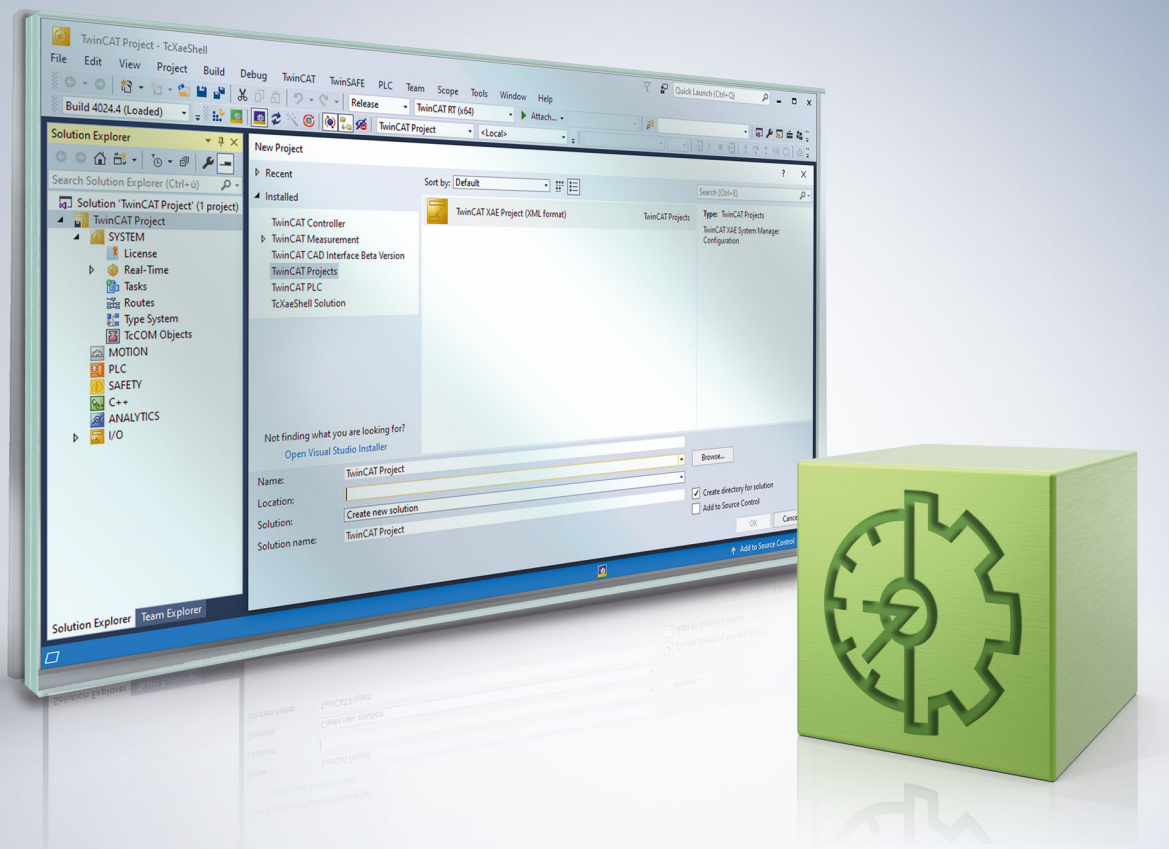
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | Automation Interface



2026-03-30 | Version: 1.6.1

Table of Contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
The documentation and the following notes and explanations must be complied with when installing and commissioning the components.
The trained specialists must always use the current valid documentation.

The trained specialists must ensure that the application and use of the products described is in line with all safety requirements, including all relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been compiled 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 notice.
Claims to modify products that have already been supplied may not be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar®, and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of the designations or trademarks contained in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

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

Copyright

Third-party trademarks

Trademarks of third parties may be used in this documentation. You can find the trademark notices here: https://www.beckhoff.com/trademarks.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

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

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

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.

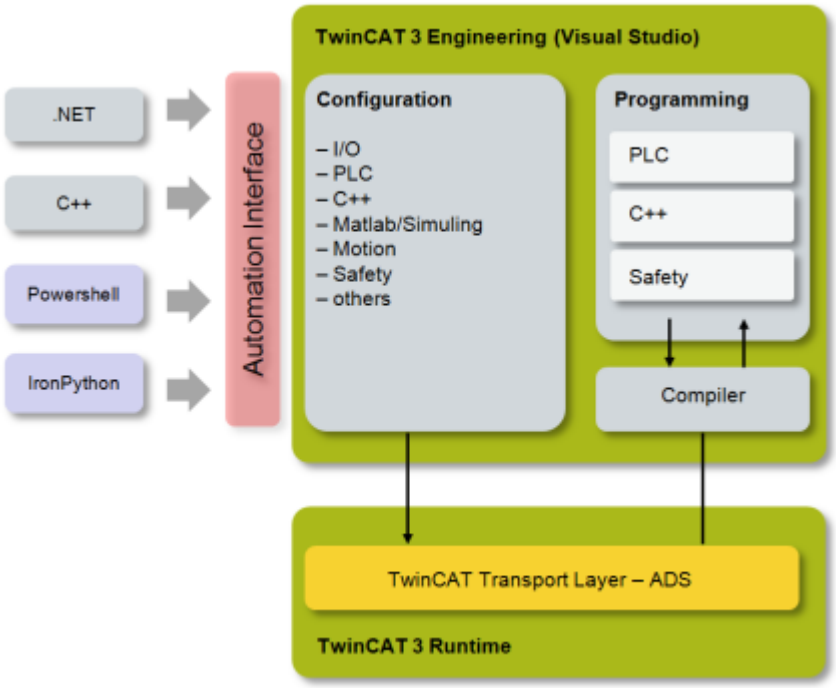
Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they 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

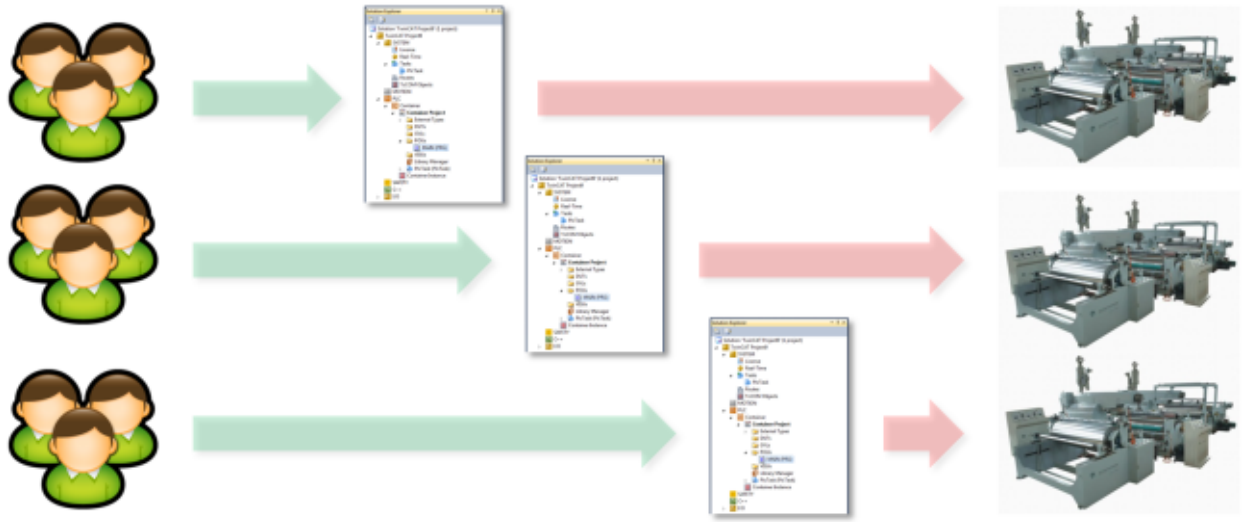
2.1 Product description

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 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. These Automation Interfaces are linked to the Visual Studio automation model, a Visual Studio extended with TwinCAT 3 functions.

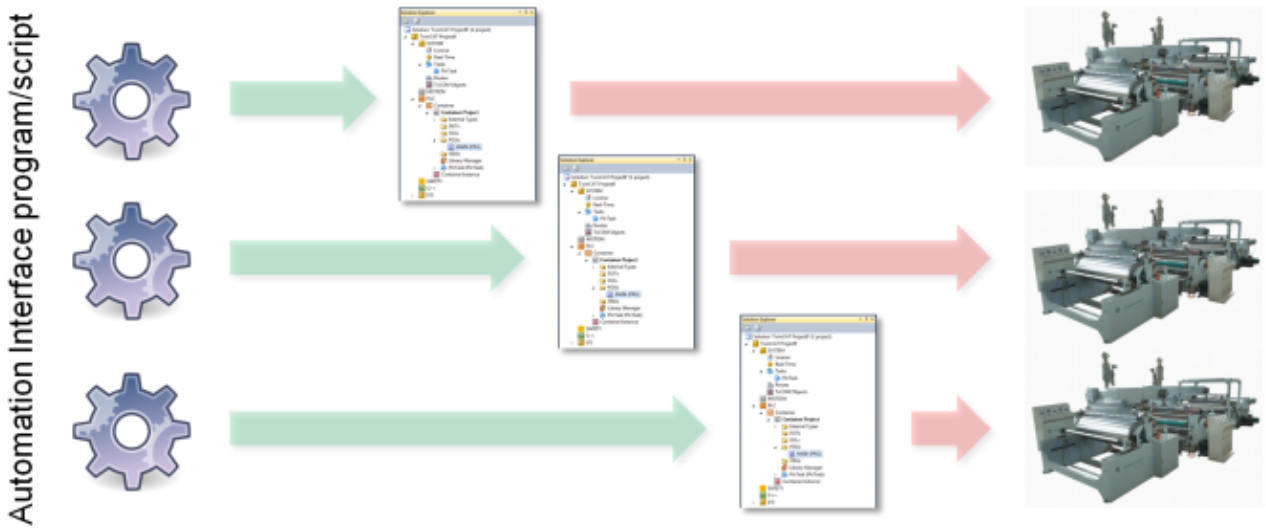


The TwinCAT Automation Interface enables an efficient development process by offering customers the possibility to automate the configuration of a comprehensive TwinCAT solution.

Previously, in the traditional engineering mindset, a machine configuration had to be manually adapted to each new project or even created from scratch, which not only meant a huge development effort associated with high costs, but was also accompanied by a considerable susceptibility to errors caused by human intervention.



Thanks to the TwinCAT Automation Interface, the process of adapting TwinCAT configurations to a new environment or even creating completely new TwinCAT configurations according to customer requirements can be automated.



The reader should now turn to the following topics:

Basics

Topic	Description
Creating/Loading TwinCAT XAE configurations [▶ 19]	Describes how to create or open a TwinCAT configuration
Navigating TwinCAT XAE [▶ 22]	Describes how to navigate through a TwinCAT configuration
Custom tree item parameters [▶ 24]	Describes how to access custom parameters of an item. This is important to access configuration parameters of a TwinCAT tree item.
Implementing a COM message filter [▶ 25]	Describes how to implement an own COM message filter to circumvent rejected COM calls

Best practice

Topic	Description
Creating and handling PLC projects [▶ 53]	Describes how to handle PLC projects
Creating and handling PLC POUs [▶ 62]	Describes how to handle PLC objects/code
Creating and handling PLC Libraries [▶ 58]	Describes how to handle PLC libraries, repositories and placeholder
Creating and handling MOTION projects [▶ 107]	Describes how to create TwinCAT Motion projects (NC-Task, Axes, ...)
Creating and handling EtherCAT devices [▶ 67]	Describes how to create EtherCAT devices and connect them to an EtherCAT topology
Creating and handling TwinCAT Measurement [▶ 99]	Describes how to handle TwinCAT Measurement projects.
Creating and handling TcCOM modules [▶ 90]	Describes how to handle TcCOM modules.
Using templates	Describes the process of template generation and template usage.
Creating and handling network variables [▶ 73]	Describes how to create network variables (publisher/subscriber variables)
Creating and handling Tasks [▶ 41]	Describes how to create Tasks and link them with other objects (PLC-Projects, ...)
From offline to online configurations [▶ 49]	Some IO devices need physical address information before the configuration can be activated. This article explains how to retrieve and set this information.
Accessing the Error List window of Visual Studio [▶ 36]	The Error List can be very helpful for debugging and diagnostic purposes
Accessing window tabs in Visual Studio [▶ 35]	Describes how to get access to Visual Studio windows.
Handling different versions of Visual Studio [▶ 29]	Describes how different Versions of Visual Studio can be used for Automation Interface
Attaching to running Visual Studio instances [▶ 33]	Demonstrates how you can attach to existing (already running) Visual Studio instances to use Automation Interface
Setting TwinCAT target platform [▶ 33]	Describes how to set the TwinCAT target platform for compilation.

Additionally, this documentation also includes a full API reference [▶ 111] of all interfaces. The How to and Sample sections offer a free composition of script code fragments, configuration steps and demo projects. They also contain an unsorted and growing list of "real-world" samples.

Also see about this

- Using Templates [▶ 43]

2.2 Version overview

The following table gives an overview about the available features of the Automation Interface related to TwinCAT 2.11, TwinCAT 3.0, TwinCAT 3.1 and a look-out to future TwinCAT versions which may be subject to change.

Feature	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Future versions
General settings				
Importing configuration templates	✓	✓	✓	✓
TwinCAT System Service handling (Run-/Config-mode)	✓	✓	✓	✓
Load/Save/Create/ Activate configurations	✓	✓	✓	✓
Support for remote TwinCAT targets	✓	✓	✓	✓
Configuring tasks with process image	✓	✓	✓	✓
Configuring tasks without process image	-	-	✓	✓
Multicore support for tasks	-	✓	✓	✓
Handling of TwinCAT licenses	-	-	-	✓
Route management				
Adding/Removing ADS routes	✓	✓	✓	✓
Broadcast Search	✓	✓	✓	✓
I/O				
Scanning for online devices	✓	✓	✓	✓
Adding/removing devices, boxes and terminals	✓	✓	✓	✓
Parameterization of devices, boxes and terminals	✓	✓	✓	✓
EtherCAT topologies	✓	✓	✓	✓
Network variables	✓	✓	✓	✓
PLC				
Mapping of variables, e.g. with I/Os or axes	✓	✓	✓	✓
Adding/removing PLC projects	✓	✓	✓	✓
Adding/removing PLC POUs, DUTs, GVLs	-	-	✓	✓

Feature	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Future versions
Getting/setting PLC code of POUs, DUTs, GVLs	-	-	✓	✓
Adding/removing PLC libraries	-	-	✓	✓
Adding/removing PLC placeholders	-	-	✓	✓
Adding/removing PLC repositories	-	-	✓	✓
Adding/removing PLC libaries to repositories	-	-	✓	✓
Saving PLC projects as a PLC library	-	-	✓	✓
Compiler and error output handling	-	-	✓	✓
PLCopen XML import/export	-	-	✓	✓
Programing language: Structured Text (ST)	-	-	✓ 2	✓ 2
Programing language: Sequential function chart (SFC)	-	-	✓ 1	✓ 1
C++				
Adding/Removing C++ project templates	-	-	-	✓
Compiler and error output handling	-	-	-	✓
Motion				
Adding/Removing NC-Tasks	-	-	✓	✓
Adding/Removing axes	-	-	✓	✓
Parameterization of axes settings	-	-	✓ 3	✓ 3
Mapping of variables, e.g. with PLC	-	-	✓	✓
TcCOM modules				
Adding/Removing TcCOM modules	-	-	✓	✓
Parameterization of TcCOM modules	-	-	✓	✓
Measurement				

Feature	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Future versions
Adding/Removing TwinCAT Measurement projects	-	-	✓	✓
Adding/Removing charts	-	-	✓	✓
Adding/Removing axes	-	-	✓	✓
Adding/Removing channels	-	-	✓	✓
Parameterization of charts, axes and channels	-	-	✓	✓
Starting/Stopping records	-	-	✓	✓

Notes	
1	possibility to implement via PLCopen XML
2	possibility to implement source code either in clear-text or PLCopen XML
3	with limitations. Some settings are stored in a binary format and cannot be edited.

2.3 Frequently Asked Questions

- **What is TwinCAT Automation Interface?**

TwinCAT Automation Interface is an interface to access the configuration of TwinCAT from an external application. This enables customers to automate the configuration of TwinCAT.

- **Can I create an offline TwinCAT configuration (without any attached devices)?**

Yes. You can create the TwinCAT configuration offline by manually attaching (without "Scan Devices) all devices and then providing online values, e.g. addresses, later after all devices have been attached. Please see our samples page for more information. There is also an How-To sample [▶ 89] which shows you how to provide addressing information for pre-configured I/O devices.

- **Which programming and scripting languages are supported?**

Every programming or scripting language that supports the COM object model is supported. Please see our system requirements [▶ 17] page for more information.

- **Which TwinCAT features are accessible via Automation Interface?**

Please see our version overview [▶ 12] page for more information about which TwinCAT features are accessible via Automation Interface.

- **What if I don't find an appropriate programming method or property for a specific setting?**

If you don't find an appropriate Automation Interface method or property for a specific setting, you may use the Import/Export feature of TwinCAT to read/write this setting. Please refer to our article about custom tree item parameters [▶ 24] for more information.

- **Can I automate the configuration of TwinCAT PLC?**

Yes. This feature will be available with TwinCAT 3.1. Please refer to our version overview [▶ 12] page for more information.

- **Can I execute Automation Interface code on TwinCAT XAR (Runtime-only) computers?**

No. To execute Automation Interface code, TwinCAT XAE (Engineering) is needed, because Automation Interface directly accesses the Visual Studio COM object to communicate with the TwinCAT configuration. However, you can use a TwinCAT XAE computer to remotely connect to a TwinCAT runtime and configure it.

- **When should I use ADS and when Automation Interface?**

This is a question which cannot be answered easily because it heavily depends on what you would like to achieve. TwinCAT Automation Interface has been designed primarily to help customers to automate the configuration of TwinCAT. If you want to cyclically read/write values to IOs or PLC variables, our ADS APIs are probably better suited.

- **Do I need to modify my Automation Interface code if I switch languages in TwinCAT XAE, e.g. from English to German?**

All TwinCAT XAE items that are language dependent (Devices, Boxes, Axes, Channels, ...) can either be accessed via the currently set XAE language or via their english name. For example, if the XAE language is changed from English to German, the term "Channel" will be displayed in XAE as "Kanal" but is still available under the name "Channel" via Automation Interface. To be fully compatible, we recommend to build your Automation Interface code based on english terminology.

Please note: This feature comes with TwinCAT 3.x only! Systems based on TwinCAT 2.x are not language independent!

- **I'm a machine builder and use a TwinCAT configuration template for all machine types and only enable/disable certain I/Os. Can I also do that with Automation Interface?**

Yes. There is an How-To sample [▶ 90] which shows you exactly how to do that.

- **Can I also create ADS routes or execute a Broadcast Search?**

Yes. Please see our samples and How-To pages for more information.

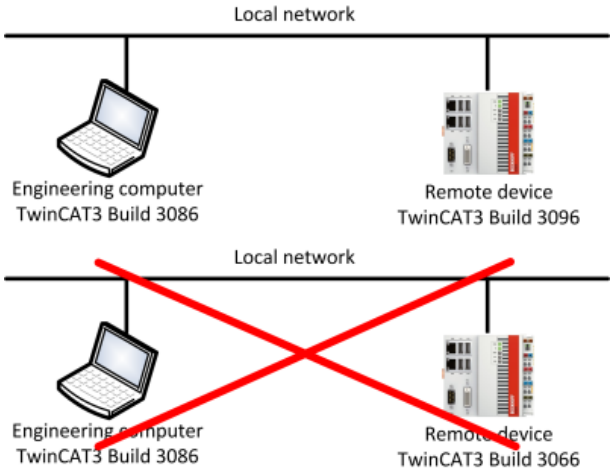
3 Installation

3.1 System requirements

The following chapter lists all hardware and software requirements for the TwinCAT Automation Interface and gives some recommendations for programming and scripting languages.

Hardware and Software

TwinCAT Automation Interface will be automatically installed by TwinCAT setup. Therefore it needs the same hardware and software system requirements as TwinCAT System Manager / TwinCAT 3 XAE (Engineering). When using the Automation Interface to configure a remote TwinCAT device, it is important that the remote version of TwinCAT equals or is higher than on the engineering computer.



Please note that you can execute Automation Interface scripts on 32-bit and 64-bit platforms, however you need to make sure that your program/script has been compiled for and runs in 32-bit mode.

Recommended programming languages

The following programming languages are recommended to use with TwinCAT Automation Interface:

- .NET languages, such as C# or Visual Basic .NET

Please note: Although C++ implementations will also work, we highly recommend using one of the languages above because of their easy and straight-forward programming concepts regarding COM. Most of the sample code in this documentation is based on C#.

Recommended scripting languages

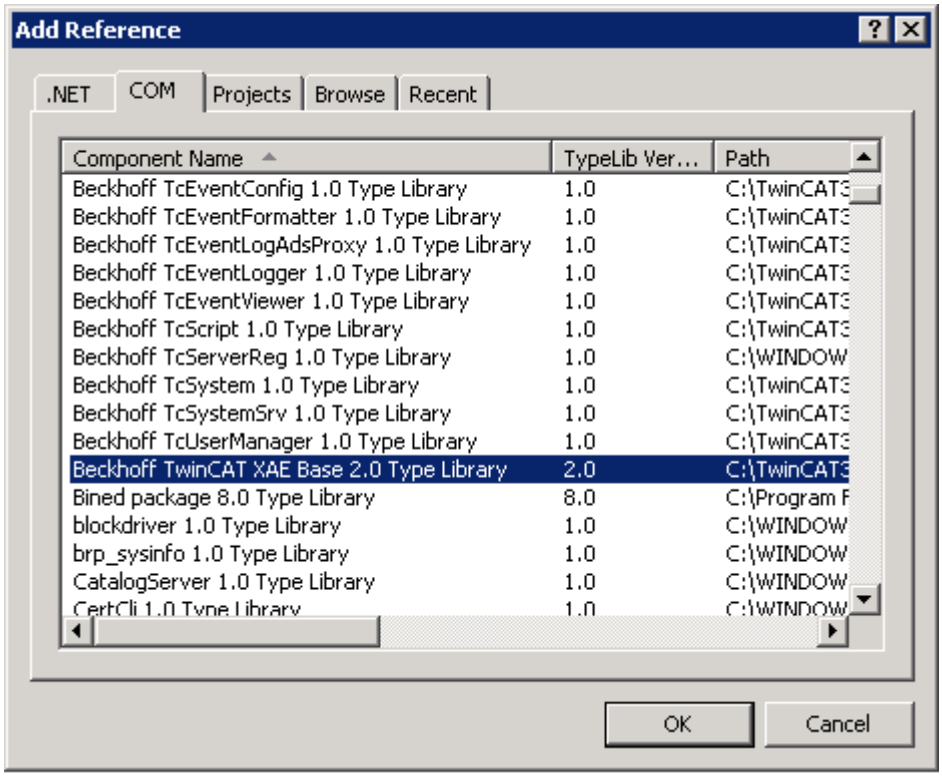
Although every scripting language with COM support may be used to access TwinCAT Automation Interface, we recommend to use the Windows Powershell as it provides the best level of integration between Operating System and application. Please note that at least TwinCAT 3.1 Build 4020.0 is required to use dynamic languages like the Windows Powershell.

3.2 Installation

All files needed for the TwinCAT Automation Interface will be installed automatically during TwinCAT setup. As mentioned in the introduction, the Automation Interface communicates with TwinCAT via COM. All needed COM objects are configured automatically so that COM-capable programming and scripting languages can access these objects.

Using the Automation Interface within a .NET application (C#, VB.NET, …)

To access the Automation Interface from a .NET application, you must add a reference to the corresponding **Beckhoff TwinCAT XAE Base** COM object (depending on your TwinCAT version, see table below) in the Visual Studio project.



Once the reference has been added, you can access the COM object via the **TCatSysManagerLib** namespace.

Please read the article Accessing TwinCAT configuration [▶ 19] where all further steps are explained in detail.

Using the Automation Interface within scripting languages

TwinCAT Automation Interface can also be used with COM-capable scripting languages, for example Windows PowerShell or IronPython. As scripting languages are being interpreted at runtime and not compiled, they always have access to all currently registered COM objects in the operating system. Therefore a reference is not needed.

Please proceed with the article Accessing TwinCAT configuration [▶ 19] which explains all further steps in more detail.

Type library versions

During the TwinCAT product lifecycle, the type library mentioned above may be delivered in different versions because of added functionalities and/or big TwinCAT versions steps. The table below gives an overview about all different type library versions.

Type library name	Type library version	TwinCAT version
Beckhoff TCatSysManager 1.1 Type Library	1.1	TwinCAT 2.11
Beckhoff TwinCAT XAE Base 2.0 Type Library	2.0	TwinCAT 3.0
Beckhoff TwinCAT XAE Base 2.1 Type Library	2.1	TwinCAT 3.1
Beckhoff TwinCAT XAE Base 3.1 Type Library	3.1	TwinCAT 3.1 Build 4020.0 and above

4 Configuration

4.1 Quickstart

Because the TwinCAT Automation Interface provides a lot of possibility, it might sometimes be hard to understand where to start. This quickstart provides a step-by-step introduction into the TwinCAT Automation Interface and the different articles in this documentation.

We recommend to read the following in-depth articles:

Step	Article	Content
1	Visual Studio ProgIDs [▶ 29]	Describes how different Visual Studio versions can be accessed via the Visual Studio API
2	Accessing TwinCAT configuration [▶ 19]	Describes how to use the Automation Interface in order to create a new TwinCAT project. It also covers the co-existence of Visual Studio API and TwinCAT Automation Interface and how they correlate with each other.
3	Navigating TwinCAT configuration [▶ 22]	Describes how to navigate through an opened TwinCAT configuration by using different Automation Interface functionalities.
4	The necessity of a COM MessageFilter [▶ 25]	Describes why every Automation Interface application should implement a custom COM MessageFilter.
5	Silent Mode [▶ 30]	Describes how to turn the Automation Interface “silent”

After these basic articles, the Best Practice [▶ 30] articles may be consulted for different topics like PLC or I/O access via the Automation Interface.

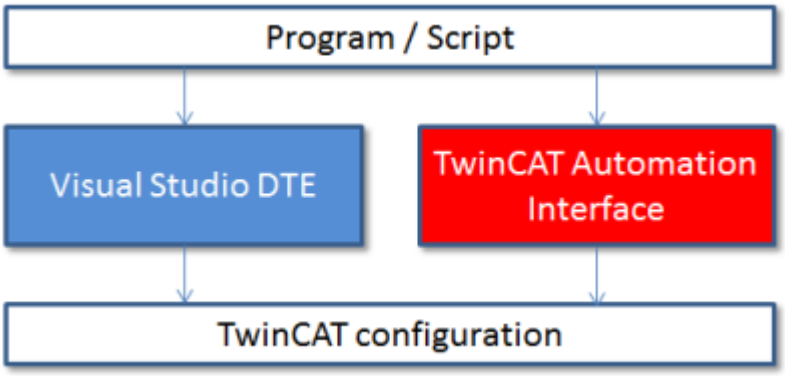
4.2 Basics

4.2.1 Accessing TwinCAT configuration

This chapter describes how to create and access a TwinCAT XAE configuration project via Automation interface. The objective of this creation process is to get access to a TwinCAT XAE project (formerly known as a TwinCAT System Manager configuration). The new TwinCAT XAE project is more sophisticated than the TwinCAT System Manager configuration known from TwinCAT2, which implies a slight concept change of project / configuration handling. TwinCAT 3 supports an extra hierarchical level combining several configurations into a Visual Studio solution container. This can for example be used to organize the configurations of distributed resources into a solution or for packaging HMI projects together with the system configuration. The solution is able to bind together all types of Visual Studio and/or TwinCAT XAE projects. When using the TwinCAT XAE Automation inferface, this means an extra level of possiblities.

Basic information

TwinCAT 3 has been fully integrated into Microsoft Visual Studio® to provide users with a standardized and flexible editor for creating and managing TwinCAT projects. When creating and/or accessing a TwinCAT configuration, Microsoft Visual Studio® and the TwinCAT Automation Interface can be used in combination. Example: If you want to create a new TwinCAT configuration via the Automation Interface, you must first call methods of the Visual Studio API to create a Visual Studio Solution Container and then add a TwinCAT project using the methods of the TwinCAT Automation Interface. This scenario is covered in some code snippets below.



In addition, the Visual Studio API (the **Visual Studio DTE**) offers developers many other functions, e.g. access to the error output window [▶ 36]. Further information on Visual Studio DTE can be found on the Microsoft MSDN website.

Please note:

- When creating a new TwinCAT project in a Visual Studio solution, you must specify a path to the TwinCAT project template [▶ 25]. Please adjust this path in the code snippets below according to your environment.
- The following code snippets use **dynamic linking** for the Visual Studio DTE objects, which means that the actual type of the object is only determined during the course of the application runtime. If you do not want to use dynamic linking but want to define the data type in advance, you must include the EnvDTE.DTE namespace in your project.

Creating TwinCAT Projects via templates

Please note that you need to add a reference to the COM object **TcatSysManagerLib** and EnvDTE.DTE (Microsoft Developement) in order to be able to use the TwinCAT Automation Interface and the Visual Studio API. The ProgID that is used in the GetTypeFromProdID() method depends on the Visual Studio version that should be used. Please have a look at this [▶ 29] documentation article for more information about the different ProgIDs.

Code Snippet (C#):

```
Type t = System.Type.GetTypeFromProgID("VisualStudio.DTE.10.0");
EnvDTE.DTE dte = System.Activator.CreateInstance(t);

dte.SuppressUI = false;
dte.MainWindow.Visible = true;

if (Directory.Exists(@"C:\Temp\SolutionFolder"))
Directory.Delete(@"C:\Temp\SolutionFolder", true);
Directory.CreateDirectory(@"C:\Temp\SolutionFolder");
Directory.CreateDirectory(@"C:\Temp\SolutionFolder\MySolution1");

dynamic solution = dte.Solution;
solution.Create(@"C:\Temp\SolutionFolder", "MySolution1");
solution.SaveAs(@"C:\Temp\SolutionFolder\MySolution1\MySolution1.sln");

string template = @"C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj"; //path to
project template
dynamic project = solution.AddFromTemplate(template, @"C:\Temp\SolutionFolder\MySolution1",
"MyProject");

ITcSysManager sysManager = project.Object as ITcSysManager;

sysManager.ActivateConfiguration();
sysManager.StartRestartTwinCAT();

project.Save();
solution.SaveAs(@"C:\Temp\SolutionFolder\MySolution1\MySolution1.sln");
```

Code Snippet (Powershell):

You can copy and paste the following code snippet into a textfile and save it as "someName.ps1". After that you can execute it directly via Windows PowerShell.

```
$targetDir = "C:\tmp\TestSolution"
$targetName = "TestSolution.tsp"
$template = "C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj"

$dte = new-object -com VisualStudio.DTE.10.0
$dte.SuppressUI = $false
$dte.MainWindow.Visible = $true

if(test-path $targetDir -pathtype container)
{
Remove-Item $targetDir -Recurse -Force
}

New-Item $targetDir -type directory

$sln = $dte.Solution
$project = $sln.AddFromTemplate($template,$targetDir,$targetName)
$systemManager = $project.Object

$targetNetId = $systemManager.GetTargetNetId()
write-host $targetNetId

$systemManager.ActivateConfiguration()
$systemManager.StartRestartTwinCAT()

$project.Save()
$solutionPath = $targetDir + "\" + $targetName
$sln.SaveAs($solutionPath)
```

Code Snippet (C++):

Within appropriate Header file (e.g the stdafx.h):

```
//the following #import imports EnvDTE based on its LIBID.
#import"libid:80cc9f66-e7d8-4ddd-85b6-d9e6cd0e93e2" version("10.0") lcid("0") raw_interfaces_only
named_guids
// Imports die "Beckhoff TCatSysManager 1.1 Type Library"
#import"libid:3C49D6C3-93DC-11D0-B162-00A0248C244B" version("1.1") lcid("0")
```

Because a known issue within VisualStudio 2010 (SP1), the generated proxy code will not be included into the C++ project. Please use the workaround described in #import Known Issue import Known Issue.

```
#include

using namespace std
using namespace TCatSysManagerLib;
using namespace EnvDTE;

int _tmain(int argc, _TCHAR* argv[])
{
CoInitialize(NULL); // COM initialisieren
cout << "Creating VisualSTudio.DTE.10.0 ...";

// creating a new instance of Visual Studio
CComPtr<_DTE> m_pDTE;
HRESULT hr = m_pDTE.CoCreateInstance(L"VisualStudio.DTE.10.0", 0, CLSCTX_ALL);
if (FAILED(hr)) { cout << " FAILED"; return 1; }
cout << " created." << endl;

// retrieves the EnvDTE.Solution-Objekt
CComPtr<_Solution> pSolution;
m_pDTE->get_Solution(&pSolution);
CComBSTR strSolutionFolder(_T("C:\\SolutionFolder")); // Solution-main directory (has to exist)
CComBSTR strSolutionName(_T("MySolution1"));
CComBSTR strTemplatePath(_T("C:\\TwinCAT\\3.1\\Components\\Base\\PrjTemplate\\TwinCAT
Project.tsproj"));

CComBSTR strSolutionPath; // Solution-Pfad (doesn't exist!)
strSolutionPath=strSolutionFolder;
strSolutionPath.Append(_T("\\"));
strSolutionPath.Append(strSolutionName);
strSolutionPath.Append(_T(".sln"));

// create the solution
hr = pSolution->Create(strSolutionFolder,strSolutionName);
```

```
CComBSTR strProjectPath(strSolutionFolder); // project path
strProjectPath.Append(_T("\\"));
strProjectPath.Append(strSolutionName);
CComBSTR strProjectName = "MyProject"; // project name // create projekt from a template
CComPtr pProject;
hr = pSolution-
>AddFromTemplate(strTemplatePath,strProjectPath,strProjectName,VARIANT_FALSE,&pProject);
// Wenn z.B. Projekt bereits besteht >> error
if (FAILED(hr)) { cout << " Project creation FAILED"; return 1; }
cout << "Project created" << endl;

// define project automation class (here the Coclass TcSysManager)
CComPtr pDispatch;
hr = pProject->get_Object(&pDispatch);

// retrieve ITcSysManager interface
CComPtr pSystemManager;
hr = pDispatch.QueryInterface(&pSystemManager);

// operate with SystemManager interface
CComBSTR netId;
netId = (pSystemManager->GetTargetNetId()).GetBSTR();
cout << "TargetNetId: " << netId << endl;
hr = pSystemManager->ActivateConfiguration();
hr = pSystemManager->StartRestartTwinCAT();

// save project and solution
hr = pProject->Save(CComBSTR());
hr = pSolution->SaveAs(strSolutionPath);
cout << "Succeeded";
return 0;
}
```

import Known Issue:

```
CComPtr<_DTE> m_pDTE;
CLSID clsid;
CLSIDFromProgID(L"VisualStudio.DTE.10.0",&clsid);
CComPtr punk;
HRESULT hr = GetActiveObject(clsid,NULL,&punk); // retrieve actual instance of Visual Studio .NET
m_pDTE = punk;
```

Please note:

ITcSysManager::NewConfiguration [▶ 114], ITcSysManager::OpenConfiguraiton [▶ 114] and ITcSysManager::SaveConfiguration [▶ 115] will create error messages in this case, because the project handling is delegated to the Visual Studio IDE (the Solution and Project instances realized by the DTE object).

4.2.2 Browsing TwinCAT configuration

In a separate article, we have already shown you how to access TwinCAT [▶ 19] via the Visual Studio Automation Model. This reference to TwinCAT is being represented by an object of type ITcSysManager [▶ 112]. From this point on we would like to discuss now how you can navigate through the TwinCAT configuration.

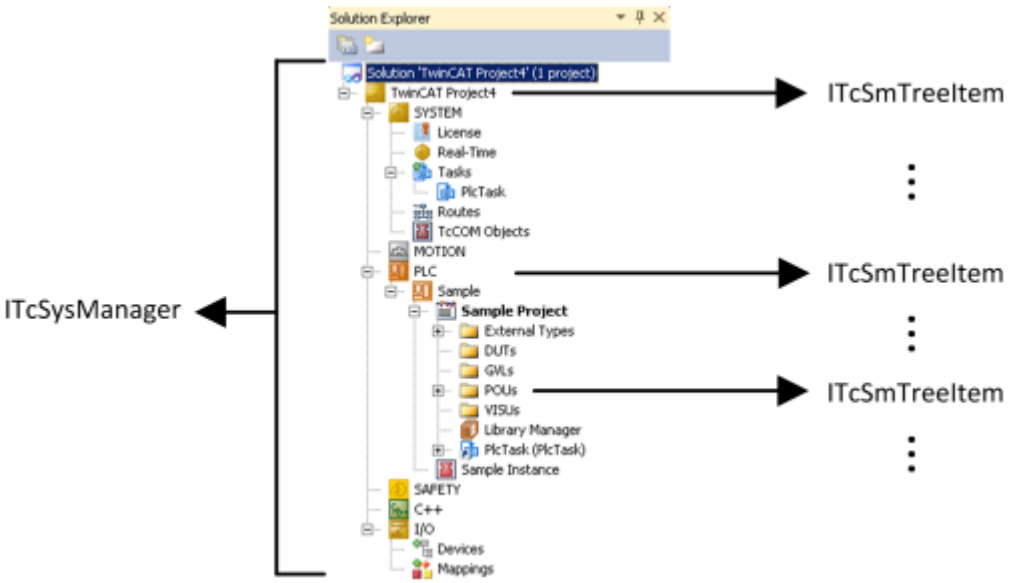
General Information

It is important to understand that all information in TwinCAT is organized in a tree-like structure. In the Automation Interface, each structure node and therefore each element of a TwinCAT configuration is represented by the ITcSmTreeItem [▶ 120] interface.

You can navigate through the TwinCAT data model in different ways, depending on the type of information you want to query.

- **Lookup methods** search for specific Tree Items using specified search criteria, e.g. the path to a Tree Item
- **Iterators** or search functions iterate over a set of queried Tree Items

Both methods are now discussed in the following article.



LookupMethods

The Lookup methods are always working on the whole data model (unfiltered).

- ITcSysManager::LookupTreeItem [▶ 118] determines a tree item with the specified absolute path name.
- ITcSysManager3::LookupTreeItemById [▶ 119] determines a tree item with the specified item type and item Id.
- ITcSmTreeItem::LookupChild [▶ 157] determines a tree item within a subtree specifed by a relative path name.

Each tree item in TwinCAT can be identified by its unique pathname. The pathname of a tree item is based on the hierarchical order of its parent item (its name) and its own name, separated by circumflex accents ('^'). To shorten the pathnames and to avoid language dependencies, the top level tree items have special abbreviations which are listed in ITcSysManager::LookupTreeItem [▶ 118].

Iterators

At the moment, three different types of iteration functions are supported:

- Browsing all tree items (unfiltered)
- Browsing main tree items
- Browsing variables / symbols only

Browsing all tree items (unfiltered)

To browse all tree items in an unfiltered way, the property ITcSmTreeItem [▶ 120]:NewEnum may be used. _NewEnum iterates over all subnodes of the currently referenced ITcSmTreeItem [▶ 120]. This (COM-) property is used by many Programming and Script languages that support the COM Enumerator model (e.g. .NET languages, VB6 or script languages) by a 'foreach' statement. For non-supporting languages like C++ the foreach loop must be implemented manually by using the IEnumVariant-interface.

We recommend to use this way to iterate through child nodes.

Sample (C#):

```
ITcSmTreeItem parent = sysMan.LookupTreeItem("TIID^Device1^EK1100");
foreach(ITcSmTreeItem child in parent)
{
 Console.WriteLine(child.Name);
}
```

Sample (C++):

```
...
#import "C:\TwinCAT3\Components\Base\TCatSysManager.tlb" // imports the System Manager / XAE Base
type library
// uses automatically created auto-pointer (see MSDN documentation of #import command)
...

void CSysManDialog::IterateCollection(TCatSysManagerLib::ITcSmTreeItemPtr parentPtr)
{
IEnumVARIANTPtr spEnum = parentPtr->_NewEnum;
ULONG nReturned = 0;
VARIANT variant[1] = {0};
HRESULT hr = E_UNEXPECTED;

do
{
hr = spEnum->Next(1, &variant[0], &nReturned);
if(FAILED(hr))
break;
for(ULONG i = 0; i < nReturned; ++i)
{
IDispatchPtr dispatchPtr;
IDispatch* pDispatch;
TCatSysManagerLib::ITcSmTreeItemPtr childPtr;
HRESULT hr;
if(variant[0].vt == VT_DISPATCH)
{
TCatSysManagerLib::ITcSmTreeItem* pChild = 0;
dispatchPtr.Attach((variant[0].pdispVal));
hr = dispatchPtr.QueryInterface(__uuidof(TCatSysManagerLib::ITcSmTreeItem),
reinterpret_cast(&pChild));
childPtr.Attach(pChild);
_bstr_t strName = pChild->GetName();
}
}
}
while(hr != S_FALSE); // S_FALSE zeigt Ende der Sammlung an
}
```

Sample (PowerShell):

```
$systemItem = $systemManager.LookupTreeItem("TIRC")
foreach($child in $systemItem)
{
write-host$child.Name
}
```

Browsing Main Tree Items (Filtered)

For browsing only the main childs of the current tree item use the ITcSmTreeItem::ChildCount [▶ 120] and ITcSmTreeItem:Child(n) [▶ 112] pair of properties. These methods only work on the direct childs (non-recursive).

Browsing Variables / Symbols only

To Browse the Variables / Symbols use the ITcSmTreeItem::VarCount(x) [▶ 120] , ITcSmTreeItem::Var(x,n) [▶ 120] pair of properties. A selection of the variable type (input variables or output variables) can be done by parameter.

4.2.3 Custom TreeItem Parameters

The ITcSmTreeItem [▶ 120] interface is supported by every TwinCAT tree item and has a very generic character. To support the specification of all the devices, boxes and terminals, together with many other different types of tree items, all custom parameters of a tree item are accessible via the XML representation of the tree item. This XML-String can be accessed by the method ITcSmTreeItem::ProduceXml [▶ 152] and its counterpart ITcSmTreeItem::ConsumeXml [▶ 153]. This function pair has the same functionality as the "Export XML Description ..." and "Import XML Description ..." commands from the main menu of the TwinCAT IDE (see snapshot below).

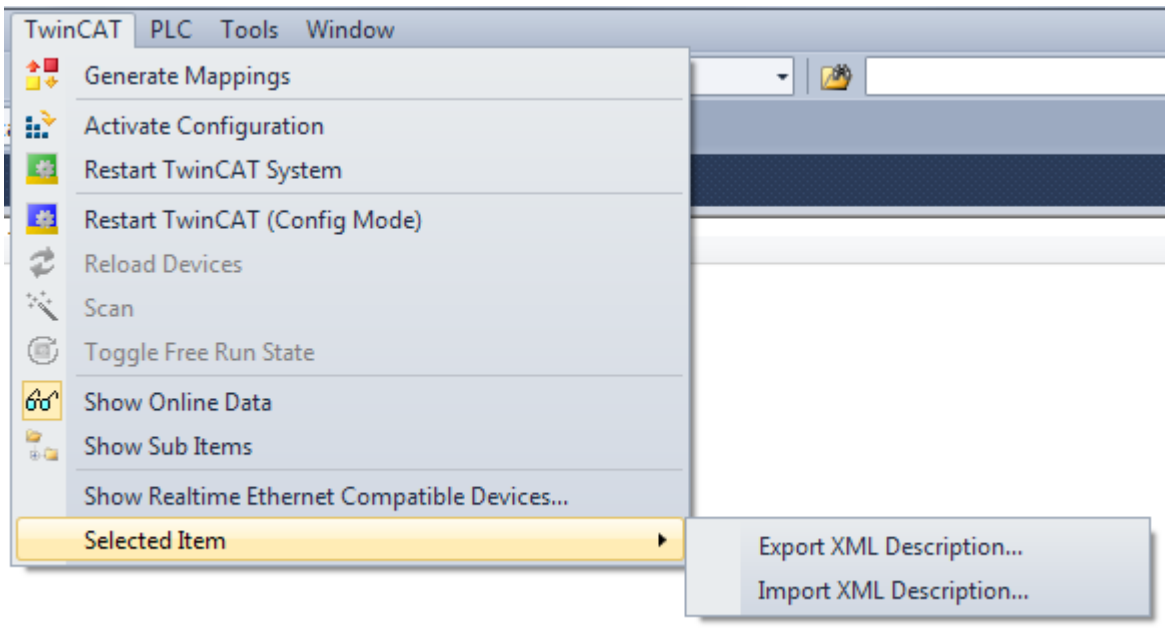


Fig. 1: TcSysMan_AutomationXmlParameters

With this Import/Export functionset, many parts of a script or automation code can be tested and tailored conveniently, before it is developed within the coding language - simply by the process of exporting tree item data, changing its content and re-importing it.

Best practice is to export the XML content first, change its content, importing it within the IDE and then, if everything goes successfully, package it into the programming/script code for handling it via the methods ProduceXml and ConsumeXml.

4.2.4 TwinCAT Project Template

When creating a new TwinCAT solution, you need to specify the path to the TwinCAT Project template - see also our article about Accessing TwinCAT XAE configuration [▶ 19].

Basics

TwinCAT version	Path to TwinCAT project template*
TwinCAT 3.0	C:\TwinCAT\3.0\Components\Base\PrjTemplate\TwinCAT Project.tsp
TwinCAT 3.1	C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj

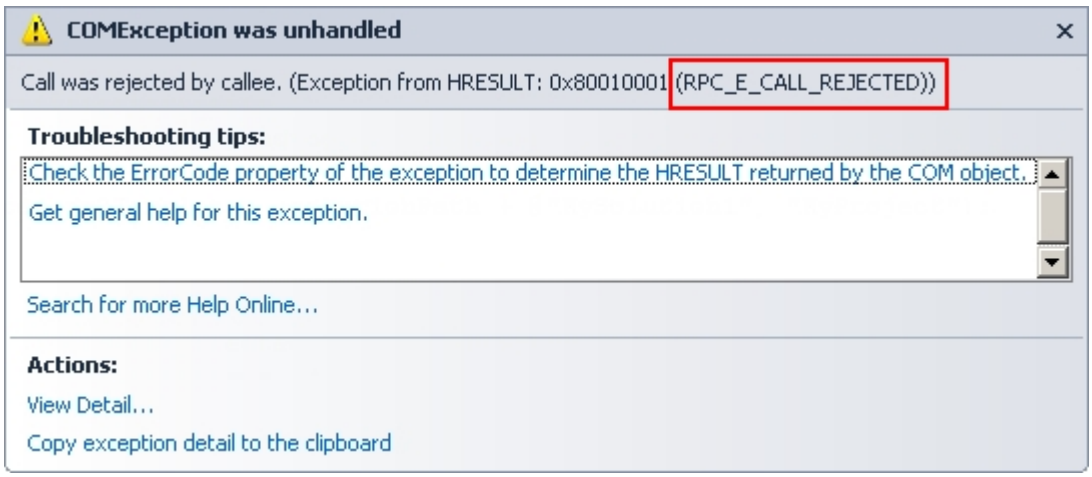
* Please note: The paths mentioned above are based on the default TwinCAT installation directory and may be different if you installed TwinCAT in another folder.

4.2.5 Implementing a COM Message Filter

Message filtering is a mechanism that allows server applications to decide if and when an incoming method call can be safely executed on one of their objects. COM generally does not know the reentrancy requirements of your application and consequently does not filter messages by default. Even though message filtering is not as significant as it was with 16-bit applications because the message queue size is now actually unlimited, you should still implement **message filtering** as a way to resolve blockages. COM will call your implementation of the **IMessageFilter** interface to find out if an application (a COM server) is blocking so that you can respond and handle the situation. For example, when accessing TwinCAT XAE via COM, the Visual Studio instance will reject further COM calls while still executing a previous COM call. As a

result, the client application will issue an RPC_E_CALL_REJECTED error and, without further intervention, will not repeat the call. By writing a user-defined message filter, the programmer has the ability to retry the COM call when the client application receives notification of a denied COM call from the COM server.

The following screenshot shows a typical error output by the Visual Studio COM server when an instance is still busy executing a previous COM call.



To avoid this situation and implement a message filter that responds to this rejected COM call, the application engineer must implement the **IMessageFilter** interface. This interface consists of three methods:

- **HandleIncomingCall(): Provides a single entry point for incoming calls**
- **MessagePending(): Indicates that a message has been received while COM is waiting for a remote call to be answered.**
- **RetryRejectedCall(): Provides the ability to respond to a rejected COM call.**

Note that message filters can only be applied to STA threads and that only one filter can be applied to each thread. Multithreaded apartments, e.g. console applications, cannot have message filters. These applications must run in an STA thread to apply message filtering. See the appendix of this documentation for more information on COM threading.

The following code snippet shows a sample of how the IMessageFilter interface can be used in C#. Note that this code is also used in many samples in our Samples section, and is also available as a separate sample download.

```
[ComImport(), Guid("00000016-0000-0000-C000-000000000046"),
InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)]
interface IOleMessageFilter
{

[PreserveSig]
int HandleInComingCall(int dwCallType, IntPtr hTaskCaller, int dwTickCount, IntPtr lpInterfaceInfo);


[PreserveSig]
int RetryRejectedCall(IntPtr hTaskCallee, int dwTickCount, int dwRejectType);


[PreserveSig]
int MessagePending(IntPtr hTaskCallee, int dwTickCount, int dwPendingType);
}
```

The following class implements this interface and adds two more methods: Register() and Revoke().

```
public class MessageFilter : IOleMessageFilter
{
public static void Register()
{
IOleMessageFilter newFilter = new MessageFilter();
IOleMessageFilter oldFilter = null;
int test = CoRegisterMessageFilter(newFilter, out oldFilter);

if (test != 0)
{
Console.WriteLine(string.Format("CoRegisterMessageFilter failed with error : {0}", test));
```

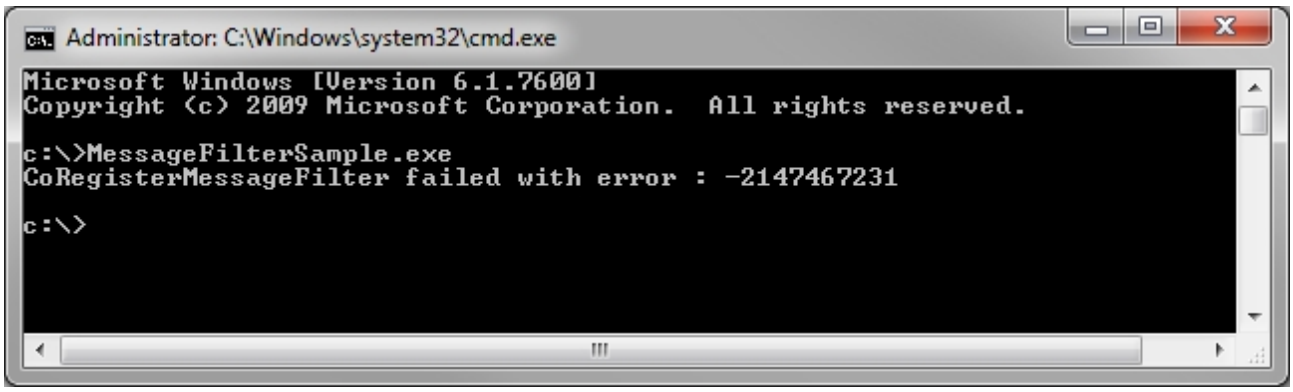
```
}
}


public static void Revoke()
{
IOleMessageFilter oldFilter = null;
int test = CoRegisterMessageFilter(null, out oldFilter);
}


int IOleMessageFilter.HandleInComingCall(int dwCallType, System.IntPtr hTaskCaller, int dwTickCount,
System.IntPtr lpInterfaceInfo)
{
//returns the flag SERVERCALL_ISHANDLED.
return 0;
}


int IOleMessageFilter.RetryRejectedCall(System.IntPtr hTaskCallee, int dwTickCount, int
dwRejectType)
{
// Thread call was refused, try again.
if (dwRejectType == 2)
// flag = SERVERCALL_RETRYLATER.
{
// retry thread call at once, if return value >=0 &
// <100.
return 99;
}
return -1;
}


int IOleMessageFilter.MessagePending(System.IntPtr hTaskCallee, int dwTickCount, int dwPendingType)
{
//return flag PENDINGMSG_WAITDEFPROCESS.
return 2;
}

// implement IOleMessageFilter interface.
[DllImport("Ole32.dll")]
private static extern int CoRegisterMessageFilter(IOleMessageFilter newFilter, out IOleMessageFilter
oldFilter);
```

An application engineer now only needs to call the Register() and Revoke() methods from another class to initialize and discard the MessageFilter. The result is that rejected COM calls are repeated as specified in the RetryRejectedCall() method.

The following code snippet shows how to call these methods in a console application written in C#. As mentioned above, console applications run in an MTA thread by default. For this reason, the Main() method must be configured to run in an STA apartment so that the message filter can be applied.

```
[STAThread]
static void Main(string[] args)
{
MessageFilter.Register();

/* ============================================================
* place COM calls for the Automation Interface here
* ...
* ...
* ============================================================ */

MessageFilter.Revoke();
}
```

If you try to apply a message filter to an application running in the MTA apartment, the following error is issued when trying to execute the CoRegisterMessageFilter() method during runtime:



For more information about the different COM threading models, see the MSDN article Understanding and Using COM Threading models. For more detailed information about the IMessageFilter interface, see the MSDN documentation regarding IMessageFilter.

The following code snippet shows how to implement a COM MessageFilter for Windows PowerShell by including it as a .NET type (C#) in PowerShell.

Code snippet (PowerShell):

```
AddMessageFilterClass('') # Call function
[EnvDteUtils.MessageFilter]::Register() # Call static Register Filter Method

$dte = New-Object -COMObject TcXaeShell.DTE.15.0
$dte.SuppressUI = $false
$dte.MainWindow.Visible = $true
$solution = $dte.Solution
# do stuff
$dte.Quit()

[EnvDTEUtils.MessageFilter]::Revoke()


function AddMessageFilterClass
{
$source = @'
namespace EnvDteUtils
{
using System;
using System.Runtime.InteropServices;

public class MessageFilter : IOleMessageFilter
{
public static void Register()
{
IOleMessageFilter newFilter = new MessageFilter();
IOleMessageFilter oldFilter = null;
CoRegisterMessageFilter(newFilter, out oldFilter);
}

public static void Revoke()
{
IOleMessageFilter oldFilter = null;
CoRegisterMessageFilter(null, out oldFilter);
}

int IOleMessageFilter.HandleInComingCall(int dwCallType, System.IntPtr hTaskCaller, int dwTickCount,
System.IntPtr lpInterfaceInfo)
{
return 0;
}

int IOleMessageFilter.RetryRejectedCall(System.IntPtr hTaskCallee, int dwTickCount, int
dwRejectType)
{
if (dwRejectType == 2)
{
return 99;
}
return -1;
}
```

```
int IOleMessageFilter.MessagePending(System.IntPtr hTaskCallee, int dwTickCount, int dwPendingType)
{
return 2;
}

[DllImport("Ole32.dll")]
private static extern int CoRegisterMessageFilter(IOleMessageFilter newFilter, out IOleMessageFilter
oldFilter);
}

[ComImport(), Guid("00000016-0000-0000-C000-000000000046"),
InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)]
interface IOleMessageFilter
{
[PreserveSig]
int HandleInComingCall(int dwCallType, IntPtr hTaskCaller, int dwTickCount, IntPtr lpInterfaceInfo);

[PreserveSig]
int RetryRejectedCall(IntPtr hTaskCallee, int dwTickCount, int dwRejectType);

[PreserveSig]
int MessagePending(IntPtr hTaskCallee, int dwTickCount, int dwPendingType);
}
}
'@
Add-Type -TypeDefinition $source
}
```

4.2.6 Handling different Visual Studio versions

This article explains how to use the TwinCAT Automation Interface with different versions of Visual Studio, e.g. Visual Studio 2017 and 2019, if you have both or more versions installed at the same time.

Visual Studio program ID

Due to the installation of TwinCAT 3 Engineering in Visual Studio, you must create an instance of Visual Studio (DTE object) in order to access the Automation Interface. To create this instance, the ProgID is required, which specifies which version of Visual Studio is to be used. Different versions of Visual Studio or TcXaeShell can be installed on a system, which can be distinguished by the ProgID. The ProgID is located in the Windows registry under HKEY_CLASSES_ROOT and has the following format: VisualStudio.DTE.X.Y.

The following table shows the ProgIDs of the currently supported Visual Studio versions:

Version	ProgID
Visual Studio 2017	VisualStudio.DTE.15.0
Visual Studio 2019	VisualStudio.DTE.16.0
Visual Studio 2022	VisualStudio.DTE.17.0
TwinCAT XAE Shell	TcXaeShell.DTE.15.0
TwinCAT XAE Shell 64-bit	TcXaeShell.DTE.17.0

Specifying the Visual Studio version in the Automation Interface code

To specify the Visual Studio version in the Automation Interface code, you must use the ProgID as a parameter in the GetTypeDromProgID() method when creating the DTE object.

Code snippet (C#):

```
Type t = System.Type.GetTypeFromProgID("VisualStudio.DTE.15.0");
EnvDTE.DTE dte = (EnvDTE.DTE) System.Activator.CreateInstance(t);
```

Code snippet (PowerShell):

```
$dte = new-object -com VisualStudio.DTE.15.0
```

4.2.7 Silent Mode

Sometimes an Automation Interface script or program should operate silently, which means without any message boxes or other visible interruptions. Although the Visual Studio DTE command “dte.Visible = true/false” may seem sufficient in most use cases, the TwinCAT Automation Interface introduces a new Silent Mode switch, which is available since TwinCAT 3.1 Build 4020.0 and above.

This new switch can be activated as follows.

Code snippet (C#):

```
var settings = dte.GetObject("TcAutomationSettings");
settings.SilentMode = true;
```

Code snippet (Powershell):

```
$settings = $dte.GetObject("TcAutomationSettings")
$settings.SilentMode = $true
```

This will suppress message box dialogs during usage of the TwinCAT Automation Interface.

4.3 Best practice

4.3.1 Visual Studio

4.3.1.1 Select projects for build process

The Visual Studio API provides all mechanisms required to select projects for a Solution Configuration. The necessary methods are part of the SolutionBuild2 class of the EnvDTE namespace. The following example demonstrates how to use the method BuildProject() from that class.

Code Snippet (Powershell):

```
$sln = $dte.Solution
$prj = $dte.Projects.Item(1) #SysMan Project
$sysManProjectName = $prj.FullName
$plcPrjProjectName = PathToPlcProjFile
$sln.SolutionBuild.BuildProject("Release|TwinCAT RT (x64)",$sysManProjectName,$true)
$sln.SolutionBuild.BuildProject("Release|TwinCAT RT (x64)",$plcPrjProjectName,$true)
```

The placeholder “pathToPlcProjFile” represents the full path to a *.plcproj file which represents a TwinCAT 3 PLC project.

4.3.1.2 Accessing TeamFoundationServer source control

This chapter provides an overview of how to access a Microsoft Team Foundation Server (TFS) programmatically using the corresponding DLLs provided by Microsoft Visual Studio®. This documentation was written for a better overview and to simplify the first steps when you try to combine the TwinCAT Automation Interface code with TFS DLLs. For more detailed documentation on the TFS API, we recommend that you refer to the corresponding MSDN articles.

This documentation deals with the following topics:

- Connecting to a TFS server
- Connecting to TeamProjects
- Working with workspaces
- Creating working folders
- Getting the latest versions
- Receiving a list of upcoming changes
- Checking in and out
- Undoing a process

The following TFS API DLLs are used within this documentation:

- Microsoft.TeamFoundation.Client
- Microsoft.TeamFoundation.VersionControl.Client

Connecting to a TFS server

The Microsoft TFS API allows you to connect your application, which contains the TwinCAT code of the Automation Interface, to your development repository on a remote server. The connection requires a string that describes the address, port and composition of your remote server, e.g. http://your-tfs:8080/tfs/DefaultCollection.

Before the connection, you must define an instance of the Uri class and the TfsConfigurationServer class. Assign the remote server link to the configuration server:

Code snippet (C#):

```
string tfsServerUrl = "http://your-server:8080/tfs/DefaultCollection";
Uri configurationServerUri = new Uri(tfsServerUrl);
TfsTeamProjectCollection teamProjects =
TfsTeamProjectCollectionFactory.GetTeamProjectCollection(configurationServerUri);
VersionControlServer verControlServer = teamProjects.GetService<VersionControlServer>();
```

Connecting to TeamProjects

You can get a list of team projects or a specific team project on the server by executing the following code snippets:

Code snippet (C#):

```
// Get all Team projects
TeamProject[] allProjects = verControlServer.GetAllTeamProjects(true);

// Get specific Team Project
TeamProject project = verControlServer.TryGetTeamProject("TeamProjectName");
```

Working with workspaces

Team Foundation works with workspaces that contain mappings to working folders. These mappings link server-side folders with local folders on your hard disk. In addition, the name of the owner and the name of your computer are saved as part of the workspace name. The workspace is a must before performing a version control task as it stores the information about files, versions and the list of pending changes.

To start with TFS Client, the API provides a *Workspace* class that stores the above information and offers an extensive set of methods to interact with the files and folders. Assuming you want to create a workspace for a folder named *folderName*, create a string containing the computer name and the folder:

Code snippet (C#):

```
// Specify workspace name for later use
String workspaceName = String.Format("{0}-{1}", Environment.MachineName, "Test_TFSAPI");

// Create new workspace
Workspace newWorkspace = verControlServer.CreateWorkspace(workspaceName,
verControlServer.AuthorizedUser);
```

There can now be a mapping between the server-side folder and your local folder specified by *folderName* under the *workspace*. To retain or delete an existing workspace, simply carry out the following methods:

Code snippet (C#):

```
// Delete workspace
bool workspaceDeleted = verControlServer.DeleteWorkspace(workspaceName,
verControlServer.AuthorizedUser);

// Get existing workspace
Workspace existingWorkspace = verControlServer.GetWorkspace(workspaceName,
verControlServer.AuthorizedUser);
```

Creating working folders

For a connection to a specified project folder, you need a relative path with regard to the root.

For example, if the project is saved on: *your-server\Development\TcSampleProjectX*, then the relative path to this folder is *$\Development\TcSampleProjectX.*

To align this folder with the projects on the server, you can iterate over a set of all registered projects on the server. A registered project is the root of the project collection below it.

Code snippet (C#):

```
// Create mapping between server and local folder
string serverFolder = String.Format("$/{0}", teamProject.Name + "/Folder/SubFolder");
string localFolder = Path.Combine(@"C:\tfs", workspaceName);
WorkingFolder workingFolder = new WorkingFolder(serverFolder, localFolder);
existingWorkspace.CreateMapping(workingFolder);
```

Getting the latest versions

As mentioned above, the Workspace class provides a rich set of methods to execute TFS commands from code. To illustrate this, here is an example of creating a link to an element in the working folder. To create a relative path:

Code snippet (C#):

```
String existingItemPath = "$/Developement/TcSampleProject/Examples/Samples00/TestPlc/POUs/
MAIN.TcPOU";
```

Or an absolute path:

```
String existingItemPath = C:/tfs/Developement/TcSampleProject/Examples/TestPlc/POUs/MAIN.TcPOU";
```

To get the latest version of the element in the example, add the following line of code:

Code snippet (C#):

```
String existingItemPath = "$/TeamProjectName/Folder/SubFolder";
GetRequest itemRequest = new GetRequest(new ItemSpec(existingItemPath, RecursionType.Full),
VersionSpec.Latest);
existingWorkspace.Get(itemRequest, GetOptions.GetAll);
```

Here the *RecursionType.Full* executes the request for all elements under the element node, but you can choose it according to the requirements of your application and the existing hierarchy of elements. For more information, see the API documentation on MSDN.

Receiving a list of upcoming changes

You can also get a list of pending changes made to the element or a collection of elements using the following line of code:

Code snippet (C#):

```
PendingChange[] pendingChanges = existingWorkspace.GetPendingChanges(existingItemPath,
RecursionType.Full);
```

When collecting elements, take into account the excessive deviations in Item[] as an argument.

Checking in and out

You can check out an element or a collection of elements:

Code snippet (C#):

```
int checkoutResult = existingWorkspace.PendEdit(existingItemPath, RecursionType.Full);
```

To check in the collection or pending changes to an element:

Code snippet (C#):

```
int checkinResult = workspace.CheckIn(pendingChanges, usercomment);
```

-Undoing a process

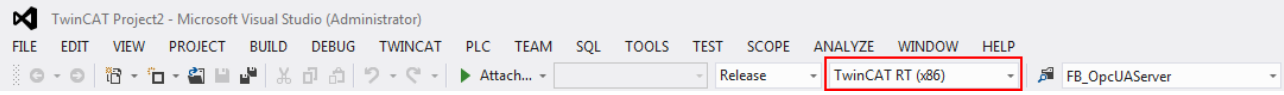
To undo a process, simply do the following:

Code snippet (C#):

```
int undoResult = existingWorkspace.Undo(existingItemPath, RecursionType.Full);
```

4.3.1.3 Setting the TwinCAT target platform

This article describes how to set the TwinCAT target platform via the Automation Interface code. The target platform determines for which target the TwinCAT configuration is to be compiled, e.g. TwinCAT x86 or TwinCAT x64, and is usually set from a TwinCAT XAE toolbar in Visual Studio.



The following code snippet assumes that you already have a DTE instance that connects to a TwinCAT configuration. It checks the currently set target platform and switches it to another platform.

Code snippet (C#):

```
ITcSysManager7 systemManager7 = pro.Object as ITcSysManager7;
ITcConfigManager configManager = systemManager7.ConfigurationManager as ITcConfigManager;
if (configManager.ActiveTargetPlatform == "TwinCAT RT (x86)")
configManager.ActiveTargetPlatform = "TwinCAT RT (x64)";
else
configManager.ActiveTargetPlatform = "TwinCAT RT (x86)";
```

Code snippet (PowerShell):

```
$configManager = $systemManager.ConfigurationManager
if ($configManager.ActiveTargetPlatform -eq "TwinCAT RT (x86)"){
  $configManager.ActiveTargetPlatform = "TwinCAT RT
(x64)"} else {$configManager.ActiveTargetPlatform = "TwinCAT RT (x86)"}
```

Changing the target hardware platform or target device with TwinCAT configuration?

This article describes how to change the target hardware platform. If you want to change the actual target device to which the TwinCAT configuration is to be written, please use the ITcSysManager::SetTargetNetId() method.

The following code snippet shows how to access the Visual Studio Configuration Manager and activate or deactivate individual TwinCAT sub-projects (PLC, C++ ...) for the build process. The snippet assumes that the TwinCAT project currently open is called "TwinCAT Project1" and that this contains a PLC project "Untitled1". It then deactivates the PLC project for the build process.

Code snippet (C#):

```
EnvDTE.SolutionContexts solutionContexts =
solution.SolutionBuild.ActiveConfiguration.SolutionContexts;
foreach (EnvDTE.SolutionContext solutionContext in solutionContexts)
{
switch(solutionContext.ProjectName)
{
case "TwinCAT Project13\\Untitled1\\Untitled1.plcproj":
solutionContext.ShouldBuild = false;
break;
}
}
```

Code snippet (PowerShell):

```
$solutionContexts = $sln.SolutionBuild.ActiveConfiguration.SolutionContexts
foreach ($solutionContext in $solutionContexts)
{
  if($solutionContext.ProjectName -eq "TestSolution\\Untitled1\\Untitled1.plcproj")
  {
    $solutionContext.ShouldBuild = $false
  }
}
```

4.3.1.4 Attaching to an existing Visual Studio instance

The following code snippets demonstrate how to attach to an existing (already running) instance of Visual Studio. The snippets have been written in C#.

The sample consists of three different methods that depend on each other. Of course, you may change this accordingly so that it better fits your application environment.The following table explains each method in more detail.

Method	Description
getRunningObjectTable()	Queries the Running Object Table (ROT) for a snapshot of all running processes in the table. Returns all found processes in a Hashtable, which are then used by getIdeInstances() for further filtering for DTE instances.
getIdeInstances()	Searches for DTE instances in the ROT snapshot and returns a Hashtable with all found instances. This Hashtable may then be used by the method attachToExistingDte() to select single DTE instances. You may change the query for candidateName according to a more specific Visual Studio progId [▶ 29] , e.g. "VisualStudio.DTE.11.0" to query for Visual Studio 2012 instances.
attachToExistingDte()	Uses the getIdeInstances() method to select a DTE instance based on its solution path and, when found, attaches a new DTE object to this instance.

getRunningObjectTable()

```
public static Hashtable GetRunningObjectTable()
{
    Hashtable result = new Hashtable();
    int numFetched;
    UCOMIRunningObjectTable runningObjectTable;
    UCOMIEnumMoniker monikerEnumerator;
    UCOMIMoniker[] monikers = new UCOMIMoniker[1];
    GetRunningObjectTable(0, out runningObjectTable);
    runningObjectTable.EnumRunning(out monikerEnumerator);
    monikerEnumerator.Reset();
    while (monikerEnumerator.Next(1, monikers, out numFetched) == 0)
    {
        UCOMIBindCtx ctx;
        CreateBindCtx(0, out ctx);
        string runningObjectName;
        monikers[0].GetDisplayName(ctx, null, out runningObjectName);
        object runningObjectVal;
        runningObjectTable.GetObject(monikers[0], out runningObjectVal);
        result[runningObjectName] = runningObjectVal;
    }
    return result;
}
```

Please note that you need to explicitely reference the CreateBindCtx() method as a DllImport from the ole32.dll, e.g.:

```
[DllImport("ole32.dll")]
private static extern int CreateBindCtx(uint reserved, out IBindCtx ppbc);
```

getIdeInstances()

```
public static Hashtable GetIDEInstances(bool openSolutionsOnly, string progId)
{
  Hashtable runningIDEInstances = new Hashtable();
  Hashtable runningObjects = GetRunningObjectTable();
  IDictionaryEnumerator rotEnumerator = runningObjects.GetEnumerator();
  while (rotEnumerator.MoveNext())
  {
    string candidateName = (string)rotEnumerator.Key;
    if (!candidateName.StartsWith("!" + progId))
      continue;
    EnvDTE.DTE ide = rotEnumerator.Value as EnvDTE.DTE;
    if (ide == null)
      continue;
    if (openSolutionsOnly)
    {
      try
```

```
      {
        string solutionFile = ide.Solution.FullName;
        if (solutionFile != String.Empty)
          runningIDEInstances[candidateName] = ide;
      }
      catch { }
    }
    else
      runningIDEInstances[candidateName] = ide;
  }
  return runningIDEInstances;
}
```

attachToExistingDte()

```
public EnvDTE.DTE attachToExistingDte(string solutionPath)
{
EnvDTE.DTE dte = null;
Hashtable dteInstances = GetIDEInstances(false, progId);
IDictionaryEnumerator hashtableEnumerator = dteInstances.GetEnumerator();

while (hashtableEnumerator.MoveNext())
{
EnvDTE.DTE dteTemp = hashtableEnumerator.Value as EnvDTE.DTE;
if (dteTemp.Solution.FullName == solutionPath)
{
Console.WriteLine("Found solution in list of all open DTE objects. " + dteTemp.Name); dte = dteTemp;
}
}
return dte;
}
```

4.3.1.5 Accessing window tabs in Visual Studio

The Visual Studio Automation Interface provides methods and properties to access active windows in Visual Studio. The following chapter demonstrates some sample code. However, we also suggest to investigate the webpages of the Microsoft Developer Network (MSDN) for more detailed information about the Visual Studio object model. This chapter provides sample code for the following tasks:

- Accessing active windows
- Closing active windows
- Opening windows from TwinCAT PLC configuration

Accessing active windows

The DTE interface provides a property called "ActiveWindow", which outputs the currently active window in Visual Studio as an object of type EnvDTE.Window.

Code snippet (C#):

```
EnvDTE.Window activeWin = dte.ActiveWindow;
```

Code snippet (PowerShell):

```
$activeWin = $dte.ActiveWindow
```

Closing active windows

Depending on the customers application, it may be necessary to close all active windows in Visual Studio before proceeding with the TwinCAT configuration via Automation Interface. The following code snippet closes all active windows but the "Solution Explorer".

Code Snippet (C#):

```
try
{
while (!dte.ActiveWindow.Caption.Contains("Solution Explorer"))
dte.ActiveWindow.Close();
}
catch (InvalidOperationException ex)
{
// use DTE.Quit() to close main window
}
```

Opening windows from the TwinCAT PLC configuration

It is also possible to open TwinCAT PLC windows, e.g. a visualization. The following code snippet opens an existing TwinCAT visualization from the PLC project "Untitled1" and makes it the active window.

Code snippet (C#):

```
string fileName = @"C:\TwinCAT Project1\TwinCAT Project1\Untitled1\VISUs\Visu.TcVIS";
dte.ItemOperations.OpenFile(fileName);
```

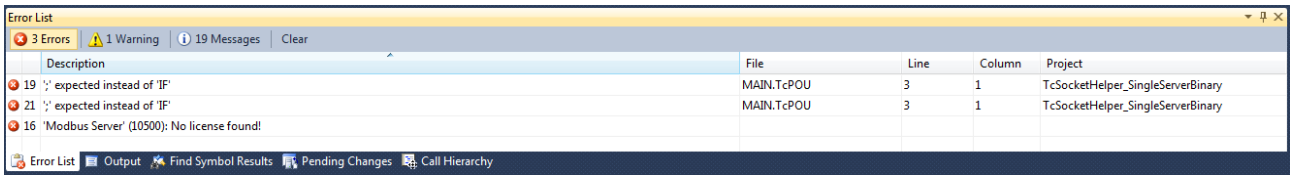
Code snippet (PowerShell):

```
$fileName = @"C:\TwinCAT Project1\TwinCAT Project1\Untitled1\VISUs\Visu.TcVIS"
dte.ItemOperations.OpenFile($fileName)
```

4.3.1.6 Accessing the Error List window of Visual Studio

This chapter describes how to access the Visual Studio Error List window. Because TwinCAT 3 integrates itself into the Visual Studio environment, reading the contents of this window is of great importance for debugging and diagnostics. To read the contents of this window, you must access the Visual Studio COM interface. Since this COM interface is also necessary for accessing TwinCAT XAE (e.g. as described in our article Accessing the TwinCAT configuration [▶ 19]), you do not need to add any additional references to your program code.

The following documentation describes how to access the Error List window, which can be very helpful, e.g. if you want to compile a PLC project and want to monitor the compilation process for any error messages. Status messages for various properties of the Visual Studio (and therefore TwinCAT) development environment are displayed in the Error List window. These properties include, for example, compilation errors or license problems that occur when compiling a project.



Reading the contents of this window can be very important, e.g. to detect errors that occur when compiling a PLC project. The following code snippet shows how the content of this window can be read using a C# application.

Code snippet (C#):

```
ErrorItems errors = dte.ToolWindows.ErrorList.ErrorItems;
for (int i = 1; i < errors.Count; i++)
{
  ErrorItem item = errors.Item(i);
}
```

Make sure that the "ToolWindows" property is available in the EnvDTE80.DTE2 namespace.

Code snippet (PowerShell):

```
$errors = $dte.ToolWindows.ErrorList.ErrorItems
for ($i=1; $i -lt $errors.Count; $i++)
{
  $item = $errors.Item($i);
}
```

As you can see, the ErrorItems property returns a collection of all items present in the error list, where each item is of type ErrorItem. You can go through this collection by making use of its Count property, for example. Each ErrorItem object has the following properties, which correspond to the columns in the Error List window (compare with screenshot above):

Property	Description
Description	Describes the error message.
FileName	Name of the file in which the error occurred.
Line	Line in which the error occurred.
Column	Column in which the error occurred.
Project	Project in which the error occurred.

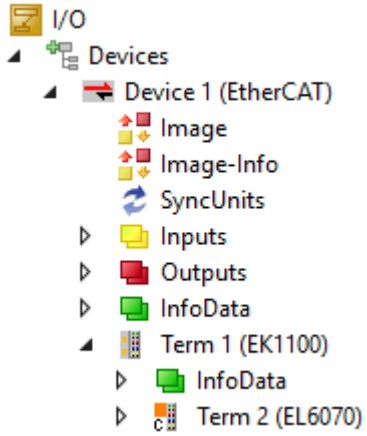
Note that the last four properties are not always used or do not make sense in every situation. For example, you can see in the screenshot above that the error message "No license found" is not linked to a specific file, line or column because it is a general TwinCAT error message.

4.3.2 Licensing

4.3.2.1 Configuration of licensing hardware

The following article describes how to configure licensing hardware (e.g. an EL6070 terminal). Essentially, this concerns the selection and configuration of the device in the TwinCAT licensing dialogs.

As a requirement for the following steps, the corresponding licensing hardware must be present in the I/O part of the TwinCAT configuration, e.g. in the case of an EL6070:



Finding all existing licensing devices

A check for all existing licensing hardware can be performed by exporting the XML description to the "License" node of the TwinCAT configuration.

Code snippet (C#):

```
ItcSmTreeItem license = systemManager.LookupTreeItem("TIRC^License");
string xmlDescription = license.ProduceXml();
```

Code snippet (PowerShell):

```
$license = $systemManager.LookupTreeItem("TIRC^License");
$xmlDescription = $license.ProduceXml();
```

The XML description lists the available licensing hardware, for example:

```
<TreeItem>
  <ItemName>License</ItemName>
  <PathName>TIRC^License</PathName>
  <ItemType>59</ItemType>
  <LicenseDef>
    <AvailableLicenseDevices>
      <LicenseDevice>
        <Name>Term 2 (EL6070)</Name>
        <PathName>TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL6070)</PathName>
        <TypeName>EL6070 1Ch. Licensing-Terminal</TypeName>
        <ObjectID>50462722</ObjectID>
      </LicenseDevice>
    </AvailableLicenseDevices>
    <Commands>
```

```
      <ActivateResponseFile/>
    </Commands>
  </LicenseDef>
</TreeItem>
```

Selecting a licensing device

To add licensing hardware to the TwinCAT license configuration, either the device name or its ObjectID can be used. The former is well-suited if the device name is known, e.g. if the I/O device was generated in advance. The latter can be determined from the XML description above.

Code snippet (C#):

```
ITcSmTreeItem el6070dev1 = license.CreateChild("NameOfDevice", 0, null, "Term 2 (EL6070)"); //
DeviceName
ITcSmTreeItem el6070dev2 = license.CreateChild("NameOfDevice", 0, null, "50462722"); // ObjectID
```

Code snippet (PowerShell):

```
$el6070dev1 = $license.CreateChild("NameOfDevice", 0, $null, "Term 2 (EL6070)"); // DeviceName
$el6070dev2 = $license.CreateChild("NameOfDevice", 0, $null, "50462722"); // ObjectID
```

System requirements

Required TwinCAT version
TwinCAT v3.1.4022.4

4.3.2.2 Activating license response files

The following article describes how license response files can be loaded via the TwinCAT Automation Interface. This function is provided via the XML description of the "License" node.

```
<TreeItem>
  <ItemName>License</ItemName>
  <PathName>TIRC^License</PathName>
  <ItemType>59</ItemType>
  <LicenseDef>
    <Commands>
      <ActivateResponseFile>
        <Path>...</Path>
        <OemGuid>...</OemGuid>
      </ActivateResponseFile>
    </Commands>
  </LicenseDef>
</TreeItem>
```

The <Path> to the license response file can be specified in the XML area <ActivateResponseFile>. This XML document can then be loaded via a ConsumeXml() to the "License" node in order to activate the license response file. The parameter <OemGuid> is only required in special cases and can be given any value, e.g. 0.

Code snippet (C#):

```
ITcSmTreeItem license = systemManager.LookupTreeItem("TIRC^License");
license.ConsumeXml(xmlDescriptionFromAbove);
```

System requirements

Required TwinCAT version
TwinCAT v3.1.4022.4

4.3.3 System

4.3.3.1 Opening and activating existing configurations

To activate a previously created configuration, an instance of TwinCAT XAE must be created, the configuration loaded and activated.

Procedure

The ProgId **"VisualStudio.DTE.10.0"** is used to create an instance of Microsoft Visual Studio®. Full control over Microsoft Visual Studio® is possible via the Visual Studio DTE object. The procedure for creating the ITcSysManager [▶ 112] interface (here the *'sysMan'* instance) is described in the chapter Accessing TwinCAT configurations [▶ 19].

Example (CSharp):

Please note that for this example you have to add a reference to "Microsoft Developer Environment 10.0" as well as to "Beckhoff TwinCAT XAE Base" to your project.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using EnvDTE100;
using System.IO;
using TCatSysManagerLib;

namespace ActivatePreviousConfiguration
{
   class Program
   {
      static void Main(string[] args)
      {
         Type t = System.Type.GetTypeFromProgID("VisualStudio.DTE.10.0");
         EnvDTE.DTE dte = (EnvDTE.DTE)System.Activator.CreateInstance(t);
         dte.SuppressUI = false;
         dte.MainWindow.Visible = true;

         EnvDTE.Solution sol = dte.Solution;
         sol.Open(@"C:\Temp\SolutionFolder\MySolution1\MySolution1.sln");

         EnvDTE.Project pro = sol.Projects.Item(1);

         ITcSysManager systemManager = pro.Object as ITcSysManager;

         sysMan.ActivateConfiguration();
         sysMan.StartRestartTwinCAT();
      }
   }
}
```

Example (PowerShell):

```
$prjDir = "C:\tmp\TestSolution\"
$prjName = "TestSolution.sln"
$prjPath = $prjDir += $prjName
$dte = new-object -com VisualStudio.DTE.10.0
$dte.SuppressUI = $false
$dte.MainWindow | %{$_.gettype().InvokeMember("Visible","SetProperty",$null,$_,$true)}

$sln = $dte.Solution
$sln.Open($prjPath)

$project = $sln.Projects.Item(1)
$systemManager = $project.Object

$systemManager.ActivateConfiguration()
$systemManager.StartRestartTwinCAT()
```

Example (VBScript):

```
dim dte,sln,proj,sysMan
set dte = CreateObject("VisualStudio.DTE.10.0")
set sln = dte.Solution
call sln.Open("C:\SolutionFolder\MySolution1.sln")
set proj = sln.Projects(1)
set sysMan = proj.Object
call sysMan.ActivateConfiguration
call sysMan.StartRestartTwinCAT
```

4.3.3.2 Opening existing projects from a TwinCAT Target

This documentation article describes how to open existing TwinCAT projects from a connected TwinCAT Target. The Target needs to be available, which means that ADS routes have to be present in order to be able to retrieve the project from the Target.

The following code snippet demonstrates how to retrieve the TwinCAT project from a connected Target Runtime.

Code snippet (C#):

```
dte.ExecuteCommand("File.OpenProjectFromTarget", "CX-123456 C:\\ProjectDir ProjectName");
```

Code snippet (Powershell):

```
$dte.ExecuteCommand("File.OpenProjectFromTarget", "CX-123456 C:\\ProjectDir ProjectName");
```

The method ExecuteCommand() from the Visual Studio API allows to trigger the TwinCAT command (File.OpenProjectFromTarget) to open the project from a connected TwinCAT Target. The three parameters are to be included in the second parameter of the ExecuteCommand() method. These three parameters are: Route name to Target, local project directory (where the project files should be stored), local project name. All three parameters are to be separated with a space from each other.

4.3.3.3 Creating and handling variable mappings

A very common scenario in which the TwinCAT Automation Interface is being used, is to automatically create variable mappings, e.g. between PLC input/output variables and their corresponding I/O counterparts. The Automation Interface provides several methods that simplify the task to create, delete or save variable mappings. The following documentation article briefly describes these methods and gives examples on how to use them. The following topics are covered:

- General information
- Link variables
- Unlink variables
- Get/Set all variable mappings
- Delete all variable mappings

General information

Variable mappings may occur between different input/output tree items in a TwinCAT project, for example:

- Between PLC input/output variables and I/O (and vice versa)
- Between PLC input/output variables and NC (and vice versa)
- Between PLC input/output variables and TcCOM objects (and vice versa)
- …

The information in this article describes Automation Interface mechanisms which may be used for all of these use cases.

Link variables

From an Automation Interface point-of-view, variable mappings are always being performed between two tree items, e.g. between a PLC input/output variable and its corresponding I/O counterpart. To link two tree items, the Automation Interface provides the method ITcSysManager::LinkVariables() which links a given source tree item with a given destination tree item.

Code snippet (C#):

```
string source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn";
string destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input";
systemManager.LinkVariables(source, destination);
```

Code snippet (Powershell):

```
$source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn"
$destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input"
$systemManager.LinkVariables($source, $destination)
```

Unlink variables

Similar to linking variables, the Automation Interface provides a method ITcSysManager::UnlinkVariables() which releases the link between a given source tree item and a given destination tree item.

Code snippet (C#):

```
string source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn";
string destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input";
systemManager.UnlinkVariables(source, destination);
```

Code snippet (Powershell):

```
$source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn"
$destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input"
$systemManager.UnlinkVariables($source, $destination)
```

Save/Restore all variable mappings

To save or restore all variable mappings in a TwinCAT project, the methods ITcSysManager2::ProduceMappingInfo() and ITcSysManager2::ConsumeMappingInfo() can be used. The former reads all variable mappings in a TwinCAT project and returns them in an XML structure that can be re-imported later by using the latter method.

Code snippet (C#):

```
ITcSysManager2 systemManager2 = (ITcSysManager2)systemManager;
string mappingInfo = systemManager2.ProduceMappingInfo();
systemManager2.ConsumeMappingInfo(mappingInfo);
```

Code snippet (Powershell):

```
$mappingInfo = $systemManager.ProduceMappingInfo()
$systemManager.ConsumeMappingInfo($mappingInfo)
```

Delete all variable mappings

To delete all variable mappings in a TwinCAT project, the method ITcSysManager3.ClearMappingInfo() may be used.

Code snippet (C#):

```
ITcSysManager3 systemManager = (ITcSysManager3)systemManager;
systemManager.ClearMappingInfo();
```

Code snippet (Powershell):

```
$systemManager.ClearMappingInfo()
```

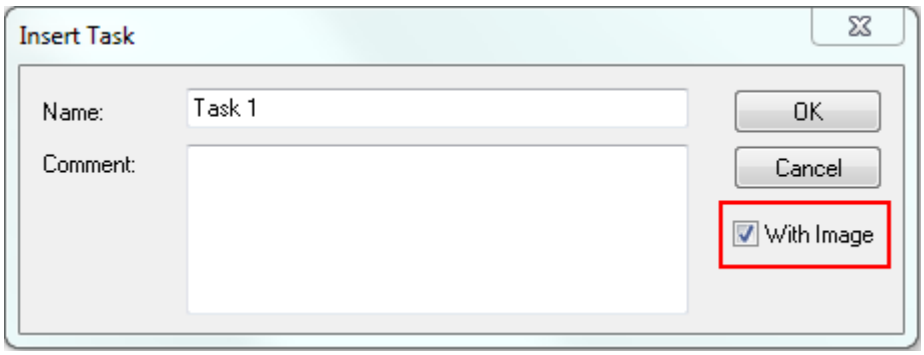
4.3.3.4 Creating and handling Tasks

This article explains how to create and handle Tasks via TwinCAT Automation Interface. It consists of the following topics:

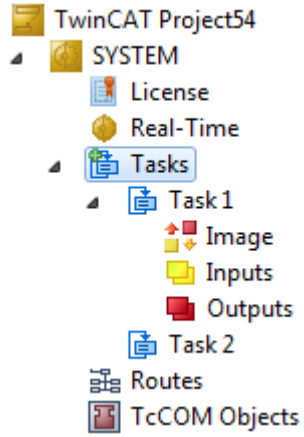
- General information
- Inserting Tasks
- Inserting input/output variables

General Information

Two task types can be configured in TwinCAT XAE and therefore also with the Automation Interface: tasks with and without a process image. When you insert a new task in TwinCAT XAE, you can decide whether you want to insert a process image or not by selecting or deselecting the corresponding checkbox in the "Insert Task" dialog box.



The inserted task then either contains three further subordinate nodes (image, inputs, outputs) or not - as shown in the following example (task 1 = with image, task 2 = without image).



Inserting Tasks

To insert a Task via Automation Interface, you can make use of the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method and the corresponding SubTypes for "With Image" (SubType = 0) and "Without Image" (SubType = 1).

Code Snippet (C#)

```
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
tasks.CreateChild("Task 1 (With Image)", 0, null, null);
```

Code Snippet (Powershell):

```
$tasks = $systemManager.LookupTreeItem("TIRT")
$tasks.CreateChild("Task 1 (With Image)", 0, $null, $null)
```

Code Snippet (C#)

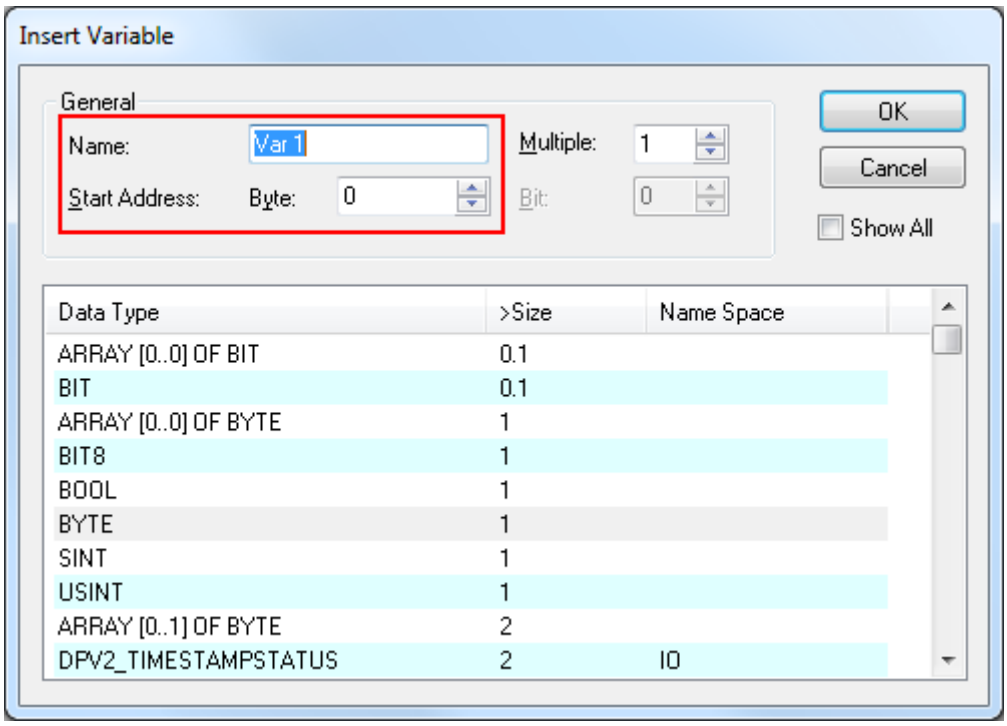
```
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
tasks.CreateChild("Task 2 (Without Image)", 1, null, null);
```

Code Snippet (Powershell):

```
$tasks = $systemManager.LookupTreeItem("TIRT")
$tasks.CreateChild("Task 1 (Without Image)", 1, $null, $null)
```

Inserting input/output variables

Input and output variables can be added to process images (task "With Image"). These can then be linked with the various I/O devices or variables of other tasks. You can use the corresponding TwinCAT XAE dialog to select the data type and the address of the input/output variables in the process image, for example. Click on "Ok" to add the variable to the process image.



This procedure can also be triggered via the Automation Interface using the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method with the corresponding variable data type as vInfo. In this case, SubType specifies the "Start Address" as shown in the dialog above.

Code snippet (C#):

```
ITcSmTreeItem task1 = systemManager.LookupTreeItem("TIRT^Task 1 (With Image)^Inputs");
task1.CreateChild("bInput", -1, null, "BOOL");
```

Code snippet (PowerShell):

```
$task1 = $systemManager.LookupTreeItem("TIRT^Task 1 (With Image)^Inputs")
$task1.CreateChild("bInput", -1, $null, "BOOL")
```

If SubType = -1, TwinCAT automatically appends the new variable to the end of the variable list.

4.3.3.5 Using Templates

The following article describes how to use templates instead of configuring each individual setting separately via the Automation Interface. The use of templates offers many advantages: they are easier to maintain, easier to replace with new templates and they offer more options when working with several teams on a TwinCAT project. This document describes the use of templates and covers the following topics:

- The general idea behind templates and their different levels
- Working with I/O templates
- Working with movement templates (axes)
- Working with PLC templates

The general idea behind templates and their different levels

The idea behind the use of templates is as simple as the Automation Interface code and the use of the Automation Interface itself. Most users are not aware that templates can exist at different levels in a TwinCAT configuration. These include:

- Templates at "configuration level":

  Templates at configuration level can exist as different TwinCAT configurations (*.sln or *.tszip file). Each configuration can contain different content and can be opened and activated by the Automation Interface code.

- Templates at "(PLC) project level":

  Templates at (PLC) project level can exist as multiple TwinCAT PLC projects, either as an unpacked folder structure (*.plcproj file) or as a container file (*.tpzip). These PLC projects can then be imported via the Automation Interface code if required.
- Templates at "structural element level":

  Templates at structural element level can exist as TwinCAT export files (*.xti). If required, these files can be imported via the Automation Interface code and contain all the settings that have been made for an EtherCAT master I/O device.

As the above description suggests, all the different template levels have one thing in common: the templates exist as files in the file system. It is considered best practice to clearly define which type of "template pool" is to be used at the beginning of the implementation of a custom Automation Interface application. A template pool describes a repository in which all template files (which can also be a mixture of different template levels) are stored. This could simply be the local (or remote) file system or a source control system (e.g. Team Foundation Server), from which template files are automatically retrieved, checked out and assembled into a new TwinCAT configuration. For a better overview and to simplify the first steps, we have created a separate documentation article that shows how the Visual Studio Object Model (DTE) is used for connection to a Team Foundation Server (TFS) and how these processes are executed via the program code. However, you can get more information about using TFS via the Automation Interface code, and we highly recommend reading the relevant articles on the Microsoft Developer Network (MSDN).

The following topics describe how each template level in the Automation Interface can be used for a different TwinCAT area (I/O, PLC, etc.).

Working with configuration templates

Templates at configuration level offer the most basic option for using templates. In this scenario, a template pool contains two or more pre-configured TwinCAT configurations and offers them as TwinCAT solution files (*.sln or *.tszip). These files can be opened via the Automation Interface using the Visual Studio method AddFromTemplate().

Code snippet (C#):

```
project = solution.AddFromTemplate(@"C:\TwinCAT Project.tszip", destination, projectName);
```

Code snippet (PowerShell):

```
$project = $solution.AddFromTemplate(@"C:\TwinCAT Project.tszip", $destination, $projectName)
```

OR

Code snippet (C#):

```
project = solution.Open(@"C:\TwinCAT Project 1.sln");
```

Code snippet (PowerShell):

```
$project = $solution.Open(@"C:\TwinCAT Project 1.sln")
```

Working with I/O templates

When working with I/O devices, users often receive the same tasks on the I/O devices again and again. This means making the same settings on the I/O devices with every TwinCAT configuration. TwinCAT offers a mechanism that saves all these settings in a special file format, the XTI file (*.xti). This file format can be used by the Automation Interface code to import a new configuration using the ImportChild() method defined in the ITcSmTreeItem interface.

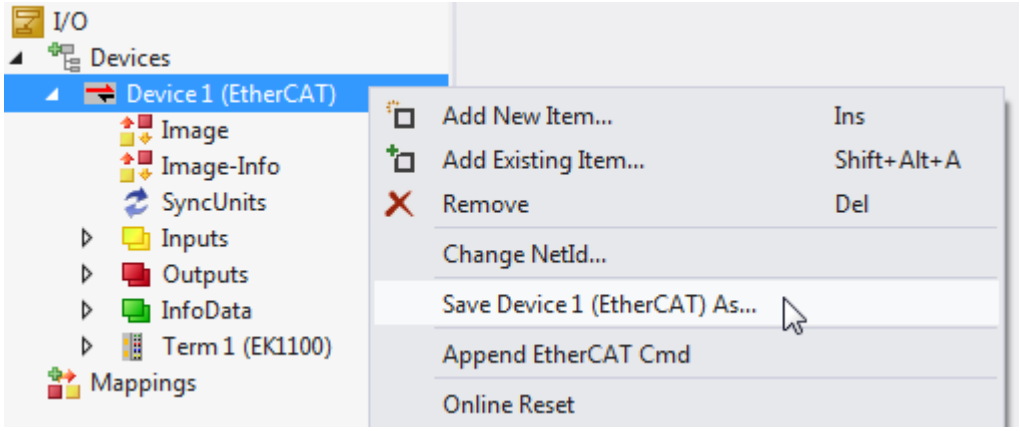
Code snippet (C#):

```
ITcSmTreeItem io = systemManager.LookupTreeItem("TIID");
ITcSmTreeItem newIo = io.ImportChild(@"C:\IoTemplate.xti", "", true, "SomeName");
```

Code snippet (PowerShell):

```
$io = $systemManager.LookupTreeItem("TIID")
$newIo = $io.ImportChild(@"C:\IoTemplate.xti", "", true, "SomeName")
```

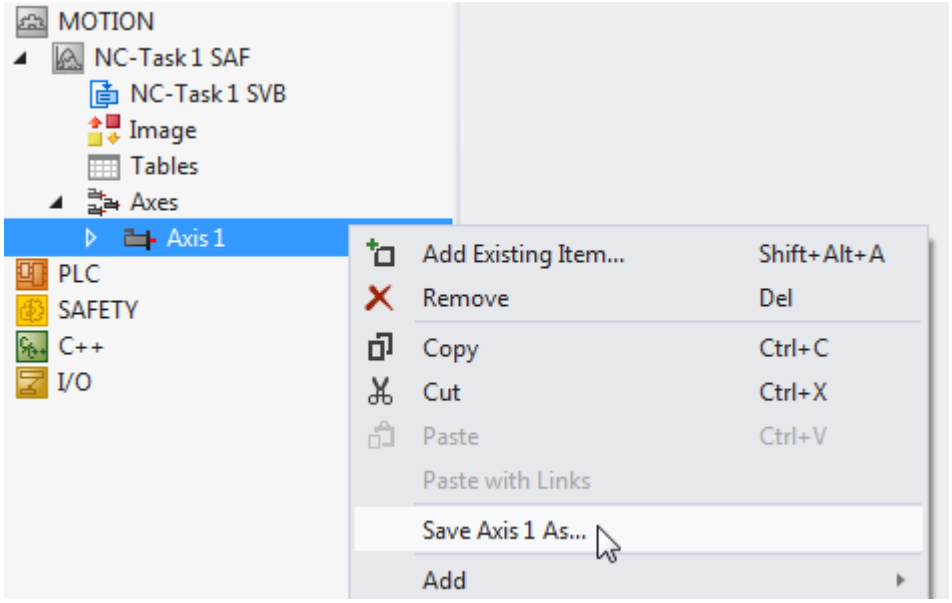
It is important to note that when exporting an I/O device within TwinCAT XAE, all sub-devices are automatically exported and are also included in the export file. The following screenshot shows how I/O devices are exported to an export file.



Remember that you can also use the Automation Interface to export an I/O device to an XTI file. The ExportChild() method from the ITcSmTreeItem interface is available for this purpose.

Working with movement templates (axes)

The use of motion axis templates is very similar to that of I/O devices. Axes can also be exported to an XTI file, either via TwinCAT XAE or via the Automation Interface code using ExportChild().



By using ImportChild(), these export files can be imported again later.

Code snippet (C#):

```
ITcSmTreeItem motionAxes = systemManager.LookupTreeItem("TINC^NC-Task^Axes");
motionAxes.ImportChild(@"C:\AxisTemplates.xti", "", true, "Axis 1");
```

Code snippet (PowerShell):

```
$motionAxes = $systemManager.LookupTreeItem("TINC^NC-Task^Axes")
$motionAxes.ImportChild(@"C:\AxisTemplates.xti", "", true, "Axis 1")
```

Working with PLC templates

PLC templates are available in two ways: You can either use the complete PLC projects as a whole or each POU individually as a template. For integration into an existing TwinCAT project, simply use the CreateChild() method of the ITcSmTreeItem interface.

Code snippet (C#):

```
ITcSmTreeItem plc = systemManager.LookupTreeItem("TIPC");
plc.CreateChild("NewPlcProject", 0, null, pathToTpzipOrTcProjFile);
```

Code snippet (PowerShell):

```
$plc = $systemManager.LookupTreeItem("TIPC")
$plc.CreateChild("NewPlcProject", 0, $null, $pathToTpzipOrTcProjFile)
```

Further options for importing existing PLC projects can be found in the best practice article on "Accessing, creating and handling PLC projects".

The next level of granularity for importing template files for the PLC is to import existing POUs, such as function blocks, structures, enumerations, global variable lists, etc. One of the reasons why a developer may opt for individual POUs as templates is that it is easier to build up a pool of existing functionalities and group them into separate POUs, e.g. different function blocks covering different sorting algorithms. When creating a project, the developer simply chooses the functionality he needs in his TwinCAT project and accesses the template tool to retrieve the corresponding POU and import it into the PLC project.

Code snippet (C#):

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^Name^Name Project");
plcProject.CreateChild("NameOfPou", 58, null, pathToPouFile);
plcProject.CreateChild(null, 58, null, stringArrayWithPathsToPouFiles);
```

Code snippet (PowerShell):

```
$plcProject = $systemManager.LookupTreeItem("TIPC^Name^Name Project")
$plcProject.CreateChild("NameOfPou", 58, $null, $pathToPouFile)
$plcProject.CreateChild($null, 58, $null, $stringArrayWithPathsToPouFiles)
```

Importing one or more templates

The POU template file may be a .TcPou file, or the corresponding DUT/GVL files. The above example shows two common ways of importing POU template files. The first is to import a single file, the second is to import multiple files at once by storing the file paths in a string array and using this string array as the vInfo parameter of CreateChild().

4.3.3.6 Accessing TwinCAT Remote Manager

This article describes how to access the TwinCAT Remote Manager functionality via the Automation Interface. The TwinCAT Remote Manager enables switching between TwinCAT 3 XAE versions installed on the same development computer. To access the Remote Manager via the Automation Interface, the following code snippets must be executed.

Code snippet (C#):

```
ITcRemoteManager remoteManager = dte.GetObject("TcRemoteManager");
remoteManager.Version = "3.1.4024.42";
```

Code snippet (PowerShell):

```
$remoteManager = $dte.GetObject("TcRemoteManager")
$remoteManager.Version = "3.1.4024.42"
```

4.3.3.7 Assigning tasks to CPU cores

Procedure

- Enabling the Cores for realtime usage.
- Paremetrization of the Cores with the LoadLimit, base cycle time (BaseTime) and a Latency Watchdog
- Assignment of Tasks to CPU resources.

The following code snippets are also available for download in our Samples section.

Sample (C#):

```
ITcSysManager3 systemManager = null;
[Flags()]
public enum CpuAffinity : ulong
{
CPU1 = 0x0000000000000001,
```

```
CPU2 = 0x0000000000000002,
CPU3 = 0x0000000000000004,
CPU4 = 0x0000000000000008,
CPU5 = 0x0000000000000010,
CPU6 = 0x0000000000000020,
CPU7 = 0x0000000000000040,
CPU8 = 0x0000000000000080,
None = 0x0000000000000000,
MaskSingle = CPU1,
MaskDual = CPU1 | CPU2,
MaskQuad = MaskDual | CPU3 | CPU4,
MaskHexa = MaskQuad | CPU5 | CPU6,
MaskOct = MaskHexa | CPU7 | CPU8,
MaskAll = 0xFFFFFFFFFFFFFFFF
}

public void AssignCPUCores()
{
ITcSmTreeItem realtimeSettings = systemManager.LookupTreeItem("TIRS");
// CPU Settings
// <TreeItem>
// <RTimeSetDef>
// <MaxCPUs>3</MaxCPUs>
// <Affinity>#x0000000000000007</Affinity>
// <CPUs>
// <CPU id="0">
// <LoadLimit>10</LoadLimit>
// <BaseTime>10000</BaseTime>
// <LatencyWarning>200</LatencyWarning>
// </CPU>
// <CPU id="1">
// <LoadLimit>20</LoadLimit>
// <BaseTime>5000</BaseTime>
// <LatencyWarning>500</LatencyWarning>
// </CPU>
// <CPU id="2">
// <LoadLimit>30</LoadLimit>
// <BaseTime>3333</BaseTime>
// <LatencyWarning>1000</LatencyWarning>
// </CPU>
// </CPUs>
// </RTimeSetDef>
// </TreeItem>

string xml = null;
MemoryStream stream = new MemoryStream();
StringWriter stringWriter = new StringWriter();
using(XmlWriter writer = XmlTextWriter.Create(stringWriter))
{
writer.WriteStartElement("TreeItem");
writer.WriteStartElement("RTimeSetDef");
writer.WriteElementString("MaxCPUs", "4");
string affinityString = string.Format("#x{0}",((ulong)
cpuAffinity.MaskQuad).ToString("x16"));
writer.WriteElementString("Affinity", affinityString);
writer.WriteStartElement("CPUs");
writeCpuProperties(writer, 0, 10, 1000, 10000, 200);
writeCpuProperties(writer, 1, 20, 5000, 10000, 500);
writeCpuProperties(writer, 2, 30, 3333, 10000, 1000);
writer.WriteEndElement(); // CPUs
writer.WriteEndElement(); // RTimeSetDef
writer.WriteEndElement(); // TreeItem
}
xml = stringWriter.ToString();
realtimeSettings.ConsumeXml(xml);
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
ITcSmTreeItem task1 = tasks.CreateChild("TaskA",1);
setTaskProperties(task1,CpuAffinity.CPU1);
ITcSmTreeItem task2 = tasks.CreateChild("TaskB",1);
setTaskProperties(task2, CpuAffinity.CPU2);

ITcSmTreeItem task3 = tasks.CreateChild("TaskC", 1);
setTaskProperties(task3, CpuAffinity.CPU3);
}

private void setTaskProperties(ITcSmTreeItem task, CpuAffinity affinityMask)
{
// <TreeItem>
// <TaskDef>
// <CpuAffinity>#x0000000000000004</CpuAffinity>
```

```
// </TaskDef>
// </TreeItem>

StringWriter stringWriter = new StringWriter();
using(XmlWriter writer = new XmlTextWriter(stringWriter))
{
writer.WriteStartElement("TreeItem");
writer.WriteStartElement("TaskDef");
string affinityString = string.Format("#x{0}", ((ulong)affinityMask).ToString("x16"));
writer.WriteElementString("CpuAffinity",affinityString);
writer.WriteEndElement();
writer.WriteEndElement();
}

string xml = stringWriter.ToString();
task.ConsumeXml(xml);
}

private void writeCpuProperties(XmlWriter writer, int id, int loadLimit, int baseTime, int
latencyWarning)
{
writer.WriteStartElement("CPU");
writer.WriteAttributeString("id", id.ToString());
writer.WriteElementString("LoadLimit", loadLimit.ToString());
writer.WriteElementString("BaseTime", baseTime.ToString());
writer.WriteElementString("LatencyWarning", latencyWarning.ToString());
writer.WriteEndElement();
}
```

4.3.3.8 Configuring TwinCAT Boot settings

The following documentation article describes how to configure the TwinCAT Boot settings via Automation Interface. For this the methods ITcSmTreeItem::ProduceXml() and ITcSmTreeItem::ConsumeXml() can be used to generate or import the following XML structure, which represents the corresponding settings in TwinCAT XAE.

```
<TreeItem>
  <System>
    <BootSettings>
      <AutoRun>true</AutoRun>
      <AutoLogon>true</AutoLogon>
      <LogonUserName>UserName</LogonUserName>
      <LogonPassword>Password</LogonPassword>
      <BootFileEncryptionType>None</BootFileEncryptionType>
    </BootSettings>
  </System>
</TreeItem>
```

4.3.3.9 Activate or deactivate IndependentProjectFile setting

To enhance engineering experience with regard to Source Control integration, TwinCAT 3 provides the possibility to store settings in separate project files – called “IndependentProjectFile”. This tree item based setting can also be activated/deactivated via TwinCAT Automation Interface. The interface ITcSmTreeItem6 provides the necessary property.

The following code snippet demonstrates how to activate this setting if it is deactivated, e.g. on an EtherCAT Master device.

Code Snippet (C#):

```
ITcSmTreeItem6 etherCatMaster = (ITcSmTreeItem6)systemManager.LookupTreeItem("TIID^Device 1
(EtherCAT)");
if (etherCatMaster.SaveInOwnFile == false)
  etherCatMaster.SaveInOwnFile = true;
```

Code Snippet (Powershell):

```
$etherCatMaster = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)")
if ($etherCatMaster.SaveInOwnFile -eq $false)
{
  $etherCatMaster.SaveInOwnFile = $true
}
```

4.3.3.10 From offline to online configurations

This article explains how to convert a TwinCAT configuration, which has been created 'offline', via TwinCAT Automation Interface to an online configuration. The term "offline" means that no physical I/Os are present at the time of configuration creation and therefore the real address information is not available, e.g. for an EtherCAT Master . The following topics are part of this article:

- General information
- Creating an offline configuration
- Switching to an online configuration

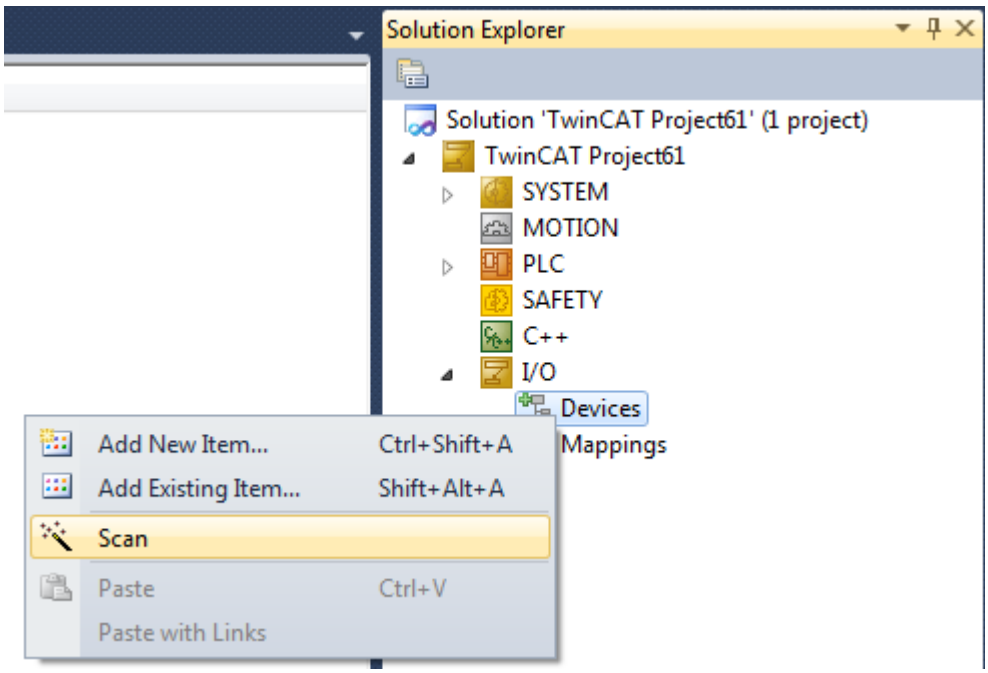
General Information

Two scenarios are common when creating a TwinCAT configuration:

- Scenario 1: This scenario is based on the real physical device on which the TwinCAT configuration is to run later. For this reason, all I/Os are **online** , already available and connected to the device.
- Scenario 2: This scenario may involve creating the configuration **offline** (i.e. without any I/O connected to the device) and later switching to an online configuration once the I/Os are available. This scenario is the main focus of this article.

Both scenarios are possible with the TwinCAT Automation Interface. However, because important information about some of the I/Os is missing in the case of scenario 2, e.g. their physical addresses, the latter is somewhat more sophisticated because the required (address) information has to be passed later.

The familiar TwinCAT XAE functionality "Scan Devices" plays an important role in this use case because it scans all available I/O devices and automatically connects them to the configuration, along with any additional information required, e.g. the physical addresses.



In the case of a physical controller, this functionality can also be called up via the Automation Interface, in which case it returns an XML representation of all I/O devices found, including their corresponding address information.

If a TwinCAT configuration is to be activated on a controller, the required address information must be set for all I/O devices belonging to the configuration. This can be done by calling up the "Scan Devices" functionality via the Automation Interface and defining the address information of the I/O devices in the configuration. This will now be explained in more detail.

Creating an offline configuration

There are several articles which explain how to create an offline TwinCAT configuration. Please refer to our Product Description [▶ 10] page to get an overview.

Switching to an online configuration

If you finally have created a TwinCAT configuration that should now be activated on the physical controller, the following steps need to be taken to ensure that all required address information is available before downloading the configuration:

- Step 1 [optional]: Connecting to the target device
- Step 2: Scanning the device for available I/Os
- Step 3: Iterating through XML and configuring address information of I/Os
- Step 4: Activating the configuration

Of course, you can also always create an online configuration directly by using the ScanDevices functionality. There is an own sample [▶ 89] which shows you exactly how to do that.

Step 1 [optional]: Connecting to the target device

If the physical controller is not located on the same machine as the Automation Interface code runs on, you can use the ITcSysManager [▶ 112]::SetTargetNetId() [▶ 117] method to connect to the remote device and then continue with the next steps.

Step 2: Scanning the device for available I/Os

The "Scan Devices" functionality mentioned above can be triggered via Automation Interface by calling the ITcSmTreeItem::ProduceXml() method on the I/O devices node.

Code Snippet (C#):

```
ITcSmTreeItem ioDevices = systemManager.LookupTreeItem("TIID");
string foundDevices = ioDevices.ProduceXml();
```

Step 3: Iterating through XML and configuring address information of I/Os

In this example we want to update the address information of an EtherCAT device which is already part of our offline configuration. The ProduceXml() in step 2 has already returned the available I/O devices on the system, which is now available in the variable 'foundDevices'. We will identify the EtherCAT device in this XML via its item sub type (111) and then update the address information of the EtherCAT Master in our configuration.

Code Snippet (C#):

```
XmlDocument doc = new XmlDocument();
doc.LoadXml(foundDevices);
XmlNodeList devices = doc.SelectNodes("TreeItem/DeviceGrpDef/FoundDevices/Device");
foreach (XmlNode device in devices)
{
xmlNode typeNode = device.SelectSingleNode("ItemSubType");
int subType = int.Parse(typeNode.InnerText);
if (subType == 111)
{
XmlNode addressInfo = device.SelectSingleNode("AddressInfo");
ITcSmTreeItem deviceToUpdate = systemManager.LookupTreeItem("TIID^EtherCAT Master");
string xmlAddress = string.Format("<TreeItem><DeviceDef>{0}</DeviceDef></TreeItem>",
addressInfo.OuterXml);
deviceToUpdate.ConsumeXml(xmlAddress);
}
}
```

Step 4: Activating the configuration

The last step only involves activating the configuration on the target device. Simply use the ITcSysManager::ActivateConfiguration() method to do that.

Code Snippet (C#):

```
sysManager.ActivateConfiguration();
```

4.3.4 ADS

4.3.4.1 Creating and handling ADS routes

Adding ADS routes via the Automation Interface can be achieved by using the ConsumeXml() method of the ITcSmTreeItem interface. However, it is important to understand the underlying XML structure before adding routes to a remote target.

XML structure

The following code snippets represent sample XML structures for adding routes to a remote target. Please note that you can either specify the IP address or the hostname of the remote target.

This code snippet will add a regular route to a remote target.

Code Snippet (XML):

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddRoute>
      <RemoteName>RouteName</RemoteName>
      <RemoteNetId>1.2.3.4.5.6</RemoteNetId>
      <RemoteIpAddr>1.2.3.4</RemoteIpAddr>
      <UserName>userName</UserName>
      <Password>password</Password>
      <NoEncryption></NoEncryption>
      <LocalName>LocalName</LocalName>
    </AddRoute>
  </RoutePrj>
</TreeItem>
```

This code snippet will add a project route to a remote target.

Code Snippet (XML):

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddProjectRoute>
      <Name>RouteName</Name>
      <NetId>1.2.3.4.5.6</NetId>
      <IpAddr>1.2.3.4</IpAddr>
    </AddProjectRoute>
  </RoutePrj>
</TreeItem>
```

The following code snippet will use the hostname instead of the IP address.

Code Snippet (XML):

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddRoute>
      <RemoteName>RouteName</RemoteName>
      <RemoteNetId>1.2.3.4.5.6</RemoteNetId>
      <RemoteHostName>CX-12345</RemoteHostName>
      <UserName>userName</UserName>
      <Password>password</Password>
      <NoEncryption></NoEncryption>
      <LocalName>LocalName</LocalName>
```

```
    </AddRoute>
  </RoutePrj>
</TreeItem>
```

And for project routes.

Code Snippet (XML):

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddProjectRoute>
      <Name>RouteName</Name>
      <NetId>1.2.3.4.5.6</NetId>
      <HostName>1.2.3.4</HostName>
    </AddProjectRoute>
  </RoutePrj>
</TreeItem>
```

Please note that the XML structure for regular and project routes may be used simultaneously.

The following code snippet creates an ADS route to a remote target that has been specified by its IP address (10.1.128.217) and its AMS NetId (10.1.128.217.1.1).

Code Snippet (C#):

```
string xmlString ="<TreeItem><ItemName>Route Settings</ItemName><PathName>TIRR</
PathName><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList><AddRoute><RemoteName>RouteName</RemoteName><RemoteNetId>10.1.128.217.1.1</
RemoteNetId><RemoteIpAddr>10.1.128.217</RemoteIpAddr><UserName>Administrator</UserName><Password>1</
Password><NoEncryption></NoEncryption></AddRoute></RoutePrj></TreeItem>";
ITcSmTreeItem routes = systemManager.LookupTreeItem("TIRR");
routes.ConsumeXml(xmlString);
```

Code Snippet (Powershell):

```
$xmlString ="<TreeItem><ItemName>Route Settings</ItemName><PathName>TIRR</
PathName><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList><AddRoute><RemoteName>RouteName</RemoteName><RemoteNetId>10.1.128.217.1.1</
RemoteNetId><RemoteIpAddr>10.1.128.217</RemoteIpAddr><UserName>Administrator</UserName><Password>1</
Password><NoEncryption></NoEncryption></AddRoute></RoutePrj></TreeItem>"
$routes = $systemManager.LookupTreeItem("TIRR")
$routes.ConsumeXml($xmlString)
```

4.3.4.2 Execute an ADS broadcast search

To trigger a TwinCAT Broadcast search and find unknown remote ADS devices, the ConsumeXml() and ProduceXml() methods from the ITcSmTreeItem interface may be used.

General broadcast search

Code Snippet (C#):

```
string xmlString = "<TreeItem><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList></RoutePrj></TreeItem>";
ITcSmTreeItem routes = sysMan.LookupTreeItem("TIRR");
routes.ConsumeXml(xmlString);
string result = routes.ProduceXml();
```

Code Snippet (Powershell):

```
$xmlString = "<TreeItem><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></TargetList></
RoutePrj></TreeItem>"
$routes = $systemManager.LookupTreeItem("TIRR")
$routes.ConsumeXml($xmlString)
$result = $routes.ProduceXml()
```

The variable “result” now contains an XML representation of all found ADS devices on the network. To select a specific device from that list, regular .NET mechanisms for XML handling may be used.

Code Snippet (C#):

```
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.Load(result);
string amsNetId = xmlDocument.SelectSingleNode("//TreeItem/RoutePrj/TargetList/Target/
IpAddr[text()=\"" + ipAddress + "\"]/../NetId").InnerText;
string name = xmlDocument.SelectSingleNode("//TreeItem/RoutePrj/TargetList/Target/IpAddr[text()=\""
+ ipAddress + "\"]/../Name").InnerText;
```

Code Snippet (Powershell):

```
$xmlDocument = [xml]$result
$localAmsNetId = $xmlDocument.TreeItem.RoutePrj.Target
```

This information might then be used to add a route to that ADS device, as described in a separate documentation article.

Direct broadcast search

Requires at least TwinCAT 3.1 Build 4020.10 or higher.

To execute a broadcast search with a given hostname or IP address, the following XML structures can be used in ConsumeXml().

XML - Search for Hostname:

```
<TreeItem>
  <RoutePrj>
    <TargetList>
      <Search>CX-12345</Search>
    </TargetList>
  </RoutePrj>
</TreeItem>
```

XML - Search for IP address:

```
<TreeItem>
  <RoutePrj>
    <TargetList>
      <Search>172.17.60.153</Search>
    </TargetList>
  </RoutePrj>
</TreeItem>
```

A subsequent ProduceXml() will return the found host as follows:

XML - Found host:

```
<TreeItem>
  <RoutePrj>
    <TargetList>
      <Target>
        <Name>CX-12345</Name>
        <NetId>172.17.60.153.1.1</NetId>
        <IpAddr>172.17.60.153</IpAddr>
        <Version>3.1.4020</Version>
        <OS>Windows 7</OS>
      </Target>
    </TargetList>
  </RoutePrj>
</TreeItem>
```

4.3.5 PLC

4.3.5.1 Accessing, creating and handling PLC projects

This chapter explains in-depth how to create, access and handle PLC projects. The following list shows all chapters in this article:

- General information about PLC projects
- Creating and handling PLC projects
- Opening existing PLC projects
- Nested projects and project instances
- Saving the PLC project as a library

- Handling online functionalities (Login, StartPlc, StopPlc)
- Setting Boot project options
- Saving project and/or solution as archive
- Calling CheckAllObjects()

General information about PLC projects

PLC projects are specified with the help of their project templates. TwinCAT currently provides two templates, which are described by means of a template file in the TwinCAT directory. The following table shows which PLC templates are available and the corresponding template file:

Template name	Template file
Standard PLC Template	C:\TwinCAT\3.x\Components\Plc\PlcTemplate\Plc Templates\Standard PLC Template.plcproj
Empty PLC Template	C:\TwinCAT\3.x\Components\Plc\PlcTemplate\Plc Templates\Empty PLC Template.plcproj

Creating and handling PLC projects

To create a new PLC project using the Automation Interface, you must navigate to the PLC node and then execute the CreateChild() method with the appropriate template file as a parameter.

Code snippet (C#):

```
ITcSmTreeItem plc = systemManager.LookupTreeItem("TIPC");
ITcSmTreeItem newProject = plc.CreateChild("NameOfProject", 0, "", pathToTemplateFile);
```

Code snippet (PowerShell):

```
$plc = $systemManager.LookupTreeItem("TIPC")
$newProject = $plc.CreateChild("NameOfProject", 0, "", pathToTemplateFile)
```

Please note

When using standard PLC templates as provided by Beckhoff, make sure that you only use the template name instead of the entire path, e.g. "Standard PLC Template".

All subsequent operations, such as the creation and handling of POUs and the corresponding code input, are described in a separate article.

Once the PLC project has been created, it can be used further by converting it into the special interface ITcPlcIECProject [▶ 162], which grants more functionalities and access to the project-specific attributes:

Code snippet (C#):

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project");
ITcPlcIECProject iecProject = (ITcPlcIECProject) plcProject;
```

Code snippet (PowerShell):

```
$plcProject = $systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project")
```

The "iecProject" object can now be used to access the methods of the ITcPlcIECProject interface, e.g. to save the PLC project as a PLC library.

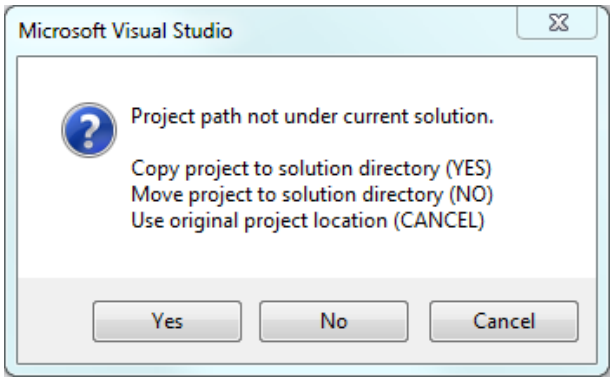
Opening existing PLC projects

To open an existing PLC-Project via Automation Interface, you need to navigate to the PLC node and then execute the CreateChild() method with the path to the corresponding PLC project file file as a parameter.

You can use three different values as SubType:

- 0: Copy project to solution directory
- 1: Move project to solution directory
- 2: Use original project location (when used, please use "" as project name parameter)

Basically, these values represent the functionalities (Yes, No, Cancel) from the following MessageBox in TwinCAT XAE:



In place of the template file you need to use the path to the PLC project (to its plcproj file) that needs to be added. As an alternative, you can also use a PLC project archive (tpzip file).

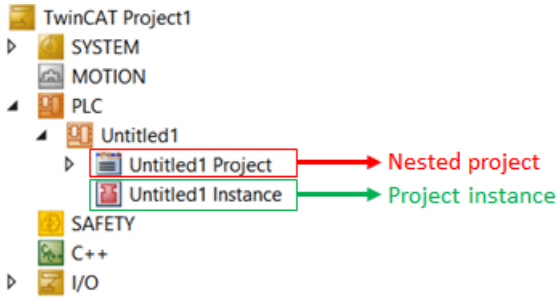
Code snippet (C#):

```
ITcSmTreeItem plc = systemManager.LookupTreeItem("TIPC");
ITcSmTreeItem newProject = plc.CreateChild("NameOfProject", 1, "", pathToProjectOrTpzipFile);
```

Code snippet (Powershell):

```
$plc = $systemManager.LookupTreeItem("TIPC")
$newProject = $plc.CreateChild("NameOfProject", 1, "", pathToProjectOrTpzipFile)
```

TwinCAT PLC projects consist of two different areas - the "Nested project" and the Project instance. The nested project (Tree Item subtype 56) contains the source code of the PLC program, whereas the project instance contains the declared input and output variables of the PLC program.



The following code snippet shows a common way to access both Tree Items in general if the full path name is not known.

Code snippet (C#):

```
ITcSmTreeItem plc = sysManager.LookupTreeItem("TIPC");
foreach (ITcSmTreeItem plcProject in plc)
{
ITcProjectRoot projectRoot = (ITcProjectRoot)plcProject;
ITcSmTreeItem nestedProject = projectRoot.NestedProject;
ITcSmTreeItem projectInstance = plcProject.get_Child(1);
}
```

Code snippet (PowerShell):

```
$plc = $sysManager.LookupTreeItem("TIPC")
ForEach( $plcProject in $plc)
{
    $nestedProject = $plcProject.NestedProject
    $projectInstance = $plcProject.get_Child(1)
}
```

Please note

A minimum TwinCAT 3.1 Build 4018 is required to access the ITcProjectRoot interface.

Saving the PLC project as a library

To save a PLC project as a PLC library, you need to make use of the ITcPlcIECProject [▶ 162]::SaveAsLibrary() [▶ 165] method.

Code snippet (C#):

```
iecProject.SaveAsLibrary(pathToLibraryFile, false);
```

Code snippet (Powershell):

```
$plcProject.SaveAsLibrary(pathToLibraryFile, $false)
```

The second parameter determines whether the library should be installed to the default repository after it has been saved as a file.

Handling online functionalities (Login, StartPlc, StopPlc, ResetCold, ResetOrigin)

Required version: TwinCAT 3.1 Build 4010 and above

The Automation Interface also provides you with PLC online features, for example to login to a PLC runtime and start/stop/reset the PLC program. These features can be accessed via the ITcSmTreeItem [▶ 120]::ProduceXml() [▶ 152] and ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] methods. These functions can be used on a ITcPlcIECProject [▶ 162] node.

XML structure:

```
<TreeItem>
<IECProjectDef>
<OnlineSettings>
<Commands>
<LoginCmd>false</LoginCmd>
<LogoutCmd>false</LogoutCmd>
<StartCmd>false</StartCmd>
<StopCmd>false</StopCmd>
</Commands>
</OnlineSettings>
</IECProjectDef>
</TreeItem>
```

Code snippet (C#):

```
string xml = "<TreeItem><IECProjectDef><OnlineSettings><Commands><LoginCmd>true</LoginCmd></
Commands></OnlineSettings></IECProjectDef></TreeItem>";
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project");
plcProject.ConsumeXml(xml);
```

Code snippet (Powershell):

The following table describes every XML node in more detail:

XML	Description
LoginCmd	true = in SPS-Laufzeit einloggen
LogoutCmd	true = aus SPS-Laufzeit ausloggen
StartCmd	true = Starten des aktuell in die Laufzeit geladenen SPS-Programms
StopCmd	true = Stoppen des aktuell in die Laufzeit geladenen SPS-Programms

Please note: In order to use commands like ResetOriginCmd, you must first execute a LoginCmd - similar to TwinCAT XAE.

Setting Boot project options

The following code snippet demonstrates how to use the ITcPlcProject interface to set Boot project options for a PLC project.

Code Snippet (C#):

```
ITcSmTreeItem plcProjectRoot = systemManager.LookupTreeItem("TIPC^PlcGenerated");
ITcPlcProject plcProjectRootIec = (ITcPlcProject) plcProjectRoot;
plcProjectRootIec.BootProjectAutostart = true;
plcProjectRootIec.GenerateBootProject(true);
```

Code snippet (Powershell):

```
$plcProject = $systemManager.LookupTreeItem("TIPC^PlcGenerated")
$plcProject.BootProjectAutostart = $true
$plcProject.GenerateBootProject($true)
```

Saving project and/or solution as archive

To save the whole TwinCAT solution in an ZIP compatible archive (*.tszip), the ITcSysManager9 interface may be used.

Code Snippet (C#):

```
ITcSysManager9 newSysMan = (ITcSysManager9)systemManager;
newSysMan.SaveAsArchive(@"C:\test.tszip");
```

Code snippet (Powershell):

```
$systemManager.SaveAsArchive("C:\test.tszip")
```

To reload a previously saved TSZIP file, the DTE method AddFromTemplate() may be used.

Code Snippet (C#):

```
dte.Solution.AddFromTemplate("C:\test.tszip",@"C:\tmp","CreatedFromTemplate");
```

Code snippet (Powershell):

```
$dte.Solution.AddFromTemplate("C:\test.tszip","C:\tmp","CreatedFromTemplate")
```

To save a specific PLC project in an ZIP compatible archive (*.tpzip), the method ITcSmTreeItem::ExportChild() may be used.

Code Snippet (C#):

```
ITcSmTreeItem plc= sysManager.LookupTreeItem("TIPC");
plc.ExportChild("PlcProject",@"C:\PlcTemplate.tpzip");
```

Code snippet (Powershell):

```
$plc = $systemManager.LookupTreeItem("TIPC")
$plc.ExportChild("PlcProject", "C:\PlcTemplate.tpzip")
```

To reload a previously saved TPZIP file, the ITcSmTreeItem::CreateChild() method may be used.

Code Snippet (C#):

```
plcConfig.CreateChild("PlcFromTemplate", 0, null, @"C:\PlcTemplate.tpzip");
```

Code snippet (Powershell):

```
$plc.CreateChild("plcFromTemplate", 0, $null, "C:\PlcTemplate.tpzip")
```

Calling CheckAllObjects()

To call the CheckAllObjects() method on the PLC Nested Project, you can use the corresponding method that is available in interface ITcPlcIECProject2.

Code snippet (C#):

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project");
ITcPlcIECProject2 iecProject = (ITcPlcIECProject2) plcProject;
iecProject.CheckAllObjects();
```

Code snippet (Powershell):

```
$plcProject = $systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project")
$plcProject.CheckAllObjects()
```

4.3.5.2 Accessing, creating and handling PLC-Libraries and -Placeholde

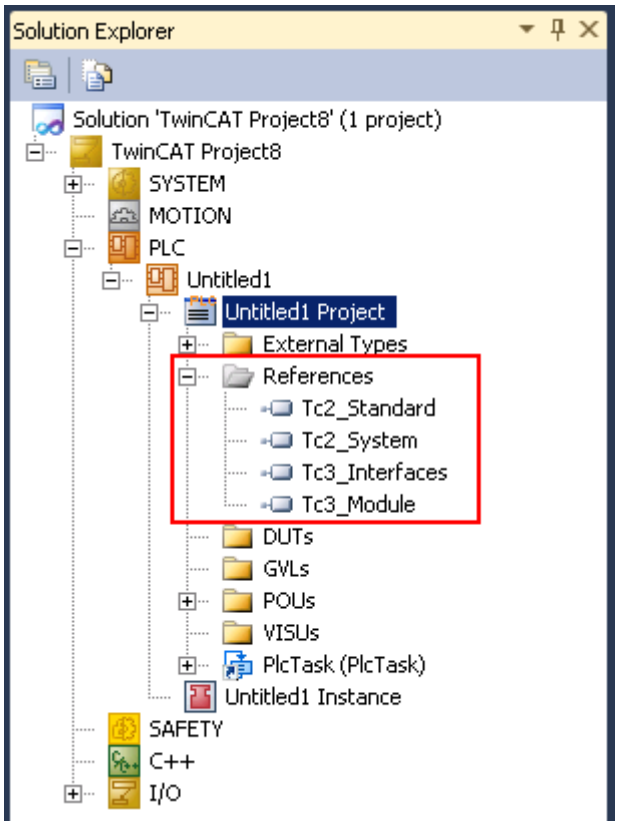
This chapter explains in-depth how to access and handle PLC libraries and PLC placeholders. The following list shows all chapters in this article:

- General information about PLC libraries and placeholders
- Navigating through references
- Adding references
- Removing references
- Scanning available libraries
- Freezing placeholder version
- Working with repositories

General information about PLC libraries and placeholders

There are two library object types in TwinCAT 3: libraries and placeholders. Further information on both types can be found in the TwinCAT 3 documentation on library management.

In a TwinCAT 3 PLC project, the references to libraries and placeholders are added as subordinate Tree Items to the reference node below the corresponding PLC project. If the "Standard PLC Project" template is chosen, certain libraries and placeholders are added to the project by default, e.g. Tc2_Standard, Tc2_System, … .



When using the Automation Interface, you can simply navigate through the reference node using the ITcSysManager [▶ 112]::LookupTreeItem() [▶ 120] method.

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
```

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
```

In order to handle this object correctly, it must be subjected to a type cast corresponding to the ITcPlcLibraryManager interface.

```
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
```

Please note that this step is not required in Windows PowerShell.

Navigating through references

You can run through all references using the ITcPlcLibraryManager [▶ 165]::References property. This property returns an ITcPlcReferences collection of either library objects (represented by ITcPlcLibrary [▶ 171]) or placeholder objects (represented by ITcPlcPlaceholderRef).

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
foreach (ITcPlcLibRef libRef in libManager.References)
{
if (libRef is ITcPlcLibrary)
{
ITcPlcLibrary library = (ITcPlcLibrary) libRef;
// do something
}
else if (libRef is ITcPlcPlaceholderRef)
{
ITcPlcPlaceholderRef placeholder = (ITcPlcPlaceholderRef) libRef;
// do something
}
}
```

The object libRef; which is used for the iteration, is of type ITcPlcLibRef. This is a common base class for ITcPlcLibrary and ITcPlcPlaceholderRef objects. In order to be able to work with one of these specific classes, we must subject the object to a corresponding type cast, as shown in the code snippet above.

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
ForEach( $libRef in $references )
{
  $libRef.LanguageIndependentName
}
```

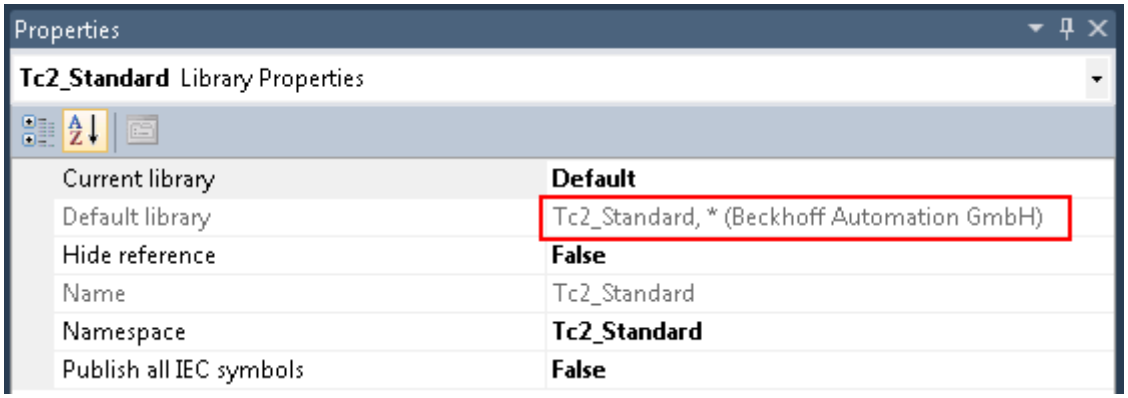
Adding references

The ITcPlcLibraryManager [▶ 165] class offers two methods that can be used to add a library or placeholder reference to a PLC project: AddLibrary() and AddPlaceholder().

A library can be added in two different ways:

- By specifying the name, version and distributor of the library
- By specifying the display name of the library
- In the case of a placeholder, by specifying the placeholder name.

The display name of a library can be defined in the properties window of the library or the placeholder:



Adding libraries:

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.AddLibrary("Tc2_MDP", "*", "Beckhoff Automation GmbH"); // name, version, distribution
list
libManager.AddLibrary("Tc2_Math, * (Beckhoff Automation GmbH)"); //monitored name
```

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.AddLibrary("Tc2_MDP", "*", "Beckhoff Automation GmbH")
$references.AddLibrary("Tc2_Math, * (Beckhoff Automation GmbH)")
```

A placeholder can be added in two different ways:

- By specifying the placeholder name, the name, the version and the distributor of the library
- By specifying the placeholder name if the placeholder already exists

Adding placeholders:

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.AddPlaceholder("Tc2_MC2_Camming"); // add existing place holder with name
libManager.AddPlaceholder("Placeholder_NC", "Tc2_NC", "*", "Beckhoff Automation GmbH");
```

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.AddPlaceholder("Tc2_MC2_Camming")
$references.AddPlaceholder("Placeholder_NC", "Tc2_NC", "*", "Beckhoff Automation GmbH")
```

Please note: When adding a new placeholder, the parameters of the AddPlaceholder() method determine its default resolution. To set the effective resolution, simply use the ITcPlcLibraryManager [▶ 165]::SetEffectiveResoltion() method.

Removing references

To remove a reference, simply use the ITcPlcLibraryManager [▶ 165]::RemoveReference() method. Because this method handles ITcPlcLibRef elements (which is the base class for ITcPlcLibrary and ITcPlcPlaceholderRef objects), you can use this method for both library and placeholder references.

Library references can be removed either by specifying their name, version and distributor or by specifying their display name.

Placeholder references can be removed by specifying their placeholder name.

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.RemoveReference("Tc2_Math"); // delete library
libManager.RemoveReference("Placeholder_NC"); // delete a placeholder
```

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.RemoveReference("Tc2_Math")
$references.RemoveReference("Placeholder_NC")
```

Scanning for available libraries

To scan the system for all available PLC libraries, simply use the ITcPlcLibraryManager [▶ 165]::ScanLibraries() [▶ 169] method. This method returns an ITcPlcReferences collection of libraries (type ITcPlcLibrary).

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
ITcPlcReferences libraries = libManager.ScanLibraries();
foreach(ITcPlcLibrary library in libraries)
{
// do something
}
```

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$libraries = $references.ScanLibraries()
ForEach( $lib in $libraries )
{
  $lib.Name
}
```

Freezing the placeholder version

It is possible to freeze the used version of a placeholder to a specific version. This can be achieved with the ITcPlcLibraryManager [▶ 165]::FreezePlaceholder() method. This method is called on an object that points to the reference node.

Code snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.FreezePlaceholder(); // freezes the version of all place holders
libManager.FreezePlaceholder("Placeholder_NC"); // freezes the version of a specific place holder
```

Code snippet (PowerShell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.FreezePlaceholder()
$references.FreezePlaceholder("Placeholder_NC")
```

Please note: The version is frozen with the effective resolution. If the effective resolution points to "*", then the latest version is used in the system.

Working with repositories

The Automation Interface provides method for handling PLC library repositories. A default repository is part of every TwinCAT installation. To create additional repositories, you can use the ITcPlcLibraryManager [▶ 165]::InsertRepository() [▶ 168] method.

Code Snippet (C#):

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.InsertRepository("TestRepository", @"C:\Temp", 0);
```

Code Snippet (Powershell):

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.InsertRepository("TestRepository", "C:\Temp", 0)
```

When installing a new library into the system, the library needs to be part of a repository. This insertion can be achieved by using the ITcPlcLibraryManager [▶ 165]::InstallLibrary() [▶ 168] method.

Code Snippet (C#):

```
libManager.InstallLibrary("TestRepository", @"C:\SomeFolder\TcTestLibrary.library", false);
```

Code Snippet (Powershell):

```
$references.InstallLibrary("TestRepository", "C:\SomeFolder\TcTestLibrary.library", $false)
```

To uninstall a library from the repository, use the ITcPlcLibraryManager [▶ 165]::UninstallLibrary() [▶ 170] method.

Code Snippet (C#):

```
libManager.UninstallLibrary("TestRepository", "Tc2_MDP", "*", "Beckhoff Automation GmbH");
```

Code Snippet (Powershell):

```
$references.UninstallLibrary("TestRepository", "Tc2_MDP", "*", "Beckhoff Automation GmbH")
```

To remove a repository, use the ITcPlcLibraryManager [▶ 165]::RemoveRepository() [▶ 169] method.

Code Snippet (C#):

```
libManager.RemoveRepository("TestRepository");
```

Code Snippet (Powershell):

```
$references.RemoveRepository("TestRepository")
```

4.3.5.3 Accessing, creating and handling PLC POUs

This chapter explains in-depth how to access PLC objects, for example POUs, Methods, Transitions, Properties, and how to access their corresponding implementation and declaration areas to handle PLC code. It also covers how to import/export PLC objects in PLCopen XML. The following list shows all chapters in this article:

- General information about PLC objects
- Accessing the implementation / declaration area of a POU
- Accessing Sub-POUs (Actions, Properties, ...)
- Creating PLC objects
- PLC access modifier
- Importing / Exporting PLCopen XML
- Import existing POUs (templates)

General information about PLC objects

Although every tree item is considered to be of type ITcSmTreeItem [▶ 120], some items need to be casted to a more special interface to gain access to all of their methods and properties, for example POUs which need to be casted to the interface ITcPlcPou [▶ 159] to get access to their unique methods and properties.

Code snippet (C#):

```
ITcSmTreeItem plcPousItem = systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated
Project^POUs");
ITcSmTreeItem newFb = plcPousItem.CreateChild("FB_TEST", 604, "", IECLANGUAGETYPES.IECLANGUAGE_ST);
ITcPlcPou fbPou = (ITcPlcPou)newFb;
```

Code snippet (Powershell):

```
$plcPousItem = $systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated Project^POUs")
$newFb = $plcPousItem.CreateChild("FB_TEST", 604, "", 6)
```

In this example, the POU *MAIN* is being created in a PLC-Project. The object *programPou* references this POU and can now be used to access specific methods and properties of the POU by using the corresponding method/property of the ITcPlcPou interface.

Accessing the implementation / declaration area of a POU

You can also gain read/write access to either the implementation or declaration area of a POU, using the ITcPlcImplementation [▶ 161] or ITcPlcDeclaration [▶ 160] interfaces, as shown in the following example.

Code snippet (C#):

```
ITcPlcDeclaration fbPouDecl = (ITcPlcDeclaration) fbPou;
ITcPlcImplementation fbPouImpl = (ITcPlcImplementation) fbPou;
string decl = fbPouDecl.DeclarationText;
string impl = fbPouImpl.ImplementationText;
string implXml = fbPouImpl.ImplementationXml;
```

Code snippet (PowerShell):

```
$decl = $newFb.DeclarationText
$impl = $newFb.ImplementationText
$implXml = $newFb.ImplementationXml
```

When comparing the two interfaces, you will notice that the accessible properties differ in both interfaces. For example, the ITcPlcImplementation interface offers a "Language" property that ITcPlcDeclaration does not. The reason for this is that, according to IEC, the declaration area is not based on a real programming language (e.g. ST), so this property would not make sense if it were used in the declaration area.

Accessing Sub-POUs (Actions, Properties, ...)

POUs may have more sub-items like Methods, Transitions, Actions and Properties. It is important to understand that not every sub-item of a POU also has both an implementation and a declaration area. Actions and Transitions, for example, only have an implementation area. Therefore, casting to one of the interfaces mentioned above is only valid if the corresponding sub-object has this area. The following table gives an overview about which areas are available in the different types of POUs.

Tree Item	Type	Declaration Area	Implementation Area
Program	POU	Yes	Yes
Function	POU	Yes	Yes
Function Block	POU	Yes	Yes
Action	POU	No	Yes
Method	POU	Yes	Yes
Property (Get/Set)	POU	Yes	Yes
Transition	POU	No	Yes
Enum	DUT	Yes	No
Struct	DUT	Yes	No
Union	DUT	Yes	No
Alias	DUT	Yes	No
Interface	Interface	Yes	No
Property (Get/Set)	Interface	Yes	Yes
Global variables	GVL	Yes	No

Creating PLC objects

The creation of PLC objects is simple and can be carried out using the CreateChild() [▶ 154] method of the ITcSmTreeItem [▶ 120] interface. The parameters of this method must be interpreted differently depending on the POU type. In the following table you will find the information required to create different POU types. Please note that the vInfo parameter may be a string if more than one parameter is required. In [...] reference is made to each array position and whether it is optional or not.

Tree Item	Type	Parameter "nSubType"	Parameter "vInfo"
Program	POU	602	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Can be used for derivation (keywords "Extends" or "Implements"). If the keywords "Extends" AND "Implements" are to be used, the keyword "Extends" is specified in this field. **[2, optional]:** Name of the interface or POU to be extended/implemented (mandatory if [1] is used). If the keywords "Extends" AND "Implements" are to be used, the library to be extended is specified in this field. **[3, optional]:** If the keywords "Extends" AND "Implements" are to be used, the keyword "Implements" is specified in this field. **[4, optional]:** If the keywords "Extends" AND "Implements" are to be used, the interface used for the derivation is specified in this field.
Function	POU	603	**[0]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. **[1]:** Return type of the function. Can be a PLC data type, e.g. DINT, BOOL, …
Function block	POU	604	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Can be used for derivation (keywords "Extends" or "Implements"). If the keywords "Extends" AND "Implements" are to be used, the keyword "Extends" is specified in this field. **[2, optional]:** Name of the interface or POU to be extended/implemented (mandatory if [1] is used). If the keywords "Extends" AND "Implements" are to be used, the library to be extended is specified in this field. **[3, optional]:** If the keywords "Extends" AND "Implements" are to be used, the keyword "Implements" is specified in this field. **[4, optional]:** If the keywords "Extends" AND "Implements" are to be used, the interface used for the derivation is specified in this field.
Action	POU	608	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Can contain PLCopen XML string with code for the action if necessary
Method	POU	609	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Return type **[2, optional]:** Access specifier, e.g. PUBLIC. PUBLIC is used by default. **[3, optional]:** May contain PLCopen XML string with code for the action if necessary
Property	POU	611	**[0]:** IEC programming language as defined by IECLANGUAGETYPES [▶ 160] **[1]:** Return type **[2, optional]:** Access specifier, e.g. PUBLIC. PUBLIC is used by default.

Tree Item	Type	Parameter "nSubType"	Parameter "vInfo"
Retrieve property	POU	613	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Access specifier, e.g. PUBLIC. PUBLIC is used by default. **[2, optional]:** May contain PLCopen XML string with code for the action if necessary
Set property	POU	614	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Access specifier, e.g. PUBLIC. PUBLIC is used by default. **[2, optional]:** May contain PLCopen XML string with code for the action if necessary
Transition	POU	616	**[0, optional]:** IEC programming language, as defined by IECLANGUAGETYPES [▶ 160]. ST (Structured Text) is used by default. **[1, optional]:** Can contain PLCopen XML string with code for the action if necessary
UML class diagram	POU	631	No vInfo parameter required, "Null" can be used.
Enum	DUT	605	**[0, optional]:** May contain declaration text if applicable
Struct	DUT	606	**[0, optional]:** May contain declaration text if applicable
Union	DUT	607	**[0, optional]:** May contain declaration text if applicable
Alias	DUT	623	**[0, optional]:** May contain declaration text if applicable
Interface	Interface	618	**[0, optional]:** Extend type
Property	Interface	612	**[0]:** Return type
Retrieve property	Interface	654	No vInfo parameter required, "Null" can be used.
Set property	Interface	655	No vInfo parameter required, "Null" can be used.
Method	Interface	610	**[0]:** Return type
Global variables	GVL	615	**[0, optional]:** May contain declaration text if applicable
PLC folder	Folder	601	No vInfo parameter required, "Null" can be used.
Parameter list	PL	629	**[0, optional]:** May contain declaration text if applicable
Visualization	Visualization	619	No vInfo parameter required, "Null" can be used.

Example: The following code snippet shows how the vInfo parameter can be used to create a function block "FB_Test", with the keywords "Extends" and/or "Implements", depending on the value of the Boolean variables *bExtends* and *bImplements*. The extended library is *ADSRDSTATE* and the implemented interface is *ITcADI*.

Code snippet (C#):

```
string[] vInfo;
bool bExtends = true;
bool bImplements = true;

if (bExtends && bImplements)
{
  vInfo= new string[5];
}
else
{
```

```
  if (bExtends || bImplements)
  {
    vInfo= new string[3];
  }
  else
  {
    vInfo= new string[1];
  }
}
vInfo[0] = language.AsString();
if (bExtends && bImplements)
{
  vInfo[1] = "Extends";
  vInfo[2] = "ADSRDSTATE";
  vInfo[3] = "Implements";
  vInfo[4] = "ITcADI";
}
else
{
  if (bExtends)
  {
    vInfo[1] = "Extends";
    vInfo[2] = "ADSRDSTATE";
  }
  else
  {
    if (bImplements)
    {
      vInfo[1] = "Implements";
      vInfo[2] = "ITcADI";
    }
  }
}

ITcSmTreeItem newPOU = parent.CreateChild("FB_Test", 604,
"", fbVInfo);
```

Please note

If the parameter vInfo only contains one value (marked with [0] in the table above), then you should not create an array of size 1, but simply a normal variable. Example: When using nSubType 618 (interface), you should use vInfo as follows.

Code snippet (C#):

```
ITcSmTreeItem interface1 = pou.CreateChild("NewInterface1", 618, "", null); // no expansion type
ITcSmTreeItem interface2 = pou.CreateChild("NewInterface2", 618, "", "ITcUnknown"); // expands
ITcUnknown interface
```

PLC access modifier

The following access modifiers are valid and may be used as a vInfo parameter, as shown in the table above: PUBLIC, PRIVATE, PROTECTED, INTERNAL.

PLCopen XML import/export

You can also import/export PLC elements in PLCopen XML via the ITcPlcIECProject [▶ 162] interface. Methods and properties of this interface can only be executed directly on a PLC project node, e.g.:

Code snippet (C#):

```
ITcSmTreeItem plcProject =
systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated
Project");
ITcPlcIECProject importExport = (ITcPlcIECProject)
plcProject;
importexport.PlcOpenExport(plcOpenExportFile,
"MyPous.POUProgram;MyPous.POUFunctionBlock");
importExport.PlcOpenImport(plcOpenExportFile,
(int)PLCIMPORTOPTIONS.PLCIMPORTOPTIONS_NONE);
```

Code snippet (PowerShell):

```
$plcProject = $systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated
Project")
$plcProject.PlcOpenExport(plcOpenExportFile, "MyPous.POUProgram;MyPous.POUFunctionBlock")
$plcProject.PlcOpenImport(plcOpenExportFile,0)
```

This code snippet assumes that a PLC project has already been added to the TwinCAT configuration, which is referenced via the *plcProject* object. This reference is converted into the object *importexport* of type ITcPlcIECProject, which is then used to export the POUs POUProgram and POUFunctionBlock (both are located in the PLC folder "MyPOUs") to an XML file and then import them again.

The available options for the import are: None (0), Rename (1), Replace (2), Skip (3)

Import existing POUs (templates)

PLC templates are available in two units: You can either use the complete PLC project as a whole or each POU individually as a template. The later is covered in this chapter. One of the reasons a developer may choose individual POUs as templates, is that it is easier to build a pool of existing functionalities and encapsulate them in separate POUs, e.g. different function blocks covering different sorting algorithms. Upon project creation, the developer simply chooses which functionality he needs in his TwinCAT project and accesses the template tool to retrieve the corresponding POU and import it to his PLC project.

Code snippet (C#):

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated Project");
plcProject.CreateChild("NameOfImportedPOU", 58, null, pathToPouFile);
plcProject.CreateChild(null, 58, null, stringArrayWithPathsToPouFiles);
```

Code snippet (Powershell):

```
$plcProject = $systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated Project")
$plcProject.CreateChild("NameOfImportedPOU", 58, $null, $pathToPouFile)
$plcProject.CreateChild($null, 58, $null, $stringArrayWithPathsToPouFiles)
```

Please note: A POU template file may not only be a .TcPou file but also the corresponding files for DUTs and/or GVLs. The example above demonstrates two common ways to import POU template files. The first is to import a single file, the second to import multiple files at once by storing the file paths to a string array and using this string array as the vInfo parameter of CreateChild().

4.3.6 I/O

4.3.6.1 Creating and handling EtherCAT devices

This article explains how to build an EtherCAT topology via TwinCAT Automation Interface. It consists of the following topics:

- General information
- Creating an EtherCAT device
- Creating EtherCAT boxes and insert into topology
- Creating EtherCAT terminals and insert into topology
- Exceptions to the ItemSubType 9099
- Changing the "Previous Port" of an EtherCAT terminal
- Adding EtherCAT slaves to a HotConnect group
- How to add EtherCAT SyncUnits
- Handle EtherCAT junction boxes (CU1128)

All of these topics cover the case of an **offline** configuration, which means the real addresses of all devices ar not know at the time of config creation. The last chapter of this article therefore also explains how to

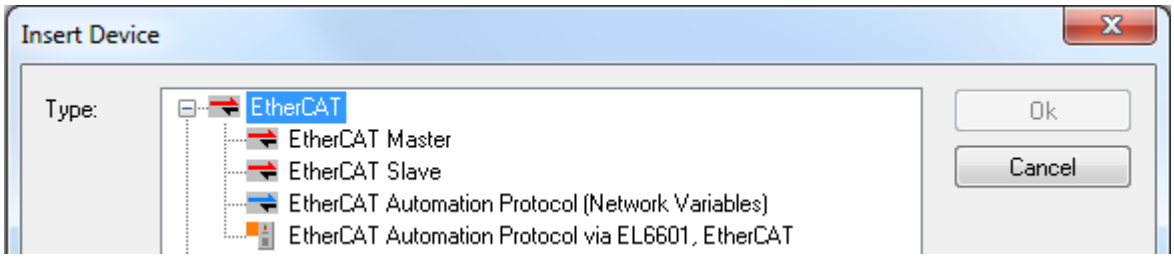
- Activate the configuration

General information

This documentation should give you an overview avout how to create and handle EtherCAT devices and their corresponding topology via Automation Interface. For an in-depth understanding about how EtherCAT works and how it is generally integrated in TwinCAT SystemManager/XAE, please consult the EtherCAT System Documentation and the corresponding chapter about EtherCAT configuration in TwinCAT. EtherCAT boxes and terminals, which are connected to an EtherCAT Master, share a common way of creation, which is explained in a separate article about E-Bus sub types [▶ 125].

Creating an EtherCAT device

The first step in creating a TwinCAT EtherCAT configuration is to create an EtherCAT device, which may involve creating an EtherCAT master, slave or automation protocol (e.g. to use network variables, as described in a separate article [▶ 73]). To create an EtherCAT master/slave, simply use the ITcSmTreeItem [▶ 154]::CreateChild() [▶ 154] method with the corresponding parameter for SubType (EtherCAT master = 111, EtherCAT slave = 130).



EtherCAT master - code snippet (C#)

```
ITcSmTreeItem devices = systemManager.LookupTreeItem("TIID");
ethercatMaster = devices.CreateChild("EtherCAT Master", 111, null, null);
```

EtherCAT master - code snippet (PowerShell)

```
$devices = $systemManager.LookupTreeItem("TIID")
$ethercatMaster = $devices.CreateChild("EtherCAT Master", 111, $null, $null)
```

EtherCAT slave - code snippet (C#)

```
ITcSmTreeItem devices = systemManager.LookupTreeItem("TIID");
ethercatSlave = devices.CreateChild("EtherCAT Slave", 130, null, null);
```

EtherCAT slave - code snippet (PowerShell)

```
$devices = $systemManager.LookupTreeItem("TIID")
$ethercatSlave = $devices.CreateChild("EtherCAT Slave", 130, $null, $null)
```

Creating EtherCAT Boxes

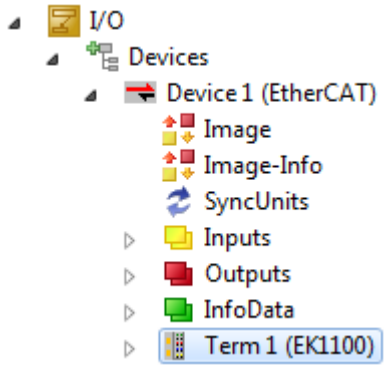
The second step involves the creation of EtherCAT Boxes, for example an EK1100 EtherCAT Coupler. As explained in the article on E-bus SubTypes [▶ 125], all subordinate Tree Items (there are a few exceptions, see below) of an EtherCAT master use a common SubType (9099) and they are identified via the product revision, which must be passed as the vInfo parameter of the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method.

Code snippet (C#)

```
ITcSmTreeItem ethercatMaster = systemManager.LookupTreeItem("TIID^EtherCAT Master");
ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100-0000-0017");
```

Code snippet (PowerShell)

```
$ethercatMaster = $systemManager.LookupTreeItem("TIID^EtherCAT Master")
$ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100-0000-0017")
```



Please note: In addition to a complete product revision, you can also use a placeholder. If you only specify "EK1100" as vInfo, Automation Interface automatically captures the latest revision number and uses it. Example:

Code snippet (C#):

```
ITcSmTreeItem ethercatMaster = systemManager.LookupTreeItem("TIID^EtherCAT Master");
ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100");
```

Code snippet (PowerShell):

```
$ethercatMaster = $systemManager.LookupTreeItem("TIID^EtherCAT Master")
$ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100")
```

Creating EtherCAT Terminals and inserting them into the topology

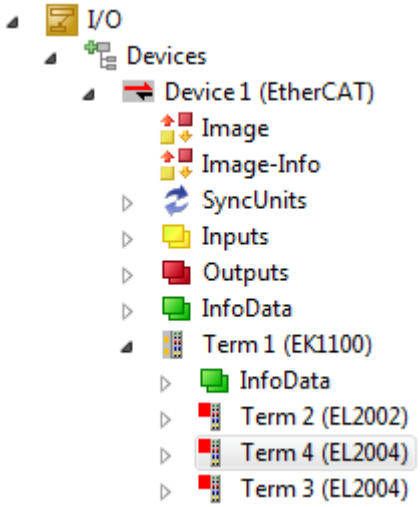
The creation of EtherCAT Terminals is based on the same concepts as the creation of EtherCAT Boxes. All terminals also share a common SubType and are identified by the product revision, which must be passed as a vInfo parameter to the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method. The parameter bstrBefore allows you to determine the position at which the terminal is inserted into the configuration.

Code snippet (C#):

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100");
ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017");
```

Code snippet (PowerShell):

```
$ek1100 = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100")
$ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017")
```



Please note: If problems occur when using the bstrBefore parameter, please try inserting the terminal at a "device level", for example:

Code snippet (C#):

```
ITcSmTreeItem device= systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)");
device.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017");
```

Code snippet (PowerShell):

```
$device= $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)")
$device.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017")
```

The new terminal is then inserted before "Term 3 (EL2004)" under the last EtherCAT Box inserted.

Please note: In addition to a complete product revision, you can also use a placeholder. If you only specify "EL2004" as vInfo, Automation Interface automatically captures the latest revision number and uses it. Example:

Code snippet (C#):

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100");
ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004");
```

Code snippet (PowerShell):

```
$ek1100 = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100")
$ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004")
```

Exceptions to the ItemSubType 9099

There are a few exceptions to the ItemSubType 9099, e.g. the RS232 terminals EP6002 (ItemSubType 9101) and EL600X (ItemSubType 9101). The following table gives an overview about all exceptions and their corresponding ItemSubType.

I/O	ItemSubType
EP6002	9101
EL6001	9101
EL6002	9101
EP6001-0002	9101
EP6002-0002	9101
EL6021	9103
EL6022	9103
EL6021-0021	9103
BK1120	9081
ILXB11	9086
EL6731	9093
EL6751	9094
EL6752	9095
EL6731-0010	9096
EL6751-0010	9097
EL6752-0010	9098
EL6601	9100
EL6720	9104
EL6631	9106
EL6631-0010	9107
EL6632	9108
EL6652-0010	9109
EL6652	9110

Changing the "Previous Port" of an EtherCAT Terminal

The previous connection of an EtherCAT Terminal determines the position of the terminal within the EtherCAT topology.

In TwinCAT XAE, the drop-down list box automatically includes all available Previous Ports. To configure this setting via the Automation Interface, you can use the methods ITcSmTreeItem [▶ 120]::ProduceXml() [▶ 152] and ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] to edit the XML description [▶ 24] of the corresponding

EtherCAT Terminal. The XML description contains one or more XML nodes <PreviousPort>, where its (their) attribute "Selected=1" defines which previous port is currently selected. Each previous connection device is identified by its physical address.

Example (XML description)

```
<TreeItem>
  <EtherCAT>
    <Slave>
      <PreviousPort Selected="1">
        <Port>B</Port>
        <PhysAddr>1045</PhysAddr>
      </PreviousPort>
      <PreviousPort>
        <Port>B</Port>
        <PhysAddr>1023</PhysAddr>
      </PreviousPort>
    </Slave>
  <EtherCAT>
</TreeItem>
```

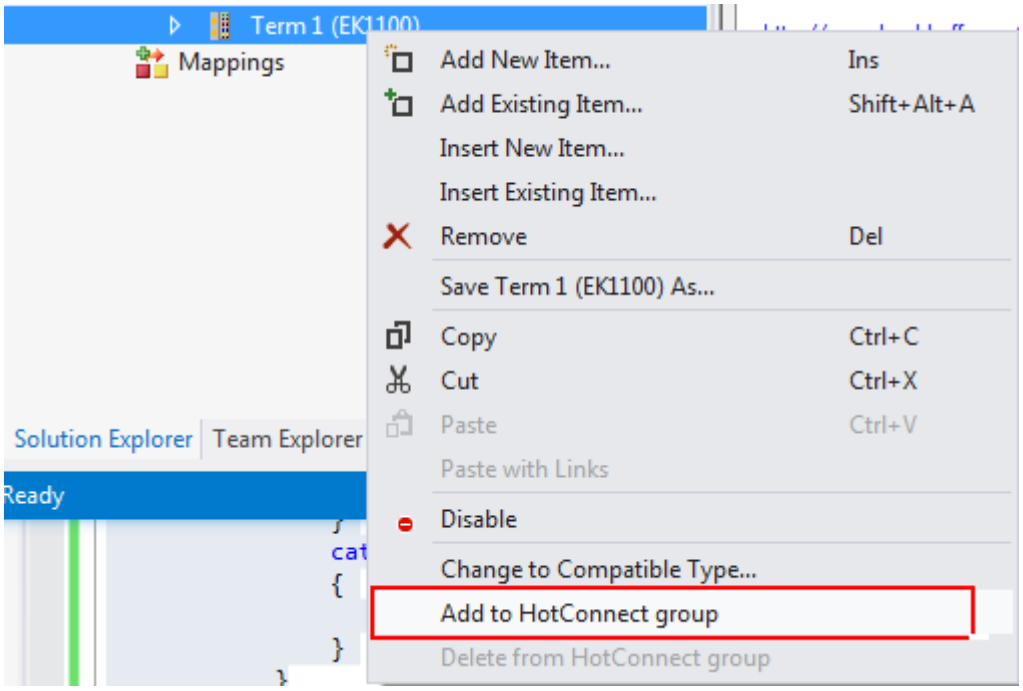
If you want to change the previous connection, you must know the <PhysAddr> of the desired device, which can also be determined via the XML description.

When inserting children into an EtherCAT configuration, the parameter bstrBefore of the ImportChild or CreateChild method can be used to specify the previous element when calling the ImportChild or CreateChild method on a slave. On an EtherCAT master, this setting must be specified via the XML description from above.

Adding EtherCAT slaves to a HotConnect group

EtherCAT HotConnect allows pre-configured sections to be added to or removed from the data traffic before the system is started or during operation. Further information about EtherCAT HotConnect can be found in our EtherCAT system documentation.

In TwinCAT XAE, an EtherCAT slave can be added to a HotConnect group by clicking on the corresponding option in the context menu of the device.



In the TwinCAT Automation Interface, the same can be achieved by using the following XML structure on the EtherCAT slave:

Example (XML description):

```
<TreeItem>
<EtherCAT>
<Slave>
```

```
<HotConnect>
<GroupName>Term 1 (EK1101)</GroupName>
<GroupMemberCnt>4</GroupMemberCnt>
<IdentifyCmd>
<Comment>HotConnect-Identität lesen</Comment>
<Requires>cycle</Requires>
<Cmd>1</Cmd>
<Adp>0</Adp>
<Ado>4096</Ado>
<DataLength>2</DataLength>
<Cnt>1</Cnt>
<Retries>3</Retries>
<Validate>
<Data>0000</Data>
<Timeout>5000</Timeout>
</Validate>
</IdentifyCmd>
</HotConnect>
</Slave>
</EtherCAT>
</TreeItem>
```

Please note:

- The <GroupMemberCnt> tag must specify the exact number of terminals plus the device itself.

How to set EtherCAT Sync Units

EtherCAT SyncUnits can be set via ITcSmTreeItem::ConsumeXml() and by using the following XML description.

Example (XML description):

```
<TreeItem>
  <EtherCAT>
    <Slave>
      <SyncUnits>
        <SyncUnit>SyncUnit1</SyncUnit>
      </SyncUnits>
    </Slave>
  </EtherCAT>
</TreeItem>
```

Handle EtherCAT junction boxes (CU1128)

EtherCAT junction boxes can be handled similar to all other tree items. The following sample code shows how to add a CU1128 box and then add two EK1100 boxes below:

Code Snippet (C#)

```
ITcSmTreeItem cu1128 = ethercatMaster.CreateChild("CU1128", 9099, null, "CU1128");
ITcSmTreeItem cu1128_devA = cu1128.get_Child(1);
ITcSmTreeItem cu1128_devB = cu1128.get_Child(2);
ITcSmTreeItem ek1100_1 = cu1128_devA.CreateChild("EK1100-1", 9099, null, "EK1100");
ITcSmTreeItem ek1100_2 = cu1128_devB.CreateChild("EK1100-2", 9099, null, "EK1100");
```

Code Snippet (Powershell)

```
$cu1128 = $ethercatMaster.CreateChild("CU1128", 9099, $null, "CU1128")
$cu1128_devA = $cu1128.get_Child(1)
$cu1128_devB = $cu1128.get_Child(2)
$ek1100_1 = $cu1128_devA.CreateChild("EK1100-1", 9099, $null, "EK1100")
$ek1100_2 = $cu1128_devB.CreateChild("EK1100-2", 9099, $null, "EK1100")
```

Activate the EtherCAT configuration

To activate a created TwinCAT configuration via Automation Interface, the ITcSysManager [▶ 112]::ActivateConfiguration() [▶ 115] method may be used. However, the previous chapters only explained how to create an offline configuration, which means that every device created does not have real addresses, e.g. the EtherCAT Master has not been linked to a physical network interface card yet. Before activating the configuration you therefore need to configure each device with online addresses. To determine the real addresses, you can run a ScanDevices on the online system, determine the real addresses via the XML

description (ITcSmTreeItem [▶ 120]::ProduceXml() [▶ 152]) and then import the address information via ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] into the created (offline) configuration. There are two HowTo samples which help you exactly with this kind of configuration:

- Scan Devices [▶ 89] via Automation Interface

4.3.6.2 Creating and handling network variables

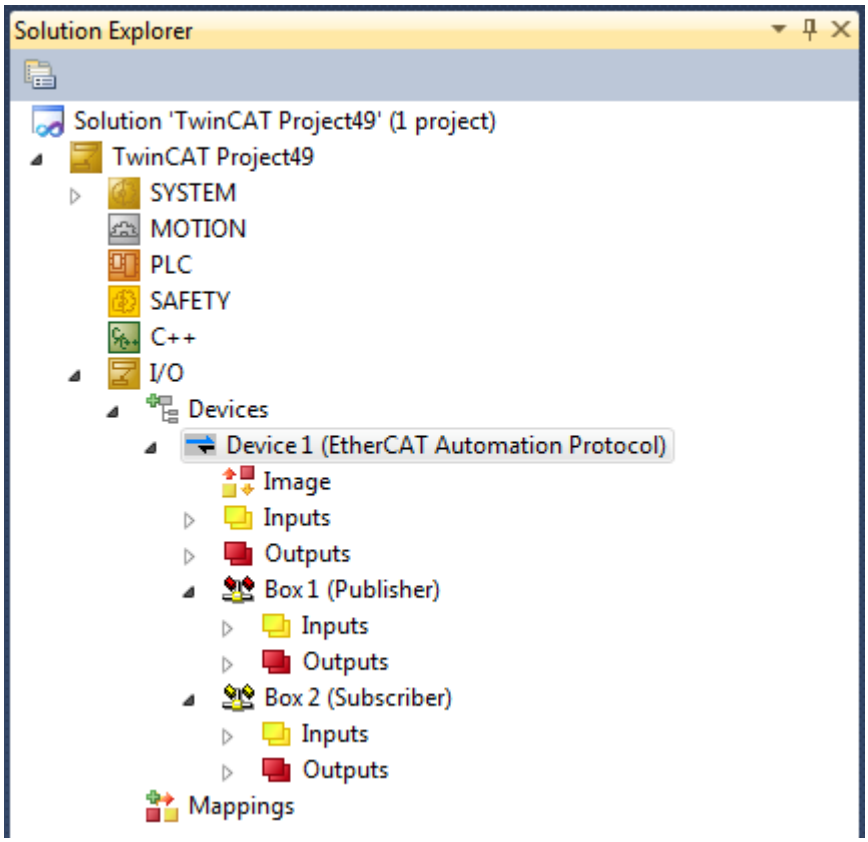
This chapter explains in-depth how to create and handle network variables. The following list shows all chapters in this article:

- General information about network variables
- Creating an EtherCAT Automation Protocol device
- Creating a Publisher box
- Creating a Subscriber box
- Setting parameters for a Publisher/Subscriber box
- Creating Publisher variables
- Creating Subscriber variables
- Linking Publisher/Subscriber variables
- Reading Publisher/Subscriber variable IDs

Please note that the Scripting Container contains a detailed sample about how to create and configure network variables with the Automation Interface.

General information about network variables

Network variables can be used to exchange data between two TwinCAT devices via an IP-based network. One device declares variables as "Publisher" (sender) and the other device receives variable values as "Subscriber". For this reason, we also speak of Publisher/Subscriber variables. TwinCAT offers you the flexibility to configure network variables directly within a TwinCAT project so that you can map them to your PLC or I/O.



Network variables use the EtherCAT Automation Protocol device for communication over the local network. For this reason, you must add this device before you can configure a Publisher and/or Subscriber Box together with the corresponding variables.

Further information about network variables and how to configure them in TwinCAT can be found here.

Creating an EtherCAT Automation Protocol device

To create the EtherCAT Automation Protocol device, you can use the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method together with the corresponding SubType of this device (112).

Code snippet (C#):

```
ITcSmTreeItem devicesNode = systemManager.LookupTreeItem("TIID");
device = devicesNode.CreateChild("Device 1 (EtherCAT Automation Protocol)", 112, null, null);
```

Code snippet (PowerShell):

```
$devicesNode = $systemManager.LookupTreeItem("TIID")
$device = $devicesNode.CreateChild("Device 1 (EtherCAT Automation Protocol)", 112, $null, $null)
```

Creating a Publisher Box

The Publisher Box is the container for Publisher variables that determine, for example, which communication pattern type is to be used for the variables it contains (unicast, multicast, broadcast). To add a Publisher Box, simply use the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method again together with the corresponding SubType (9051).

Code snippet (C#):

```
ITcSmTreeItem eapDevice = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)");
pubBox = eapDevice.CreateChild("Box 1 (Publisher)", 9051, null, null);
```

Code snippet (PowerShell):

```
$eapDevice = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)")
$pubBox = $eapDevice.CreateChild("Box 1 (Publisher)", 9051, $null, $null)
```

The small code snippet adds a Publisher Box to the previously created EtherCAT Automation Protocol device. To configure the communication pattern of this box, you must customize its settings using the ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] method. This is described in more detail in the EtherCAT Automation Protocol example of the Scripting Container or further down on this page.

Creating a Subscriber Box

The Subscriber Box is the container for Subscriber variables that determine, for example, which communication pattern type is to be used for the variables it contains (unicast, multicast, broadcast). To add a Publisher Box, simply use the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method again together with the corresponding SubType (9052).

Code snippet (C#):

```
ITcSmTreeItem eapDevice = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)");
subBox = eapDevice.CreateChild("Box 1 (Subscriber)", 9052, null, null);
```

Code snippet (PowerShell):

```
$eapDevice = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)")
$subBox = $eapDevice.CreateChild("Box 1 (Subscriber)", 9052, $null, $null)
```

The small code snippet adds a Subscriber Box to the previously created EtherCAT Automation Protocol device. To configure the communication pattern of this box, you must customize its settings using the ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] method. This is described in more detail in the EtherCAT Automation Protocol example of the Scripting Container or further down on this page.

Creating Publisher variables

After you have successfully added a Publisher Box, you can now add Publisher variables to this box by using the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method together with the required parameters for SubType (0) and vInfo, which define the data type of the Publisher variables, e.g. "BOOL".

Code snippet (C#):

```
ITcSmTreeItem pubBox = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)^Box 1 (Publisher)");
pubVar = pubBox.CreateChild("MAIN.bTestVar", 0, null, "BOOL");
```

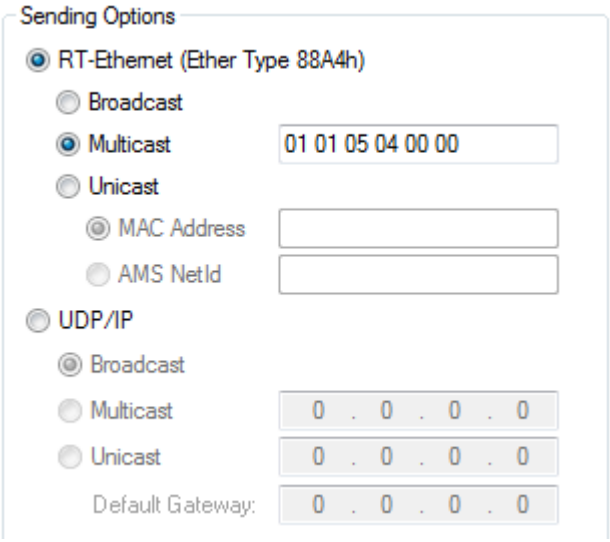
Code snippet (PowerShell):

```
$pubBox = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)^Box 1
(Publisher)")
$pubVar = $pubBox.CreateChild("MAIN.bTestVar", 0, $null, "BOOL")
```

See also the scripting container example "EtherCAT Automation Protocol" for more information.

Setting parameters for a Publisher/Subscriber Box

Two communication patterns must be configured on a Publisher and/or Subscriber Box: **RT-Ethernet** or **UDP/IP** . The following screenshot shows the corresponding TwinCAT XAE configuration tab.



You can find more detailed information about these options here.

To configure the box for **RT-Ethernet** , you must use the ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] method to import the following XML structure:

```
<TreeItem>
<BoxDef>
<FieldbusAddress>1</FieldbusAddress>
<AmsAddress>
<AmsPort>0</AmsPort>
<AmsPortTimeout>5</AmsPortTimeout>
</AmsAddress>
<NvPubDef>
<Udp Enabled="false"/>
<MacAddress>00 00 00 00 00 00</MacAddress>
<IoDiv>
<Divider>1</Divider>
<Modulo>0</Modulo>
</IoDiv>
<VLAN>
<Enable>false</Enable>
<Id>0</Id>
<Prio>0</Prio>
</VLAN>
<ArpInterval>1000</ArpInterval>
<DisableSubscriberMonitoring>false</DisableSubscriberMonitoring>
<TargetChangeable>false</TargetChangeable>
</NvPubDef>
</BoxDef>
```

```
</TreeItem>
```

The following table shows how the nodes marked **in bold** must be adapted according to the desired communication pattern.

RT-Ethernet communication pattern	<PublisherNetId>	<MacAddress>
Broadcast	0.0.0.0.0.0	FF FF FF FF FF FF
Multicast	0.0.0.0.0.0	Must contain a multicast MAC address, see here for more information.
Unicast - MAC address	0.0.0.0.0.0	Must contain a unicast MAC address, see here for more information.
Unicast - AmsNetId	Contains AmsNetId of target computer.	00 00 00 00 00 00

Import the following XML structure if you want to use **UDP/IP** :

```
<TreeItem>
<BoxDef>
<FieldbusAddress>1</FieldbusAddress>
<AmsAddress>
<AmsPort>0</AmsPort>
<AmsPortTimeout>5</AmsPortTimeout>
</AmsAddress>
<NvPubDef>
<Udp Enabled="true">
<Address>0.0.0.0</Address>
<Gateway>0.0.0.0</Gateway>
</Udp>
<IoDiv>
<Divider>1</Divider>
<Modulo>0</Modulo>
</IoDiv>
<VLAN>
<Enable>false</Enable>
<Id>0</Id>
<Prio>0</Prio>
</VLAN>
<ArpInterval>1000</ArpInterval>
<DisableSubscriberMonitoring>false</DisableSubscriberMonitoring>
<TargetChangeable>false</TargetChangeable>
</NvPubDef>
</BoxDef>
</TreeItem>
```

The following table shows how the nodes marked **in bold** must be adapted according to the desired communication pattern. The "Activated" attribute must be set to "true" on the <Udp> node.

RT-Ethernet communication pattern	<Address>	<Gateway>
Broadcast	255.255.255.255	0.0.0.0
Multicast	Must contain a multicast IP address. See here for more information.	Contains a default gateway, if applicable, to which the packages for routing must be sent. Otherwise leave at 0.0.0.0
Unicast	Must contain a (unicast) IP address. See here for more information.	Contains a default gateway, if applicable, to which the packages for routing must be sent. Otherwise leave at 0.0.0.0

Creating Subscriber variables

After you have successfully added a Publisher Box, you can now add Publisher variables to this box by using the ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] method together with the required parameters for SubType (0) and vInfo, which define the data type of the Publisher variables, e.g. "BOOL".

Code snippet (C#):

```
ITcSmTreeItem subBox = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)^Box 1 (Subscriber)");
subVar = pubBox.CreateChild("MAIN.bTestVar", 0, null, "BOOL");
```

Code snippet (PowerShell):

```
$subBox = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)^Box 1
(Subscriber)")
$subVar = $pubBox.CreateChild("MAIN.bTestVar", 0, $null, "BOOL")
```

See also the scripting container example "EtherCAT Automation Protocol" for more information.

Linking Publisher/Subscriber variables

To link Publisher/Subscriber variables with PLC variables, simply use the ITcSysManager [▶ 112]::Linkvariables() [▶ 116] method.

Code snippet (C#):

```
systemManager.LinkVariales("TIPC^PLC Project^PLC Project Instance^PlcTask Outputs^MAIN.bTestVar",
"TIID^Device 1 (EtherCAT Automation Protocol)^Box 1 (Publisher)^MAIN.bTestVar^Outputs^VarData");
```

Code snippet (PowerShell):

```
$systemManager.LinkVariales("TIPC^PLC Project^PLC Project Instance^PlcTask Outputs^MAIN.bTestVar",
"TIID^Device 1 (EtherCAT Automation Protocol)^Box 1 (Publisher)^MAIN.bTestVar^Outputs^VarData")
```

See also the scripting container example "EtherCAT Automation Protocol" for more information.

Reading Publisher/Subscriber variable IDs

The following code snippet reads all variable IDs from a Publisher Box and saves them in the list array "ids". This can also be used for Subscriber variables.

Code snippet (C#):

```
List<uint> ids = new List<uint>();
ITcSmTreeItem pubBox = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)^Box 1 (Publisher)");
foreach(ITcSmTreeItem var in pubBox)
{
if (var.ItemType == 35) // 35 = publisher variable, 36 = subscriber variable
{
string varStr = var.ProduceXml();
XmlDocument varXml = new XmlDocument();
varXml.LoadXml(varStr);
XmlNode id = varXml.SelectSingleNode("//TreeItem/NvPubVarDef/NvId");
ids.Add(Convert.ToUInt32(id.InnerXml));
}
}
```

Code snippet (PowerShell):

```
$ids = New-Object System.Collections.ArrayList
$pubBox = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)^Box 1
(Publisher)")
foreach($var in $pubBox)
{
  if($var.ItemType -eq 35)
  {
    $varXml = [Xml]$var.ProduceXml()
    $id = $varXml.TreeItem.NvPubVarDef.NvId
    $ids.Add($id)
  }
}
```

4.3.6.3 Creating and handling Profinet devices

This article explains how to create and handle Profinet I/O devices via TwinCAT Automation Interface. The following topics are being discussed:

- Creating Profinet devices
- Adding Profinet boxes
- Adding Profinet modules
- Adding Profinet Sub-Modules

Creating Profinet devices

To create Profinet I/O devices (Controller/Device), the ITcSmTreeItem::CreateChild() method can be used. The Sub Type sets the actual type that should be added.

Name	Sub Type
Profinet Controller (RT)	113
Profinet Controller CCAT (RT)	140
Profinet Controller EL6631 (RT, EtherCAT)	119
Profinet Controller EL6632 (RT + IRT, EtherCAT)	126
Profinet Device (RT)	115
Profinet Device CCAT (RT)	142
Profinet Device CCAT (RT + IRT)	143
Profinet Device EL6631 (RT, EtherCAT)	118

Code Snippet (C#):

```
ITcSmTreeItem io = sysManager.LookupTreeItem("TIID");
ITcSmTreeItem profinetController = io.CreateChild("Profinet Controller", 113, null, null);
```

Code Snippet (Powershell):

```
$io = $sysManager.LookupTreeItem("TIID")
$profinetController = $io.CreateChild("Profinet Controller", 113, $null, $null)
```

Adding Profinet boxes

To create boxes below a Profinet device, the ITcSmTreeItem::CreateChild() may be used. The Sub Type depends on the box that should be added. The following table gives an overview about possible values:

Name	Sub Type
BK9102	9125
EK9300	9128
EL6631	9130

In addition some knowledge about the corresponding Profinet GSD file is required to use the vInfo parameter properly. The vInfo parameter is composed of the following syntax:
PathToGSDfile#ModuleIdentNumber#BoxFlags#DAPNumber

The ModuleIdentNumber can be determined from within the GSD file, e.g. via the XML structure <ProfileBody><ApplicationProcess><DeviceAccessPointList><DeviceAccessPointItem ModuleIdentNumber>. The ModuleIdentNumber is usually unique. If not, the DAPNumber specifies the position in the DeviceAccessPointList.

The following BoxFlags are currently interpreted:

Name	Value	Description
GENERATE_NAME_FROM_PAB	0x0004	Profinet name will be generated via process image
GET_STATIONNAME	0x0400	Profinet name from TC config will be used (tree item name)
SET_NOT_IP_TO_OS	0x4000	CE-only: Profinet IP will not be registered in OS

Code Snippet (C#):

```
ITcSmTreeItem profinetEL6631 = profinetController.CreateChild("EL6631", 9130, null, "C:\\TwinCAT\
\3.1\\Config\\Io\\Profinet\\GSDML-V2.31-beckhoff-EL6631-20140508.xml#0x3");
```

Code Snippet (Powershell):

```
$profinetEL6631 = $profinetController.CreateChild("EL6631", 9130, $null, "C:\\TwinCAT\\3.1\\Config\
\Io\\Profinet\\GSDML-V2.31-beckhoff-EL6631-20140508.xml#0x3")
```

Adding Profinet modules

Profinet modules are created below the API node of a Profinet box. The API node is automatically created when adding a Profinet box to the TwinCAT configuration. To add Profinet modules the ITcSmTreeItem::CreateChild() method may be used. The SubType determines the position of the module within the <ProfileBody><ApplicationProcess><ModuleList> XML structure of the corresponding GSD file.

Code Snippet (C#):

```
ITcSmTreeItem profinetEL6631api = profinetEL6631.get_Child(1);
ITcSmTreeItem profinetModule1 = profinetEL6631api.CreateChild("", 30, null, null); // SubType 30 =
200 Byte In-Out
ITcSmTreeItem profinetModule2 = profinetEL6631api.CreateChild("", 12, null, null); // SubType 12 = 8
Byte In-Out
ITcSmTreeItem profinetModule3 = profinetEL6631api.CreateChild("", 8, null, null); // SubType 8 = 4
Byte Out
```

Code Snippet (Powershell):

```
$profinetEL6631api = $profinetEL6631.get_Child(1)
$profinetModule1 = $profinetEL6631api.CreateChild("", 30, $null, $null)
$profinetModule2 = $profinetEL6631api.CreateChild("", 12, $null, $null)
$profinetModule3 = $profinetEL6631api.CreateChild("", 8, $null, $null)
```

Adding Profinet Sub-Modules

Profinet Sub-Modules can be added below a so-called Profinet Modular Module. The handling is very similar to when adding regular Profinet modules. The Sub Type determines the position of the Sub-Module within the <ProfileBody><ApplicationProcess><ModuleList> XML structure of the corresponding GSD file.

Code Snippet (C#):

```
ITcSmTreeItem profinetModularModule = profinetEL6631api.CreateChild("", 98, null, null); // SubType
98 = Modular
ITcSmTreeItem modularModule1 = profinetModularModule.CreateChild("", 12, null, null); // SubType 12
= 8 Byte In-Out
ITcSmTreeItem modularModule2 = profinetModularModule.CreateChild("", 14, null, null); // SubType 14
= 16 Byte Out
ITcSmTreeItem modularModule3 = profinetModularModule.CreateChild("", 31, null, null); // SubType 31
= 8 Word In
```

Code Snippet (Powershell):

```
$profinetModularModule = $profinetEL6631api.CreateChild("", 98, $null, $null)
$modularModule1 = $profinetModularModule.CreateChild("", 12, $null, $null)
$modularModule2 = $profinetModularModule.CreateChild("", 14, $null, $null)
$modularModule3 = $profinetModularModule.CreateChild("", 31, $null, $null)
```

4.3.6.4 Creating and handling Profibus devices

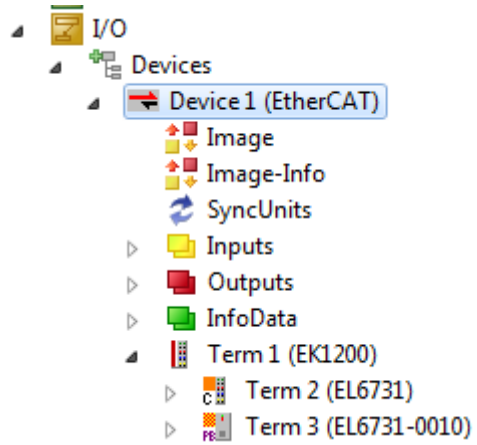
This article describes how Profibus master and slave devices are created and handled using the TwinCAT Automation Interface. It covers the following key topics:

- Creating and adding a Profibus master
- Searching for and configuring a suitable device (EL6731, FC310x, etc.)
- Creating and adding a Profibus slave
- Searching for and configuring a suitable slave device (EL6731-0010, etc.)
- Changing the fieldbus address

Creating and adding a Profibus master

1. To create a Profibus master device, open a new or existing TwinCAT configuration

2. Scan all devices. (These actions can also be carried out via the Automation Interface)



3. Create a system manager object and navigate to the devices

Code snippet (C#):

```
project = solution.Projects.Item(1);
sysman = (ItcSysManager)project.Object;
ITcSmTreeItem io = (ITcSmTreeItem)sysman.LookupTreeItem("TIID");
```

Code snippet (PowerShell):

```
$project = $sln.Projects.Items(1)
$sysman = $project.Object
$io = $sysman.LookupTreeItem("TIID")
```

Use the ITcSmTreeItem:CreateChild method to add a Profibus master.

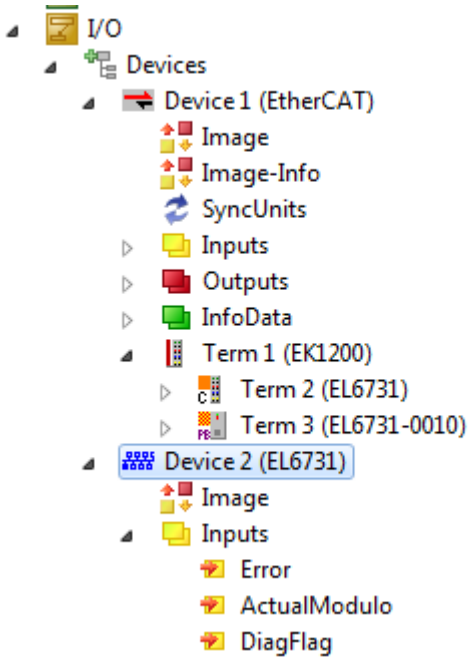
Code snippet (C#):

```
ITcSmTreeItem5 profi_master = (ITcSmTreeItem5)io.CreateChild("Device 2 (EL6731)", 86, "", null);
```

Code snippet (PowerShell):

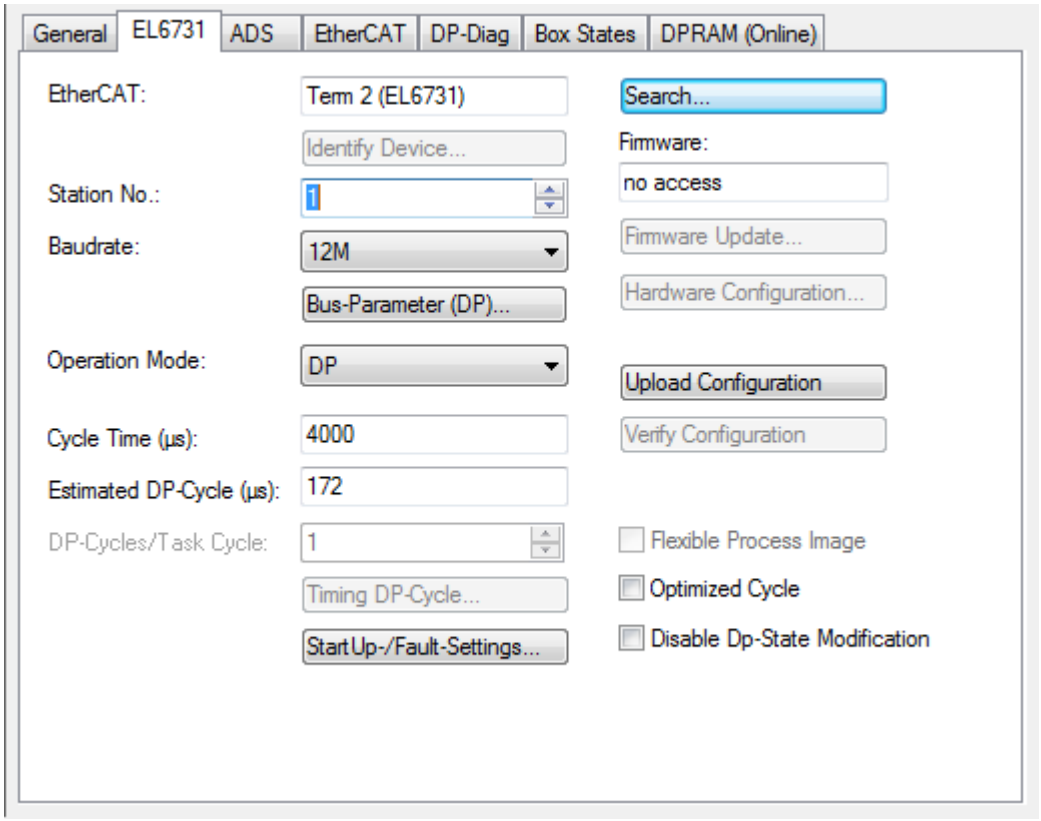
```
$profi_master = $io.CreateChild("Device 2 (EL6731) ", "86", "", $null)
```

For other Profibus masters, enter the correct ItemSubtype listed here. This will add the new device as shown in the screenshot:



Searching for and requesting a Profibus master device in the list:

The newly added Profibus master must be configured, which is normally done in TwinCAT by pressing the search button and choosing the correct device from the list.



This can be done via the Automation Interface:

Code snippet (C#):

```
string availableMaster = profi_master.ResourcesCount;
profi_master.ClaimResources(1);
```

Code snippet (PowerShell):

```
$availableMaster = $profi_master.ResourcesCount
$profi_master.ClaimResources(1)
```

ITcSmTreeItem5:ResourcesCount shows the number of compatible Profibus master devices, and ITcSmTreeItem5:ClaimResources takes the index of the CANopen device that is to be configured as the master.

Creating and adding a Profibus slave

A Profibus slave can be added to the current configuration as follows:

Code snippet (C#):

```
ITcSmTreeItem5 profi_slave = (ITcSmTreeItem5)io.CreateChild("Device 3 (EL6731-0010)", 97, null);
```

Code snippet (PowerShell):

```
$profi_slave = $io.CreateChild("Device 3 (EL6731-0010)", "97", "", $null)
```

Searching for and requesting a Profibus slave

As in the case of Profibus masters, the number of Profibus slaves can be published using the following code:

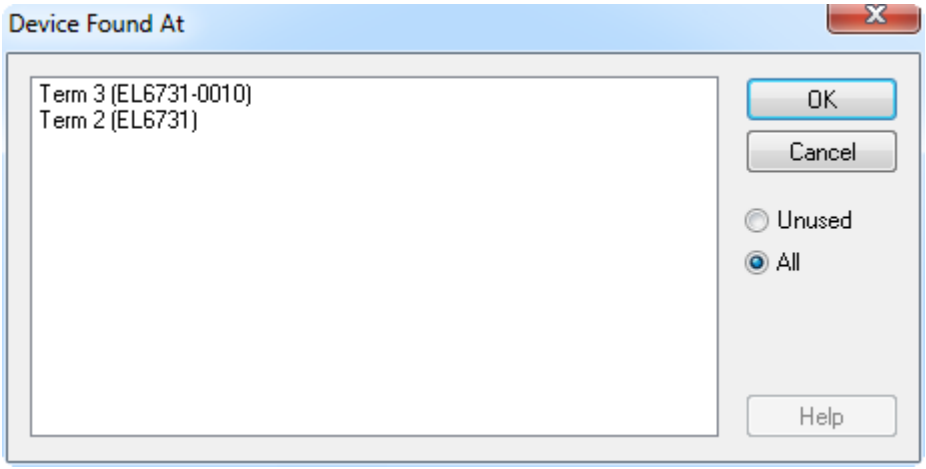
Code snippet (C#):

```
string availableSlaves = profi_slave.ResourcesCount;
profi_slave.ClaimResource(1);
```

Code snippet (PowerShell):

```
$availableSlaves = $profi_slave.ResourcesCount
$profi_slave.ClaimResources(1)
```

The last line in the code designates the EL6731-0010 device as a Profibus slave, similar to the dialog box that appears in the TwinCAT user interface.



Changing the fieldbus address

To change the fieldbus address (station no.) of a Profibus box in a configuration, a TwinCAT System Manager instance must be created and the configuration opened. The LookupTreeItem [▶ 118] method of the ITcSysManager [▶ 112] interface returns an ITcSmTreeItem [▶ 120] interface pointer implemented by the Tree Item referenced by its pathname [▶ 10]. This interface contains a ConsumeXml [▶ 153] method of the Tree Item.

Procedure

The procedure for creating the ITcSysManager [▶ 112] interface (the '*sysMan*' instance here) is described in the chapter Accessing TwinCAT configurations [▶ 19]. This interface has a LookupTreeItem [▶ 118] method that returns an ITcSmTreeItem [▶ 120] pointer to a Tree Item specified by its path name [▶ 10]. To change the fieldbus address (station no.) of a Profibus box "*TIID^Device 1 (FC310x)^Box 1 (BK3100)*" in 44, the following code can be used:

Code snippet (C#):

```
ItcSmTreeItem item = sysMan.LookupTreeItem("TIID^Device 1 (FC310x)^Box 1 (BK3100)");
item.ConsumeXml("44");
```

Code snippet (PowerShell):

```
$item = $sysMan.LookupTreeItem("TIID^Device 1 (FC310x)^Box 1 (BK3100)")
$item.ConsumeXml("44")
```

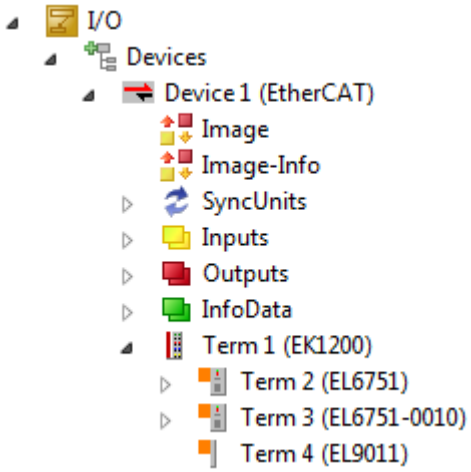
4.3.6.5 Creating and handling CANOpen devices

This article describes how CANopen master and slave devices are created and handled using the TwinCAT Automation Interface. It covers the following key topics:

- Creating and adding a CANopen master
- Searching for and configuring suitable devices (EL6751, FC510x etc.)
- Creating and adding a CANopen slave
- Searching for and configuring suitable devices (EL6751-0010 etc.)
- Importing DBC files via the Automation Interface

Creating and adding a CANopen master

To create a CANopen master device, open a new or existing TwinCAT configuration and scan all devices. Remember that these actions can also be carried out via the Automation Interface.



Create a system manager object and navigate to the devices.

Code snippet (C#):

```
project = solution.Projects.Item(1);
sysman = (ITcSysManager)project.Object;
ITcSmTreeItem io = (ITcSmTreeItem)sysman.LookupTreeItem("TIID");
```

Code snippet (PowerShell):

```
$project = $sln.Projects.Items(1)
$sysman = $project.Object
$io = $sysman.LookupTreeItem("TIID")
```

Use the ITcSmTreeItem:CreateChild method to add a CANopen master:

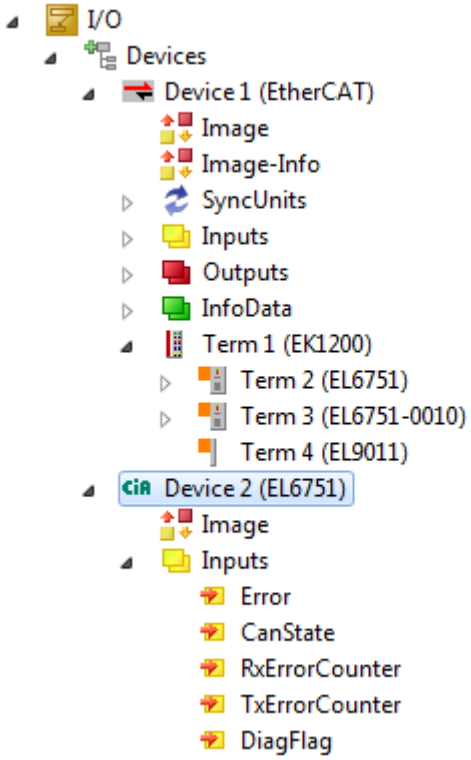
Code snippet (C#):

```
ITcSmTreeItem5 can_master = (ITcSmTreeItem5)io.CreateChild("Device 2 (EL6751)", 87, "", null);
```

Code snippet (PowerShell):

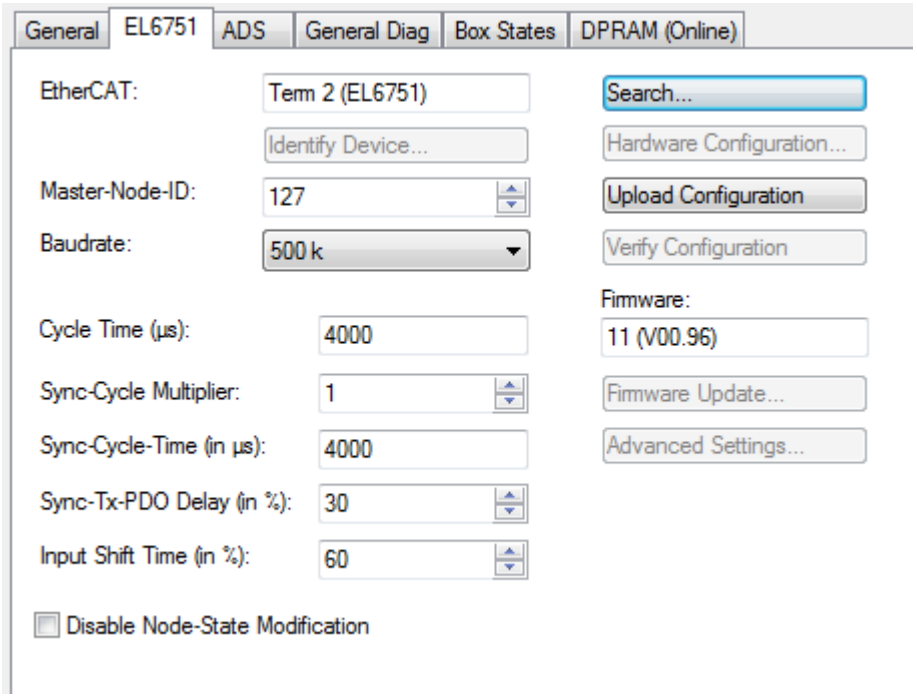
```
$can_master = $io.CreateChild("Device 2 (EL6751)", "87", "", $null)
```

Enter the correct ItemSubtype listed here [▶ 127] for other CANopen master devices. This will add the new device as shown in the screenshot:



Searching for and requesting a CANopen master device in the list:

The newly added CANopen master must be configured, which is normally done in TwinCAT by pressing the search button and choosing the correct device from the list.



This can be done via the Automation Interface:

Code snippet (C#):

```
string availableMaster = can_master.ResourceCount;
can_master.ClaimResources(1);
```

Code snippet (PowerShell):

```
$availableMaster = $can_master.ResourceCount
$can_master.ClaimResources(1)
```

ITcSmTreeItem5:ResourceCount shows the number of compatible CANopen master devices, and ITcSmTreeItem5:ClaimResources takes the index of the CANopen device that is to be configured as the master.

Creating and adding a CANopen slave

A CANopen slave can be added to the current configuration as follows:

Code snippet (C#):

```
ITcSmTreeItem5 can_slave = (ITcSmTreeItem5)io.CreateChild("Device 3 (EL6751-0010)", "98", null);
```

Code snippet (PowerShell):

```
$can_slave = $io.CreateChild("Device 3 (EL6751-0010)", "98", $null)
```

Searching for and requesting a CANopen slave

As in the case of CANopen masters, a list of CANopen slaves can be published using the following code:

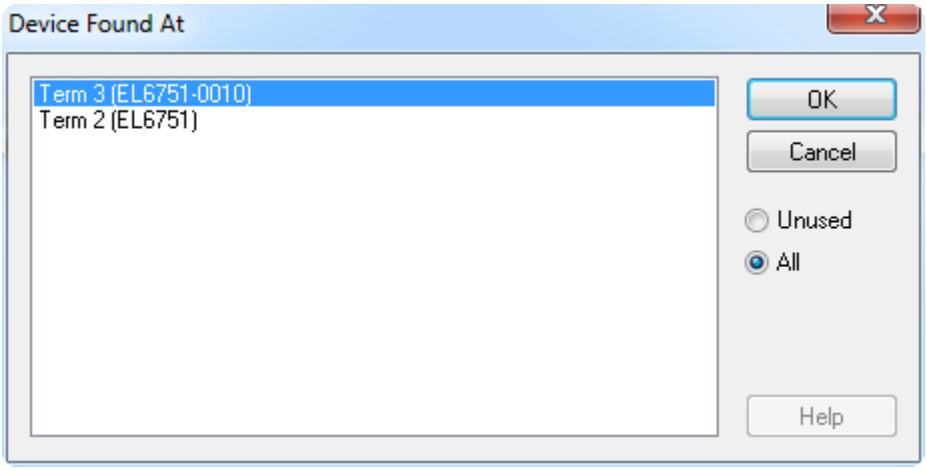
Code snippet (C#):

```
string availableSlaves = can_slave.ResourceCount;
can_slave.Claimresources(1);
```

Code snippet (PowerShell):

```
$availableSlaves = $can_slave.ResourceCount
$can_slave.Claimresources(1)
```

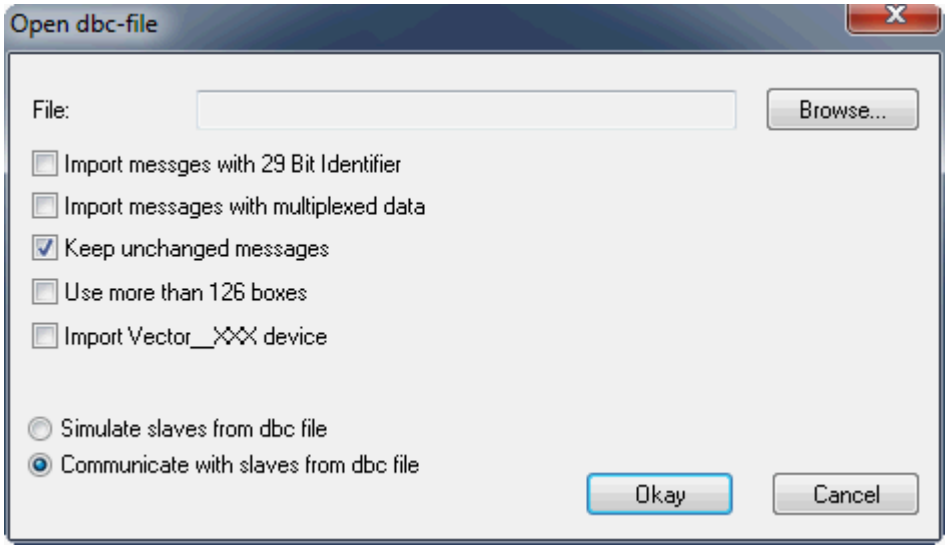
The last line in the code designates the EL6731-0010 device as a CANopen slave, similar to the dialog box that appears in the TwinCAT user interface.



Importing DBC files via the Automation Interface

Required version: TwinCAT 3.1 Build 4018 or higher

TwinCAT enables the import of the DBC file via a dialog box, as shown in the screenshot. By right-clicking on a CANopen master, e.g. EL6751, and choosing "Import dbc-file", a dialog box prompts the user to search the DBC file. Click OK to load the CANopen configuration automatically.



Automation Interface also supports the DBC file import. To do this, navigate to the CANopen master tree item and export the XML file via `ITcSmTreeItem:ProduceXml()`. In the XML file, add the path to the DBC file to be imported, along with the other properties that are part of the dialog box shown above.

```
<DeviceDef>
    <AmsPort>28673</AmsPort>
    <AmsNetId>172.17.251.109.2.1</AmsNetId>
    <AddressInfo>
        <Ecat>
            <EtherCATDeviceId>0</EtherCATDeviceId>
        </Ecat>
    </AddressInfo>
    <MaxBoxes>126</MaxBoxes>
    <ScanBoxes>false</ScanBoxes>
    <CanOpenMaster>
        <ImportDbcFile>
            <FileName>c:\dbc_file_folder\dbc_file_to_be_imported.dbc</FileName>
            <ImportExtendedMessages>false</ImportExtendedMessages>
            <ImportMultiplexedDataMessages>false</ImportMultiplexedDataMessages>
            <KeepUnchangedMessages>true</KeepUnchangedMessages>
            <ExtBoxesSupport>false</ExtBoxesSupport>
            <VectorXXXSupport>false</VectorXXXSupport>
            <CommunicateWithSlavesFromDbcFile>true</CommunicateWithSlavesFromDbcFile>
        </ImportDbcFile>
    </CanOpenMaster>
    <Fcxxxx>
        <CalculateEquiTimes>0</CalculateEquiTimes>
```

```
        <NodeId>127</NodeId>
        <Baudrate>500 k</Baudrate>
        <DisableNodeStateModification>false</DisableNodeStateModification>
    </Fcxxxx>
</DeviceDef>
```

Import the modified XML file back into the TwinCAT configuration via `ITcSmTreeImte:ConsumeXml()`. The configuration is now loaded under CANopen master.

4.3.6.6 Creating and handling Devicenet devices

Just like any other I/O component, Devicenet devices may be added via the TwinCAT Automation Interface by using the CreateChild() method of the ITcSmTreeItem interface. The SubType specifies the device that should be added.

Code Snippet (C#):

```
ITcSmTreeItem io = sysManager.LookupTreeItem("TIID");
ITcSmTreeItem devicenet1 = io.CreateChild("Device 1 (EL6752)", 88, null, null);
ITcSmTreeItem devicenet2 = io.CreateChild("Device 2 (EL6752-0010)", 99, null, null);
```

Code Snippet (Powershell):

```
$io = $sysManager.LookupTreeItem("TIID")
$devicenet1 = $io.CreateChild("Device 1 (EL6752)", 88, $null, $null)
$devicenet2 = $io.CreateChild("Device 2 (EL6752-0010)", 99, $null, $null)
```

The following table gives an overview about all Devicenet I/O devices and their corresponding SubTypes. If a device should be missing, you can always find out the SubType yourself by following our documentation article about the XML description of a tree item [▶ 24].

Device	SubType
Devicenet Master FC52xx PCI	41
Devicenet Master EL6752 EtherCAT	88
Devicenet Slave FC52xx PCI	62
Devicenet Slave EL6752 EtherCAT	99
Devicenet Master CX1500-M520 PC104	73
Devicenet Slave CX1500-B520-PC104	74
Devicenet Monitor FC52xx PCI	59

Devicenet boxes can also be attached via CreateChild(). The SubType specifies the box that should be added.

Code Snippet (C#):

```
ITcSmTreeItem devicenet1box = devicenet1.CreateChild("Box 2 (EL6752-0010)", 5203, null, null);
```

Code Snippet (Powershell):

```
$devicenet1box = $devicenet1.CreateChild("Box 2 (EL6752-0010)", 5203, $null, $null)
```

To add variables to a Devicenet box, simply use the following code snippet. The vInfo parameter specifies the data type of the variable.

Code Snippet (C#):

```
ITcSmTreeItem inputVars = devicenet1box.LookupChild("Inputs");
inputVars.CreateChild("TestVarInt", 0, null, "INT");
inputVars.CreateChild("TestVarBool", 0, null, "BOOL");
```

Code Snippet (Powershell):

```
$inputVars = $devicenet1box.LookupChild("Inputs")
$inputVars.CreateChild("TestVarInt", 0, $null, "INT")
$inputVars.CreateChild("TestVarBool", 0, $null, "BOOL")
```

4.3.6.7 Creating and handling AX5000 devices

This documentation article describes how to create AX5000 devices and and modify their mailbox startup list as well as their power management, PDO and motor condfiguration. All of these settings can be modifies by using the XML description [▶ 24] of the AX5000 TreeItem. This article consists of the following chapters:

- Creating an AX5000 I/O device
- Overview XML description
- Parametrization and template generation
- Available tools and samples
- Mappings

Creating an AX5000 I/O device

AX5000 devices can be created by using the CreateChild() method. Please consult the documentation article Creating and handling EtherCAT devices [▶ 67] for more information.

Code snippet (C#)

```
ITcSmTreeItem etherCatMaster = sysManager.LookupTreeItem("TIID^EtherCAT Master");
ITcSmTreeItem ax5000 = ethercatMaster.CreateChild("AX5000", 9099, "", "AX5203");
```

Code snippet (Powershell)

```
$etherCatMaster = $sysManager.LookupTreeItem("TIID^EtherCAT Master")
$ax5000 = $ethercatMaster.CreateChild("AX5000", 9099, "", "AX5203")
```

Overview XML description

The XML description of an AX5000 device consists of the following parts (references to an XML node written as XPath).

Part	Description
/TreeItem/EtherCAT/Slave/Mailbox/SoE/InitCmds/	Mailbox startup list for drive, including the power management and motor configuration.
/TreeItem/EtherCAT/Slave/ProcessData/TxPdo	PDO configuration
/TreeItem/EtherCAT/Slave/ProcessData/RxPdo	

Mailbox startup list

The mailbox startup list therefore includes many different types of init commands, which are identified via their IDN. Please note that the IDN that is displayed in TwinCAT XAE does not equal the IDN in the XML description, which is why a mapping needs to happen. For example: The IDN P-0-0050 (Motor construction type) equals <IDN>32818</IDN> (0x800A hex). To make this mapping and the identification which init command belongs to which type (drive, motor, power management), the tools that are mentioned below already include a sample mapping for an AX5000_0000_0210.

Import of startup list

Please note that the startup list always needs to be imported as a whole (including **all** init commands). This is also true if only one or two init commands have been changed.

PDO configuration

The PDO configuration is straight-forward and consists of a TxPdo (Inputs) and RxPdo (Outputs) section. Each PDO entry is identified by its Index (hex) and contains a name, bit length and data type. The optional property “Fixed” describes whether this entry can be deleted via TwinCAT XAE.

AdsInfo

When creating a template, the <AdsInfo> section does not need to be present and can be removed from the template file because this information is generated by the TwinCAT XAE.

Import of PDO configuration

Please note that the PDO configuration always needs to be imported as a whole. This is also true if only one or two PDO entries have been changed.

Parametrization and template generation

It is recommended to create templates for different AX5000 configurations and also a different template for each channel (A and B) of a drive. These templates can then be assembled to a full AX5000 XML description, which can be imported either via the XAE menu entry “TwinCAT -> SelectedItem -> Import XML description” or via the Automation Interface method ConsumeXml().

In order to make the template generation process easier, several tools and samples for the Windows Powershell and C# are available for download. All of these tools are further described below.

Available tools and samples

The following table provides an overview about the available tools and samples that provide help with handling AX5000 configurations via Automation Interface.

Name	Language	Description
CreateTcIoAx5000_Templates	Windows Powershell	Demonstrates how to generate template files out of a pre-configured AX5000 configuration.
CreateTcIoAx5000_Config	Windows Powershell	Demonstrates how to create an AX5000 configuration based on template files.

Please note that these scripts do not use the TwinCAT Automation Interface API but instead provide help with the preparation of XML configuration files.

CreateTcIoAx5000_Templates

This Windows Powershell script demonstrates how to generate template files out of a pre-configured AX5000 configuration. As a prerequisite, the AX5000 configuration needs to be exported to XML via the TwinCAT XAE menu entry “TwinCAT -> Selected Item -> Export XML description”. This has been done demonstratively for two configuration (SelectedItem_ExportXml_Test1.xml and SelectedItem_ExportXml_Test2.xml). The path to one of these files is then configured in the local variable $FullNameXmlExport within the Powershell script.

The script then loads the config file and subsequently extracts the XML configuration entries for the mailbox startup list, PDOs, power management and motor configuration and saves this information in separate template files (for each channel and for each type of configuration), which can later be re-added to a configuration by using the script “CreateTcIoAx5000_Config”.

CreateTcIoAx5000_Config

This Windows Powershell scripts demonstrates how to build a full AX5000 configuration out of different template files. These template files provide the settings for the PDO configuration as well as the mailbox startup list with their init commands for power management and motor configuration. The script output is a full AX5000 XML description, which can be imported on a AX5000 TreeItem either by using the menu entry “TwinCAT -> SelectedItem -> Import XML description” or by using the Automation Interface method ConsumeXml().

At the beginngin of the script, the local variable $FullNameConfigFile includes the path to the output file. Next, the script provides three different AX5000 configurations to choose from, each using a different template combination from the templates that were previously generated by using the script CreateTcIoAx5000_Templates (see above). Depending on the chosen config, the script then builds the AX5000 XML description by importing the different configuration parts into their corresponding XML section.

Mappings

Process data mapping can be performed via regular Automation Interface mechanisms. Please consult the article Creating and handling variable mappings [▶ 40] for more information.

4.3.6.8 Scanning for devices and boxes

When creating a new configuration it is often necessary to align the TwinCAT XAE configuration to the actually available hardware. One option to fullfill this is to start a new TwinCAT XAE configuration from scratch and process the following steps:

- Creation of a new TwinCAT XAE configuration
- Setting the address of the target system
- Scan of the available devices
- Addition and parametrization of the devices to be used
- Scanning and insertion of boxes for each device

Procedure

The procedure to create the ITcSysManager [▶ 112] interface (the '*systemManager*' instance here) is described in the chapter Accessing TwinCAT Configurations. [▶ 19] This interface has a LookupTreeItem [▶ 118] method that returns a ITcSmTreeItem [▶ 120] pointer to a specific tree item given by its pathname [▶ 10], in this case the shortcut "TIID" which references the I/O devices node.

Code snippet (C#):

```
ITcSysManager3 systemManager = null;

public void ScanDevicesAndBoxes()
{
  systemManager.SetTargetNetId("1.2.3.4.5.6");
  ITcSmTreeItem ioDevicesItem = systemManager.LookupTreeItem("TIID");
  string scannedXml = ioDevicesItem.ProduceXml(false);
  XmlDocument xmlDoc = newXmlDocument();
  xmlDoc.LoadXml(scannedXml);
  XmlNodeList xmlDeviceList = xmlDoc.SelectNodes("TreeItem/DeviceGrpDef/FoundDevices/Device");
  List<ITcSmTreeItem> devices = newList<ITcSmTreeItem>();
  int deviceCount = 0;
  foreach (XmlNode node in xmlDeviceList)
  {
    int itemSubType = int.Parse(node.SelectSingleNode("ItemSubType").InnerText);
    string typeName = node.SelectSingleNode("ItemSubTypeName").InnerText;
    XmlNode xmlAddress = node.SelectSingleNode("AddressInfo");
    ITcSmTreeItem device = ioDevicesItem.CreateChild(string.Format("Device_{0}",+
+deviceCount),itemSubType,string.Empty,null);
    string xml = string.Format("<TreeItem><DeviceDef>{0}</DeviceDef></
TreeItem>",xmlAddress.OuterXml);
    device.ConsumeXml(xml);
    devices.Add(device);
  }
  foreach (ITcSmTreeItem device in devices)
  {
    string xml = "<TreeItem><DeviceDef><ScanBoxes>1</ScanBoxes></DeviceDef></TreeItem>";
    try
    {
      device.ConsumeXml(xml);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Warning: {0}",ex.Message);
    }
```

```
    foreach (ITcSmTreeItem box in device)
    {
      Console.WriteLine(box.Name);
    }
  }
}
```

4.3.6.9 Enabling and disabling I/O devices

To disable/enable a tree item in a configuration, an instance of the TwinCAT System Manager has to be created and the configuration has to be opened. The LookupTreeItem [▶ 118] method of the ITcSysManager [▶ 112] interface returns a ITcSmTreeItem [▶ 120] interface pointer implemented by the tree item referenced by its pathname [▶ 10]. This interface contains a Disabled [▶ 120] property of the tree item.

Procedure

The Procedure to create the ITcSysManager [▶ 112] interface (the '*sysMan'* instance here) is described in the chapter Accessing TwinCAT Configurations. [▶ 19] This interface has a LookupTreeItem [▶ 118] method that returns a ITcSmTreeItem [▶ 120] pointer to a specific tree item given by its pathname [▶ 10]. To disable/ enable the tree item "*TIID^EtherCAT Master*" the following code snippets can be used.

Sample (CSharp):

```
ITcSmTreeItem item = sysMan.LookupTreeItem("TIID^EtherCAT Master");
item.Disabled = DISABLED_STATE.SMDS_DISABLED;
```

Please note, that, for this sample, you need to add both a reference to "Microsoft Developer Environment 10.0" and "Beckhoff TCatSysManager Library 1.1" to your project.

Sample (PowerShell):

```
$DISABLED_STATE = @{"SMDS_NOT_DISABLED" = "0"; "SMDS_DISABLED" = "1"}
$item = $systemManager.LookupTreeItem("TIID^EtherCAT Master")
$item.Disabled = $DISABLED_STATE.SMDS_DISABLED
```

Sample (VBScript):

```
dim dte,sln,proj,sysMan
set dte = CreateObject("VisualStudio.DTE.10.0")
set sln = dte.Solution
call sln.Open("C:\SolutionFolder\MySolution1.sln")
set proj = sln.Projects(1)
set sysMan = proj.Object
set item = sysMan.LookupTreeItem("TIID^EtherCAT Master")
item.Disabled = SMDS_DISABLED '(oder item.Disabled = SMDS_NOT_DISABLED um Tree Item zu aktivieren)
```

4.3.7 TcCOM

4.3.7.1 Creating and handling TcCOM modules

This chapter explains how to add existing TcCOM modules to a TwinCAT configuration and parameterize them. The following topics will be briefly covered in this chapter:

- Acquiring a reference to “TcCOM Objects” node
- Adding existing TcCOM modules
- Iterating through added TcCOM modules
- Setting CreateSymbol flag for parameters
- Setting CreateSymbol flag for Data Areas
- Setting Context (Tasks)
- Linking variables

Acquiring a reference to “TcCOM Objects” node

In a TwinCAT configuration, the “TcCOM Objects” node is located under “SYSTEM^TcCOM Objects”. Therefore you can acquire a reference to this node by using the method ITcSysManager::LookupTreeItem() in the following way:

Code Snippet (C#):

```
ITcSmTreeItem tcComObjects = systemManager.LookupTreeItem("TIRC^TcCOM Objects");
```

Code Snippet (Powershell):

```
$tcComObjects = $systemManager.LookupTreeItem("TIRC^TcCOM Objects")
```

The code above assumes that there is already a systemManager objects present in your AI code.

Adding existing TcCOM modules

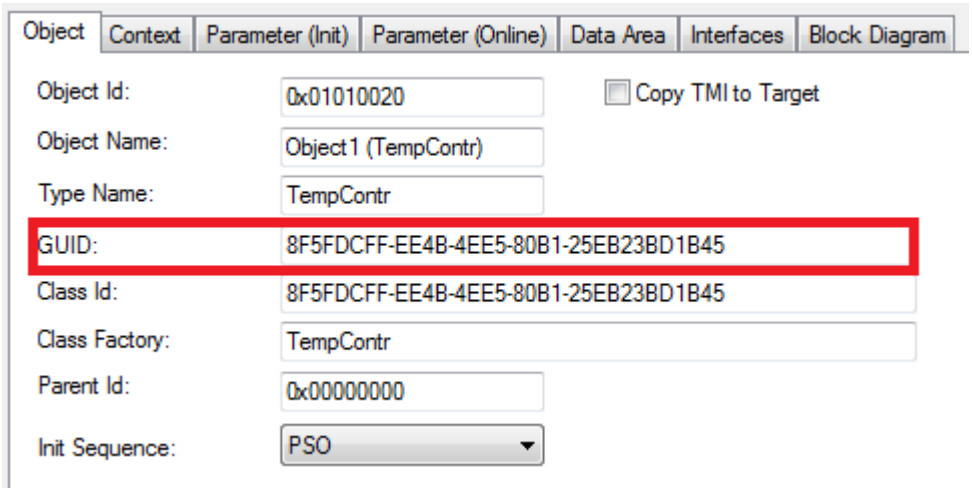
To add existing TcCOM modules to your TwinCAT configuration, these modules need to be detectable by TwinCAT. This can be achieved by either of the following ways:

- Copying TcCOM modules to folder %TWINCAT3.XDIR”\CustomConfig\Modules\
- Editing %TWINCAT3.XDIR”\Config\Io\TcModuleFolders.xml to add a path to a folder of your choice and place the modules within that folder

Both ways will be sufficient to make the TcCOM modules detectable by TwinCAT.

A TcCOM module is being identified by its GUID or name:

- This GUID can be used to add a TcCOM module to a TwinCAT configuration via the ITcSmTreeItem::CreateChild() method. The GUID can be determined in TwinCAT XAE via the properties page of a TcCOM module.



Alternatively, you can also determine the GUID via the TMC file of the TcCOM module.

```
<TcModuleClass>
  <Modules>
    <Module GUID="{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}">
    ...
    </Module>
  </Modules>
</TcModuleClass>
```

Let’s assume that we already own a TcCOM module that is registered in and detectable by TwinCAT. We now would like to add this TcCOM module, which has the GUID {8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45} to our TwinCAT configuration. This can be done by the following way:

Code Snippet (C#):

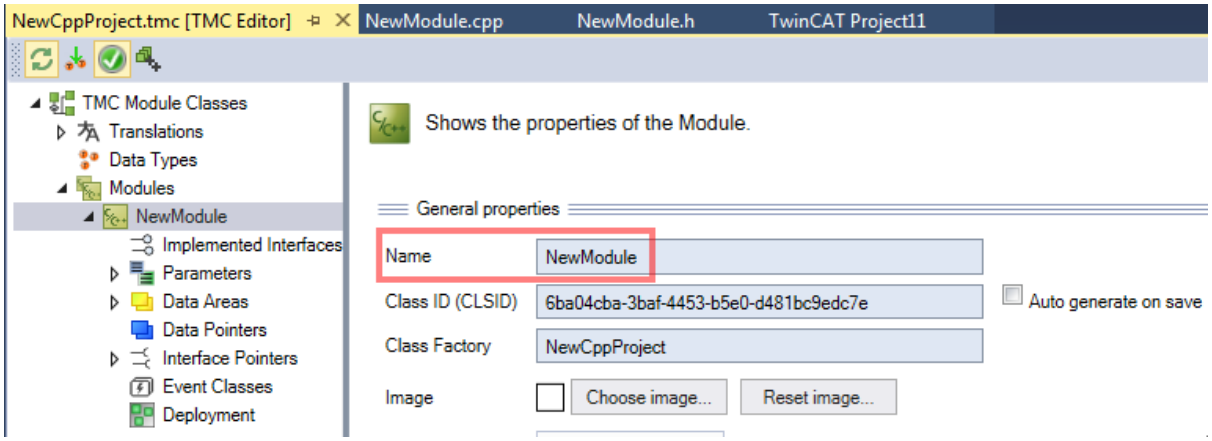
```
Dictionary<string,Guid> tcomModuleTable = new Dictionary<string,Guid>();
tcomModuleTable.Add("TempContr",Guid.Parse("{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}"));
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 0, "",
tcomModuleTable["TempContr"]);
```

Code Snippet (Powershell):

```
$tcomModuleTable = @{}
$tcomModuleTable.Add("TempContr", "{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}")
$tempController = $tcComObjects.CreateChild("Test", 0, "", $tcomModuleTable["TempContr"])
```

Please note that the vInfo parameter of the method ItcSmTreeItem::CreateChild() contains the GUID of the TcCOM module which is used to identify the module in the list of all registered TcCOM modules in that system.

- This name can be used to add a TcCOM module to a TwinCAT configuration via the ITcSmTreeItem::CreateChild() method. The name can be determined in TwinCAT XAE via the TMC Editor.



- This can be done by the following way:

Code Snippet (C#):

```
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 1, "", "NewModule");
```

Code Snippet (Powershell):

```
$tempController = $tcComObjects.CreateChild("Test", 0, "", "NewModule")
```

Iterating through added TcCOM modules

To iterate through all added TcCOM module instances, you may use the ITcModuleManager2 interface. The following code snippet demonstrates how to use this interface.

Code Snippet (C#):

```
ITcSysManager3 sysManager3 = sysManager as ITcSysManager3;

ITcModuleManager3 moduleManager = sysManager3.GetModuleManager() as ITcModuleManager3;
foreach (ITcModuleInstance2 moduleInstance in moduleManager)
{
  string moduleType = moduleInstance.ModuleTypeName;
  string instanceName = moduleInstance.ModuleInstanceName;
  Guid classId = moduleInstance.ClassID;
  uint objId = moduleInstance.oid;
  uint parentObjId = moduleInstance.ParentOID;
}
```

Code Snippet (Powershell):

```
$moduleManager = $systemManager.GetModuleManager()
ForEach( $moduleInstance in $moduleManager )
{
  $moduleType = $moduleInstance.ModuleTypeName
  $instanceName = $moduleInstance.ModuleInstanceName
  $classId = $moduleInstance.ClassID
  $objId = $moduleInstance.oid
  $parentObjId = $moduleInstance.ParentOID
}
```

Please note that every module object can also be interpreted as an ITcSmTreeItem, therefore the following type cast would be valid:

Code Snippet (C#):

```
ITcSmTreeItem treeItem = moduleInstance As ITcSmTreeItem;
```

Please note: Powershell uses dynamic data types by default.

Setting CreateSymbol flag for parameters

The CreateSymbol (CS) flag for parameters of a TcCOM module can be set via its XML description. The following code snippet demonstrates how to activate the CS flag for the Parameter “CallBy”.

Code Snippet (C#):

```
bool activateCS = true;
// First step: Read all Parameters of TcCOM module instance
string tempControllerXml = tempController.ProduceXml();
XmlDocument tempControllerDoc = new XmlDocument();
tempControllerDoc.LoadXml(tempControllerXml);
XmlNode sourceParameters = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
Parameters");

// Second step: Build target XML (for later ConsumeXml())
XmlDocument targetDoc = new XmlDocument();
XmlElement treeItemElement = targetDoc.CreateElement("TreeItem");
XmlElement moduleInstanceElement = targetDoc.CreateElement("TcModuleInstance");
XmlElement moduleElement = targetDoc.CreateElement("Module");
XmlElement parametersElement = (XmlElement) targetDoc.ImportNode(sourceParameters, true);
moduleElement.AppendChild(parametersElement);
moduleInstanceElement.AppendChild(moduleElement);
treeItemElement.AppendChild(moduleInstanceElement);
targetDoc.AppendChild(treeItemElement);

// Third step: Look for specific parameter (in this case "CallBy") and read its CreateSymbol
attribute
XmlNode destModule = targetDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module");
XmlNode callByParameter = destParameters.SelectSingleNode("Parameters/Parameter[Name='CallBy']");
XmlAttribute createSymbol = callByParameter.Attributes["CreateSymbol"];

createSymbol.Value = "true";

// Fifth step: Write prepared XML to configuration via ConsumeXml()
string targetXml = targetDoc.OuterXml;
tempController.ConsumeXml(targetXml);
```

Code Snippet (Powershell):

```
$tempControllerXml = [Xml]$tempController.ProduceXml()
$sourceParameters = $tempControllerXml.TreeItem.TcModuleInstance.Module.Parameters

[System.XML.XmlDocument] $targetDoc = New-Object System.XML.XmlDocument
[System.XML.XmlElement] $treeItemElement = $targetDoc.CreateElement("TreeItem")
[System.XML.XmlElement] $moduleInstanceElement = $targetDoc.CreateElement("TcModuleInstance")
[System.XML.XmlElement] $moduleElement = $targetDoc.CreateElement("Module")
[System.XML.XmlElement] $parametersElement = $targetDoc.ImportNode($sourceParameters, $true)
$moduleElement.AppendChild($parametersElement)
$moduleInstanceElement.AppendChild($moduleElement)
$treeItemElement.AppendChild($moduleInstanceElement)
$targetDoc.AppendChild($treeItemElement)

$destModule = $targetDoc.TreeItem.TcModuleInstance.Module
$callByParameter = $destmodule.SelectSingleNode("Parameters/Parameter[Name='CallBy']")

$callByParameter.CreateSymbol = "true"

$targetXml = $targetDoc.OuterXml
$tempController.ConsumeXml($targetXml)
```

Setting CreateSymbol flag for Data Areas

The CreateSymbol (CS) flag for Data Areas of a TcCOM module can be set via its XML description. The following code snippet demonstrates how to activate the CS flag for the Data Area “Input”. Please note that the procedure is pretty much the same as for parameters.

Code Snippet (C#):

```
bool activateCS = true;
// First step: Read all Data Areas of a TcCOM module instance
string tempControllerXml = tempController.ProduceXml();
XmlDocument tempControllerDoc = new XmlDocument();
tempControllerDoc.LoadXml(tempControllerXml);
XmlNode sourceDataAreas = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
```

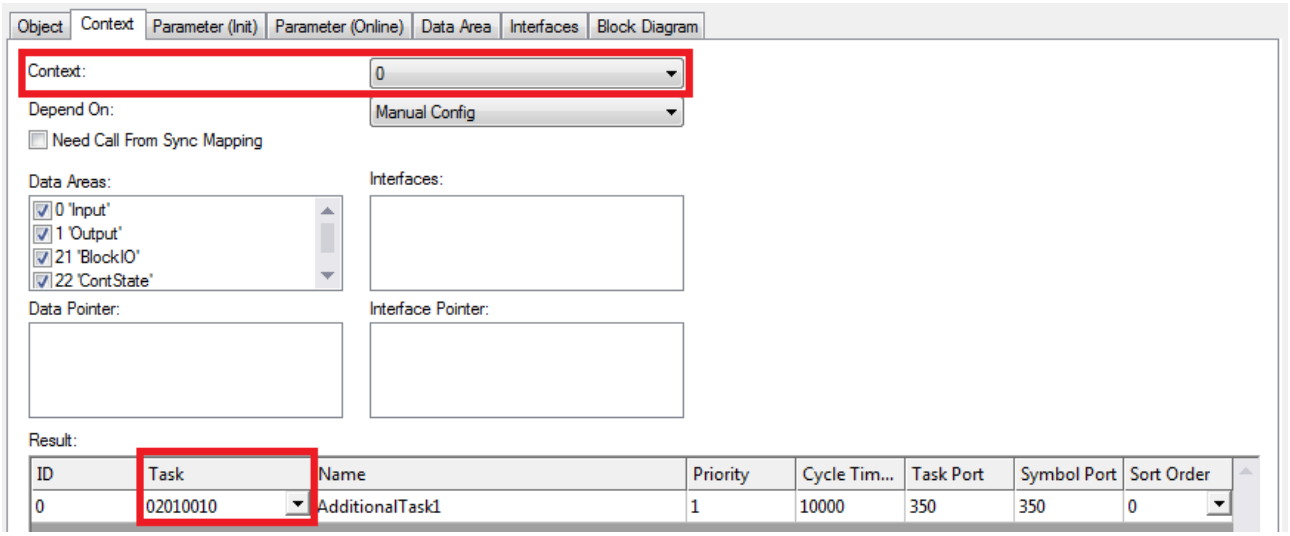
```
DataAreas");

// Second step: Build target XML (for later ConsumeXml())
XmlDocument targetDoc = new XmlDocument();
XmlElement treeItem = targetDoc.CreateElement("TreeItem");
XmlElement moduleInstance = targetDoc.CreateElement("TcModuleInstance");
XmlElement module = targetDoc.CreateElement("Module");
XmlElement dataAreas = (XmlElement)
targetDoc.ImportNode(sourceDataAreas, true);
module.AppendChild(dataAreas);
moduleInstance.AppendChild(module);
treeItem.AppendChild(moduleInstance);
targetDoc.AppendChild(treeItem);

// Third step: Look for specific Data Area (in this case "Input") and read its CreateSymbol
attribute
XmlElement dataArea = (XmlElement)targetDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
DataAreas/DataArea[ContextId='0' and Name='Input']");
XmlNode dataAreaNo = dataArea.SelectSingleNode("AreaNo");
XmlAttribute createSymbol = dataAreaNoNode.Attributes["CreateSymbols"];

// Fourth step: Set CreateSymbol attribute to true if it exists. If not, create attribute and set
its value
if (createSymbol != null)
string oldValue = createSymbol.Value;
else
{
createSymbol = targetDoc.CreateAttribute("CreateSymbols");
dataAreaNo.Attributes.Append(createSymbol);
}
createSymbol.Value = XmlConvert.ToString(activateCS);

// Fifth step: Write prepared XML to configuration via ConsumeXml()
string targetXml = targetDoc.OuterXml;
tempController.ConsumeXml(targetXml);
```

Code Snippet (Powershell):

```
$tempControllerXml = [Xml]$tempController.ProduceXml()
$sourceDataAreas = $tempControllerXml.TreeItem.TcModuleInstance.Module.DataAreas

[System.XML.XmlDocument] $targetDoc = New-Object System.XML.XmlDocument
[System.XML.XmlElement] $treeItem = $targetDoc.CreateElement("TreeItem")
[System.XML.XmlElement] $moduleInstance = $targetDoc.CreateElement("TcModuleInstance")
[System.XML.XmlElement] $module = $targetDoc.CreateElement("Module")
[System.XML.XmlElement] $dataAreas = $targetDoc.ImportNode($sourceDataAreas, $true)
$module.AppendChild($dataAreas)
$moduleInstance.AppendChild($module)
$treeItem.AppendChild($moduleInstance)
$targetDoc.AppendChild($treeItem)

$destModule = $targetDoc.TreeItem.TcModuleInstance.Module
[System.XML.XmlElement] $dataArea = $destModule.SelectSingleNode("DataAreas/DataArea[ContextId='0'
and Name='Input']")
$dataAreaNo = $dataArea.SelectSingleNode("AreaNo")
$dataAreaNo.CreateSymbols = "true"

// Fifth step: Write prepared XML to configuration via ConsumeXml()
$targetXml = $targetDoc.OuterXml
$tempController.ConsumeXml($targetXml)
```

Setting Context (Tasks)

Every TcCOM module instance needs to be run in a specific context (task). This can be done via the ITcModuleInstance2::SetModuleContext() method. This method awaits two parameters: ContextId and TaskObjectId. Both are equivalent to the corresponding parameters in TwinCAT XAE:



Please note that the TaskObjectId is shown in hex in TwinCAT XAE.

Code Snippet (C#):

```
ITcModuleInstance2 tempControllerMi = (ITcModuleInstance2) tempController;
tempControllerMi.SetModuleContext(0, 33619984);
```

You can determine the TaskObjectId via the XML description of the corresponding task, for example:

Code Snippet (C#):

```
ITcSmTreeItem someTask = systemManager.LookupTreeItem("TIRT^SomeTask");
string someTaskXml = someTask.ProduceXml();
XmlDocument someTaskDoc = new XmlDocument();
someTaskDoc.LoadXml(someTaskXml);
XmlNode taskObjectIdNode = someTaskDoc.SelectSingleNode("TreeItem/ObjectId");
string taskObjectIdStr = taskObjectId.InnerText;
uint taskObjectId = uint.Parse(taskObjectIdStr, NumberStyles.HexNumber);
```

Linking variables

Linking variables of a TcCOM module instance to PLC/IO or other TcCOM modules can be done by using regular Automation Interface mechanisms, e.g. ITcSysManager::LinkVariables().

4.3.8 C++

4.3.8.1 Creating and handling C++ projects and modules

This chapter explains in-depth how to create, access and handle TwinCAT C++ projects. The following list shows all chapters in this article:

- General information about C++ projects
- Creating new C++ projects
- Creating new module within a C++ project
- Opening existing C++ projects
- Creating module instances
- Calling TMC Code Generator
- Calling Publish Modules command
- Setting C++ Project Properties
- Building project

General information about C++ projects

C++ projects are specified by their so-called project template, which are used by the “TwinCAT C++ Project Wizard”. Inside a project multiple modules could be defined by module templates, which are used by the “TwinCAT Class Wizard”.

TwinCAT-defined templates are documented in the Section C++ / Wizards.

The customer could define own templates, which is documented at the corresponding sub-section if C++ Section / Wizards.

Create C++ projects

To create a new C++ project using the Automation Interface, you must navigate to the C++ node and then execute the CreateChild() method with the appropriate template file as a parameter.

Code snippet (C#):

```
ITcSmTreeItem cpp = systemManager.LookupTreeItem("TIXC");
ITcSmTreeItem cppProject = cpp.CreateChild("NewCppProject", 0, "", pathToTemplateFile);
```

Code snippet (PowerShell):

```
$cpp = $systemManager.LookupTreeItem("TIXC")
$newProject = $cpp.CreateChild("NewCppProject", 0, "", $pathToTemplateFile)
```

To instantiate a driver project, use "TcVersionedDriverWizard" as `pathToTemplateFile`.

Creating new module within a C++ project

Within a C++ project usually a TwinCAT Module Wizard is used to let the wizard create a module by a template.

Code snippet (C#):

```
ITcSmTreeItem cppModule = cppProject.CreateChild("NewModule", 1, "", pathToTemplateFile);
```

Code snippet (PowerShell):

```
$cppModule = $cppProject.CreateChild("NewModule", 0, "", $pathToTemplateFile);
```

As example for instantiating a Cyclic IO module project please use "TcModuleCyclicCallerWizard " as `pathToTemplateFile`.

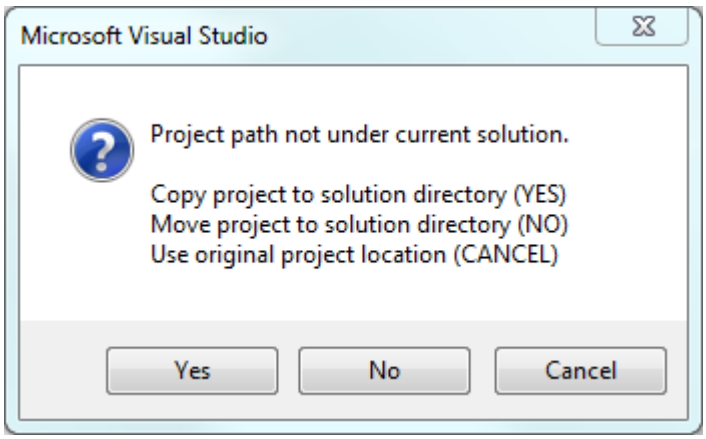
Open existing C++ projects

To open an existing C++ project using Automation Interface, you must navigate to the C++ node and then execute the CreateChild() method with the path of the corresponding C++ project file as a parameter.

You can use three different values as SubType:

- 0: Copy project to Solution directory
- 1: Move project to the Solution directory
- 2: Use the original project location (specify "" as NameOfProject parameter)

Basically, these values represent the functionalities (Yes, No, Cancel) of the following MessageBox in TwinCAT XAE:



Instead of the template file, you must use the path to the C++ project (or its vcxproj file) that must be added. Alternatively, you can also use a C++ project archive (tczip file).

Code snippet (C#):

```
ITcSmTreeItem cpp = systemManager.LookupTreeItem("TIXC");
ITcSmTreeItem newProject = cpp.CreateChild("NameOfProject", 1, "", pathToProjectOrTczipFile);
```

Code snippet (PowerShell):

```
$cpp = $systemManager.LookupTreeItem("TIXC")
$newProject = $cpp.CreateChild("NameOfProject", 1, "", $pathToProjectOrTczipFile)
```

Please note that C++ projects cannot be renamed, so the original project name must be specified. (cf. Renaming TwinCAT C++ projects)

Creating module instances

TcCOM Modules could be created at the System -> TcCOM Modules node. Please see documentation there [▶ 90].

The same procedure could also be applied to the C++ project node to add TcCOM instances at that place (`$newProject` at the code on top of this page.).

Calling the TMC Code Generator

The TMC Code Generator can be called to generate the C++ code according to the changes in the TMC file or the C++ project.

Code snippet (C#):

```
string startTmcCodeGenerator = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<StartTmcCodeGenerator>
<Active>true</Active>
</StartTmcCodeGenerator>
</Methods>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(startTmcCodeGenerator);
```

Code snippet (PowerShell):

```
$startTmcCodeGenerator = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<StartTmcCodeGenerator>
<Active>true</Active>
</StartTmcCodeGenerator>
</Methods>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($startTmcCodeGenerator)
```

Calling the Publish Modules command

The publication covers the development of the project for all platforms. The compiled module is made available for export as described in the section Module handling for C++.

Code snippet (C#):

```
string publishModules = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<PublishModules>
<Active>true</Active>
</PublishModules>
</Methods>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(publishModules);
```

Code snippet (PowerShell):

```
$publishModules = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<PublishModules>
<Active>true</Active>
</PublishModules>
</Methods>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($publishModules)
```

Setting C++ project properties

C++ projects offer various options for the build and deployment process.
These can be set via the Automation Interface.

Code snippet (C#):

```
string projProps = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<BootProjectEncryption>Target</BootProjectEncryption>
<TargetArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</TargetArchiveSettings>
<FileArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</FileArchiveSettings>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(projProps);
```

Code snippet (PowerShell):

```
$projProps = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<BootProjectEncryption>Target</BootProjectEncryption>
<TargetArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</TargetArchiveSettings>
<FileArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</FileArchiveSettings>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($projProps)
```

The values "None" and "Target" are valid for the BootProjectEncryption.
Both other settings are "false" and "true" values.

Set up project

To build the project or solution, you can use the corresponding classes and methods of the Visual Studio API [▶ 30].

4.3.9 Measurement

4.3.9.1 Creating and handling TwinCAT Measurement projects

The TwinCAT Automation Interface provides methods and properties to create and access TwinCAT Measurement projects. The following chapter describes how some basic tasks can be solved with such a TwinCAT project and includes information on the following topics:

- Requirements
- Creating a TwinCAT Measurement project
- Creating a TwinCAT Scope configuration
- Creating, accessing and handling diagrams
- Creating, accessing and handling axes
- Creating, accessing and handling channels
- Starting and stopping recordings
- Further information on TwinCAT Measurement can be found on the corresponding webpage in our information system.

Requirements

The integration of TwinCAT Measurement projects via the Automation Interface is possible from TwinCAT 3.1 Build 4013.

The interface definitions required for handling TwinCAT Measurement projects are located in the TwinCAT.Measurement.AutomationInterface.dll (this is located in the installation directory of the TE130x scope view)

IMeasurementScope:

int StartRecord();	Starts the recording. *The command is independent of the selected node, as the underlying scope project is always used.*
int StopRecord();	Stops the recording. *The command is independent of the selected node, as the underlying scope project is always used.*
int Disconnect();	Disconnects the connection to the ScopeServer. The recording is discarded.
int SaveSVD(string filePath);	Saves the recording to the specified file path.
int ExportCSV(string filePath);	Exports the recorded data in CSV format under the specified file path.
int ExportBinary(string filePath);	Exports the recorded data in binary format under the specified file path.
int ExportTDMS(string filePath);	Exports the recorded data to the TDMS format under the specified file path.
int ExportDAT(string filePath);	Exports the recorded data in DAT format under the specified file path.
int CreateChild(out object item, string name = "", int elementType = 0);	Creates a new element under the currently selected one. The element type specifies which type of element is to be created (if several options exist). The element types are listed in the table below.
int ChangeName(string name);	Changes the name of the element.
int ShowControl();	Opens the editor of the project
int CloseControl();	Closes the editor of the project
int LookUpChild(string name, out object childNode);	Searches for the element matching the name in the project tree.

The following element types are available as an enum in TwinCAT.Measurement.AutomationInterface.dll for the CreateChild method:

CursorAlignment	Vertical = 1, Horizontal = 2
MarkerType	TimeMarker = 100, XMarker = 101, YMarker = 102
ChartElement	AxisGroup = 0
ChartType	YT = 0, XY = 1, ARRAY = 2, SingleBar = 3
AcquisitionType	ADS = 0, OPC = 1
ConditionType	Area = 0, Threshold = 1
AxisGroupMember	Channel = 0, RoundShape = 1, AngularShape = 2
TriggerSetType	ChannelSet = 0, DirectorySet = 1
ChannelMember	AcquisitionInterpreter = 0, DynamicStyle = 1

Creating a TwinCAT Measurement project

TwinCAT Measurement is a global "container" that can host one or more measurement projects, e.g. a TwinCAT Scope configuration. Similar to a regular TwinCAT configuration, each project is first described with the aid of a template file. This template file is used when adding a new project to the solution, which can be done by calling the AddFromTemplate() method from the Visual Studio DTE object. Note that this procedure is the same when adding a regular TwinCAT project. The following code snippet assumes that you have already acquired a DTE instance and created a Visual Studio solution, as shown in our article about Accessing TwinCAT configuration [▶ 19]. A reference to the DTE instance is stored in the "dte" object.

Code snippet (C#):

```
EnvDTE.Project scopeProject = dte.Solution.AddFromTemplate(template, destination, name);
```

Definition of the parameters:

template	The standard template files are located in the installation directory of TE130X-Scope-View (e.g. C:\TwinCAT\Functions\TE130X-Scope-View\Templates\Projects\) and have the file type "tcmproj".
destination	Path in which the new configuration is to be saved on the hard disk.
name	Name of the project configuration.

Creating a TwinCAT Scope configuration

A TwinCAT Scope project stands for a recording configuration. This means that all elements inserted into the project are subject to the same recording settings. You can add a Scope project via the Automation Interface by specifying the appropriate "TwinCAT Scope Project" template when adding a project using the AddFromTemplate() method, as described above.

Creating, accessing and handling diagrams

Several diagrams can exist in parallel in a scope configuration. To add diagrams to an existing scope project, simply use the CreateChild() method from the IMeasurementScope interface. The following code snippet assumes that a TwinCAT Measurement project has already been created and that a reference to this project has been saved in the "scopeProject" object.

Code snippet (C#):

```
Project MeasurementProject = dte.Solution.Projects.Item(1);
ProjectItem ScopeProjectItem = MeasurementProject.ProjectItems.Item(1);
IMeasurementScope ScopeObj = (IMeasurementScope)ScopeProjectItem.Object;
ScopeObj.CreateChild(out object chartObj); // Erzeugen eines neuen YT-Charts im Scope

ProjectItem piChart = chartObj as ProjectItem;
IMeasurementScope imsChart = piChart.Object as IMeasurementScope;
imsChart.ChangeName("New Chart");
imsChart.CreateChild(out object axisObj); // Erzeugen einer neuen Achsengruppe unterhalb des Charts

ProjectItem piAxis = axisObj as ProjectItem;
IMeasurementScope imsAxis = piAxis.Object as IMeasurementScope;
imsAxis.ChangeName("New Axis");
imsAxis.CreateChild(out object channelObj); // Erzeugen eines neuen Kanals unterhalb der
Achsengruppe
ProjectItem piChannel = channelObj as ProjectItem;
IMeasurementScope imsChannel = piChannel.Object as IMeasurementScope;
imsChannel.ChangeName("New Channel");

IMeasurementScope DataPoolObj = (IMeasurementScope)ScopeProjectItem.ProjectItems.Item(1).Object;
DataPoolObj.CreateChild(out object acqObj); // Erzeugen einer neuen Acquisition im DataPool
ProjectItem piAcq = acqObj as ProjectItem;
IMeasurementScope imsAcq = piAcq.Object as IMeasurementScope;
imsAcq.ChangeName("New Acq");

foreach (Property prop in piChannel.Properties)
{
Debug.WriteLine(prop.Name + " - " + prop.Value); // Wir geben einmal alle Properties mit Nammen und
aktuellem Wert aus um uns einen Überblick zu verschaffen.
if (prop.Name == "Y-Data.Acquisition")
```

```
prop.Value = imsAcq; // Wir setzen die Acquisition des Channels auf die zuvor angelegte Acquisition
im DataPool
}
```

Starting and stopping recordings

The corresponding methods of the IMeasurementScope interface can be used to start/stop a configured scope recording:

Code snippet (C#):

```
ScopeObj.StartRecord()
ScopeObj.StopRecord();
```

4.3.9.2 Creating and handling TwinCAT Analytics projects

The TwinCAT Automation Interface provides methods and properties to create and access TwinCAT Measurement projects. The following chapter describes how some basic tasks can be solved with a TwinCAT project and includes information on the following topics:

- Requirements
- Creating a TwinCAT Measurement project
- Creating, accessing and handling a TwinCAT Analytics configuration
- Creating, accessing and handling networks
- Creating, accessing and handling functions
- Starting and stopping an analysis
- Further information on TwinCAT Measurement can be found on the corresponding webpage in our information system.

Creating a TwinCAT Measurement project

TwinCAT Measurement is a global "container" that can host one or more measurement projects, e.g. a TwinCAT Scope configuration. Similar to a regular TwinCAT configuration, each project is first described with the aid of a template file. This template file is used when adding a new project to the solution, which can be done by calling the AddFromTemplate() method from the Visual Studio DTE object. Note that this procedure is the same when adding a regular TwinCAT project. The following code snippet assumes that you have already acquired a DTE instance and created a Visual Studio solution, as shown in our article about accessing TwinCAT configurations. A reference to the DTE instance is stored in the "dte" object.

```
EnvDTE.Project scopeProject = dte.Solution.AddFromTemplate(template, destination, name);
```

Creating, accessing and handling a TwinCAT Analytics configuration

A TwinCAT Analytics project represents an analysis configuration. This means that all elements inserted into the project are subject to the same analysis settings. You can add an analytics project via the Automation Interface by specifying the appropriate "TwinCAT Analytics Project" template when adding a project using the AddFromTemplate() method, as described above.

The EnvDTE.Project element thus added can then be mapped to the IMeasurementAnalyticsProject interface, which provides the following methods:

int StartAnalytics()	Starts the analysis process.
int StopAnalytics();	Stops the analysis process.
int AddReferencedScope();	Adds an instance of a TwinCAT Scope project to the solution and links the analytics elements. The TwinCAT Scope instance can be adapted with the IMeasurementScope interface via Automation Interface.
int ChangeName(string name);	Method of changing the project name.
int ShowControl();	Brings the Microsoft Visual Studio® Editor to the foreground to display the Analytics project.
int CloseControl();	Closes the editor.
int GetAvailableModules(out Hashtable modules);	Fills a hash table with the data from the analytics engine modules provided. The hash table might look like this: {02040109-0000-0000-f000-000000000064} "Min Max Avg 1Ch" {02040103-0000-0000-f000-000000000064} "Edge Counter OnOff 2Ch" {02040102-0000-0000-f000-000000000064} "Edge Counter OnOff 1Ch" {02040101-0000-0000-f000-000000000064} "Edge Counter 1Ch" The GUIDs are necessary for later generation of the analysis functions.
int AddNetwork(out ProjectItem item, string name = "");	Adds a network to the Analytics project and issues the instance of the EnvDTE.ProjectItem with which you can continue. The object EnvDTE.ProjectItem.Object can be mapped to the IMeasurementAnalyticsNetwork interface.

Creating, accessing and handling networks

A TwinCAT Analytics network represents the level at which function blocks are created, managed and visualized.

A network can contain multiple instances of different functions, as well as other networks as sub-networks.

Each network is represented within a Microsoft Visual Studio® Editor instance, so that the internal docking mechanisms can be used to simultaneously represent different networks.

The IMeasurementAnalyticsNetwork interface defines the following methods:

int ShowControl();	Brings the Microsoft Visual Studio® Editor to the foreground to display the Analytics network.
int CloseControl();	Closes the editor.
int AddFunction(out ProjectItem item, Guid guid, string name = "");	Adds an analysis function to the network. The function is determined by the corresponding GUID, which can be selected via the interface method IMeasurementAnalyticsProject.GetAvailableModules(out Hashtable modules). The instance of EnvDTE.ProjectItem.Object can be mapped to the IMeasurementAnalyticsFunction interface.
int AddNetwork(out ProjectItem item, string name = "");	The object EnvDTE.ProjectItem.Object can be mapped to the IMeasurementAnalyticsNetwork interface.
int ChangeName(string name);	Changes the name of the network.
int AddNetworkTemplate(out ProjectItem item, string path);	Adds a network from a template. The template must be specified at file level. The object EnvDTE.ProjectItem.Object can be mapped to the IMeasurementAnalyticsNetwork interface.

Creating, accessing and handling functions

A TwinCAT Analytics Function represents the level of the analysis function.

Each function has its own input and output variables as well as various configuration parameters.

The IMeasurementAnalyticsFunction interface defines the following methods:

int ShowControl();	Brings the Microsoft Visual Studio® Editor to the foreground to display the Analytics project.
int CloseControl();	Closes the editor.
int ChangeName(string name);	Changes the name of the function.
int SetInputVariable(string input, int inputIndex);	Sets the input variable to the specified index of the function on the input string. The input string must be in XML format as an example from the TargetBrowser. TargetBrowserExportInfo To obtain the XML-formatted string, you can simply drag and drop the desired variable from the TargetBrowser into a text editor. Please note that the text editor must be started as an administrator if the TargetBrowser host (e.g. Microsoft Visual Studio®) has also been started as an administrator.

No interface is required to adjust the parameters of the function.

To do this, you can simply search the EnvDTE-Properties list of the EnvDTE.ProjectItem.Properties object for the EnvDTE.Property and set its value.

4.3.9.3 Creating and handling Analytics Logger and Stream Helper

The TwinCAT Automation Interface provides methods and properties to create and access TwinCAT Analytics Logger and Stream Helper. All functions are described below with C# code samples.

• Creating and deleting a Data Logger

• Creating and deleting a stream helper

• Parameterizing an already created Data Logger

• Parameterizing streams

• Selecting symbols to be logged.

All these operations are related to the respective nodes of the Analytics Configuration, Analytics Data Logger, Streams and Stream Helpers in a TwinCAT XAE project.

4.3.9.3.1 Creating and deleting a DataLogger

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics Configuration Node
ITcSmTreeItem analyticsConfig = sysManager.LookupTreeItem("TIAN");

analyticsConfig.CreateChild("MyDataLoggerName", 1, null, null);

analyticsConfig.DeleteChild("MyDataLoggerName");
```

4.3.9.3.2 Creating and deleting a StreamHelper

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics Configuration Node
ITcSmTreeItem analyticsConfig = sysManager.LookupTreeItem("TIAN");

analyticsConfig.CreateChild("MyStreamHelperName", 0, null, null);

analyticsConfig.DeleteChild("MyStreamHelperName_Obj1 (StreamHelper)");
```

When deleting, note the addition "_Obj1 (StreamHelper)" appended to the name passed in CreateChild.

4.3.9.3.3 Parameterizing an already created DataLogger

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics DataLogger Node
ITcSmTreeItem dataLogger = sysManager.LookupTreeItem("TIAN^MyDataLoggerName");

string sXmlDoc = dataLogger.ProduceXml();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(sXmlDoc);

XmlElement elemSetParam = xmlDoc.CreateElement("SetParameter");

XmlAttribute attrParamName = xmlDoc.CreateAttribute("name");
attrParamName.Value = "Data Format";

XmlAttribute attrParamValue = xmlDoc.CreateAttribute("value");
attrParamValue.Value = "ANALYTICS_FORMAT_FILE";

elemSetParam.Attributes.Append(attrParamName);
elemSetParam.Attributes.Append(attrParamValue);
xmlDoc.DocumentElement.AppendChild(elemSetParam);

string sConsumeXml = xmlDoc.OuterXml;
```

```
dataLogger.ConsumeXml(sConsumeXml);
```

The code snippet shows the use of the Produce-Consume-XML mechanism, where the XML text describing the project node can be read, modified and written again, which can be followed by actions on the part of the System Manager. The imported XML text contains, among other things, a listing of the parameters with name and current value. The name corresponds exactly to the name of the parameter under the **Parameter (Init)** tab of the DataLogger project node. This also applies to the respective values.

4.3.9.3.4 Parameterizing a stream

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics DataLogger Node
ITcSmTreeItem stream = sysManager.LookupTreeItem("TIAN^MyDataLoggerName^PlcStream1");

string sXmlDoc = stream.ProduceXml();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(sXmlDoc);

XmlElement elemSetParam = xmlDoc.CreateElement("SetParameter");

XmlAttribute attrParamName = xmlDoc.CreateAttribute("name");
attrParamName.Value = "Max ADS Buffer";

XmlAttribute attrParamValue = xmlDoc.CreateAttribute("value");
attrParamValue.Value = "23";

elemSetParam.Attributes.Append(attrParamName);
elemSetParam.Attributes.Append(attrParamValue);
xmlDoc.DocumentElement.AppendChild(elemSetParam);

string sConsumeXml = xmlDoc.OuterXml;
```

```
dataLogger.ConsumeXml(sConsumeXml);
```

The code snippet shows the use of the Produce-Consume-XML mechanism, where the XML text describing the project node can be read, modified and written again, which can be followed by actions on the part of the System Manager. The imported XML text contains, among other things, a listing of the parameters with name and current value. The name corresponds exactly to the name of the parameter under the **Data Handling** tab of the stream project node. This also applies to the respective values.

4.3.9.3.5 Selecting symbols

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics DataLogger Node
ITcSmTreeItem stream = sysManager.LookupTreeItem("TIAN^MyDataLoggerName^PlcStream1");

string sXmlDoc = stream.ProduceXml();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(sXmlDoc);

XmlElement elemSetSymbol = xmlDoc.CreateElement("SetSymbol");

XmlAttribute attrSymbolName = xmlDoc.CreateAttribute("name");
attrSymbolName.Value = "MAIN.stTestStructSimple.nMember1";

XmlAttribute attrEnable = xmlDoc.CreateAttribute("value");
attrEnable.Value = "true";

elemSetSymbol.Attributes.Append(attrSymbolName);
elemSetSymbol.Attributes.Append(attrEnable);

xmlDoc.DocumentElement.AppendChild(elemSetSymbol);
```

```
string sConsumeXml = xmlDoc.OuterXml;
stream.ConsumeXml(sConsumeXml);
```

4.3.10 Motion

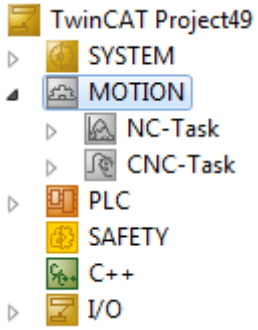
4.3.10.1 Creating and handling Motion projects

This chapter explains in-depth how to create and handle Motion projects. The following list shows all chapters in this article:

- General information about TwinCAT 3 Motion
- Creating a NC-Task
- Creating axes
- Parameterize axes

General information about TwinCAT Motion

TwinCAT 3 Motion consists of the following three components: NC I, NC PTP and CNC. It is therefore an assembly of function groups used for the control and regulation of axes or of synchronized axis groups. Further information about TwinCAT Motion can be found in TwinCAT 3 Motion documentation.



Creating a NC-Task

The first step to create a TwinCAT 3 Motion project is to create a so-called NC-Task. In TwinCAT Automation Interface, you can create this task simply by calling the method ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154] and using the SubType parameter 1, as the following code snippet shows.

Code Snippet (C#):

```
ITcSmTreeItem ncConfig = systemManager.LookupTreeItem("TINC");
ncConfig.CreateChild("NC-Task", 1);
```

Code Snippet (Powershell):

```
$ncConfig = $systemManager.LookupTreeItem("TINC")
$ncConfig.CreateChild("NC-Task", 1)
```

Creating axes

Since an NC task does not contain any axes by default, you must add them, again using the CreateChild() method, only this time on a reference to an axis node below the NC task.

Code snippet (C#):

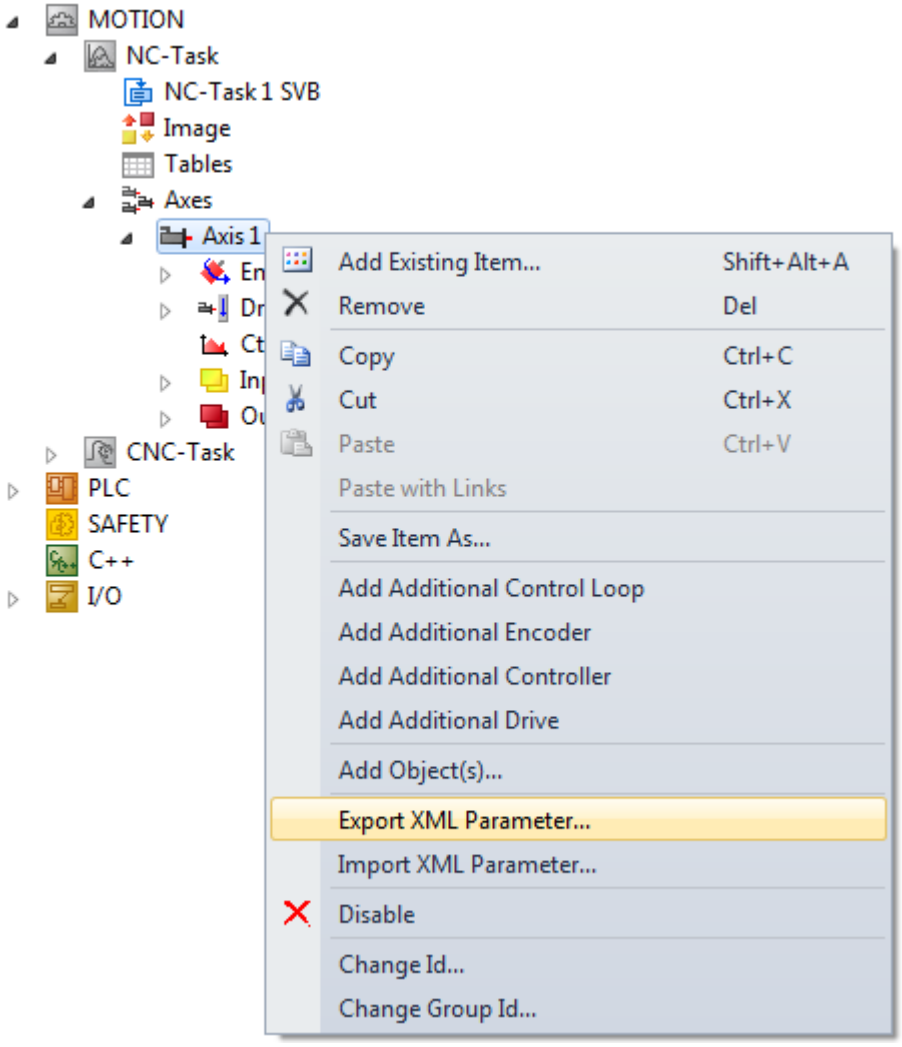
```
ITcSmTreeItem axes = systemManager.LookupTreeItem("TINC^NC-Task^Axes");
axes.CreateChild("Axis 1", 1);
```

Code snippet (PowerShell):

```
$axes = $systemManager.LookupTreeItem("TINC^NC-Task^Axes")
$axes.CreateChild("Axis 1", 1)
```

Parameterizing axes

Axes can be parameterized via their XML description [▶ 24]. To obtain an exemplary XML description, you can add an axis manually in TwinCAT XAE and use the corresponding entry in the context menu or add the axis via the Automation Interface and use the ITcSmTreeItem [▶ 120]::ProduceXml() [▶ 152] method to obtain it.



Code snippet (C#):

```
ITcSmTreeItem axis = systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1");
string xmlDescription = axis.ProduceXml();
```

Code snippet (PowerShell):

```
$axis = $systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1")
$xmlDescription = $axis.ProduceXml()
```

You can adapt this XML description according to your needs and then import it again using the ITcSmTreeItem [▶ 120]::ConsumeXml() [▶ 153] method to parameterize your axis.

Code snippet (C#):

```
ITcSmTreeItem axis = systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1");
axis.ConsumeXml(xmlDescription);
```

Code snippet (PowerShell):

```
$axis = $systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1")
$axis.ConsumeXml($xmlDescription)
```

4.3.11 Variant management

4.3.11.1 Access to TwinCAT variant management

This article describes access to the functions of the TwinCAT variant management via the Automation Interface. Access is possible from iTcSysManager14 (TCatSysManagerLib V 3.3.0.0). The following functions are supported:

1. Adding project variants and groups of variants
2. Setting the active variant
3. Activate settings for variant management

1. Adding project variants and groups of variants

Code snippet (C#):

```
string variantConfig = "<?xml version=\"1.0\"?><ProjectVariants><Group><Name>Group1</
Name><Member>Variant1</Member><Member>Variant2</Member></Group><Group><Name>Group2</
Name><Member>Variant2</Member><Member>Variant3</Member></Group><Variant><Name>Variant1</Name></
Variant><Variant><Name>Variant2</Name></Variant><Variant><Name>Variant3</Name></Variant></
ProjectVariants>";
sysManager.ProjectVariantConfig = variantConfig;
```

2. Setting the active variant

Code snippet (C#):

```
sysManager.CurrentProjectVariant = "Variant3";
sysManager.CurrentProjectVariant = "[Group1]";
```

3. Activate settings for variant management

Code snippet (C#):

```
ITcSmTreeItem9 el2004_1 = (ITcSmTreeItem9)sysManager.LookupTreeItem("TIID^EtherCAT
Master^EK1100-1^EL2004-1");
// activate the "disable" setting for the Variant Management
el2004_1.PvDisable = true;
// choose a variant and disable it only for this variant
sysManager.CurrentProjectVariant = "Variant3";
el2004_1.Disabled = DISABLED_STATE.SMDS_DISABLED;
```

4.3.12 Safety

4.3.12.1 Creating and handling safety projects

This chapter explains in detail how to create, access and handle safety projects. The following list contains all chapters of this article:

- General information about safety projects
- Opening existing safety projects

General information about safety projects

- The TwinCAT Automation Interface enables existing TwinCAT safety projects to be imported into the TwinCAT configuration. For this purpose, users can use either the corresponding *.splcproj file or the container format *.tfzip as the source template.

Opening existing safety projects

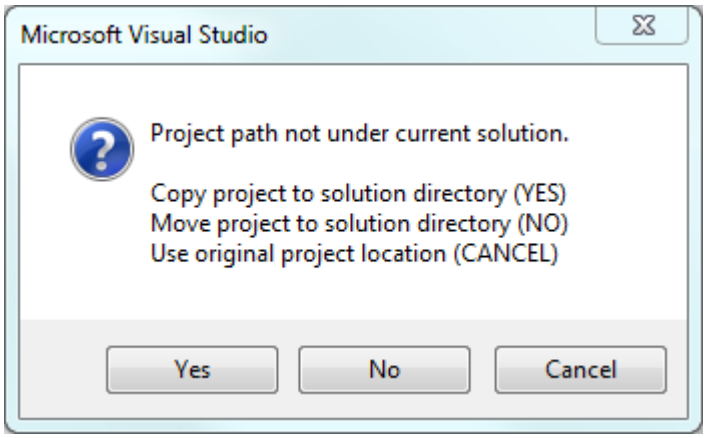
To open an existing safety project using Automation Interface, you must navigate to the safety node and then execute the CreateChild() method with the path of the corresponding safety project file as a parameter.

You can use three different values as SubType:

- 0: Copy project to solution directory
- 1: Move project to solution directory

- 2: Use original project location (if used, use "" as project name parameter)

Basically, these values represent the functionalities (Yes, No, Cancel) of the following MessageBox in TwinCAT XAE:



You can either use the path to the safety project (or its *.splcproj file) that needs to be added or you can use the safety project archive file (*.tfzip).

Code snippet (C#):

```
ITcSmTreeItem safety = systemManager.LookupTreeItem("TISC");
ITcSmTreeItem newProject = safety.CreateChild("NameOfProject", 0, null, pathToProjectOrTfzipFile);
```

Code snippet (PowerShell):

```
$safety = $systemManager.LookupTreeItem("TISC")
$newProject = $safety.CreateChild("NameOfProject", 0, "", pathToProjectOrTfzipFile)
```

5 API

5.1 Reference

This chapter contains a documentation of all classes and methods of the TwinCAT Automation Interface. The provided interfaces can be divided into different "levels" in which the higher level interfaces represent the primary interfaces and therefore the basic interaction with the Automation Interface. Please note that this differentiation comes only from a logical point-of-view, to get a better understanding about which interfaces are most important and which interfaces are of secondary importance.

Level 1 interfaces

As mentioned in our introduction [▶ 10], there are only two main interfaces which are being used for navigating and referencing tree items in TwinCAT configuration.

Main class	Description	Available since
ITcSysManager [▶ 112]	Base class to create and parameterize a TwinCAT configuration	TwinCAT 2.11
ITcSmTreeItem [▶ 120]	Represents a tree item within a TwinCAT configuration	TwinCAT 2.11

Level 2 interfaces

These interfaces are considered as "helper classes" which are always used together with level 1 classes, for example to cast an ITcSmTreeItem object into a more specific type of tree item, for example a POU (ITcPlcPou) or a linked task (ITcTaskReference).

Helper class	Description	Available since
ITcPlcLibraryManager [▶ 165]	Defines methods and properties for PLC library management	TwinCAT 3.1
ITcPlcPou [▶ 159]	Defines methods and properties to handle PLC POUs	TwinCAT 3.1
ITcPlcDeclaration [▶ 160]	Defines methods to read/write the declaration area of a PLC POU	TwinCAT 3.1
ITcPlcImplementation [▶ 161]	Defines methods to read/write the implementation area of a PLC POU	TwinCAT 3.1
ITcPlcProject [▶ 158]	Defines methods and properties regarding a PLC project, e.g. setting the project as a boot project	TwinCAT 3.1
ITcPlcIECProject [▶ 162]	Defines methods needed to import/ export PLC projects in PLCopen XML and also install them as a PLC library	TwinCAT 3.1
ITcPlcTaskReference [▶ 174]	Defines methods and properties to link the PLC project to a task	TwinCAT 3.1
ITcPlcLibrary [▶ 171]	Helper class which represents a single PLC library	TwinCAT 3.1
ITcPlcLibraries [▶ 172]	Helper class which represents a collection of PLC libraries	TwinCAT 3.1
ITCPlcReferences [▶ 171]	Helper class which represents a collection of ITcPlcLibRef objects (and therefore references in a PLC project)	TwinCAT 3.1
ITcPlcLibRef [▶ 172]	Helper class which represents a base class for ITcPlcLibrary and ITcPlcPlaceholderRef objects	TwinCAT 3.1
ITcPlcPlaceholderRef [▶ 173]	Helper class which represents a single PLC placeholder	TwinCAT 3.1
ITcPlcLibRepository [▶ 173]	Helper class which represents a single PLC library repository	TwinCAT 3.1
ITcPlcLibRepositories [▶ 173]	Helper class which represents a collection of PLC library repositories	TwinCAT 3.1

5.2 ITcSysManager

5.2.1 ITcSysManager

ITcSysManager is the main interface of the TwinCAT Automation Interface. This interface allows basic operations to configure TwinCAT 3 XAE and consists of several methods for doing so. Over the years, the ITcSysManager interface has been extended with more functionalities to give customers a better way to access all Automation Interface features. However, due to restrictions in the COM object model, these features needed to be added as separate interfaces to the Automation Interface. Therefore, each time a new set of features was added, these features were assembled in a new interface which was named ITcSysManagerX, where X is a number which is incremented each time a new interface was added. The following tables explain which methods are part of the ITcSysManager interface and which have been added to each new "feature-set" interface.

Methods

ITcSysManager methods	Description	Available since
NewConfiguration [▶ 114]	Generates a new configuration	TwinCAT 2.11
OpenConfiguration [▶ 114]	Loads a prior created configuration file (WSM file)	TwinCAT 2.11
SaveConfiguration [▶ 115]	Saves the configuration in a file with the given name or with the current name	TwinCAT 2.11
ActivateConfiguration [▶ 115]	Activates the configuration (same as "Save To Registry")	TwinCAT 2.11
LookupTreeItem [▶ 118]	Looks up to a configuration item (item in the tree) by name and returns a ITcSmTreeItem [▶ 120] interface	TwinCAT 2.11
StartRestartTwinCAT [▶ 115]	Starts or Restarts the TwinCAT System	TwinCAT 2.11
IsTwinCATStarted [▶ 115]	Evaluates if the TwinCAT System is running	TwinCAT 2.11
LinkVariables [▶ 116]	Links two variables given by names	TwinCAT 2.11
UnlinkVariables [▶ 116]	Clears the link between two variables given by names or all links from one variable.	TwinCAT 2.11

ITcSysManager2 methods	Description	Available since
SetTargetNetId [▶ 117]	Set the target NetId of the currently opened TwinCAT configuration.	TwinCAT 2.11
GetTargetNetId [▶ 117]	Gets the target NetId of the currently opened TwinCAT configuration.	TwinCAT 2.11
GetLastErrorMessages [▶ 117]	Get the last error messages which occurred in the TwinCAT subsystem.	TwinCAT 2.11

ITcSysManager3 methods	Description	Available since
LookupTreeItemById [▶ 119]	Looks for a configuration tree item with the specifed Item id.	TwinCAT 2.11
ProduceMappingInfo [▶ 119]	Produces a Xml-Description of the actual configuration mappings.	TwinCAT 3.1
ClearMappingInfo	Clears the mapping info.	TwinCAT 2.11

Comments

The ITcSysManager interface contains two methods used for navigating within TwinCAT XAE: ITcSysManager::LookupTreeItem [▶ 118] and ITcSysManager3::LookupTreeItemById [▶ 119]. A detailed explanation of browsing TwinCAT XAE can be found in the chapter TreeItem Browsing Models [▶ 22].

Warning: The three methods ITcSysManager::NewConfiguration [▶ 114], ITcSysManager::OpenConfiguration [▶ 114] and ITcsSysManager::SaveConfiguration [▶ 115] are only available in Compatiblity Mode [▶ 19]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

The **ITcSysmanager** and the ITcSmTreeItem [▶ 120] interface allows full access to a TwinCAT configuration. In the How to... section of this documentation there is a long (but incomplete) list of samples how to manipulate a TwinCAT configuration automatically.

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 2.11 and above

5.2.2 ITcSysManager::NewConfiguration

The NewConfiguration() method generates a new TwinCAT configuration file.

```
HRESULT NewConfiguration();
```

Return Values

S_OK	Function has returned a value.
E_ACCESSDENIED	The actual document is locked in the System Manager-Instance. This is if at least one reference to the system manager-object or one of the Tree Items is opened.
E_FAIL	Function failed.

Comments

Warning: The three methods ITcSysManager::NewConfiguration [▶ 114], ITcSysManager::OpenConfiguration [▶ 114] and ITcsSysManager::SaveConfiguration [▶ 115] are only available in Compatiblity Mode [▶ 19]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

5.2.3 ITcSysManager::OpenConfiguration

The OpenConfiguration() method loads a previously created TwinCAT configuration file.

```
HRESULT OpenConfiguration(BSTRbstrFile);
```

Parameters

bstrFile	[in, defaultvalue(L"")] contains the file path of the configuration file that should be loaded or an empty string if a new configuration should generated. The currently running configuration of a target device may also be read by using "CURRENTCONFIG".

Return Values

S_OK	Function has returned a value.
E_ACCESSDENIED	The actual document is locked in the system manager-instance. This is if at least one reference tot he system manager-object or one of the Tree Items is opened.
E_INVALIDARG	The path doesn’t point to a valid configfile.

Comments

Warning: The three methods ITcSysManager::NewConfiguration [▶ 114], ITcSysManager::OpenConfiguration [▶ 114] and ITcsSysManager::SaveConfiguration [▶ 115] are only available in Compatiblity Mode [▶ 19]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

5.2.4 ITcSysManager::SaveConfiguration

The SaveConfiguration() method saves a TwinCAT configuration in a file with the specified name.

```
HRESULT SaveConfiguration(BSTRbstrFile);
```

Parameters

bstrFile	[in, defaultvalue(L"")] contains the path, where the config file should be saved. When *bstrFile* is an empty character string, the actual file name is used.

Return Values

S_OK	Function has returned a value.
E_INVALIDARG	The file path is invalid.

Comments

Warning: The three methods ITcSysManager::NewConfiguration [▶ 114], ITcSysManager::OpenConfiguration [▶ 114] and ITcsSysManager::SaveConfiguration [▶ 115] are only available in Compatiblity Mode [▶ 19]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

5.2.5 ITcSysManager::ActivateConfiguration

The ActivateConfiguration() method activates the TwinCAT configuration (same as "Save To Registry"). A following start or restart of the TwinCAT system must be performed to activate the configuration physically.

```
HRESULT ActivateConfiguration();
```

Return Values

S_OK	Function has returned a value.
E_FAIL	The function failed.

5.2.6 ITcSysManager::IsTwinCATStarted

The IsTwinCATStarted() method evaluates if the TwinCAT System is running.

```
HRESULT IsTwinCATStarted(VARIANT_BOOL*pStarted);
```

Parameters

pStarted	[out, retval] points to the storage location of the boolean value, which contains the result.

Return Values

S_OK	Function returned a value.

5.2.7 ITcSysManager::StartRestartTwinCAT

The StartRestartTwinCAT() method starts or restarts the TwinCAT System. If TwinCAT is already started, the function performs a restart, if TwinCAT is stopped it performs a start.

```
HRESULT StartRestartTwinCAT();
```

Return Values

Requirements

S_OK	Function has returned a value.
E_FAIL	TwinCAT couldn't be started.

5.2.8 ITcSysManager::LinkVariables

The LinkVariables() method links two variables, which are specified by their names. The two variables represented by their tree path name will be linked. The path names must have the same syntax as described in ITcSysManager::LookupTreeItem [▶ 118]. The same shortcuts can be used.

```
HRESULT LinkVariables(BSTRbstrV1, BSTRbstrV2, longoffs1, longoffs2, longsize);
```

Parameters

bstrV1	[in] path name of the first variable. The full path name is required and each branch must be separated by a circumflex accent '^' or a tab.
bstrV2	[in] path name of the second variable. The full path name is required and each branch must be separated by a circumflex accent '^' or a tab.
offs1	[in, defaultvalue(0)] bit offset of the first variable (used if the two variables have different sizes or not the whole variable should be linked).
offs2	[in, defaultvalue(0)] bit offset of the second variable.
size	[in, defaultvalue(0)] bit count how many bits should linked. If *size* is 0 the minimum of the variable size of variable one and two is used.

Return Values

S_OK	function returns successfully.
TSM_E_ITEMNOTFOUND (0x98510001)	one or both of the path name(s) does not qualify an existing tree item.
TSM_E_INVALIDITEMTYPE (0x98510002)	one or both of the tree item(s) is not a variable.
TSM_E_MISMATCHINGITEMS (0x98510004)	the two variables cannot linked together. May be you have tried to link an output of one task with an output of another task or an output of a task with an input of a device or to variables of the same owner.
E_INVALIDARG	the values of *offs1*, *offs2* and/or *size* does not fit to the variables.

5.2.9 ITcSysManager::UnlinkVariables

The UnlinkVariables() method unlinks two variables, which are specified by their names, or clears all links from the first variable if the name *bstrV2* of the second variable is empty. The two variables represented by their tree path name will be unlinked. The path names must have the same syntax as described in ITcSysManager::LookupTreeItem [▶ 118]. The same shortcuts can be used.

```
HRESULT UnlinkVariables(BSTRbstrV1, BSTRbstrV2);
```

Parameters

bstrV1	[in] path name of the first variable. The full path name is required and each branch must be separated by a circumflex accent '^' or a tab.
bstrV2	[in, defaultvalue(L"")] path name of the second variable. If set the full path name is required and each branch must be separated by a circumflex accent '^' or a tab.

Return Values

S_OK	function returns successfully.
S_FALSE	the two variables have no link between them.
TSM_E_ITEMNOTFOUND (0x98510001)	one or both of the path name(s) does not qualify an existing tree item.
TSM_E_INVALIDITEMTYPE (0x98510002)	one or both of the tree item(s) is not a variable.
TSM_E_CORRUPTEDLINK (0x98510005)	the two variables cannot unlinked.

Comments

If *bstrV2* is an empty string the function clears all links of variable given by *bstrV1*. If *bstrV2* is not empty only an existing link between both variables will be deleted.

5.2.10 ITcSysManager2::GetTargetNetId

The GetTargetNetId() method returns the NetId of the current TwinCAT system.

```
HRESULT GetTargetNetId();
```

Parameters

None

Return Values

STRING	returns target's NetId

5.2.11 ITcSysManager2::SetTargetNetId

The SetTargetNetId() method sets the NetId of the current TwinCAT system.

```
HRESULT SetTargetNetId(STRING netId);
```

Parameters

netId	represents the target's NetId.

Return Values

S_OK	function returns successfully.

5.2.12 ITcSysManager2::GetLastErrorMessages

The GetLastErrorMessages() metod returns the last error messages.

```
HRESULT GetLastErrorMessages();
```

Parameters

None

Return Values

STRING	Returns last error messages.

5.2.13 ITcSysManager::LookupTreeItem

The LookupTreeItem() method returns a *ITcTreeItem* pointer of a tree item given by it's full path name.

```
HRESULT LookupTreeItem(BSTRbstrItem, ITcSmTreeItem**pipItem);
```

Parameters

bstrItem	[in] path name of the tree item looking for. The full path name is required and each branch must be separated by a circumflex accent '^' or a tab. A list of shortcuts for the main tree items is listed below.
pipItem	[out, retval] points to the location of a ITcSmTreeItem [▶ 120] interface pointer on return. The interface pointer gives access to specific methods belonging to the tree item.

Return Values

S_OK	function returns successfully.
TSM_E_ITEMNOTFOUND (0x98510001)	the path name does not qualify an existing tree item.

Shortcuts

The main tree items that exists in every configuration file can be accessed via shortcuts. These shortcuts are language neutral and require less memory:

```
"TIIC": shortcut for "I/O Configuration"
"TIID": shortcut for "I/O Configuration^I/O Devices" or "I/O Configuration" TAB "I/O Devices"
"TIRC": shortcut for "Real-Time Configuration"
"TIRR": shortcut for "Real-Time Configuration^Route Settings"
"TIRT": shortcut for " Real-Time Configuration^Additional Tasks" or " Real-Time Configuration" TAB
"Additional Tasks"
"TIRS": shortcut for " Real-Time Configuration^Real-Time Settings" or " Real-Time Configuration" TAB
"Real-Time Settings"
"TIPC": shortcut for "PLC Configuration"
"TINC": shortcut for "NC Configuration"
"TICC": shortcut for "CNC Configuration"
"TIAC": shortcut for "CAM Configuration"
```

Sample (C++):

```
ITcSmTreeItem* ipItem;

BSTR bstrItem = L"TIID^Device 1 (C1220)";

if ( SUCCEEDED(spTsm->LookupTreeItem( bstrItem, &ipItem ))
)

{

// do anything with ipItem


ipItem->Release();

}
```

Sample (VB): Dim ipItem As ITcSmTreeItem
set ipItem = spTsm.LookupTreeItem("TIID^Device 1 (C1220)")
' do anything with ipItem

Comments

5.2.14 ITcSysManager3::LookupTreeItemById

The LookupTreeItemById() method returns a *ITcTreeItem* pointer of a tree item given by it's full path name.

```
HRESULT LookupTreeItemById(longitemType, longitemId, ITcSmTreeItem**pipItem);
```

Parameters

itemType	[in] Item type of the TreeItem to find.
itemId	[in] ID of the TreeItem
pipItem	[out, retval] points to the location of a ITcSmTreeItem [▶ 120] interface pointer on return. The interface pointer gives access to specific methods belonging to the tree item.

Return Values

S_OK	function returns successfully.
TSM_E_ITEMNOTFOUND (0x98510001)	the *itemType itemId* combination doesn't qualify a valid tree item.

5.2.15 ITcSysManager3::ProduceMappingInfo

Generates an XML output that includes all currently configured mappings, e.g. between PLC and I/O.

```
HRESULT ProduceMappingInfo();
```

Parameters

none

Return Values

STRING: Returns XML structure that includes all configured mappings. The following snippet shows an example for this structure:

```
<VarLinks>
<OwnerA Name="TIID^Device 1 (EtherCAT)">
<OwnerB Name="TIXC^Untitled2^Untitled2_Obj1 (CModule1)">
<Link VarA="Term 1 (EK1100)^Term 3 (EL1008)^Channel 5^Input" VarB="Inputs^Value" />
<Link VarA="Term 1 (EK1100)^Term 2 (EL2008)^Channel 4^Output" VarB="Outputs^Value" />
</OwnerB>
</OwnerA>
<OwnerA Name="TIPC^Untitled1^Untitled1 Instance">
<OwnerB Name="TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)">
<Link VarA="PlcTask Outputs^MAIN.bOutput1" VarB="Channel 1^Output" />
<Link VarA="PlcTask Outputs^MAIN.bOutput3" VarB="Channel 3^Output" />
<Link VarA="PlcTask Outputs^MAIN.bOutput2" VarB="Channel 2^Output" />
</OwnerB>
<OwnerB Name="TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL1008)">
<Link VarA="PlcTask Inputs^MAIN.bInput1" VarB="Channel 1^Input" />
<Link VarA="PlcTask Inputs^MAIN.bInput3" VarB="Channel 3^Input" />
<Link VarA="PlcTask Inputs^MAIN.bInput2" VarB="Channel 2^Input" />
<Link VarA="PlcTask Inputs^MAIN.bInput4" VarB="Channel 4^Input" />
</OwnerB>
</OwnerA>
</VarLinks>
```

This example shows mappings between PLC <--> I/O and TcCOM (C++) <--> I/O.

5.2.16 ITcSysManager3::ConsumeMappingInfo

Consumes an XML structure that includes the mapping information for a project.

```
HRESULT ConsumeMappingInfo(BSTR bstrXml);
```

Parameters

bstrXml

[in]: String with XML structure. The XML mapping information can be acquired by using ITcSysManager3::ProduceMappingInfo() [▶ 119].

5.3 ITcSmTreeItem

5.3.1 ITcSmTreeItem

Each tree item in a TwinCAT XAE configuration is represented by an instance of the *ITcSmTreeItem* interface, which enables various interactions with the tree item.

A tree item of this interface will be, for example, returned by calling the ITCatSysManager::LookupTreeItem method, which is used to navigate through the tree.

Properties

ITcSmTreeItem Property	Type	Access	Description
Name	BSTR	RW	Name of tree item
Comment	BSTR	RW	Comment.
Disabled	BOOL	RW	Get/Set state of tree item which can be one of the following enum values: • SMDS_NOT_DISABLED (item is enabled) • SMDS_DISABLED (item is disabled) • SMDS_PARENT_DISABLED (**read only**, set if one of its parent is disabled)
PathName	BSTR	R	Path of tree item in TwinCAT XAE. The branches are separated by '^'. The PathName may be used in other method calls, e.g. ITCatSysManager::LookupTreeItem [▶ 118]. Please note that this property uniquely identifies a tree item in TwinCAT XAE.
ItemType	ENUM	R	Categorization of a tree item, e.g. Devices, Boxes, PLC, ... As defined by item types [▶ 121].
ItemSubType	LONG	RW	Sub type [▶ 124] of a tree item.
Parent	ITcSmTreeItem*	R	Pointer to the parent tree item.
ChildCount	LONG	R	Number of childs. Childs counted by this property enclose only main childs of the tree item (e.g. boxes are main childs of a device but not the device process image). To access all childs use the _NewEnum property.
Child(LONG n)	ITcSmTreeItem*	R	ITcSmTreeItem pointer of the n-th child
VarCount((LONG x)	LONG	R	Number of variables belonging to the tree item. x = 0 counted the input variables, x = 1 the outputs
Var(LONG x, LONG n)	ITcSmTreeItem*	R	ITcSmTreeItem pointer of the n-th variable. x = 0 uses the input variables, x = 1 the outputs
_NewEnum	IUnknown* (IEnumVariant*)	R	Returns a enum interface that enumerates **all** child tree items of the current tree item. This property may be used, for example, by a For-Each statement.

Methods

ITcSmTreeItem Methods	Description	Available since
CreateChild [▶ 154]	Creates a child tree item.	TwinCAT 2.11
DeleteChild [▶ 156]	Deletes a child tree item.	TwinCAT 2.11
ImportChild [▶ 157]	Imports a child item from the clipboard or a previously exported file.	TwinCAT 2.11
ExportChild [▶ 157]	Exports a child item to the clipboard or a file.	TwinCAT 2.11
ProduceXml [▶ 152]	Returns a String containing the XML representation of the item, with all its item-specific data and parameters.	TwinCAT 2.11
ConsumeXml [▶ 153]	Consumes a String containing the XML representation of the tree item, with all its item-specific data and parameters.	TwinCAT 2.11
GetLastXmlError [▶ 158]	Gets the error message of the last erroneous ConsumeXml() call.	TwinCAT 2.11
LookupChild [▶ 157]	Searches for a child with the specified relative path.	TwinCAT 2.11

ITcSmTreeItem2 Methods	Description	Available since
ResoucesCount	For interal use only	TwinCAT 2.11
ChangeChildSubType	Changes the SubType of the ITcSmTreeItem.	TwinCAT 2.11
ClaimResources	For internal use only	TwinCAT 2.11

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 2.11 and above

5.3.2 ITcSmTreeItem Item Types

Every tree item in TwinCAT System Manager / TwinCAT XAE is being **categorized** into various **groups** , e.g. devices, boxes, task, ... You can check the item type of a tree item by manually adding it to TwinCAT System Manager or XAE and then exporting its XML description via the corresponding menu entry.

- **TwinCAT System Manager:** Actions --> Export XML description
- **TwinCAT XAE:** TwinCAT --> Selected item --> Export XML description

```
<TreeItem>
    <ItemName>Device 1 (EtherCAT)</ItemName>
    <PathName>TIID^Device 1 (EtherCAT)</PathName>
    <ItemType>2</ItemType>
    <ItemId>1</ItemId>
    <ObjectId>#x03010010</ObjectId>
    <ItemSubType>111</ItemSubType>
```

In the resulting XML file, the item type is represented by the node <ItemType>.

General item types

Item type	Tag	Description
0	TREEITEMTYPE_UNKNOWN	---
1	TREEITEMTYPE_TASK	---
9	TREEITEMTYPE_IECPRJ	---
10	TREEITEMTYPE_CNCPRJ	---
11	TREEITEMTYPE_GSDMOD	Module of a Profibus GSD device
12	TREEITEMTYPE_CDL	---
13	TREEITEMTYPE_IECLZS	---
14	TREEITEMTYPE_LZSGRP	---
15	TREEITEMTYPE_IODEF	---
16	TREEITEMTYPE_ADDTASKS	---
17	TREEITEMTYPE_DEVICEGRP	---
18	TREEITEMTYPE_MAPGRP	---
30	TREEITEMTYPE_CANPDO	---
31	TREEITEMTYPE_RTIMESET	---
32	TREEITEMTYPE_BCPLC_VARS	---
33	TREEITEMTYPE_FILENAME	---
34	TREEITEMTYPE_DNETCONNECT	---
37	TREEITEMTYPE_FLBCMD	---
43	TREEITEMTYPE_EIPCONNECTION	---
44	TREEITEMTYPE_PNIOAPI	---
45	TREEITEMTYPE_PNIOMOD	---
46	TREEITEMTYPE_PNIOSUBMOD	---
47	TREEITEMTYPE_ETHERNETPROTOCOL	---
200	TREEITEMTYPE_CAMDEF	---
201	TREEITEMTYPE_CAMGROUP	---
202	TREEITEMTYPE_CAM	---
203	TREEITEMTYPE_CAMENCODER	---
204	TREEITEMTYPE_CAMTOOLGRP	---
205	TREEITEMTYPE_CAMTOOL	---
300	TREEITEMTYPE_LINEDEF	---
400	TREEITEMTYPE_ISGDEF	---
401	TREEITEMTYPE_ISGCHANNEL	---
402	TREEITEMTYPE_ISGAGROUP	---
403	TREEITEMTYPE_ISGAXIS	---
500	TREEITEMTYPE_RTSCONFIG	---
501	TREEITEMTYPE_RTSAPP	---
502	TREEITEMTYPE_RTSAPPTASK	---
503	TREEITEMTYPE_RTSADI	---
504	TREEITEMTYPE_CPPCONFIG	---
505	TREEITEMTYPE_SPLCCONFIG	---

I/O item types

Item type	Tag	Description
2	TREEITEMTYPE_DEVICE	I/O Device
3	TREEITEMTYPE_IMAGE	Process Image
4	TREEITEMTYPE_MAPPING	---
5	TREEITEMTYPE_BOX	I/O Box (e.g. "BK2000", child of I/O Devices)
6	TREEITEMTYPE_TERM	I/O Terminal (child of terminal couplers (box))
7	TREEITEMTYPE_VAR	Variable
8	TREEITEMTYPE_VARGRP	Variable Group (e.g. "Inputs")
35	TREEITEMTYPE_NVPUBLISHERVAR	---
36	TREEITEMTYPE_NVSUBSCRIBERVAR	---

PLC item types

Item type	Tag	Description
600	TREEITEMTYPE_PLCAPP	PLC application (root PLC object) [1]
601	TREEITEMTYPE_PLCFOLDER	PLC folder [1]
602	TREEITEMTYPE_PLCPOUPROG	POU Program [1]
603	TREEITEMTYPE_PLCPOUFUNC	POU Function [1]
604	TREEITEMTYPE_PLCPOUFB	POU Function Block [1]
605	TREEITEMTYPE_PLCDUTENUM	DUT enum data type [1]
606	TREEITEMTYPE_PLCDUTSTRUCT	DUT struct data type [1]
607	TREEITEMTYPE_PLCDUTUNION	DUT union data type [1]
608	TREEITEMTYPE_PLCACTION	PLC action [1]
609	TREEITEMTYPE_PLCMETHOD	PLC method [1]
610	TREEITEMTYPE_PLCITFMETH	PLC interface method [1]
611	TREEITEMTYPE_PLCPROP	PLC property [1]
612	TREEITEMTYPE_PLCITFPROP	PLC interface property [1]
613	TREEITEMTYPE_PLCPROPGET	PLC property getter [1]
614	TREEITEMTYPE_PLCPROPSET	PLC property setter [1]
615	TREEITEMTYPE_PLCGVL	GVL (Global variable list) [1]
616	TREEITEMTYPE_PLCTRANS	PLC Transition [1]
617	TREEITEMTYPE_PLCLIBMAN	PLC library manager [1]
618	TREEITEMTYPE_PLCITF	PLC interface [1]
619	TREEITEMTYPE_PLCVISOBJ	PLC visual object [1]
620	TREEITEMTYPE_PLCVISMAN	PLC visual manager [1]
621	TREEITEMTYPE_PLCTASK	PLC task object [1]
622	TREEITEMTYPE_PLCPROGREF	PLC program reference [1]
623	TREEITEMTYPE_PLCDUTALIAS	DUT Alias
624	TREEITEMTYPE_PLCEXTDATATYPECONT	PLC external data type container [1]
625	TREEITEMTYPE_PLCTMCDESCRIPTION	PLC TMC description file [1]
654	TREEITEMTYPE_PLCITFPROPGET	PLC interface property getter
655	TREEITEMTYPE_PLCITFPROPSET	PLC interface property setter

NC item types

Item type	Tag	Description
19	TREEITEMTYPE_NCDEF	---
20	TREEITEMTYPE_NCAXISES	---
21	TREEITEMTYPE_NCCHANNEL	NC Channel
22	TREEITEMTYPE_NCAXIS	NC Axis
23	TREEITEMTYPE_NCENCODER	---
24	TREEITEMTYPE_NCDRIVE	---
25	TREEITEMTYPE_NCCONTROLLER	---
26	TREEITEMTYPE_NCGROUP	---
27	TREEITEMTYPE_NCINTERPRETER	---
40	TREEITEMTYPE_NCTABLEGRP	---
41	TREEITEMTYPE_NCTABLE	---
42	TREEITEMTYPE_NCTABLESLAVE	---

Requirements

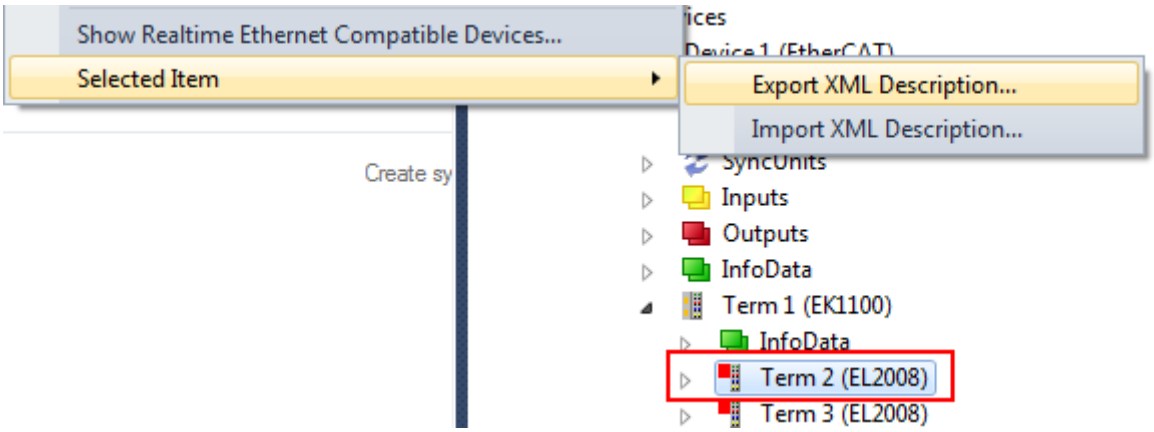
Notes	
[1]	requires TwinCAT 3.1

5.3.3 Tree item sub types

5.3.3.1 ITcSmTreeItem Item Sub Types

Item sub types specify what kind of **device** , **box** or **terminal** is being used, for example a sub type of 2408 identifies a KL2408 digital output terminal. You can check the sub type of an item by manually adding it in TwinCAT System Manager or XAE and then export its XML description via the corresponding menu entry.

- **TwinCAT System Manager:** Actions --> Export XML description
- **TwinCAT XAE:** TwinCAT --> Selected item --> Export XML description



In the resulting XML file, the sub type is represented by the node <ItemSubType>. In the above example, we exported the XML description of an EL2008 terminal. The XML file shows that this terminal has a sub type of 9099.

```
<TreeItem>
  <ItemName>Term 2 (EL2008)</ItemName>
  <PathName>TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)</PathName>
  <ItemType>5</ItemType>
  <ItemId>2</ItemId>
  <ObjectId>#x03020002</ObjectId>
  <ItemSubType>9099</ItemSubType>
  <ItemSubTypeName>EL2008 8Ch. Dig. Output 24V, 0.5A</ItemSubTypeName>
  <ChildCount>0</ChildCount>
  <Disabled>false</Disabled>
```

The following tables will give you a good overview about some of the available sub types. If your devices is not listed, please perform the above steps to determine the sub type of your specific device.

Shortcuts:

- Devices [▶ 127]
- Boxes [▶ 134]
- Terminals: E-Bus [▶ 125] (ELxxxx)
- Terminals: K-Bus digital input [▶ 140] (KL1xxx)
- Terminals: K-Bus digital output [▶ 142] (KL2xxx)
- Terminals: K-Bus analog input [▶ 145] (KL3xxx)
- Terminals: K-Bus analog output [▶ 147] (KL4xxx)
- Terminals: K-Bus position measurement [▶ 147] (KL5xxx)
- Terminals: K-Bus communication [▶ 148] (KL6xxx)
- Terminals: K-Bus power [▶ 149] (KL8xxx)
- Terminals: K-Bus safety [▶ 151] (KLx90x)
- Terminals: K-Bus system [▶ 150] (KL9xxx)

5.3.3.2 ITcSmTreeItem Item Sub Types: E-Bus

Due to their architecture, E-Bus **boxes** , **terminals** and **modules** will be handled differently than their K-Bus counterparts, e.g. during creation using the CreateChild() [▶ 154] method. As each K-Bus terminal will be specified according to its specific sub type, E-Bus terminals are recognized via one common sub type and then specified via their "**Product Revision** " which will be used as the vInfo parameter in the CreateChild() method.

Sub type	Description
9099	Generic sub type for all EtherCAT terminals. In case of CreateChild() [▶ 154], a specific terminal will be defined via vInfo parameter.

There are a few exceptions to this rule, e.g. for RS232 terminals. The following table gives an overview about these exceptions:

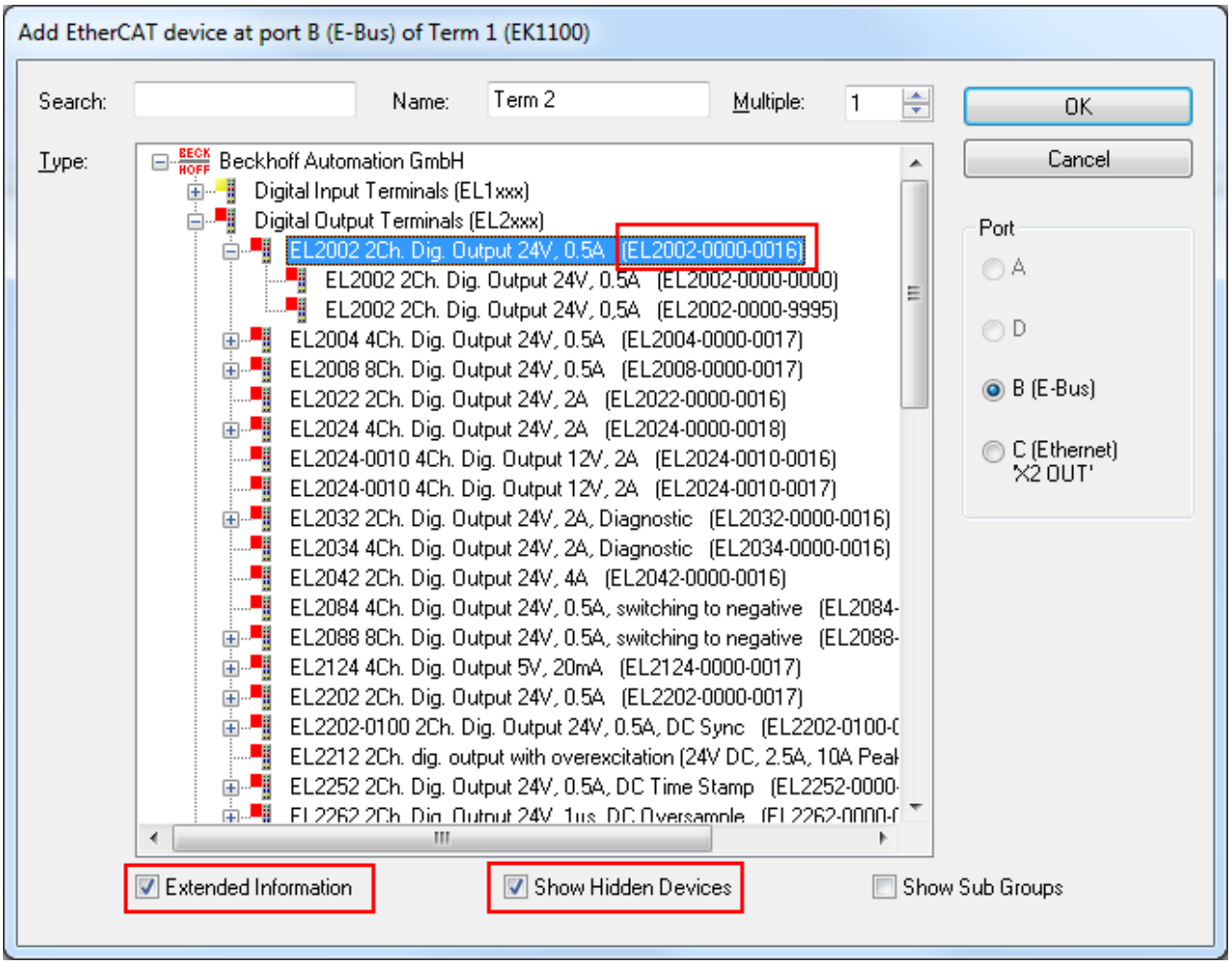
I/O	ItemSubType	Description
EP6002	9101	RS232 / RS422 / RS485 interface terminal
EL6001	9101	RS232 interface terminal
EL6002	9101	RS232 interface terminal (2-Channel)
EL6021	9103	RS422 / RS485 interface terminal
EL6022	9103	RS422 / RS485 interface terminal (2-Channel)

Code Snippet (C#)

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^EtherCAT Master^EK1100");
ek1100.CreateChild("EL2002 - 1", 9099, "", "EL2002-0000-0016");
```

Product revision

Each E-Bus box/terminal/module has its own product revision, which you can view either by exporting its XML description [▶ 24] or in the "Add Device" dialog in TwinCAT XAE.



For example, the EL2002 digital output terminal has the product revision EL2002-0000-0016, as you can also see in its XML description:

```
<EtherCAT>
- <Slave>
  - <Info>
    - <Name>
        <![CDATA[ Term 3 (EL2002)  ]]>
      </Name>
      <PhysAddr>1002</PhysAddr>
      <AutoIncAddr>65535</AutoIncAddr>
      <Physics>KK</Physics>
      <VendorId>2</VendorId>
      <ProductCode>131215442</ProductCode>
      <RevisionNo>1048576</RevisionNo>
      <SerialNo>0</SerialNo>
      <ProductRevision>EL2002-0000-0016</ProductRevision>
      <Type>EL2002</Type>
    </Info>
```

To get the XML description of a tree item, simply do the following:

- **TwinCAT 2:** Add the item to System Manager, select it and, from the menu, choose "Actions" --> "Export XML description"
- **TwinCAT 3:** Add the item to XAE, select it and, from the menu, choose "TwinCAT" --> "Selected item" --> "Export XML description"

5.3.3.3 Devices

5.3.3.3.1 ITcSmTreeItem Item Sub Types: Devices

Devices: Miscellaneous

Sub type	Tag	Description
0	IODEVICETYPE_UNKNOWN	---
6	IODEVICETYPE_BKHFPC	Beckhoff Industrial-PC C2001
9	IODEVICETYPE_LPTPORT	LPT Port
10	IODEVICETYPE_DPRAM	Generic DPRAM
11	IODEVICETYPE_COMPORT	COM Port
18	IODEVICETYPE_FCXXXX	Beckhoff-FeldbusCard
32	IODEVICETYPE_SMB	Motherboard System Management Bus
43	IODEVICETYPE_BKHFNCBP	Beckhoff NC Rückwand
44	IODEVICETYPE_SERCANSPCI	Sercos Master (SICAN/IAM PCI)
46	IODEVICETYPE_SERCONPCI	Sercon 410B or 816 Chip Master or Slave (PCI)
53	IODEVICETYPE_BKHFAH2000	Beckhoff AH2000 (Hydraulik Backplane)
55	IODEVICETYPE_AH2000MC	Beckhoff-AH2000 mit Profibus-MC

Devices: Beckhoff CX Terminal Devices

Sub type	Tag	Description
120	IODEVICETYPE_CX5000	CX5000 Terminal Device
135	IODEVICETYPE_CX8000	CX8000 Terminal Device
105	IODEVICETYPE_CX9000_BK	CX9000 Terminal Device
65	IODEVICETYPE_CX1100_BK	CX1100 Terminal Device
124	IODEVICETYPE_CCAT	Beckhoff CCAT Adapter

Devices: Beckhoff CP Devices

Sub type	Tag	Description
14	IODEVICETYPE_BKHFCP2030	Beckhoff CP2030 (Pannel-Link)
31	IODEVICETYPE_BKHFCP9030	Beckhoff CP9030 (Pannel-Link with UPS, ISA)
52	IODEVICETYPE_BKHFCP9040	Beckhoff CP9040 (CP-PC)
54	IODEVICETYPE_BKHFCP9035	Beckhoff CP9035 (Pannel-Link with UPS, PCI)
116	IODEVICETYPE_BKHFCP6608	Beckhoff CP6608(IXP PC)

Devices: Beckhoff BC/BX Devices

Sub type	Tag	Description
77	IODEVICETYPE_BX_BK	BX Klemmenbus Interface
78	IODEVICETYPE_BX_M510	BX SSB-Master
103	IODEVICETYPE_BC8150	BCXX50 Serial Slave
104	IODEVICETYPE_BX9000	BX9000 Ethernet Slave
107	IODEVICETYPE_BC9050	BC9050 Etherent Slave
108	IODEVICETYPE_BC9120	BC9120 Etherent Slave
110	IODEVICETYPE_BC9020	BC9020 Etherent Slave

Devices: Beckhoff EtherCAT

Sub type	Tag	Description
94	IODEVICETYPE_ETHERCAT	Obsolete: EtherCAT Master. Use "111" instead.
111	IODEVICETYPE_ETHERCATPROT	EtherCAT Master
130	IODEVICETYPE_ETHERCATSLV	EtherCAT Slave
106	IODEVICETYPE_EL6601	EtherCAT Automation Protocol via EL6601
112	IODEVICETYPE_ETHERNETNVPROT	EtherCAT Automation Protocol (Network variables)
144	IODEVICETYPE_ETHERCATSIMULATION	EtherCAT-Simulation

Devices: Beckhoff Lightbus Master/Slave

Sub type	Tag	Description
67	IODEVICETYPE_CX1500_M200	PC104 Lightbus-Master
68	IODEVICETYPE_CX1500_B200	PC104 Lightbus-Slave
36	IODEVICETYPE_FC200X	Beckhoff-Lightbus-I/II-PCI-Card
114	IODEVICETYPE_EL6720	Beckhoff-Lightbus-EtherCAT-Klemme
1	IODEVICETYPE_C1220	Beckhoff Lightbus ISA interface card C1220
2	IODEVICETYPE_C1200	Beckhoff Lightbus ISA interface card C1200

Devices: Beckhoff Profibus Master/Slave

Sub type	Tag	Description
69	IODEVICETYPE_CX1500_M310	PC104 ProfiBus-Master
70	IODEVICETYPE_CX1500_B310	PC104 ProfiBus-Slave
33	IODEVICETYPE_PBMON	Beckhoff-PROFIBUS-Monitor
38	IODEVICETYPE_FC3100	Beckhoff-Profibus-PCI-Card
56	IODEVICETYPE_FC3100MON	Beckhoff-Profibus-Monitor-PCI-Karte
60	IODEVICETYPE_FC3100SLV	Beckhoff-Profibus-PCI-Karte als Slave
79	IODEVICETYPE_BX_B310	BX ProfiBus-Slave
83	IODEVICETYPE_BC3150	BCxx50 ProfiBus-Slave
86	IODEVICETYPE_EL6731	Beckhoff-Profibus-EtherCAT-Klemme
97	IODEVICETYPE_EL6731SLV	Beckhoff-Profibus-Slave-EtherCAT-Klemme

Devices: Beckhoff CANopen Master/Slave

Sub type	Tag	Description
71	IODEVICETYPE_CX1500_M510	PC104 CANopen-Master
72	IODEVICETYPE_CX1500_B510	PC104 CANopen-Slave
39	IODEVICETYPE_FC5100	Beckhoff-CanOpen-PCI-Card
58	IODEVICETYPE_FC5100MON	Beckhoff-CANopen-Monitor-PCI-Karte
61	IODEVICETYPE_FC5100SLV	Beckhoff-CanOpen-PCI-Karte als Slave
81	IODEVICETYPE_BX_B510	BX CANopen-Slave
84	IODEVICETYPE_BC5150	BCxx50 CANopen-Slave
87	IODEVICETYPE_EL6751	Beckhoff-CanOpen-EtherCAT-Klemme
98	IODEVICETYPE_EL6751SLV	Beckhoff-CanOpen-Slave-EtherCAT-Klemme

Devices: Beckhoff DeviceNet Master/Slave

Sub type	Tag	Description
73	IODEVICETYPE_CX1500_M520	PC104 DeviceNet-Master
74	IODEVICETYPE_CX1500_B520	PC104 DeviceNet-Slave
41	IODEVICETYPE_FC5200	Beckhoff-DeviceNet-PCI-Card
59	IODEVICETYPE_FC5200MON	Beckhoff-DeviceNet-Monitor-PCI-Karte
62	IODEVICETYPE_FC5200SLV	Beckhoff-DeviceNet-PCI-Karte als Slave
82	IODEVICETYPE_BX_B520	BX DeviceNet-Slave
85	IODEVICETYPE_BC5250	BCxx50 DeviceNet-Slave
88	IODEVICETYPE_EL6752	Beckhoff-DeviceNet-EtherCAT-Klemme
99	IODEVICETYPE_EL6752SLV	Beckhoff-DeviceNet-Slave-EtherCAT-Klemme

Devices: Beckhoff Sercos Master/Slave

Sub type	Tag	Description
75	IODEVICETYPE_CX1500_M750	PC104 Sercos-Master
76	IODEVICETYPE_CX1500_B750	PC104 Sercos-Slave
48	IODEVICETYPE_FC7500	Beckhoff-SERCOS-PCI-Card

Devices: Ethernet

Sub type	Tag	Description
66	IODEVICETYPE_ENETRTMP	Ethernet Real Time Miniport
109	IODEVICETYPE_ENETADAPTER	Real-Time Ethernet Adapter (Multiple Protocol Handler)
138	---	Real-Time Ethernet Protocol (BK90xx, AX2000-B900)
45	IODEVICETYPE_ETHERNET	Virtual Ethernet Interface

Devices: USB

Sub type	Tag	Description
57	IODEVICETYPE_USB	Virtual USB Interface
125	---	Virtual USB Interface (Remote)

Devices: Hilscher

Sub type	Tag	Description
4	IODEVICETYPE_CIF30DPM	ISA ProfiBus-Master 2 kByte (Hilscher Card)
5	IODEVICETYPE_CIF40IBSM	ISA Interbus-S-Master 2 kByte (Hilscher-Card)
12	IODEVICETYPE_CIF30CAN	ISA CANopen-Master (Hilscher-Card)
13	IODEVICETYPE_CIF30PB	ISA ProfiBus-Master 8 kByte (Hilscher-Card)
16	IODEVICETYPE_CIF30IBM	ISA Interbus-S-Master (Hilscher-Card)
17	IODEVICETYPE_CIF30DNM	ISA DeviceNet-Master (Hilscher-Card)
19	IODEVICETYPE_CIF50PB	PCI ProfiBus-Master 8 kByte (Hilscher-Card)
20	IODEVICETYPE_CIF50IBM	PCI Interbus-S-Master (Hilscher-Card)
21	IODEVICETYPE_CIF50DNM	PCI DeviceNet-Master (Hilscher-Card)
22	IODEVICETYPE_CIF50CAN	PCI CANopen-Master (Hilscher-Card)
23	IODEVICETYPE_CIF60PB	PCMCIA ProfiBus-Master (Hilscher-Card)
24	IODEVICETYPE_CIF60DNM	PCMCIA DeviceNet-Master (Hilscher-Card)
25	IODEVICETYPE_CIF60CAN	PCMCIA CANopen-Master (Hilscher-Card)
26	IODEVICETYPE_CIF104DP	PC104 ProfiBus-Master 2 kByte (Hilscher-Card)
27	IODEVICETYPE_C104PB	PC104 ProfiBus-Master 8 kByte (Hilscher-Card)
28	IODEVICETYPE_C104IBM	PC104 Interbus-S-Master 2 kByte (Hilscher-Card)
29	IODEVICETYPE_C104CAN	PC104 CANopen-Master (Hilscher-Card)
30	IODEVICETYPE_C104DNM	PC104 DeviceNet-Master (Hilscher-Card)
35	IODEVICETYPE_CIF60IBM	PCMCIA Interbus-S-Master (Hilscher-Card)
49	IODEVICETYPE_CIF30IBS	ISA Interbus-S-Slave (Hilscher-Card)
50	IODEVICETYPE_CIF50IBS	PCI Interbus-S-Slave (Hilscher-Card)
51	IODEVICETYPE_C104IBS	PC104 Interbus-S-Slave (Hilscher-Card)
89	IODEVICETYPE_COMPB	COM ProfiBus-Master 8 kByte (Hilscher-Karte)
90	IODEVICETYPE_COMIBM	COM Interbus-S-Master (Hilscher-Karte)
91	IODEVICETYPE_COMDNM	COM DeviceNet-Master (Hilscher-Karte)
92	IODEVICETYPE_COMCAN	COM CANopen-Master (Hilscher-Karte)
93	IODEVICETYPE_COMIBS	COM CANopen-Slave (Hilscher-Karte)

Sub type	Tag	Description
100	IODEVICETYPE_C104PPB	PC104+ ProfiBus-Master 8 kByte (Hilscher-Karte)
101	IODEVICETYPE_C104PCAN	PC104+ CANopen-Master (Hilscher-Karte)
102	IODEVICETYPE_C104PDNM	PC104+ DeviceNet-Master (Hilscher-Karte)

Devices: Profinet / Profibus

Sub type	Tag	Description
3	IODEVICETYPE_SPC3	ProfiBus Slave (Siemens Card)
7	IODEVICETYPE_CP5412A2	ProfiBus-Master (Siemens-Card)
34	IODEVICETYPE_CP5613	PCI ProfiBus-Master (Siemens-Card)
113	IODEVICETYPE_PROFINETIOCONTROLLER	PROFINET Master
115	IODEVICETYPE_PROFINETIODEVICE	PROFINET Slave

Devices: Indramat

Sub type	Tag	Description
8	IODEVICETYPE_SERCANSISA	Sercos Master (Indramat)

Devices: Phoenix

Sub type	Tag	Description
15	IODEVICETYPE_IBSSCIT	Interbus-S-Master (Phoenix-Card)
47	IODEVICETYPE_IBSSCRIRTLK	Interbus-S-Master with Slave-Part LWL Basis (Phoenix-Card)
63	IODEVICETYPE_IBSSCITPCI	PCI Interbus-S-Master (Phoenix-Karte)
64	IODEVICETYPE_IBSSCRILKPCI	PCIInterbus-S-Master mit Slave-Teil auf LWL Basis (Phoenix-Karte)
80	IODEVICETYPE_IBSSCRIRTPCI	PCIInterbus-S-Master mit Slave-Teil auf Kupfer Basis (Phoenix-Karte)

5.3.3.4 Boxes

5.3.3.4.1 ITcSmTreeItem Item Sub Types: Boxes

Sub type	Tag	Description
0	BOXTYPE_UNKNOWN	---
1	BOXTYPE_BK2000	BK2000 Lightbus coupler
2	BOXTYPE_M1400	M1400 Lightbus digital input/output module
3	BOXTYPE_M2400	M2400 Lightbus input/output module
4	BOXTYPE_M3120_1	M3120 Lightbus incremental encoder interface
5	BOXTYPE_M3120_2	M3120 Lightbus incremental encoder interface
6	BOXTYPE_M3120_3	M3120 Lightbus incremental encoder interface
7	BOXTYPE_M3120_4	M3120 Lightbus incremental encoder interface
8	BOXTYPE_M3000	M3000 absolute / incremental encoder
9	BOXTYPE_C1120	---
10	BOXTYPE_BK2010	BK2010 Lightbus coupler
11	BOXTYPE_AX2000	---
12	BOXTYPE_M2510	---
20	BOXTYPE_BK2020	BK2020 Lightbus coupler
21	BOXTYPE_BC2000	---
31	BOXTYPE_FOX20	---
32	BOXTYPE_FOX50	---
33	BOXTYPE_FOXRK001	---
34	BOXTYPE_FOXRK002	---
35	BOXTYPE_CP1001	---
40	BOXTYPE_IPXB2	---
41	BOXTYPE_ILXB2	---
42	BOXTYPE_ILXC2	---
50	BOXTYPE_TSMBOX_200	---
51	BOXTYPE_BX2000	---
52	BOXTYPE_CX1500_B200	---
1001	BOXTYPE_BK3000	---
1002	BOXTYPE_BK3100	---
1003	BOXTYPE_PBDP_GSD	---
1004	BOXTYPE_BK3010	---
1005	BOXTYPE_BK3110	---
1006	BOXTYPE_BK3500	---
1007	BOXTYPE_LC3100	---
1008	BOXTYPE_PBDP_DRIVE	---
1009	BOXTYPE_BK3020	---
1010	BOXTYPE_BK3120	---
1011	BOXTYPE_BC3100	---
1012	BOXTYPE_PBDP_DRIVE2	---
1013	BOXTYPE_PBDP_DRIVE3	---
1014	BOXTYPE_PBDP_DRIVE4	---
1015	BOXTYPE_PBDP_DRIVE5	---
1016	BOXTYPE_PBDP_DRIVE6	---
1017	BOXTYPE_PBDP_DRIVE7	---

Sub type	Tag	Description
1018	BOXTYPE_PBDP_DRIVE8	---
1019	BOXTYPE_BK3150	---
1020	BOXTYPE_BC3150	---
1021	BOXTYPE_BK3XXX	---
1022	BOXTYPE_BC3XXX	---
1030	BOXTYPE_IPXB3	---
1031	BOXTYPE_ILXB3	---
1032	BOXTYPE_ILXC3	---
1040	BOXTYPE_TSMBOX_310	---
1041	BOXTYPE_BX3100	---
1042	BOXTYPE_CX1500_B310	---
1043	BOXTYPE_FC310X_SLAVE	---
1044	BOXTYPE_EL6731_SLAVE	---
1051	BOXTYPE_AX2000_B310	---
1100	BOXTYPE_TCPBDPSLAVE	---
1101	BOXTYPE_TCFDLAGAG	---
1102	BOXTYPE_TCMPI	---
1103	BOXTYPE_TCPBMCSLAVE	---
1104	BOXTYPE_TCPBMCSLAVE2	---
1105	BOXTYPE_TCPBMCSLAVE3	---
1106	BOXTYPE_TCPBMCSLAVE4	---
1107	BOXTYPE_TCPBMCSLAVE5	---
1108	BOXTYPE_TCPBMCSLAVE6	---
1109	BOXTYPE_TCPBMCSLAVE7	---
1110	BOXTYPE_TCPBMCSLAVE8	---
1111	BOXTYPE_TCPBMONSLAVE	---
2001	BOXTYPE_BK4000	---
2002	BOXTYPE_IBS_GENERIC	---
2003	BOXTYPE_IBS_BK	---
2004	BOXTYPE_BK4010	---
2005	BOXTYPE_BK4500	---
2006	BOXTYPE_BK4510	---
2007	BOXTYPE_IBS_SLAVEBOX	---
2008	BOXTYPE_BC4000	---
2009	BOXTYPE_BK4020	---
2020	BOXTYPE_CP2020	---
2030	BOXTYPE_IPXB4	---
2031	BOXTYPE_ILXB4	---
2032	BOXTYPE_ILXC4	---
3001	BOXTYPE_SERCOSAXIS	---
3011	BOXTYPE_BK7500	---
3012	BOXTYPE_BK7510	---
3013	BOXTYPE_BK7520	---
3021	BOXTYPE_SERCOSMASTERBOX	---
3031	BOXTYPE_SERCOSSLAVEBOX	---
4001	BOXTYPE_BK8100	BK8100 bus coupler
4002	BOXTYPE_BK8110	BK8110 bus coupler
4003	BOXTYPE_BK8000	BK8000 bus coupler

Sub type	Tag	Description
4004	BOXTYPE_BK8010	BK8010 bus coupler
4005	BOXTYPE_CP9040	CP9040 PCB
4011	BOXTYPE_BC8000	BC8000 bus terminal controller
4012	BOXTYPE_BC8100	BC8100 bus terminal controller
4030	BOXTYPE_IPXB80	---
4031	BOXTYPE_ILXB80	---
4032	BOXTYPE_ILXC80	---
4040	BOXTYPE_IPXB81	---
4041	BOXTYPE_ILXB81	---
4042	BOXTYPE_ILXC81	---
4050	BOXTYPE_BC8150	BC8150 bus terminal controller
5001	BOXTYPE_BK5100	BK5100 bus coupler
5002	BOXTYPE_BK5110	BK5110 bus coupler
5003	BOXTYPE_CANNODE	---
5004	BOXTYPE_BK5120	BK5120 bus coupler
5005	BOXTYPE_LC5100	---
5006	BOXTYPE_CANDRIVE	---
5007	BOXTYPE_AX2000_B510	---
5008	BOXTYPE_BK5150	BK5150 bus coupler
5009	BOXTYPE_BK5151	BK5151 bus coupler
5011	BOXTYPE_BC5150	BC5150 bus terminal controller
5030	BOXTYPE_IPXB51	---
5031	BOXTYPE_ILXB51	---
5032	BOXTYPE_ILXC51	---
5040	BOXTYPE_TSMBOX_510	---
5041	BOXTYPE_BX5100	BX5100 CANopen bus terminal controller
5042	BOXTYPE_CX1500_B510	---
5043	BOXTYPE_FC510XSLV	---
5050	BOXTYPE_TCCANSLAVE	---
5051	BOXTYPE_CANQUEUE	CAN Interface
5201	BOXTYPE_BK5200	---
5202	BOXTYPE_BK5210	---
5203	BOXTYPE_DEVICENET	---
5204	BOXTYPE_BK5220	---
5205	BOXTYPE_LC5200	---
5211	BOXTYPE_BK52XX	---
5212	BOXTYPE_BC52XX	---
5230	BOXTYPE_IPXB52	---
5231	BOXTYPE_ILXB52	---
5232	BOXTYPE_ILXC52	---
5250	BOXTYPE_TCDNSLAVE	---
9001	BOXTYPE_BK9000	BK9000 Ethernet TCP/IP bus coupler
9002	BOXTYPE_BK9100	BK9100 Ethernet TCP/IP bus coupler
9005	BOXTYPE_BK9050	BK9050 Ethernet TCP/IP bus coupler
9011	BOXTYPE_BC9000	BC9000 Ethernet TCP/IP bus terminal controller

Sub type	Tag	Description
9012	BOXTYPE_BC9100	BC9100 Ethernet TCP/IP bus terminal controller
9013	BOXTYPE_BX9000	BX9000 Ethernet TCP/IP bus terminal controller
9014	BOXTYPE_BX9000SLV	---
9015	BOXTYPE_BC9050	BC9050 Ethernet TCP/IP bus terminal controller
9016	BOXTYPE_BC9050SLV	BC9050 Ethernet TCP/IP bus terminal controller slave
9017	BOXTYPE_BC9120	BC9120 Ethernet TCP/IP bus terminal controller
9018	BOXTYPE_BC9120SLV	BC9120 Ethernet TCP/IP bus terminal controller slave
9019	BOXTYPE_BC9020	BC9020 Ethernet TCP/IP bus terminal controller
9020	BOXTYPE_BC9020SLV	BC9020 Ethernet TCP/IP bus terminal controller slave
9030	BOXTYPE_IPXB9	---
9031	BOXTYPE_ILXB9	---
9032	BOXTYPE_ILXC9	---
9041	BOXTYPE_REMOTETASK	---
9051	BOXTYPE_NV_PUB	Network Publisher.
9052	BOXTYPE_NV_SUB	Network Subscriber.
9053	BOXTYPE_NV_PUBVAR	Publisher variable.
9054	BOXTYPE_NV_SUBVAR	Subscriber variable.
9055	BOXTYPE_NV_PUBDATA	Publisher data object.
9056	BOXTYPE_NV_SUBDATA	Subscriber data object.
9061	BOXTYPE_AX2000_B900	---
9071	BOXTYPE_FLB_FRAME	---
9081	BOXTYPE_BK1120	---
9085	BOXTYPE_IPXB11	---
9086	BOXTYPE_ILXB11	---
9087	BOXTYPE_ILXC11	---
9105	BOXTYPE_FSOESLAVE	---
9121	BOXTYPE_PNIODEVICE	---
9122	BOXTYPE_PNIOTCDEVICE	---
9123	BOXTYPE_PNIODEVICEINTF	Profinet TwinCAT Device Interface
9124	BOXTYPE_PNIO_DRIVE	---
9125	BOXTYPE_PNIOBK9103	---
9126	BOXTYPE_PNIOILB903	---
9132	BOXTYPE_BK9053	BK9053 K-Bus coupler, PROFINET IO RT
9133	BOXTYPE_EIPSLAVEINTF	---
9143	BOXTYPE_PTPSLAVEINTF	---
9151	BOXTYPE_RAWUDPINTF	---
9500	BOXTYPE_BK9500	BK9500
9510	BOXTYPE_BK9510	BK9510
9520	BOXTYPE_BK9520	BK9520
9591	BOXTYPE_CPX8XX	---
9700	BOXTYPE_CX1102	---

Sub type	Tag	Description
9701	BOXTYPE_CX1103	---
9702	BOXTYPE_CX1190	---

5.3.3.5 Terminals

5.3.3.5.1 ITcSmTreeItem Item Sub Types: Terminals: K-Bus digital input (KL1)

Sub type	Description
1002	KL1002 2-Channel digital input terminal
1012	KL1012 2-Channel digital input terminal
1032	KL1032 2-Channel digital input terminal
1052	KL1052 2-Channel digital input terminal
1104	KL1104 4-Channel digital input terminal
1114	KL1114 4-Channel digital input terminal
1124	KL1124 4-Channel digital input terminal
1154	KL1154 4-Channel digital input terminal
1164	KL1164 4-Channel digital input terminal
1184	KL1184 4-Channel digital input terminal
1194	KL1194 4-Channel digital input terminal
1212	KL1212 2-Channel digital input terminal
1232	KL1232 2-Channel digital input terminal
1302	KL1302 2-Channel digital input terminal
1304	KL1304 4-Channel digital input terminal
1312	KL1312 2-Channel digital input terminal
1314	KL1314 4-Channel digital input terminal
1352	KL1352 2-Channel digital input terminal
1362	KL1362 2-Channel digital input terminal
1382	KL1382 2-Channel digital input terminal
1402	KL1402 2-Channel digital input terminal
1404	KL1404 4-Channel digital input terminal
1408	KL1408 8-Channel digital input terminal
1412	KL1412 2-Channel digital input terminal
1414	KL1414 4-Channel digital input terminal
1418	KL1418 8-Channel digital input terminal
1434	KL1434 4-Channel digital input terminal
1488	KL1488 8-Channel digital input terminal
1498	KL1498 8-Channel digital input terminal
1501	KL1501 1-Channel digital input terminal
1512	KL1512 2-Channel digital input terminal
1702	KL1702 2-Channel digital input terminal
1712	KL1712 2-Channel digital input terminal
16778928	KL1712-0060 2-Channel digital input terminal
1722	KL1722 2-Channel digital input terminal
1804	KL1804 4-Channel digital input terminal
1808	KL1808 8-Channel digital input terminal
1809	KL1809 16-Channel digital input terminal
1814	KL1814 4-Channel digital input terminal
1819	KL1819 16-Channel digital input terminal
1859	KL1859 8-Channel digital input terminal
1862	KL1862 16-Channel digital input terminal
16779078	KL1862-0010 16-Channel digital input terminal
1872	KL1872 16-Channel digital input terminal
1889	KL1889 16-Channel digital input terminal
1202	KL1202 2-Channel digital input terminal

5.3.3.5.2 ITcSmTreeItem Item Sub Types: Terminals: K-Bus digital output (KL2)

Sub type	Description
2408	KL2408 8-Channel digital output terminal
2012	KL2012 2-Channel digital output terminal
2022	KL2022 2-Channel digital output terminal
2032	KL2032 2-Channel digital output terminal
2114	KL2114 4-Channel digital output terminal
2124	KL2124 4-Channel digital output terminal
2134	KL2134 4-Channel digital output terminal
2184	KL2184 4-Channel digital output terminal
2212	KL2212 2-Channel digital output terminal
2404	KL2404 4-Channel digital output terminal
2212	KL2212 2-Channel digital output terminal
2404	KL2404 4-Channel digital output terminal
2408	KL2408 8-Channel digital output terminal
2284	KL2284 4-Channel digital output terminal
2424	KL2424 4-Channel digital output terminal
2442	KL2442 2-Channel digital output terminal
2488	KL2488 8-Channel digital output terminal
2502	KL2502 2-Channel PWM output terminal
2512	KL2512 2-Channel PWM output terminal
2521	KL2521 1-Channel Pulse Train output terminal
2531	KL2531 1-Channel Stepper Motor terminal
16779747	KL2531-1000 1-Channel Stepper Motor terminal
2532	KL2532 2-Channel DC Motor amplifier output terminal
2535	KL2535 2-Channel PWM amplifier output terminal
2541	KL2541 1-Channel Stepper Motor terminal
16779757	KL2541-1000 1-Channel Stepper Motor terminal
2542	KL2542 2-Channel DC Motor amplifier terminal
2545	KL2545 2-Channel PWM amplifier output terminal
2552	KL2552 2-Channel DC Motor amplifier terminal
2602	KL2602 2-Channel output relay terminal
2612	KL2612 2-Channel output relay terminal
2622	KL2622 2-Channel output relay terminal
2631	KL2631 1-Channel power output relay terminal
2641	KL2641 1-Channel power output relay terminal
2652	KL2652 2-Channel power output relay terminal
2701	KL2701 1-Channel solid state load relay terminal
2702	KL2702 2-Channel output solid sate relay terminal
16779918	KL2702-0002 2-Channel output solid state relay terminal
33557134	KL2702-0020 2-Channel output solid state relay terminal
2712	KL2712 2-Channel triac output terminal
2722	KL2722 2-Channel triac output terminal
2732	KL2732 2-Channel triac output terminal
2744	KL2744 4-Channel output solid state relay
2751	KL2751 1-Channel universal dimmer terminal
33557183	KL2751-1200 1-Channel universal dimmer terminal
2761	KL2761 1-Channel universal dimmer terminal

Sub type	Description
2784	KL2784 4-Channel output terminal
2791	KL2791 1-Channel speed controller terminal
33557223	KL2791-1200 1-Channel speed controller terminal
2794	KL2794 4-Channel output terminal
2808	KL2808 8-Channel output terminal
2809	KL2809 16-Channel output terminal
2872	KL2872 16-Channel output terminal
2889	KL2889 16-Channel output terminal

5.3.3.5.3 ITcSmTreeItem Item Sub Types: Terminals: K-Bus analog input (KL3)

Sub type	Description
3001	KL3001 1-Channel analog input terminal
3002	KL3002 3-Channel analog input terminal
3011	KL3011 1-Channel analog input terminal
3012	KL3012 2-Channel analog input terminal
3021	KL3021 1-Channel analog input terminal
3022	KL3022 2-Channel analog input terminal
3041	KL3041 1-Channel analog input terminal
3042	KL3042 2-Channel analog input terminal
3044	KL3044 4-Channel analog input terminal
3051	KL3051 1-Channel analog input terminal
3052	KL3052 2-Channel analog input terminal
3054	KL3054 4-Channel analog input terminal
3061	KL3061 1-Channel analog input terminal
3062	KL3062 2-Channel analog input terminal
3064	KL3064 4-Channel analog input terminal
3102	KL3102 2-Channel analog input terminal
3112	KL3112 2-Channel analog input terminal
3122	KL3122 2-Channel analog input terminal
3132	KL3132 2-Channel analog input terminal
3142	KL3142 2-Channel analog input terminal
3152	KL3152 2-Channel analog input terminal
3158	KL3158 8-Channel analog input terminal
3162	KL3162 2-Channel analog input terminal
3172	KL3172 2-Channel analog input terminal
33557604	KL3172-0500 2-Channel analog input terminal
67112036	KL3172-1000 2-Channel analog input terminal
3182	KL3182 2-Channel analog input terminal
3201	KL3201 1-Channel analog input terminal
3202	KL3202 2-Channel analog input terminal
3204	KL3204 4-Channel analog input terminal
33557640	KL3208-0010 8-Channel analog input terminal
3222	KL3222 2-Channel analog input terminal
3228	KL3228 8-Channel analog input terminal
3302	KL3302 2-Channel analog input terminal
3311	KL3311 1-Channel analog input terminal
3312	KL3312 2-Channel analog input terminal
3314	KL3314 4-Channel analog input terminal
3351	KL3351 1-Channel resistor bridge terminal
50334999	KL3351-0001 1-Channel resistor bridge terminal
3356	KL3356 1-Channel precise resistor bridge terminal
3361	KL3361 1-Channel oscilloscope terminal
3362	KL3362 2-Channel oscilloscope terminal
3403	KL3403 3-Phase power measuring terminal
3404	KL3404 4-Channel analog input terminal
3408	KL3408 8-Channel analog input terminal
3444	KL3444 4-Channel analog input terminal
3448	KL3448 8-Channel analog input terminal
3454	KL3454 4-Channel analog input terminal

Sub type	Description
3458	KL3458 8-Channel analog input terminal
3464	KL3464 4-Channel analog input terminal
3468	KL3468 8-Channel analog input terminal

5.3.3.5.4 ITcSmTreeItem Item Sub Types: Terminals: K-Bus analog output (KL4)

Sub type	Description
4001	KL4001 1-Channel analog output terminal
4002	KL4002 2-Channel analog output terminal
4004	KL4004 4-Channel analog output terminal
4011	KL4011 1-Channel analog output terminal
4012	KL4012 2-Channel analog output terminal
4021	KL4021 1-Channel analog output terminal
4022	KL4022 2-Channel analog output terminal
4031	KL4031 1-Channel analog output terminal
4032	KL4032 2-Channel analog output terminal
4034	KL4034 4-Channel analog output terminal
4112	KL4112 2-Channel analog output terminal
4122	KL4122 2-Channel analog output terminal
4132	KL4132 2-Channel analog output terminal
4404	KL4404 4-Channel analog output terminal
4408	KL4408 8-Channel analog output terminal
4414	KL4414 4-Channel analog output terminal
4418	KL4418 8-Channel analog output terminal
4424	KL4424 4-Channel analog output terminal
4428	KL4428 8-Channel analog output terminal
4434	KL4434 4-Channel analog output terminal
4438	KL4438 8-Channel analog output terminal
4494	KL4494 2-Channel analog output terminal

5.3.3.5.5 ITcSmTreeItem Item Sub Types: Terminals: K-Bus measuring (KL5)

Sub type	Description
5001	KL5001 1-Channel SSI encoder terminal
5051	KL5051 1-Channel Bi-SSI encoder terminal
16782267	KL5051-0010 1-Channel Bi-SSI encoder terminal
5101	KL5101 incremental encoder 5V terminal
5111	KL5111 incremental encoder 24V terminal
5121	KL5121 4-Channel line-motion-controller terminal
5151	KL5151 1-Channel incremental encoder terminal
33559583	KL5151-0021 1-Channel incremental encoder terminal
16782367	KL5151-0050 2-Channel incremental encoder terminal
5152	KL5152 2-Channel incremental encoder terminal

5.3.3.5.6 ITcSmTreeItem Item Sub Types: Terminals: K-Bus communication (KL6)

Sub type	Description
16783217	KL6001 interface RS232C terminal
50337649	KL6001 interface RS232 terminal
16783227	KL6011 interface TTY terminal
50337659	KL6011 interface TTY terminal
16783237	KL6021 interface RS422/485 terminal
50337669	KL6021 interface RS485 terminal
100669317	KL6021 interface RS485 terminal
100669327	KL6031 interface RS232 terminal
50337679	KL6031 interface RS232 terminal
100669337	KL6041 interface RS485 terminal
50337689	KL6041 interface RS485 terminal
16783257	KL6041-0100 interface RS485 terminal
6051	KL6051 data exchange terminal
335550521	KL6201 ASI-Master terminal
369104953	KL6201 ASI-Master terminal
402659385	KL6201 ASI-Master terminal
335550531	KL6211 ASI-Master terminal
369104963	KL6211 ASI-Master terminal
402659395	KL6211 ASI-Master terminal
6224	KL6224 I/O-Link Master terminal
16783440	KL6224 I/O-Link Master terminal
33560656	KL6224 I/O-Link Master terminal
50337872	KL6224 I/O-Link Master terminal
33560733	KL6301 EIB terminal
33560833	KL6401 LON terminal
33561013	KL6581 EnOcean terminal
33561203	KL6771 MP-Bus Master terminal
6781	KL6781 M-Bus interface terminal
6811	KL6811 DALI-Master terminal
6821	KL6821 eDALI-Master terminal

5.3.3.5.7 ITcSmTreeItem Item Sub Types: Terminals: K-Bus power (KL8)

Sub type	Description
33562433	KL8001 1-Channel power terminal
8001	KL8001 3-Channel power terminal
8519	KL8519 16 digital input terminal
8524	KL8524 2x4 digital output terminal
8528	KL8528 8 digital output terminal
8548	KL8548 8-Channel analog output terminal
8610	KL8610 1-Channel adapter terminal KL8601
16785826	KL8610 2-Channel adapter terminal KL8601
33563042	KL8610 3-Channel adapter terminal KL8601
50340258	KL8610 4-Channel adapter terminal KL8601
67117474	KL8610 5-Channel adapter terminal KL8601
83894690	KL8610 6-Channel adapter terminal KL8601
100671906	KL8610 7-Channel adapter terminal KL8601
117449122	KL8610 8-Channel adapter terminal KL8601

5.3.3.5.8 ITcSmTreeItem Item Sub Types: Terminals: K-Bus system (KL9)

Sub type	Description
9010	KL9010 end terminal
9020	KL9020 bus extension end terminal
9050	KL9050 bus extension coupler terminal
9060	KL9060 adapter terminal
9070	KL9070 shield terminal
9080	KL9080 separation terminal
9100	KL9100 power supplier terminal
9110	KL9110 power supplier terminal
9150	KL9150 power supplier terminal
9160	KL9160 power supplier terminal
9180	KL9180 potential connection terminal
9181	KL9181 potential connection terminal
9182	KL9182 potential connection terminal
9183	KL9183 potential connection terminal
9184	KL9184 potential connection terminal
9185	KL9185 potential connection terminal
9186	KL9186 potential connection terminal
9187	KL9187 potential connection terminal
9188	KL9188 potential connection terminal
9189	KL9189 potential connection terminal
9190	KL9190 power feed terminal
9195	KL9195 shield terminal
9200	KL9200 power supplier terminal
9210	KL9210 power supplier terminal
9250	KL9250 power supplier terminal
9260	KL9260 power supplier terminal
9290	KL9290 power supplier terminal
9300	KL9300 4-Channel diode array terminal
9301	KL9301 7-Channel diode array terminal
9302	KL9302 7-Channel diode array terminal
9309	KL9309 interface terminal for KL85xx
9400	KL9400 K-Bus power supplier terminal
9505	KL9505 power supplier terminal
167781665	KL9505-0010 power suppler terminal
9508	KL9508 power supplier terminal
167781668	KL9508-0010 power supplier terminal
9510	KL9510 power supplier terminal
167781670	KL9510-0010 power supplier terminal
9512	KL9512 power supplier terminal
167781672	KL9512-0010 power supplier terminal
9515	KL9515 power supplier terminal
167781675	KL9515-0010 power supplier terminal
9528	KL9528 power supplier terminal
9540	KL9540 surge filter field supply terminal
9550	KL9550 surge filter system and field supply terminal
9560	KL9560 power supplier terminal
9570	KL9570 buffer capacitor terminal

5.3.3.5.9 ITcSmTreeItem Item Sub Types: Terminals: K-Bus safety (KLx90x)

Sub type	Description
1904	KL1904 4-Channel safety input terminal
1908	KL1908 8-Channel safety input terminal
2904	KL2904 4-Channel safety output terminal
6904	KL6904 safety logic (7 TwinSAFE connections) terminal
16784120	KL6904 safety logic (15 TwinSAFE connections) terminal

5.3.3.6 Modules

5.3.3.6.1 ITcSmTreeItem Item Sub Types: Modules: K-Bus digital input (KM1)

Sub type	Description
838861802	KM1002 16-Channel digital input module
838861804	KM1004 32-Channel digital input module
838861808	KM1008 64-Channel digital input module
838861812	KM1012 16-Channel digital input module
838861814	KM1014 32-Channel digital input module
838861818	KM1018 64-Channel digital input module
838862444	KM1644 4-Channel digital input module

5.3.3.6.2 ITcSmTreeItem Item Sub Types: Modules: K-Bus digital output (KM2)

Sub type	Description
838862802	KM2002 16-Channel digital output module
838862804	KM2004 32-Channel digital output module
838862808	KM2008 64-Channel digital output module
838862822	KM2022 16-Channel digital output module
838862842	KM2042 16-Channel digital output module
838863404	KM2604 4-Channel digital output relay module
838863414	KM2614 4-Channel digital output relay module
838863442	KM2642 2-Channel digital power output relay module
838863452	KM2652 2-Channel digital power output relay module
16779990	KM2774 4-Channel blinds control terminal
2774	KM2774-1001 4-Channel blinds control terminal

5.3.3.6.3 ITcSmTreeItem Item Sub Types: Modules: K-Bus analog input (KM3)

Sub type	Description
838864501	KM3701 1-Channel differential pressure measuring
872418933	KM3701-0340 1-Channel differential pressure measuring
838864502	KM3702 2-Channel absolute pressure measuring
838864512	KM3712 2-Channel absolute pressure measuring

5.3.3.6.4 ITcSmTreeItem Item Sub Types: Modules: K-Bus analog output (KM4)

Sub type	Description
838865402	KM4602 2-Channel analog output terminal

5.3.3.6.5 ITcSmTreeItem Item Sub Types: Modules: K-Bus communication (KM6)

Sub type	Description
6551	KM6551 IEEE802.15.4 terminal
838867463	KM6663 Ethernet changeover switch terminal

5.3.4 ITcSmTreeItem::ProduceXml

The ProduceXml() method returns a XML string with item specific information and parameter.

```
HRESULT ProduceXml(VARIANT_BOOLbRecursive, BSTR* pXML);
```

Parameters

bRecursive	[in, defaultvalue(0)] Optional parameter for future use.
pXML	[out, retval] Contains the XML representation of the item specific data and parameter.

Return Values

S_OK	function returns successfully.
E_POINTER	pXML pointer is invalid.

Comments

The following XML string is an incomplete example of the resulting information if the tree item is a I/O device of the type *SERCOS Master/Slave FC750x*. This string can be used as input for the IXMLDOMDocument::loadXML method to create a XML DOM document.

```
<?xml version="1.0"?>
<TreeItem>
<ItemName>Device 1 (FC750x)</ItemName>
<PathName>TIID^Device 1 (FC750x)</PathName>
<ItemType>2</ItemType>
<ItemId>1</ItemId>
<ItemSubType>48</ItemSubType>
<ItemSubTypeName>SERCOS Master/Slave FC750x, PCI [Beckhoff FC7502 PCI]</ItemSubTypeName>
<ChildCount>0</ChildCount>
<Disabled>0</Disabled>
<DeviceDef>
<AmsPort>0</AmsPort>
<AddressInfo>
<Pci>
<BusNo>0</BusNo>
<SlotNo>16</SlotNo>
<IrqNo>9</IrqNo>
<FcChannel>0</FcChannel>
</Pci>
</AddressInfo>
<SercosMaster>
<Baudrate>0</Baudrate>
<OperationMode>0</OperationMode>
<SendPower>1</SendPower>
<AccessTime>200</AccessTime>
<ShiftTime>50</ShiftTime>
<StartupToPhase4>1</StartupToPhase4>
<CheckTiming>1</CheckTiming>
</SercosMaster>
</DeviceDef>
</TreeItem>
```

See also

ITcSmTreeItem::ConsumeXML [▶ 153]

5.3.5 ITcSmTreeItem::ConsumeXml

ITcSmTreeItem

The ConsumeXml() method consumes a BSTR containing the XML representation with item specific data and updates found parameters. This method is used to change item parameters not directly accessible by the ITcSmTreeItem interface.

```
HRESULT ConsumeXml(BSTRbstrXML);
```

Parameters

bstrXML	[in] string with the XML representation of the item specific parameter. The corresponding parameter will be updated in the System Manager database

Return Values

S_OK	function returns successfully.
E_FAIL	the *bstrXML* string does not contain a valid xml document.

Comments

The document can only contain the specific parameter that should be changed. The document structure has to fit to the item specific XML tree, but parameter that should not be changed can be omitted. The following document is a minimal example that can be used to change the specific parameter *CheckNumberBoxes* of the item (in this case a C1220 fieldbus card). If the parameters in the document are not known by the item, they will be ignored.

```
<TreeItem><DeviceDef><DevC1220Def><CheckNumberBoxes>0</CheckNumberBoxes></
DevC1220Def></DeviceDef></TreeItem>
```

The set of parameter of a specific tree item is defined in the xml schema document that comes with the TwinCAT System Manager. The parameter of a specific item can also evaluated by calling the ITcSmTreeItem::ProduceXML [▶ 152] method. The resulting xml string contains all parameters of that item, but not all of them are changeable. The xml string can contain any number and hierarchical order of xml elements that are valid in terms of the xml schema. It is allowed to change only one parameter at a time (like in the example above), change a set of parameters at once or delivers the full parameter set that ITcSmTreeItem::ProduceXML [▶ 152] returns (normally with same parameters changed).

There are some special xml elements that are not corresponding to parameters, they will "execute" a function. An example is the <Rescan> element of a PLC project tree item. The string

```
<TreeItem><PlcDef><ReScan>1</ReScan></PlcDef></TreeItem>
```

as a parameter of **ConsumeXml** will cause the System Manager to rescan the PLC project (like a manually rescan by pressing the "Rescan" button on a PLC project). The parameters and the functions that are available are documented in the xml schema file.

See also

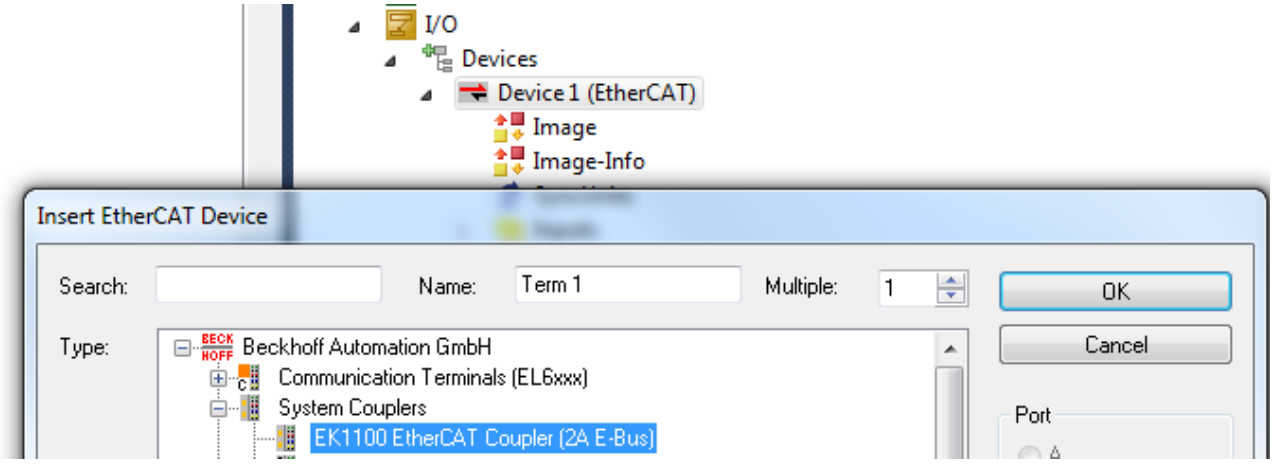
ITcSmTreeItem::ProduceXML [▶ 152]

5.3.6 ITcSmTreeItem::CreateChild

Creates a child item on a parent node. The child is being specified by its subtype [▶ 124].

```
HRESULT CreateChild(BSTRbstrName,long nSubType, BSTRbstrBefore, VARIANT vInfo,
ITcSmTreeItem**pipItem);
```

The following example demonstrates this behavior in better detail.



In this example, an EK1100 coupler is being added to an EtherCAT Master device (Device 1) in TwinCAT XAE. In Automation Interface, this could be done with the CreateChild() method. The parent node (Device 1 EtherCAT) has the item type '2' (TREEITEMTYPE_DEVICE). The sub type specifies the child item and therefore the EK1100, which is sub type '9099' (BOXTYPE_EXXXXX).

Parameters

bstrName	[in] Item name of the new child item.
nSubtype:	[in] S [▶ 124]ub type [▶ 124] of the new child item. The usable sub type depends on the item type [▶ 121] (the category)of the parent tree item. For example, a PLC Functionblock may only be added to the item type PLCFOLDER and not to a DEVICE.
bstrBefore	[in, defaultvalue("")] If set, the parameter contains the name of another child item before that the new item should be inserted.
vInfo	[in, optional] An optional parameter with additional creation information, which depends on the sub type [▶ 124]. The different dependencies are listed in the table below.
pipItem	[out, retval] Points to the location of a ITcSmTreeItem [▶ 120] interface pointer that receives the result.

Return Values

S_OK	function returns successfully.
E_POINTER	the location of the returning pointer is invalid
TSM_E_INVALIDITEMSUBTYPE (0x98510003)	the given sub type is invalid.
TSM_E_ITEMNOTFOUND (0x98510001)	The item *bstrBefore* was not found.

Optional vInfo parameter

Depending on theitem subtypeof the new child item,someadditional informationmay berequiredto create the child item. This information can be provided via the *vInfo* parameter.

Untergeordnetes Element: Element-Subtyp	vInfo-Parameter
E-Bus Box / Klemme / Modul (Element-Subtyp 9099)	Contains the Identity Object (CoE 1018h, VendorId, ProductCode and optional RevisionNo and SerialNo) of the EtherCAT box. The type of the variant must be an array of LONG (VT_I4|VT_ARRAY, 2-4 elements). Alternatively contains a BSTR of the following formats (with %X = value in hex notation, e.g. "V00000002_P044c2c52_R00000000"): Especially for Beckhoff E-Bus terminals / modules, please read the corresponding E-Bus article [▶ 125].
Interbus-Box. (Element-Subtyp 2002)	Contains the IdentCode and LengthCode of the Interbus box. The type of the variant must be an unsigned short (VT_I2), the low byte contains the IdentCode the high byte the LengthCode.
AX2000 (Element-Subtyp 5007) CANDrive (Element-Subtyp 5006)	Optionally contains a bool value (VT_BOOL) and if set an additional variable for the `Following Error`will be created.
DeviceNET (Element-Subtyp 5203) TcDNSSlave (Element-Subtyp 5250) CX1500 (Element-Subtyp 1042) FC520X Slave (Element-Subtyp 1043)	Optionally contains the file path to an EDS file (VT_BSTR).
BK3000 und alle übrigen PROFIBUS-Box-Typen	Optionally contains the file path to an GSD file (VT_BSTR).
Variable	Optionally contains the bit address of the new variable. The type of the variant can be a SHORT or a LONG (VT_I2 or VT_I4).
SPS-POU Funktionsblock (Element-Subtyp 604)	Contains IEC language type as Integer, as defined by IECLANGUAGETYPES [▶ 160].
SPS-POU Funktion (Element-Subtyp 603)	Contains a string[2] array, which hols the following values: • array[0] = IEC language type as Integer, as defined by IECLANGUAGETYPES [▶ 160]. • array[1] = return data type of function. May be any PLC data type, for example DINT, BOOL, ...

5.3.7 ITcSmTreeItem::DeleteChild

Deletes a child item.

```
HRESULT DeleteChild(BSTRbstrName);
```

Parameters

bstrName	[[in] Item name of the child item that should be deleted.

Return Values

S_OK	function returns successfully.
E_ACCESSDENIED	it is not allowed to delete the child item.
TSM_E_ITEMNOTFOUND (0x98510001)	The item *bstrBefore* was not found.

5.3.8 ITcSmTreeItem::ImportChild

Imports a child item from the clipboard or a previously exported file.

```
HRESULT ImportChild(BSTRbstrFile, BSTRbstrBefore, VARIANT_BOOLbReconnect, BSTRbstrName,
ITcSmTreeItem**pipItem);
```

Parameters

bstrFile	[in, defaultvalue(L"")] File name of the file from which the new child will be imported. If no file name specified (empty string) the child will be imported from the clipboard.
bstrBefore	[in, defaultvalue(L"")] If set, the parameter contains the name of another child item in front of which the new item should be inserted. If not set, the child will be appended at the end.
bReconnect	[in, defaultvalue(VARIANT_TRUE)] An optional flag that instructs the System Manager to try to reconnect the variables from the imported item to other variables in the configuration (by name).
bstrName	[in, defaultvalue(L"")] If set, overrides the child item name with its name in the import file.
pipItem	[out, retval] Points to the location of a ITcSmTreeItem [▶ 120] interface pointer that receives the result.

Return Values

S_OK	function returns successfully.
NTE_NOT_FOUND (0x80090011)	the file can not be found/opened.
NTE_BAD_SIGNATURE (0x80090006)	the file does not contain a valid tree item.
TSM_E_MISMATCHINGITEMS (0x98510004)	the item in the file is not a valid child item.

5.3.9 ITcSmTreeItem::ExportChild

Exports a child item to the clipboard or a file.

```
HRESULT ExportChild(BSTRbstrName, BSTRbstrFile);
```

Parameters

bstrName	[in] Name of the child being exported.
bstrFile	[in, defaultvalue("")] File name of the file to which the child will be exported. If no file name specified (empty string) the child will be exported to the clipboard.

Return Values

S_OK	function returns successfully.
TSM_E_ITEMNOTFOUND (0x98510001)	The item *bstrName* was not found.

5.3.10 ITcSmTreeItem::LookupChild

Returns a *ITcTreeItem* pointer of a descendant child tree item given by it's relative path name.

```
HRESULT LookupChild(BSTRbstrName, ITcSmTreeItem**pipItem);
```

Parameters

bstrName	[in] relative path of the tree item you are looking for. The relative path name is required and each branch must be separated by a circumflex accent '^' or a tab.
pipItem	[out, retval] points to the location of a ITcSmTreeItem [▶ 120] interface pointer on return. The interface pointer gives access to specific methods belonging to the tree item.

Return Values

S_OK	function returns successfully.
TSM_E_ITEMNOTFOUND (0x98510001)	the path name does not qualify an existing tree item.

5.3.11 ITcSmTreeItem::GetLastXmlError

Gets the Item path / Error message of the last erroneous ConsumeXml [▶ 153] call.

```
HRESULT GetLastXmlError(BSTR *pXML);
```

Parameters

pXML	[out, retval] Error message.

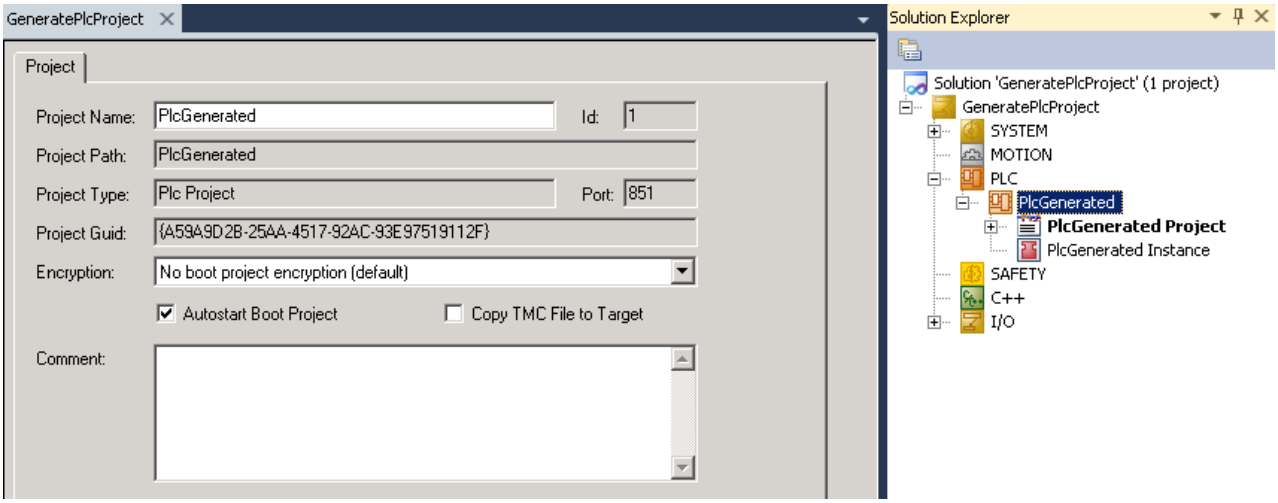
Return Values

S_OK	function returns successfully.

5.4 ITcPlcProject

5.4.1 ITcPlcProject

The class ITcPlcProject enables developers to set properties for a PLC project. It usually targets the root node of a PLC project, as shown in the picture below.



The following C# code snippet shows an example about how this class may be used in Automation Interface code:

```
ITcSmTreeItem plcProjectRootItem = systemManager.LookupTreeItem("TIPC^PlcGenerated");
ITcPlcProject iecProjectRoot = (ITcPlcProject)plcProjectRootItem;
iecProjectRoot.BootProjectAutostart = true;
iecProjectRoot.GenerateBootProject (true);
```

Please note: If you would like to**compile** a PLC project, please use the compiler functionalities of the Visual Studio COM objectEnvDTE, as shown in many of our Samples .

Methods

ITcPlcProject methods	Description	Available since
GenerateBootProject() [▶ 159]	Equals the entry "Activate Boot project" from the TwinCAT XAE context menu	TwinCAT 3.1

Properties

ITcPlcProject properties	Get/Set	Description	Available since
BootProjectAutoStart	Yes / Yes	Equals the checkbox "Autostart Boot Project" in the dialog shown above	TwinCAT 3.1
BootProjectEncryption	Yes / Yes	Equals the dropdown box "Encryption" in the dialog shown above	TwinCAT 3.1
TmcFileCopy	Yes / Yes	Equals the checkbox "Copy TMC File to Target" in the dialog shown above	TwinCAT 3.1

Version information

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.4.2 ITcPlcProject::GenerateBootProject

Activates or deactivates the PLC project as a boot project, depending on the bool parameter you specify.

```
HRESULT GenerateBootProject(VARIANT_BOOL bActivate);
```

Parameters

bActivate	[in, optional, defaultvalue(-1)] Specifies if the boot project should be activated

5.5 ITcPlcPou

5.5.1 ITcPlcPou

To handle POUs and their content within a TwinCAT 3 project, the interfaces ITcPlcPou, ITcPlcDeclaration [▶ 160] and ITcPlcImplementation [▶ 161] may be used. For example, if you would like to create a new function block for a PLC project and fill it with code, you can use these interfaces to do so. For more information, please refer to our best practise article about "How to access and create PLC-Objects".

Properties

ITcPlcPou properties	Get / Set	Description	Available since
DocumentXml	Yes / Yes	Gets/Sets PLC code of a POU in XML format (**not** PLCopen XML)	TwinCAT 3.1
ReturnType	Yes / No	Return type of the POU, for example the return type of a function. May be any data type known to the PLC, for example BOOL, DINT, ... The return type is being set when creating the POU via the CreateChild() [▶ 154] method.	TwinCAT 3.1

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

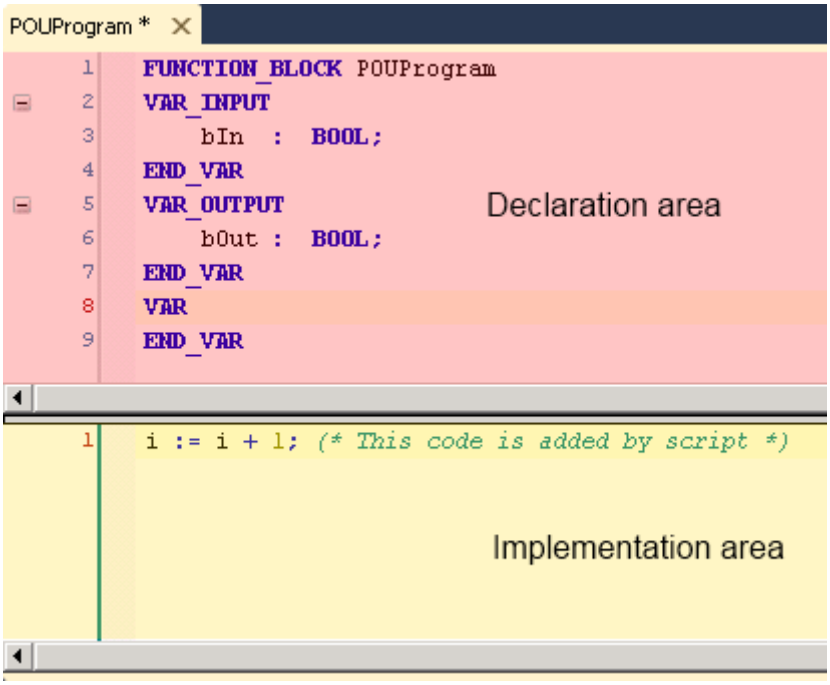
5.5.2 IECLanguageTypes

The enumeration IECLanguageTypes defines different programming languages according to IEC standard and may be used when creating a new POU via the method ITcSmTreeItem [▶ 120]::CreateChild() [▶ 154].

Entry	Value	Description
IECLANGUAGE_NONE	0	---
IECLANGUAGE_ST	1	Structured Text
IECLANGUAGE_IL	2	Instruction List
IECLANGUAGE_SFC	3	Sequential Function Chart
IECLANGUAGE_FBD	4	Function Block Diagram
IECLANGUAGE_CFC	5	Continous Function Chart
IECLANGUAGE_LD	6	Ladder Diagram

5.6 ITcPlcDeclaration

The interface ITcPlcDeclaration provides access to the declaration area of PLC POUs and their sub-nodes (like Actions, Methods, ...). Please also see the best practice article "Handling PLC-Objects" for more information about how to use this interface.



Properties (in Vtable- Order)

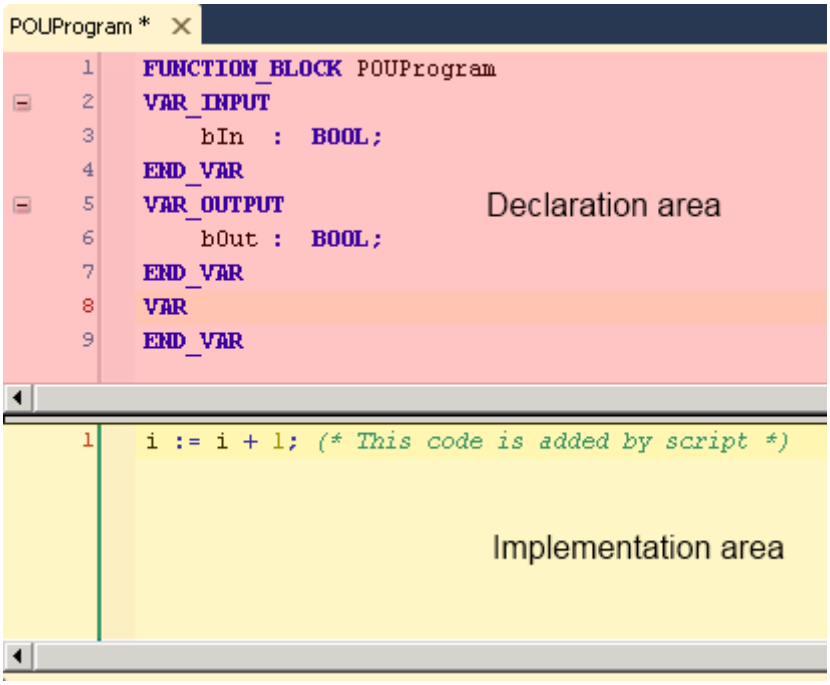
ITcPlcDeclaration properties	Get / Set	Description	Available since
DeclarationText	Yes / Yes	Represents the declaration area of the item and gets/sets its content in cleartext	TwinCAT 3.1

Version information

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.7 ITcPlcImplementation

The interface ITcPlcImplementation provides access to the implementation area of PLC POUs and their sub-nodes (like Actions, Methods, ...). Please also see the best practice article about "Handling PLC-Objects" for more information about how to use this interface.



Properties (in VTable Order

ITcPlcImplementation properties	Get / Set	Description	Available since
ImplementationText	Yes / Yes	Represents the declaration area of the item and gets/sets its content in cleartext	TwinCAT 3.1
ImplementationXml	Yes / Yes	Gets/Sets the content in XML format (**not** PLCopen XML)	TwinCAT 3.1
Language	Yes / No	Gets the IEC language used in the implementation area	TwinCAT 3.1

Version information

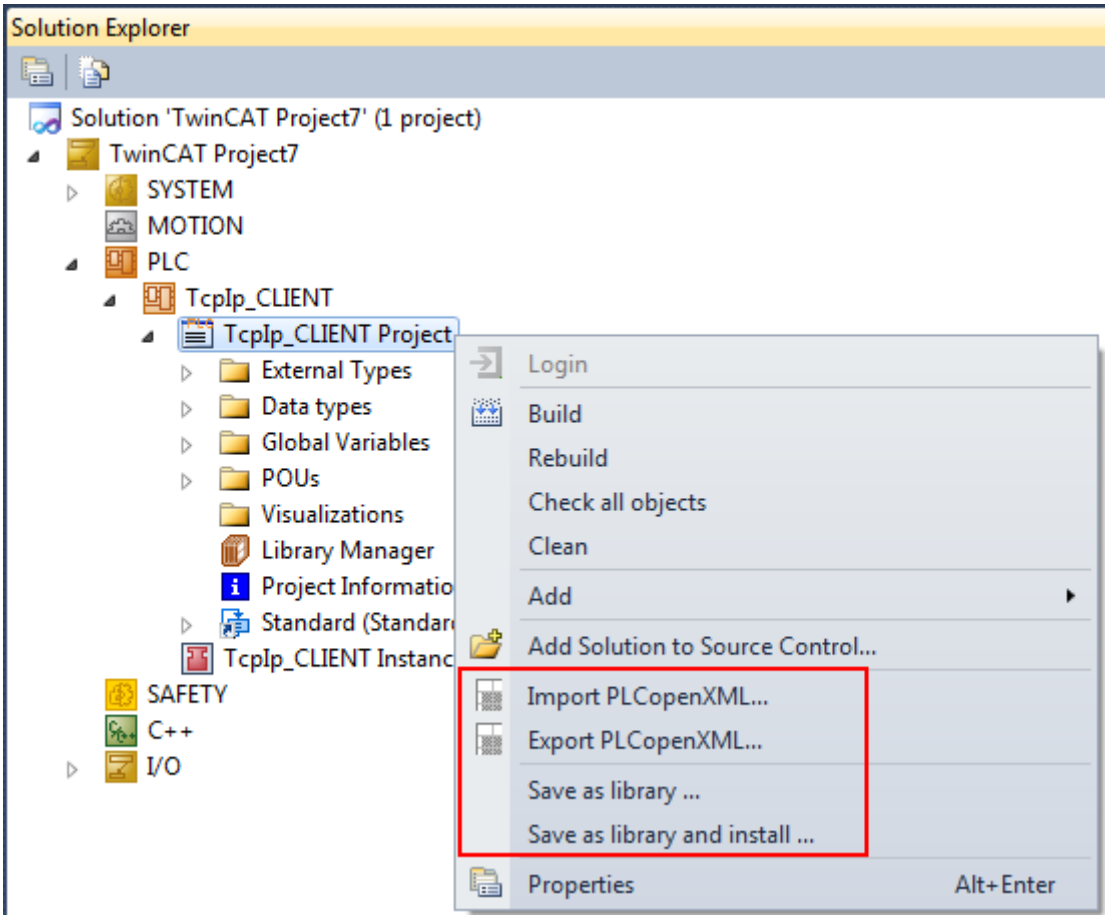
Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.8 ITcPlcIECProject

5.8.1 ITcPlcIECProject

The interface ITcPlcIECProject provides methods for importing and exporting PLC projects according to the PlcOpen XML standard or saving PLC projects as a PLC library. Compared to the TwinCAT XAE GUI, this interface represents the following four options:



Methods (in VTable Order)

ITcPlcIECProject methods	Description	Available since
PlcOpenExport() [▶ 164]	Exports the specified tree nodes and their content to a PlcOpen conform XML file	TwinCAT 3.1
PlcOpenImport() [▶ 164]	Imports a PlcOpen conform XML file and its content	TwinCAT 3.1
SaveAsLibrary() [▶ 165]	Saves the selected PLC project as a PLC library and optionally also installs it.	TwinCAT 3.1

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.8.2 PlcImportOptions

This enum contains the following options and is being used when importing a PlcOpen conform XML file via method ITcPlcIECProject [▶ 162]::PlcOpenImport() [▶ 164].

Element	Description	Supported By
PLCIMPORTOPTIONS_NONE	---	TwinCAT 3.1
PLCIMPORTOPTIONS_RENAME	In case the imported item already exists in the PLC project, it is automatically renamed	TwinCAT 3.1
PLCIMPORTOPTIONS_REPLACE	In case the imported item already exists in the PLC project, the existing project is being replaced	TwinCAT 3.1
PLCIMPORTOPTIONS_SKIP	In case the imported item already exists in the PLC project, skip it	TwinCAT 3.1

5.8.3 ITcPlcIECProject::PlcOpenExport

Exports the specified tree nodes and their content to a PlcOpen conform XML file. A path to the XML file and the tree nodes will be specified in a parameter of this method.

```
HRESULT PlcOpenExport(BSTR bstrFile, BSTR bstrSelection);
```

Parameters

bStrFile

[in] : Path to the XML file

bstrSelection

[in] : Selection of tree items which should be exported. The items are separated by a semicolon, e.g. z.B. "POUs.FBItem1;POUs.FBItem2"

Return Values

S_OK	Items successfully exported into PlcOpen XML file.

5.8.4 ITcPlcIECProject::PlcOpenImport

Imports a PlcOpen conform XML file and its content. A path to the XML file will be passed to the method via a parameter.

```
HRESULT PlcOpenImport(BSTR bstrFile, int options, BSTR bstrSelection, VARIANT_BOOL
bFolderStructure);
```

Parameters

bstrFile

[in] : Path to the XML file.

options

[in, optional, defaultvalue(0)] : Options for import, according to enum PLCIMPORTOPTIONS [▶ 163].

bstrSelection

[in, optional, defaultvalue("")] : Selects which items from the XML structure should be imported

bFolderStructure

[in, optional, defaultvalue(-1)] : If set to true (-1), the folder structure of the imported objects remains the same, if this information is available in the XML file

Return Values

S_OK	PlcOpen XML file successfully imported.

5.8.5 ITcPlcIECProject::SaveAsLibrary

Saves the PLC project as a PLC library and optionally installs it.

```
HRESULT SaveAsLibrary(BSTR bstrLibraryPath, VARIANT_BOOL binstall);
```

Parameters

bStrLibraryPath

[in] : Path to the location where the PLC library should be saved to

binstall

[in] : set to "true" if PLC library should also be installed

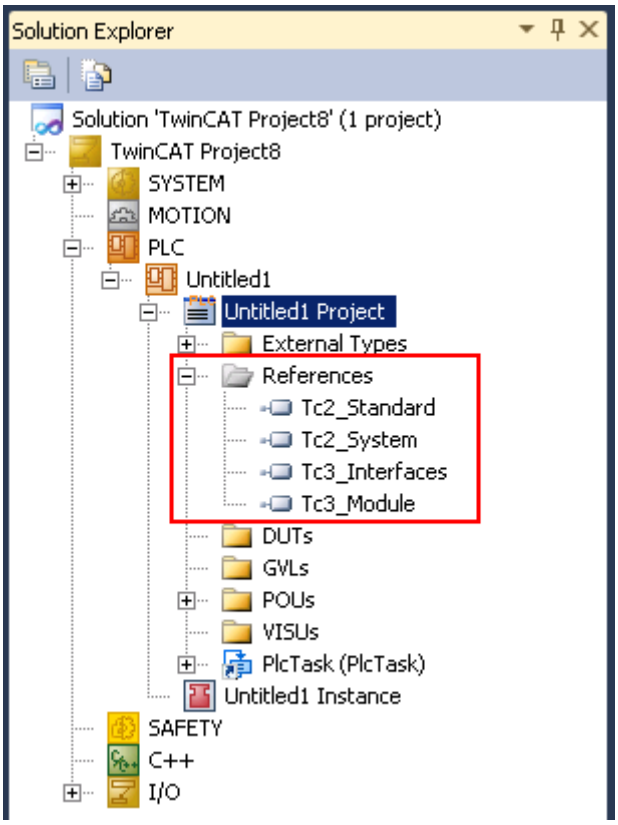
Return Values

S_OK	PLC project successfully saved as a PLC library.

5.9 ITcPlcLibraryManager

5.9.1 ITcPlcLibraryManager

The ITcPlcLibraryManager interface extends the Automation Interface by enabling access to PLC libraries of a TwinCAT 3 PLC project or PLC repositories of a TwinCAT system.



Methods (in VTable Order)

ITcPlcLibraryManager methods	Description	Available since
AddLibrary() [▶ 167]	Adds a library to a PLC project.	TwinCAT 3.1
AddPlaceholder() [▶ 167]	Adds a placeholder to a PLC project	TwinCAT 3.1
InsertRepository() [▶ 168]	Creates a repository, which represents a logical container for several libraries.	TwinCAT 3.1
InstallLibrary() [▶ 168]	Installs a library into a repository.	TwinCAT 3.1
MoveRepository() [▶ 168]	Changes the position of the repository in the repository location list.	TwinCAT 3.1
RemoveReference() [▶ 168]	Removes library from the PLC project.	TwinCAT 3.1
RemoveRepository() [▶ 169]	Removes a repository.	TwinCAT 3.1
ScanLibraries() [▶ 169]	Returns a list of **all** libraries found in the system. The returned object is of type ITcPlcLibraries [▶ 172], which is a collection of ITcPlcLibrary [▶ 171] objects.	TwinCAT 3.1
SetEffectiveResolution() [▶ 169]	Sets the Effective Resolution of a placeholder	TwinCAT 3.1
UninstallLibrary() [▶ 170]	Uninstalls a library from a repository.	TwinCAT 3.1

Properties (in VTable Order)

ITcPlcLibraryManager properties	Get / Set	Description	Available since
References	Yes / No	Gets an object of type ITcPlcReferences [▶ 171], which is a collection of ITcPlcLibrary [▶ 171] or ITcPlcPlaceholderRef objects. Represents a list of all references added to the PLC project.	TwinCAT 3.1
Repositories	Yes / No	Gets an object of type ITcPlcLibRepositories [▶ 173], which is a collection of ITcPlcLibRepository [▶ 173] objects. Represents a list of all currently configured repositories.	TwinCAT 3.1

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.9.2 ITcPlcLibraryManager::AddLibrary

Adds a library to the PLC project. A library can either be added by its attributes (Name, Version, Company) or by its display text.

```
HRESULT AddLibrary(
BSTR bstrLibName,
BSTR bstrVersion,
BSTR bstrCompany
);
```

```
HRESULT AddLibrary(BSTR bstrLibName, BSTR bstrVersion, BSTR bstrCompany);
```

Parameters

bstrLibName	[in] Library name.
bstrVersion	[in, optional, defaultvalue("")] Library version.
bstrCompany	[in, optional, defaultvalue("")] Company which created the library.

Return Values

S_OK	Library successfully added..

Comments

Two common ways to add a PLC library are:

- libraryManager.AddLibrary("Tc2_MDP", "3.2.0.0", "Beckhoff Automation GmbH"); // Adding library via its attributes
- libraryManager.AddLibrary("Tc2_MDP, 3.2.0.0 (Beckhoff Automation GmbH)"); // Adding library via its display name

Where librayManager is an object of type ITcPlcLibraryManager.

5.9.3 ITcPlcLibraryManager::AddPlaceholder

Adds a placeholder to the PLC project. A placeholder can either be added by its attributes (Placeholder name, library name, library version, library distributor) or by its name if the placeholder already exists.

```
HRESULT AddPlaceholder(BSTR bstrPlaceholderName, BSTR bstrDefaultLibName, BSTR bstrDefaultVersion,
BSTR bstrDefaultDistributor);
```

Parameters

bstrPlaceholderName	[in] Placeholder name.
bstrDefaultLibName	[in] Default library name which the placeholder points to.
bstrVersion	[in, optional, defaultvalue("")] Default library version.
bstrCompany	[in, optional, defaultvalue("")] Company which created the library.

Return Values

S_OK	Placeholder successfully added..

5.9.4 ITcPlcLibraryManager::InsertRepository

Adds a new library repository at the specified position. The position represents the index at which the repositories is located in the repository list in TwinCAT 3. In addition to the index, a repository is identified via its name and path to a directory in the file system.

```
HRESULT InsertRepository(BSTR bstrName, BSTR rootFolder, int iIndex);
```

Parameters

bstrName	[in] Repository name.
rootFolder	[in] Path to repository (file system).
iIndex	[in] Indicates on which position the repository is located in the list of repositories.

Return Values

S_OK	Repository successfully inserted

5.9.5 ITcPlcLibraryManager::InstallLibrary

Installs a library into an existing library repository.

```
HRESULT InstallLibrary(BSTR bstrRepositoryName, BSTR bstrLibPath, VARIANT_BOOL bOverwrite);
```

Parameters

bstrRepositoryName	[in] : Name of repository, where the library should be inserted
bstrLibPath	[in] : Path to the library
bOverwrite	[in] : If another library already exists, set this parameter to overwrite it

Return Values

S_OK	Library installation successful.

5.9.6 ITcPlcLibraryManager::MoveRepository

Moves the repository to a new position in the list of repositories. The position is marked by its index, which starts at 0.

```
HRESULT MoveRepository(BSTR bstrRepositoryName, int iNewIndex);
```

Parameters

bstrRepositoryName	[in] : Name of the repository
iNewIndex	[in] : Index of the repository

5.9.7 ITcPlcLibraryManager::RemoveReference

Removes a reference from the actual PLC project. A reference can either be a library or a placeholder.

Please note: In case of a library, this only removes the reference, not the actual library file from the repository. For this, the method UninstallLibrary [▶ 170]() needs to be used.

Similar to the method AddLibrary [▶ 167](), a library can be removed either by specifying its attributes (Name, Version, Company) or display text.

```
HRESULT RemoveReference(BSTR bstrLibName, BSTR bstrVersion, BSTR bstrCompany);
```

Parameters

bstrLibName	[in] : Name of library
bstrVersion	[in, optional, defaultvalue("")] :
bstrCompany	[in, optional, defaultvalue("")]

Comments

Two common ways to remove a PLC library are:

- libraryManager.RemoveReference("Tc2_MDP", "3.2.0.0", "Beckhoff Automation GmbH"); // Removing library via its attributes
- libraryManager.RemoveReference("Tc2_MDP, 3.2.0.0 (Beckhoff Automation GmbH)"); // Removing library via its display name

Where librayManager is an object of type ITcPlcLibraryManager.

5.9.8 ITcPlcLibraryManager::RemoveRepository

Removes a library repository. A repository is specified by its unique name.

```
HRESULT RemoveRepository(BSTR bstrName);
```

Parameters

bstrName

[in] Name of repository.

5.9.9 ITcPlcLibraryManager::ScanLibraries

Returns a collection of all registered libraries in all repositories.

```
HRESULT ScanLibraries(ITcPlcLibraries** ppEnumLibraries);
```

Parameters

ppEnumLibraries	[out, retval] Returns object of type ITcPlcLibraries [▶ 172], which is a collection of ITcPlcLibrary [▶ 171] objects.

Comments

This method provides the same functionality as importing the following XML structure via ConsumeXml() [▶ 153] on the tree node "Library Manager":

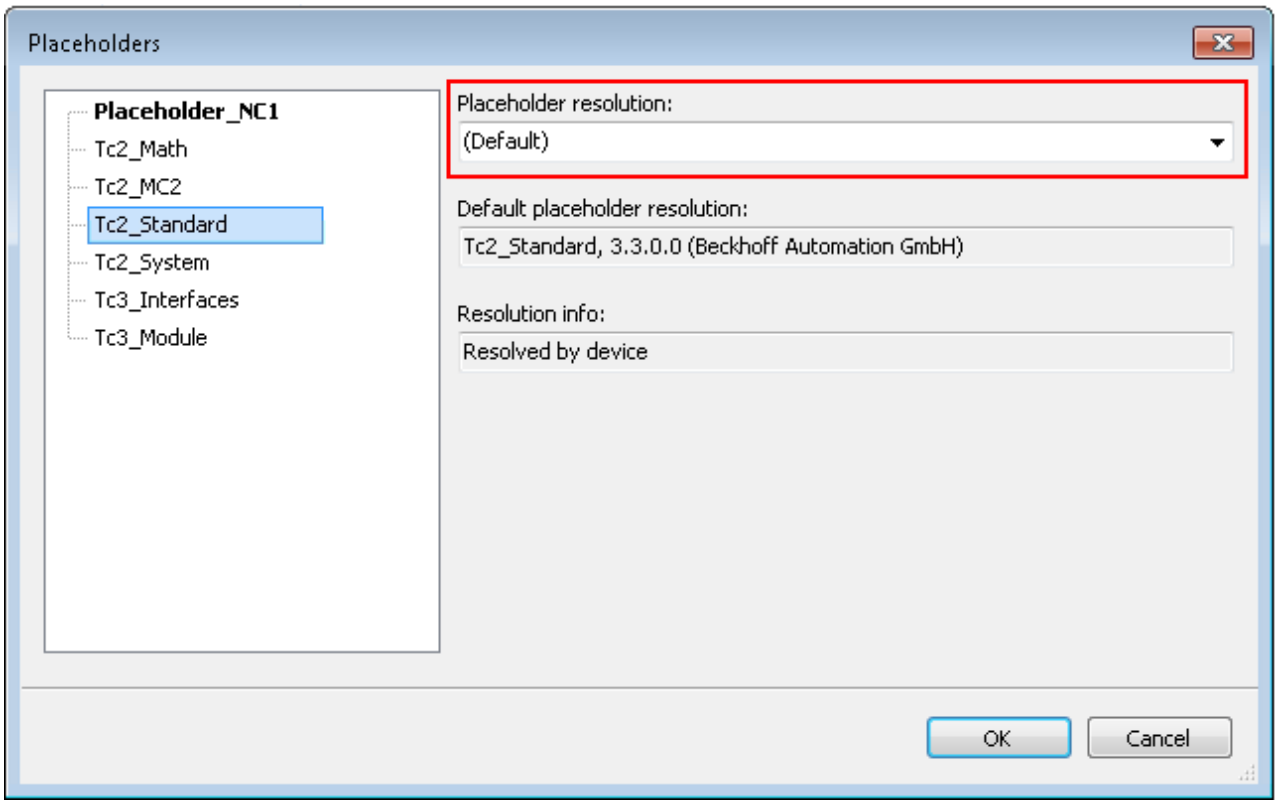
```
<TreeItem>
<PlcLibDef>
<ScanLibraries>
<Active>true</Active>
</ScanLibraries>
</PlcLibDef>
</TreeItem>
```

5.9.10 ITcPlcLibraryManager::SetEffectiveResolution

Sets the Effective Resolution, meaning the effective library, of a placeholder.

```
HRESULT SetEffectiveResolution(BSTR bstrPlaceholderName, BSTR strLibName, BSTR bstrVersion, BSTR
bstrDistributor);
```

In TwinCAT XAE, the Effective Resolution can be set via the Placeholder configuration window.



Please note: The Default Resolution of a placeholder is set when the placeholder is added via ITcPlcLibraryManager [▶ 165]::AddPlaceholder() [▶ 167].

Parameters

bstrPlaceholderName	[in] Placeholder name for which the Effective Resolution should be set.
bstrLibName	[in] Library name for Effective Resolution.
bstrVersion	[in, optional, defaultvalue("")] Library version.
bstrDistributor	[in, optional, defaultvalue("")] Company which created the library.

Return Values

S_OK	Effective Resolution successfully set.

5.9.11 ITcPlcLibraryManager::UninstallLibrary

Uninstalls a library from a repository.

```
HRESULT UninstallLibrary(BSTR bstrRepositoryName, BSTR bstrLibraryName, BSTR bstrVersion, BSTR
bstrDistributor);
```

Parameters

bstrRepositoryName	[in] : Name of repository
bstrLibraryName	[in] : Name of library
bstrVersion	[in, optional] : Version of library
bstrDistributor	[in, optional] : Distributor of library

5.10 ITcPlcReferences

ITcPlcReferences represents a collection of ITcPlcLibRef [▶ 172] objects, which is returned for example when using the property ITcPlcLibraryManager [▶ 165]::References.

Methods (in VTable Order)

Requirements

ITcPlcLibRepositories methods	Description	Available since
get_Item() [▶ 174]	Returns an item of type ITcPlcLibRef [▶ 172] which is located on a specified position.	TwinCAT 3.1

Properties (in VTable Order)

ITcPlcLibRepositories properties	Get / Set	Description	Available since
Count	Yes / No	Returns the amount of ITcPlcLibRef [▶ 172] objects in the collection	TwinCAT 3.1

Version information

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.11 ITcPlcLibrary

Represents a single PLC library when used with collection ITcPlcLibraries [▶ 172] and method ITcPlcLibraryManager [▶ 165]::ScanLibraries() [▶ 169] or property ITcPlcLibraryManager [▶ 165]::References.

Properties (in VTable Order)

ITcPlcLibrary properties	Get / Set	Description	Available since
DisplayName	Yes / No	Display name to identify library in library list	TwinCAT 3.1
Distributor	Yes / No	Library creator	TwinCAT 3.1
Name	Yes / No	Library name	TwinCAT 3.1
Version	Yes / No	Library version	TwinCAT 3.1

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.12 ITcPlcLibraries

5.12.1 ITcPlcLibraries

ITcPlcLibraries represents a collection of ITcPlcLibrary [▶ 171] objects, for example when using method ITcPlcLibraryManager [▶ 165]::ScanLibraries() [▶ 169] or property ITcPlcLibraryManager [▶ 165]::Libraries.

Methods (in VTable Order)

Requirements

ITcPlcLibraries methods	Description	Available since
get_Item() [▶ 172]	Returns item of type ITcPlcLibrary [▶ 171] which is located on a specified position.	TwinCAT 3.1

Properties (in VTable Order)

ITcPlcLibraries properties	Get / Set	Description	Available since
Count	Yes / No	Returns the amount of ITcPlcLibrary [▶ 171] objects in the collection	TwinCAT 3.1

Version information

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.12.2 ITcPlcLibraries::get_Item

Returns ITcPlcLibrary object on specified position.

```
HRESULT AddLibrary(long n, ITcPlcLibrary** pipType);
```

Parameters

n	[in] Position of item in list.
pipType	[out, retval] Returns object of type ITcPlcLibrary

5.13 ITcPlcLibRef

ITcPlcLibRef represents a base class for either ITcPlcLibrary [▶ 171] or ITcPlcPlaceholderRef [▶ 173] objects.

Properties (in VTable Order)

Requirements

ITcPlcLibRepositories properties	Get / Set	Description	Available since
Name	Yes / No	Returns the name of the ITcPlcLibRef object	TwinCAT 3.1

Version information

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.14 ITcPlcPlaceholderRef

Represents a single PLC placeholder when used with collection ITcPlcReferences [▶ 171] and method ITcPlcLibraryManager [▶ 165]::ScanLibraries() [▶ 169] or property ITcPlcLibraryManager [▶ 165]::References.

Properties (in VTable Order)

ITcPlcLibrary properties	Get / Set	Description	Available since
DisplayName	Yes / No	Display name to identify library in library list	TwinCAT 3.1
Distributor	Yes / No	Library creator	TwinCAT 3.1
Name	Yes / No	Library name	TwinCAT 3.1
Version	Yes / No	Library version	TwinCAT 3.1

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.15 ITcPlcLibRepository

The ITcPlcLibRepository interface represents a single repository, for example when used with collection ITcPlcLibRepositories [▶ 173] and property ITcPlcLibraryManager [▶ 165]::Repositories.

Properties

ITcPlcLibRepository properties	Get / Set	Description	Available since
Folder	Yes / No	Path to repository (file system)	TwinCAT 3.1
Name	Yes / No	Repository name.	TwinCAT 3.1

Version information

Requirements

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.16 ITcPlcLibRepositories

5.16.1 ITcPlcLibRepositories

ITcPlcLibraries represents a collection of ITcPlcLibRepository [▶ 173] objects, for example when using property ITcPlcLibraryManager [▶ 165]::Repositories.

Methods (in VTable Order)

Requirements

ITcPlcLibRepositories methods	Description	Available since
get_Item() [▶ 174]	Returns item of type ITcPlcLibRepository [▶ 173] which is located on a specified position.	TwinCAT 3.1

Properties (in VTable Order)

ITcPlcLibRepositories properties	Get / Set	Description	Available since
Count	Yes / No	Returns the amount of ITcPlcLibRepository [▶ 173] objects in the collection	TwinCAT 3.1

Version information

Required TwinCAT version
This interface is supported in TwinCAT 3.1 and above

5.16.2 ITcPlcLibRepositories::get_Item

Returns ITcPlcLibRepository object on specified position.

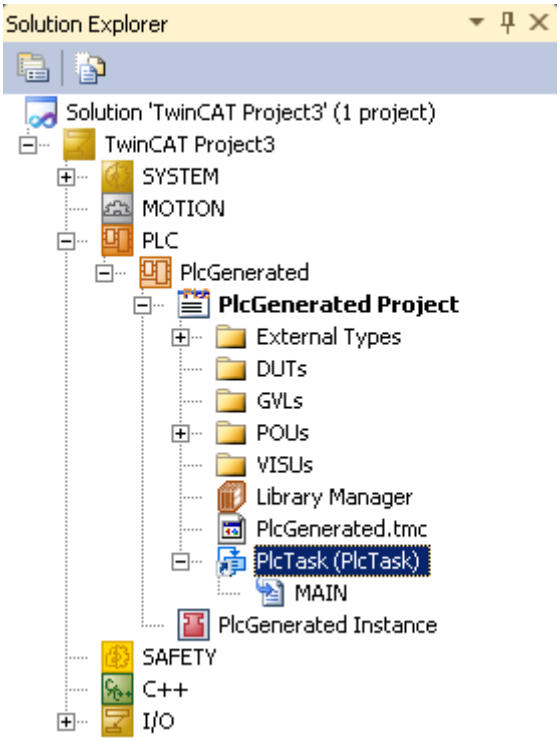
```
HRESULT AddLibrary(long n, ITcPlcLibRepository** ppRepo);
```

Parameters

n	[in] Position of item in list.
pipType	[out, retval] Returns object of type ITcPlcLibRepository

5.17 ITcPlcTaskReference

The ITcPlcTaskReference interface enables programmers to get or set the currently linked task of a PLC project. This matches the following TwinCAT XAE entry:



Properties (in VTable Order)

Requirements

ITcPlcTaskReference properties	Get / Set	Description	Available since
LinkedTask	Yes / Yes	Gets or sets the linked task of a PLC project. When setting a new linked task, the task will be specified as a string which represents the path to the task in the TwinCAT XAE tree.	TwinCAT 3.1

6 Samples

Examples can be found on Beckhoff's official GitHub account:

https://github.com/Beckhoff/TC_AI_DOTNET_Samples

7 Appendix

7.1 Miscallaneous error codes

The following error codes are the HRESULT values of the Automation Interface methods as described in the API reference [▶ 111].

```
typedefenum TCSYSMANAGERHRESULTS
{
[helpstring("ITcSmTreeItem not found!" (ITcSmTreeItem nicht gefunden!))]
TSM_E_ITEMNOTFOUND = 0x98510001,
[helpstring("Invalid Item Type!" (Ungültiger Elementtyp!))]
TSM_E_INVALIDITEMTYPE = 0x98510002,
[helpstring("Invalid SubItem Type!" (Ungültiger Unterelementtyp!))]
TSM_E_INVALIDITEMSUBTYPE = 0x98510003,
[helpstring("Mismatching Items!" (Nicht übereinstimmende Elemente!))]
TSM_E_MISMATCHINGITEMS = 0x98510004,
[helpstring("Corrupted Link" (Fehlerhafte Verknüpfung))]
TSM_E_CORRUPTEDLINK = 0x98510005,
[helpstring("Item still referenced!" (Element immer noch referenziert!))]
TSM_E_ITEMREFERENCED = 0x98510006,
[helpstring("Item already deleted!" (Element bereits gelöscht!))]
TSM_E_ITEMDELETED = 0x98510007,
[helpstring("XML Error" (XML-Fehler))]
TSM_E_XMLERROR = 0x98510008,
} TCSYSMANAGERHRESULTS;
```

Please note that the following COM error list is only a snippet and is not complete:

Failure	Description
RPC_E_CALL_REJECTED	The COM server has rejected the request. Please read our article on the implementation of a custom message filter [▶ 25] to work around this error.
Return value of the CoRegisterMessageFilter() method <> 0	A possible cause of this error may be that the COM message filter has been applied to an MTA apartment. Message filters can only be applied to STA threads. See the message filter [▶ 25] article to learn more about message filters, including STA and MTA.
Error message "A reference to Beckhoff TwinCAT XAE Base 2.0 Type Library could not be added" when referencing the type library in Microsoft Visual Studio®.	The type library is not properly registered. Please register the type library again by executing the following command: *C:\Windows\Microsoft .NET\Framework\v4.0.xxxxx\regtlibv12.exe C:\TwinCAT3\Components\Base\TCatSysManager.tlb* xxxxxx is the version of the currently installed version of .NET Framework 4.0.

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

AMD and AMD Ryzen™ are trademarks of Advanced Micro Devices, Inc.

CANopen and CANopen FD are registered trademarks of CAN in AUTOMATION - International Users and Manufacturers Group e.V.

DALI, DALI-2, D4i, DALI+, and DiiA are trademarks in various countries in the exclusive use of the Digital Illumination Interface Alliance.

DeviceNet and EtherNet/IP are trademarks of ODVA, Inc.

EnOcean® is a registered trademark of EnOcean GmbH

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

PCI Express®, PCIe®, PCI™ and PCI HOT PLUG™ are trademarks or registered trademarks and/or service marks of PCI-SIG.

Sercos is a registered trademark of Sercos International e.V.

More Information:
www.beckhoff.com/te1000/

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