BECKHOFF New Automation Technology

Manual | EN

Automation Interface

TwinCAT 2 | System

2023-01-25 | Version: 1.2

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

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

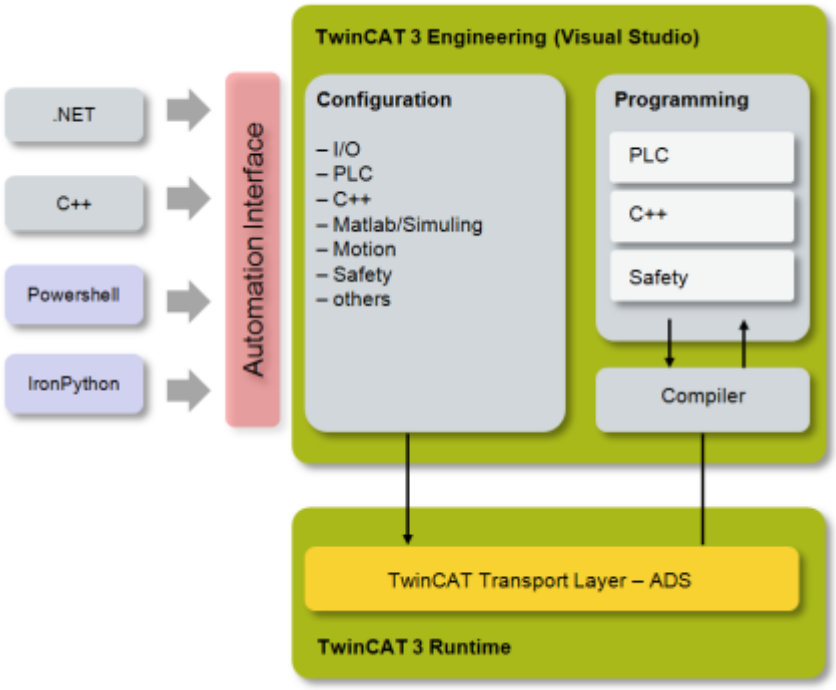
In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

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

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

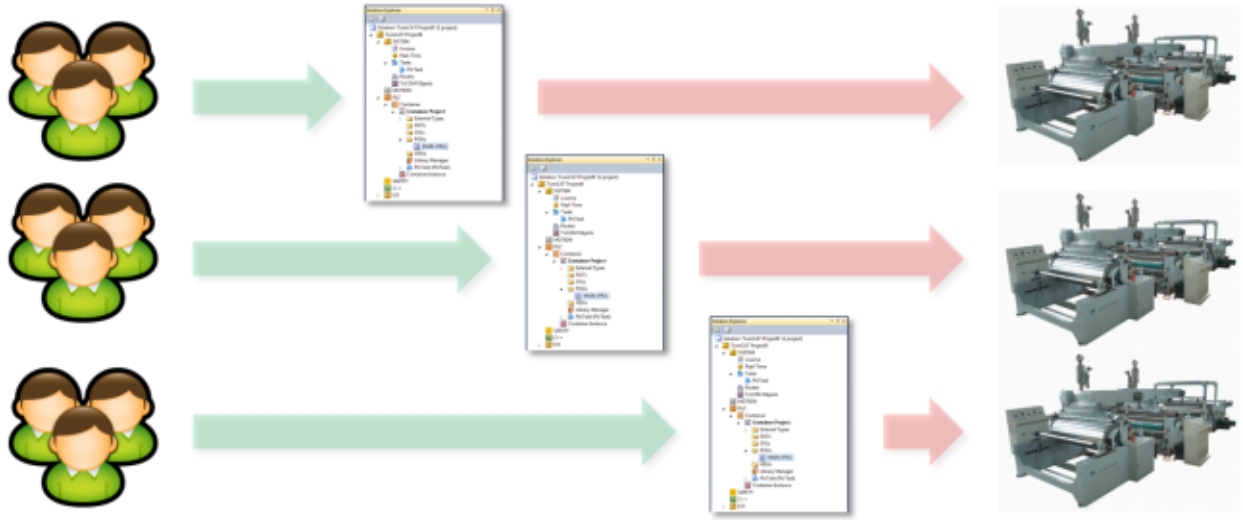
2 Overview

The TwinCAT Automation Interface enables the automatic creation and manipulation of TwinCAT XAE configurations via programming/scripting code. The automation of a TwinCAT configuration is available through so-called automation interfaces which can be accessed from all COM-capable programming languages (e.g. C++ or .NET) and also from dynamic script languages - e.g. Windows PowerShell, IronPython or even legacy VBscript.

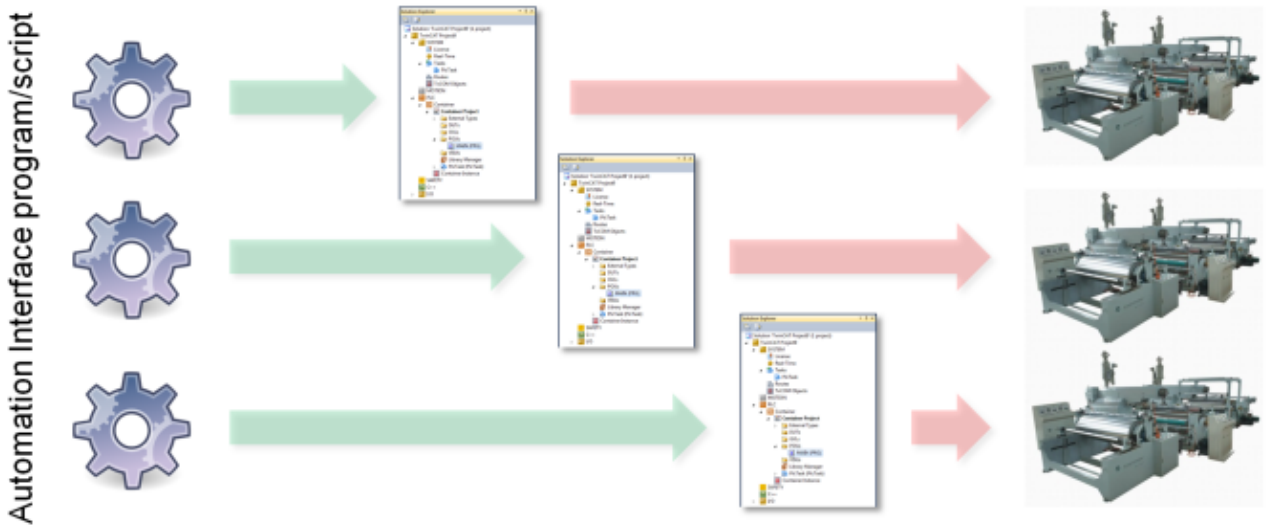


TwinCAT Automation Interface enables an efficient engineering process by giving customers the possibility to automate the configuration of a full TwinCAT project configuration.

Traditionally, a machine configuration had to be manually adapted to each new project or had even to be created from scratch, which could not only involve a tremendous amount of engineering time and therefore costs but is also error-prone because of human mistakes.



With TwinCAT Automation Interface, the process of adapting TwinCAT configurations to a new environment or even create new TwinCAT configurations from scratch can be automated according to the customer's needs.



Readers should continue with the following topics:

Basics

Topic	Description
Creating/Loading TwinCAT XAE configurations [▶ 17]	Describes how to create or open a TwinCAT configuration
Navigating TwinCAT XAE [▶ 18]	Describes how to navigate through a TwinCAT configuration
Custom tree item parameters [▶ 20]	Describes how to access custom parameters of an item. This is important to access configuration parameters of a TwinCAT tree item.

Best practice

Topic	Description
Creating and handling EtherCAT devices [▶ 23]	Describes how to create EtherCAT devices and connect them to an EtherCAT topology
From offline to online configuration [▶ 27]	Describes how add address information to an offline created configuration

Additionally, this documentation also includes a full API reference [▶ 30] of all interfaces. The How to [▶ 69] and Sample [▶ 67] 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.

2.1 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)	✓	✓	✓	✓

Feature	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Future versions
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	-	-	✓	✓
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	-	-	✓	✓

Feature	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Future versions
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				
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.

2.2 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 [▶ 67] page for more information. There is also an How-To sample [▶ 77] 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 [▶ 14] page for more information.

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

Please see our version overview [▶ 9] 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 [▶ 20] 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 [▶ 9] 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.

- **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 [▶ 71] which shows you exactly how to do that.

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

Yes. Please see our samples [▶ 67] and How-To [▶ 69] pages for more information.

- **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 building your Automation Interface code based on english terminology.

NOTE

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

3 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. 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.

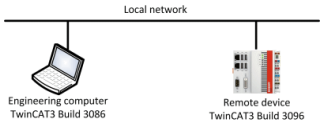


Fig. 1: Valid scenario: Remote device > Engineering system

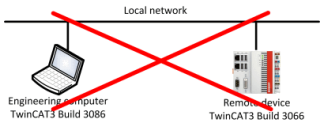


Fig. 2: Invalid scenario: Remote device < Engineering system

NOTE

Compatibility constrains

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#, VB.NET, F#

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, the following scripting languages are recommended:

- Windows PowerShell (for more detailed information, please visit MSDN Windows PowerShell)
- IronPython (for more detailed information, please visit IronPython)

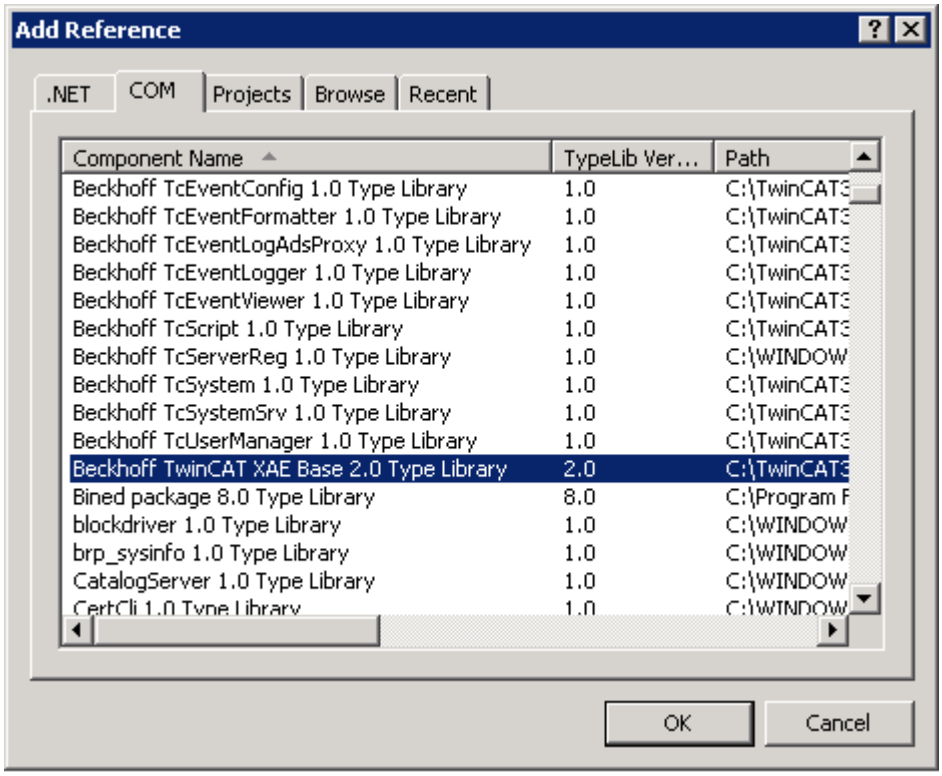
Most of the sample code in this documentation is based on one of these scripting languages.

4 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 need to add a reference to the corresponding COM object **Beckhoff TwinCAT XAE Base** (depending on your TwinCAT version, see table below) within the Visual Studio project.



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

Please proceed with the article Accessing TwinCAT configuration [▶ 17] which explains all further steps in more 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 [▶ 17] 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.

Requirements

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

5 Configuration

5.1 Basics

5.1.1 Accessing TwinCAT configuration

This chapter describes how to create and access a TwinCAT configuration project via Automation interface. The objective of this creation process is to get access to a TwinCAT System Manager configuration.

The following code snippets demonstrate how to get access to TwinCAT and create a System Manager configuration.

Sample (C#):

```
TcSysManager systemManager = new TcSysManager();
systemManager.NewConfiguration();
//...Creating configuration...
systemManager.SaveConfiguration("C:\Sample.tsm");
```

Sample (Powershell):

```
$systemManager = new-object -comobject TCatSysManager.TcSysManager
$systemManager.NewConfiguration()
//...Creating configuration...
$systemManager.SaveConfiguration("C:\Sample.tsm")
```

Sample (IronPython):

```
sysManType = System.Type.GetTypeFromProgID("TCatSysManager.TcSysManager")
systemManager = System.Activator.CreateInstance(sysManType)
systemManager.NewConfiguration()
systemManager.SaveConfiguration("C:\Sample.tsm")
```

Sample (C++):

```
#include "stdafx.h"
#include <atlbase.h>

#import "C:\TwinCAT\Io\TCatSysManager.tlb"

int _tmain(int argc, _TCHAR* argv[])
{
  HRESULT hr = CoInitialize(NULL);

  if (SUCCEEDED(hr))
  {
    TCatSysManagerLib::ITcSysManager3Ptr pTCatSysMan;
    HRESULT hr = pTCatSysMan.CreateInstance("TCatSysManager.TcSysManager.1");

    CComBSTR bstrFile(L"C:\\Untitled.tsm");

    hr = pTCatSysMan->OpenConfiguration(bstrFile.Detach());

    if(SUCCEEDED(hr))
    {
      hr = pTCatSysMan->ActivateConfiguration();

      if(SUCCEEDED(hr))
      {
        hr = pTCatSysMan->StartRestartTwinCAT();
      }
    }

    if (FAILED(hr))
    {
      CComPtr<IErrorInfo> pErrorInfo;
      hr = ::GetErrorInfo(0,&pErrorInfo);

      if (hr == S_OK)
      {
         CComBSTR pbstrError(L"");
         hr = pErrorInfo->GetDescription(&pbstrError);
         wprintf((LPWSTR)pbstrError);
```

```
        }
      }
    }

    CoUninitialize();
    return 0;
}
```

5.1.2 Browsing TwinCAT configuration

In a separate article, we have already shown you how to access TwinCAT [▶ 17] via the Visual Studio Automation Model. This reference to TwinCAT is being represented by an object of type ITcSysManager [▶ 30]. 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 ordered in a tree-like structure. In Automation Interface, every tree node, and therefore every element of a TwinCAT configuration, is represented by the interface ITcSmTreeItem [▶ 37].

Navigating the TwinCAT data model can be done in different ways, depending on the sort of information that should be retrieved.

- **Lookup-Methods** are searching for specific tree items via specified search criterias, e.g. the path to a tree item
- **Iterators** or browsing functions iterate over a set of retrieved tree items

Both methods are now being discussed in the following article.

LookupMethods

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

- ITcSysManager::LookupTreeItem [▶ 35] determines a tree item with the specified absolute path name.
- ITcSysManager3::LookupTreeItemById [▶ 36] determines a tree item with the specified item type and item Id.
- ITcSmTreeItem::LookupChild [▶ 65] determines a tree item within a subtree specified 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 [▶ 35].

Iterators

Now, 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 [▶ 37]::_NewEnum may be used. _NewEnum iterates over all subnodes of the currently referenced ITcSmTreeItem [▶ 37]. 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
// Usage of automatically generated Auto-Pointers (see MSDN documentation of #import statement)
...

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), rein
terpret_cast(&pChild));
                    childPtr.Attach(pChild);
                    _bstr_t strName = pChild->GetName();
               }
          }
     }
     while(hr != S_FALSE); // S_FALSE indicates end of collection
}
```

Sample (IronPython):

```
parent = systemManager.LookupTreeItem("TIRC");
for child in parent : print child.Name
```

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 [▶ 37] and ITcSmTreeItem:Child(n) [▶ 30] 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) [▶ 37] , ITcSmTreeItem::Var(x,n) [▶ 37] pair of properties. A selection of the variable type (input variables or output variables) can be done by parameter.

5.1.3 Custom TreeItem Parameters

The ITcSmTreeItem [▶ 37] 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 [▶ 60] and its counterpart ITcSmTreeItem::ConsumeXml [▶ 61]. 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).

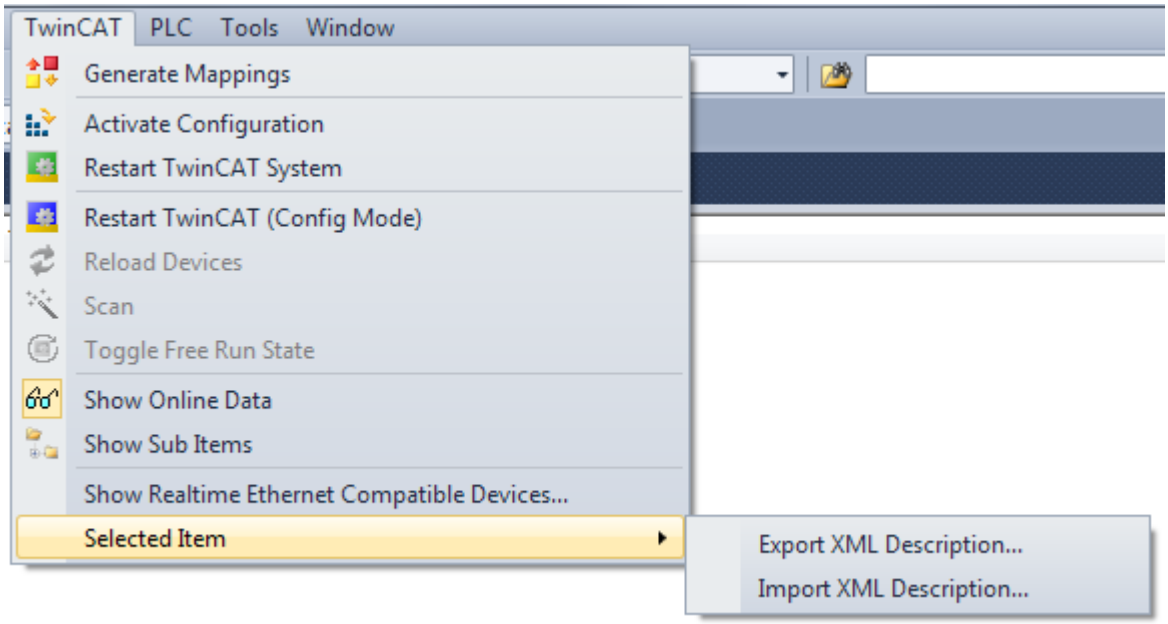


Fig. 3: TcSysMan_AutomationXmlParameters

With this Import/Export functionset, good parts of any 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.

Using this workflow ensures that all changed properties and methods are valid (the Xml stream is self-documenting) and in which way they can be used (for example as Read-Only-Property, as Read-Write-Property or as a method trigger).

Reading Properties

In this example, the Network Adapter setting of an TwinCAT configuration should be detected (here an CX9001): Here the Adapter Settings are available within the DeviceDef specific part of the XML Data.

```
<TreeItem>
    <ItemName>RTEthernet</ItemName>
    <PathName>TIID^RTEthernet</PathName>
    <ItemType>2</ItemType>
    <ItemId>3</ItemId>
    <ItemSubType>66</ItemSubType>
    <ItemSubTypeName>Real-Time Ethernet (Standard)</ItemSubTypeName>
    <ChildCount>0</ChildCount>
    <Disabled>0</Disabled>
    <CreateSymbols>0</CreateSymbols>
    <TreeImageId>3</TreeImageId>
    <DeviceDef>
        <AmsPort>28675</AmsPort>
        <AddressInfo>
            <Pnp>
                <DeviceDesc>TCIXPNPE1</DeviceDesc>
                <DeviceName>TCIXPNPE1</DeviceName>
                <DeviceData>000105015c2a</DeviceData>
            </Pnp>
        </AddressInfo>
        <ScanBoxes>0</ScanBoxes>
    </DeviceDef>
</TreeItem>
```

Fig. 4: TcSysMan_AutomationXmlParameters001

The Adapter's MAC-Address can be found at the XmlPath "TreeItem/DeviceDef/AddressInfo/Pnp/DeviceData". With this information the script code that reads this information can be written:

Code Snippet (IronPython) for TwinCAT 3:

```
import System
import clr
import System.IO;

clr.AddReference('System.Xml')

from System.Xml import XmlDocument, XmlTextReader
from System.IO import File,FileInfo, Directory, DirectoryInfo

#create Visual Studio
t = System.Type.GetTypeFromProgID("VisualStudio.DTE.10.0");
dte = System.Activator.CreateInstance(t)
solution = dte.Solution

if (Directory.Exists(r"C:\SolutionFolder")):
    Directory.Delete(r"C:\SolutionFolder",True)

Directory.CreateDirectory(r"C:\SolutionFolder")
Directory.CreateDirectory(r"C:\SolutionFolder\MySolution1");

solution.Create(r"C:\SolutionFolder","MySolution1")
solution.SaveAs(r"C:\SolutionFolder\MySolution1\MySolution1.sln")

template = r"C:\TwinCAT3\Components\Base\PrjTemplate\TwinCAT Project.tsp"
project = solution.AddFromTemplate(template,"C:\SolutionFolder\MySolution1","MyProject")

#Get the Specific System Manager Interface
sysMan = project.Object

sysMan = project.Object
ioDevices = sysMan.LookupTreeItem("TIID")
treeItem = ioDevices.CreateChild("RTEthernet",66,"",None) #Adding the RT-Ethernet Device
xml = treeItem.ProduceXml()

xmlDoc = XmlDocument()
xmlDoc.LoadXml(xml)
```

```
test = xmlDoc.SelectSingleNode("TreeItem/DeviceDef/AddressInfo/Pnp/DeviceData")
deviceDataString = test.InnerText;
print deviceDataString # Print the MacID of the Device (contains 000105015c2a here)
```

Writing Properties

In this example, the Adapter settings should also be written to the configuration. Shortening the original XML to the following form and changing the Adapter parameters, will activate the new parameter set within the configuration (as long the parameters are correct). To test the settings before writing the script code, the "Import Xml Descriptions" Menu function is very helpful.

```
<TreeItem>
    <DeviceDef>
        <AddressInfo>
            <Pnp>
                <DeviceDesc>TCIXPNPE2</DeviceDesc>
                <DeviceName>TCIXPNPE2</DeviceName>
                <DeviceData>00010501958f</DeviceData>
            </Pnp>
        </AddressInfo>
    </DeviceDef>
</TreeItem>
```

After the parameter change has been tested from within TwinCAT XAE, it is easy to write the script code that creates the valid parameterset and feeds the ITcTreeItem [▶ 37] via the method ITcTreeItem::ConsumeXml(xml) [▶ 61], for example:

```
xml = '<TreeItem> \
        <DeviceDef>\
            <AddressInfo>\
                <Pnp>\
                    <DeviceDesc>TCIXPNPE2</DeviceDesc>\
                    <DeviceName>TCIXPNPE2</DeviceName>\
                    <DeviceData>00010501958f</DeviceData>\
                </Pnp>\
            </AddressInfo>\
        </DeviceDef>\
    </TreeItem>'
device = sysMan.LookupTreeItem("TIID^RTEthernet")
device.ConsumeXml(xml)
```

Calling Methods

In some cases, specific methods should be triggered, for example the ScanBoxes functionality from TwinCAT XAE, which scans the device for all connected boxes. Looking at the XML-example from the above chapter Reading-Properties, the XML-Content contains a flag <ScanBoxes>0</ScanBoxes>. This sort of XmlElement is called a "Method Trigger" and should indicate that the "ScanBoxes" functionality is currently not triggered. Consuming the following XML-Code at the device item will start a "ScanBoxes" and insert all found Boxes / Terminals into the configuration.

Xml Data:

```
<TreeItem>
    <DeviceDef>
        <ScanBoxes>1</ScanBoxes>
    </DeviceDef>
</TreeItem>
```

Code Snippet (IronPython):

```
xml = '<TreeItem> \
        <DeviceDef>\
            <ScanBoxes>1</ScanBoxes>\
        </DeviceDef>\
```

```
       </TreeItem>'
device = sysMan.LookupTreeItem("TIID^RTEthernet")
device.ConsumeXml(xml)
```

5.2 Best practice

5.2.1 Building an EtherCAT topology

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

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

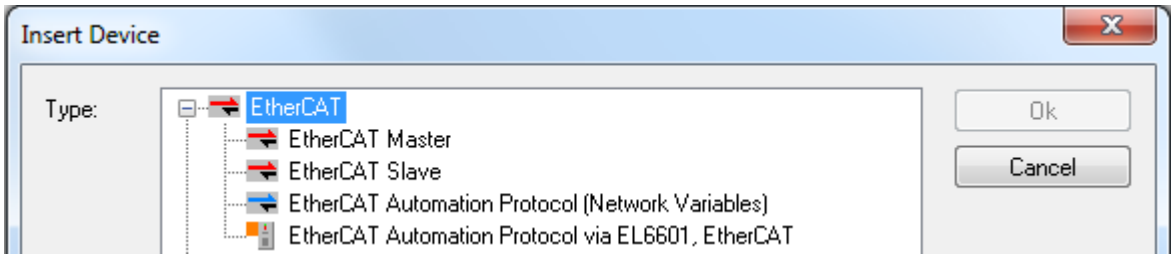
- Activate the configuration

General information

This documentation should give you an overview about 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 [▶ 43].

Creating an EtherCAT device

The first step in creating a TwinCAT EtherCAT configuration is to create an EtherCAT device, which may involve either to create an EtherCAT Master, Slave or Automation Protocol (e.g. to use network variables, as covered in a separate article). To create an EtherCAT Master/Slave, simply make use of the ITcSmTreeItem [▶ 62]::CreateChild() [▶ 62] 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 Slave - Code Snippet (C#)

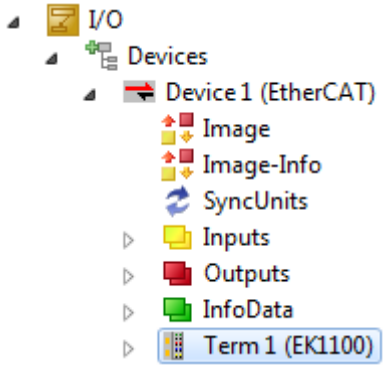
```
ITcSmTreeItem devices = systemManager.LookupTreeItem("TIID");
ethercatSlave = devices.CreateChild("EtherCAT Slave", 130, null, null);
```

Creating EtherCAT boxes

The second step involves creating EtherCAT boxes, for example an EK1100 EtherCAT Coupler. As the article about E-Bus SubTypes [▶ 43] explains, every child item (there a few exceptions - see below) of an EtherCAT Master uses a common SubType (9099) and will be identified via the Product Revision, which needs to be used in the vInfo parameter of the ITcSmTreeItem [▶ 37]::CreateChild() [▶ 62] method.

Code Snippet (C#)

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



In addition to the full product revision, you can also use a "wildcard". If you only specify "EK1100" as vInfo, Automation Interface will automatically detect and use the newest revision number.

Example:

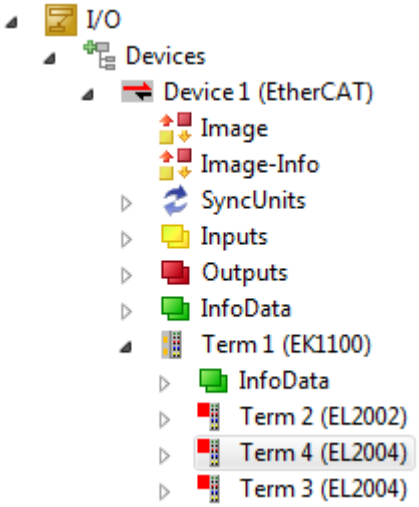
```
ITcSmTreeItem ethercatMaster = systemManager.LookupTreeItem("TIID^EtherCAT Master");
ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100");
```

Creating EtherCAT terminals and insert into topology

The creation of EtherCAT terminals is based on the same concepts as EtherCAT boxes. Every terminal also shares a common SubType and will be identified via the Product Revision, which needs to be used in the vInfo parameter of the ITcSmTreeItem [▶ 37]::CreateChild() [▶ 62] method. The parameter bstrBefore lets you choose the position on which the terminal should be 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");
```



Should you have trouble using the bstrBefore parameter, please try to insert the terminal on a "device-level". See the following example.

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

The new terminal will then be inserted before "Term 3 (EL2004)" under the last inserted EtherCAT box.

In addition to the full product revision, you can also use a "wildcard". If you only specify "EL2004" as vInfo, Automation Interface will automatically detect and use the newest revision number. See the following example.

```
ITcSmTreeItem 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	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)

Changing the "Previous Port" of an EtherCAT terminal

The Previous Port of an EtherCAT terminal determines the position of the terminal inside the EtherCAT topology.

In TwinCAT XAE, the DropDown box automatically includes all available previous ports. To configure this setting via Automation Interface, you can make use of the ITcSmTreeItem [▶ 37]::ProduceXml() [▶ 60] and ITcSmTreeItem [▶ 37]::ConsumeXml() [▶ 61] methods to modify the XML description [▶ 20] of the corresponding EtherCAT terminal. This XML description therefore includes one or more XML nodes <PreviousPort>, where its attribute "Selected=1" determines, which previous port has been currently selected. Each previous port device is identified by its physical address.

Example (XML description)

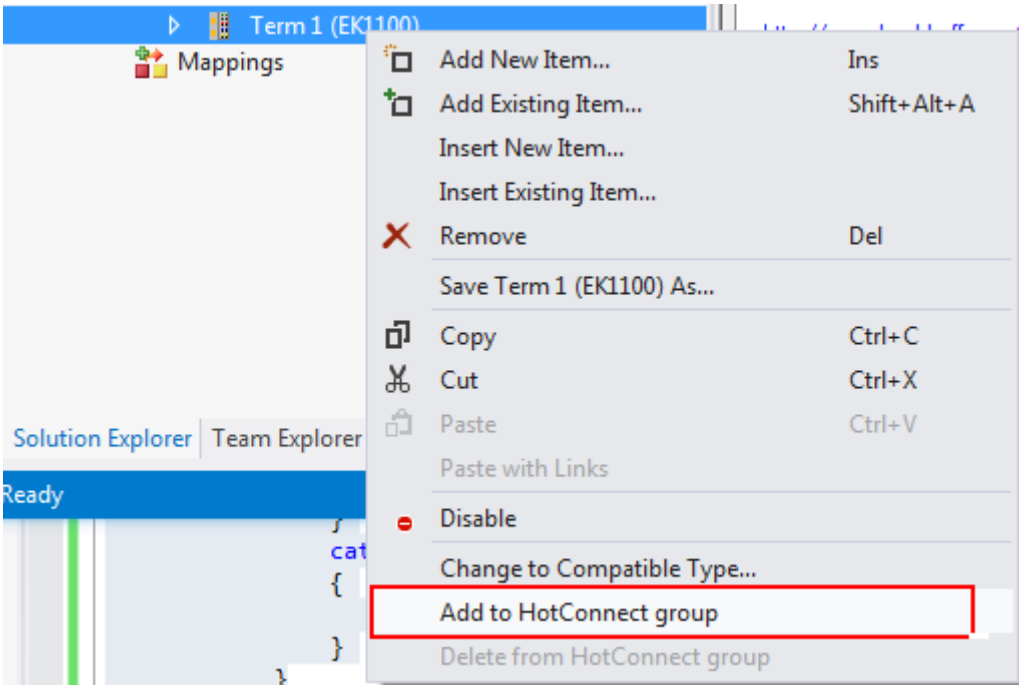
```
<TreeItem>
  ...
  <EtherCAT>
        ...
       <Slave>
           ...
            <PreviousPort Selected="1">
                 <Port>B</Port>
                 <PhysAddr>1045</PhysAddr>
            </PreviousPort>
            <PreviousPort>
                 <Port>B</Port>
                 <PhysAddr>1023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

If you want to change the Previous port you need to know the <PhysAddr> of the desired device, which can also be determined via its XML description.

Adding EtherCAT slaves to a HotConnect group

EtherCAT HotConnect allows pre-configured sections to be removed from or added to the data traffic before the start or during operation of the system. For more information about EtherCAT HotConnect please read our EtherCAT System documentation.

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



In TwinCAT Automation Interface, the same can be achieved by consuming 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>read hot connect identity</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>
```

The <GroupMemberCnt> tag needs to specify the exact number of terminals plus the device itself.

Activate the EtherCAT configuration

To activate a created TwinCAT configuration via Automation Interface, the ITcSysManager [▶ 30]::ActivateConfiguration() [▶ 33] 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 [▶ 37]::ProduceXml() [▶ 60]) and then import the address information via ITcSmTreeItem [▶ 37]::ConsumeXml() [▶ 61] into the created (offline) configuration. There are two HowTo samples which help you exactly with this kind of configuration:

- Scan Devices [▶ 77] via Automation Interface

5.2.2 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

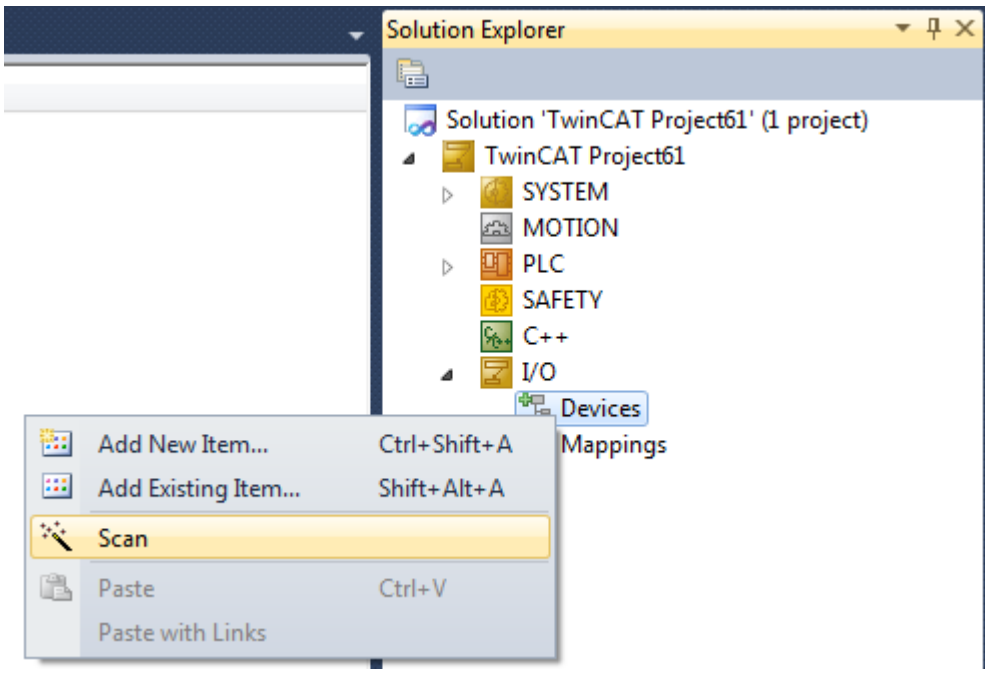
General information

When creating a TwinCAT configuration, there are two common scenarios:

- Scenario 1: this scenario is based on the real physical device, on which the TwinCAT configuration should run later. Therefore, all I/Os are **online** and already available and attached to the device.
- Scenario 2: this scenario could involve creating the configuration **offline** (that means without any I/Os attached to the engineering device) and later switch to an online configuration when the I/Os are available. This is the scenario which will be the primary focus of this article.

Both scenarios are possible via TwinCAT Automation Interface. However, as scenario 2 lacks some important information about some of the I/Os, e.g. their physical addresses, this scenario could be slightly more complex because you need to hand in the required (address) information later.

The familiar TwinCAT XAE functionality "Scan Devices" plays a significant role in this use-case because it scans for all available I/O devices and automatically attaches them to the configuration - together with all needed additional information, e.g. physical addresses.



On a physical controller, this functionality can also be called via Automation Interface and then returns an XML representation of all found I/O devices including their corresponding address information.

When a TwinCAT configuration should be activated on the controller, the required address information needs to be set for all I/O devices that are part of the configuration. This can be done by calling the "Scan Devices" functionality via Automation Interface and setting the address information for the I/O devices in the configuration. This will be explained in more detail now.

Creating an offline configuration

There are several articles which explain how to create an offline TwinCAT configuration. Please refer to our Product Description [▶ 8] 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 [▶ 77] 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 [▶ 30]::SetTargetNetId() [▶ 35] 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();
```

6 API

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 [▶ 8], 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 [▶ 30]	Base class to create and parameterize a TwinCAT configuration	TwinCAT 2.11
ITcSmTreeItem [▶ 37]	Represents a tree item within a TwinCAT configuration	TwinCAT 2.11

6.1 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 [▶ 31]	Generates a new configuration	TwinCAT 2.11
OpenConfiguration [▶ 32]	Loads a prior created configuration file (WSM file)	TwinCAT 2.11
SaveConfiguration [▶ 32]	Saves the configuration in a file with the given name or with the current name	TwinCAT 2.11
ActivateConfiguration [▶ 33]	Activates the configuration (same as "Save To Registry")	TwinCAT 2.11
LookupTreeItem [▶ 35]	Looks up to a configuration item (item in the tree) by name and returns a ITcSmTreeItem [▶ 37] interface	TwinCAT 2.11
StartRestartTwinCAT [▶ 33]	Starts or Restarts the TwinCAT System	TwinCAT 2.11
IsTwinCATStarted [▶ 33]	Evaluates if the TwinCAT System is running	TwinCAT 2.11
LinkVariables [▶ 33]	Links two variables given by names	TwinCAT 2.11
UnlinkVariables [▶ 34]	Clears the link between two variables given by names or all links from one variable.	TwinCAT 2.11

ITcSysManager2 methods	Description	Available since
SetTargetNetId	Set the target NetId of the currently opened TwinCAT configuration.	TwinCAT 2.11

ITcSysManager2 methods	Description	Available since
GetTargetNetId [▶ 34]	Gets the target NetId of the currently opened TwinCAT configuration.	TwinCAT 2.11
GetLastErrorMessages	Get the last error messages which occurred in the TwinCAT subsystem.	TwinCAT 2.11

ITcSysManager3 methods	Description	Available since
LookupTreeItemById	Looks for a configuration tree item with the specifed Item id.	TwinCAT 2.11
ProduceMappingInfo	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 [▶ 35] and ITcSysManager3::LookupTreeItemById [▶ 36]. A detailed explanation of browsing TwinCAT XAE can be found in the chapter TreeItem Browsing Models [▶ 18].

NOTE

The three methods ITcSysManager::NewConfiguration [▶ 31], ITcSysManager::OpenConfiguration [▶ 32] and ITcsSysManager::SaveConfiguration [▶ 32] are only available in Compatiblity Mode [▶ 17]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

The **ITcSysmanager** and the ITcSmTreeItem [▶ 37] 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

Required TwinCAT version
This interface is supported in TwinCAT 2.11 and above

6.1.1 ITcSysManager::NewConfiguration

The NewConfiguration() method generates a new TwinCAT configuration file.

```
HRESULT NewConfiguration();
```

Return Values

S_OK	the function returns successfully.
E_ACCESSDENIED	the current document in the system manager instance is locked. This occurs if at least one reference to the system managers object or one of the tree elements are open.
E_FAIL	the function fails.

Comments

NOTE

The three methods ITcSysManager::NewConfiguration [▶ 31], ITcSysManager::OpenConfiguration [▶ 32] and ITcsSysManager::SaveConfiguration [▶ 32] are only available in Compatiblity Mode [▶ 17]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

6.1.2 ITcSysManager::OpenConfiguration

The OpenConfiguration() methid 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 generate. The currently running configuration of a target device may also be read by using "CURRENTCONFIG".

Return Values

S_OK	function returns successfully.
E_ACCESSDENIED	the current document in the system manager instance is locked. This occurs if at least one reference to the system managers object or one of the tree elements are open.
E_INVALIDARG	the given file path does not refers to a valid configuration file.

Comments

NOTE

The three methods ITcSysManager::NewConfiguration [▶ 31], ITcSysManager::OpenConfiguration [▶ 32] and ITcsSysManager::SaveConfiguration [▶ 32] are only available in Compatiblity Mode [▶ 17]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

6.1.3 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 file path in that the configuration file that should be saved. If *bstrFile* is an empty string, the current file name will be used.

Return Values

S_OK	function returns successfully.
E_INVALIDARG	the given file path is invalid.

Comments

NOTE

The three methods ITcSysManager::NewConfiguration [▶ 31], ITcSysManager::OpenConfiguration [▶ 32] and ITcsSysManager::SaveConfiguration [▶ 32] are only available in Compatiblity Mode [▶ 17]. Calling them in standard mode will throw an E_NOTSUPPORTED Exception.

6.1.4 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 returns successfully.
E_FAIL	the function fails.

6.1.5 ITcSysManager::IsTwinCATStarted

The IsTwinCATStarted() method evaluates if the TwinCAT System is running.

```
HRESULT IsTwinCATStarted(VARIANT_BOOL*pStarted);
```

Parameters

pStarted	[out, retval] Points to the location of a boolean value that receives the result.

Return Values

S_OK	function returns successfully.

6.1.6 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 returns successfully.
E_FAIL	TwinCAT could not be started.

6.1.7 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 [▶ 35]. The same shortcuts [▶ 35] 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 path name(s) does not qualify an existing tree item.
TSM_E_INVALIDITEMTYPE (0x98510002)	one or both tree item(s) is not a variable.
TSM_E_MISMATCHINGITEMS (0x98510004)	the two variables cannot link together. Maybe 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.

6.1.8 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 [▶ 35]. The same shortcuts [▶ 35] 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.

6.1.9 ITcSysManager2::GetTargetNetId

The GetTargetNetId() method returns the NetId of the current TwinCAT system.

```
HRESULT GetTargetNetId();
```

Parameters

None

Return Values

STRING	returns target's NetId.

6.1.10 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.

6.1.11 ITcSysManager2::GetLastErrorMessages

The GetLastErrorMessages() metod returns the last error messages.

```
HRESULT GetLastErrorMessages();
```

Parameters

None

Return Values

STRING	Returns last error messages.

6.1.12 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 [▶ 37] 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

Shortcuts

The main tree items that exist in every configuration file can be accessed via shortcuts. These shortcuts are language neutral and require less memory:

Shortcut	Description
"**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
```

6.1.13 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 [▶ 37] 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.

6.1.14 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.

6.1.15 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() [▶ 36].

6.2 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 [▶ 35] 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)

ITcSmTreeItem Property	Type	Access	Description
			• 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 [▶ 35]. 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 [▶ 39].
ItemSubType	LONG	RW	Sub type [▶ 41] 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 [▶ 62]	Creates a child tree item.	TwinCAT 2.11
DeleteChild [▶ 63]	Deletes a child tree item.	TwinCAT 2.11
ImportChild [▶ 64]	Imports a child item from the clipboard or a previously exported file.	TwinCAT 2.11
ExportChild [▶ 64]	Exports a child item to the clipboard or a file.	TwinCAT 2.11
ProduceXml [▶ 60]	Returns a String containing the XML representation of the item, with all its item-specific data and parameters.	TwinCAT 2.11
ConsumeXml [▶ 61]	Consumes a String containing the XML representation of the tree item, with all its item-specific data and parameters.	TwinCAT 2.11
GetLastXmlError [▶ 66]	Gets the error message of the last erroneous ConsumeXml() call.	TwinCAT 2.11
LookupChild [▶ 65]	Searches for a child with the specified relative path.	TwinCAT 2.11

ITcSmTreeItem2 Methods	Description	Available since
ResoucesCount [▶ 65]	For interal use only	TwinCAT 2.11
ChangeChildSubType [▶ 65]	Changes the SubType of the ITcSmTreeItem.	TwinCAT 2.11
ClaimResources [▶ 66]	For internal use only	TwinCAT 2.11

ITcSmTreeItem3 Methods	Description	Available since
CompareItem [▶ 66]	Compares the actual ITcSmTreeItem to the specified other.	TwinCAT 2.11

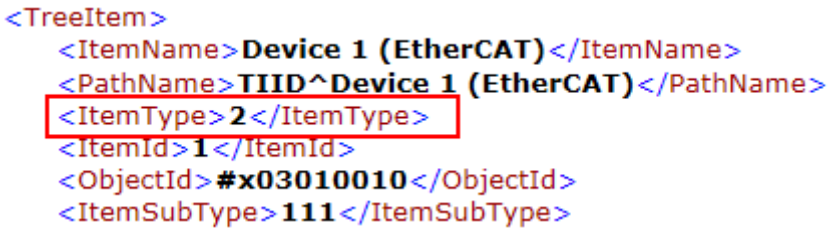
Version information

Required TwinCAT version
This interface is supported in TwinCAT 2.11 and above

6.2.1 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



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	---

Item type	Tag	Description
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]

Item type	Tag	Description
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

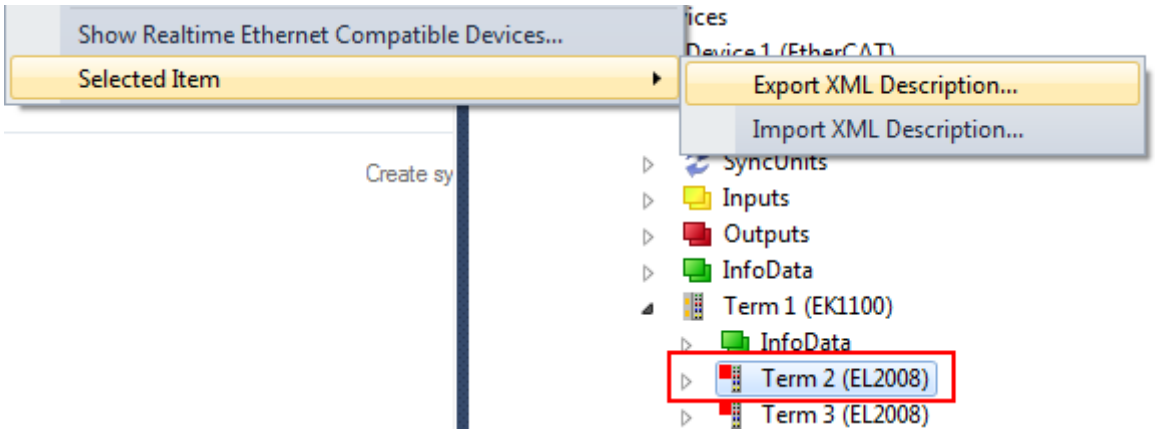
Notes	
[1]	requires TwinCAT 3.1

6.2.2 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 [▶ 45]
- Boxes [▶ 48]
- Terminals: E-Bus [▶ 43] (ELxxxx)
- Terminals: K-Bus digital input [▶ 52] (KL1xxx)
- Terminals: K-Bus digital output [▶ 53] (KL2xxx)
- Terminals: K-Bus analog input [▶ 55] (KL3xxx)
- Terminals: K-Bus analog output [▶ 56] (KL4xxx)
- Terminals: K-Bus position measurement [▶ 56] (KL5xxx)
- Terminals: K-Bus communication [▶ 57] (KL6xxx)
- Terminals: K-Bus power [▶ 57] (KL8xxx)
- Terminals: K-Bus safety [▶ 59] (KLx90x)
- Terminals: K-Bus system [▶ 58] (KL9xxx)

6.2.2.1 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() [▶ 62] 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() [▶ 62], 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:

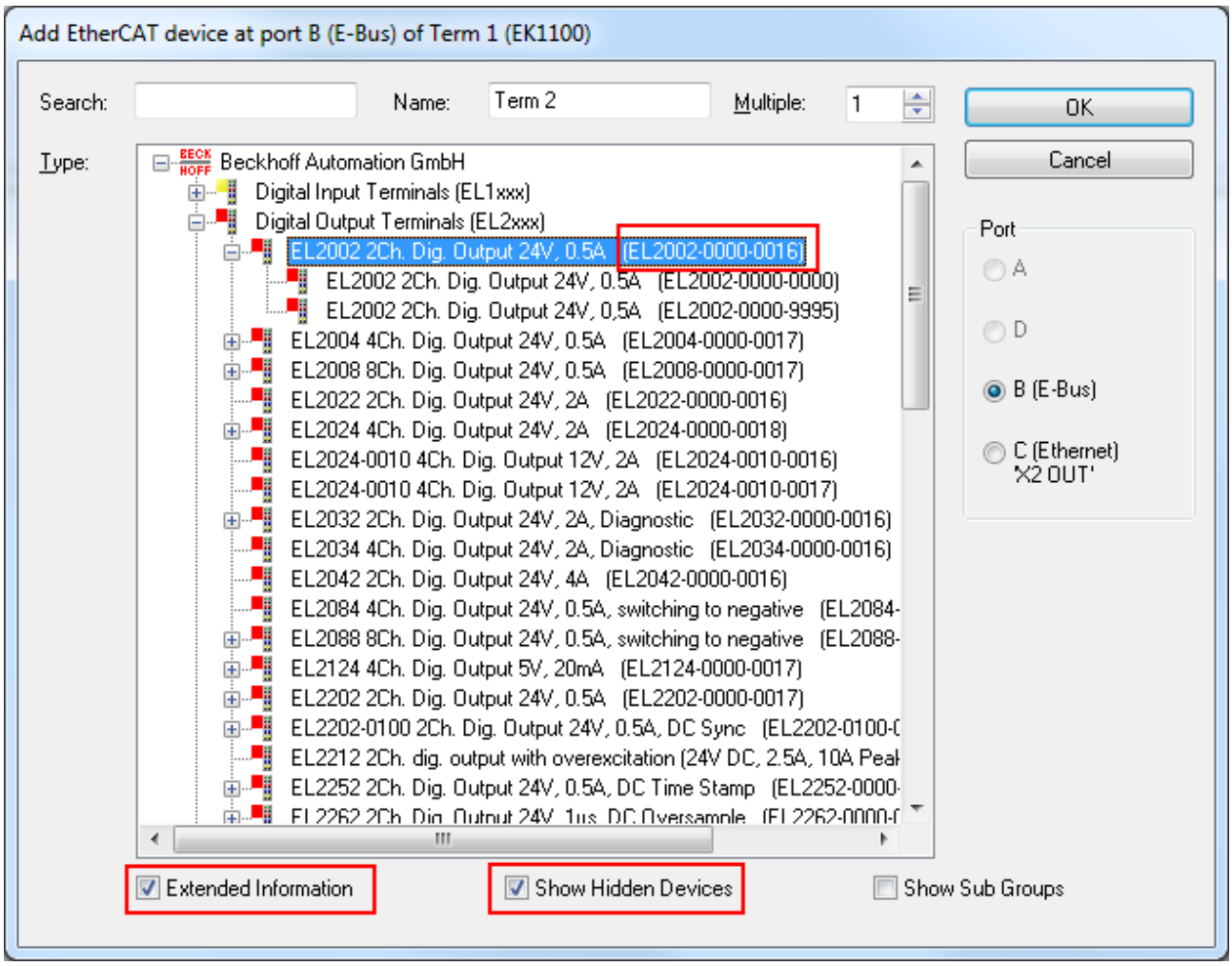
I/O	ItemSubType	Description
EP6002	9101	RS232 / RS422 / RS485 interface terminal
EL6001	9101	RS232 interface terminal
EL6002	9101	RS232 interface terminal (2-Channel)
EL6021	9103	RS422 / RS485 interface terminal
EL6022	9103	RS422 / RS485 interface terminal (2-Channel)

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 [▶ 20] 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"

6.2.2.2 Devices

11

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.

Sub type	Tag	Description
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)

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

Sub type	Tag	Description
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)

Sub type	Tag	Description
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)
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)

6.2.2.3 Boxes

Requirements

Sub type	Tag	Description
0	BOXTYPE_UNKNOWN	---
1	BOXTYPE_BK2000	BK2000 Lightbus coupler
2	BOXTYPE_M1400	M1400 Lightbus digital input/output module

Sub type	Tag	Description
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	---
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	---

Sub type	Tag	Description
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
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	---

Sub type	Tag	Description
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
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

Sub type	Tag	Description
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	---
9701	BOXTYPE_CX1103	---
9702	BOXTYPE_CX1190	---

6.2.2.4 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

Sub type	Description
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

6.2.2.5 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

Sub type	Description
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 state 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
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

6.2.2.6 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

Sub type	Description
3454	KL3454 4-Channel analog input terminal
3458	KL3458 8-Channel analog input terminal
3464	KL3464 4-Channel analog input terminal
3468	KL3468 8-Channel analog input terminal

6.2.2.7 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

6.2.2.8 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

6.2.2.9 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

6.2.2.10 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

Sub type	Description
100671906	KL8610 7-Channel adapter terminal KL8601
117449122	KL8610 8-Channel adapter terminal KL8601

6.2.2.11 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

Sub type	Description
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

6.2.2.12 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

6.2.2.13 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

6.2.2.14 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

6.2.2.15 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

Sub type	Description
838864512	KM3712 2-Channel absolute pressure measuring

6.2.2.16 Modules: K-Bus analog output (KM4)

Sub type	Description
838865402	KM4602 2-Channel analog output terminal

6.2.2.17 Modules: K-Bus communication (KM6)

Sub type	Description
6551	KM6551 IEEE802.15.4 terminal
838867463	KM6663 Ethernet changeover switch terminal

6.2.3 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 [▶ 61]

6.2.4 ITcSmTreeItem::ConsumeXml

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 [▶ 37] 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 must 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 parameters 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 [▶ 60] 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 [▶ 60] 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

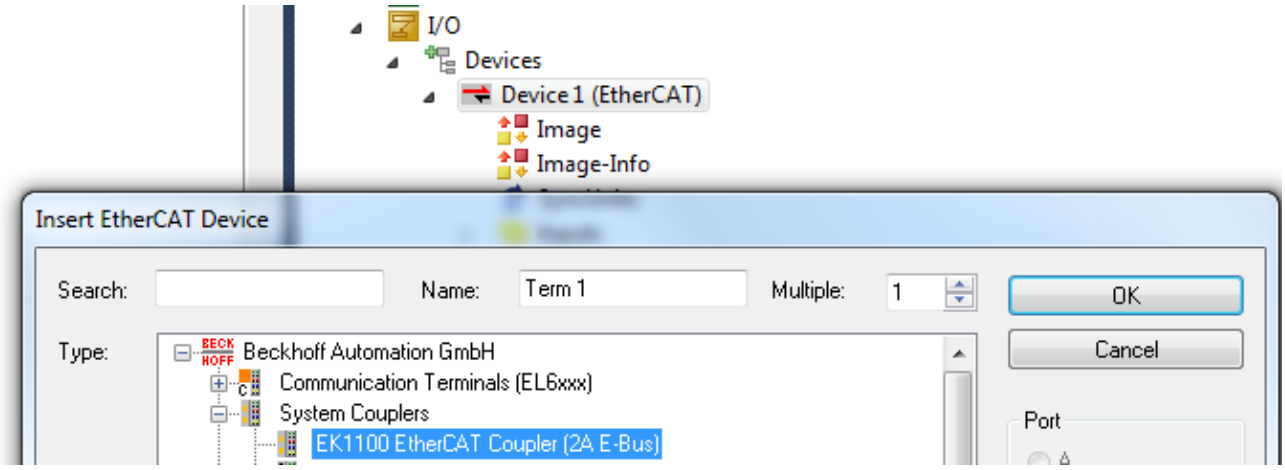
ITcSmTreeItem::ProduceXML [▶ 60]

6.2.5 ITcSmTreeItem::CreateChildITcSmTreeItem

Creates a child itemon a parent node. The childis being specified by it'ssubtype [▶ 41].

```
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 [▶ 41]ub type [▶ 41] of the new child item. The usable sub type depends on theitem type [▶ 39](the category)of theparenttree 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, whichdepends on the sub type [▶ 41].The different dependencies are listed in the table below.
pipItem	[out, retval] Points to the location of a ITcSmTreeItem [▶ 37] 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.

Child: Item sub type	vInfo parameter(s)
E-Bus box / terminal / module (item sub type 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"): • V%X_P%X_R%X_S%X • V%X_P%X_R%X • V%X_P%X Especially for Beckhoff E-Bus terminals / modules, please read the corresponding E-Bus article [▶ 43].
Interbus box. (item sub type 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 (item sub type 5007) CANDrive (item sub type 5006)	Optionally contains a bool value (VT_BOOL) and if set an additional variable for the `Following Error`will be created.
DeviceNET (item sub type 5203) TcDNSSlave (item sub type 5250) CX1500 (item sub type 1042) FC520X Slave (item sub type 1043)	Optionally contains the file path to an EDS file (VT_BSTR).
BK3000 and all other PROFIBUS box types	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).
PLC POU Functionblock (item sub type 604)	Contains IEC language type as Integer, as defined by IECLANGUAGETYPES.
PLC POU Function (item sub type 603)	Contains a string[2] array, which hols the following values: • array[0] = IEC language type as Integer, as defined by IECLANGUAGETYPES. • array[1] = return data type of function. May be any PLC data type, for example DINT, BOOL, ...

Also see about this

- ITcSmTreeItem [▶ 37]

6.2.6 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.

6.2.7 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 [▶ 37] 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.

6.2.8 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.

6.2.9 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 [▶ 37] 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.

6.2.10 ITcSmTreeItem2::ResourcesCount

For internal use only.

```
HRESULT ResourcesCount(long*pnCount);
```

Parameters

pnCount	[out,retval] Count of resources.

Return Values

S_OK	function returns successfully.

6.2.11 ITcSmTreeItem::ChangeChildSubType

For internal use only.

```
HRESULT ChangeChildSubType(BSTRbstrChild, longnNewSubType,VARIANT_BOOLbAggressive, VARIANTvInfo, ITc
SmTreeItem2**pipItem);
```

Parameters

bstrChild	[in]
nNewSubType	[in]
bAggressive	[in, defaultvalue(-1)]
vInfo	[in, optional]
pipItem	[out, retval]

Return Values

S_OK	function returns successfully.
TSM_E_ITEMNOTFOUND (0x98510001)	the path name does not qualify an existing tree item.

6.2.12 ITcSmTreeItem2::ClaimResources

For internal use only.

```
HRESULT ClaimResources(longresourceIdx);
```

Parameters

resourceIdx	[in,defaultValue(1)]

Return Values

S_OK	function returns successfully.

6.2.13 ITcSmTreeItem3::CompareItem

Compares the current ITcSmTreeItem [▶ 37] with a specified second one for its XML content (ProduceXml comparison). This comparison can be done only for the specified nodes or for the whole subtree (recursive).

```
HRESULT CompareItem(
  ITcSmTreeItem* pItem,
  VARIANT_BOOL bRecursive,
  long* pnRet
);
```

Parameters

pItem	[in] Reference to the ITcSmTreeItem [▶ 37] to be compared with the current ITcSmTreeItem [▶ 37].
bRecursive	[in, defaultvalue(0)] Indicates that the comparison should be processed recursively for the whole subtree (Subtree comparison)
pnRet	[out, retval]: Return value: 0 for Equality, ret > 0 or ret < 0 for InEquality.

Return Values

S_OK	Compare Item succeeded.

6.2.14 ITcSmTreeItem::GetLastXmlError

Gets the Item path / Error message of the last erroneous ConsumeXml [▶ 61] call.

```
HRESULT GetLastXmlError(BSTR *pXML);
```

Parameters

pXML	[out, retval] Error message.

Return Values

S_OK	function returns successfully.

7 Samples

Description	Programming language	Download	Comments
Sample 1: Adding of Additional Tasks and Variables; TwinCAT Start / Stop	C++	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425899275/.exe (Self extracting)	Visual Studio 2008 project
Sample 2: OpenConfiguration and CloseConfiguration, Scan Devices, Scan Boxes, Scan Terminals, Adding EtherCAT Devices, Import/Export	C#	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425900683/.zip	Visual Studio 2010 project (.NET 2.0)
Sample 3: Linking a PLC project	C# PowerShell	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425902091/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425903499/.zip	Visual Studio 2010 project (.NET 4.0) Requires Windows PowerShell
Sample 4: Activating a configuration	C# PowerShell IronPython	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425904907/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425906315/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425907723/.zip	Visual Studio 2010 project (.NET 4.0) Requires Windows PowerShell Requires IronPython Interpreter
Sample 5: Linking PLC variables to IO	C# PowerShell IronPython	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425909131/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425910539/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425911947/.zip	Visual Studio 2010 project (.NET 4.0) Requires Windows PowerShell Requires IronPython Interpreter
Sample 6: Activate configuration on a remote target	C# PowerShell IronPython	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425913355/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425914763/.zip	Visual Studio 2010 project (.NET 4.0) Requires Windows PowerShell Requires IronPython Interpreter

Description	Programming language	Download	Comments
		https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425916171/.zip	
Sample 7: Creating System Manager project with multiple EtherCAT boxes, including PLC and IO linking (very detailed sample)	C# PowerShell IronPython	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425917579/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425918987/.zip https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425920395/.zip	Visual Studio 2010 project (.NET 4.0) Requires Windows PowerShell Requires IronPython Interpreter
Sample 8: Scanning for new devices and boxes	C#	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425921803/.zip	Visual Studio 2010 project (.NET 2.0)
Sample 9: Adding an ADS route to a remote target	C#	https://infosys.beckhoff.com/content/1033/tcautomationinterface/Resources/12425923211/.zip	Visual Studio 2010 project (.NET 2.0)

8 How to

Instead of downloadable sample code which comes included in a Visual Studio project, this section provides code snippets for several tasks.

Please note that many code snippets are also included in some way or another in one of our sample code downloads [▶ 67], which should allow you to understand its usage in an overall picture.

How To
How to - Activate a previously created configuration [▶ 69]
How to - Link variables [▶ 70]
How to - Unlink variables [▶ 70]
How to - Disable/enable a tree item (e.g. an I/O device) [▶ 71]
How to - Change the path to a plc project and/or rescan the project [▶ 71]
How to - Change the fieldbus address (station no.) of a profibus box [▶ 72]
How to - Export/import the information of a child of a tree item [▶ 73]
How to - Enumerate through the child's of a tree item [▶ 74]
How to - Add child items (devices, boxes, terminals, variables etc.) [▶ 75]
How to - Update the address information of I/O devices [▶ 75]
How to - Create/change a linkage of a nc axis, encoder or drive to an I/O device, box or terminal [▶ 76]
How to - Exchange a fieldbus device and move all childs to the new one [▶ 77]
How to - Add a route to a remote ADS target [▶ 79]
How to - Execute a broadcast search [▶ 81]

8.1 How to activate a previously created configuration

Requirements

To activate a previously created configuration, an instance of the TwinCAT System Manager must be created, the configuration has to be loaded and activated.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface allows same basic operations with TwinCAT configurations.

Sample (vbscript):

```
dim tsm
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
call tsm.ActivateConfiguration
```

Sample (javascript):

```
var tsm
tsm = ActiveXObject("TCatSysManager.TcSysManager");
tsm.OpenConfiguration("c:\twincat\myConfig.wsm");
tsm.ActivateConfiguration();
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30]

Built on Tuesday, Mai 01, 2001

8.2 How to link variables

Requirements

To link variables in a configuration, an instance of the TwinCAT System Manager must be created, and the two variables must be referenced by their pathnames

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LinkVariables [▶ 33] method. To link the two variables "*TIPC^Project 1^Standard^Outputs^MyOutput*" and "*TIID^Device 1 (C220)^Box 1^Term 1^Channel 1^Output*" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
call tsm.LinkVariables("TIPC^Project1^Standard^Outputs^MyOutput", "TIID^Device 1 (C220)^Box1^Term 1^
Channel 1^Output")
call tsm.SaveConfiguration
```

If the two variables have different sizes, the LinkVariables [▶ 33] method has three further optional parameters to specify which bits and bytes are to be linked together.

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], ITcSysManager::LinkVariables [▶ 33]

Built on Tuesday, Mai 01, 2001

8.3 How to unlink variables

Requirements

To unlink variables in a configuration, an instance of the TwinCAT System Manager must be created and the two variables to be unlinked from each other must be referenced by their pathnames

An alternative is to reference only to one variable and all linkages of that variable will be removed.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has an UnlinkVariables [▶ 34] method. To unlink the variable "TIPC^Project 1^Standard^Outputs^MyOutput" from "TIID^Device 1 (C220)^Box 1^Term 1^Channel 1^Output" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
call tsm.UnlinkVariables("TIPC^Project1^Standard^Outputs^MyOutput", "TIID^Device 1 (C220)^Box1^Term
1^Channel 1^Output")
call tsm.SaveConfiguration
```

If all linkages from "TIPC^Project 1^Standard^Outputs^MyOutput" should be removed, the following code can be used:

```
dim tsm
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
call tsm.UnlinkVariables("TIPC^Project1^Standard^Outputs^MyOutput")
call tsm.SaveConfiguration
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], ITcSysManager::UnlinkVariables [▶ 34]

Built on Tuesday, Mai 01, 2001

8.4 How to disable/enable a tree item (e.g. an I/O device)

Requirements

To disable/enable a tree item in a configuration, an instance of the TwinCAT System Manager must be created, and the configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the tree item referenced by its pathname.

This interface contains a Disabled property of the tree item.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname. To disable/enable the tree item "*TIID^Device 1 (C1220)*" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
dim item
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1(C220)")
item.Disabled = SMDS_DISABLED '(or item.Disabled =SMDS_NOT_DISABLED to enable the tree item)
call tsm.SaveConfiguration
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], Disabled

Built on Tuesday, Mai 01, 2001

8.5 How to change the path to a plc project and/or rescan the project

Requirements

To change the path to a plc project and/or rescan the project in a configuration, an instance of the TwinCAT System Manager must be created, and the configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the tree item referenced by its pathname

This interface contains a ConsumeXml [▶ 61] method of the tree item.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname. To change the path to a plc project and rescan the project "*TIPC^MyProject*" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
dim item
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIPC^MyProject")
call item.ConsumeXml("<TreeItem><PlcDef><ProjectPath>c:\twincat\plc\NewProject.pro</
ProjectPath><ReScan>1</ReScan></PlcDef></TreeItem>")

call tsm.SaveConfiguration
```

The next sample just rescans the current project:

```
dim tsm
dim item
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIPC^MyProject")
call item.ConsumeXml("<TreeItem><PlcDef><ReScan>1</ReScan></PlcDef></TreeItem>")

call tsm.SaveConfiguration
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ConsumeXml [▶ 61]

Built on Tuesday, Mai 01, 2001

8.6 How to change the fieldbus address (station no.) of a profibus box

Requirements

To change the the fieldbus address (station no.) of a profibus box in a configuration, an instance of the TwinCAT System Manager must be created, and the configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the tree item referenced by its pathname.

This interface contains a ConsumeXml [▶ 61] method of the tree item.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname. To change the the fieldbus address (station no.) of a profibus box "*TIID^Device 1 (FC310x)^Box 1 (BK3100)*" to 44 the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
dim item
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1 (FC310x)^Box 1(BK3100)")
```

```
call item.ConsumeXml("<TreeItem><BoxDef><FieldbusAddress>44</FieldbusAddress></BoxDef></TreeItem>")

call tsm.SaveConfiguration
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ConsumeXml [▶ 61]

Built on Tuesday, Mai 01, 2001

8.7 How to export/import the information of a child of a tree item

Requirements

To export/import the information of a child of a tree item in a configuration, an instance of the TwinCAT System Manager must be created, and a configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the parent tree item referenced by the parents pathname.

This interface contains an ExportChild [▶ 64] and an ImportChild [▶ 64] method of the tree item.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname. To export the information of a tree item "*TIID^Device 1 (FC310x)^Box 1 (BK3100)*" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
dim item
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1(FC310x)")
call item.ExportChild("Box 1 (BK3100)","c:\twincat\box1.tce")
```

To import the information of a tree item as a child of "*TIID^Device 1 (FC310x)*" the following vbscript code can be used:

```
dim tsm
dim item
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1(FC310x)")
call item.ImportChild("c:\twincat\box1.tce")
```

The ImportChild [▶ 64] method has same optional parameters that are described in the documentation of that method.

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ExportChild [▶ 64], ImportChild [▶ 64]

Built on Tuesday, Mai 01, 2001

8.8 How to enumerate through the child's of a tree item

Requirements

To enumerate through the child's of a tree item in a configuration, an instance of the TwinCAT System Manager must be created and a configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the tree item referenced by its pathname.

This interface contains two ways of child enumeration. The first one is the _NewEnum property that is automatically used by the VB ForEach statement and enumerates over all childs of a tree item. This includes for example the process image child and the input and output childs of an I/O device. The second way is to use the ChildCount and Child(i) > property that allows to enumerate over the "main" child (e.g. only over the I/O boxes of an I/O device).

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname. To enumerate through the all childs of a tree item "*TIID^Device 1 (FC310x)^Box 1 (BK3100)*" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
dim item
dim str
dim itemChild
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1(FC310x)")
For Each Child In item
    Set itemChild = Child
    str = itemChild.Name
Next Child
```

To enumerate only through the main childs of a tree item "*TIID^Device 1 (FC310x)^Box 1 (BK3100)*" the following vbscript code can be used:

```
dim tsm
dim item
dim str
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1(FC310x)")
For i = 1 To item.ChildCount
    str = item.Child(i).Name
Next
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], _NewEnum, ChildCount, Child(i)

Built on Tuesday, Mai 01, 2001

8.9 How to add child items (devices, boxes, terminals, variables etc.)

Requirements

To add child items (devices, boxes, terminals, variables etc.) in a configuration, an instance of the TwinCAT System Manager has to be created and a configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the parent tree item referenced by the parents pathname.

This interface contains a CreateChild method.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname. To add a child item to a tree item "*TIID^Device 1 (FC310x)*" the following vbscript code can be used:

Sample (vbscript):

```
dim tsm
dim item
dim str
dim itemChild
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\twincat\myConfig.wsm")
set item = tsm.LookupTreeItem("TIID^Device 1(FC310x)")
call item.CreateChild("Box 1 (BK3100)",BOXTYPE_BK3100)
```

The tree item type of the child is implicit given by the parent type (e.g. child of I/O devices are always I/O boxes, child of bus couplers are always terminals...). The item sub type must be specified as a parameter of CreateChild [▶ 62]. The CreateChild [▶ 62] method has same optional parameters that are described in the documentation of that method.

Optional *CreateChild()* parameter 'vInfo':

See CreateChild [▶ 62].

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], CreateChild [▶ 62]

Built on Tuesday, Mai 01, 2001

8.10 How to update the address information of I/O devices

Requirements

To update the address information of I/O devices in a configuration, an instance of the TwinCAT System Manager must be created, and a configuration must be opened. A call to the ProduceXml [▶ 60] method of the "I/O devices" tree item retrieves the current address information of all installed I/O devices (internally by calling the scan devices function). This address information can be used to update a specific I/O device in the configuration.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface. This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to the "I/O Devices" tree item (use the abbreviation "TIID").

Sample (vbscript):

```
Set tsm = CreateObject("TCatSysManager.TcSysManager")
Call tsm.OpenConfiguration("c:\test.wsm")
Set Item = tsm.LookupTreeItem("TIID")
Set xml = CreateObject("Msxml.DOMDocument")
```

The following call to Item.ProduceXml scans the installed I/O devices (among other things) and returns a XML string that contains these information that will be stored in the XML document by the method xml.loadXml.

```
Call xml.loadXml(Item.ProduceXml)
```

The following line sets a DOM node list object that contains all found devices as DOM nodes

```
Set devs = xml.selectNodes("TreeItem/DeviceGrpDef/FoundDevices/Device")
For i = 0 To devs.length - 1
```

If the ItemSubType of the device corresponds to the type of device to update...

```
    If devs(i).selectSingleNode("ItemSubType").nodeTypedValue = 48 Then
        ' 48 = IODEVICETYPE_FC7500
```

The AddressInfo node of that device will be extracted

```
        Set ainfo = devs(i).selectSingleNode("AddressInfo")
```

A ITcSmTreeItem pointer of that device in the configuration will be evaluated

```
        Set dev = tsm.LookupTreeItem("TIID^MyDev")
```

And finally the address info will be stored to the device

```
       Call dev.ConsumeXml(ainfo.xml)
       Call tsm.SaveConfiguration
       Exit For
    End If
Next
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ConsumeXml [▶ 61], ProduceXml [▶ 60]

Built on Tuesday, Mai 01, 2001

8.11 How to create/change a linkage of a nc axis, encoder or drive to an I/O device, box or terminal

Requirements

To create/change a linkage of a nc axis, encoder or drive to an I/O device, box or terminal in a configuration, an instance of the TwinCAT System Manager has to be created and a configuration has to be opened.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface.

Sample (vbscript):

```
Set tsm = CreateObject("TCatSysManager.TcSysManager")
Call tsm.OpenConfiguration("c:\test.wsm")
Set Item = tsm.LookupTreeItem("TINC^NC-Task 1SAF^Axes^Axis 1")
Call Item.ConcumeXml("<TreeItem><NcAxisDef><IoItem><PathName>TIID^Device1 (FC750x)^Box 1 (SercosAxis
)</PathName></IoItem></NcAxisDef></TreeItem>")
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ConsumeXml [▶ 61]

Built on Tuesday, Mai 01, 2001

8.12 How to exchange a fieldbus device and move all childs to the new one

Requirements

To exchange a fieldbus device and move all childs to the new one, an instance of the TwinCAT System Manager must be created and a configuration has to be opened. The LookupTreeItem [▶ 35] method of the ITcSysManager [▶ 30] interface returns a ITcSmTreeItem [▶ 37] interface pointer implemented by the old fieldbus device. This interface contains a ExportChild [▶ 64] and a ImportChild [▶ 64] method of the tree item.

Procedure

The ProgId "**TCatSysManager.TcSysManager**" can be used to create an instance of the System Manager that implements the ITcSysManager [▶ 30] interface.

Sample (vbscript):

```
set tsm = CreateObject("TCatSysManager.TcSysManager")
call tsm.OpenConfiguration("c:\exchange.wsm")
set item = tsm.LookupTreeItem("TIID")
set itemNew = item.CreateChild("Device 2 (FC310x)",38)
set itemOld = tsm.LookupTreeItem("TIID^Device 1(CP5412)")
for i = 1 to itemOld.ChildCount
    call itemOld.ExportChild(itemOld.Child(i).Name)
    call itemNew.ImportChild()
next
set itemOld = nothing
call item.DeleteChild("Device 1 (CP5412)")
call tsm.SaveConfiguration("c:\exchange2.wsm")
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ExportChild [▶ 64], ImportChild [▶ 64]

Built on Tuesday, Mai 01, 2001

8.13 How to scan devices and boxes

Requirements

When creating a new configuration, it is often necessary to align the TwinCAT XAE configuration to the actually available hardware. One option to fulfill 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 [▶ 30] interface (the '*systemManager*' instance here) is described in the chapter Accessing TwinCAT Configurations. [▶ 17] This interface has a LookupTreeItem [▶ 35] method that returns a ITcSmTreeItem [▶ 37] pointer to a specific tree item given by its pathname, in this case the shortcut "TIID" which references the I/O devices node.

Code snippet (C#):

```
ITcSysManager3 systemManager = null; // How to create the System Manager instance and how to open co
nfiguration is shown in Concepts chapter

public void ScanDevicesAndBoxes()
{
     systemManager.SetTargetNetId("1.2.3.4.5.6"); // Setting of the target NetId.
     ITcSmTreeItem ioDevicesItem = systemManager.LookupTreeItem("TIID"); // Get The IO Devices Node

     // Scan Devices (Be sure that the target system is in Config mode!)
     string scannedXml = ioDevicesItem.ProduceXml(false); // Produce Xml implicitly starts the ScanD
evices on this node.

     XmlDocument xmlDoc = newXmlDocument();
     xmlDoc.LoadXml(scannedXml); // Loads the Xml data into an XML Document
     XmlNodeList xmlDeviceList = xmlDoc.SelectNodes("TreeItem/DeviceGrpDef/FoundDevices/Device");
     List<ITcSmTreeItem> devices = newList<ITcSmTreeItem>();

     int deviceCount = 0;

     // Add all found devices to configuration
     foreach (XmlNode node in xmlDeviceList)
     {
          // Add a selection or subrange of devices
          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); // Consume Xml Parameters (here the Address of the Device)
          devices.Add(device);
     }

     // Scan all added devices for attached boxes
     foreach (ITcSmTreeItem device in devices)
     {
          string xml = "<TreeItem><DeviceDef><ScanBoxes>1</ScanBoxes></DeviceDef></
TreeItem>"; // Using the "ScanBoxes XML-Method"
          try
          {
               device.ConsumeXml(xml); // Consume starts the ScanBoxes and inserts every found box/
terminal into the configuration
          }
          catch (Exception ex)
          {
               Console.WriteLine("Warning: {0}",ex.Message);
          }

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

Code snippet (IronPython):

```
import xml.etree.ElementTree as ET

# Create the System Manage
instance “sysMan” first
```

```
ioDevicesItem = sysMan.LookupTreeItem("TIID") # Get the IO Devices Node
scannedXml = ioDevicesItem.ProduceXml(False)  # Produce Xml implicitly starts the ScanDevices on thi
s node

xmlDoc =      ET.fromstring(scannedXml) # Loads the Xml data into an XML Document
xmlDeviceList = xmlDoc.findall("./DeviceGrpDef/FoundDevices/Device")
devices = [] # Container to store Scanned devices
deviceCount = 0


# Add the Scanned devices
for device in xmlDeviceList:
       typeName = device.findtext("ItemSubTypeName")
    itemSubType = device.findtext("ItemSubType")
     xmlAddress = device.find("AddressInfo")
 deviceCount = deviceCount + 1;
  device =      ioDevicesItem.CreateChild("Device_" + str(deviceCount) + typeName, itemSubType,"",No
ne)
      xml = "<TreeItem><DeviceDef>" + ET.tostring(xmlAddress) + "</DeviceDef></TreeItem>"
    device.ConsumeXml(xml); # Consumes Xml Parameters (here the Address of the Device)
    devices.append(device);

for device in devices:
      xml = "<TreeItem><DeviceDef><ScanBoxes>true</ScanBoxes></DeviceDef></TreeItem>"
      try:
      print device.ConsumeXml(xml) # Consumes starts the ScanBoxes and inserts every found box/
Terminal into the config
     except:
        print "Scan Failed!"

for box in device:
   print box.Name
```

8.14 How to add routes to a remote ADS device

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

XML structure

The following snippet is a sample XML structure for adding routes to a remote target.

Please note that you can either specify the IP address or the hostname of the remote target.

```
<TreeItem>
    <ItemName>Route Settings</ItemName>
    <PathName>TIRR</PathName>
    <RoutePrj>
        <TargetList>
            <!-- Triggers Broadcast search -->
            <BroadcastSearch>true</BroadcastSearch>
        </TargetList>
        <AddRoute>
            <!-- Description of the route on the remote device, can be any name -->
            <RemoteName>RouteName</RemoteName>
            <!-- AdsNetID of the remote device -->
            <RemoteNetId>1.2.3.4.5.6</RemoteNetId>
            <!-- IP Address of the remote target -->
            <RemoteIpAddr>1.2.3.4</RemoteIpAddr>
            <!-- or -->
            <!-- Hostname of the remote target -->
            <RemoteHostName>hostName</RemoteHostName>
            <!-- Username used to connect to remote device -->
            <UserName>userName</UserName>
            <!-- Password used to connect to remote device -->
            <Password>password</Password>
            <!-- Encrypt password -->
            <NoEncryption></NoEncryption>
            <!-- Description of the route on the local device, can be any name-->
            <LocalName>LocalName</LocalName>
        </AddRoute>
        <!-- Use for adding a project route -->
```

```
        <AddProjectRoute>
            <!-- Description of the route on the local device, can be any name-->
            <Name>RouteName</Name>
            <!-- AdsNetID of the remote device -->
            <NetId>1.2.3.4.5.6</NetId>
            <!-- Hostname of the remote target -->
            <HostName>hostName</HostName>
            <!-- or -->
            <!-- IP Address of the remote target -->
            <IpAddr>1.2.3.4</IpAddr>
        </AddProjectRoute>
    </RoutePrj>
</TreeItem>
```

Sample (C#):

The following code snippet can also be downloaded in our sample section [▶ 67] (Sample 13). It consumes the following XML structure on the node "Route Settings":

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

Please adapt the NetIds and IP addresses to your needs.

```
class Program
{
   private static string _tsmPath="C:\\Temp\\Sample.tsm";
   private static TcSysManager _sysManager;

   static voidMain(string[]args)
   {
        _sysManager = new TcSysManager();
        _sysManager.NewConfiguration();

       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 routesNode=_sysManager.LookupTreeItem("TIRR");
        routesNode.ConsumeXml(xmlString);

        _sysManager.SaveConfiguration(_tsmPath);
   }
}
```

See Also

How to... section [▶ 69], ITcSysManager [▶ 30], LookupTreeItem [▶ 35], ITcSmTreeItem [▶ 37], ConsumeXml [▶ 61]

Also see about this

- Samples [▶ 67]

8.15 How to execute a broadcast search

Requirements

To find unknown remote ADS devices, a broadcast search needs to be executed. This can be done by making use of the ConsumeXml() and ProduceXml() methods of the TwinCAT Automation Interface.

Sample (C#):

```
class Program
{
    private static string _tsmPath = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location) + "
\\Templates\\Sample.tsm";
    private static TcSysManager _sysManager;

    static void Main(string[] args)
    {
        _sysManager = new TcSysManager();
        _sysManager.NewConfiguration();

        /* ==========================================
                     * XML String as shown above
                     * ========================================== */
       string xmlString = "<TreeItem><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList></RoutePrj></TreeItem>";

       /* ==========================================

* Lookup System Manager node "SYSTEM^Route Settings" using Shortcut "TIRR" and consume XML string
                     * ========================================== */
       ITcSmTreeItem routesNode = _sysManager.LookupTreeItem("TIRR");
       routesNode.ConsumeXml(xmlString);

       /* ==========================================
        * Calling ProduceXml() on the same node executes the broadcast search and returns its result
s
        * ========================================== */
       string result = routesNode.ProduceXml();
       Console.WriteLine(result);

       /* ==========================================
        * Save configuration
        * ========================================== */
        sysManager.SaveConfiguration(_tsmPath);
   }
}
```

More Information:
www.beckhoff.com/automation

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