder™ to generate code.

```
cfg = coder.config('lib','ecoder',false); % No Embedded Code
cfg.GenCodeOnly = true;                   % Do not compile, we use twincat build toolchain instead
cfg.GenerateExampleMain = 'GenerateCodeOnly';
cfg.GenerateMakefile = true;
```

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 [▸ 20] 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:

- File path in the Engineering Repository: %TwinCATInstallDir% \3.1\Repository\<TE140x Module Vendor>**Tc3_BaseStatistics**\<Version>\
- 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

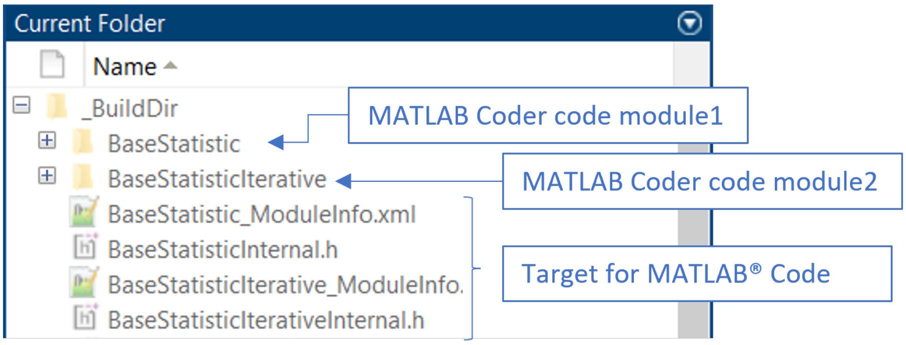
Where should all source files be stored?

`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?

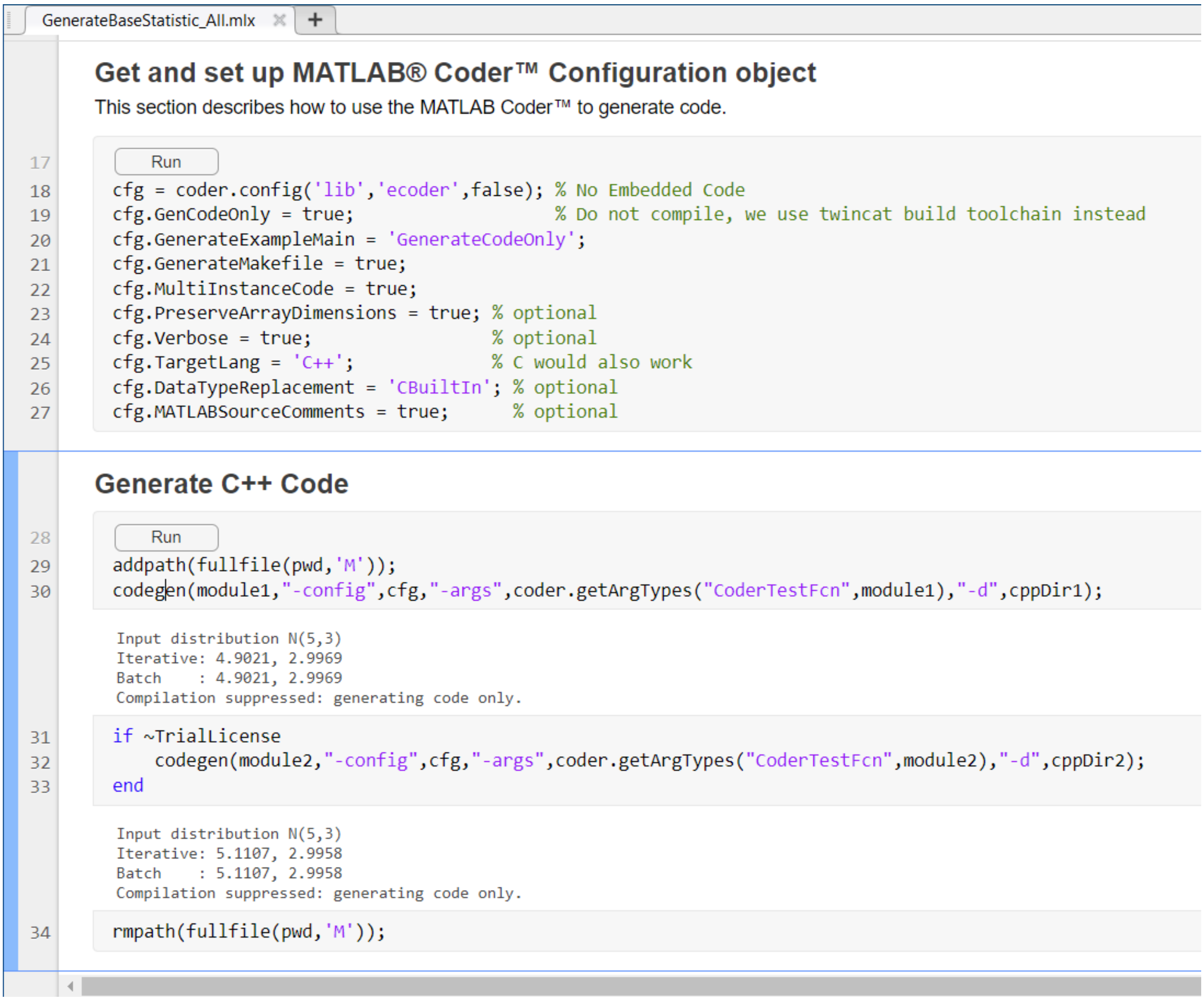
The MATLAB® functions are named here with the variables `module1` and `module2`; the two MATLAB® functions BaseStatistics and BaseStatisticsIteravtive (stored in subfolder M) are to be transferred to TwinCAT objects accordingly.

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™.


Current Folder
Name
_BuildDir
BaseStatistic
BaseStatisticIterative
BaseStatistic_ModuleInfo.xml
BaseStatisticInternal.h
BaseStatisticIterative_ModuleInfo.
BaseStatisticIterativeInternal.h
MATLAB Coder code module1
MATLAB Coder code module2
Target for MATLAB® Code

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.



Get and set up MATLAB® Coder™ Configuration object

This section describes how to use the MATLAB Coder™ to generate code.

```
cfg = coder.config('lib','ecoder',false); % No Embedded Code
cfg.GenCodeOnly = true;                   % Do not compile, we use twincat build toolchain instead
cfg.GenerateExampleMain = 'GenerateCodeOnly';
cfg.GenerateMakefile = true;
cfg.MultiInstanceCode = true;
cfg.PreserveArrayDimensions = true; % optional
cfg.Verbose = true;                 % optional
cfg.TargetLang = 'C++';             % C would also work
cfg.DataTypeReplacement = 'CBuiltIn'; % optional
cfg.MATLABSourceComments = true;      % optional
```

Generate C++ Code

```
addpath(fullfile(pwd,'M'));
codegen(module1,"-config",cfg,"-args",coder.getArgTypes("CoderTestFcn",module1),"-d",cppDir1);
```

```
Input distribution N(5,3)
Iterative: 4.9021, 2.9969
Batch    : 4.9021, 2.9969
Compilation suppressed: generating code only.
```

```
if ~TrialLicense
    codegen(module2,"-config",cfg,"-args",coder.getArgTypes("CoderTestFcn",module2),"-d",cppDir2);
end
```

```
Input distribution N(5,3)
Iterative: 5.1107, 2.9958
Batch    : 5.1107, 2.9958
Compilation suppressed: generating code only.
```

```
rmpath(fullfile(pwd,'M'));
```

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\ BaseStatisticIterative.

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 representation [▶ 43].

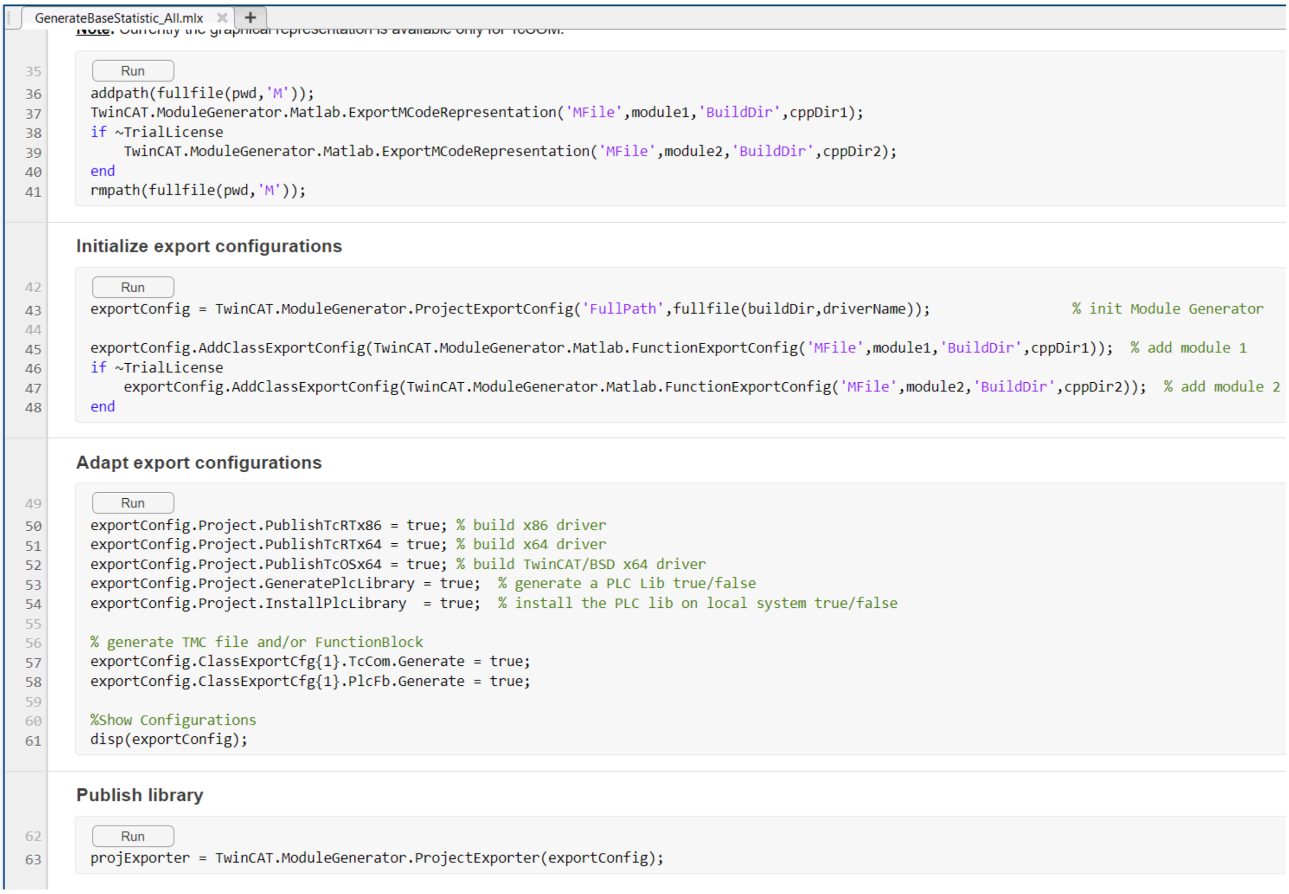
```
TwinCAT.ModuleGenerator.Matlab.ExportMCodeRepresentation('MFile',module1,'BuildDir',cppDir1);
```

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 [▶ 21].



Note: Currently the graphical representation is available only for TcCOM.

```
addpath(fullfile(pwd,'M'));
TwinCAT.ModuleGenerator.Matlab.ExportMCodeRepresentation('MFile',module1,'BuildDir',cppDir1);
if ~TrialLicense
    TwinCAT.ModuleGenerator.Matlab.ExportMCodeRepresentation('MFile',module2,'BuildDir',cppDir2);
end
rmpath(fullfile(pwd,'M'));
```

Initialize export configurations

```
exportConfig = TwinCAT.ModuleGenerator.ProjectExportConfig('FullPath',fullfile(buildDir,driverName));              % init Module Generator

exportConfig.AddClassExportConfig(TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig('MFile',module1,'BuildDir',cppDir1));  % add module 1
if ~TrialLicense
    exportConfig.AddClassExportConfig(TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig('MFile',module2,'BuildDir',cppDir2));  % add module 2
end
```

Adapt export configurations

```
exportConfig.Project.PublishTcRTx86 = true; % build x86 driver
exportConfig.Project.PublishTcRTx64 = true; % build x64 driver
exportConfig.Project.PublishTcOSx64 = true; % build TwinCAT/BSD x64 driver
exportConfig.Project.GeneratePlcLibrary = true;  % generate a PLC Lib true/false
exportConfig.Project.InstallPlcLibrary  = true;  % install the PLC lib on local system true/false

% generate TMC file and/or FunctionBlock
exportConfig.ClassExportCfg{1}.TcCom.Generate = true;
exportConfig.ClassExportCfg{1}.PlcFb.Generate = true;

%Show Configurations
disp(exportConfig);
```

Publish library

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

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.

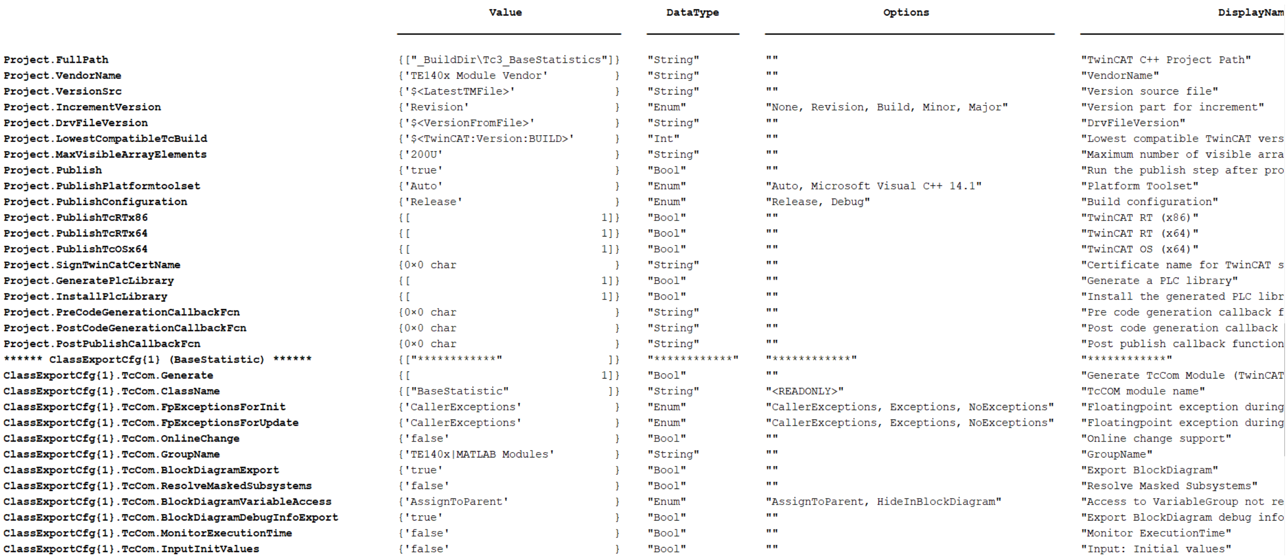
```
exportConfig.AddClassExportConfig(TwinCAT.ModuleGenerator.Matlab.FunctionExportConfig('MFile',module
1,'BuildDir',cppDir1));
```

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.



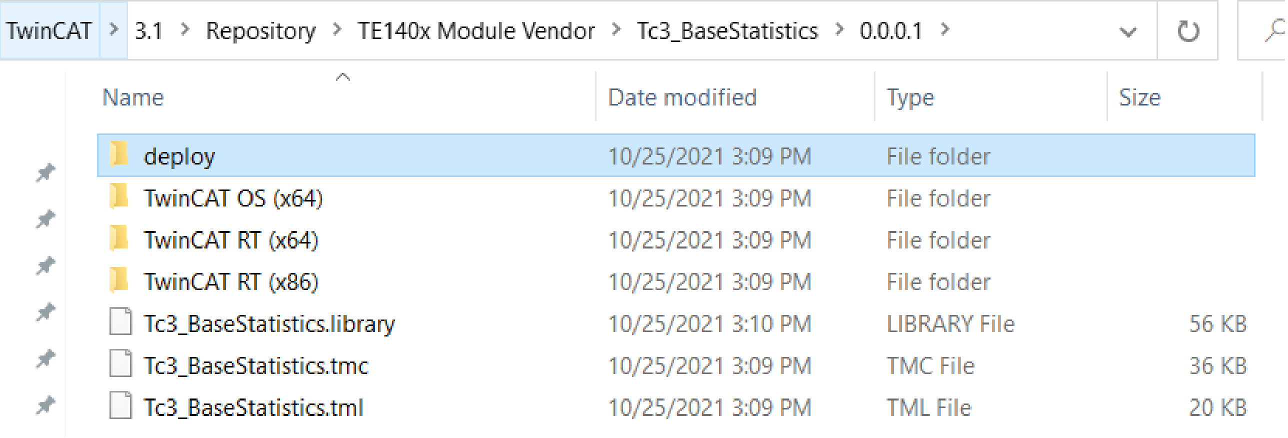
```
                                                              Value                          DataType                        Options                                    DisplayNam
                                                  _______________________________    _____________    _________________________________________    _________________________

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

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>



TwinCAT › 3.1 › Repository › TE140x Module Vendor › Tc3_BaseStatistics › 0.0.0.1 ›

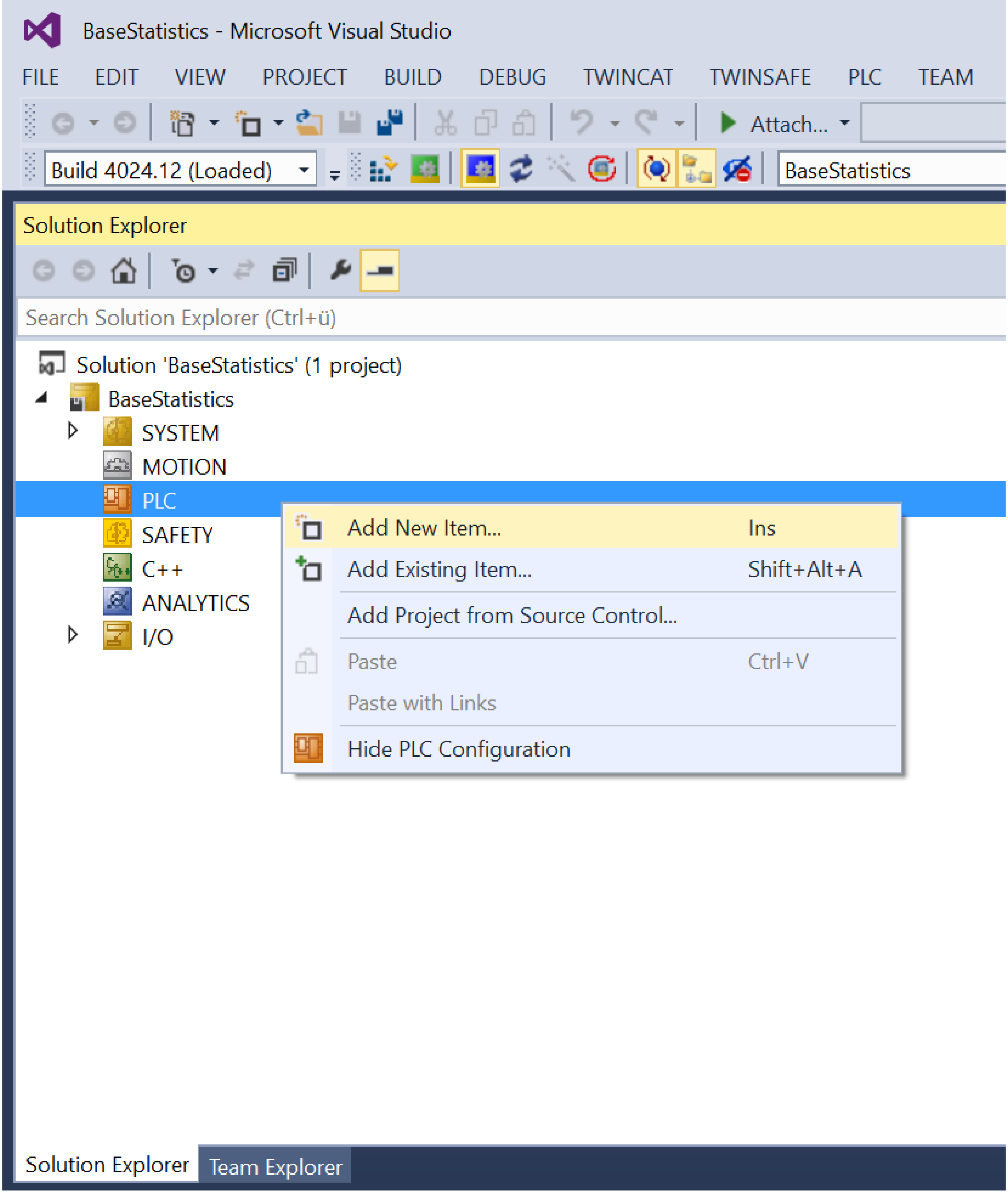
Name	Date modified	Type	Size
deploy	10/25/2021 3:09 PM	File folder	
TwinCAT OS (x64)	10/25/2021 3:09 PM	File folder	
TwinCAT RT (x64)	10/25/2021 3:09 PM	File folder	
TwinCAT RT (x86)	10/25/2021 3:09 PM	File folder	
Tc3_BaseStatistics.library	10/25/2021 3:10 PM	LIBRARY File	56 KB
Tc3_BaseStatistics.tmc	10/25/2021 3:09 PM	TMC File	36 KB
Tc3_BaseStatistics.tml	10/25/2021 3:09 PM	TML File	20 KB

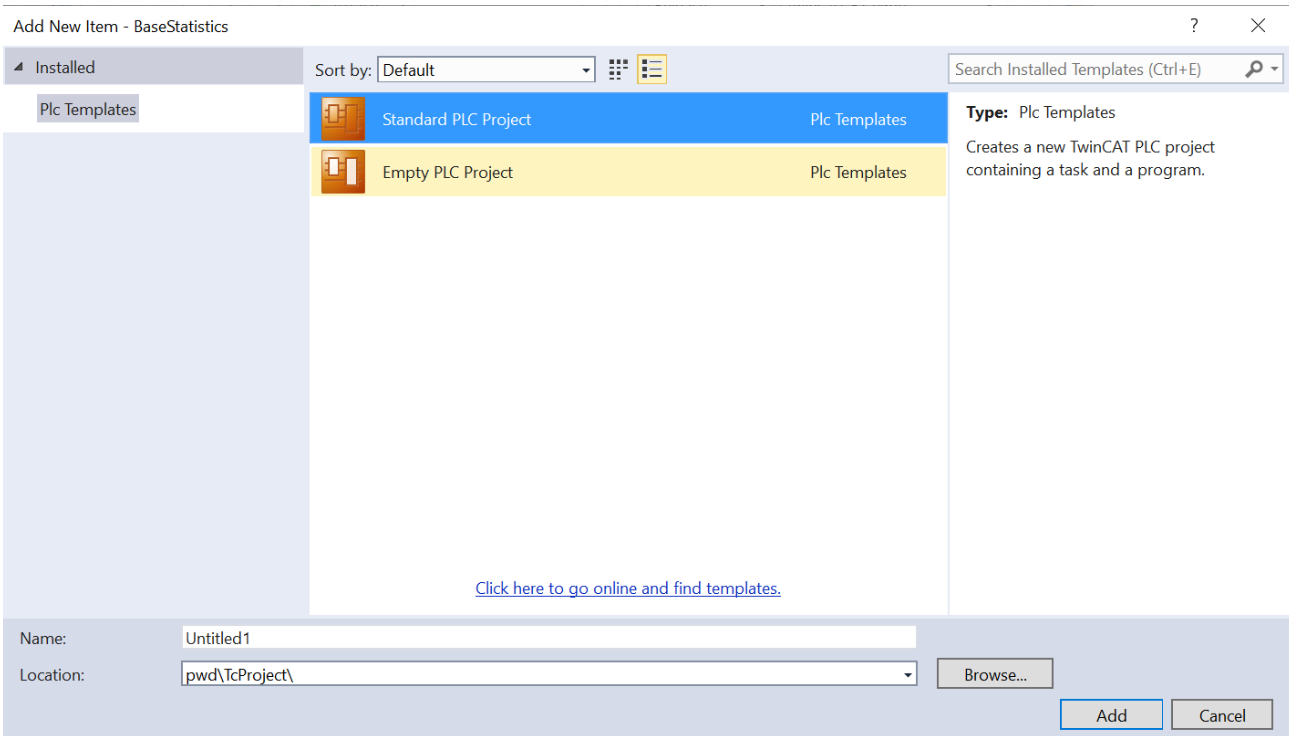
You can copy the folder to any number of TwinCAT XAE systems, so that the modules are available there. Only the *.library must be installed in TwinCAT via the PLC library repository. Please note that the folder structure is not changed during copying.

Use PLC library in TwinCAT 3

✓ Starting from a new TwinCAT solution, create a PLC project:

1. Perform the following menu steps.


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



Add New Item - BaseStatistics

Sort by: Default

Standard PLC Project	Plc Templates
Empty PLC Project	Plc Templates

Click here to go online and find templates.

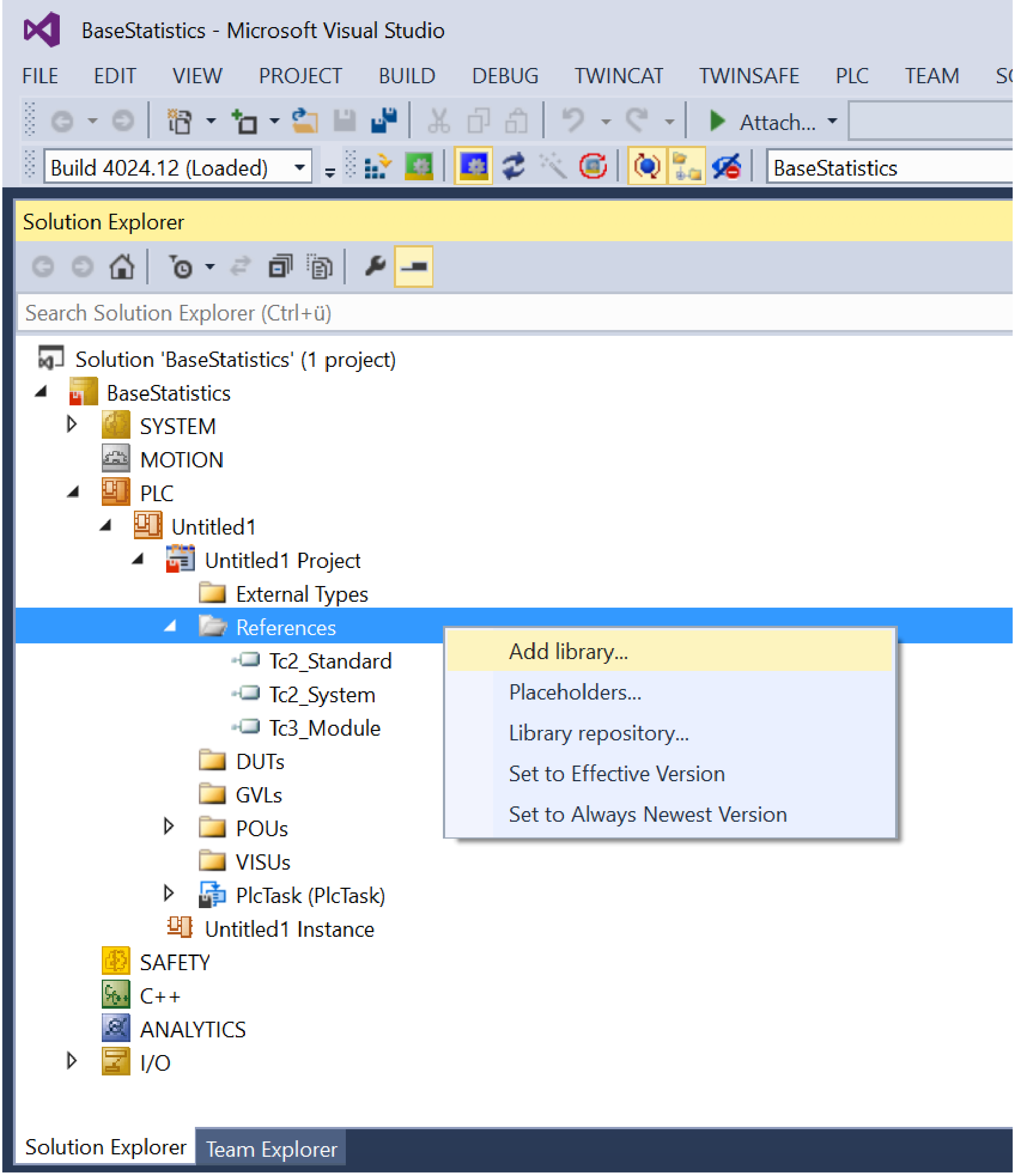
Type: Plc Templates

Creates a new TwinCAT PLC project containing a task and a program.

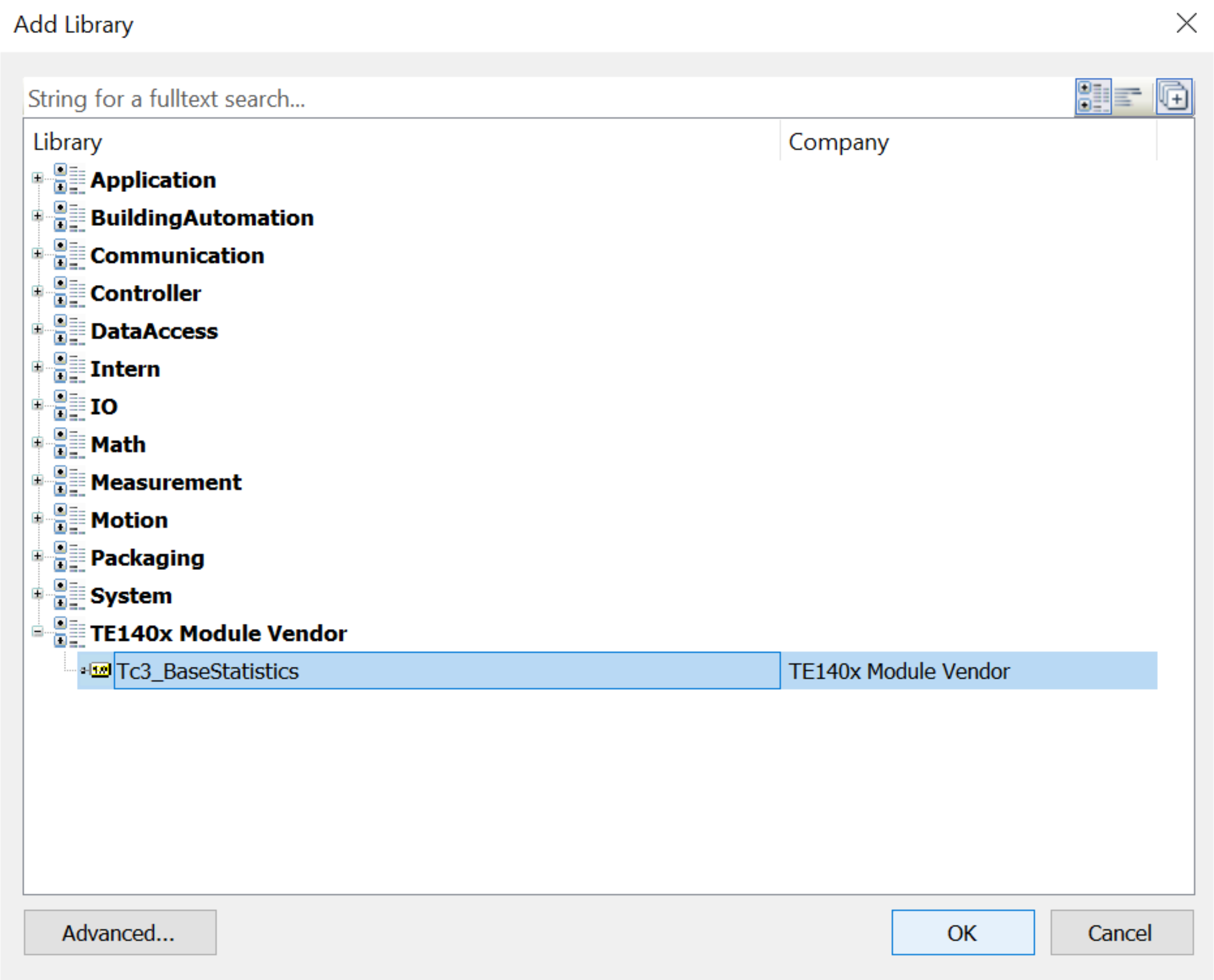
Name: Untitled1

Location: pwd\TcProject\

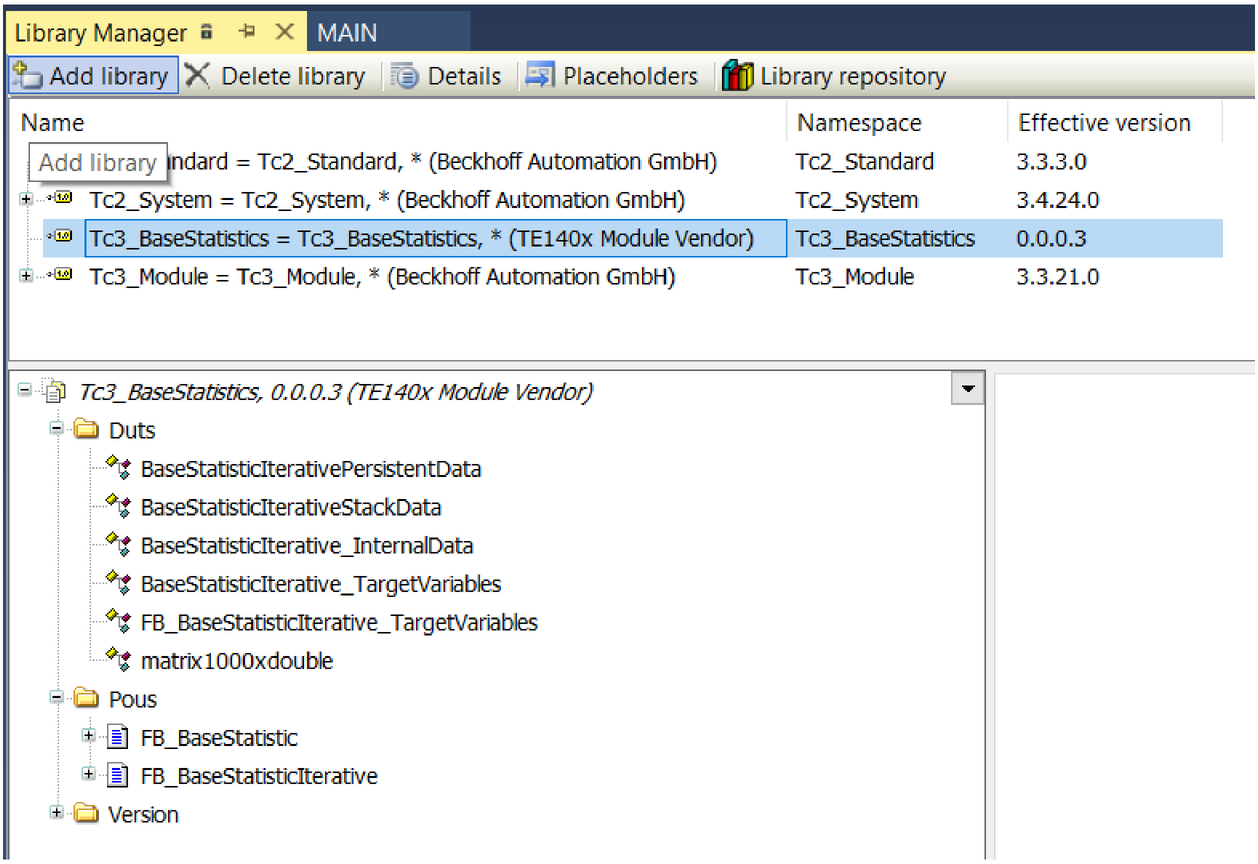
2. Then add the newly created (and already installed) PLC library:



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



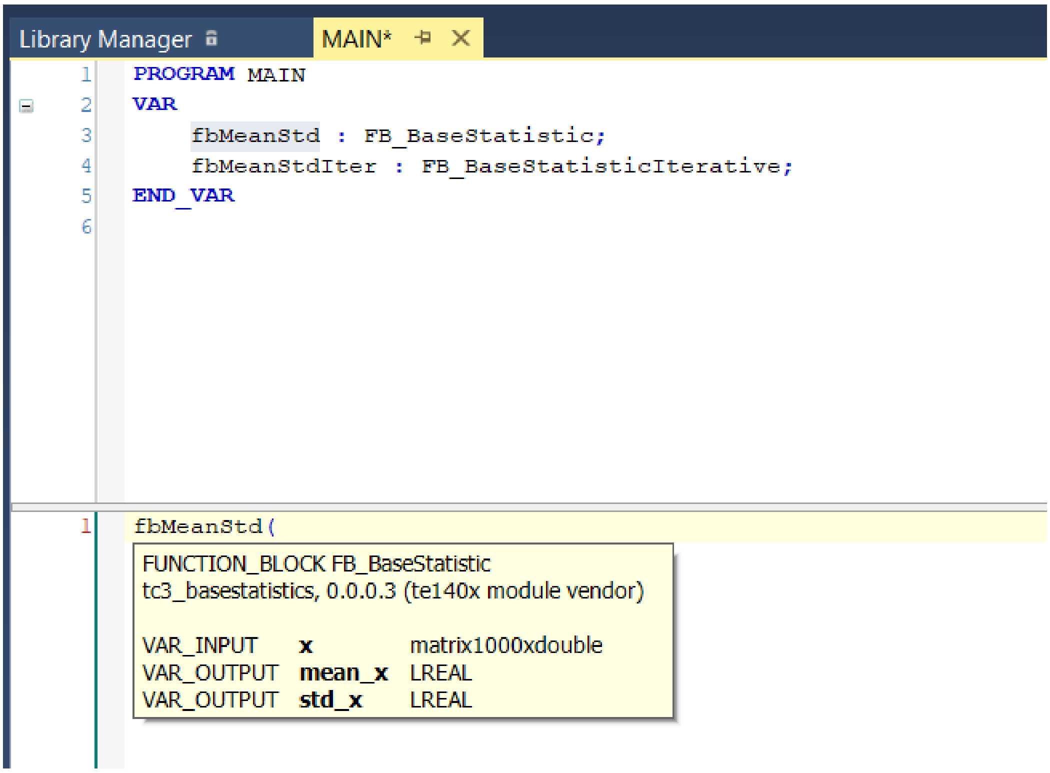
3. Get an overview of the data types and function blocks:



Name	Namespace	Effective version
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH)	Tc2_Standard	3.3.3.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH)	Tc2_System	3.4.24.0
Tc3_BaseStatistics = Tc3_BaseStatistics, * (TE140x Module Vendor)	Tc3_BaseStatistics	0.0.0.3
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH)	Tc3_Module	3.3.21.0

```
Tc3_BaseStatistics, 0.0.0.3 (TE140x Module Vendor)
    Duts
        BaseStatisticIterativePersistentData
        BaseStatisticIterativeStackData
        BaseStatisticIterative_InternalData
        BaseStatisticIterative_TargetVariables
        FB_BaseStatisticIterative_TargetVariables
        matrix1000xdouble
    Pous
        FB_BaseStatistic
        FB_BaseStatisticIterative
    Version
```

4. Insert instances of the function blocks into your PLC and use them in your application:



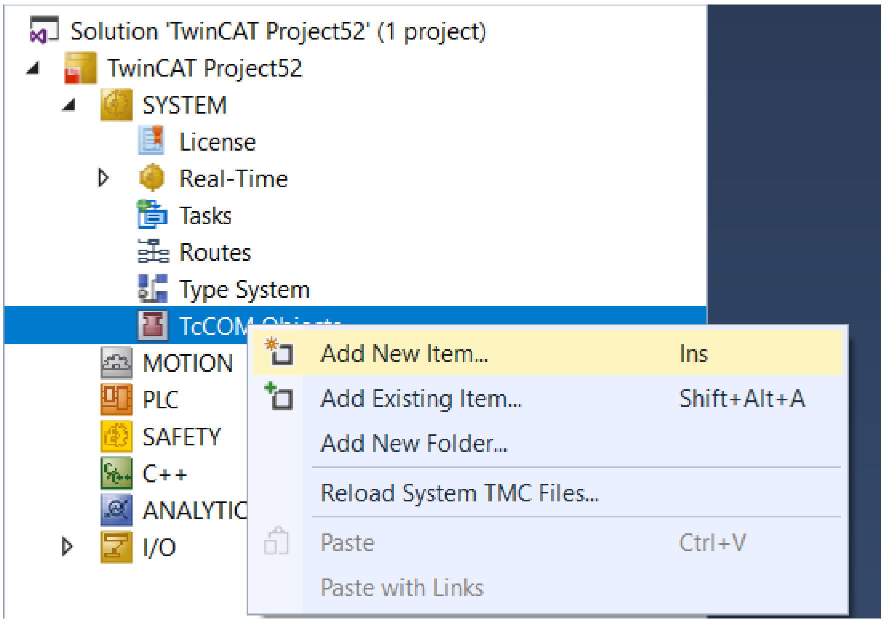
```
PROGRAM MAIN
VAR
    fbMeanStd : FB_BaseStatistic;
    fbMeanStdIter : FB_BaseStatisticIterative;
END_VAR
```

```
fbMeanStd(
```

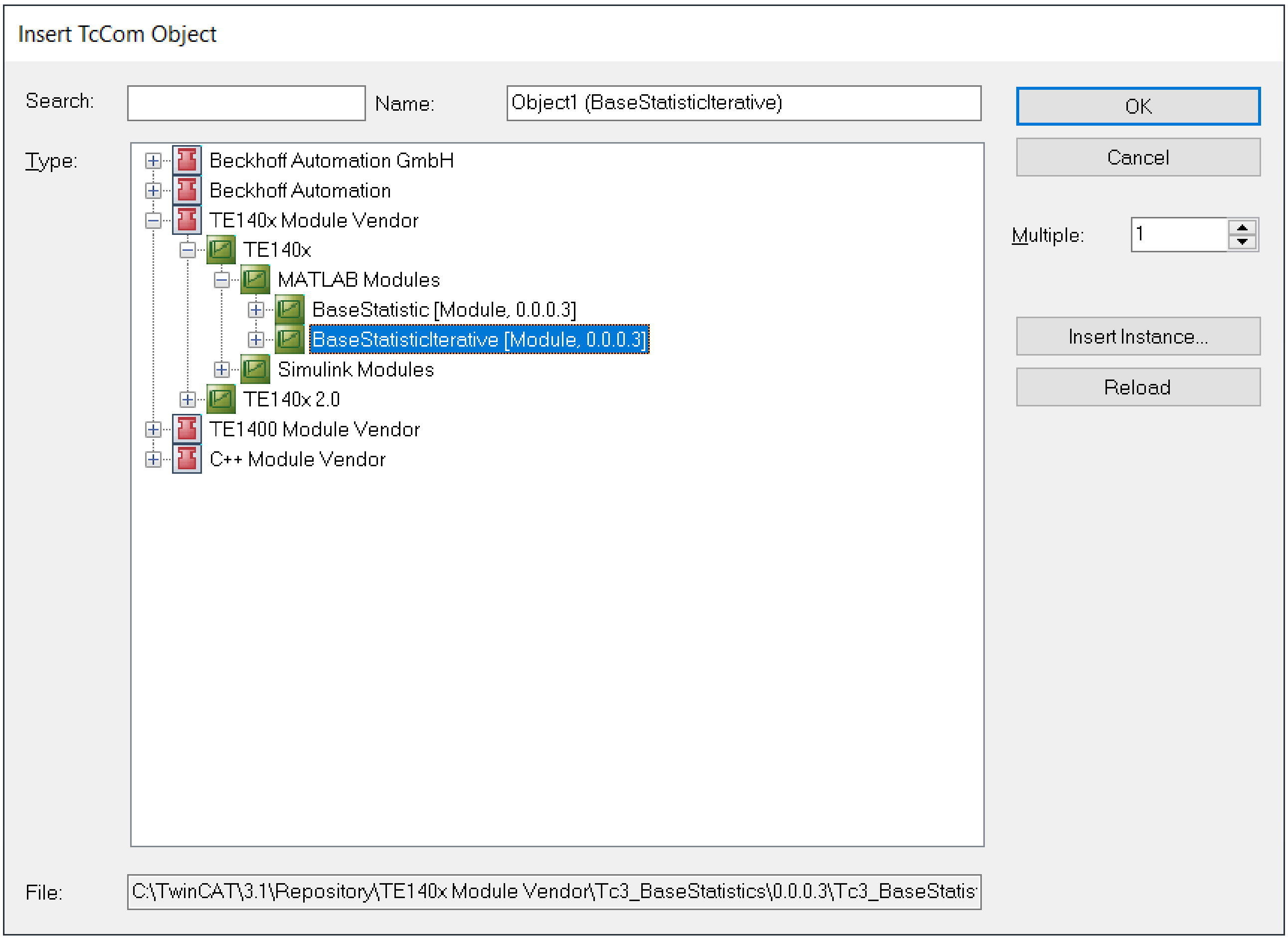
```
FUNCTION_BLOCK FB_BaseStatistic
tc3_basestatistics, 0.0.0.3 (te140x module vendor)

VAR_INPUT    x       matrix1000xdouble
VAR_OUTPUT   mean_x  LREAL
VAR_OUTPUT   std_x   LREAL
```

Using TcCOM objects in TwinCAT 3

5. Insert a new TcCOM object.



6. Select the corresponding TcCOM object:


Insert TcCom Object

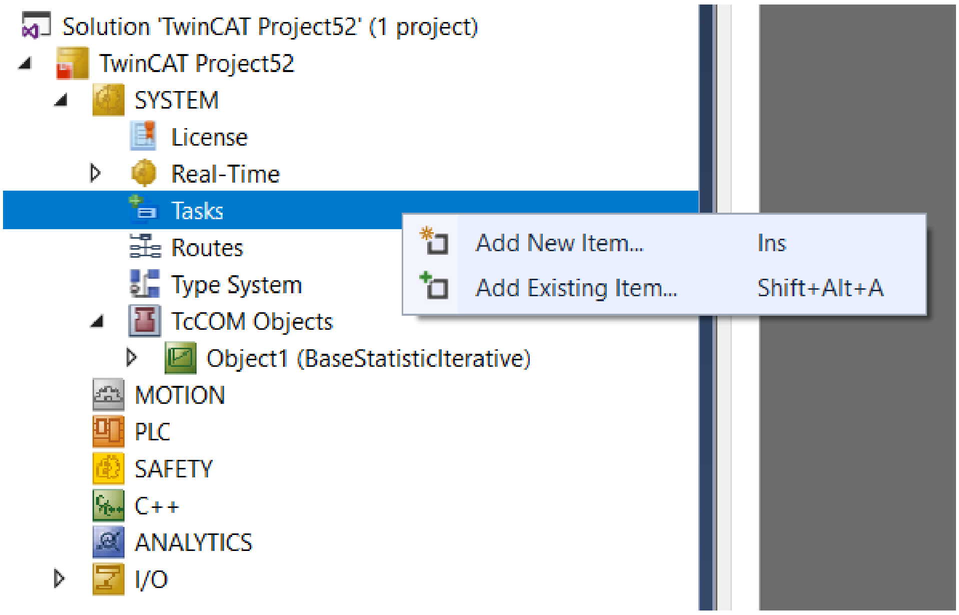
Search:

Name: Object1 (BaseStatisticIterative)

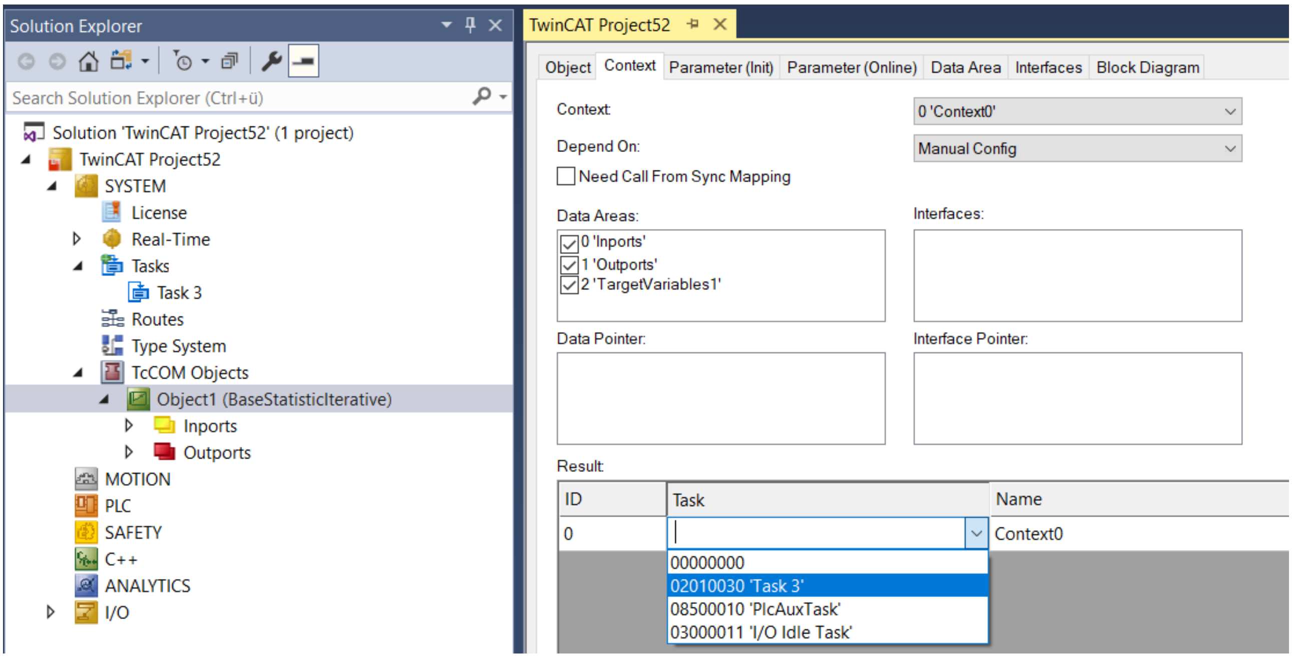
Multiple: 1

File: C:\TwinCAT\3.1\Repository\TE140x Module Vendor\Tc3_BaseStatistics\0.0.0.3\Tc3_BaseStatis

7. Create a new cyclic task of type *TwinCAT Task.*



8. 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.



Context: 0 'Context0'

Depend On: Manual Config

□ Need Call From Sync Mapping

Data Areas:
☑ 0 'Inports'
☑ 1 'Outports'
☑ 2 'TargetVariables1'

Interfaces:

Data Pointer:

Interface Pointer:

Result

ID	Task	Name
0		Context0

00000000
02010030 'Task 3'
08500010 'PlcAuxTask'
03000011 'I/O Idle Task'

⇨ 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 [▶ 43].

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?

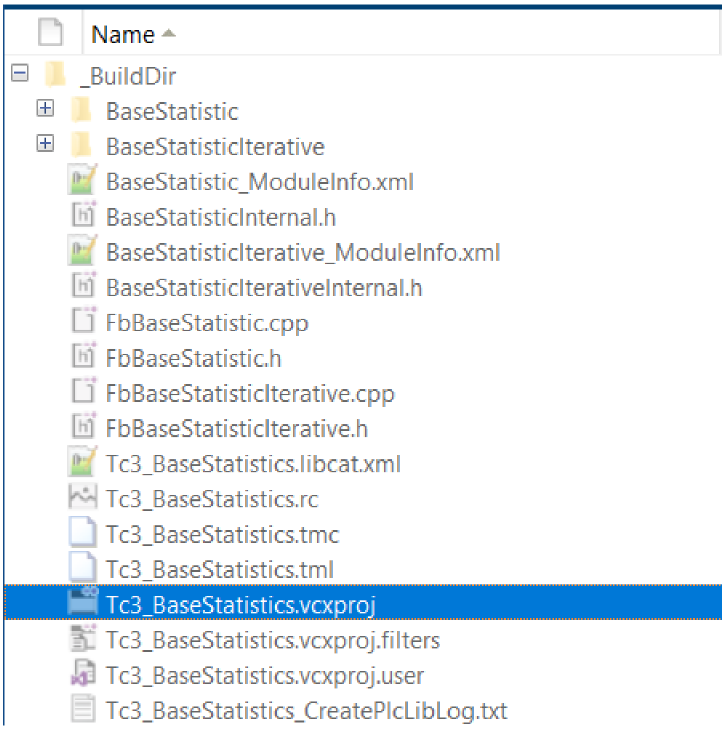
- Source code is generated.
- Log files are generated.
- The TwinCAT objects, drivers (*.tmx) and description files (*.tmc, *.library, ...), are created.

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*.



Name

_BuildDir
BaseStatistic
BaseStatisticIterative
BaseStatistic_ModuleInfo.xml
BaseStatisticInternal.h
BaseStatisticIterative_ModuleInfo.xml
BaseStatisticIterativeInternal.h
FbBaseStatistic.cpp
FbBaseStatistic.h
FbBaseStatisticIterative.cpp
FbBaseStatisticIterative.h
Tc3_BaseStatistics.libcat.xml
Tc3_BaseStatistics.rc
Tc3_BaseStatistics.tmc
Tc3_BaseStatistics.tml
Tc3_BaseStatistics.vcxproj
Tc3_BaseStatistics.vcxproj.filters
Tc3_BaseStatistics.vcxproj.user
Tc3_BaseStatistics_CreatePlcLibLog.txt

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

```
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:

```
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 Creating an export configuration [▶ 39]. The export configuration is appended under Properties in the cell array ClassExportCfg.

Sample call see Modul-Generator-Quickstart [▶ 23].

```
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:

```
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:

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

Loads the project export configuration from the current path.

```
Edit
```

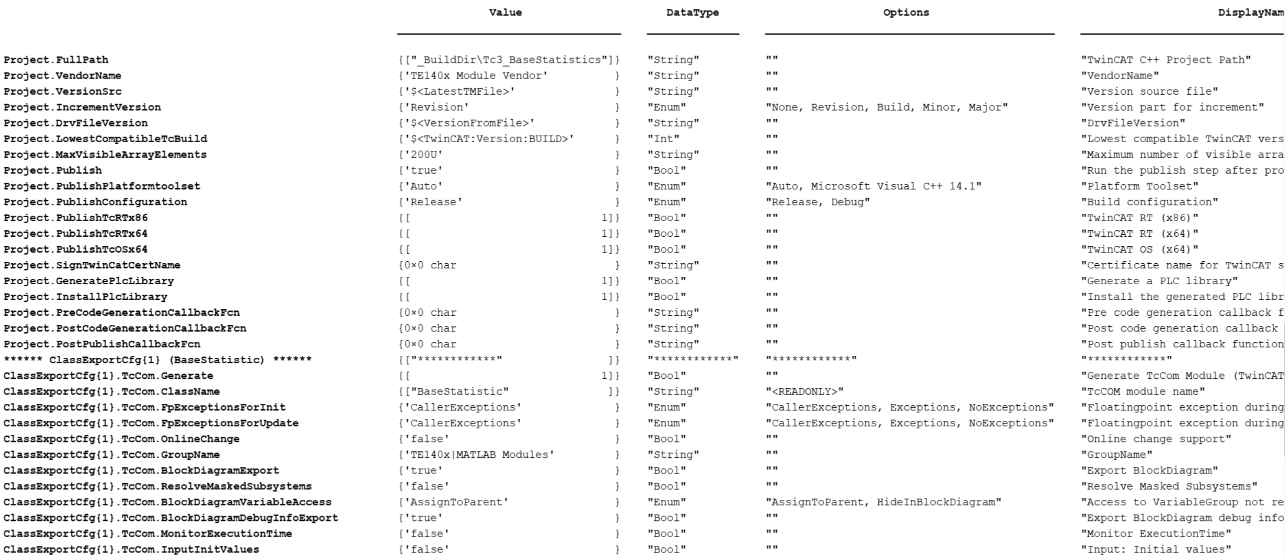
Opens a graphical configuration interface for configuring the project export configuration.

```
disp
```

Gives an overview in the MATLAB® Command Window of the current project export configuration. See Quick start [▶ 23].

Sample call:

`exportConfig.disp` alternatively `disp(exportConfig)`



```
                                                      Value                    DataType                     Options                                   DisplayNam
                                              ___________________________    ____________    ________________________________________________    ______________________________

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

Properties of the class TwinCAT.ModuleGenerator.ProjectExportConfig

`Project`

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

`Project.FullPath`

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

`Project.VendorName`

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.

`Project.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`).

`Project.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.

`Project.Publish`

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

`Project.PublishPlatformtoolset`

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

`Project.PublishTcRTx86`

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

`Project.PublishTcRTx64`

If TRUE, then XAR is built on a Windows 64-bit platform.

`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 [▶ 11].

`Project.TmxArchive`

Enter a path and file name here as a string in order to create a Creating TMX archives [▶ 39]. Example: Project.TmxArchive = "c:\archives\[Date]-[Time]-[LibName][LibVersion].exe" creates a self-extracting TMX archive under c:\archies. 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:

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

The PreCodeGenerationCallbackFcn property is then set to "MyCallback". By default, the Settings of the TwinCAT module generator [▶ 39] 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:

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

The PostCodeGenerationCallbackFcn property is then set to "MyCallback". By default, the Settings of the TwinCAT module generator [▶ 39] 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:

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

The PostPublishCallbackFcn property is then set to "MyCallback". By default, the Settings of the TwinCAT module generator [▶ 39] 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`

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:

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

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

`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`

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

More in Exception handling [▶ 55].

`TcCom.FpExceptionsForUpdate`

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

More in Exception handling [▶ 55].

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

`TcCom.ExecutionInfoOutput`

If TRUE, another output is created at the TcCOM with information in case of occurring exceptions. More in Exception handling [▶ 55].

`TcCom.OnlineChange`

If TRUE, then the TcCOM can be replaced by Online Change while TwinCAT XAR is in Run mode. See also Online Change for Target for Simulink®.

`TcCom.TcComWrapperFb`

If TRUE, a is created in the generated PLC library.

`TcCom.TcComWrapperFbProperties`

If TRUE, properties for module parameters are created at .

`TcCom.TcComWrapperFbPropertyMonitoring`

Options: NoMonitoring, CyclicUpdate, ExecutionUpdate

Specifies the monitoring attribute of the properties. In the default case, "No Monitoring" is set, i.e. no attribute is set.

Setting in MATLAB	Attribute on property
ExecutionUpdate	{attribute 'monitoring' := 'variable'}
CyclicUpdate	{attribute 'monitoring' := 'call'}

`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, ThrowExceptions, SuppressExceptions, LogExceptions, LogAndHold, LogAndCatch, LogAndDump, LogHoldAndDump, LogCatchAndDump

More in Exception handling [▶ 55].

`PlcFb. FpExceptionsForUpdate`

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

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

More in Exception handling [▶ 55].

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:

```
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.

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

Creating TwinCAT objects with the Module Generator

```
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.

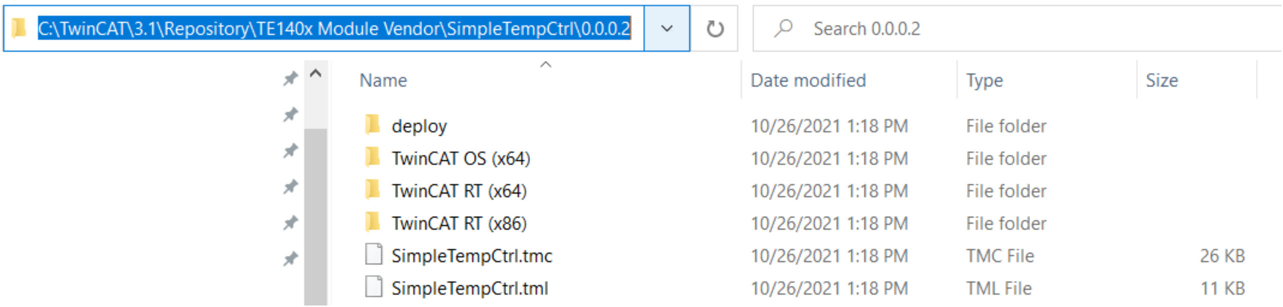
Sample call:

```
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:



C:\TwinCAT\3.1\Repository\TE140x Module Vendor\SimpleTempCtrl\0.0.0.2

Name	Date modified	Type	Size
deploy	10/26/2021 1:18 PM	File folder	
TwinCAT OS (x64)	10/26/2021 1:18 PM	File folder	
TwinCAT RT (x64)	10/26/2021 1:18 PM	File folder	
TwinCAT RT (x86)	10/26/2021 1:18 PM	File folder	
SimpleTempCtrl.tmc	10/26/2021 1:18 PM	TMC File	26 KB
SimpleTempCtrl.tml	10/26/2021 1:18 PM	TML File	11 KB

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.

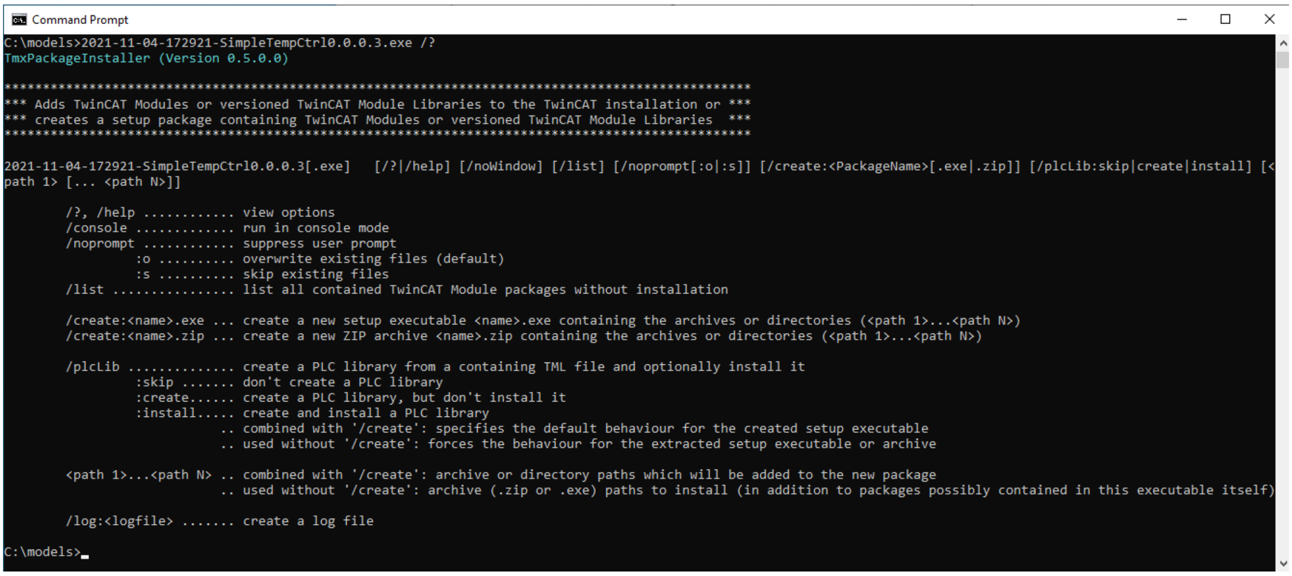
To do this, use the Project property of the module generator:

```
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:



```
C:\models>2021-11-04-172921-SimpleTempCtrl0.0.0.3.exe /?
TmxPackageInstaller (Version 0.5.0.0)

*************************************************************************************************
*** Adds TwinCAT Modules or versioned TwinCAT Module Libraries to the TwinCAT installation or ***
*** creates a setup package containing TwinCAT Modules or versioned TwinCAT Module Libraries  ***
*************************************************************************************************

2021-11-04-172921-SimpleTempCtrl0.0.0.3[.exe]   [/?|/help] [/noWindow] [/list] [/noprompt[:o|:s]] [/create:<PackageName>[.exe|.zip]] [/plcLib:skip|create|install] [<
path 1> [... <path N>]]

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

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

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

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

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

C:\models>_
```

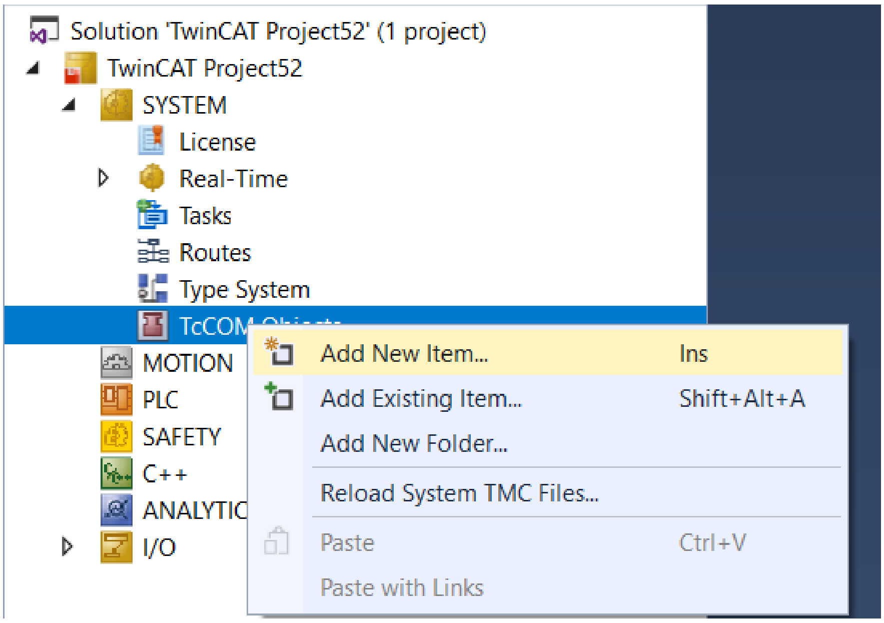
For example, the <tmxarchive>.exe /plclib:install command creates (from the *.tml file) and installs the PLC library on your local engineering PC.

8 Application of modules in TwinCAT

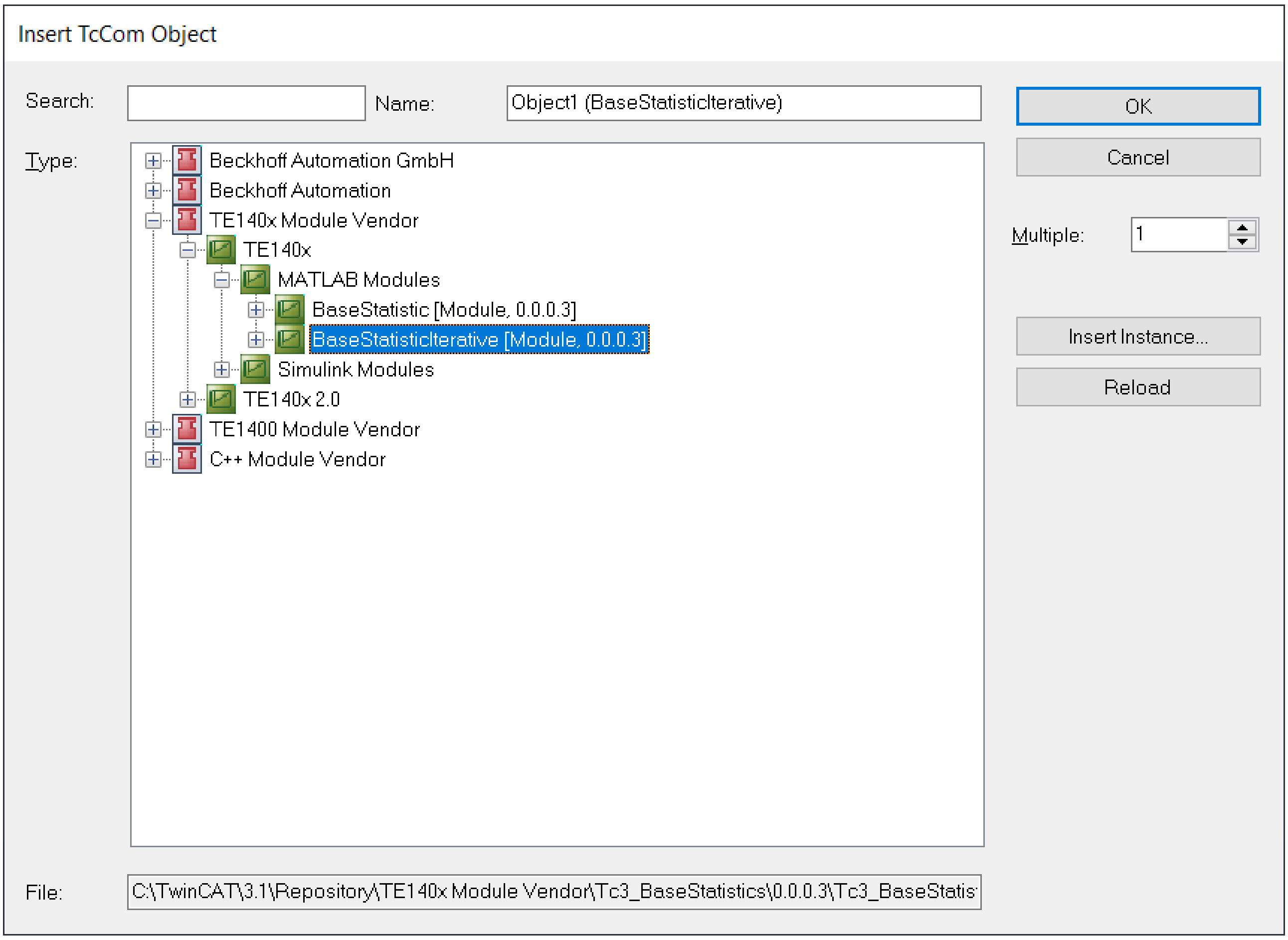
8.1 Working with the TcCOM module

Using TcCOM objects in TwinCAT 3

1. Insert a new TcCOM object.



2. Select the corresponding TcCOM object:


Insert TcCom Object

Search:

Name: Object1 (BaseStatisticIterative)

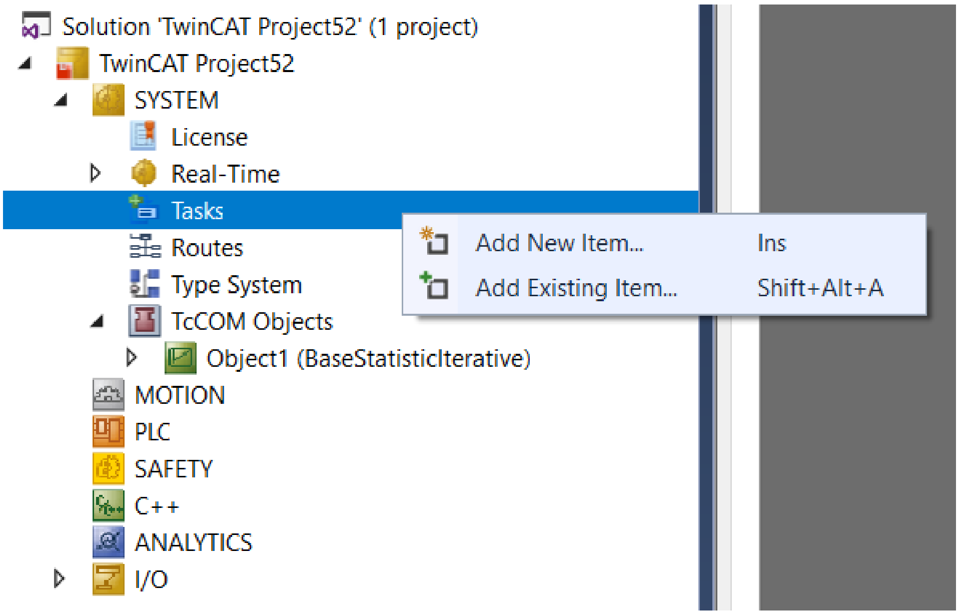
Type:

- Beckhoff Automation GmbH
- Beckhoff Automation
- TE140x Module Vendor
  - TE140x
    - MATLAB Modules
      - BaseStatistic [Module, 0.0.0.3]
      - BaseStatisticIterative [Module, 0.0.0.3]
    - Simulink Modules
  - TE140x 2.0
- TE1400 Module Vendor
- C++ Module Vendor

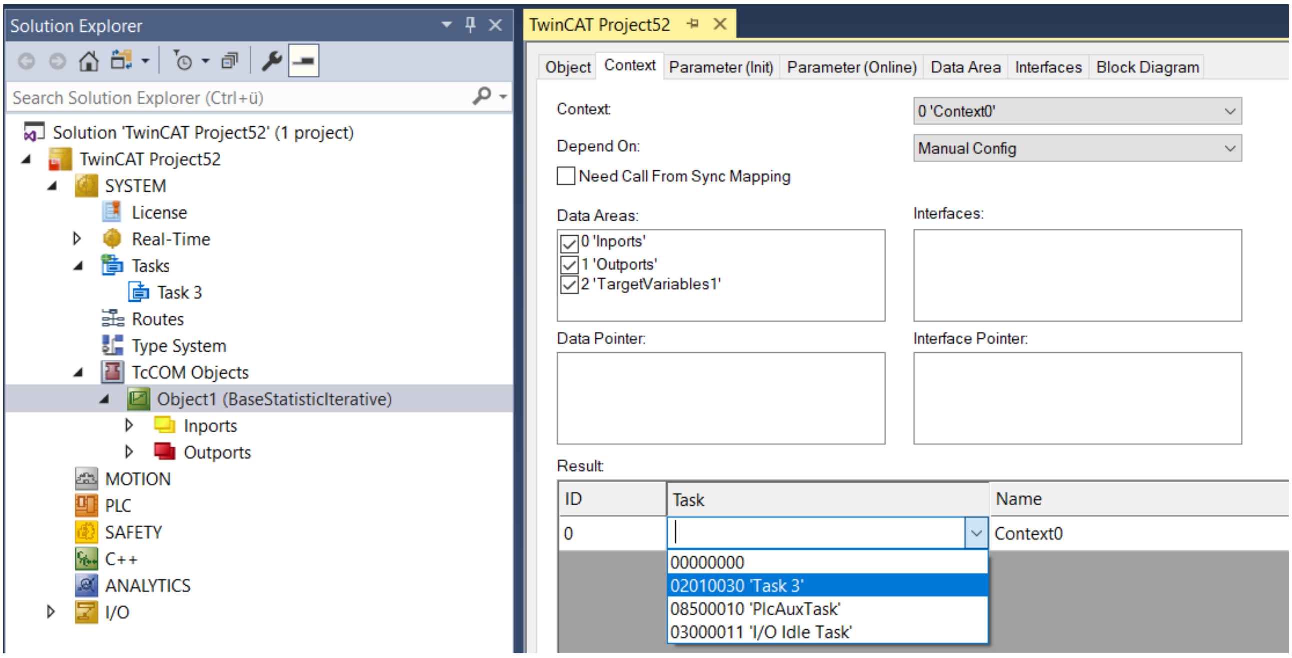
Multiple: 1

File: C:\TwinCAT\3.1\Repository\TE140x Module Vendor\Tc3_BaseStatistics\0.0.0.3\Tc3_BaseStatis

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.



Context: 0 'Context0'

Depend On: Manual Config

□ Need Call From Sync Mapping

Data Areas:

☑ 0 'Inports'
☑ 1 'Outports'
☑ 2 'TargetVariables1'

Interfaces:

Data Pointer:

Interface Pointer:

Result

ID	Task	Name
0		Context0

00000000
02010030 'Task 3'
08500010 'PlcAuxTask'
03000011 'I/O Idle Task'

⇨ 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 [▶ 43].

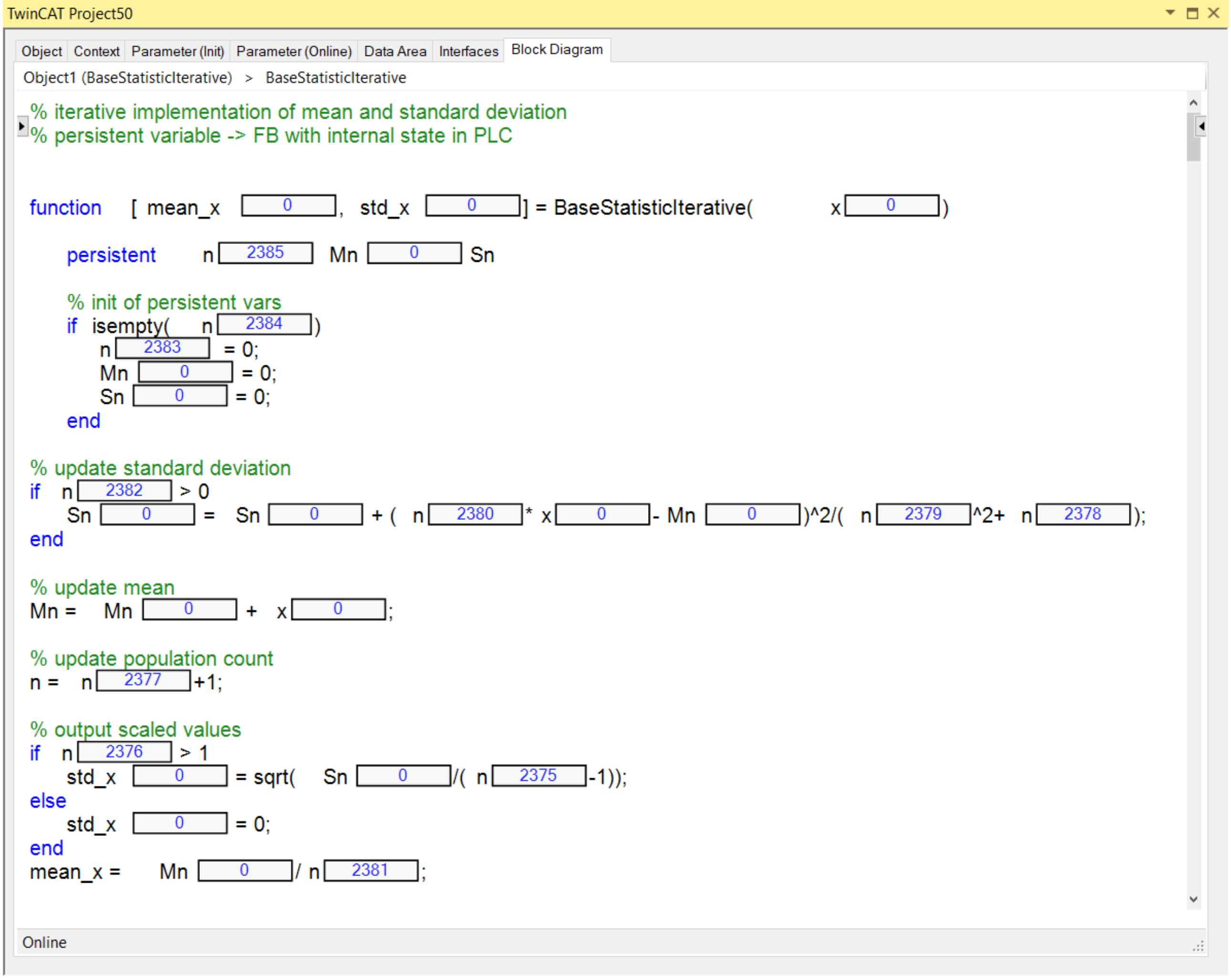
You can also call the TcCOM object from the PLC via the TcCOM wrapper FB or even instantiate it dynamically. For more, see: Calling a TcCOM object from the PLC [▶ 51].

Besides working with a TcCOM object, you can also use the functions created in MATLAB® directly as a PLC function block (PLC-FB) without having to worry about a TcCOM object. See: Working with the PLC library [▶ 46].

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.

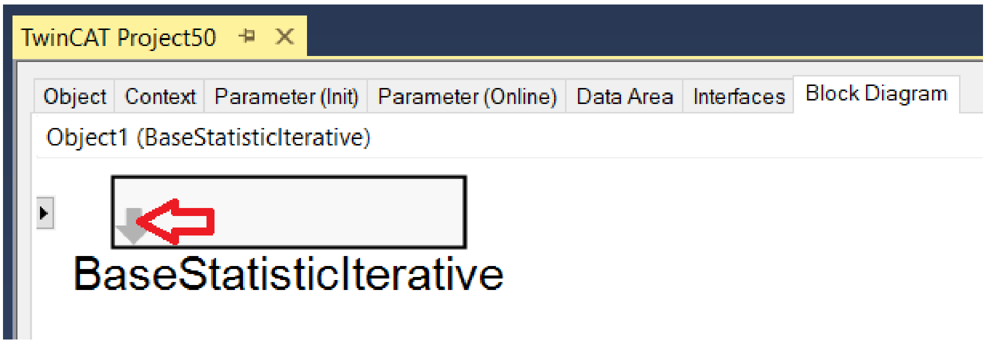


```
% iterative implementation of mean and standard deviation
% persistent variable -> FB with internal state in PLC


function  [ mean_x  0 ,  std_x  0 ] = BaseStatisticIterative(        x 0 )

    persistent     n 2385  Mn  0  Sn

    % init of persistent vars
    if  isempty(   n 2384 )
        n 2383  = 0;
        Mn  0  = 0;
        Sn  0  = 0;
    end

% update standard deviation
if  n 2382  > 0
    Sn  0  =  Sn  0  + (  n 2380 * x 0 - Mn  0 )^2/(  n 2379 ^2+  n 2378 );
end

% update mean
Mn =   Mn  0  +  x 0 ;

% update population count
n =  n 2377 +1;

% output scaled values
if  n 2376  > 1
    std_x  0  = sqrt(   Sn  0 /(  n 2375 -1));
else
    std_x  0  = 0;
end
mean_x =    Mn  0 / n 2381 ;
```

8.1.1.1.1 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.

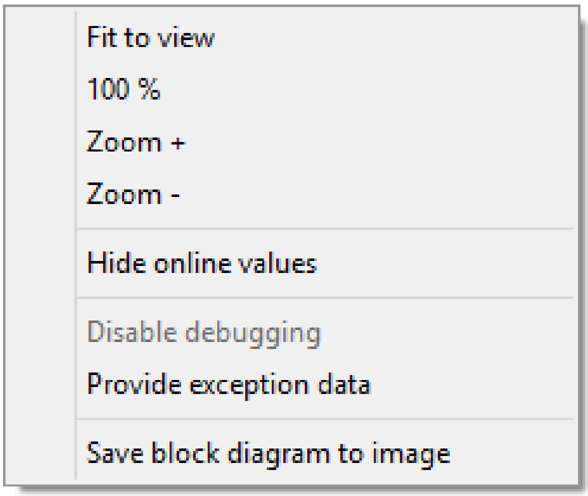
On the top level you will find the created module in block representation. Select the gray arrow in the block to display the content.



Shortcut functions:

Shortcut	Function
Space	Zoom to current size of the block diagram tab
Backspace	Switch to the next higher hierarchical level
ESC	Switch to the next higher hierarchical level
CTRL + "+"	Zoom in
CTRL + "-"	Zoom out
F5	Attach Debugger (*System- > Real-Time -> C++ Debugger -> Enable C++ Debugger* must be activated)

Context menu functions:

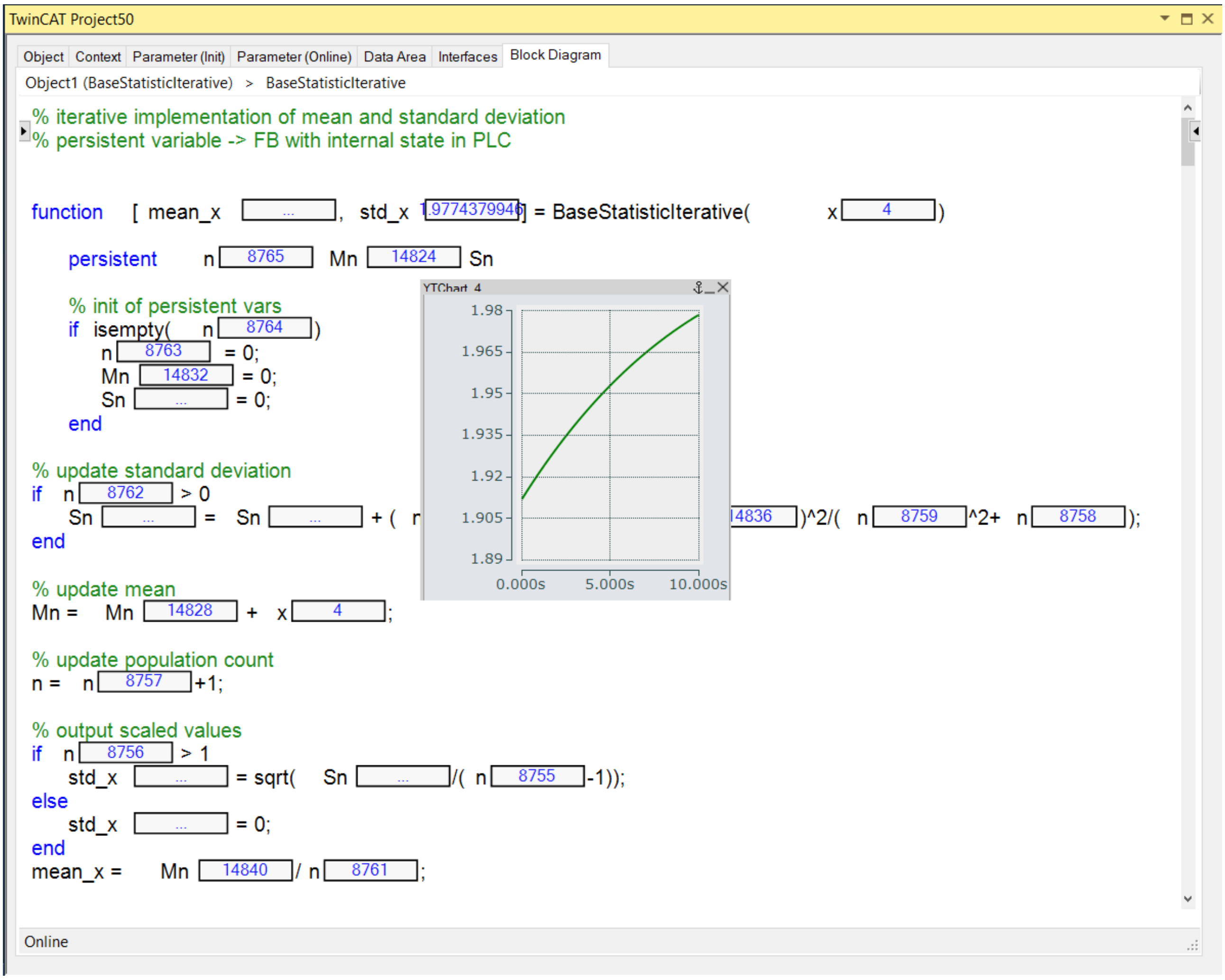


8.1.1.1.2 Display of signal curves

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.

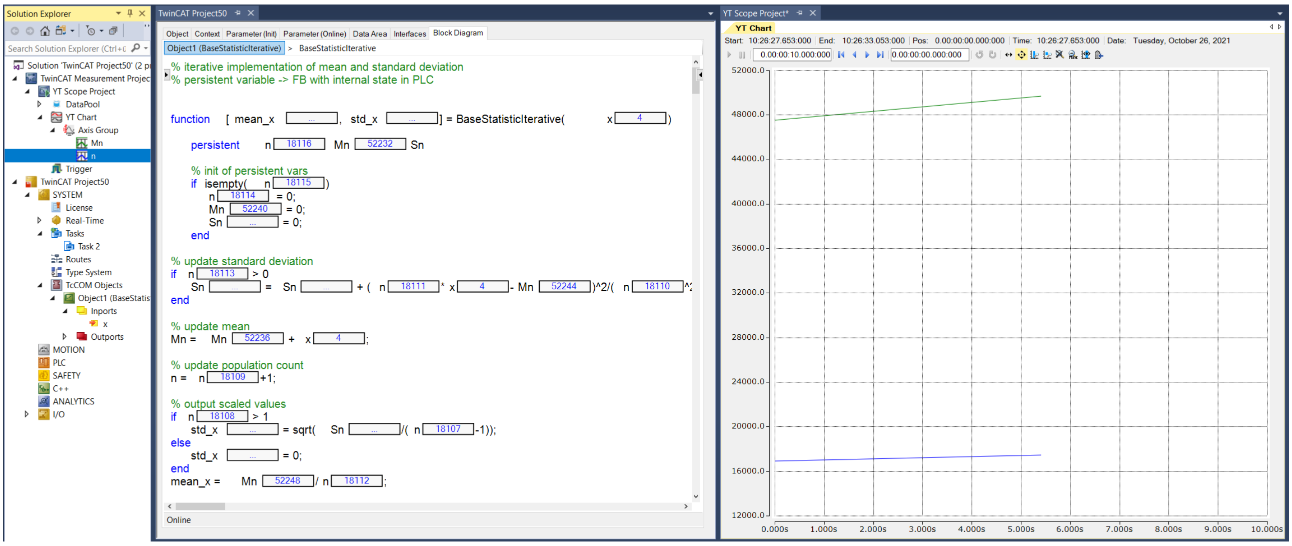
Variables that can be displayed in scope have a trailing black frame in the code display. In this frame, the values are displayed in blue during operation.

Drag&Drop a "blue variable" onto the block diagram window to open a Mini.Scope.



```
% iterative implementation of mean and standard deviation
% persistent variable -> FB with internal state in PLC


function   [ mean_x  ...  ,  std_x  1.9774379946] = BaseStatisticIterative(        x  4 )

    persistent    n  8765   Mn  14824   Sn

    % init of persistent vars
    if  isempty(   n  8764 )
        n  8763   = 0;
        Mn  14832   = 0;
        Sn  ...   = 0;
    end

% update standard deviation
if   n  8762  > 0
    Sn  ...   =   Sn  ...   + (  r  14836 )^2/(  n  8759 ^2+  n  8758 );
end

% update mean
Mn =    Mn  14828   +   x  4 ;

% update population count
n =    n  8757 +1;

% output scaled values
if   n  8756   > 1
    std_x  ...   = sqrt(   Sn  ... /(  n  8755 -1));
else
    std_x  ...   = 0;
end
mean_x =      Mn  14840  /  n  8761 ;
```

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.



```
% iterative implementation of mean and standard deviation
% persistent variable -> FB with internal state in PLC


function   [ mean_x  ...  ,  std_x  ...  ] = BaseStatisticIterative(        x  4 )

    persistent    n  18116   Mn  52232   Sn

    % init of persistent vars
    if  isempty(   n  18115 )
        n  18114   = 0;
        Mn  52240   = 0;
        Sn  ...   = 0;
    end

% update standard deviation
if   n  18113  > 0
    Sn  ...   =   Sn  ...   + (  n  18111 *  x  4 - Mn  52244 )^2/(  n  18110 ^2
end

% update mean
Mn =    Mn  52236   +   x  4 ;

% update population count
n =    n  18109 +1;

% output scaled values
if   n  18108   > 1
    std_x  ...   = sqrt(   Sn  ... /(  n  18107 -1));
else
    std_x  ...   = 0;
end
mean_x =      Mn  52248  /  n  18112 ;
```

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

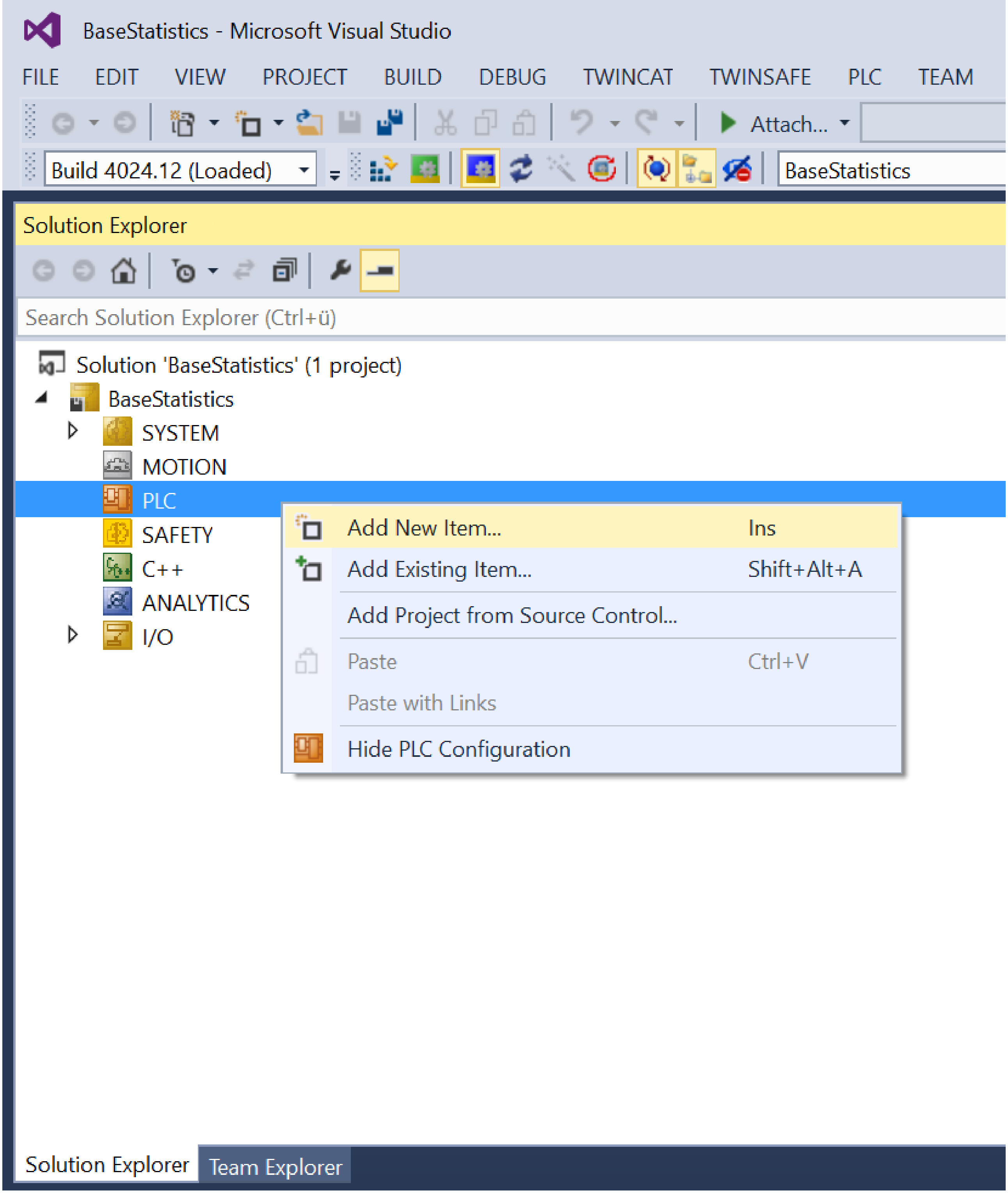
- The input variables
- The output variables
- Persistent variables (MATLAB definition `persistent var1 … varN`)

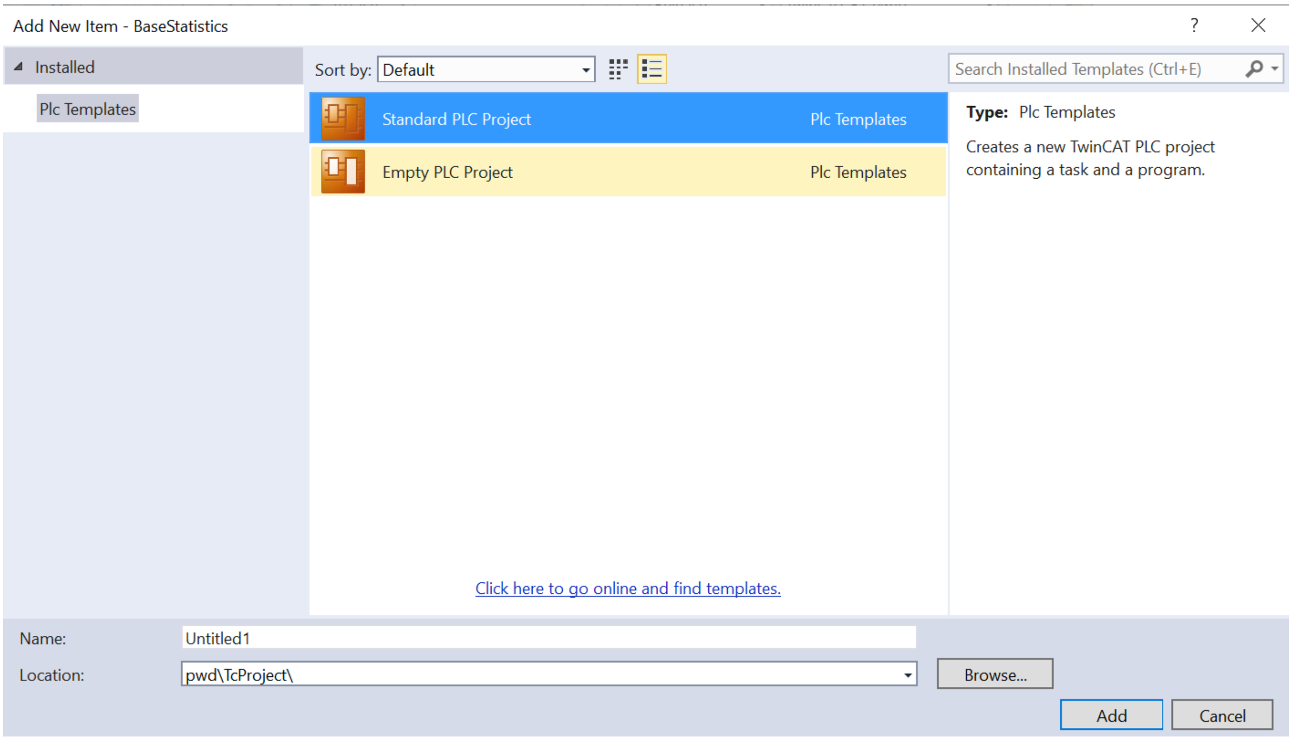
8.2 Working with the PLC library

Use PLC library in TwinCAT 3

✓ Starting from a new TwinCAT solution, create a PLC project:

1. Perform the following menu steps.


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



Add New Item - BaseStatistics

?
Installed
Plc Templates
Sort by: Default
Search Installed Templates (Ctrl+E)
Standard PLC Project Plc Templates
Empty PLC Project Plc Templates
Click here to go online and find templates.
Browse...
Add
Cancel

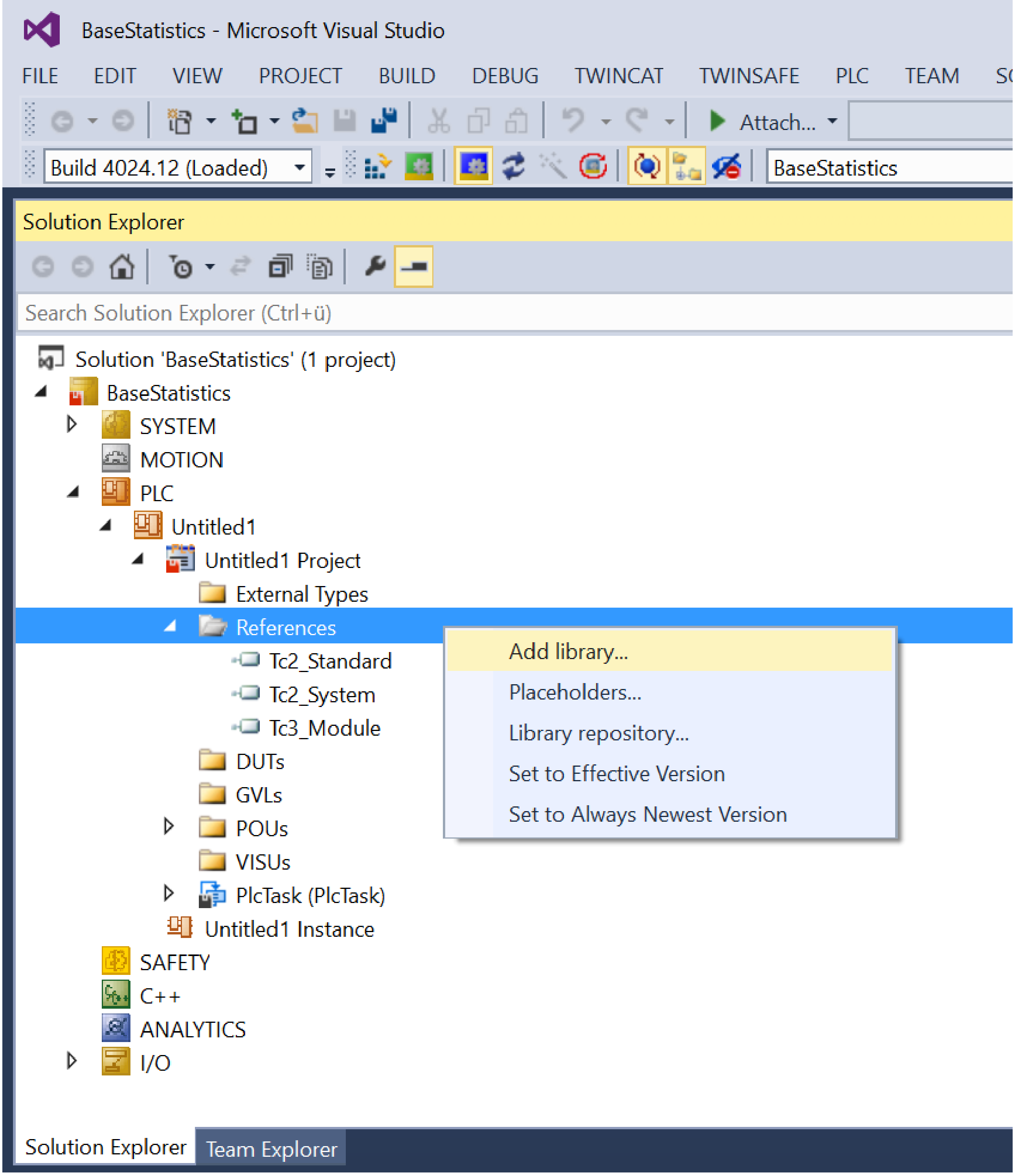
Type: Plc Templates

Creates a new TwinCAT PLC project containing a task and a program.

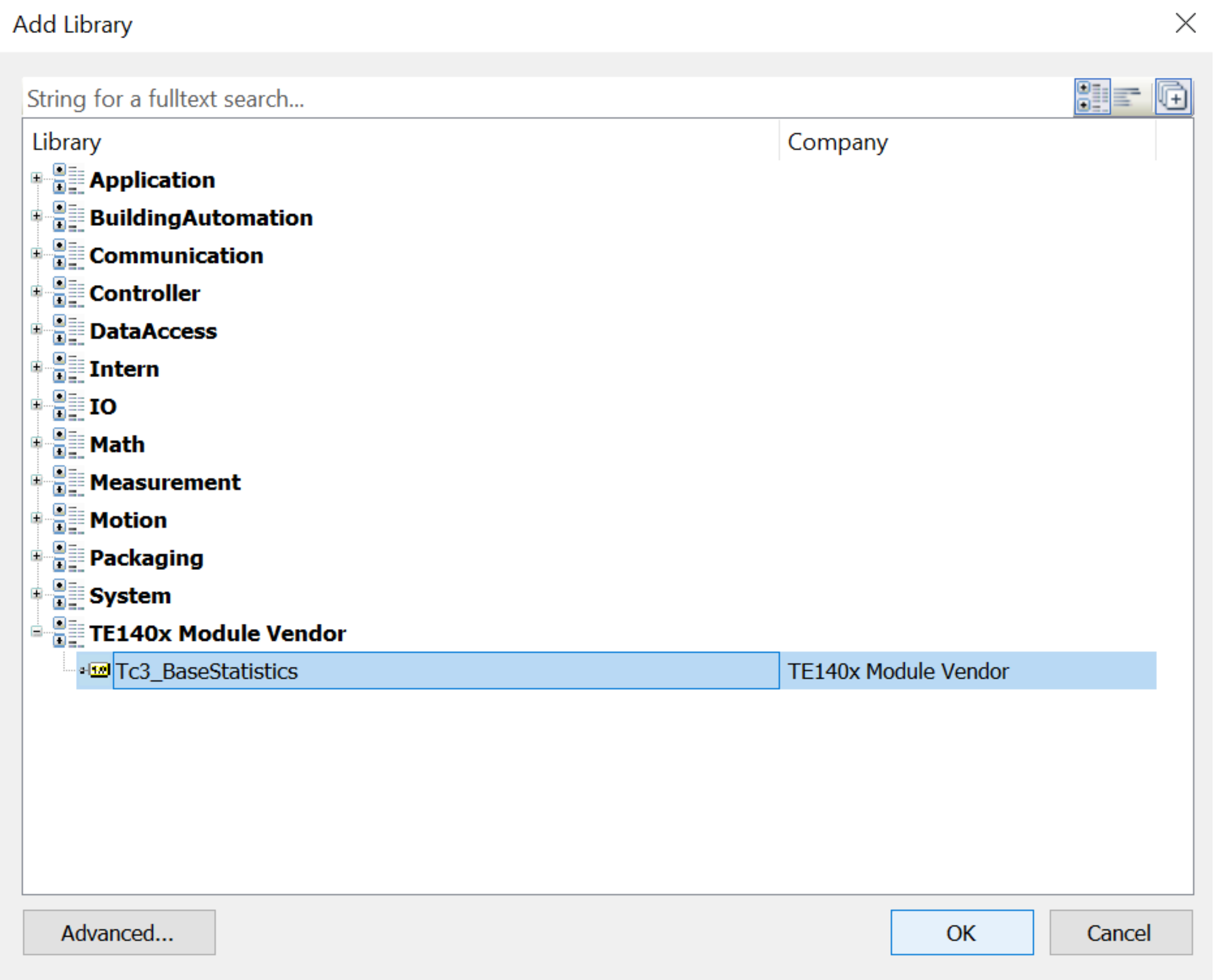
Name: Untitled1

Location: pwd\TcProject\

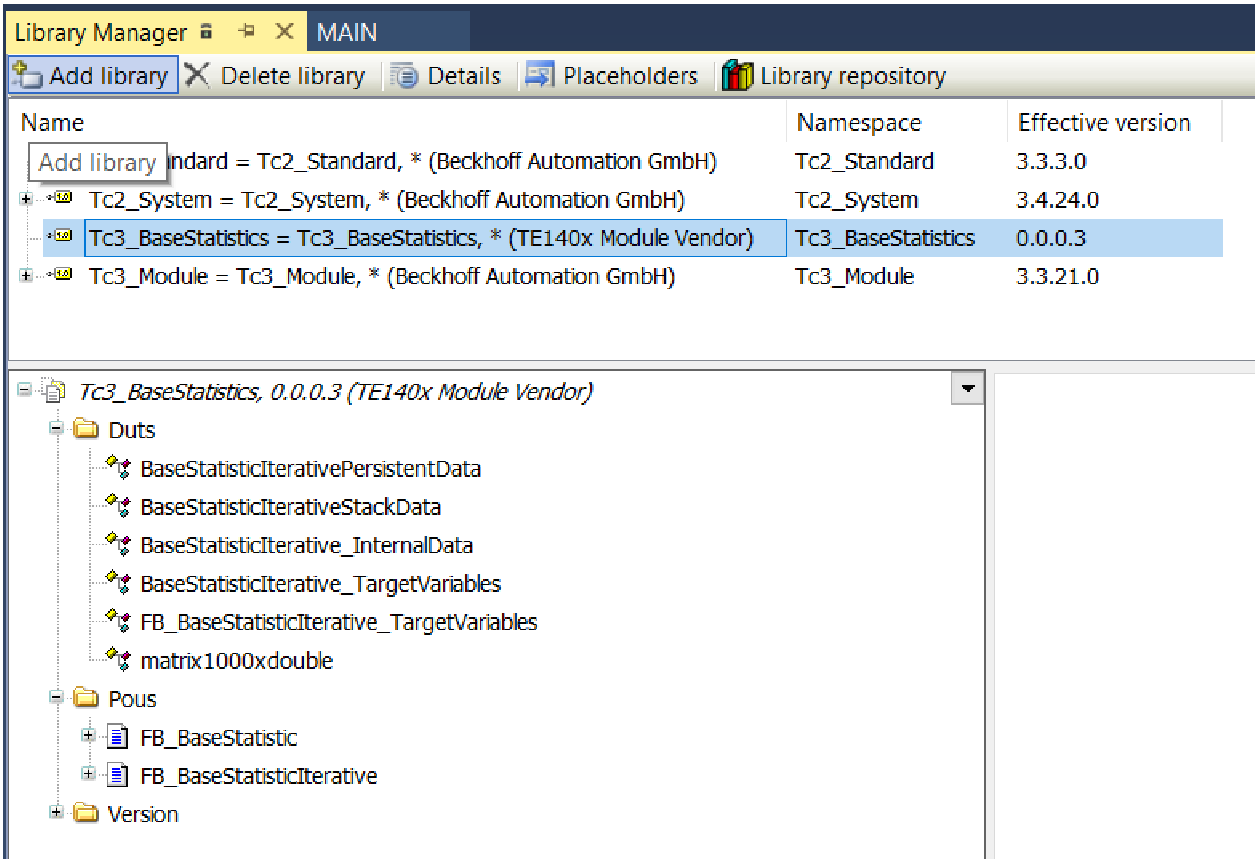
2. Then add the newly created (and already installed) PLC library:



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



3. Get an overview of the data types and function blocks:

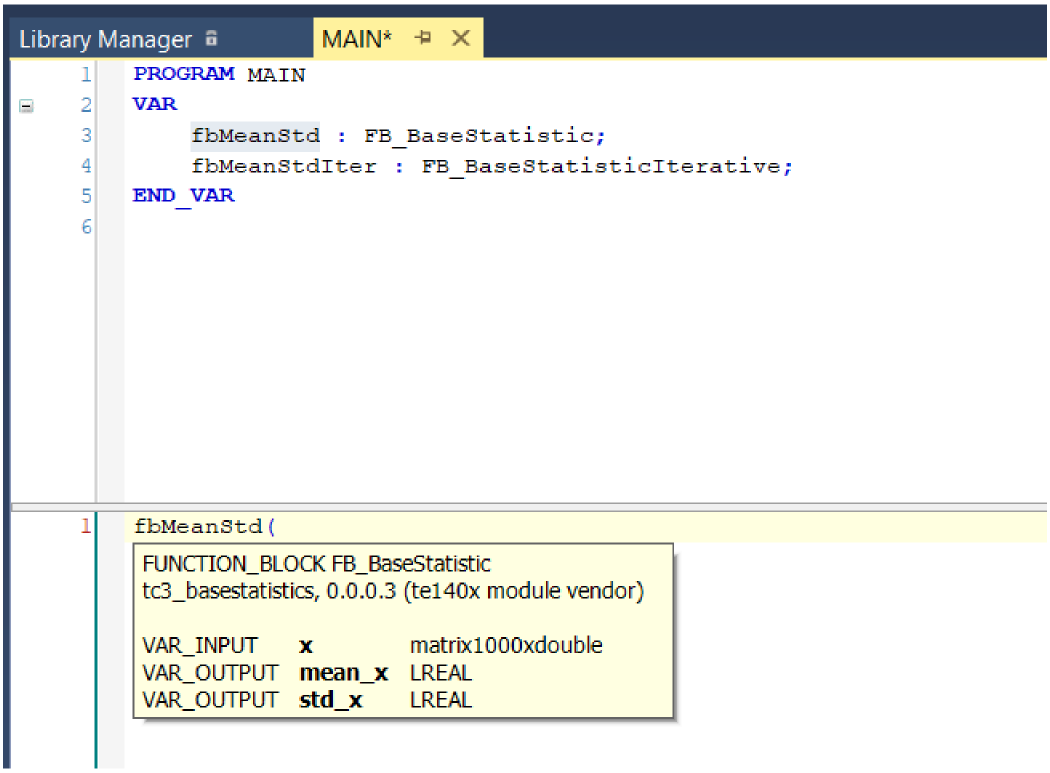


Name	Namespace	Effective version
ndard = Tc2_Standard, * (Beckhoff Automation GmbH)	Tc2_Standard	3.3.3.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH)	Tc2_System	3.4.24.0
Tc3_BaseStatistics = Tc3_BaseStatistics, * (TE140x Module Vendor)	Tc3_BaseStatistics	0.0.0.3
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH)	Tc3_Module	3.3.21.0

Tc3_BaseStatistics, 0.0.0.3 (TE140x Module Vendor)

- Duts
  - BaseStatisticIterativePersistentData
  - BaseStatisticIterativeStackData
  - BaseStatisticIterative_InternalData
  - BaseStatisticIterative_TargetVariables
  - FB_BaseStatisticIterative_TargetVariables
  - matrix1000xdouble
- Pous
  - FB_BaseStatistic
  - FB_BaseStatisticIterative
- Version

4. Insert instances of the function blocks into your PLC and use them in your application:



```
PROGRAM MAIN
VAR
    fbMeanStd : FB_BaseStatistic;
    fbMeanStdIter : FB_BaseStatisticIterative;
END_VAR

```

```
fbMeanStd(
```

FUNCTION_BLOCK FB_BaseStatistic
tc3_basestatistics, 0.0.0.3 (te140x module vendor)

VAR_INPUT	**x**	matrix1000xdouble
VAR_OUTPUT	**mean_x**	LREAL
VAR_OUTPUT	**std_x**	LREAL

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).

8.2.2 Calling a TcCOM object from the PLC

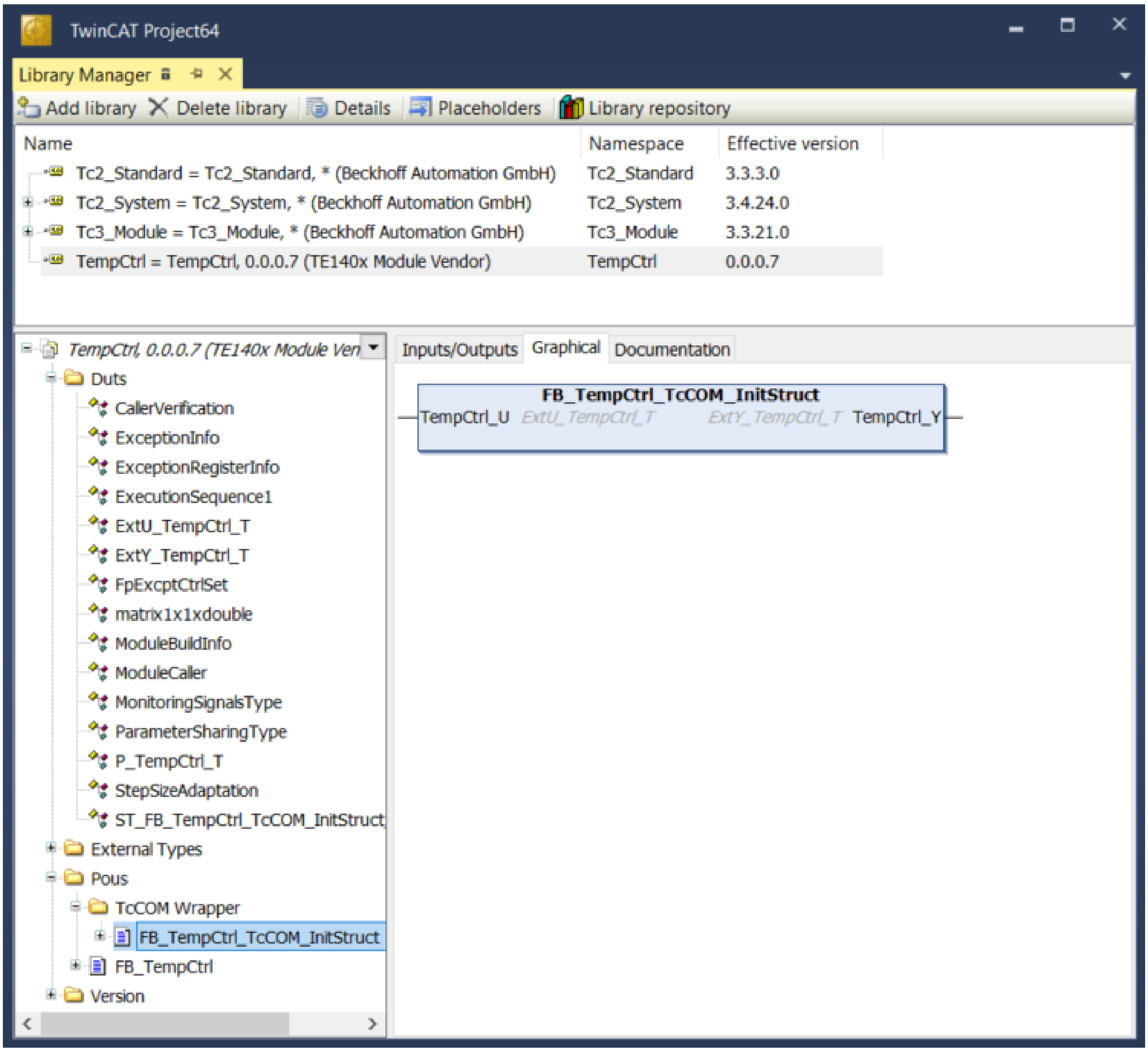
Creating a TcCOM wrapper FB

Set in the export configuration:

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

Working with the TcCOM Wrapper FB

1. Create a PLC project.
2. Add the desired library under **References**.



Name	Namespace	Effective version
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH)	Tc2_Standard	3.3.3.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH)	Tc2_System	3.4.24.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH)	Tc3_Module	3.3.21.0
TempCtrl = TempCtrl, 0.0.0.7 (TE140x Module Vendor)	TempCtrl	0.0.0.7

⇨ Under **Pous/TcCOM Wrapper** you get a function block that you can instantiate in the PLC. In addition, necessary data types are created in the Duts folder.

Referencing a static module instance

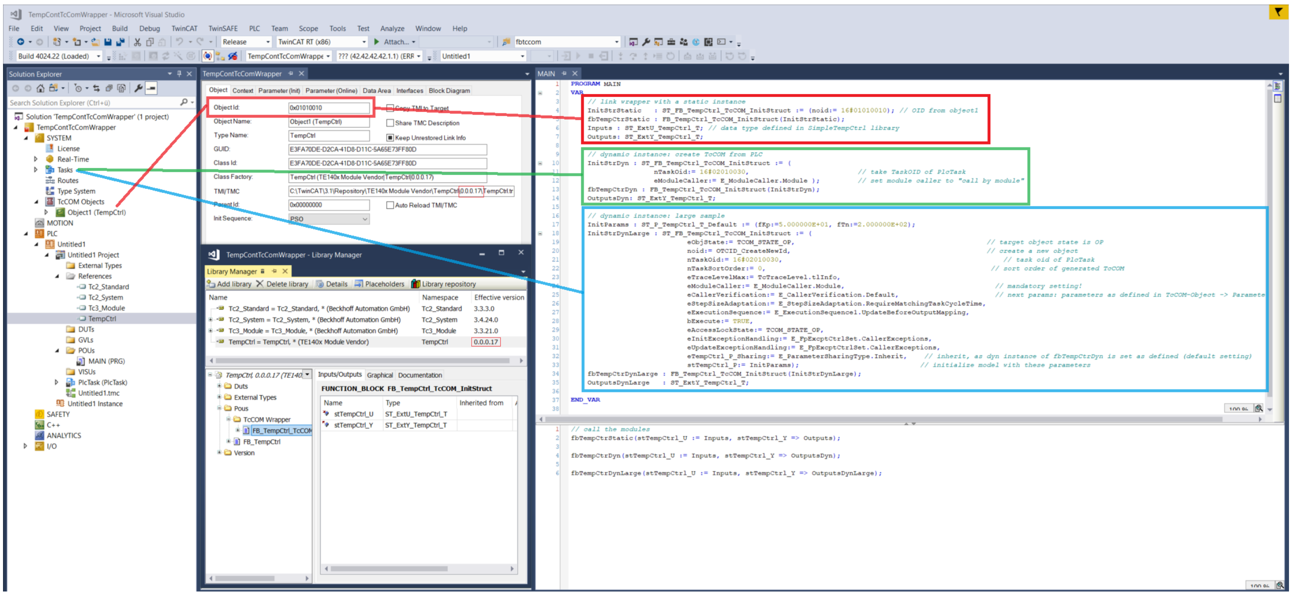
The function block can be used to access module instances previously created in the XAE, e.g. under **System > TcCOM Objects**. For this static case, the object ID of the corresponding module instance must be transferred during declaration of the function block instance. See red area in graphic below.

- The instance of the TcCOM object and the calling PLC must run in the same task.
- On the instance of the TcCOM object, make sure that under Parameter (Init) the entry *ModuleCaller* is set to *Module* and not to *CyclicTask.*
- In this case, the required memory for the TcCOM is obtained from the *non paged pool* of the system.

Dynamic instantiation and referencing from the PLC

The function block can also be used in such a way that a TcCOM object is generated from the PLC and linked to the wrapper. In the graphic below, this is the green area as a minimum example and the blue area with extended parameterization.

- The TaskOid of the PLC task must be used to specify the real-time task in which the wrapper is called.
- The ModuleCaller must also be set to *Module* here (via the Init structure).
- In this case, the required memory for the TcCOM is obtained from the router memory.



Object Id: 0x01010010 □ Copy TMI to Target
Object Name: Object1 (TempCtrl) □ Share TMC Description
Type Name: TempCtrl ☒ Keep Unrestored Link Info
GUID: E3FA70DE-D2CA-41D8-D11C-5A65E73FF80D
Class Id: E3FA70DE-D2CA-41D8-D11C-5A65E73FF80D
Class Factory: TempCtrl (TE140x Module Vendor|TempCtrl|0.0.0.17)
TMI/TMC: C:\TwinCAT\3.1\Repository\TE140x Module Vendor\TempCtrl\0.0.0.17\TempCtrl.tr
Parent Id: 0x00000000 □ Auto Reload TMI/TMC
Init Sequence: PSO

Name	Namespace	Effective version
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH)	Tc2_Standard	3.3.3.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH)	Tc2_System	3.4.24.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH)	Tc3_Module	3.3.21.0
TempCtrl = TempCtrl, * (TE140x Module Vendor)	TempCtrl	0.0.0.17

```
PROGRAM MAIN
VAR
    // link wrapper with a static instance
    InitStrStatic    : ST_FB_TempCtrl_TcCOM_InitStruct := (noid:= 16#01010010); // OID from object1
    fbTempCtrStatic : FB_TempCtrl_TcCOM_InitStruct(InitStrStatic);
    Inputs : ST_ExtU_TempCtrl_T; // data type defined in SimpleTempCtrl library
    Outputs: ST_ExtY_TempCtrl_T;

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

    // dynamic instance: large sample
    InitParams : ST_P_TempCtrl_T_Default := (fKp:=5.000000E+01, fTn:=2.000000E+02);
    InitStrDynLarge : ST_FB_TempCtrl_TcCOM_InitStruct := (
                        eObjState:= TCOM_STATE_OP,                                      // target object state is OP
                        noid:= OTCID_CreateNewId,                                       // create a new object
                        nTaskOid:= 16#02010030,                                            // task oid of PlcTask
                        nTaskSortOrder:= 0,                                              // sort order of generated TcCOM
                        eTraceLevelMax:= TcTraceLevel.tlInfo,
                        eModuleCaller:= E_ModuleCaller.Module,                            // mandatory setting!
                        eCallerVerification:= E_CallerVerification.Default,               // next params: parameters as defined in TcCOM-Object -> Paramete
                        eStepSizeAdaptation:= E_StepSizeAdaptation.RequireMatchingTaskCycleTime,
                        eExecutionSequence:= E_ExecutionSequence1.UpdateBeforeOutputMapping,
                        bExecute:= TRUE,
                        eAccessLockState:= TCOM_STATE_OP,
                        eInitExceptionHandling:= E_FpExcptCtrlSet.CallerExceptions,
                        eUpdateExceptionHandling:= E_FpExcptCtrlSet.CallerExceptions,
                        eTempCtrl_P_Sharing:= E_ParameterSharingType.Inherit,    // inherit, as dyn instance of fbTempCtrDyn is set as defined (default setting)
                        stTempCtrl_P:= InitParams);                             // initialize model with these parameters
    fbTempCtrDynLarge : FB_TempCtrl_TcCOM_InitStruct(InitStrDynLarge);
    OutputsDynLarge   : ST_ExtY_TempCtrl_T;

END_VAR
```

```
// call the modules
fbTempCtrStatic(stTempCtrl_U := Inputs, stTempCtrl_Y => Outputs);

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

fbTempCtrDynLarge(stTempCtrl_U := Inputs, stTempCtrl_Y => OutputsDynLarge);
```

Fig. 2:

The source code for the graph shown above is available in MATLAB® via the Command Window:

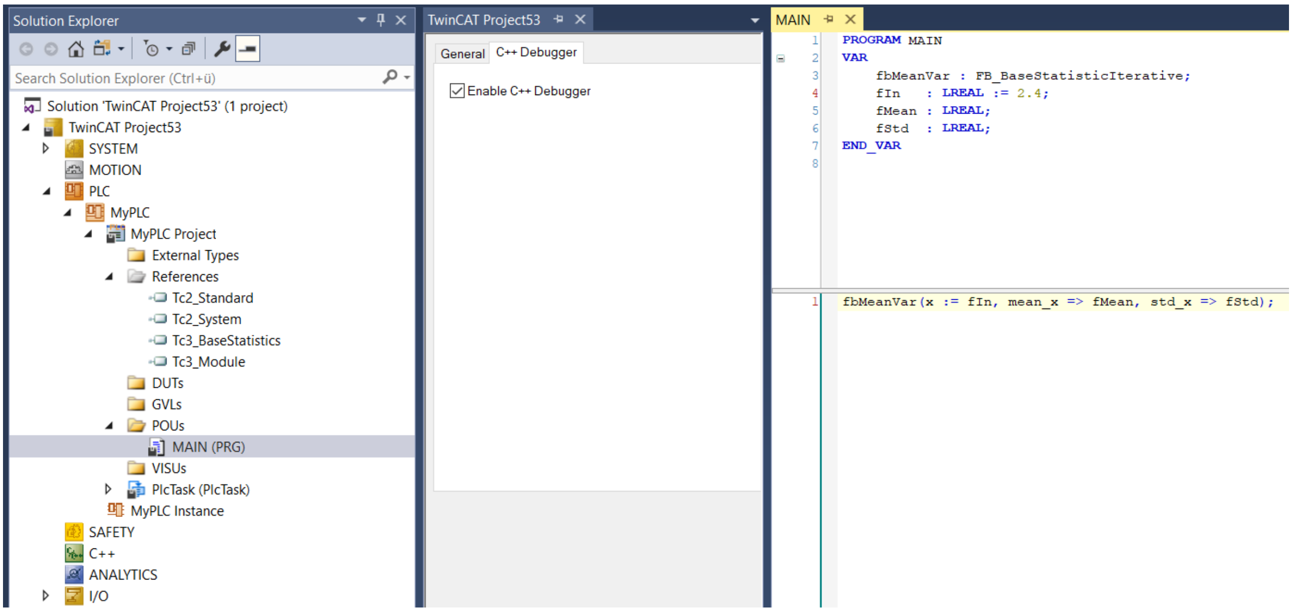
```
TwinCAT.ModuleGenerator.Samples.Start('TcComWrapper TemperatureController')
```

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.



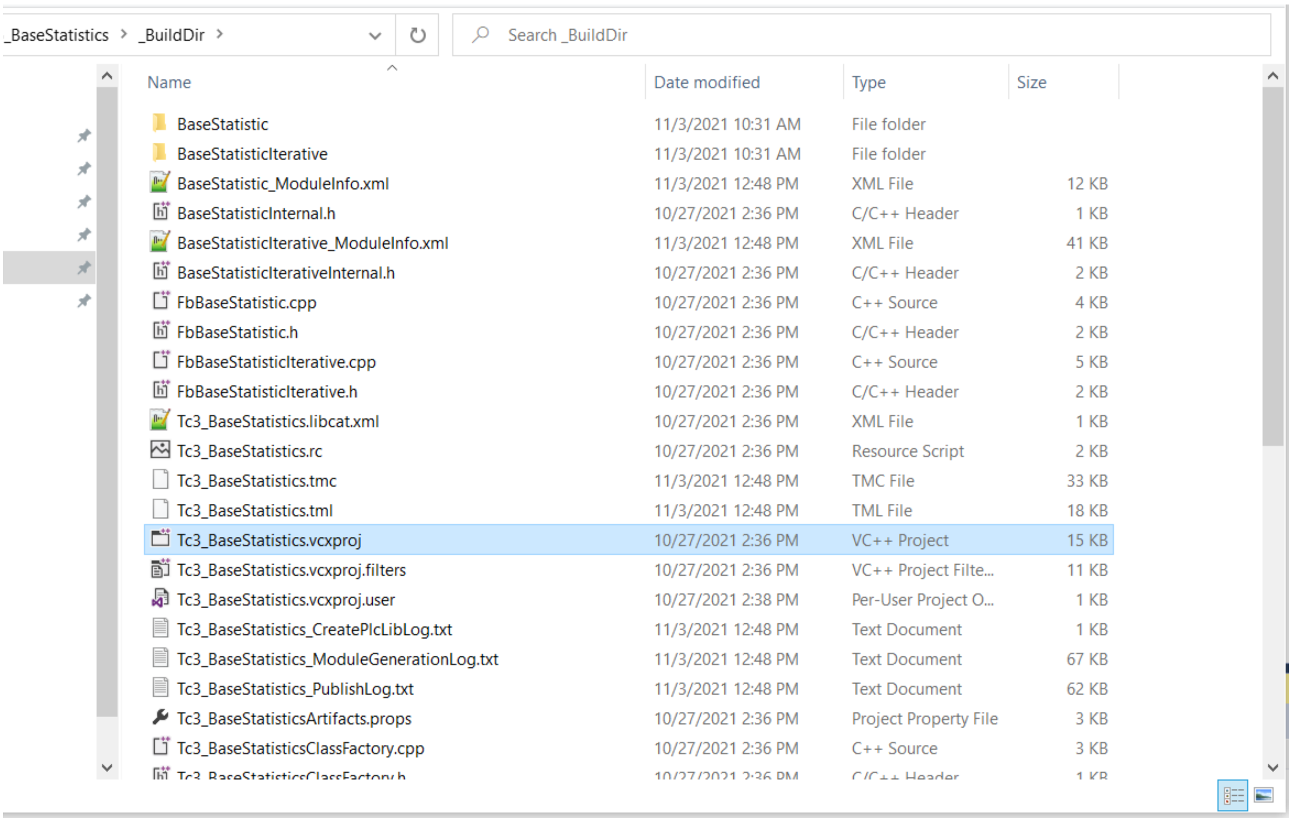
☑ Enable C++ Debugger

```
PROGRAM MAIN
VAR
    fbMeanVar : FB_BaseStatisticIterative;
    fIn   : LREAL := 2.4;
    fMean : LREAL;
    fStd  : LREAL;
END_VAR

fbMeanVar(x := fIn, mean_x => fMean, std_x => fStd);
```

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 [▶ 33].

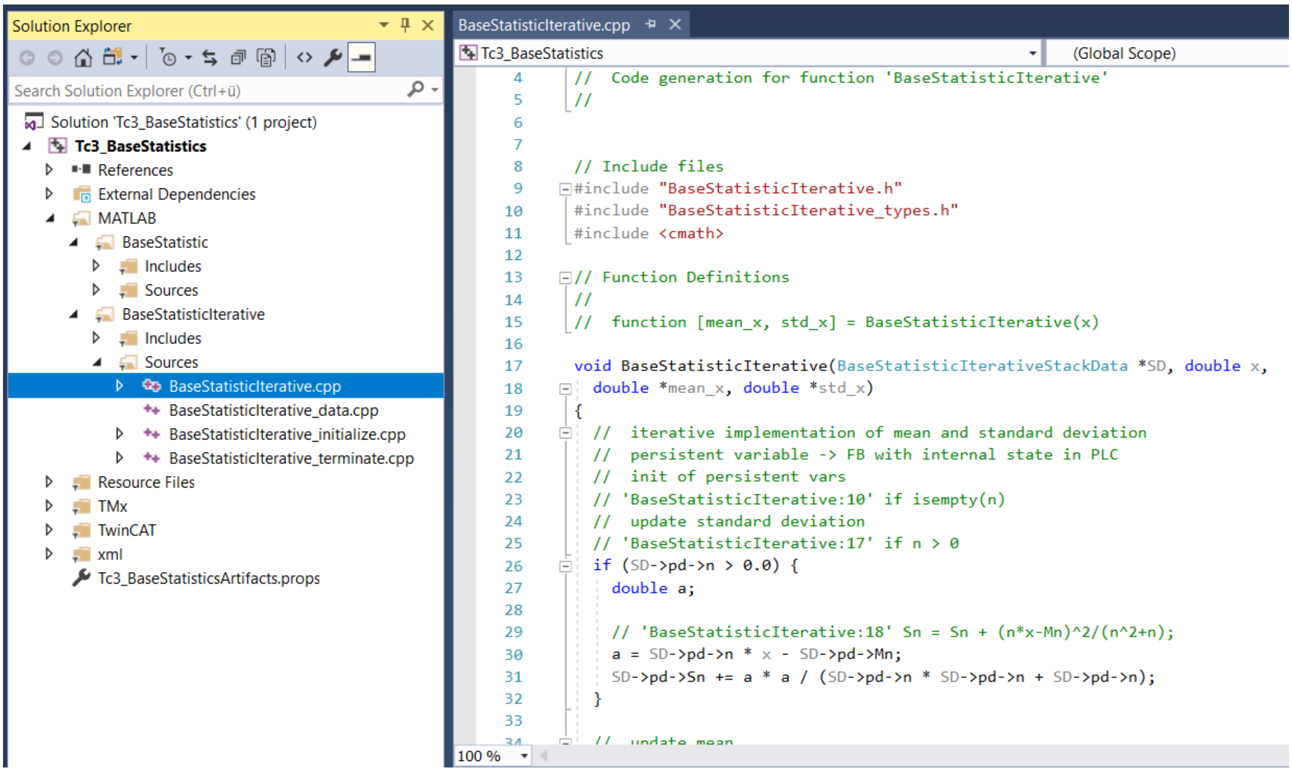
   ⇨ You can open the Visual Studio project directly, or add it to your TwinCAT solution under C++ with "Add existing Item".



Name	Date modified	Type	Size
BaseStatistic	11/3/2021 10:31 AM	File folder	
BaseStatisticIterative	11/3/2021 10:31 AM	File folder	
BaseStatistic_ModuleInfo.xml	11/3/2021 12:48 PM	XML File	12 KB
BaseStatisticInternal.h	10/27/2021 2:36 PM	C/C++ Header	1 KB
BaseStatisticIterative_ModuleInfo.xml	11/3/2021 12:48 PM	XML File	41 KB
BaseStatisticIterativeInternal.h	10/27/2021 2:36 PM	C/C++ Header	2 KB
FbBaseStatistic.cpp	10/27/2021 2:36 PM	C++ Source	4 KB
FbBaseStatistic.h	10/27/2021 2:36 PM	C/C++ Header	2 KB
FbBaseStatisticIterative.cpp	10/27/2021 2:36 PM	C++ Source	5 KB
FbBaseStatisticIterative.h	10/27/2021 2:36 PM	C/C++ Header	2 KB
Tc3_BaseStatistics.libcat.xml	10/27/2021 2:36 PM	XML File	1 KB
Tc3_BaseStatistics.rc	10/27/2021 2:36 PM	Resource Script	2 KB
Tc3_BaseStatistics.tmc	11/3/2021 12:48 PM	TMC File	33 KB
Tc3_BaseStatistics.tml	11/3/2021 12:48 PM	TML File	18 KB
Tc3_BaseStatistics.vcxproj	10/27/2021 2:36 PM	VC++ Project	15 KB
Tc3_BaseStatistics.vcxproj.filters	10/27/2021 2:36 PM	VC++ Project Filte...	11 KB
Tc3_BaseStatistics.vcxproj.user	10/27/2021 2:38 PM	Per-User Project O...	1 KB
Tc3_BaseStatistics_CreatePlcLibLog.txt	11/3/2021 12:48 PM	Text Document	1 KB
Tc3_BaseStatistics_ModuleGenerationLog.txt	11/3/2021 12:48 PM	Text Document	67 KB
Tc3_BaseStatistics_PublishLog.txt	11/3/2021 12:48 PM	Text Document	62 KB
Tc3_BaseStatisticsArtifacts.props	10/27/2021 2:36 PM	Project Property File	3 KB
Tc3_BaseStatisticsClassFactory.cpp	10/27/2021 2:36 PM	C++ Source	3 KB
Tc3_BaseStatisticsClassFactory.h	10/27/2021 2:36 PM	C/C++ Header	1 KB

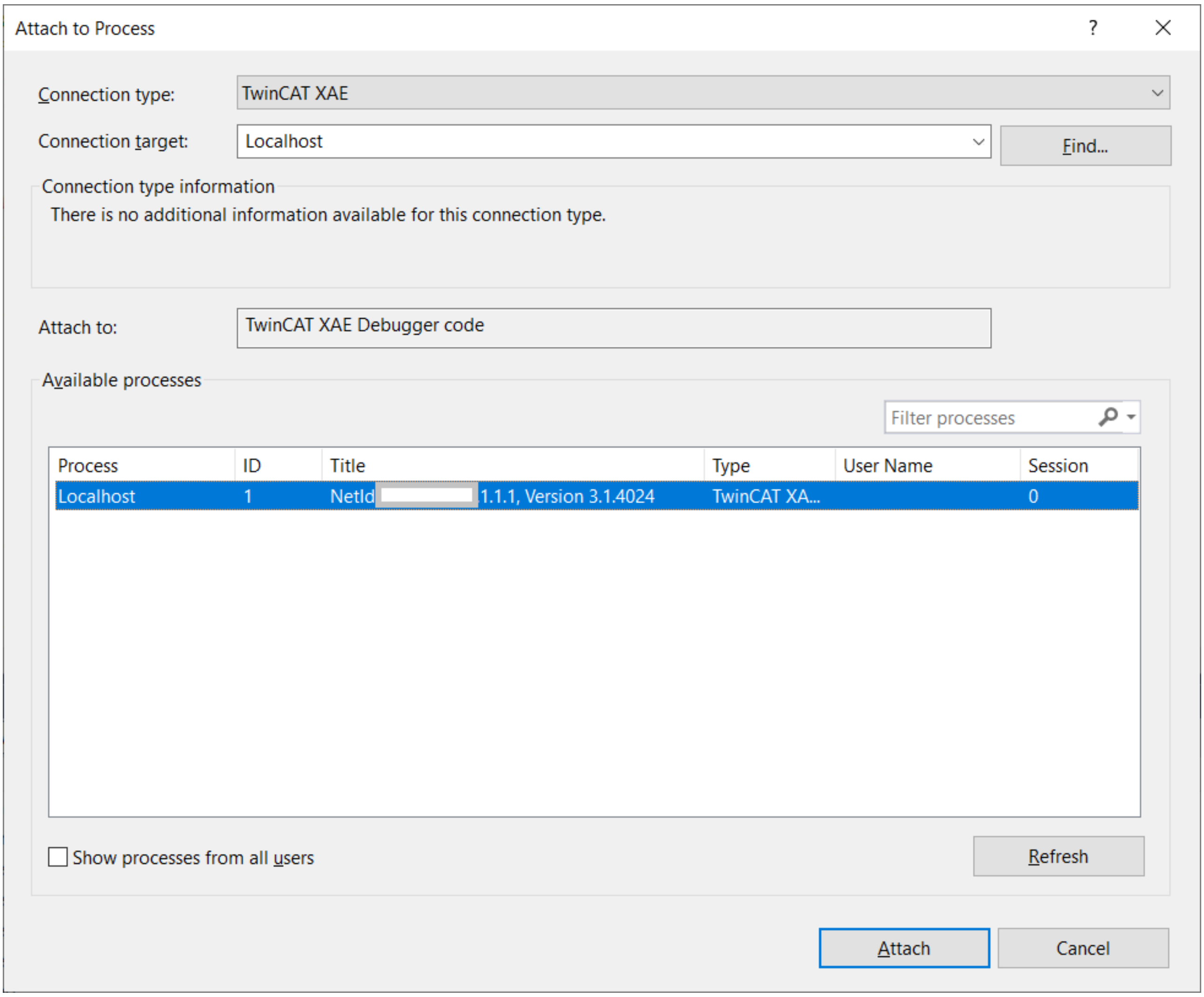
3. In the MATLAB® folder in the Visual Studio solution, view the subfolders that bear the name of the MATLAB® function created.

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


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'Tc3_BaseStatistics' (1 project)
Tc3_BaseStatistics
References
External Dependencies
MATLAB
BaseStatistic
Includes
Sources
BaseStatisticIterative
Includes
Sources
BaseStatisticIterative.cpp
BaseStatisticIterative_data.cpp
BaseStatisticIterative_initialize.cpp
BaseStatisticIterative_terminate.cpp
Resource Files
TMx
TwinCAT
xml
Tc3_BaseStatisticsArtifacts.props
BaseStatisticIterative.cpp
Tc3_BaseStatistics
(Global Scope)
100 %

```
//  Code generation for function 'BaseStatisticIterative'
//


// Include files
#include "BaseStatisticIterative.h"
#include "BaseStatisticIterative_types.h"
#include <cmath>

// Function Definitions
//
//  function [mean_x, std_x] = BaseStatisticIterative(x)

void BaseStatisticIterative(BaseStatisticIterativeStackData *SD, double x,
  double *mean_x, double *std_x)
{
  //  iterative implementation of mean and standard deviation
  //  persistent variable -> FB with internal state in PLC
  //  init of persistent vars
  // 'BaseStatisticIterative:10' if isempty(n)
  //  update standard deviation
  // 'BaseStatisticIterative:17' if n > 0
  if (SD->pd->n > 0.0) {
    double a;

    // 'BaseStatisticIterative:18' Sn = Sn + (n*x-Mn)^2/(n^2+n);
    a = SD->pd->n * x - SD->pd->Mn;
    SD->pd->Sn += a * a / (SD->pd->n * SD->pd->n + SD->pd->n);
  }

  //  update mean
```

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**.


Attach to Process

Connection type: TwinCAT XAE

Connection target: Localhost

Connection type information

There is no additional information available for this connection type.

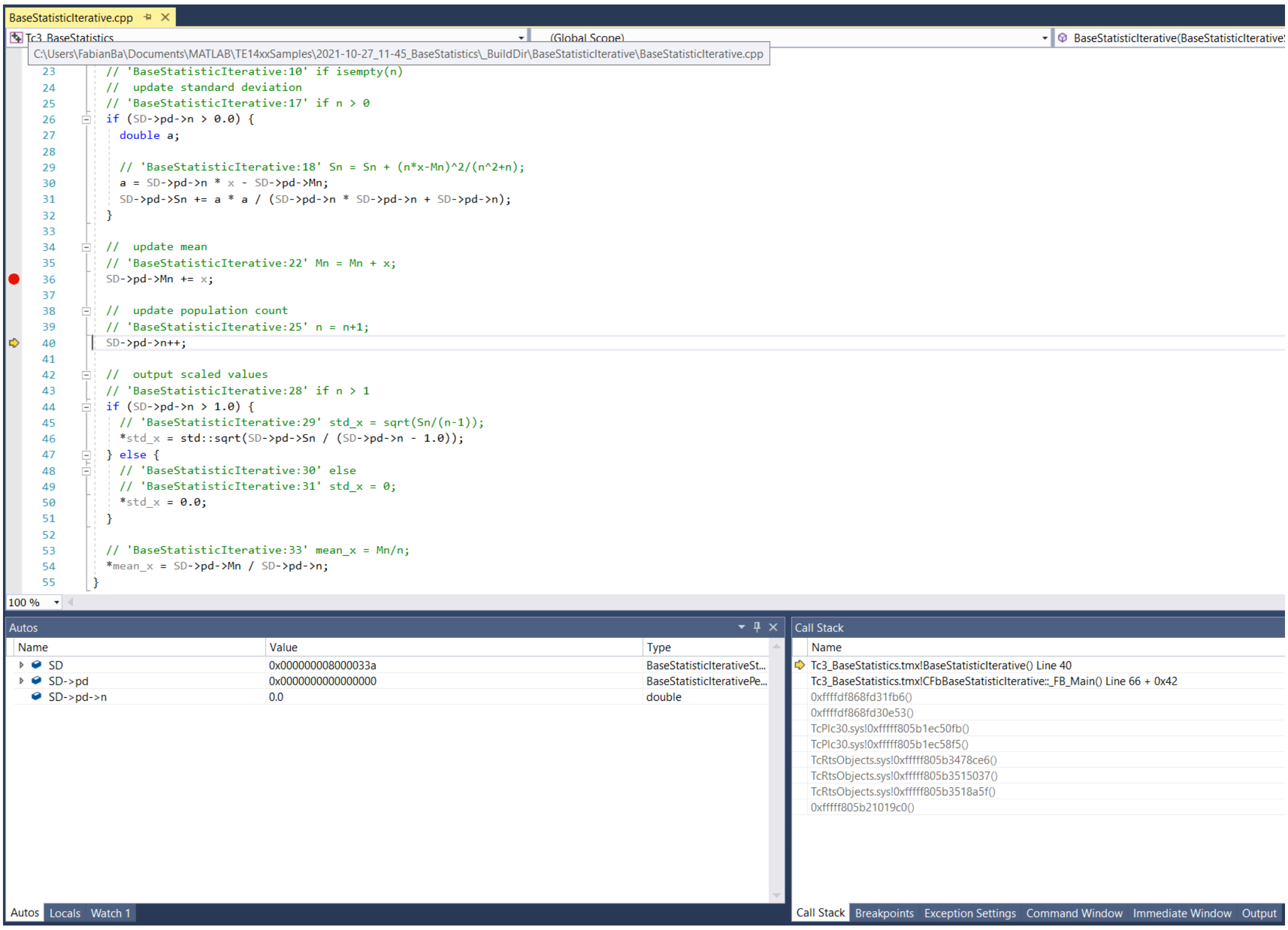
Attach to: TwinCAT XAE Debugger code

Available processes

Process	ID	Title	Type	User Name	Session
Localhost	1	NetId [redacted] 1.1.1, Version 3.1.4024	TwinCAT XA...		0

□ Show processes from all users

5. Set breakpoints in your C++ code and step through your code as usual.



C:\Users\FabianBa\Documents\MATLAB\TE14xxSamples\2021-10-27_11-45_BaseStatistics_BuildDir\BaseStatisticIterative\BaseStatisticIterative.cpp

```cpp
// 'BaseStatisticIterative:10' if isempty(n)
//  update standard deviation
// 'BaseStatisticIterative:17' if n > 0
if (SD->pd->n > 0.0) {
  double a;

  // 'BaseStatisticIterative:18' Sn = Sn + (n*x-Mn)^2/(n^2+n);
  a = SD->pd->n * x - SD->pd->Mn;
  SD->pd->Sn += a * a / (SD->pd->n * SD->pd->n + SD->pd->n);
}

//  update mean
// 'BaseStatisticIterative:22' Mn = Mn + x;
SD->pd->Mn += x;

//  update population count
// 'BaseStatisticIterative:25' n = n+1;
SD->pd->n++;

//  output scaled values
// 'BaseStatisticIterative:28' if n > 1
if (SD->pd->n > 1.0) {
  // 'BaseStatisticIterative:29' std_x = sqrt(Sn/(n-1));
  *std_x = std::sqrt(SD->pd->Sn / (SD->pd->n - 1.0));
} else {
  // 'BaseStatisticIterative:30' else
  // 'BaseStatisticIterative:31' std_x = 0;
  *std_x = 0.0;
}

// 'BaseStatisticIterative:33' mean_x = Mn/n;
*mean_x = SD->pd->Mn / SD->pd->n;
}
```

Autos

Name	Value	Type
SD	0x000000008000033a	BaseStatisticIterativeSt...
SD->pd	0x0000000000000000	BaseStatisticIterativePe...
SD->pd->n	0.0	double

Call Stack

```
Name
Tc3_BaseStatistics.tmx!BaseStatisticIterative() Line 40
Tc3_BaseStatistics.tmx!CFbBaseStatisticIterative::_FB_Main() Line 66 + 0x42
0xffffdf868fd31fb6()
0xffffdf868fd30e53()
TcPlc30.sys!0xfffff805b1ec50fb()
TcPlc30.sys!0xfffff805b1ec58f5()
TcRtsObjects.sys!0xfffff805b3478ce6()
TcRtsObjects.sys!0xfffff805b3515037()
TcRtsObjects.sys!0xfffff805b3518a5f()
0xfffff805b21019c0()
```

8.4 Exception handling

When processing the C++ code autogenerated from MATLAB® or Simulink® in TwinCAT, floating point exceptions can occur at runtime, for example if an unexpected value is passed into a function during programming. The handling of such exceptions is described below.

What is a floating point exception?

A floating point exception occurs when an arithmetically not exactly executable operation is instructed in the floating point unit of the CPU. IEEE 754 defines these cases: *inexact, underflow, overflow, divide-by-zero, invalid-operation*. If one of these cases occurs, a status flag is set, which indicates that the arithmetic operation cannot be executed exactly. It is further defined that each arithmetic operation must return a result – one that in the majority of cases leads to the possibility of ignoring the exception.

For example, a division by zero results in +inf or -inf. If a value is divided by inf in the further code, this results in zero, so that no consequential problems are to be expected. However, if inf is multiplied or other arithmetic operations are performed with inf, these are *invalid operations,* whose result is represented as a Not-a-Number (NaN).

How does the TwinCAT Runtime react in case of exceptions?

TwinCAT C++ Debugger not active

The following explanations only apply if the C++ debugger is **not** activated on the TwinCAT runtime system. When the C++ debugger is enabled, exceptions are caught by the debugger and can be handled, see Debugging.

Standard behavior

Default setting in TwinCAT is that at *divide-by-zero* and *invalid-operation* the execution of the program is stopped and TwinCAT issues an error message.

Task setting: Floating Point Exceptions

This default setting can be changed on the level of each TwinCAT task. If the checkbox "Floating Point Exception" is unchecked, an exception **does not** lead to a TwinCAT stop and **no** error message is issued. This setting is then valid for all objects that are called by this task. As a consequence, care must be taken in the application that NaN and inf values are handled accordingly in the program code.

Check for NaN and Inf

If, for example, a NaN is passed on via mapping to a TwinCAT object that has activated floating point exceptions, an arithmetic operation with NaN naturally leads to an exception in this object and subsequently to a TwinCAT stop. Therefore, NaN or inf must be checked directly after mapping. In the PLC, corresponding functions are available in the Tc2_Utilities library, e.g. LrealIsNaN.

Try-Catch statement

Another way to handle exceptions is to embed them in a try-catch statement. In the PLC the instructions __TRY, __CATCH, __FINALLY, __ENDTRY are available for this purpose. If floating point exceptions are enabled on the calling task and an exception occurs within the Try-Catch, it is caught in the Catch branch and can be handled. Accordingly, no variables are set to inf or NaN in this approach. However, it is also important to note that the code in the Try branch is run through only up to the point of the exception and then a jump is made to the Catch branch. In the application code, it should be noted that internal states in the Try branch may not be consistent.

Dump Files

From TwinCAT 3.1.4024.22 (XAR), dump files can be created at runtime in case of exceptions in the TcCOM object.

Specification of the behavior in case of exceptions on object level

In addition to the possibility of influencing the behavior in the event of exceptions at task level, the behavior can also be specified at the level of a TwinCAT object, i.e. the generated TcCOM or the generated PLC function block.

On the object level, a wealth of possibilities can be realized with the TwinCAT Target for Simulink®. Basically, however, all the options presented below are based on the above principles.

Optional ExecutionInfo Output

If exceptions are handled at object level, it makes sense to make corresponding information about occurred exceptions accessible at the object output. This output can be used to query whether an exception has occurred, what kind of exception it was, whether a dump file has been written, etc.

For the TcCOM object you can activate an additional output "ExecutionInfo [▶ 36]":

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

ExecutionInfo Output for PLC-FB

Currently the ExecutionInfo is only available for the TcCOM object. If you want to call the code from the PLC, use the TcCOM-Wrapper-FB.

The ExecutionInfo output is a structure with the following entries:

ExcecutionInfo structure

Entry	Data type	Meaning
CycleCount	ULINT	Current cycle count (independent of an exception)
ExceptionCount	ULINT	Number of exceptions that have occurred so far

Entry	Data type	Meaning
ActException	TcMgSdk.ExceptionInfo	More detailed explanation of the current exception (only first exception in the current cycle)

TcMgSdk.ExceptionInfo

ExceptionCode	DINT	Code of the exception, cf. Microsoft Help
TmxName	STRING(127)	Name of the tmx driver that threw the exception.
TmxVersion	ARRAY[0..3] OF UDINT	Version of the tmx driver that threw the exception.
InstructionAddr	UDINT	Relative address in memory; location where the exception occurred.
ReturnAddr	ARRAY[0..3] OF UDINT	Return addresses
DumpCreated	BOOLEAN	TRUE if a dump file was created for the exception.

With the *InstructionAddr* it is possible to judge if the exception with the given *ExceptionCode* always occurs at the same place in the source code. If the *InstructionAddr* is the same for repeating exceptions, it always occurs at the same point in the code. Via *ReturnAddr* you can see where the calls came from that led to the location of the exception. So you can judge if the call that leads to the exception always takes the same call path. If the code is called from outside the Tmx driver, there is a 0 in the *ReturnAddr*.

Definition of the object behavior in case of occurring exceptions

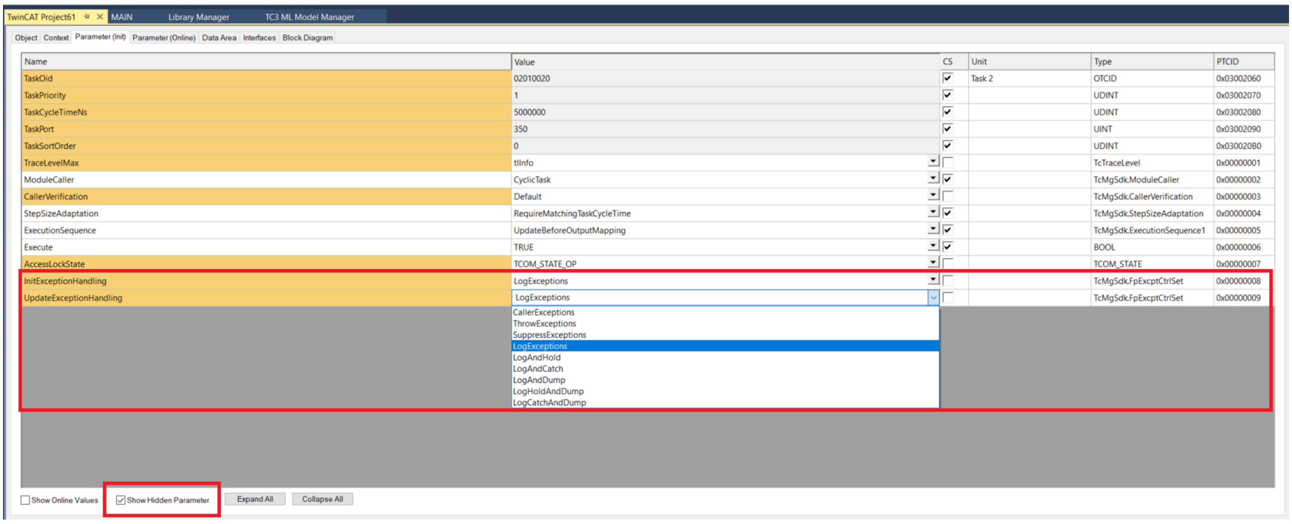
The behavior of a TcCOM object in case of exceptions can be set separately for the initialization phase and for the update phase under the properties of the class TwinCAT.ModuleGenerator.ProjectExportConfig:

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

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

Options are: `CallerExceptions, ThrowExceptions, SuppressExceptions, LogExceptions, LogAndHold, LogAndCatch, LogAndDump, LogHoldAndDump, LogCatchAndDump`.

If you are working with an already compiled TcCOM in TwinCAT, you can also change the settings on the object instance afterwards. To do this, use the tab **Parameters (Init)** and select **Show hidden Parameters**.



Name	Value	CS	Unit	Type	PTCID
TaskOid	02010020	☑	Task 2	OTCID	0x03002060
TaskPriority	1	☑		UDINT	0x03002070
TaskCycleTimeNs	5000000	☑		UDINT	0x03002080
TaskPort	350	☑		UINT	0x03002090
TaskSortOrder	0	☑		UDINT	0x030020B0
TraceLevelMax	tlInfo	☐		TcTraceLevel	0x00000001
ModuleCaller	CyclicTask	☑		TcMgSdk.ModuleCaller	0x00000002
CallerVerification	Default	☐		TcMgSdk.CallerVerification	0x00000003
StepSizeAdaptation	RequireMatchingTaskCycleTime	☑		TcMgSdk.StepSizeAdaptation	0x00000004
ExecutionSequence	UpdateBeforeOutputMapping	☑		TcMgSdk.ExecutionSequence1	0x00000005
Execute	TRUE	☑		BOOL	0x00000006
AccessLockState	TCOM_STATE_OP	☐		TCOM_STATE	0x00000007
InitExceptionHandling	LogExceptions	☐		TcMgSdk.FpExcptCtrlSet	0x00000008
UpdateExceptionHandling	LogExceptions	☐		TcMgSdk.FpExcptCtrlSet	0x00000009

CallerExceptions
ThrowExceptions
SuppressExceptions
LogExceptions
LogAndHold
LogAndCatch
LogAndDump
LogHoldAndDump
LogCatchAndDump

☐ Show Online Values ☑ Show Hidden Parameter

The settings for the PLC-FB FB_<ModelName> in the PLC library are independent of the settings for the TcCOM object.

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

Note that the other PLC function block (FB_<ModelName>_TcCOM) is a wrapper for a TcCOM object [▶ 51] and therefore the exception settings from the TcCOM section are valid when it is used.

A total of 9 different settings are available.

A total of 9 different settings are available.

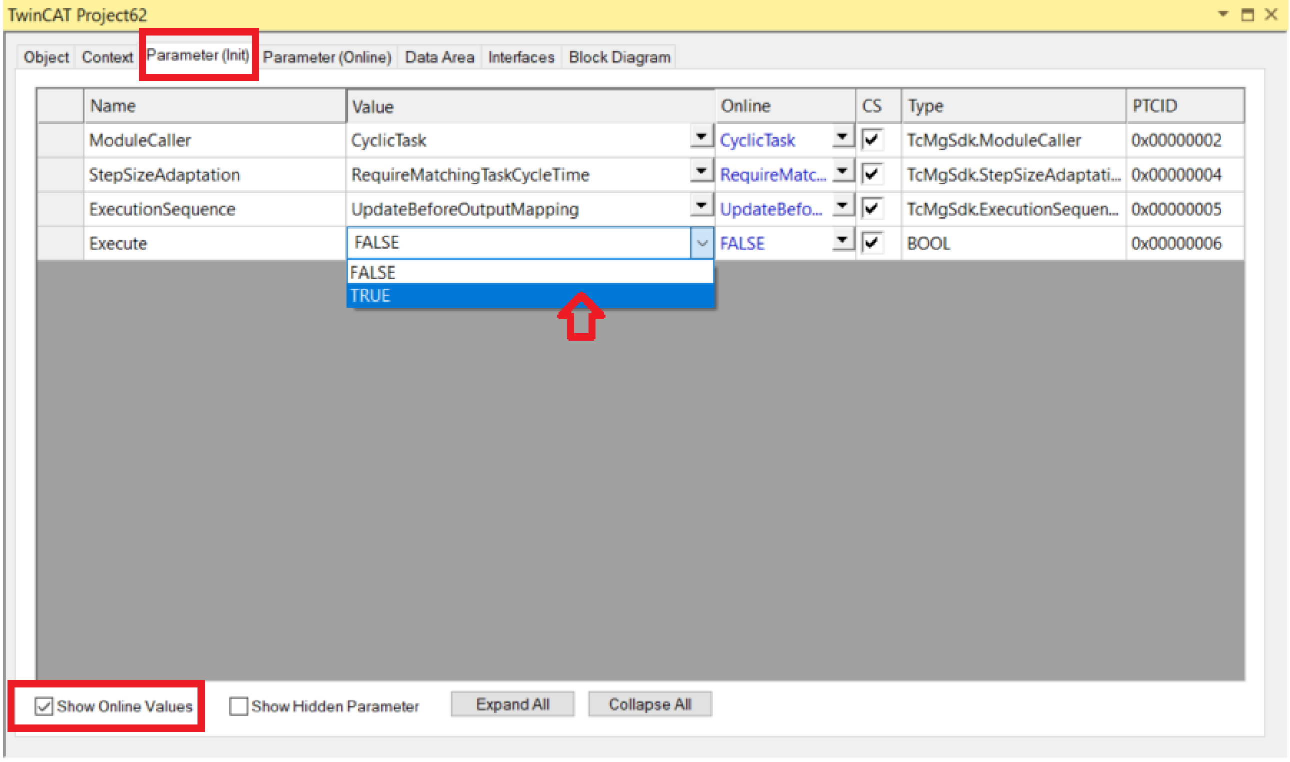
- **CallerExceptions** (default): Exceptions are thrown as configured at the calling task.
- **ThrowExceptions**: Exceptions in the TwinCAT object are thrown in any case, regardless of how the task is configured.
  - An exception causes a TwinCAT error message and a TwinCAT stop
- **SuppressExceptions**: Exceptions are not thrown, regardless of how the task is configured.
  - An exception does not cause a TwinCAT stop.
  - Outputs or internal states can be NaN or inf.
- **LogExceptions**: Exceptions are thrown, but do not lead to a TwinCAT stop.
  - An exception does not cause a TwinCAT stop.
  - Outputs or internal states can be NaN or inf.
  - The ExecutionInfo output is filled with information about an exception in the current cycle. If several exceptions occur in one cycle, only the first exception is displayed at the output. When the TwinCAT object is called again, the information is reset.
- **LogAndHold**: Exceptions are thrown. The execution of the TwinCAT object is stopped.
  - An exception does not cause a TwinCAT stop.
  - Outputs or internal states can be NaN or inf.
  - The ExecutionInfo output is filled with information about an exception in the current cycle. If several exceptions occur in one cycle, only the first exception is displayed at the output. When the TwinCAT object is called again, the information is reset.
  - The execution of the TwinCAT object is stopped after an exception occurs. TwinCAT itself remains in run mode. Restart execution: .
- **LogAndCatch**: Exceptions are caught with try-catch in the TwinCAT object. The execution of the TwinCAT object is stopped.
  - An exception does not cause a TwinCAT stop.
  - Outputs or internal states **cannot** contain NaN or inf.
  - The ExecutionInfo output is filled with information about an exception in the current cycle.
  - The execution of the code ends at the point of the exception. From there, the program jumps to the catch junction, i.e. internal states can be inconsistent.
  - The execution of the TwinCAT object is stopped after an exception occurs. TwinCAT itself remains in run mode. Restart execution: .
- **LogAndDump, LogHoldAndDump and LogCatchAndDump**
  - Behavior like LogExceptions
  - Additionally a dump file is stored on the runtime system in the TwinCAT folder *Boot*. For more on dump files, see here [▶ 61].

Handle execution stop of a TwinCAT object

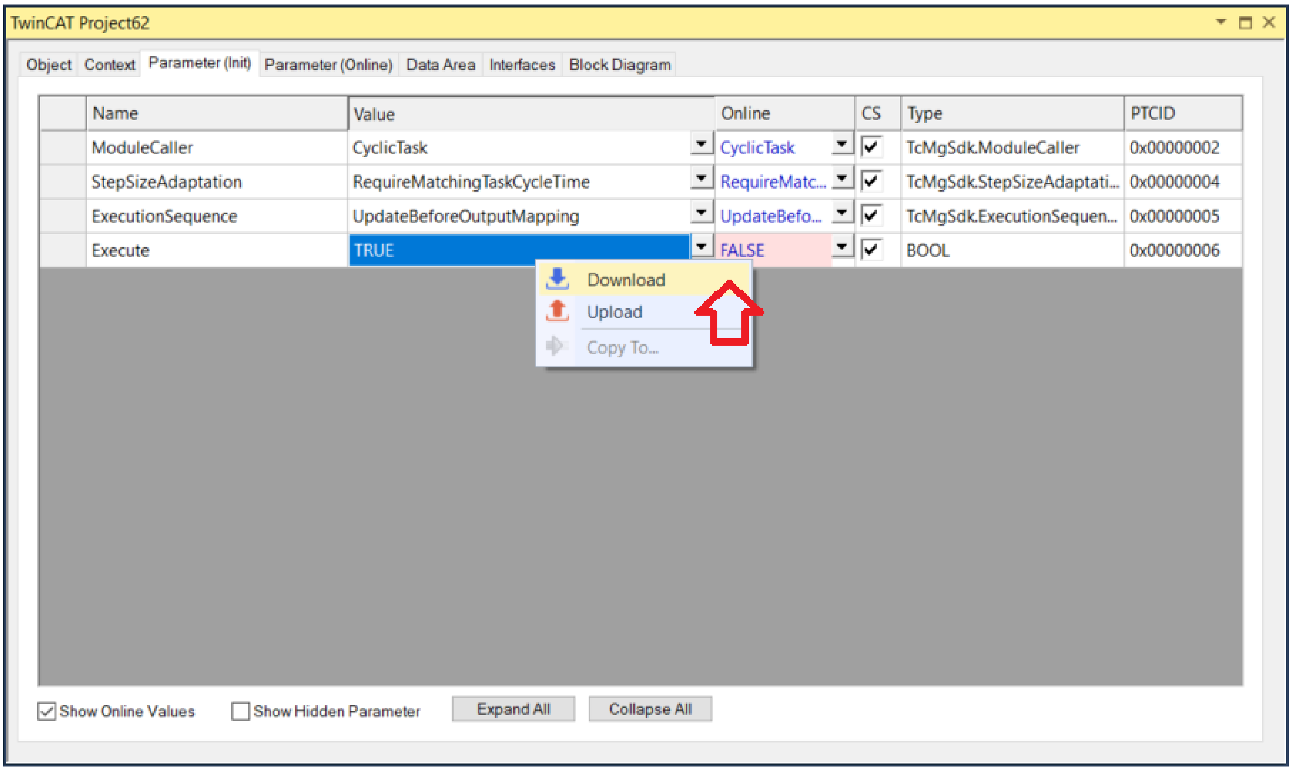
LogAndHold and LogHoldAndDump

In the event of an exception, execution of the code in the TcCOM object concerned is stopped by setting the Execute parameter to FALSE. The parameter can be read or written from the XAE and via ADS.

In the XAE, you can display and change the online values of the TcCOM object under Parameters (Init).

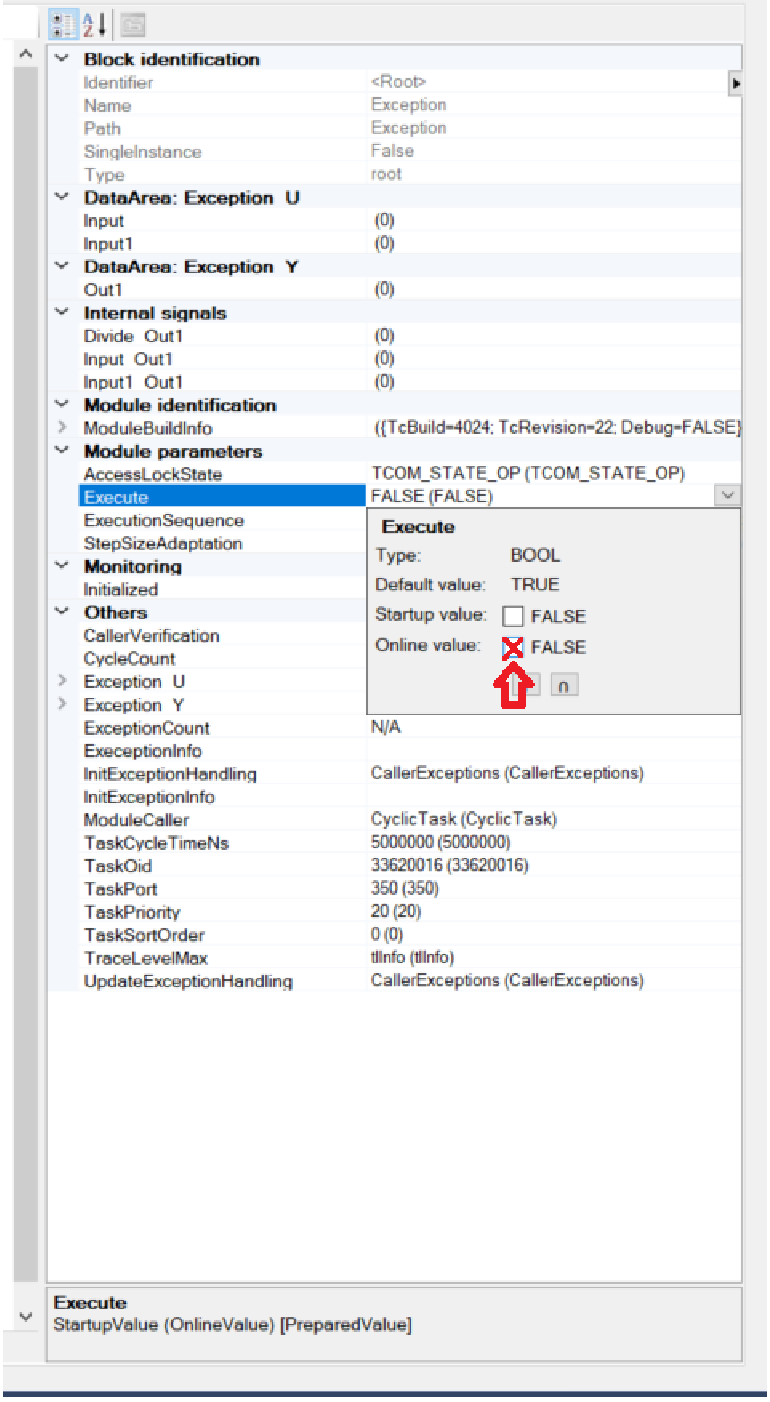


Name	Value	Online	CS	Type	PTCID
ModuleCaller	CyclicTask	CyclicTask	☑	TcMgSdk.ModuleCaller	0x00000002
StepSizeAdaptation	RequireMatchingTaskCycleTime	RequireMatc...	☑	TcMgSdk.StepSizeAdaptati...	0x00000004
ExecutionSequence	UpdateBeforeOutputMapping	UpdateBefo...	☑	TcMgSdk.ExecutionSequen...	0x00000005
Execute	FALSE	FALSE	☑	BOOL	0x00000006



Name	Value	Online	CS	Type	PTCID
ModuleCaller	CyclicTask	CyclicTask	☑	TcMgSdk.ModuleCaller	0x00000002
StepSizeAdaptation	RequireMatchingTaskCycleTime	RequireMatc...	☑	TcMgSdk.StepSizeAdaptati...	0x00000004
ExecutionSequence	UpdateBeforeOutputMapping	UpdateBefo...	☑	TcMgSdk.ExecutionSequen...	0x00000005
Execute	TRUE	FALSE	☑	BOOL	0x00000006

In the block diagram the parameter is offered to you under *Module parameters*.

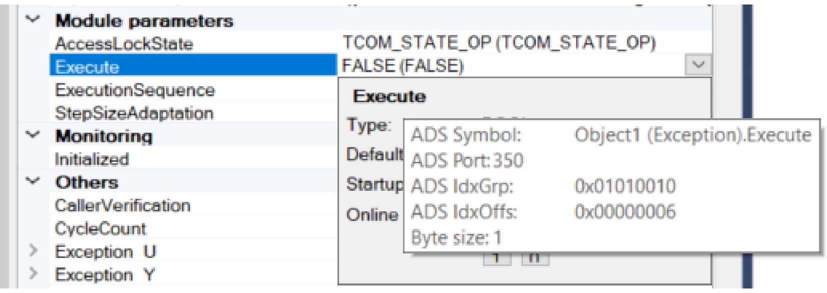


Block identification	
Identifier	<Root>
Name	Exception
Path	Exception
SingleInstance	False
Type	root
DataArea: Exception U	
Input	(0)
Input1	(0)
DataArea: Exception Y	
Out1	(0)
Internal signals	
Divide Out1	(0)
Input Out1	(0)
Input1 Out1	(0)
Module identification	
ModuleBuildInfo	({TcBuild=4024; TcRevision=22; Debug=FALSE}
Module parameters	
AccessLockState	TCOM_STATE_OP (TCOM_STATE_OP)
Execute	FALSE (FALSE)
ExecutionSequence	
StepSizeAdaptation	
Monitoring	
Initialized	
Others	
CallerVerification	
CycleCount	
Exception U	
Exception Y	
ExceptionCount	N/A
ExeceptionInfo	
InitExceptionHandling	CallerExceptions (CallerExceptions)
InitExceptionInfo	
ModuleCaller	CyclicTask (CyclicTask)
TaskCycleTimeNs	5000000 (5000000)
TaskOid	33620016 (33620016)
TaskPort	350 (350)
TaskPriority	20 (20)
TaskSortOrder	0 (0)
TraceLevelMax	tlInfo (tlInfo)
UpdateExceptionHandling	CallerExceptions (CallerExceptions)

Execute
Type: BOOL
Default value: TRUE
Startup value: □ FALSE
Online value: ☒ FALSE

Execute
StartupValue (OnlineValue) [PreparedValue]

If you move the mouse over the Execute name in the change dialog, you will be shown the ADS address of the parameter, as with all other parameters. This allows you to set the parameter also by ADS.

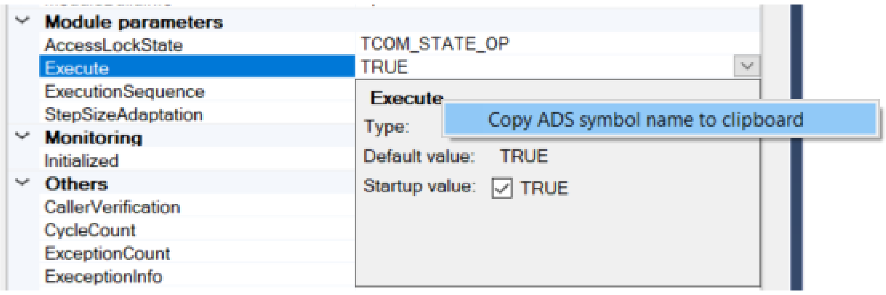


Module parameters	
AccessLockState	TCOM_STATE_OP (TCOM_STATE_OP)
Execute	FALSE (FALSE)
ExecutionSequence	
StepSizeAdaptation	
Monitoring	
Initialized	
Others	
CallerVerification	
CycleCount	
Exception U	
Exception Y	

Execute
Type:
Default
Startup
Online

ADS Symbol: Object1 (Exception).Execute
ADS Port: 350
ADS IdxGrp: 0x01010010
ADS IdxOffs: 0x00000006
Byte size: 1

By right-clicking on the name **Execute** you can also save the ADS symbol information to the clipboard. This also applies to all other parameters.



Module parameters
AccessLockState TCOM_STATE_OP
Execute TRUE
ExecutionSequence
StepSizeAdaptation
Monitoring
Initialized
Others
CallerVerification
CycleCount
ExceptionCount
ExeceptionInfo

Execute
Type:
Default value: TRUE
Startup value: ☑ TRUE

LogAndCatch and LogCatchAndDump

In addition to the parameter *Execute*, the online parameter *Initialized* also changes to FALSE in the case of LogAndCatch and LogCatchAndDump. The module must be reinitialized before the module can perform a calculation again. This is necessary because internal states can no longer be consistent. Reinitialization can only be performed by returning the TcCOM object to the "Init" state and moving it to OP again.

At runtime, only TcCOM objects that have no mappings can be shut down, otherwise active mappings would block the shutdown. A new initialization is only possible by restarting the entire TwinCAT Runtime.

A more flexible alternative is to use the TcCOM-Wrapper-FB [▶ 51] in the PLC. This can be used to call the TcCOM from the PLC and does not require any mappings to access its inputs and outputs. Accordingly, the TcCOM object can also be reinitialized during runtime.

```
PROGRAM MAIN
VAR
   stInitTemp : ST_FB_SimpleTempCtrl_TcCOM_InitStruct := (nOid := 16#01010010);
   fbTempCtr  : FB_SimpleTempCtrl_TcCOM_InitStruct(stInitTemp);
   Inputs     : ST_ExtU_SimpleTempCtrl_T;
   Outputs    : ST_ExtY_SimpleTempCtrl_T;
   ExecutionOut : ST_ExecutionInfo2;
END_VAR
```

```
// check if TcCOM is in OP mode and all set
IF fbTempCtr.bExecute = TRUE AND fbTempCtr.bInitialized = TRUE AND fbTempCtr.nObjectState = TCOM_STA
TE.TCOM_STATE_OP THEN

    // call the module
    fbTempCtr(stSimpleTempCtrl_U := Inputs, stSimpleTempCtrl_Y => Outputs, stExecutionInfo => Execut
ionOut);

    // handle exceptions
    IF ExecutionOut.ActException.ExceptionCode <> 0 THEN
       // collect exception information
       (* ...... *)

       // reinit TcCOM
       fbTempCtr.Reinit(stReInit := stInitTemp);
    END_IF

 END_IF
```

Note that the ReInit method is executed synchronously, i.e. depending on the cycle time and the time required to reinitialize, cycle overruns may occur.

Dump files

Writing the dump file may take a few cycles. It is best to use a separate task for the TcCOM object or the PLC-FB in question that does not block any important tasks.

Dump files are only written with a TwinCAT XAR version >= 3.1.4024.22, otherwise you get a corresponding warning.

In the case of *LogAndDump* the execution of the code is continued cyclically after the occurrence of an exception, accordingly exceptions can occur cyclically which could lead to persistent cycle timeouts. Therefore, the online value of the parameter *UpdateExceptionHandling* is set to *LogExceptions* after the dump file has been written, i.e. the writing of dump files is deactivated, but can subsequently be switched on again, e.g. by ADS or intervention via the XAE under parameter (Init).

The created dump file is stored on the runtime PC in the boot folder and can be copied from there to another PC for analysis. If you use a TwinCAT version lower than 3.1.4024.x you can open the dump files with WinDbg and start your analysis.

8.5 Using Realtime Monitor time stamps

MATLAB® commands, such as tic and toc, are popular ways to analyze the performance of code sections in MATLAB®. These commands are not usable in this form during TwinCAT runtime.

For this purpose, TwinCAT provides the TwinCAT Realtime Monitor, which evaluates time stamps in the source code and displays them for analysis. Setting Realtime Monitor time stamps is supported in MATLAB® code, i.e. the time stamps are set in MATLAB® and can be evaluated by the Realtime Monitor after code generation and instantiation in TwinCAT. Running time stamps in MATLAB® results in output to the MATLAB® console.

Class: TwinCAT.ModuleGenerator.Realtime.LogMark

Methods: Start, Stop and Mark

MATLAB® documentation: `doc("TwinCAT.ModuleGenerator.Realtime.LogMark")`

Example in MATLAB®

```
TwinCAT.ModuleGenerator.Samples.Start("BaseStatisticsLogMark")
```

9 FAQ

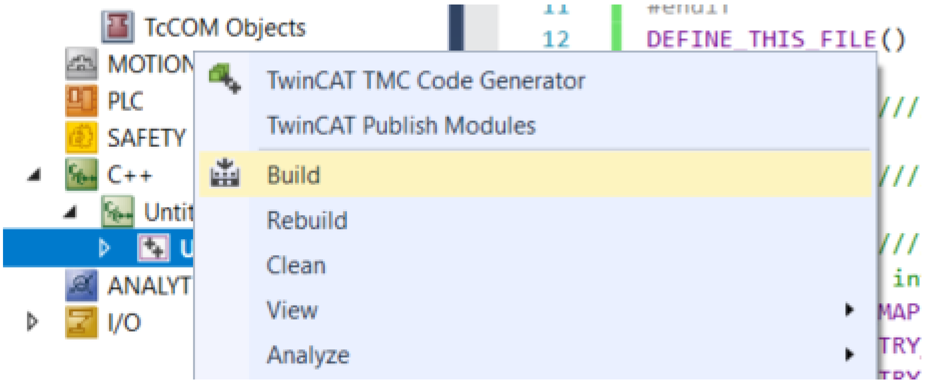
9.1 Build of a sample fails

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.

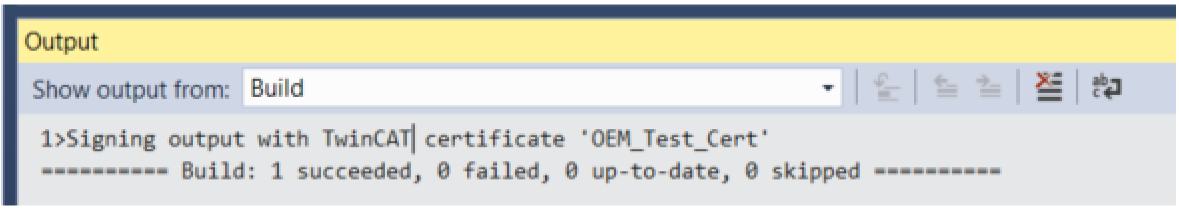
✓ To test the platform toolset without the influence of MATLAB® please create a TwinCAT Versioned C++ project in TwinCAT (open TwinCAT in Visual Studio) .

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.



Output

Show output from: Build

```
1>Signing output with TwinCAT certificate 'OEM_Test_Cert'
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

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

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

- **Direct file access:**: the access to the file system of the IPC is restricted from the TwinCAT runtime. To read and write files from TwinCAT, use: fopen, fclose, fread and fwrite. See sample: `TwinCAT.ModuleGenerator.Samples.Show('FileAccess')`.
- **Direct hardware access**: direct access to devices/interfaces requires a corresponding driver, e.g. RS232, USB, network card, etc. It is not possible to access the device drivers of the operating system from the real-time context. 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.
- **Precompiled libraries**: it is possible that during code generation by the MATLAB Coder™, no platform-independent C/C++ code is generated, but precompiled libraries are included. In these cases, no real-time execution in TwinCAT is possible. The coder.HardwareImplementation setting helps to check whether generic C/C++ code can be generated. For example

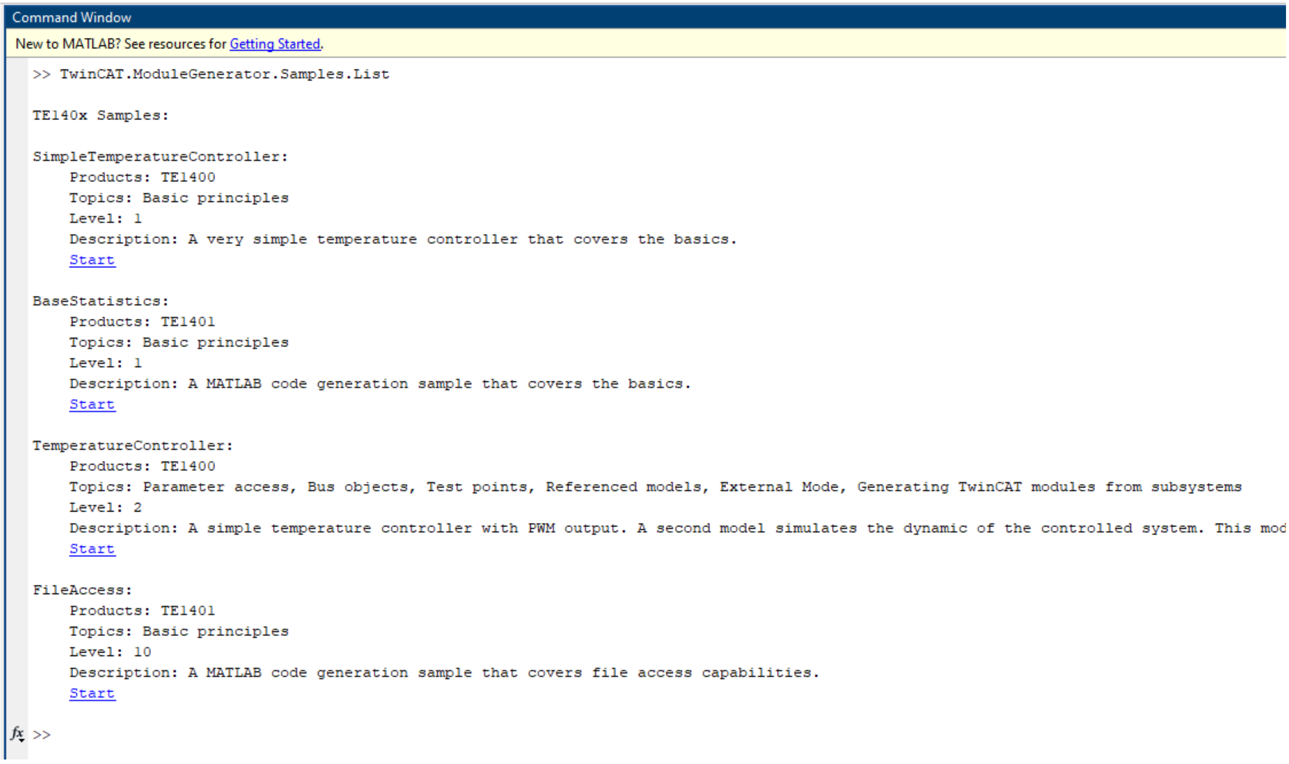
`coder.HardwareImplementation.ProdHWDeviceType = 'Generic->32-bit x86 compatible';` and `coder.HardwareImplementation.TargetHWDeviceType = 'Generic->32-bit x86 compatible';`

10 Samples

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

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

```
TwinCAT.ModuleGenerator.Samples.List
```



Command Window

New to MATLAB? See resources for Getting Started.

```
>> TwinCAT.ModuleGenerator.Samples.List

TE140x Samples:

SimpleTemperatureController:
    Products: TE1400
    Topics: Basic principles
    Level: 1
    Description: A very simple temperature controller that covers the basics.
    Start

BaseStatistics:
    Products: TE1401
    Topics: Basic principles
    Level: 1
    Description: A MATLAB code generation sample that covers the basics.
    Start

TemperatureController:
    Products: TE1400
    Topics: Parameter access, Bus objects, Test points, Referenced models, External Mode, Generating TwinCAT modules from subsystems
    Level: 2
    Description: A simple temperature controller with PWM output. A second model simulates the dynamic of the controlled system. This mod
    Start

FileAccess:
    Products: TE1401
    Topics: Basic principles
    Level: 10
    Description: A MATLAB code generation sample that covers file access capabilities.
    Start

fx >>
```

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

Also available are

```
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.

```
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++ 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.

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

```
function this = Tc3AutomationInterface
```

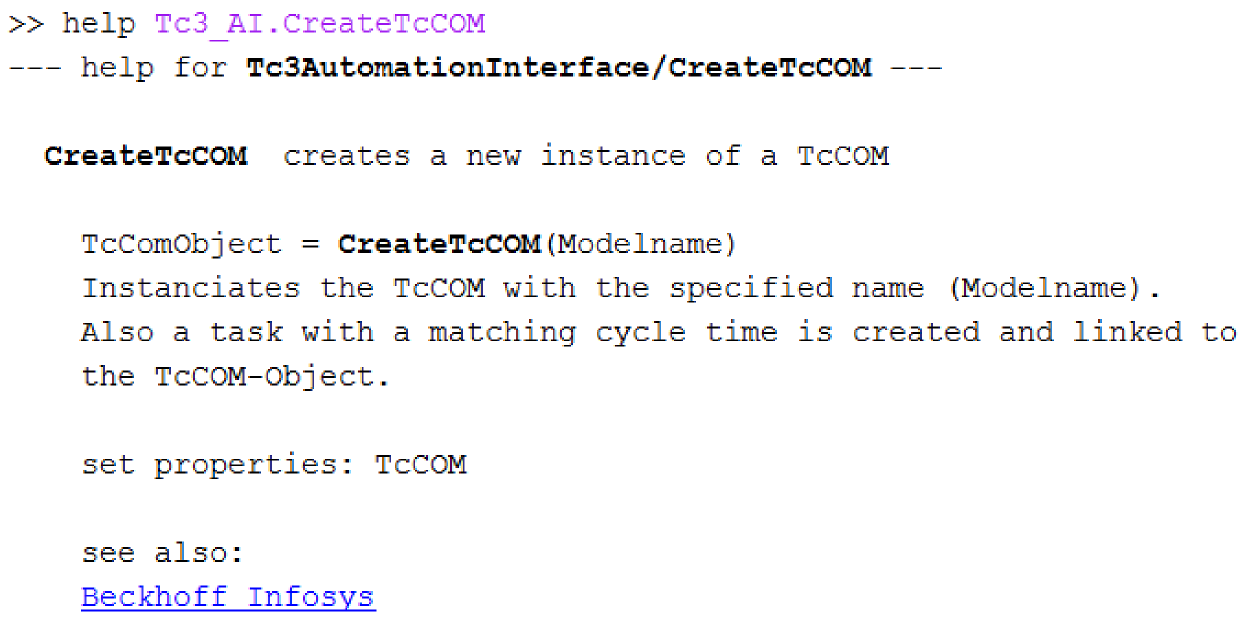
The constructor loads all necessary assemblies and, if successful, sets the AssembliesLoaded property to TRUE. The loaded assemblies are:

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

Selected methods of the class

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

Use the MATLAB® help functions in order to view the function and the parameters of the method.



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

  CreateTcCOM  creates a new instance of a TcCOM

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

    set properties: TcCOM

    see also:
    Beckhoff Infosys
```

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

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

```
function ActivateOnDevice(this, AmsNetId)
```

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

Static methods

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

```
function vsVersions = GetInstalledVisualStudios
```

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

Documents about this

🗎 AutomationInterfaceMATLAB (Resources/zip/5776206091.zip)

More Information:
www.beckhoff.com/te1401

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