Manual

MATLAB®/Simulink®

TwinCAT 3

Version: 1.2
Date: 2018-11-13
# Table of contents

1 Foreword .................................................................................................................................................. 5  
  1.1 Notes on the documentation........................................................................................................... 5  
  1.2 Safety instructions .......................................................................................................................... 6  
2 Overview.................................................................................................................................................. 7  
3 Block diagram ........................................................................................................................................ 10  
  3.1 Using the block diagram .................................................................................................................. 10  
  3.2 Integrating the block diagram controls ............................................................................................ 11  
  3.3 Debugging ......................................................................................................................................... 12  
  3.4 Display signal curves ....................................................................................................................... 17  
  3.5 Module parameterization in the block diagram ................................................................................. 19  
4 ADS communication from MATLAB ................................................................................................... 21  
  4.1 Examples ........................................................................................................................................... 21  
    4.1.1 Accessing an array in the PLC .................................................................................................. 21  
    4.1.2 Event driven reading ............................................................................................................... 22  
5 TwinCAT Automation Interface: Use in MATLAB® ............................................................................ 25  
  5.1 Sample: Tc3AutomationInterface .................................................................................................. 25
1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development. We reserve the right to revise and change the documentation at any time and without prior announcement. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

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

Patent Pending

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

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany. The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

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

Tip or pointer

This symbol indicates information that contributes to better understanding.
2 Overview

MATLAB®/Simulink®

MATLAB®/Simulink® is a tool for computer-aided modeling, simulation and analysis of physical or biological systems, for example. It has been widely used for many years in research and development. The program is developed and distributed by the company The Mathworks. The models of the simulated systems are usually implemented graphically in the form of block diagrams, as is customary in control technology. The standard Simulink library contains a large number of function blocks that can be used to represent the behavior of any system. In addition to this standard library, there are toolboxes for various application areas such as event-based systems, physical systems, control engineering, signal and image processing and much more, which save time and simplify modeling.

Use of MATLAB®/Simulink® in automation technology

One of the strengths of Simulink®, which is particularly interesting for automation technology, is model-based controller optimization. Functions from the higher-level MATLAB® software environment with corresponding toolboxes can be used for designing control systems based on calculation methods derived from control theory. Controllers designed in this way can then be simulated and validated in Simulink®, based on a model of the closed control loop.

The software can be used to model and test not only simple control loops and systems, but also whole production plants. This offers several advantages in the development process of a machine. For example, for initial testing of the control software it is sufficient to use a suitable model of the machine for virtual commissioning, rather than a physical machine. This also reduces the risk of damage to the machine caused by faulty control software during initial commissioning. For virtual commissioning the TwinCAT 3 function TE1111 EtherCAT Simulation enables simulation of an EtherCAT segment.

The various Simulink® toolboxes open up a wide range of further application options, including sequential machine control with Stateflow® or measurement data evaluation by adding a streaming signal processing system with the DSP Systems Toolbox™.

TE1400 TwinCAT Target for MATLAB®/Simulink®

To use a model from MATLAB®/Simulink® in the actual machine after successful testing, the corresponding algorithms can be transferred to a PLC program through manual coding, for example. However, automatic conversion of already implemented algorithms to real-time capable program components is simpler and significantly less error-prone.

TE1400 TwinCAT Target for MATLAB®/Simulink® can be used to generate real-time capable modules from MATLAB®/Simulink®, which can be executed in the TwinCAT 3 runtime. These modules can be instantiated, parameterized and debugged multiple times in the TwinCAT 3 development environment.
Further information can be found in section TE1400 TwinCAT Target for MATLAB®/Simulink®.

TE1410 interface for MATLAB®/Simulink®

The TE1410 interface for MATLAB®/Simulink® can be used for data exchange between TwinCAT3 and MATLAB®/Simulink®. The data exchange takes place via ADS with the aid of Simulink blocks, which are provided in a Simulink library.
Further information can be found in section TE1410 interface for MATLAB®/Simulink®.

**Webinars concerning TE1400 and TE1410**

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Referent</th>
</tr>
</thead>
<tbody>
<tr>
<td>01.10.2013</td>
<td>TwinCAT 3</td>
<td>Matlab®/Simulink®-integration: Introduction, application examples, TC3 Interface for Matlab®/Simulink®. (german)</td>
</tr>
</tbody>
</table>

Overview of the Beckhoff webinars (german)
3 Block diagram

When generating a TwinCAT module from MATLAB®/Simulink®, the block diagram can optionally also be exported. In this case, the block diagram can be displayed in the TwinCAT development environment under the Block Diagram tab of the module instance:

This control not only enables navigation through the entire structure of the block diagram, it also enables viewing and changing of parameter values, display of signal values and their history, and - in debugging mode - module debugging via breakpoints. The control is designed such that it can also be used in a separate visualization.

Installation

The block diagram setup is executed with the TwinCAT 3 Setup and does not have to be executed separately. The version number of the block diagram can be viewed by right-clicking in the control and selecting About TC3 BlockDiagram. Alternatively, you can view the version number in the Control Panel under Programs and Features (entry Beckhoff TwinCAT 3 BlockDiagram).

3.1 Using the block diagram

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

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

Shortcut functions:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>Zoom to current size of the block diagram tab</td>
</tr>
<tr>
<td>Backspace</td>
<td>Switch to the next higher hierarchy level</td>
</tr>
</tbody>
</table>
Context menu functions:

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

3.2 Integrating the block diagram controls

The option displaying the block diagram in the TwinCAT XAE environment can also be integrated in separate visualizations.

The following steps are required:

1. Create a new Windows Forms application
2. Add the TwinCAT.BlockDiagramm.dll to the toolbox:
3. Select "Choose Items ..." from the context menu
4. Navigate to TwinCAT.Blockdiagram.dll, which can be found under <TwinCAT installation path>\3.1\Components\TcBlockDiagram

5. Add a TcBlockdiagram control instance to the Windows Forms object using drag & drop

3.3 Debugging

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

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

Prerequisites for debugging within the block diagram are:

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

Setting breakpoints in the block diagram

1. After attaching the debugger to the TwinCAT runtime, the possible breakpoints are assigned to the blocks in the block diagram and represented as points. Clicking on the desired breakpoint activates it, so that execution of the module instance is stopped next time the associated block is executed. The color of the point provides information about the current state of the breakpoint:
   - Grey: Breakpoint inactive
   - Red: Breakpoint active. The program is stopped next time this block is executed
   - Yellow point in the middle: Breakpoint hit. Program execution is currently stopped at this point
   - Blue point in the middle: Breakpoint hit (as yellow), but in a different instance of the module.
2. Additional information, such as the corresponding C++ code section, can be found in the tooltip for the breakpoint:

Breakpoints are not always assigned to a single block. In many cases, the functions of several blocks are consolidated in a code section or even a line in the underlying C++ code. This means that several blocks can share the same breakpoint. Therefore, activation of a breakpoint in the block diagram may also result in changes in the point display in other blocks.

**Evaluating exceptions**

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

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

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

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

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

3.4 Display signal curves

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

Display signal curves in the block diagram

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

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

Display the scope in the block diagram

The title bar of the scope window offers the following options

| Close window |
| Keep window in the foreground through all block diagram hierarchies |
| Minimize window to the title bar |
Displaying the scope in the block diagram control requires a Scope View Professional license. In Visual Studio, this is not necessary.

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

Display signal curves in TwinCAT Scope

If the item is dropped onto an axis in a Scope project, rather than the block diagram, the signal is added there.

Add a signal in a TwinCAT Scope project

3.5 Module parameterization in the block diagram

Section Parameterization of the generated module describes the general parameterization options for a TcCom object. The following section focuses on parameterization in the block diagram.

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

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

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

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

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

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

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

Parameterization in the block diagram

Parameterizable blocks are marked with a yellow box in the block diagram.
Double-clicking on the block or a single click on the yellow box brings up a window with the parameters that can be changed.

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

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

The icons in the title bar have the following functions:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>❌</td>
<td>Close window</td>
</tr>
<tr>
<td>🥄</td>
<td>Keep window in the foreground across all block diagram hierarchy levels</td>
</tr>
<tr>
<td>⬆️</td>
<td>Keep window open at the current block diagram hierarchy level</td>
</tr>
<tr>
<td>₋️</td>
<td>Minimize window to title bar</td>
</tr>
</tbody>
</table>
ADS communication from MATLAB

No additional software is required for ADS communication from a MATLAB script. After installation of TwinCAT, the .NET class library "TwinCAT.ADS.dll" is available, for example. This provides .NET classes for realizing an ADS communication that can be used directly in the MATLAB script.

### 4.1 Examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1: Accessing an array in the PLC</td>
<td><a href="https://infosys.beckhoff.com/content/1033/tc3_matlabOverview/Resources/zip/9007200794646283.zip">Link</a></td>
</tr>
<tr>
<td>Example 2: Event driven reading</td>
<td><a href="https://infosys.beckhoff.com/content/1033/tc3_matlabOverview/Resources/zip/9007200794644619.zip">Link</a></td>
</tr>
</tbody>
</table>

#### 4.1.1 Accessing an array in the PLC

**Overview**

This example is based on Visual C# Example 1. [Link](https://infosys.beckhoff.com/content/1033/tc3_matlabOverview/Resources/zip/9007200794646283.zip) contains the files described here.

**Matlab function**

```matlab
function[] = Matlab_ADS_Sample()
```

**Import Ads.dll**

```matlab
AdsAssembly = NET.addAssembly('C:\TwinCAT\AdsApi\NET\v2.0.50727\TwinCAT.Ads.dll');
import TwinCAT.Ads.*
```

**Create instance of class TcAdsClient**

```matlab
tcClient = TcAdsClient;
```

**Establish connection to the port 851 on the local computer**

```matlab
tcClient.Connect(851);
try
hVar = tcClient.CreateVariableHandle('Main.PLCVar');
catch err
tcClient.Dispose();
msgbox(err.message,'Fehler beim Erstellen des Variablenhandles','error');
error(err.message);
end
```

**AdsStream for recording data**

```matlab
dataStream = AdsStream(100 * 2);
binRead = AdsBinaryReader(dataStream);
```
Read array
```
tcClient.Read(hVar,dataStream);
for i=1:1:100
disp(binRead.ReadInt16)
end
catch err
tcClient.Dispose();
msgbox(err.message,'Fehler beim Lesen des Arrays','error');
error(err.message);
end
```

Release resource
```
tcClient.Dispose();
```

PLC program
```
PROGRAM MAIN
VAR
PLCVar : ARRAY [0..99] OF INT;
Index: BYTE;
END_VAR
FOR Index := 0 TO 99 DO
PLCVar[Index] := 3500 + INDEX;
END_FOR
```

4.1.2 Event driven reading

Overview
This example is based on Visual C# Example 2. [https://infosys.beckhoff.com/content/1033/tc3_matlab_overview/Resources/zip/9007200794644619.zip](https://infosys.beckhoff.com/content/1033/tc3_matlab_overview/Resources/zip/9007200794644619.zip) contains the files described here.

The function "OnNotification.m" is automatically called when an ADS notification arrives. The script file "MatlabADSSample_Notification.m" registers the notifications and removes them after a certain time.

MatlabAdsSample_Notification.m
The script file is executed once. First, the ADS notifications are created, and the function "OnNotification" is registered as a callback. After a certain time, during which ADS notifications were received, the notifications are deregistered.

Import Ads.dll
```
AdsAssembly = NET.addAssembly('C:\TwinCAT\AdsApi\NET\v4.0.30319\TwinCAT.Ads.dll');
import TwinCAT.Ads::*;
```

Create instance of class TcAdsClient
```
tcClient = TcAdsClient;
```

Establish connection to the port 851 on the local computer
```
tcClient.Connect(851);
```

Auxiliary variables for dealing with notifications
```
% ADS stream
dataStream = AdsStream(31);
% reader
binRead = AdsBinaryReader(dataStream);
% notification handles
hConnect = zeros(7,1);
% variables to be read from target by notification
boolVarName = 'MAIN.boolVal';
intVarName = 'MAIN.intVal';
dintVarName = 'MAIN.dintVal';
sintVarName = 'MAIN.sintVal';
lrealVarName = 'MAIN.lrealVal';
realVarName = 'MAIN.realVal';
stringVarName = 'MAIN.stringVal';
```
Create device notifications

```
try
    % Register callback function
    tcClient.addlistener('AdsNotification', @OnNotification);
    % Register notifications (the variable names are also used as "userData", which can be evaluated by
    % the callback function)
    hConnect(1) = tcClient.AddDeviceNotification(boolVarName, dataStream, 0, 1,
        AdsTransMode.OnChange, 100, 0, boolVarName);
    hConnect(2) = tcClient.AddDeviceNotification(intVarName, dataStream, 1, 2, AdsTransMode.OnChange, 100, 0, intVarName);
    hConnect(3) = tcClient.AddDeviceNotification(dintVarName, dataStream, 3, 4, AdsTransMode.OnChange, 100, 0, dintVarName);
    hConnect(4) = tcClient.AddDeviceNotification(sintVarName, dataStream, 7, 1, AdsTransMode.OnChange, 100, 0, sintVarName);
    hConnect(5) = tcClient.AddDeviceNotification(lrealVarName, dataStream, 8, 8, AdsTransMode.OnChange, 100, 0, lrealVarName);
    hConnect(6) = tcClient.AddDeviceNotification(realVarName, dataStream, 16, 4, AdsTransMode.OnChange, 100, 0, realVarName);
    hConnect(7) = tcClient.AddDeviceNotification(stringVarName, dataStream, 20, 11, AdsTransMode.OnChange, 100, 0, stringVarName);
    % Listen to ADS notifications for 20 seconds
    pause(20);
    catch err
        msgbox(err.message, 'Error reading array via ADS', 'error');
        disp(['Error registering ADS notifications: ' err.message]);
    end

Deregister ADS notifications

for idx = 1:length(hConnect)
    tcClient.DeleteDeviceNotification(hConnect(idx));
end

Close connection

tcClient.Dispose();

Callback function “OnNotification”

```
function OnNotification(sender, e)

Set data stream offset

e.DataStream.Position = e.Offset;

The element “UserData” is used to transfer the variable name:

valuenname = char(e.UserData);

Read variables from the MATLAB workspace

hConnect = evalin('base', 'hConnect');
binRead = evalin('base', 'binRead');

Variable assignment based on the handle, and conversion of the read value to a string

```
if ( e.NotificationHandle == hConnect(1))
    strValue = num2str(binRead.ReadBoolean());
elseif( e.NotificationHandle == hConnect(2) )
    strValue = num2str(binRead.ReadInt16());
elseif( e.NotificationHandle == hConnect(3) )
    strValue = num2str(binRead.ReadInt32());
elseif( e.NotificationHandle == hConnect(4) )
    strValue = num2str(binRead.ReadSByte());
elseif( e.NotificationHandle == hConnect(5) )
    strValue = num2str(binRead.ReadDouble());
elseif( e.NotificationHandle == hConnect(6) )
    strValue = num2str(binRead.ReadSingle());
elseif( e.NotificationHandle == hConnect(7) )
    strValue = char(System.String(binRead.ReadChars(11)));
end
```
Output of the received value

disp(sprintf('Notification received at %04d-%02d-%02d %02d:%02d:%02.3f:	 %s = 	 %s',clock(), valuename, strValue));

PLC program

PROGRAM MAIN
VAR
boolVal : BOOL;
intVal : INT := 222;
dintVal : DINT;
sintVal : SINT;
lrealVal : LREAL := 22.2;
realVal : REAL;
stringVal : STRING(10);
END_VAR
dintVal := dintVal+1;
5 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 examples for use from C# and PowerShell (Automation Interface Configuration).

In order to simplify the entry from MATLAB® for you, you can find below an example implementation for MATLAB® on the basis of a MATLAB class, which you can use, modify and expand.

5.1 Sample: Tc3AutomationInterface

Overview

The sample code consists of two m-files

- `Tc3AutomationInterface.m`: MATLAB class that implements several frequently used methods
- `Tc3AutomationInterface_Testbench.m`: MATLAB script that calls the MATLAB class as an example.

The sample code additionally contains the `TctComMessageFilter.dll`. You can download the sample here: https://infosys.beckhoff.com/content/1033/tc3_matlab_overview/Resources/zip/5776206091.zip

The MATLAB script

The MATLAB script provides an example 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\TE1400\Target for MATLAB®/Simulink®`. Then copy the file folder from the directory `\TE1400\Sample_TemperatureController\PrecompiledTcComModules\Actual TwinCAT versions\` into the publish directory.

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

The MATLAB class

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.

Constructor

```
function this = Tc3AutomationInterface
```
The constructor loads all necessary assemblies and, if successful, sets the AssembliesLoaded property to TRUE. The loaded assemblies are

- TwinCAT.Ads: ADS library, e.g. for reading and changing the XAR state.
- System.Xml: Library for parsing XML files.

Selected methods of the class

function TcComObject = CreateTcCOM(this, Modelname)

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

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

CreateTcCOM creates a new instance of a TcCOM

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

set properties: TcCOM

see also:
Beckhoff Infosys
```

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

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

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.