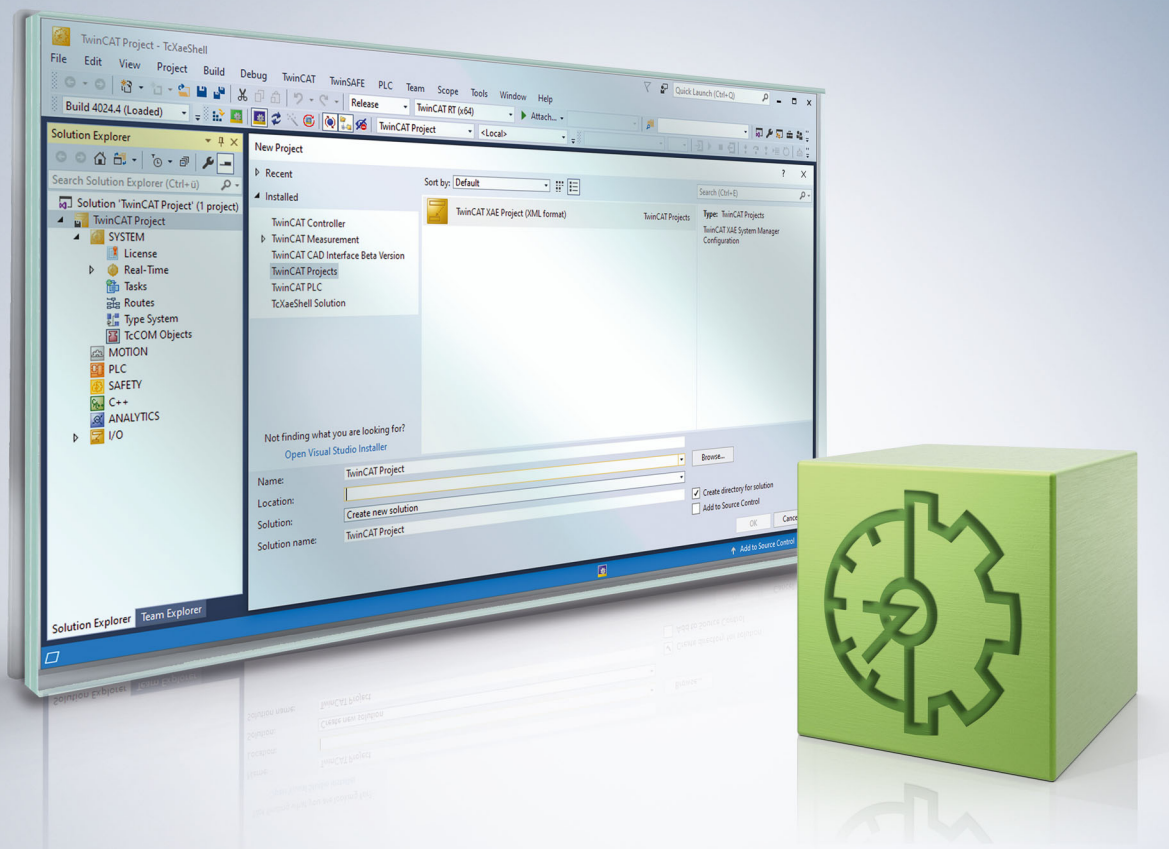
BECKHOFF New Automation Technology

Handbuch | DE

Automation Interface

TwinCAT 3



31.08.2022 | Version: 1.5

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

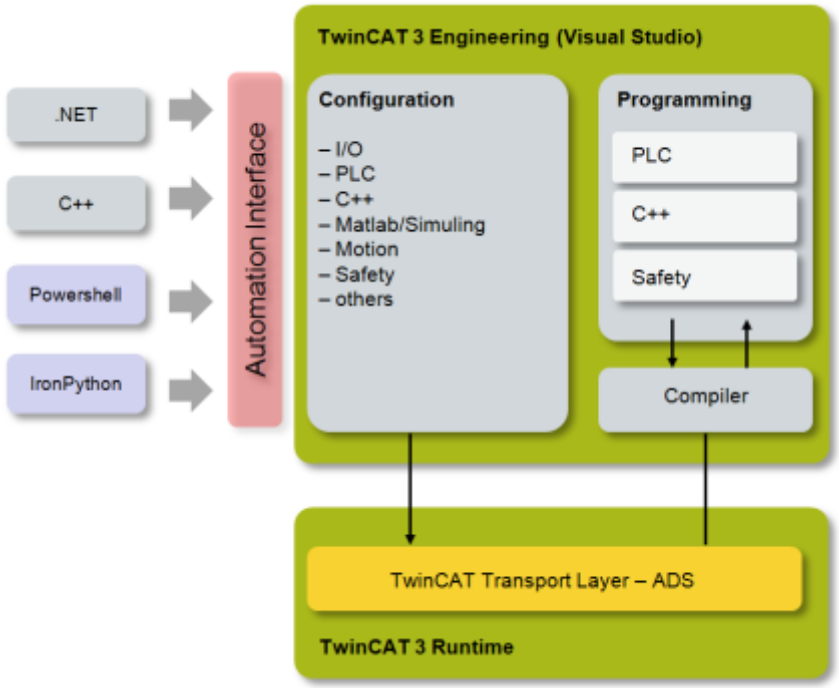
Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

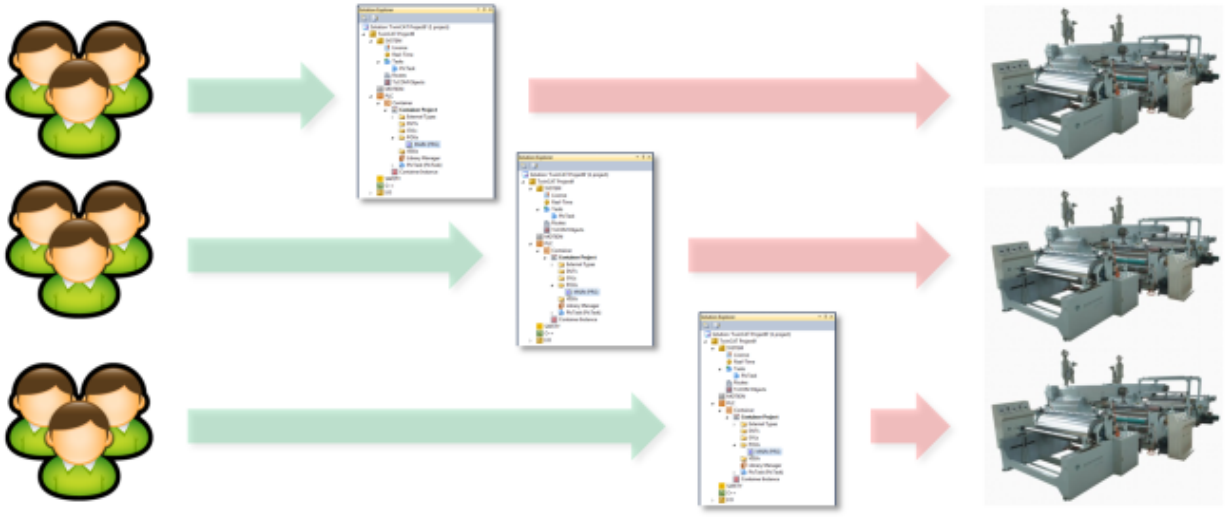
2.1 Produktbeschreibung

Mit dem TwinCAT Automation Interface können TwinCAT XAE Konfigurationen per Programmier-/Skriptcodes automatisch erzeugt und bearbeitet werden. Die Automatisierung einer TwinCAT-Konfiguration steht dank sogenannter Automation Interfaces zur Verfügung, auf die über alle COM-fähigen Programmiersprachen (z.B. C++ oder .NET) und auch über dynamische Scriptsprachen wie Windows PowerShell, IronPython oder sogar das (veraltete) VBscript zugegriffen werden kann. Diese Automation Interfaces sind an das Visual Studio-Automatisierungsmodell gebunden, einem mit TwinCAT3-Funktionen erweiterten Visual Studio.

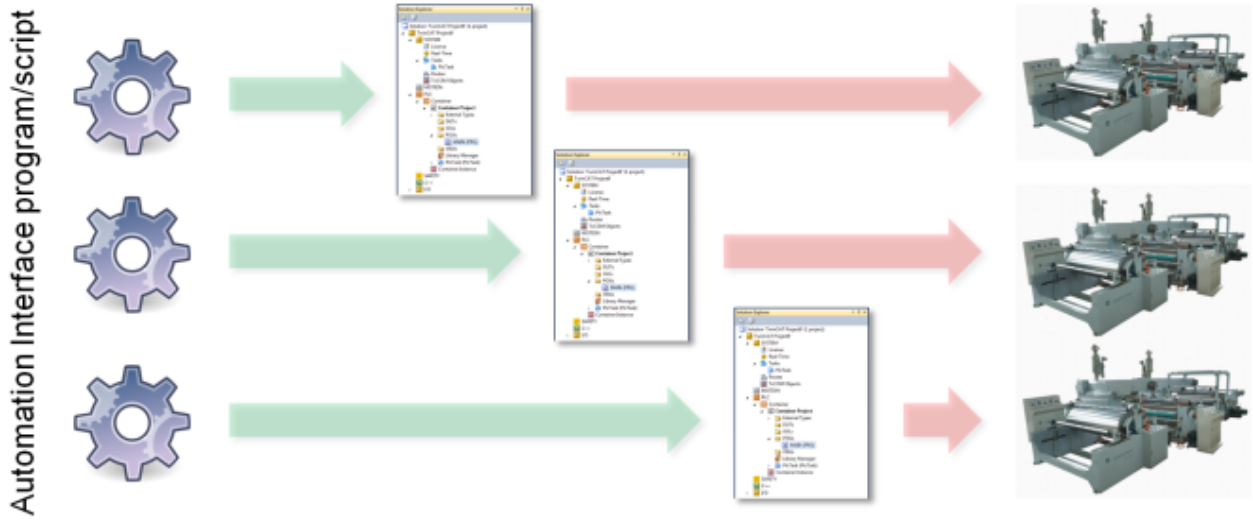


Das TwinCAT-Automation Interface ermöglicht einen effizienten Entwicklungsprozess, indem den Kunden die Möglichkeit geboten wird, die Konfiguration einer umfassenden TwinCAT-Lösung zu automatisieren.

Bisher, im traditionellen Engineering-Gedanken, musste eine Maschinenkonfiguration manuell an jedes neue Projekt angepasst oder sogar ganz neu erstellt werden, was nicht bloß einen riesigen, mit hohen Kosten verbundenen Entwicklungsaufwand bedeutete, sondern auch mit einer erheblichen, durch menschliches Eingreifen verursachten Fehleranfälligkeit einherging.



Dank des TwinCAT-Automation Interface kann der Prozess der Anpassung von TwinCAT-Konfigurationen an eine neue Umgebung oder sogar die Erstellung ganz neuer TwinCAT-Konfigurationen entsprechend den Anforderungen der Kunden automatisiert werden.



Der Leser sollte sich nun den folgenden Themen zuwenden:

Grundlagen

Thema	Beschreibung
TwinCAT XAE Konfigurationen erstellen/laden [▶ 20]	Beschreibt, wie eine TwinCAT-Konfiguration erzeugt oder geöffnet wird.
TwinCAT XAE Navigation [▶ 23]	Beschreibt, wie durch eine TwinCAT-Konfiguration navigiert wird.
Benutzerdefinierte Tree Item Parameter [▶ 25]	Beschreibt, wie auf benutzerdefinierte Parameter eines Elements zugegriffen wird. Dies ist wichtig für den Zugriff auf die Konfigurationsparameter eines Tree Items.
Implementierung eines COM-Nachrichtenfilters [▶ 27]	Beschreibt, wie ein eigener COM-Nachrichtenfilter zu implementieren ist, um abgewiesene COM-Aufrufe zu umgehen

Best practice

Thema	Beschreibung
Erstellung von und Umgang mit SPS-Projekten [▶ 57]	Beschreibt den Umgang mit SPS-Projekten
Erstellung von und Umgang mit SPS-POUs [▶ 66]	Beschreibt den Umgang mit SPS-Objekten/Code
Erstellung von und Umgang mit SPS-Bibliotheken [▶ 61]	Beschreibt den Umgang mit SPS-Bibliotheken, Repositories und Platzhaltern
Erstellung von und Umgang mit MOTION-Projekten [▶ 113]	Beschreibt die Erstellung von TwinCAT Motion-Projekten (NC-Task, Achsen, ...)
Erstellung von und Umgang mit EtherCAT-Teilnehmern [▶ 71]	Beschreibt die Erstellung von EtherCAT-Teilnehmern und deren Anschluss an eine EtherCAT-Topologie
Erstellung von und Umgang mit TwinCAT Measurement [▶ 105]	Beschreibt den Umgang mit TwinCAT Measurement -Projekten.
Erstellung von und Umgang mit TcCOM Modulen [▶ 96]	Beschreibt den Umgang mit TcCOM-Modulen.
Verwendung von Templates [▶ 45]	Beschreibt den Prozess der Erstellung und Verwendung von Templates.
Erstellung von und Umgang mit Netzwerkvariablen [▶ 77]	Beschreibt die Erstellung von Netzwerkvariablen (Publisher/Subscriber-Variablen)
Erstellung von und Umgang mit Tasks [▶ 44]	Beschreibt die Erstellung von Tasks und deren Verknüpfung mit anderen Objekten (SPS-Projekten, ...)
Von Offline zu Online-Konfigurationen [▶ 51]	Einige I/O Geräte benötigen physikalische Adressinformationen, bevor die Konfiguration aktiviert werden kann. Dieser Artikel erläutert, wie diese Informationen beschafft und eingestellt werden.
Zugriff auf das Fehlerlistenfenster von Visual Studio [▶ 38]	Die Fehlerliste kann für das Debugging und die Diagnose sehr hilfreich sein
Zugriff auf Fenster-Registerkarten in Visual Studio [▶ 37]	Beschreibt den Zugriff auf Visual Studio Fenster.
Umgang mit verschiedenen Versionen von Visual Studio [▶ 30]	Beschreibt die Verwendung verschiedener Versionen von Visual Studio für das Automation Interface.
Anbindung an laufende Visual Studio-Instanzen [▶ 35]	Beschreibt, wie Sie sich mit bestehenden (bereits laufenden) Visual Studio-Instanzen verbinden können, um das Automation Interface zu verwenden.
TwinCAT Zielplattform einstellen [▶ 34]	Beschreibt die Einstellung der TwinCAT-Zielplattform zur Kompilierung.

Darüber hinaus enthält diese Dokumentation eine vollständige API-Referenz [▶ 117] aller Schnittstellen. In den Abschnitten Wie... und Beispiel [▶ 182] finden Sie eine lose Zusammenstellung von Skriptcodefragmenten, Konfigurationsschritten und Beispielprojekten. Sie umfassen zudem eine unsortierte und wachsende Liste von „realen“ Beispielen.

2.2 Versionsübersicht

Die folgende Tabelle gibt einen Überblick über die vorhandenen Eigenschaften des Automation Interface in Bezug auf TwinCAT 2.11, TwinCAT 3.0, TwinCAT 3.1 und einen Ausblick für zukünftige TwinCAT Versionen. Beachten Sie, dass insbesondere die Angaben bezüglich zukünftiger TwinCAT-Versionen Änderungen unterworfen sein können.

Eigenschaft	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Zukünftige Versionen
Allgemeine Einstellungen				
Konfigurationsvorlagen importieren	✓	✓	✓	✓
TwinCAT System Service Handling (Run-/Config-Modus)	✓	✓	✓	✓
Konfigurationen laden/speichern/ erstellen/aktivieren	✓	✓	✓	✓
Unterstützung für Remote TwinCAT-Ziele	✓	✓	✓	✓
Tasks mit Prozessabbild konfigurieren	✓	✓	✓	✓
Tasks ohne Prozessabbild konfigurieren	-	-	✓	✓
Mehrkernunterstützung für Tasks	-	✓	✓	✓
Umgang mit TwinCAT-Lizenzen	-	-	-	✓
Route-Management				
ADS-Routen hinzufügen/ entfernen	✓	✓	✓	✓
Broadcast-Suche	✓	✓	✓	✓
I/O				
Nach Online-Geräten suchen	✓	✓	✓	✓
Geräte, Boxen und Klemmen hinzufügen/ entfernen	✓	✓	✓	✓
Geräte, Boxen und Klemmen parametrisieren	✓	✓	✓	✓
EtherCAT-Topologien	✓	✓	✓	✓
Netzwerkvariablen	✓	✓	✓	✓
SPS				
Variablen mappen, z.B. mit I/Os oder Achsen	✓	✓	✓	✓

Eigenschaft	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Zukünftige Versionen
SPS-Projekte hinzufügen/ entfernen	✓	✓	✓	✓
SPS-POUs, DUTs, GVLs hinzufügen/ entfernen	-	-	✓	✓
SPS-Code von POUs, DUTs, GVLs abrufen/ setzen	-	-	✓	✓
SPS-Bibliotheken hinzufügen/ entfernen	-	-	✓	✓
SPS-Platzhalter hinzufügen/ entfernen	-	-	✓	✓
SPS-Repositories hinzufügen/ entfernen	-	-	✓	✓
SPS-Bibliotheken in/aus Repositories hinzufügen/ entfernen	-	-	✓	✓
SPS-Projekte als SPS-Bibliothek speichern	-	-	✓	✓
Compiler und Fehlerbehandlung	-	-	✓	✓
PLCopen XML Import/Export	-	-	✓	✓
Programmiersprache: Strukturierter Text (ST)	-	-	✓ 2	✓ 2
Programmiersprache: Ablaufsprache (AS)	-	-	✓ 1	✓ 1
C++				
C++-Projektvorlagen hinzufügen/ entfernen	-	-	-	✓
Compiler und Fehlerbehandlung	-	-	-	✓
Motion				
NC-Tasks hinzufügen/ entfernen	-	-	✓	✓
Achsen hinzufügen/ entfernen	-	-	✓	✓
Achseinstellungen parametrisieren	-	-	✓ 3	✓ 3

Eigenschaft	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	Zukünftige Versionen
Variablen mappen, z.B. mit SPS	-	-	✓	✓
TcCOM-Module				
TcCOM-Module hinzufügen/ entfernen	-	-	✓	✓
TcCOM-Module parametrisieren	-	-	✓	✓
Measurement				
Hinzufügen/ Entfernen von TwinCAT Measurement Projekten	-	-	✓	✓
Hinzufügen/ Entfernen von Charts	-	-	✓	✓
Hinzufügen/ Entfernen von Axes	-	-	✓	✓
Hinzufügen/ Entfernen von Kanälen	-	-	✓	✓
Parametrisierung von Charts, Axes und Kanälen	-	-	✓	✓
Starten/Stoppen von Aufnahmen	-	-	✓	✓

Anmerkungen	
[1]	Implementierung über PLCopen XML möglich.
[2]	Quellcode kann entweder als Klartext oder als PLCopen XML implementiert werden.
[3]	Mit Einschränkungen möglich: Einige Einstellungen sind in binärem Format gespeichert und können nicht bearbeitet werden.

2.3 Häufig gestellte Fragen

- **Was ist das TwinCAT Automation Interface?**

Das TwinCAT Automation Interface ist eine Schnittstelle um von einer externen Anwendung auf die Konfiguration von TwinCAT zuzugreifen. So kann der Kunde die Konfiguration von TwinCAT automatisieren.

- **Kann ich eine offline TwinCAT-Konfiguration erstellen (ohne angeschlossenes Gerät)?**

Ja. Sie können eine TwinCAT-Konfiguration offline erstellen, indem Sie alle Geräte manuell verbinden (ohne „;ScanDevices“;) und anschließend die Werte, z.B. Adressen, online zur Verfügung stellen, nachdem alle Geräte verbunden wurden. Siehe unsere Seite Beispiele [▶ 182] für weitere Informationen. Dort finden Sie auch ein "Wie..."-Beispiel [▶ 95] das Ihnen zeigt, wie Sie Adressinformationen für vorkonfigurierte E/A-Geräte bereitstellen können.

- **Welche Programmier- und Scriptsprachen werden unterstützt?**

Jede Programmier- oder Scriptsprache, die das COM-Objektmodell unterstützt, wird unterstützt. Siehe unsere Seite Systemanforderungen [▶ 17] für weitere Informationen.

- **Welche TwinCAT-Einstellungen sind über ein Automation Interface zugänglich?**

Siehe unsere Versionsübersicht [▶ 12] Seite für weitere Informationen über die TwinCAT-Einstellungen, die über das Automation Interface zugänglich sind.

- **Was, wenn ich keine geeignete Programmiermethode oder Eigenschaft für eine spezifische Einstellung finde?**

Wenn Sie keine geeignete Automation Interfacemethode oder -eigenschaft für eine spezifische Einstellung finden, können Sie die XML-Import/Export-Funktion von TwinCAT benutzen, um diese Einstellung zu lesen/schreiben. Siehe unseren Artikel über Benutzerdefinierte Strukturelementparameter [▶ 25] für weitere Informationen.

- **Kann ich die Konfiguration von TwinCAT SPS automatisieren?**

Ja. Die Eigenschaft wird mit TwinCAT 3.1 verfügbar sein. Siehe unsere Seite Versionsübersicht [▶ 12] für weitere Informationen.

- **Kann ich Automation Interface-Code auf TwinCAT XAR (nur Laufzeit) Computer ausführen?**

Nein. Um Automation Interface-Code ausführen zu können, wird TwinCAT XAE (Engineering) benötigt, weil das Automation Interface direkt auf das Visual Studio COM-Objekt zwecks Kommunikation mit der TwinCAT-Konfiguration zugreift. Allerdings können Sie einen TwinCAT XAE Computer benutzen, um auf eine TwinCAT-Laufzeit per Fernzugriff zuzugreifen und sie zu konfigurieren.

- **Wann sollte ich ADS und wann das Automation Interface verwenden?**

Diese Frage ist nicht leicht zu beantworten, weil die Antwort in erheblichem Maße davon abhängt, was Sie erreichen wollen. Das TwinCAT-Automation Interface wurde hauptsächlich dazu entwickelt, den Kunden dabei zu helfen, die Konfiguration von TwinCAT zu automatisieren. Wenn Sie regelmäßig IO-Werte oder SPS-Variablen lesen bzw. in sie schreiben möchten, dann sind unsere ADS APIs möglicherweise besser geeignet.

- **Muss ich meinen Automation Interface-Code anpassen, wenn ich zwischen den Sprachen in TwinCAT XAE, z.B. vom Englischen ins Deutsche wechsele?**

Alle TwinCAT XAE Elemente, die sprachenabhängig sind (Geräte, Boxen, Achsen, Kanäle, ...), können sowohl über den Namen in der aktuell eingestellten XAE-Sprache als auch über deren englischen Namen erreicht werden. Wenn die XAE-Sprache z.B. vom Englischen zum Deutschen wechselt, dann wird der Begriff „Channel“ in XAE als „Kanal“ eingeblendet, ist aber immer noch unter dem Namen „Channel“ über das Automation Interface verfügbar. Für eine uneingeschränkte Kompatibilität empfehlen wir Ihnen, Ihren Automation Interface-Code auf der Grundlage der englischen Terminologie zu erstellen.

Bitte beachten: Diese Eigenschaft wird erst mit TwinCAT 3.x verfügbar sein! Die auf TwinCAT 2.x basierenden Systeme sind nicht sprachenunabhängig!

- **Ich bin Maschinenbauer und verwende eine TwinCAT-Konfigurationsvorlage für alle Maschinentypen und aktiviere/deaktiviere lediglich bestimmte I/Os. Kann ich das gleiche mit dem Automation Interface tun?**

Ja. Es gibt ein "Wie..."-Beispiel [▶ 96] das Ihnen genau erklärt was zu tun ist.

- **Kann ich auch ADS-Routen erstellen oder ein Broadcast Search durchführen?**

Ja. Siehe unsere Beispiele [▶ 182] und Wie...-Seiten für mehr Informationen.

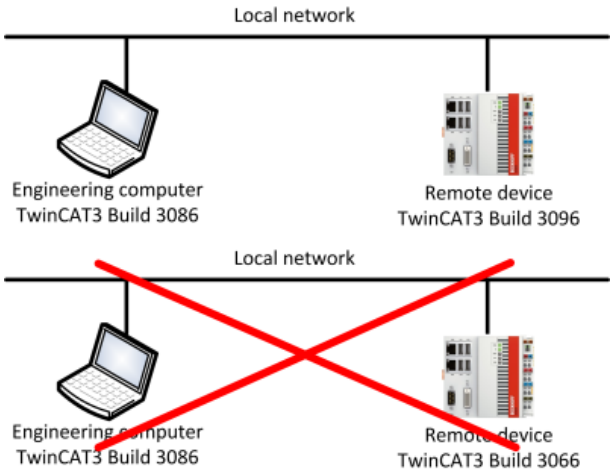
3 Installation

3.1 Systemanforderungen

Das folgende Kapitel listet alle Hardware- und Software-Anforderungen für das TwinCAT-Automation Interface auf und gibt einige Empfehlungen zu Programmier- und Scriptsprachen.

Hard- und Software

Das TwinCAT-Automation Interface wird automatisch bei der TwinCAT-Einrichtung installiert. Demzufolge gelten die gleichen Hardware- und Software-Anforderungen wie für TwinCAT System Manager / TwinCAT 3 XAE (Engineering). Wenn das Automation Interface für die Konfiguration eines Remote TwinCAT-Teilnehmers verwendet wird, ist es wichtig, dass die Remote-Version von TwinCAT gleich oder höher als diejenige auf dem Entwicklungsrechner ist.



Beachten Sie, dass Sie das Automation Interface-Skript auf 32-Bit- und 64-Bit Plattformen ausführen können, dass Sie aber darauf achten müssen, dass Ihr Programm/Skript für den 32-Bit-Modus kompiliert wurde und im 32-Bit-Modus läuft.

Empfohlene Programmiersprachen

Für die Verwendung mit dem TwinCAT-Automation Interface werden folgende Programmiersprachen empfohlen:

- .NET Sprachen, wie z. B. C# oder Visual Basic .NET

Bitte beachten: Auch wenn C++ Implementierungen funktionieren, empfehlen wir dringend den Gebrauch einer der oben aufgeführten Sprachen, aufgrund deren einfachen und direkten Programmierungskonzepte hinsichtlich COM. In dieser Dokumentation wird meistens C# in den Beispielen verwendet.

Empfohlene Scriptsprachen

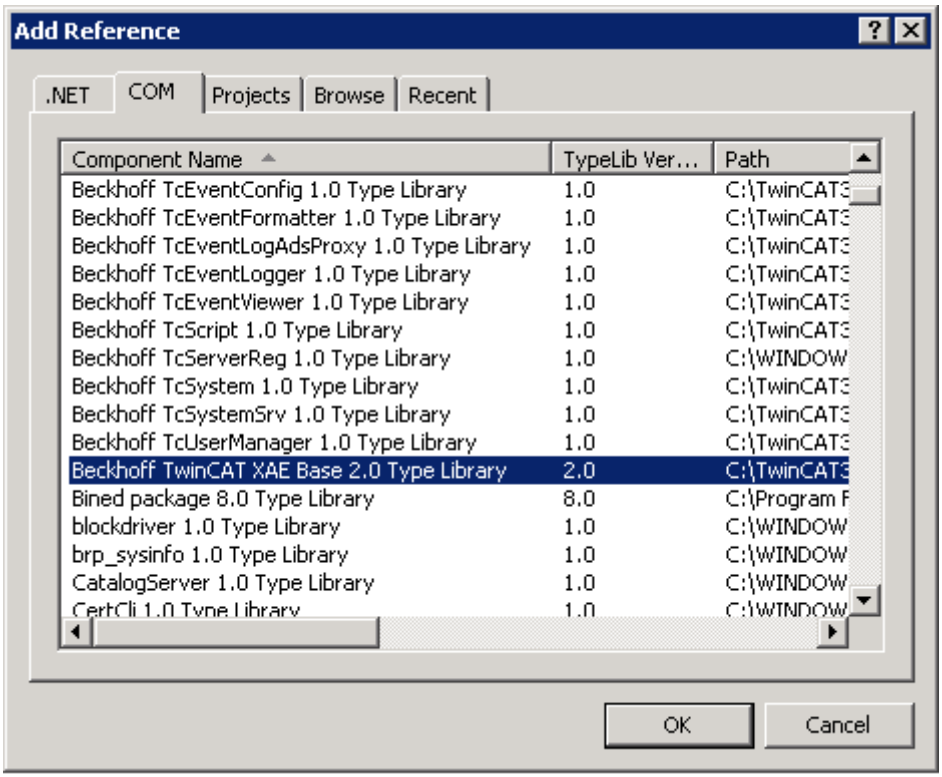
Wenngleich jede Skriptsprache mit COM-Support für den Zugriff auf TwinCAT Automation Interface verwendet werden kann, empfehlen wir die Verwendung von Windows Powershell, da es das höchste Integrationsniveau zwischen Betriebssystem und Anwendung bietet. Denken Sie daran, dass mindestens TwinCAT 3.1 Build 4020.0 erforderlich ist, um dynamische Sprachen wie Windows Powershell zu verwenden.

3.2 Installation

Alle für das TwinCAT-Automation Interface benötigten Dateien werden automatisch im Verlauf des TwinCAT-Setups installiert. Wie bereits in der Einleitung erwähnt, kommuniziert das Automation Interface über COM mit TwinCAT. Alle erforderlichen COM-Objekte werden automatisch konfiguriert, so dass COM-fähige Programmier- und Scriptsprachen auf diese Objekte zugreifen können.

Verwendung des Automation Interface innerhalb einer .NET-Anwendung (C#, VB.NET, ...)

Um auf das Automation Interface von einer .NET Anwendung aus zuzugreifen, müssen Sie eine Referenz zum entsprechenden COM-Objekt **Beckhoff TwinCAT XAE Base** (abhängig von Ihrer TwinCAT-Version, siehe Tabelle unten) im Visual Studio-Projekt hinzufügen.



Nachdem die Referenz hinzugefügt wurde, können Sie auf das COM-Objekt über den Namensraum **TCatSysManagerLib** zugreifen.

Bitte lesen Sie den Artikel Auf TwinCAT-Konfiguration zugreifen [▶ 20] worin alle weiteren Schritte ausführlich erläutert werden.

Verwendung des Automation Interface in Scriptsprachen

Das TwinCAT-Automation Interface kann ebenfalls mit COM-fähigen Scriptsprachen, wie z.B. Windows PowerShell oder IronPython verwendet werden. Da Scriptsprachen während der Laufzeit interpretiert und nicht vorher kompiliert werden, haben sie immer Zugriff auf alle im Betriebssystem aktuell registrierten COM-Objekte. Deswegen ist eine Referenz nicht erforderlich.

Bitte lesen Sie den Artikel Auf TwinCAT-Konfiguration zugreifen [▶ 20] worin alle weiteren Schritte ausführlich erläutert werden.

Typbibliotheksversionen

Im Verlauf des TwinCAT-Produktlebenszyklus kann die oben erwähnte Typbibliothek in unterschiedlichen Versionen geliefert werden, weil gegebenenfalls Funktionalitäten hinzugefügt und/oder größere TwinCAT-Versionsschritte vollzogen wurden. Die nachfolgende Tabelle gibt einen Überblick über alle unterschiedlichen Typbibliotheksversionen:

Typbibliotheksname	Typbibliotheksversion	TwinCAT-Version
Beckhoff TCatSysManager 1.1 Typbibliothek	1.1	TwinCAT 2,11
Beckhoff TwinCAT XAE Base 2.0 Typbibliothek	2.0	TwinCAT 3.0
Beckhoff TwinCAT XAE Base 2.1 Typbibliothek	2.1	TwinCAT 3.1
Beckhoff TwinCAT XAE Base 3.1 Typbibliothek	3.1	TwinCAT 3.1 Build 4020.0 und höher

4 Konfiguration

4.1 Quickstart

Da das TwinCAT Automation Interface eine Menge Möglichkeiten bietet, könnte es manchmal schwierig sein zu wissen, wo man anfangen soll. Dieser Schnelleinstieg bietet eine Schritt-für-Schritt-Einführung in das TwinCAT Automation Interface und die verschiedenen Artikel dieser Dokumentation.

Wir empfehlen, die folgenden ausführlichen Artikel zu lesen:

Schritt	Artikel	Inhalt
1	Visual Studio ProgIDs [▶ 30]	Beschreibt, wie man über Visual Studio API auf verschiedene Visual Studio Versionen zugreifen kann
2	Zugriff auf die TwinCAT-Konfiguration [▶ 20]	Beschreibt, wie man das Automation Interface verwenden kann, um ein neues TwinCAT Projekt zu erstellen. Es umfasst ebenfalls die Koexistenz von Visual Studio API und TwinCAT Automation Interface und wie diese miteinander korrelieren.
3	Navigation durch die TwinCAT-Konfiguration [▶ 23]	Beschreibt, wie man unter Verwendung verschiedener Automation Interface Funktionalitäten durch eine geöffnete TwinCAT-Konfiguration navigieren kann.
4	Die Notwendigkeit eines COM MessageFilter [▶ 27]	Beschreibt, warum jede Automation Interface Anwendung einen spezifischen COM MessageFilter implementieren sollte.
5	Stumm-Modus [▶ 31]	Beschreibt, wie man das Automation Interface „stumm“ stellt

Nach diesen grundlegenden Artikeln könnten die Best Practice [▶ 31] Artikel zu verschiedenen Themen konsultiert werden, wie z. B. SPS oder I/O-Zugriff via Automation Interface.

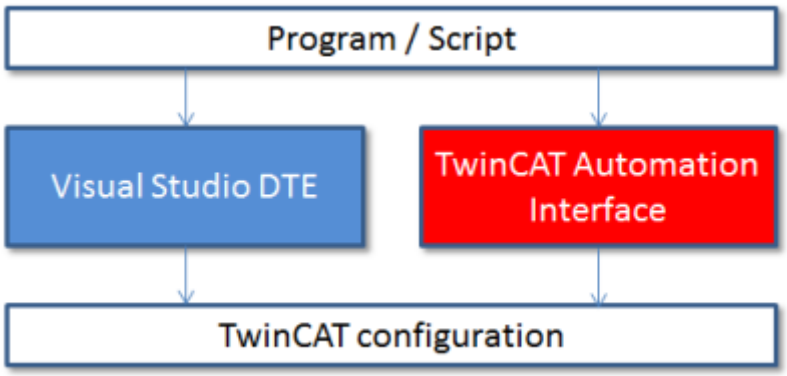
4.2 Grundlagen

4.2.1 Zugriff auf die TwinCAT-Konfiguration

In diesem Kapitel wird der Zugriff auf ein TwinCAT XAE Konfigurationsprojekt über Automation Interface beschrieben. Das Ziel dieses Erstellungsprozesses besteht darin, Zugriff auf ein TwinCAT XAE-Projekt (früher bekannt als eine TwinCAT System Manager-Konfiguration) zu erhalten. Das neue TwinCAT XAE-Projekt ist anspruchsvoller, als die aus TwinCAT2 bekannte TwinCAT System Manager-Konfiguration. Dies impliziert eine geringfügige Konzeptänderung im Umgang mit einem Projekt / einer Konfiguration. TwinCAT 3 unterstützt eine zusätzliche hierarchische Ebene, die mehrere Konfigurationen in einen Visual Studio Solution Container zusammenfasst. Dies kann z.B. dazu genutzt werden, die Konfigurationen von dezentralen Ressourcen in eine Solution zu organisieren oder für das Kombinieren von HMI-Projekten zusammen mit der Systemkonfiguration. Die Solution ist in der Lage, alle Typen von Visual Studio und/oder TwinCAT XAE-Projekten zusammenzufassen. Bei Verwendung des TwinCAT XAE- Automation Interface bedeutet dies ein zusätzliches Level an Möglichkeiten.

Grundlegende Informationen

TwinCAT 3 wurde vollständig in Visual Studio integriert, um den Benutzern einen standardisierten und möglichst flexiblen Editor für die Erstellung und Verwaltung von TwinCAT-Projekten an die Hand zu geben. Bei der Erstellung von und/oder dem Zugriff auf eine TwinCAT-Konfiguration können Visual Studio und das TwinCAT-Automation Interface im Verbund verwendet werden. Beispiel: Wenn Sie eine neue TwinCAT-Konfiguration über das Automation Interface erstellen möchten, müssen Sie zunächst Methoden der Visual Studio-API aufrufen, um einen Visual Studio Solution Container zu erstellen und anschließend ein TwinCAT-Projekt mit Hilfe der Methoden des TwinCAT-Automation Interface hinzufügen. Dieses Szenario wird in einigen Code-Ausschnitten weiter unten behandelt.



Darüber hinaus bietet die Visual Studio-API (die sogenannte **Visual Studio DTE**) den Entwicklern viele weitere Funktionen, z.B. den Zugriff auf das Fehlerausgabefenster [▶ 38]. Weitere Informationen zu Visual Studio DTE finden Sie auf der Microsoft MSDN Internetseite.

Bitte beachten:

- Bei der Erstellung eines neuen TwinCAT-Projekts in einer Visual Studio Solution müssen Sie einen Pfad zur TwinCAT-Projektvorlage [▶ 26] angeben. Bitte passen Sie diesen Pfad in den Code-Ausschnitten unten entsprechend Ihrer Umgebung an.
- Die folgenden Code-Ausschnitte verwenden **dynamisches Verknüpfen** für die Visual Studio DTE-Objekte, was bedeutet, dass der tatsächliche Typ des Objekts erst im Verlauf der Anwendungslaufzeit bestimmt wird. In dem Fall, in dem Sie das dynamische Verknüpfen nicht verwenden, sondern den Datentyp im Voraus festlegen möchten, müssen Sie den Namensraum EnvDTE.DTE in Ihr Projekt einbinden.

TwinCAT-Projekte über Vorlagen erstellen

Denken Sie daran, dass Sie eine Referenz zu dem COM-Objekt **TcatSysManagerLib** und EnvDTE.DTE (Microsoft Developement) hinzufügen, um in der Lage zu sein, das TwinCAT Automation Interface und die Visual Studio API zu verwenden. Die ProgID, die in der GetTypeFromProdID() Methode verwendet wird, hängt von der Visual Studio Version ab, die verwendet werden soll. Schauen Sie sich diesen [▶ 30] Artikel an, um weitere Informationen über die verschiedenen ProgIDs zu erhalten.

Code-Ausschnitt (C#):

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

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

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

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

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

ITcSysManager sysManager = project.Object;

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

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

Code-Ausschnitt (Powershell):

Sie können den folgenden Code-Ausschnitt kopieren, in eine Textdatei einfügen und Letztere als "someName.ps1" speichern. Anschließend können Sie diese direkt über Windows PowerShell ausführen.

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

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

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

New-Item $targetDir -type directory

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

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

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

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

Code-Ausschnitt (C++):

Im entsprechender Header-Datei (z.B. stdafx.h):

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

Aufgrund eines bekannten Problems in VisualStudio 2010 (SP1) wird der generierte Proxy-Code nicht in das C++ Projekt eingebunden. Verwenden Sie bitte die in #import Known Issue (bekanntes Problem importieren) beschriebene provisorische Lösung.

```
#include

using namespace std
using namespace TCatSysManagerLib;
using namespace EnvDTE;

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

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

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

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

    // create the solution
```

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

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

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

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

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

Bekanntes Problem importieren:

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

Bitte beachten:

ITcSysManager::NewConfiguration [▶ 120], ITcSysManager::OpenConfiguration [▶ 120] und ITcSysManager::SaveConfiguration [▶ 121] werden in diesem Fall Fehlermeldungen verursachen, weil die Projektverwaltung an das Visual Studio IDE (die vom DTE-Objekt realisierten Instanzen von Solution und Projekt) delegiert wurde.

4.2.2 TwinCAT-Konfiguration durchsuchen

Wir haben bereits in einem getrennten Artikel beschrieben, wie man über das Visual Studio-Automatisierungsmodell auf TwinCAT zugreifen [▶ 20] kann. Diese Referenz auf TwinCAT wird mit Hilfe eines Objekts vom Typ ITcSysManager [▶ 118] repräsentiert. Wir würden im Folgenden gerne besprechen, wie Sie durch die TwinCAT-Konfiguration navigieren können.

Allgemeine Informationen

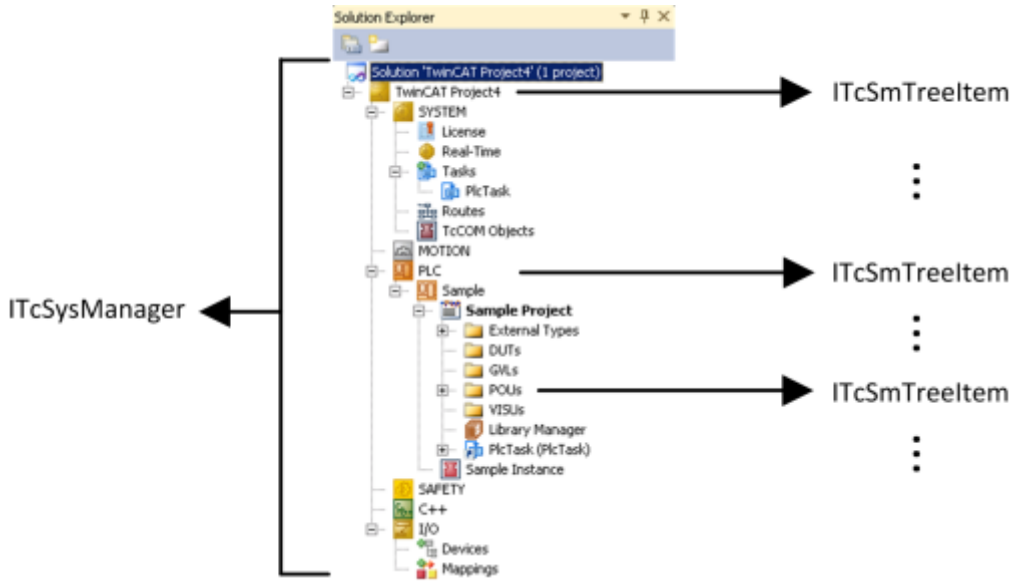
Es ist wichtig zu verstehen, dass die gesamte Information in TwinCAT in einer baumähnlichen Struktur geordnet ist. Im Automation Interface wird jeder Strukturknoten und demzufolge jedes Element einer TwinCAT-Konfiguration durch die ITcSmTreeItem [▶ 127]-Schnittstelle dargestellt.

Es kann auf verschiedenen Wegen durch das TwinCAT-Datenmodell navigiert werden, und zwar abhängig davon, welche Art Information Sie abfragen möchten.

- **Lookup-Methoden** suchen nach spezifischen Tree Items über spezifizierte Suchkriterien, z.B. der Pfad zu einem Tree Item

- **Iteratoren** oder Suchfunktionen iterieren über einen Satz von abgefragten Tree Items

Beide Methoden werden nun im folgenden Artikel besprochen.



LookupMethods

Die Lookup-Methoden betreffen immer das gesamte Datenmodell (ungefiltert).

- ITcSysManager::LookupTreeItem [▶ 124] bestimmt ein Tree Item mit dem angegebenen vollständigen Pfadnamen.
- ITcSysManager3::LookupTreeItemById [▶ 125] bestimmt ein Tree Item mit angegebenem Elementtyp und Element-Id.
- ITcSmTreeItem::LookupChild [▶ 164] bestimmt ein Tree Item innerhalb eines Unterbaums, der mit relativem Pfandnamen spezifiziert ist.

In TwinCAT kann jedes Tree Item mit Hilfe seines eindeutigen Pfadnamens identifiziert werden. Der Pfadname eines Tree Item basiert auf der hierarchischen Ordnung seines übergeordneten Elements (dessen Name) und seinem eigenen Namen, getrennt durch Zirkumflexe ('^'). Um die Pfadnamen zu verkürzen und Sprachabhängigkeiten zu vermeiden, haben die höchstrangigen Tree Items spezielle Kurzformen, die in ITcSysManager::LookupTreeItem [▶ 124] aufgelistet sind.

Iteratoren

Derzeit werden drei verschiedene Arten von Iterationsfunktionen unterstützt.

- Alle Tree items durchsuchen (ungefiltert)
- Haupttree items durchsuchen
- Nur Variablen / Symbole durchsuchen

Alle Tree Items durchsuchen (ungefiltert)

Zum ungefilterten Durchsuchen aller Tree Items kann die Eigenschaft ITcSmTreeItem [▶ 127]:NewEnum verwendet werden. _NewEnum iteriert über alle Unterknoten des aktuell referenzierten ITcSmTreeItem [▶ 127]. Diese (COM-) Eigenschaft wird von vielen Programmier- und Scriptsprachen verwendet, die das COM Enumerator-Modell (z.B. .NET-Sprachen, VB6 oder Scriptsprachen) durch eine 'foreach'-Anweisung unterstützen. Bei Sprachen, die das nicht unterstützen, wie C++, muss die foreach-Schleife manuell implementiert werden unter Verwendung der IEnumVariant-Schnittstelle.

Wir empfehlen diese Vorgehensweise bei der Iteration durch untergeordnete Knoten.

Beispiel (C#):

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

Beispiel (C++):

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

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

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

Beispiel (PowerShell):

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

Haupttree items durchsuchen (gefiltert)

Zum ausschließlichen Durchsuchen der untergeordneten Hauptelemente des aktuellen Tree items verwenden Sie bitte das Eigenschaftenpaar ITcSmTreeItem::ChildCount [▶ 127] und ITcSmTreeItem:Child(n) [▶ 118]. Diese Methoden wirken nur auf die direkten untergeordneten Elemente (nicht rekursiv).

Nur Variablen / Symbole durchsuchen

Zum Durchsuchen der Variablen / Symbole benutzen Sie das Eigenschaftenpaar ITcSmTreeItem::VarCount(x) [▶ 127] , ITcSmTreeItem::Var(x,n) [▶ 127]. Der Variablentyp (Ein- oder Ausgangsvariablen) kann über Parameter festgelegt werden.

4.2.3 Benutzerdefinierte TreeItem-Parameter

Die ITcSmTreeItem [▶ 127]-Schnittstelle wird von jedem TwinCAT-Tree Item unterstützt und zeichnet sich durch einen sehr allgemeinen Charakter aus. Um die Spezifikation aller Geräte, Boxen und Klemmen, zusammen mit vielen anderen verschiedenen Tree Item-Typen zu unterstützen, sind alle benutzerdefinierten

Parameter eines Tree Items über die XML-Darstellung des Tree Items zugänglich. Auf die XML-Zeichenkette kann mit der ITcSmTreeItem::ProduceXml [▶ 158]-Methode und ihrem Gegenpart ITcSmTreeItem::ConsumeXml [▶ 159] zugegriffen werden. Dieses Funktionenpaar hat die gleiche Funktionalität wie die Befehle "XML-Beschreibung exportieren ..." und "XML-Beschreibung importieren ..." im Hauptmenü von TwinCAT IDE (siehe Screenshot unten).

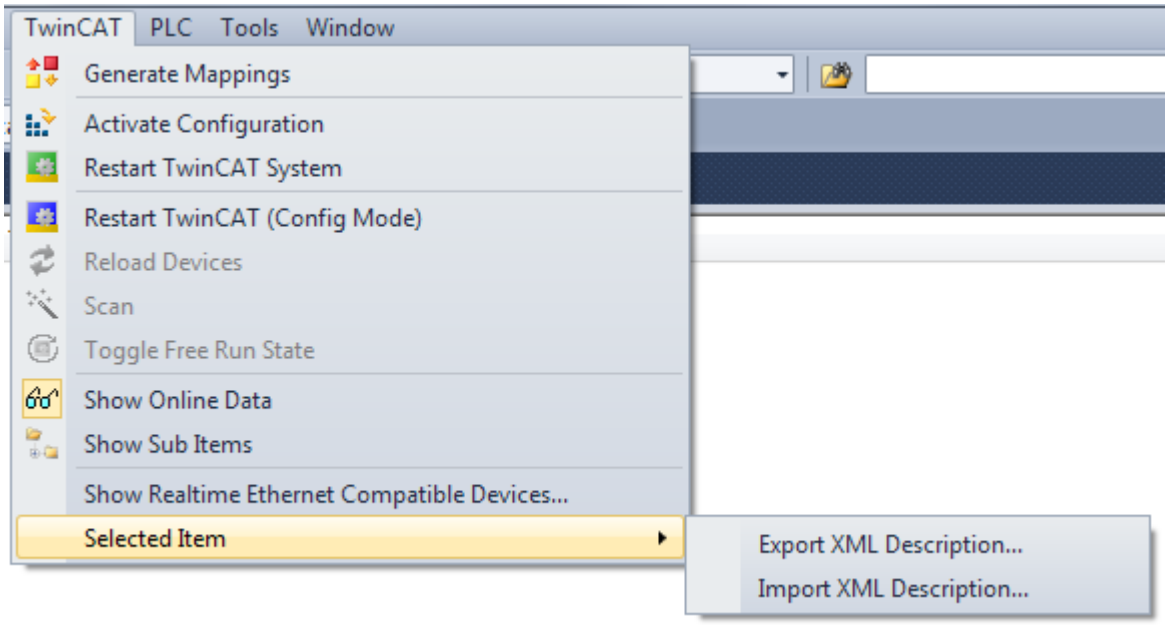


Abb. 1: TcSysMan_AutomationXmlParameters

Mit Hilfe dieses Import-/Exportfunktionensatzes können viele Abschnitte von einem Skript- oder Automatisierungscode bequem getestet und maßgeschneidert werden, bevor sie in der Codiersprache entwickelt werden. Es genügt die Tree Item-Daten zu exportieren, den Inhalt zu bearbeiten und dann wieder zu importieren.

Die beste Praxis besteht darin, den XML-Inhalt zunächst zu exportieren, den Inhalt zu bearbeiten, dann in die IDE zu importieren und dann, wenn alles funktioniert, in den Programmier-/Skriptcode für die Handhabung mit den ProduceXml und ConsumeXml-Methoden zu packen.

4.2.4 TwinCAT-Projektvorlage

Bei der Erstellung einer neuen TwinCAT Solution müssen Sie den Pfad zur TwinCAT-Projektvorlage angeben. Siehe auch unseren Artikel Auf TwinCAT XAE-Konfiguration zugreifen [▶ 20].

Grundlagen

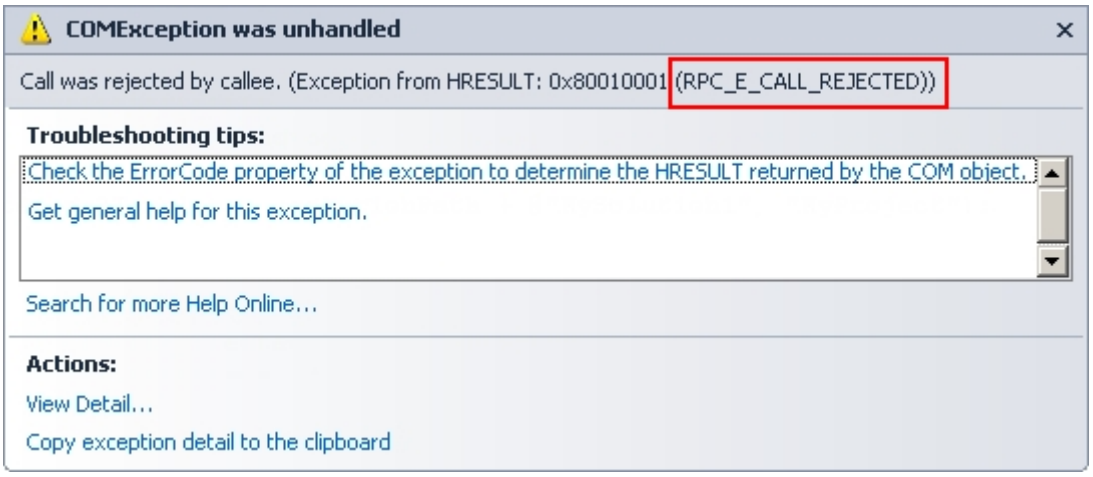
TwinCAT-Version	Pfad zur TwinCAT-Projektvorlage*
TwinCAT 3.0	C:\TwinCAT\3.0\Components\Base\PrjTemplate\TwinCAT Project.tsp
TwinCAT 3.1	C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj

* Bitte beachten: Die oben angegebenen Pfade basieren auf einem standardmäßiges TwinCAT Installierungsverzeichnis und können unterschiedlich sein, wenn Sie TwinCAT in einen anderen Ordner installiert haben.

4.2.5 Implementierung eines COM-Nachrichtenfilters

Nachrichtenfilterung ist ein Mechanismus, mit dessen Hilfe Serveranwendungen entscheiden können, ob und wann ein eingehender Methodenaufruf sicher auf eines ihrer Objekte ausgeführt werden kann. COM kennt im Allgemeinen die Anforderungen bezüglich der Eintrittsinvarianz Ihrer Anwendung nicht und filtert demzufolge standardmäßig nicht die Nachrichten. Auch wenn die Nachrichtenfilterung nicht mehr so bedeutend ist, wie sie es mit 16-Bit-Anwendungen war, weil die Größe der Nachrichtenwarteschlange jetzt eigentlich unbegrenzt ist, sollten Sie immer noch **Nachrichtenfilterung** als Möglichkeit zur Lösung von Blockaden implementieren. COM wird Ihre Implementierung der **IMessageFilter**-Schnittstelle aufrufen, um herauszufinden, ob eine Anwendung (ein COM Server) blockiert, so dass Sie reagieren und die Situation behandeln können. Zum Beispiel, beim Zugriff auf TwinCAT XAE über COM wird die Visual Studio-Instanz weitere COM-Aufrufe abweisen, während noch ein früherer COM-Aufruf ausgeführt wird. Die Folge ist, dass die Client-Anwendung einen RPC_E_CALL_REJECTED-Fehler ausgibt und, ohne weitere Intervention, den Aufruf nicht wiederholen wird. Durch das Verfassen eines benutzerdefinierten Nachrichtenfilters hat der Programmierer die Möglichkeit, den COM-Aufruf zu wiederholen, wenn die Client-Anwendung die Notifizierung eines verweigerten COM-Aufrufs durch den COM-Server erhält.

Der folgende Screenshot zeigt eine typische Fehlerausgabe durch den Visual Studio COM-Server, wenn eine Instanz immer noch mit der Ausführung eines vorherigen COM-Aufrufs beschäftigt ist.



Um diese Situation zu vermeiden und einen Nachrichtenfilter zu implementierten, der auf diesen abgewiesenen COM-Aufruf reagiert, muss der Anwendungsingenieur die **IMessageFilter**-Schnittstelle implementieren. Diese Schnittstelle besteht aus drei Methoden:

- **HandleIncomingCall(): Stellt einen einzigen Einsprungpunkt für eingehende Aufrufe bereit**
- **MessagePending(): Zeigt an, dass eine Nachricht eingegangen ist, während COM auf die Beantwortung eines Fernaufrufs wartet.**
- **RetryRejectedCall(): Bietet die Möglichkeit auf einen abgewiesenen COM-Aufruf zu reagieren.**

Beachten Sie, dass Nachrichtenfilter nur auf STA-Threads angewendet werden können und dass nur ein Filter auf jeden Thread angewendet werden kann. Multithreaded Apartments, z.B. Konsolenanwendungen, können keine Nachrichtenfilter haben. Diese Anwendungen müssen in einem STA-Thread laufen, um Nachrichtenfilterung anzuwenden. Im Anhang dieser Dokumentation finden Sie weitere Informationen zum COM-Threading.

Der folgende Code-Ausschnitt zeigt ein Beispiel, wie die IMessageFilter-Schnittstelle in C# verwendet werden kann. Beachten Sie, dass dieser Code auch in vielen Beispielen in unserem Abschnitt Beispiele [▶ 182] verwendet wird, und auch als getrennter Beispiel-Download zur Verfügung steht.

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

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


[PreserveSig]
```

```
int RetryRejectedCall(IntPtr hTaskCallee, int dwTickCount, int dwRejectType);


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

Die folgende Klasse implementiert diese Schnittstelle und fügt zwei weitere Methoden hinzu: Register() und Revoke().

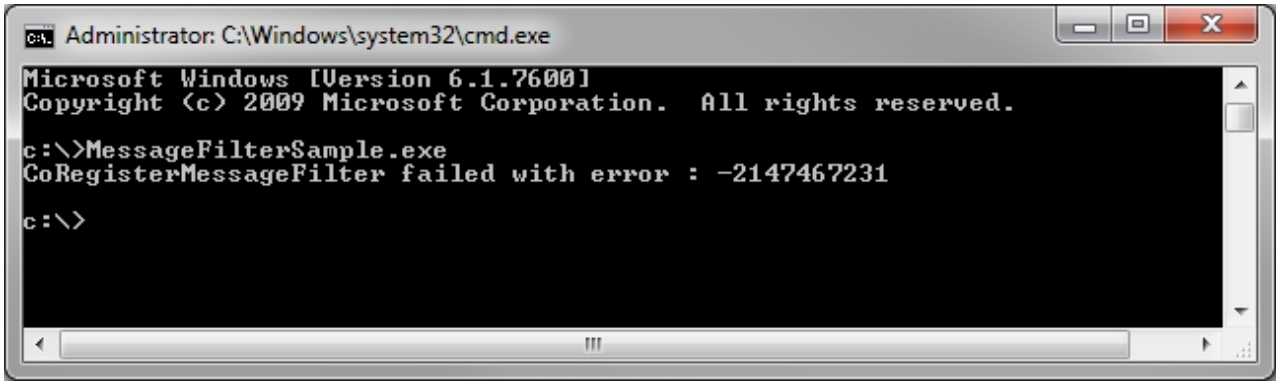
```
public class MessageFilter : IOleMessageFilter
{
public static void Register()
{
IOleMessageFilter newFilter = newMessageFilter();
IOleMessageFilter oldFilter = null;
int test = CoRegisterMessageFilter(newFilter, out oldFilter);

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


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


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


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


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

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

Ein Anwendungsingenieur muss nun lediglich die Methoden Register() und Revoke() aus einer anderen Klasse aufrufen, um den MessageFilter zu initialisieren und zu verwerfen. Die Folge ist, dass abgewiesene COM-Aufrufe wie in der RetryRejectedCall()-Methode festgelegt wiederholt werden.

Der folgende Code-Ausschnitt zeigt, wie diese Methoden in einer in C# geschriebenen Konsolenanwendung aufzurufen sind. Wie oben erwähnt laufen Konsolenanwendungen standardmäßig in einem MTA-Thread. Aus diesem Grund muss die Main()-Methode für die Ausführung in einem STA-Apartment konfiguriert werden, damit der Nachrichtenfilter angewendet werden kann.

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

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

MessageFilter.Revoke();
}
```

Wenn Sie versuchen, einen Nachrichtenfilter auf eine im MTA-Apartment laufende Anwendung anzuwenden, wird der folgende Fehler ausgegeben, wenn versucht wird, die Methode CoRegisterMessageFilter() während der Laufzeit auszuführen:



Weitere Informationen über die verschiedenen COM Threading Modelle finden Sie im MSDN-Artikel Understanding and Using COM Threading models. Weitere ausführliche Informationen über die IMessageFilter-Schnittstelle finden Sie in der MSDN-Dokumentation bezüglich IMessageFilter.

Der folgende Code-Ausschnitt zeigt, wie ein COM MessageFilter für Windows Powershell implementiert wird, indem dieser als .NET-Typ (C#) in PowerShell eingebunden wird.

Code-Ausschnitt (Powershell):

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

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

[EnvDTEUtils.MessageFilter]::Revoke()


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

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

public static void Revoke()
{
```

```
IOleMessageFilter oldFilter = null;
CoRegisterMessageFilter(null, out oldFilter);
}

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

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

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

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

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

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

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

4.2.6 Umgang mit verschiedenen Versionen von Visual Studio

Dieser Artikel erläutert die Verwendung des TwinCAT-Automation Interface mit unterschiedlichen Versionen von Visual Studio, z.B. Visual Studio 2010 und 2012, wenn Sie beide oder mehrere Versionen gleichzeitig installiert haben. Er besteht aus folgenden Themen:

- Visual Studio Programm-ID
- Visual Studio-Version im Code des Automation Interface spezifizieren

Visual Studio Programm-ID

Wie Sie bereits in unseren grundlegenden Artikeln gesehen haben, müssen Sie aufgrund der Integration von TwinCAT 3 in die Visual Studio-Umgebung ein Visual Studio DTE-Objekt erstellen, bevor Sie den Code für das Automation Interface implementieren können. Wenn Sie mehr als eine Visual Studio-Version auf Ihrem Engineering-PC installiert haben und gerne die Version auswählen möchten, die mit dem Automation Interface-Code zu verwenden ist, müssen Sie die sogenannte Programm-ID (ProgID) der entsprechenden Visual Studio-Version bestimmen. Die ProgID befindet sich in der Windows-Registrierung unter HKEY_CLASSES_ROOT und weist folgendes Format auf: VisualStudio.DTE.X.Y.

Die folgende Tabelle zeigt die ProgIDs der derzeit unterstützten Visual Studio-Versionen:

Version	ProgID
Visual Studio 2010	VisualStudio.DTE.10.0
Visual Studio 2012	VisualStudio.DTE.11.0
Visual Studio 2013	VisualStudio.DTE.12.0
Visual Studio 2015	VisualStudio.DTE.14.0
Visual Studio 2017	VisualStudio.DTE.15.0
TwinCAT XAE Shell	TcXaeShell.DTE.15.0

Visual Studio-Version im Code des Automation Interface spezifizieren

Zwecks Spezifizierung der Visual Studio-Version im Code des Automation Interface müssen Sie die ProgID als Parameter bei der GetTypeDromProgID()-Methode bei der Erstellung des DTE-Objekts verwenden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.2.7 Stumm-Modus

Manchmal sollte ein Skript oder Programm des Automation Interface stumm laufen, was bedeutet, ohne Message Boxen oder andere sichtbare Unterbrechungen. Wenngleich der Visual Studio DTE Befehl „dte.Visible = true/false“ in den meisten Fällen ausreichend erscheint, führt das TwinCAT Automation Interface einen neuen Schalter für den Stumm-Modus ein, der ab TwinCAT 3.1 Build 4020.0 und höher verfügbar ist.

Dieser neue Schalter kann wie folgt aktiviert werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Dies unterdrückt die Dialoge mittels Message Box während der Nutzung des TwinCAT Automation Interface.

4.3 Bewährtes Vorgehen

4.3.1 Visual Studio

4.3.1.1 Projekte zum Build-Prozess auswählen

Die Visual Studio API bietet alle erforderlichen Mechanismen zur Auswahl von Projekten für eine „Solution Configuration“. Die erforderlichen Methoden sind Bestandteil der SolutionBuild2 Klasse des EnvDTE Namensraums. Das folgende Beispiel zeigt, wie die Methode BuildProject() von dieser Klasse zu verwenden ist.

Code-Ausschnitt (Powershell):

```
$sln = $dte.Solution
$prj = $dte.Projects.Item(1) #SysMan Project
$sysManProjectName = $prj.FullName
```

```
$plcPrjProjectName = PathToPlcProjFile
$sln.SolutionBuild.BuildProject("Release|TwinCAT RT (x64)",$sysManProjectName,$true)
$sln.SolutionBuild.BuildProject("Release|TwinCAT RT (x64)",$plcPrjProjectName,$true)
```

Der Platzhalter „pathToPlcProjFile“ repräsentiert den vollständigen Pfad zu einer *.plcproj Datei, die ein TwinCAT 3 SPS-Projekt darstellt.

4.3.1.2 Zugriff auf TeamFoundationServer Source-Control

Dieses Kapitel bietet einen Überblick darüber, wie man auf einen Microsoft Team Foundation Server (TFS) programmgesteuert zugreifen kann, in dem die entsprechenden DLLs verwendet werden, die durch Visual Studio bereitgestellt werden. Diese Dokumentation wurde für einen besseren Überblick geschrieben und um die ersten Schritte einfacher zu gestalten, wenn Sie versuchen, den TwinCAT Automation Interface-Code mit TFS DLLs zu kombinieren. Für eine detailliertere Dokumentation über die TFS API empfehlen wir, die entsprechenden MSDN Artikel aufzurufen.

Dies Dokumentation befasst sich mit den folgenden Themen:

- Verbindung zu einem TFS Server
- Verbindung zu TeamProjects
- Arbeit mit Arbeitsplätzen
- Erstellung von Arbeitsordnern
- Neueste Versionen erhalten
- Eine Liste von anstehenden Änderungen erhalten
- Ein- und Auschecken
- Vorgang rückgängig machen

Die folgenden TFS API DLLs werden innerhalb dieser Dokumentation verwendet:

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

Verbindung zu einem TFS Server

Die Microsoft TFS API ermöglicht einen Anschluss Ihrer Anwendung, die den TwinCAT Code des Automation Interface enthält, an Ihr Entwicklungs-Repository auf einem Remote-Server. Die Verbindung erfordert einen String, der die Adresse, den Port und die Zusammenstellung Ihres Remote-Servers beschreibt, z. B. http://your-tfs:8080/tfs/DefaultCollection.

Vor der Verbindung müssen Sie eine Instanz der Uri-Klasse und der TfsConfigurationServer-Klasse definieren. Weisen Sie den Remote-Server-Link dem Konfigurationsserver zu:

Code-Ausschnitt (C#):

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

Verbindung zu TeamProjects

Sie können eine Liste der Teamprojekte oder eines spezifischen Teamprojekts auf dem Server erhalten, indem Sie die folgenden Code-Ausschnitte ausführen:

Code-Ausschnitt (C#):

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

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

Arbeit mit Arbeitsplätzen

Team Foundation arbeitet mit Arbeitsplätzen, die Mappings zu Arbeitsordnern enthalten. Diese Mappings verknüpfen serverseitige Ordner mit lokalen Ordnern auf Ihrer Festplatte. Zudem werden der Name des Inhabers und der Name Ihres Computers als Bestandteil des Arbeitsplatznamens gespeichert. Der Arbeitsplatz ist ein Muss, bevor eine Versionssteuerungsaufgabe ausgeführt wird, da er die Informationen über Dateien, Versionen und die Liste der anstehenden Änderungen speichert.

Zum Start mit TFS Client bietet die API eine *Workspace* Klasse, welche die vorstehend angeführten Informationen speichert und einen umfangreichen Satz an Methoden anbietet, um mit den Dateien und Ordnern zu interagieren. Angenommen, Sie möchten einen Arbeitsplatz für einen Ordner mit dem Namen *folderName* erstellen, erstellen Sie einen String, der den Computername und den Ordner enthält:

Code-Ausschnitt (C#):

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

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

Es kann nun ein Mapping zwischen dem serverseitigen Ordner und Ihrem lokalen nach *folderName* spezifizierten Ordner unter dem *Workspace* geben. Zum Erhalt oder zum Löschen eines bestehenden Arbeitsplatzes führen Sie einfach die folgenden Methoden aus:

Code-Ausschnitt (C#):

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

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

Erstellung von Arbeitsordnern

Zur Verbindung mit einem spezifizierten Projektordner benötigen Sie einen relativen Pfad im Hinblick auf den Stamm.

Zum Beispiel, wenn das Projekt gespeichert ist auf: *your-server\Development\TcSampleProjectX*, dann ist der relative Pfad zu diesem Ordner *$\Development\TcSampleProjectX.*

Zur Angleichung dieses Ordners an die Projekte auf dem Server können Sie über einen Satz aller registrierten Projekte auf dem Server iterieren. Ein registriertes Projekt ist der Stamm der darunter befindlichen Projektsammlung.

Code-Ausschnitt (C#):

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

Neueste Versionen erhalten

Wie bereits erwähnt, bietet die Arbeitsplatz-Klasse einen reichhaltigen Satz an Methoden, um TFS-Befehle aus dem Code heraus auszuführen. Zur Illustration folgt ein Beispiel zur Erstellung eines Links zu einem Element im Arbeitsordner. Zur Erstellung eines relativen Pfads:

Code-Ausschnitt (C#):

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

Oder eines absoluten Pfads:

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

Zum Erhalt der aktuellsten Version des Elements im Beispiel, die folgende Codezeile hinzufügen:

Code-Ausschnitt (C#):

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

Hier führt das *RecursionType.Full* die Anfrage für alle Elemente unter dem Elementknoten aus, aber Sie können es nach den Anforderungen Ihrer Anwendung und der bestehenden Hierarchie der Elemente auswählen. Weitere Informationen erhalten Sie unter der API Dokumentation auf MSDN.

Eine Liste von anstehenden Änderungen erhalten

Sie können ebenfalls eine Liste der anhängigen Änderungen erhalten, die an dem Element oder einer Sammlung von Elementen vorgenommen werden, und zwar durch die folgende Codezeile:

Code-Ausschnitt (C#):

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

Berücksichtigen Sie bei der Sammlung von Elementen die übergroßen Abweichungen bei Item[] als Argument.

Ein- und Auschecken

Sie können ein Element oder eine Sammlung von Elementen auschecken:

Code-Ausschnitt (C#):

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

Zum Einchecken der Sammlung oder anstehenden Änderungen zu einem Element:

Code-Ausschnitt (C#):

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

–Vorgang rückgängig machen

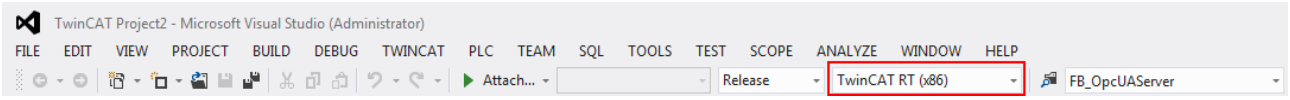
Zum Rückgängigmachen eines Vorgangs einfach Folgendes ausführen:

Code-Ausschnitt (C#):

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

4.3.1.3 Setting TwinCAT target platform

Dieser Artikel beschreibt, wie die TwinCAT Zielplattform über Automation Interface Code eingestellt wird. Die Zielplattform bestimmt, für welches Ziel die TwinCAT-Konfiguration kompiliert werden soll, z. B. TwinCAT x86 oder TwinCAT x64, und wird normalerweise von einer TwinCAT XAE Toolbar in Visual Studio aus eingestellt.



Der folgende Code-Ausschnitt geht davon aus, dass Sie bereits über eine DTE-Instanz verfügen, die zu einer TwinCAT-Konfiguration verbindet. Er überprüft die aktuell eingestellte Zielplattform und stellt sie zu einer anderen Plattform um.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

HINWEIS

Zielhardwareplattform oder Zielgerät mit TwinCAT-Konfiguration ändern?

Dieser Artikel beschreibt, wie die Zielhardwareplattform geändert werden kann. Wenn Sie das tatsächliche Zielgerät ändern möchten, auf das die TwinCAT-Konfiguration geschrieben werden soll, verwenden Sie bitte die Methode ITcSysManager::SetTargetNetId().

Der folgende Code-Ausschnitt zeigt, wie Sie Zugriff auf den Visual Studio Configuration Manager erhalten und einzelne TwinCAT-Teilprojekte (SPS, C++ …) für den Build-Prozess aktivieren oder deaktivieren. Der Ausschnitt geht davon aus, dass das derzeit geöffnete TwinCAT-Projekt als „TwinCAT-Projekt1" bezeichnet wird und dieses ein SPS-Projekt „Untitled1" enthält. Er deaktiviert anschließend das SPS-Projekt für den Build-Prozess.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.3.1.4 Anbindung an eine bestehende Visual Studio-Instanz

Die folgenden Code-Ausschnitte zeigen, wie die Anbindung an eine bestehende (bereits laufende) Instanz von Visual Studio vorgenommen wird. Die Ausschnitte wurden in C# geschrieben.

Das Beispiel setzt sich aus drei verschiedenen Methoden zusammen, die voneinander abhängig sind. Selbstverständlich können Sie Anpassungen vornehmen, damit es besser zu Ihrer Anwendungsumgebung passt. Die folgende Tabelle erläutert jede Methode im Einzelnen.

Methode	Beschreibung
getRunningObjectTable()	Befragt die Running Object Table (ROT) nach einem Snapshot aller laufenden Prozesse in der Tabelle. Gibt alle gefundenen Prozesse in eine Hash-Tabelle zurück, die anschließend von getIdeInstances() zur weiteren Filterung im Hinblick auf DTE-Instanzen verwendet wird.
getIdeInstances()	Sucht nach DTE-Instanzen in dem ROT-Snapshot und gibt eine Hash-Tabelle mit allen gefunden Instanzen zurück. Diese Hash-Tabelle kann dann von der attachToExistingDte()-Methode zwecks Auswahl einzelner DTE-Instanzen verwendet werden. Sie können die Abfrage für candidateName entsprechend einer spezifischeren Visual Studio progId [▶ 30] ändern, z.B. "VisualStudio.DTE.11.0" um nach Visual Studio 2012-Instanzen zu suchen.
attachToExistingDte()	Verwendet die getIdeInstances()-Methode um auf der Grundlage ihres Lösungspfads eine DTE-Instanz zu wählen und, wenn gefunden, ein neues DTE-Objekt an diese Instanz anzubinden.

getRunningObjectTable()

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

Bitte achten Sie darauf, dass Sie die CreateBindCtx() Methode als einen DllImport von der ole32.dll ausdrücklich referenzieren müssen, z. B.:

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

getIdeInstances()

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

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

attachToExistingDte()

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

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

4.3.1.5 Zugriff auf Fenster im Visual Studio

Das Visual Studio Automation Interface stellt diverse Methoden und Properties zur Verfügung, um auf aktive Fenster im Visual Studio zugreifen zu können. Das folgende Kapitel stellt Beispiel-Code für diese Funktionalität zur Verfügung. Desweiteren empfehlen wir als detailiertes Nachschlagewerk die Webseiten des Microsoft Developer Network (MSDN), welche ausführliche Informationen über das Visual Studio Objektmodell zur Verfügung stellen. Dieses Kapitel behandelt die folgenden Aufgaben:

- Zugriff auf aktive Fenster
- Schliessen von aktiven Fenstern
- Öffnen von Fenstern aus der TwinCAT SPS Konfiguration

Zugriff auf aktive Fenster

Die DTE-Schnittstelle bietet eine Eigenschaft namens "ActiveWindow", die das derzeit aktive Fenster in Visual Studio als ein Objekt des Typs EnvDTE.Window ausgibt.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Schliessen von aktiven Fenstern

Abhängig von der Anwendung des Kunden ist es möglicherweise zweckmäßig, vor der Konfiguration von TwinCAT über das Automation Interface zunächst alle offenen Fenster zu schliessen. Der folgende Code-Ausschnitt schliesst alle aktiven Fenster bis auf den "Solution Explorer".

Code-Ausschnitt (C#):

```
try
{
while (!dte.ActiveWindow.Caption.Contains("Solution Explorer"))
dte.ActiveWindow.Close();
}
catch (InvalidOperationException ex)
```

```
{
// use DTE.Quit() to close main window
}
```

Fenster von der TwinCAT SPS-Konfiguration aus öffnen

Es ist ebenfalls möglich, Fenster von TwinCAT SPS zu öffnen, z. B. eine Visualisierung. Der folgende Code-Ausschnitt öffnet eine bestehende TwinCAT Visualisierung vom SPS-Projekt "Untitled1" und macht sie zum aktiven Fenster.

Code-Ausschnitt (C#):

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

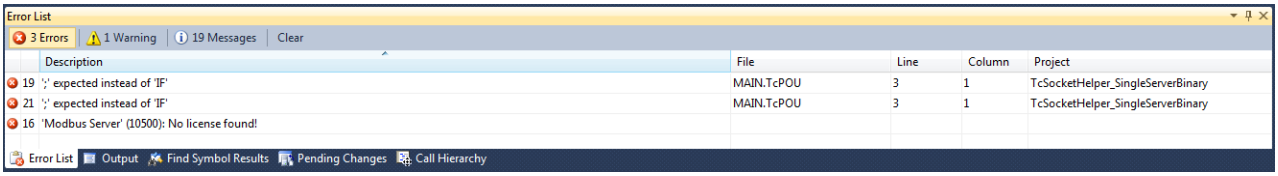
Code-Ausschnitt (Powershell):

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

4.3.1.6 Zugriff auf das Fehlerlistenfenster von Visual Studio

Dieses Kapitel beschreibt den Zugriff auf das Fehlerlistenfenster von Visual Studio. Weil TwinCAT 3 sich selbst in die Visual Studio-Umgebung integriert, ist das Lesen des Inhalts dieses Fensters für das Debugging und die Diagnose von großer Bedeutung. Um den Inhalt dieses Fensters zu lesen, müssen Sie auf die Visual Studio COM-Schnittstelle zugreifen. Da diese COM-Schnittstelle auch für den Zugang zu TwinCAT XAE notwendig ist (z.B. wie in unserem Artikel Zugriff auf die TwinCAT-Konfiguration [▶ 20] beschrieben), müssen Sie keine zusätzlichen Referenzen zu Ihrem Programmcode hinzufügen.

Die folgende Dokumentation beschreibt den Zugriff auf das Fehlerlistenfenster, der sehr hilfreich sein kann, z.B. wenn Sie ein SPS-Projekt kompilieren möchten, und dabei den Kompilierungsprozess auf etwaige Fehlermeldungen hin beobachten möchten. Im Fehlerlistenfenster werden Statusmeldungen für verschiedene Eigenschaften der Visual Studio (und somit TwinCAT) Entwicklungsumgebung eingeblendet. Zu diesen Eigenschaften gehören z.B. bei der Kompilierung eines Projekts auftretende Übersetzungsfehler oder Lizenzprobleme.



Das Lesen des Inhalts dieses Fensters kann sehr wichtig sein, z.B. um bei der Kompilierung eines SPS-Projekts auftretende Fehler zu erkennen. Der folgende Code-Ausschnitt zeigt, wie der Inhalt dieses Fensters mit Hilfe einer C#-Anwendung gelesen werden kann.

Code-Ausschnitt (C#):

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

Achten Sie darauf, dass die Eigenschaft „ToolWindows“ im Namensraum EnvDTE80.DTE2 verfügbar ist.

Code-Ausschnitt (Powershell):

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

Wie Sie sehen können, gibt die Eigenschaft ErrorItems eine Sammlung aller in der Fehlerliste vorhandenen Elemente zurück, wobei jedes Element vom Typ ErrorItem ist. Sie können diese Sammlung durchlaufen, indem Sie sich z.B. deren Eigenschaft Count zunutze machen. Jedes ErrorItem-Objekt hat die folgenden Eigenschaften, die mit den Spalten im Fehlerlistenfenster übereinstimmen (vergleiche mit Screenshot oben):

Eigenschaft	Beschreibung
Beschreibung	Beschreibt die Fehlermeldung.
FileName	Name der Datei, in welcher der Fehler auftrat.
Line	Zeile, in welcher der Fehler auftrat.
Column	Spalte, in welcher der Fehler auftrat.
Project	Projekt, in welchem der Fehler auftrat.

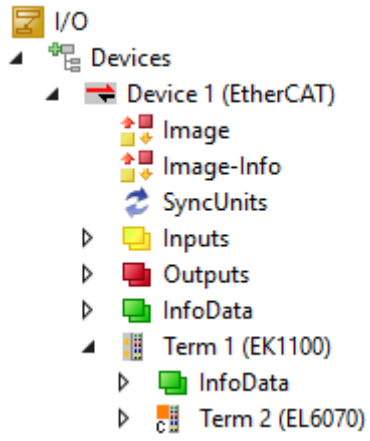
Beachten Sie, dass die vier letzten Eigenschaften nicht immer verwendet werden, bzw. nicht in jeder Situation Sinn ergeben. So sehen Sie z.B. im Screenshot oben, dass die Fehlermeldung „No license found“ (Keine Lizenz gefunden) nicht an eine bestimmte Datei, Zeile oder Spalte gebunden ist, weil sie eine allgemeine TwinCAT-Fehlermeldung ist.

4.3.2 Lizenzierung

4.3.2.1 Konfiguration einer Lizenzierungshardware

Der folgende Artikel beschreibt wie Lizenzierungshardware (z.B. eine EL6070 Klemme) konfiguriert wird. Im Wesentlichen geht es hierbei um die Selektion und Konfiguration des Geräts in den TwinCAT Lizenzierungsdialogen.

Als Voraussetzung für die folgenden Schritte muss die entsprechende Lizenzierungshardware im I/O Teil der TwinCAT Konfiguration vorhanden sein, z. B. bei einer EL6070:



Finden aller vorhandenen Lizenzierungsgeräte

Eine Überprüfung auf alle vorhandenen Lizenzierungshardware kann durch einen Export der XML-Beschreibung auf dem „License“ Knoten der TwinCAT Konfiguration erfolgen.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die XML-Beschreibung listet die verfügbare Lizenzierungshardware auf, zum Beispiel:

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

Auswahl eines Lizenzierungsgeräts

Um eine Lizenzierungshardware zur TwinCAT Lizenzkonfiguration hinzuzufügen, kann entweder der Gerätename oder dessen ObjectID verwendet werden. Ersteres eignet sich gut, wenn der Gerätename bekannt ist, z. B. wenn das I/O-Gerät im Vorfeld generiert wurde. Letzteres kann aus der obigen XML-Beschreibung ermittelt werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Systemvoraussetzungen

Vorausgesetzte TwinCAT Version

TwinCAT v3.1.4022.4

4.3.2.2 Aktivieren von Lizenz-Antwortdateien

Der folgende Artikel beschreibt wie Lizenz-Antwortdateien über das TwinCAT Automation Interface eingespielt werden können. Diese Funktionalität wird über die XML-Beschreibung des „License“ Knotens zur Verfügung gestellt.

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

Im XML-Bereich <ActivateResponseFile> kann der <Path> zur Lizenzantwortdatei angegeben werden. Dieses XML-Dokument kann dann über ein ConsumeXml() auf dem „License“ Knoten eingespielt werden, um die Lizenzantwortdatei zu aktivieren. Der Parameter <OemGuid> wird nur in Sonderfällen benötigt und kann mit einem beliebigen Wert, z. B. 0, versehen werden.

Code-Ausschnitt (C#):

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

Systemvoraussetzungen

Vorausgesetzte TwinCAT Version
TwinCAT v3.1.4022.4

4.3.3 System

4.3.3.1 Bestehende Konfigurationen öffnen und aktivieren

Um eine zuvor erzeugte Konfiguration zu aktivieren, muss eine Instanz von TwinCAT XAE erzeugt, die Konfiguration geladen und aktiviert werden.

Ablauf

Die ProgId "**VisualStudio.DTE.10.0"** wird für die Erstellung einer Instanz von Visual Studio verwendet. Über das Visual Studio DTE-Objekt ist eine volle Kontrolle über Visual Studio möglich. Die Vorgehensweise für die Erstellung der ITcSysManager [▶ 118]-Schnittstelle (hier die '*sysMan*'-Instanz) wird im Kapitel Auf TwinCAT-Konfigurationen zugreifen [▶ 20]beschrieben.

Beispiel (CSharp):

Beachten Sie, dass Sie für dieses Beispiel Ihrem Projekt sowohl eine Referenz auf "Microsoft Developer Environment 10.0", als auch auf "Beckhoff TwinCAT XAE Base" hinzufügen müssen.

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

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

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

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

ITcSysManager sysMan = pro.Object;

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

Beispiel (PowerShell):

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

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

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

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

Beispiel (VBScript):

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

4.3.3.2 Bestehende Projekte von einem TwinCAT Target öffnen

Dieser Artikel beschreibt, wie bestehende TwinCAT Projekte von einem verbundenen TwinCAT Target geöffnet werden. Das Target muss verfügbar sein, was bedeutet, dass ADS Routen vorhanden sein müssen, um das Projekt von dem Target abrufen zu können.

Der folgende Code-Ausschnitt zeigt, wie das TwinCAT Projekt von einer verbundenen Target Runtime abgerufen wird.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die Methode ExecuteCommand() von der Visual Studio API ermöglicht das Auslösen des TwinCAT Befehls (File.OpenProjectFromTarget), um das Projekt eines verbundenen TwinCAT Targets zu öffnen. Die drei Parameter sind in den zweiten Parameter der ExecuteCommand() Methode aufzunehmen. Diese drei Parameter sind: Routenname zum Target, lokales Projektverzeichnis (wo die Projektdateien gespeichert werden sollen), lokaler Projektname. Alle drei Parameter sind voneinander mit einem Leerzeichen zu trennen.

4.3.3.3 Erstellung von und Umgang mit Variablenzuordnungen

Ein sehr geläufiges Szenario, bei dem das TwinCAT-Automation Interface verwendet wird, ist die automatische Erzeugung der Variablenzuordnungen, z.B. zwischen SPS-Ein-/Ausgangsvariablen und deren entsprechenden I/O-Gegenstücken. Das Automation Interface bietet mehrere Methoden, um die Erstellung, Löschung oder Speicherung von Variablenzuordnungen zu vereinfachen. Der nachfolgende Artikel beschreibt diese Methoden kurz und erläutert deren Verwendung anhand einiger Beispiele. Die folgenden Themen werden behandelt:

- Allgemeine Informationen
- Variablen verknüpfen
- Variablenverknüpfungen auflösen
- Alle Variablenzuordnungen abrufen/festlegen
- Alle Variablenzuordnungen löschen

Allgemeine Informationen

Variablenzuordnungen können zwischen unterschiedlichen Ein-/Ausgangs-Tree Items in einem TwinCAT Projekt auftreten, z.B.:

- Zwischen SPS-Ein-/Ausgangsvariablen und I/O (und umgekehrt)
- Zwischen SPS-Ein-/Ausgangsvariablen und NC (und umgekehrt)
- Zwischen SPS-Ein-/Ausgangsvariablen und TcCOM-Objekten (und umgekehrt)
- …

In diesem Artikel werden die Mechanismen des Automation Interface beschrieben, die für alle diese Fälle verwendet werden können.

Variablen verknüpfen

Aus Sicht des Automation Interface werden Variablenzuordnungen immer zwischen zwei Tree Items vorgenommen, z.B. zwischen einer SPS-Ein-/Ausgangsvariablen und dem entsprechenden I/O-Gegenstück. Für die Verknüpfung von zwei Tree Items stellt das Automation Interface die Methode ITcSysManager::LinkVariables() zur Verfügung, die ein bestimmtes Quellenstrukturelement mit einem bestimmten Ziel-Tree Item verknüpft.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Variablenverknüpfungen auflösen

Mit der Verknüpfung von Variablen vergleichbar bietet das Automation Interface eine Methode ITcSysManager::UnlinkVariables() mit Hilfe derer die Verknüpfung zwischen einem bestimmten Quellen-Tree Item und einem bestimmten Ziel-Tree Item aufgelöst werden kann.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Alle Variablenzuordnungen Speichern/Wiederherstellen

Die Methoden ITcSysManager2::ProduceMappingInfo() und ITcSysManager2::ConsumeMappingInfo() können dazu verwendet werden, um alle Variablenzuordnungen eines TwinCAT Projekts zu speichern oder wiederherzustellen. Die erste Methode liest alle Variablenzuordnungen in einem TwinCAT Projekt und gibt diese in eine XML-Struktur zurück, die später mit Hilfe der letzteren Methode reimportiert werden kann.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Alle Variablenzuordnungen löschen

Mit Hilfe der Methode ITcSysManager3.ClearMappingInfo() können alle Variablenzuordnungen in einem TwinCAT Projekt gelöscht werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

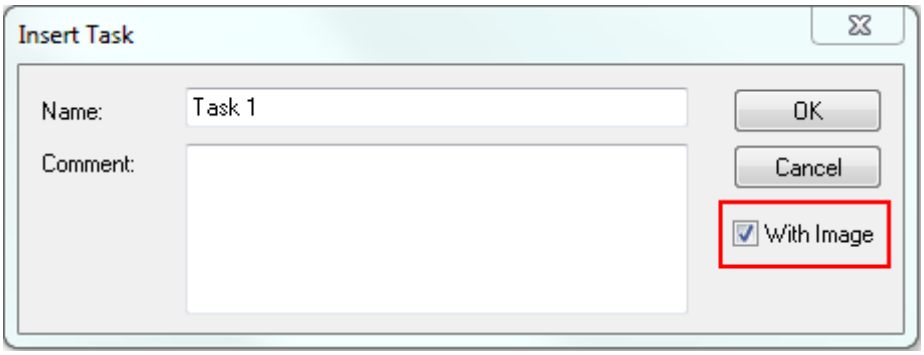
4.3.3.4 Erstellung von und Umgang mit Tasks

In diesem Artikel wird beschrieben, wie Tasks mit Hilfe des TwinCAT Automation Interface erstellt und gehandhabt werden. Er besteht aus folgenden Themen:

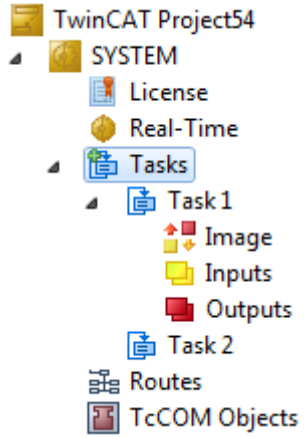
- Allgemeine Informationen
- Tasks einfügen
- Ein-/Ausgangsvariablen einfügen

Allgemeine Informationen

Es können zwei Task-Typen in TwinCAT XAE und daher auch mit dem Automation Interface konfiguriert werden: Tasks mit und ohne Prozessabbild. Wenn Sie ein neues Task in TwinCAT XAE einfügen, dann können Sie entscheiden, ob Sie ein Prozessabbild einfügen möchten oder nicht, in dem Sie die entsprechende Checkbox im Dialogfenster „Task einfügen“ auswählen oder nicht.



Daraufhin beinhaltet die eingefügte Task entweder drei weitere untergeordnete Knoten (Abbild, Inputs, Outputs) oder nicht – wie im folgenden Beispiel gezeigt (Task 1 = mit Abbild, Task 2 = ohne Abbild).



Tasks einfügen

Um eine Task über das Automation Interface einzufügen, können Sie die Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] mit den entsprechenden SubTypes für "Mit Abbild" (SubType = 0) und "Ohne Abbild" (SubType = 1) verwenden.

Code-Ausschnitt (C#)

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

Code-Ausschnitt (Powershell):

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

Code-Ausschnitt (C#)

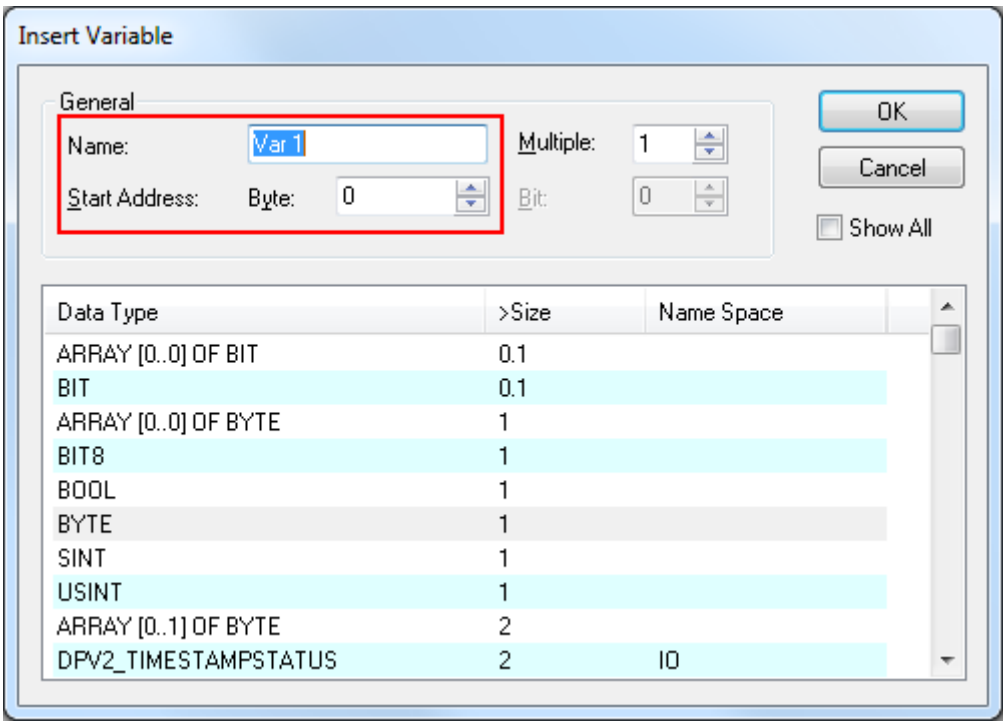
```
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
tasks.CreateChild("Task 2 (Without Image)", 1, null, null);
```

Code-Ausschnitt (Powershell):

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

Ein-/Ausgangsvariablen einfügen

An Prozessabbildern (Task „Mit Abbild“) können Ein- und Ausgangsvariablen angefügt werden, welche dann mit den verschiedenen E/A-Geräten oder Variablen anderer Tasks verknüpft werden können. Über den entsprechenden TwinCAT XAE-Dialog können Sie z.B. den Datentyp und die Adresse der Ein-/Ausgangsvariablen im Prozessabbild auswählen. Durch Klicken auf „Ok“ wird die Variable dem Prozessabbild hinzugefügt.



Diese Prozedur kann auch über das Automation Interface unter Verwendung der Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] mit entsprechendem Variablendatentyp als vInfo ausgelöst werden. In diesem Falle spezifiziert SubType die „Startadresse“ wie oben im Dialog gezeigt.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Wenn SubType = -1, dann hängt TwinCAT die neue Variable automatisch ans Ende der Variablenliste an.

4.3.3.5 Verwendung von Templates

Der folgende Artikel beschreibt, wie Vorlagen verwendet werden, statt jede einzelne Einstellung separat über das Automation Interface zu konfigurieren. Die Verwendung von Vorlagen bietet viele Vorteile: Sie sind einfacher zu pflegen, einfacher durch neue Vorlagen auszutauschen und sie bieten mehr Möglichkeiten, wenn mit mehreren Teams an einem TwinCAT-Projekt gearbeitet wird. Dieses Dokument beschreibt die Nutzung von Vorlagen und behandelt die folgenden Themen:

- Die allgemeine Idee hinter Vorlagen und ihren verschiedenen Ebenen

- Arbeit mit I/O Vorlagen
- Arbeit mit Bewegungsvorlagen (Achsen)
- Arbeit mit SPS-Vorlagen

Die allgemeine Idee hinter Vorlagen und ihren verschiedenen Ebenen

Die Idee hinter der Nutzung von Vorlagen ist so einfach wie der Code des Automation Interface und die Anwendung des Automation Interface selbst. Den meisten Benutzern ist nicht bekannt, dass Vorlagen in einer TwinCAT-Konfiguration auf verschiedenen Ebenen existieren können. Hierzu zählen:

- Vorlagen auf „Konfigurationsebene“:

  Vorlagen auf Konfigurationsebene können als verschiedene TwinCAT-Konfigurationen existieren (*.sln oder *.tszip Datei). Jede Konfiguration kann unterschiedlichen Inhalt umfassen und durch den Code des Automation Interface geöffnet und aktiviert werden.
- Vorlagen auf „(SPS) Projektebene“:

  Vorlage auf (SPS) Projektebene können als mehrere TwinCAT SPS-Projekte existieren, entweder als ungepackte Ordnerstruktur (*.plcproj Datei) oder als Containerdatei (*.tpzip). Diese SPS-Projekte können dann bei Bedarf über den Code des Automation Interface importiert werden.
- Vorlagen auf „Strukturelementebene“:

  Vorlagen auf Strukturelementebene können als sogenannte TwinCAT Exportdateien (*.xti) existieren. Diese Dateien können bei Bedarf über den Code des Automation Interface importiert werden und enthalten alle Einstellungen, die für ein EtherCAT Master I/O Gerät vorgenommen wurden.

Wie die vorstehende Beschreibung nahelegt, haben alle verschiedenen Vorlagenebenen eine Sache gemeinsam: die Vorlagen existieren als Dateien im Dateisystem. Es gilt als Best Practice, dass zu Beginn der Implementierung einer eigenen Anwendung des Automation Interface eindeutig definiert wird, welche Art von „Vorlagenpool“ verwendet wird. Ein Vorlagenpool beschreibt ein Repository, in dem alle Vorlagendateien (die ebenfalls eine Mischung verschiedener Vorlagenebenen sein können) gespeichert werden. Dies könnte einfach das lokale (oder ferne) Dateisystem sein oder ebenfalls ein Source-Control-System (z. B. Team Foundation Server), von dem aus Vorlagedateien automatisch abgerufen, ausgecheckt und zu einer neuen TwinCAT-Konfiguration zusammengesetzt werden. Zur besseren Übersicht und zur Vereinfachung der ersten Schritte haben wir einen separaten Dokumentationsartikel erstellt, der aufzeigt, wie das Visual Studio Object Model (DTE) zur Verbindung an einen Team Foundation Server (TFS) verwendet wird und wie diese Vorgänge über den Programmcode ausgeführt werden. Sie erhalten jedoch weitere Informationen über die Verwendung von TFS über den Code des Automation Interface, wobei wir sehr empfehlen, die entsprechenden Artikel des Microsoft Developer Network (MSDN) zu lesen.

Die folgenden Themen beschreiben, wie jede Vorlagenebene in des Automation Interface für einen unterschiedlichen TwinCAT- Bereich (I/O, SPS usw.) verwendet werden kann.

Arbeit mit Konfigurationsvorlagen

Vorlagen auf Konfigurationsebene bieten die grundlegendste Möglichkeit der Verwendung von Vorlagen. In diesem Szenario enthält ein Vorlagenpool zwei oder mehr vorkonfigurierte TwinCAT-Konfigurationen und bietet sie als TwinCAT Lösungsdateien (*.sln oder *.tszip) an. Diese Dateien können über das Automation Interface durch Verwendung der Visual Studio Methode AddFromTemplate() geöffnet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

OR

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Arbeit mit I/O Vorlagen

Bei der Arbeit mit I/O Geräten erhalten Benutzer häufig dieselben Aufgaben auf den I/O Geräten immer wieder. Dies bedeutet dieselben Einstellungen an den I/O Geräten mit jeder TwinCAT-Konfiguration machen. TwinCAT bietet einen Mechanismus, der alle diese Einstellungen in einem speziellen Dateiformat speichert, der sogenannten XTI Datei (*.xti). Dieses Dateiformat kann von dem Code des Automation Interface verwendet werden, um eine neue Konfiguration unter Verwendung der Methode ImportChild() zu importieren, die in der Schnittstelle ITcSmTreeItem definiert ist.

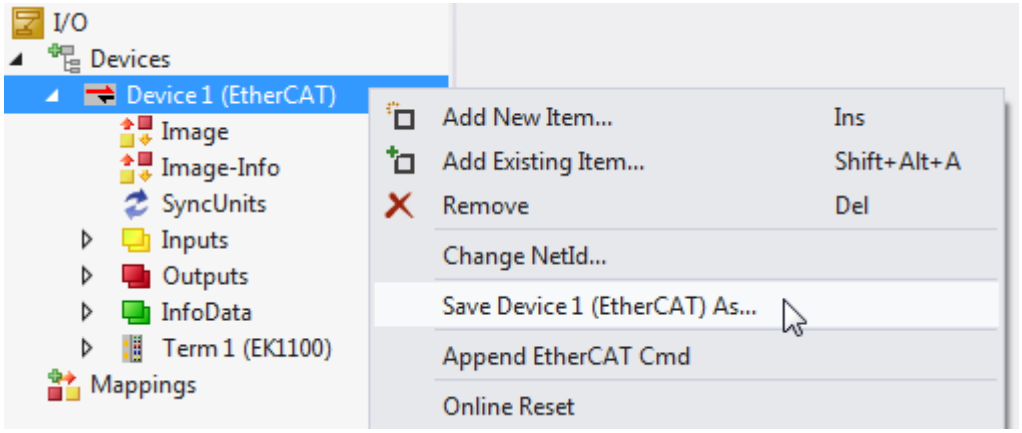
Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

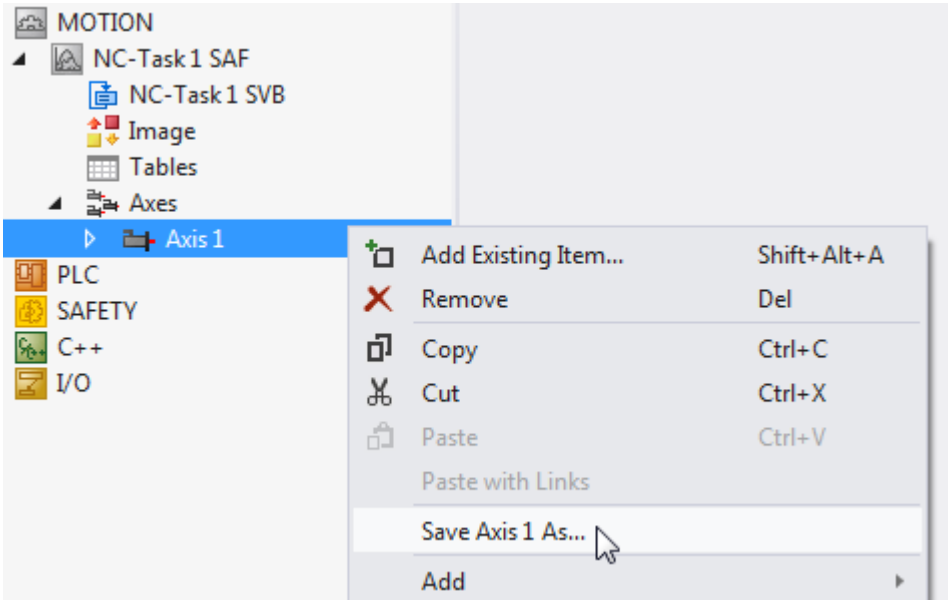
Dabei ist es wichtig darauf hinzuweisen dass bei einem Export eines I/O Geräts innerhalb von TwinCAT XAE alle Teilgeräte automatisch exportiert werden und in der Exportdatei ebenfalls enthalten sind. Der folgende Screenshot zeigt, wie I/O Geräte in eine Exportdatei exportiert werden.



Denken Sie daran, dass Sie ebenfalls das Automation Interface verwenden können, um ein I/O Gerät in eine XTI Datei zu exportieren. Hierfür steht die Methode ExportChild() aus der ITcSmTreeItem Schnittstelle zur Verfügung.

Arbeit mit Bewegungsvorlagen (Achsen)

Die Verwendung von Bewegungsachsenvorlagen ist sehr ähnlich zu der von I/O Geräten. Achsen können ebenfalls in eine XTI-Datei exportiert werden, entweder über TwinCAT XAE oder über den Code des Automation Interface unter Verwendung von ExportChild().



Durch Verwendung von ImportChild() können diese Exportdateien später wieder importiert werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Arbeit mit SPS-Vorlagen

SPS-Vorlagen sind auf zwei Arten verfügbar: Sie können entweder die kompletten SPS-Projekte als Ganzes verwenden oder jede POU einzeln als Vorlage. Zur Integration in ein bestehendes TwinCAT-Projekt verwenden Sie einfach die CreateChild() Methode der ITcSmTreeItem Schnittstelle.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Weitere Optionen zum Import bestehender SPS-Projekte erhalten Sie in dem Best Practice Artikel über „Zugriff auf, Erstellung von und Umgang mit SPS-Projekten.“

Die nächste Granularitätsebene zum Import von Vorlagendateien für die SPS ist der Import von bestehenden POUs, wie z. B. Funktionsblöcke, Strukturen, Aufzählungen, globale Variablenliste usw. Einer der Gründe dafür, warum sich ein Entwickler für einzelne POUs als Vorlagen entscheiden kann, besteht darin, dass es einfacher ist, einen Pool von bestehenden Funktionalitäten aufzubauen und in separaten POUs zusammenzufassen, z. B. verschiedene Funktionsblöcke, die verschiedene Sortieralgorithmen abdecken. Bei der Projekterstellung wählt der Entwickler einfach die Funktionalität aus, die er in seinem TwinCAT-Projekt benötigt und greift auf das Vorlagetool zu, um die entsprechende POU abzurufen und diese in das SPS-Projekt zu importieren.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

HINWEIS

Eine oder mehrere Vorlagen importieren

Die POU Vorlagedatei kann nicht nur eine .TcPou Datei sein, sondern auch die entsprechenden Dateien für DUTs und/oder GVLs. Das vorstehende Beispiel gibt zwei übliche Wege wieder, um POU Vorlagedateien zu importieren. Der erste besteht darin, eine einzige Datei zu importieren, der zweite, mehrere Dateien auf einmal zu importieren, indem die Dateipfade in einem String-Array gespeichert werden und dieses String-Array als vInfo Parameter von CreateChild() verwendet wird.

4.3.3.6 Zugriff auf TwinCAT Remote Manager

Dieser Artikel beschreibt, wie über das Automation Interface auf die TwinCAT Remote Manager Funktionalität zugegriffen wird. Der TwinCAT Remote Manager ermöglicht das Umschalten zwischen TwinCAT 3 XAE Versionen, die auf demselben Entwicklungscomputer installiert sind. Zum Zugriff auf den Remote Manager über das Automation Interface müssen einfach die folgenden Code-Ausschnitte ausgeführt werden.

Code-Ausschnitt (C#):

```
ITcRemoteManager remoteManager = dte.GetObject("TcRemoteManager");
remoteManager.Version = "3.1.4020.0";
```

Code-Ausschnitt (Powershell):

```
$remoteManager = $dte.GetObject("TcRemoteManager")
$remoteManager.Version = "3.1.4020.0"
```

4.3.3.7 Zuweisung von Aufgaben an CPU-Kerne

Ablauf

- Kerne für Echtzeitverwendung freigeben.
- Kerne mit LoadLimit, Basiszykluszeit (BaseTime) und einem Latenz-Watchdog parametrisieren
- Tasks CPU-Ressourcen zuweisen.

Die folgenden Code-Ausschnitte stehen ebenfalls als Download in unserem Abschnitt Beispiele zur Verfügung. (Beispiel 3)

Beispiel (C#):

```
ITcSysManager3 systemManager = null;
[Flags()]
public enum CpuAffinity : ulong
{
CPU1 = 0x0000000000000001,
CPU2 = 0x0000000000000002,
CPU3 = 0x0000000000000004,
CPU4 = 0x0000000000000008,
CPU5 = 0x0000000000000010,
CPU6 = 0x0000000000000020,
CPU7 = 0x0000000000000040,
CPU8 = 0x0000000000000080,
None = 0x0000000000000000,
MaskSingle = CPU1,
MaskDual = CPU1 | CPU2,
MaskQuad = MaskDual | CPU3 | CPU4,
MaskHexa = MaskQuad | CPU5 | CPU6,
MaskOct = MaskHexa | CPU7 | CPU8,
MaskAll = 0xFFFFFFFFFFFFFFFF
}

public void AssignCPUCores()
{
ITcSmTreeItem realtimeSettings = systemManager.LookupTreeItem("TIRS");
// CPU Settings
// <TreeItem>
// <RTimeSetDef>
// <MaxCPUs>3</MaxCPUs>
```

```
 // <Affinity>#x0000000000000007</Affinity>
// <CPUs>
// <CPU id="0">
// <LoadLimit>10</LoadLimit>
// <BaseTime>10000</BaseTime>
// <LatencyWarning>200</LatencyWarning>
// </CPU>
// <CPU id="1">
// <LoadLimit>20</LoadLimit>
// <BaseTime>5000</BaseTime>
// <LatencyWarning>500</LatencyWarning>
// </CPU>
// <CPU id="2">
// <LoadLimit>30</LoadLimit>
// <BaseTime>3333</BaseTime>
// <LatencyWarning>1000</LatencyWarning>
// </CPU>
// </CPUs>
// </RTimeSetDef>
// </TreeItem>

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

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

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

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

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

private void writeCpuProperties(XmlWriter writer, int id, int loadLimit, int baseTime, int
latencyWarning)
{
writer.WriteStartElement("CPU");
writer.WriteAttributeString("id", id.ToString());
```

```
writer.WriteElementString("LoadLimit", loadLimit.ToString());
writer.WriteElementString("BaseTime", baseTime.ToString());
writer.WriteElementString("LatencyWarning", latencyWarning.ToString());
writer.WriteEndElement();
}
```

4.3.3.8 Konfiguration von TwinCAT Boot Einstellungen

Der folgende Artikel beschreibt, wie die TwinCAT Boot Einstellungen über das Automation Interface konfiguriert werden. Hierfür können die Methoden ITcSmTreeItem::ProduceXml() und ITcSmTreeItem::ConsumeXml() verwendet werden, um die folgende XML-Struktur zu erstellen oder zu importieren, welche die entsprechenden Einstellungen in TwinCAT XAE darstellt.

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

4.3.3.9 Aktivierung oder Deaktivierung der IndependentProjectFile Einstellung

Zur Verbesserung der Entwicklungsarbeit im Hinblick auf die Source Control Integration bietet TwinCAT 3 die Möglichkeit, die Einstellungen in separaten Projektdateien zu speichern – der sogenannten „IndependentProjectFile“. Die TreeItem-basierte Einstellung kann ebenfalls über das TwinCAT Automation Interface aktiviert/deaktiviert werden. Die Schnittstelle ITcSmTreeItem6 bietet die erforderliche Eigenschaft.

Der folgende Code-Ausschnitt zeigt, wie diese Einstellung aktiviert wird, z. B. auf einem EtherCAT Master Gerät.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.3.3.10 Von Offline zu Online-Konfigurationen

In diesem Artikel wird erläutert, wie eine TwinCAT-Konfiguration, die „offline“ erstellt wurde, mit Hilfe des TwinCAT-Automation Interface in eine Online-Konfiguration umgewandelt wird. Der Begriff „offline“ bedeutet, dass zum Zeitpunkt der Erstellung der Konfiguration keine physikalischen I/Os vorliegen und demzufolge die realen Adressen z.B. für einen EtherCAT Master nicht verfügbar sind. Die folgenden Punkte werden in diesem Artikel behandelt:

- Allgemeine Informationen
- Eine Offline-Konfiguration erstellen
- Wechsel zu einer Online-Konfiguration

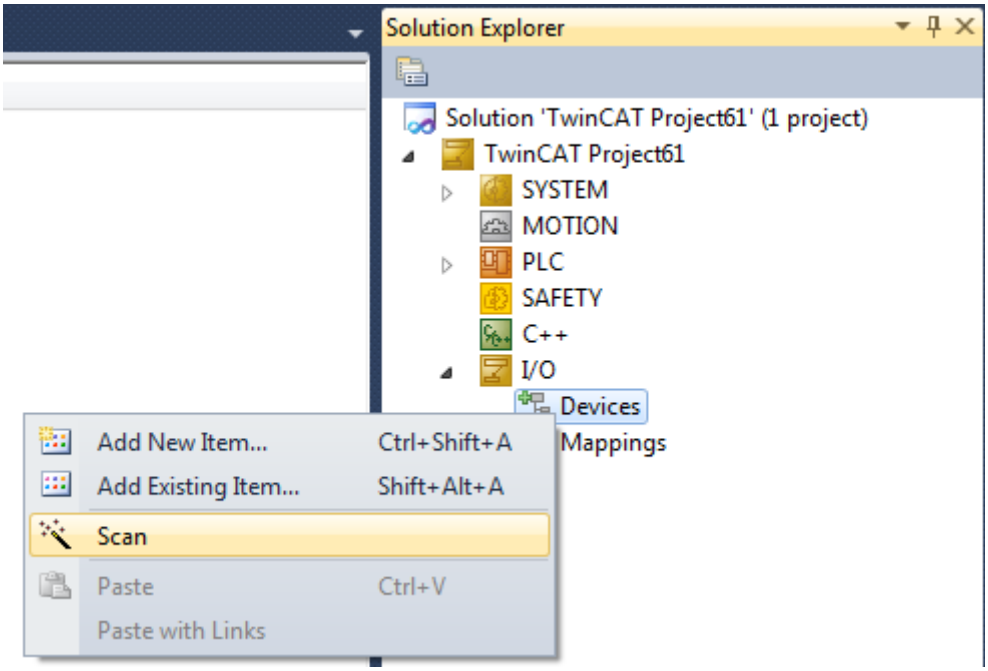
Allgemeine Informationen

Bei der Erstellung einer TwinCAT-Konfiguration sind zwei Szenarien geläufig:

- Szenario 1: Dieses Szenario basiert auf dem realen physikalischen Gerät, auf dem die TwinCAT-Konfiguration später laufen soll. Aus diesem Grunde sind alle I/Os **online** , bereits verfügbar und mit dem Gerät verbunden.
- Szenario 2: Zu diesem Szenario gehört möglicherweise die **Offline** -Erstellung der Konfiguration (d.h. ohne dass irgendein I/O mit dem Gerät verbunden ist) und der spätere Wechsel zu einer Online-Konfiguration, sobald die I/Os verfügbar sind. Auf dieses Szenario wird in diesem Artikel das Hauptaugenmerk gelegt.

Beide Szenarien sind mit dem TwinCAT-Automation Interface möglich. Weil aber im Falle des Szenarios 2 wichtige Informationen über einige der I/O fehlen, z.B. deren physikalische Adressen, ist Letzteres etwas komplexer, weil die erforderliche (Adress-) Information später übergeben werden muss.

Die vertraute TwinCAT XAE-Funktionalität "Scan Devices" spielt in diesem Verwendungsfall eine bedeutende Rolle, weil sie alle verfügbaren E/A-Geräte scannt und diese automatisch mit der Konfiguration verbindet, zusammen mit allen erforderlichen zusätzlichen Informationen, z.B. den physikalischen Adressen.



Im Falle einer physikalischen Steuerung kann diese Funktionalität auch über das Automation Interface aufgerufen werden, wobei sie dann eine XML-Darstellung aller gefundenen E/A-Geräte einschließlich deren entsprechenden Adressinformation zurückgibt.

Wenn eine TwinCAT-Konfiguration auf einer Steuerung aktiviert werden soll, dann muss die erforderliche Adressinformation für alle zur Konfiguration gehörenden E/A-Geräte gesetzt sein. Dies kann mit dem Aufruf der „Scan Devices“ Funktionalität über das Automation Interface und Festlegung der Adressinformation der E/A-Geräte in der Konfiguration erledigt werden. Das wird nun genauer erläutert.

Eine Offline-Konfiguration erstellen

Die Erstellung einer offline TwinCAT-Konfiguration wird in mehreren Artikeln beschrieben. Unsere Seite Produktbeschreibung [▶ 10] gibt Ihnen einen Überblick.

Wechsel zu einer Online-Konfiguration

Wenn Sie schlussendlich eine TwinCAT-Konfiguration erstellt haben, die nun auf einer physikalischen Steuerung aktiviert werden soll, dann befolgen Sie bitte die folgenden Schritte, um sicherzustellen, dass alle erforderlichen Adressinformationen vor dem Herunterladen der Konfiguration verfügbar sind.

- Schritt 1 [optional]: Verbindung mit dem Zielgerät
- Schritt 2: Das Gerät nach allen verfügbaren I/Os durchsuchen
- Schritt 3: Durch XML iterieren und die Adressinformation der I/Os konfigurieren

- Schritt 4: Die Konfiguration aktivieren

Selbstverständlich können Sie auch jederzeit eine Online-Konfiguration direkt unter Verwendung der ScanDevices-Funktionalität erstellen. Es gibt ein entsprechendes Beispiel [▶ 95], das Ihnen genau zeigt, wie das gemacht wird.

Schritt 1 [optional]: Verbindung mit dem Zielgerät

Befindet sich die physikalische Steuerung nicht auf derselben Maschine, auf welcher der Automation Interface-Code läuft, können Sie die Methode ITcSysManager [▶ 118]::SetTargetNetId() [▶ 124] verwenden, um die Verbindung mit dem Remote-Teilnehmer herzustellen, um anschließend mit den nächsten Schritten fortzufahren.

Schritt 2: Das Gerät nach allen verfügbaren I/Os durchsuchen

Die bereits erwähnte "Scan Devices"-Funktionalität kann über das Automation Interface mittels Aufruf der ITcSmTreeItem::ProduceXml()-Methode auf dem E/A-Geräteknoten ausgelöst werden.

Code-Ausschnitt (C#):

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

Schritt 3: Durch XML iterieren und die Adressinformation der I/Os konfigurieren

In diesem Beispiel wollen wir die Adressinformation eines EtherCAT-Teilnehmers, der bereits Teil unserer Offline-Konfiguration ist, aktualisieren. Die in Schritt 2 ausgeführte Methode ProduceXml() hat bereits die auf dem System verfügbaren E/A-Geräte zurückgegeben, die nun in der Variablen ‘foundDevices' verfügbar sind. Wir werden den EtherCAT-Teilnehmer in dieser XML über dessen Element-Subtyp (111) identifizieren und anschließend die Adressinformation des EtherCAT Masters in unserer Konfiguration aktualisieren.

Code-Ausschnitt (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);
}
}
```

Schritt 4: Die Konfiguration aktivieren

Der letzte Schritt beinhaltet lediglich die Aktivierung der Konfiguration auf dem Zielgerät. Verwenden Sie hierzu einfach die ITcSysManager::ActivateConfiguration()-Methode.

Code-Ausschnitt (C#):

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

4.3.3.11 Zugriff auf TwinCAT Variantenmanagement

Dieser Artikel beschreibt den Zugriff auf die Funktionalitäten des TwinCAT Variantenmanagements über das Automation Interface. Der Zugriff ist ab dem ITcSysManager14 (TCatSysManagerLib V 3.3.0.0) möglich. Es werden die folgenden Funktionen unterstützt:

1. Hinzufügen von Projektvarianten und Gruppen von Varianten
2. Setzen der aktiven Variante

3. Aktivieren von Einstellungen für das Variantenmanagement

1. Hinzufügen von Projektvarianten und Gruppen von Varianten

Code-Ausschnitt (C#):

```
string variantConfig = "<?xml version=\"1.0\"?><ProjectVariants><Group><Name>Group1</
Name><Member>Variant1</Member><Member>Variant2</Member></Group><Group><Name>Group2</
Name><Member>Variant2</Member><Member>Variant3</Member></Group><Variant><Name>Variant1</Name></
Variant><Variant><Name>Variant2</Name></Variant><Variant><Name>Variant3</Name></Variant></
ProjectVariants>";
sysManager.ProjectVariantConfig = variantConfig;
```

2. Setzen der aktiven Variante

Code-Ausschnitt (C#):

```
sysManager.CurrentProjectVariant = "Variant3";
sysManager.CurrentProjectVariant = "[Group1]";
```

3. Aktivieren von Einstellungen für das Variantenmanagement

Code-Ausschnitt (C#):

```
ITcSmTreeItem9 el2004_1 = (ITcSmTreeItem9)sysManager.LookupTreeItem("TIID^EtherCAT
Master^EK1100-1^EL2004-1");
// activate the "disable" setting for the Variant Management
el2004_1.PvDisable = true;
// choose a variant and disable it only for this variant
sysManager.CurrentProjectVariant = "Variant3";
el2004_1.Disabled = DISABLED_STATE.SMDS_DISABLED;
```

4.3.4 ADS

4.3.4.1 Erstellung von und Umgang mit ADS-Routen

Das Hinzufügen von ADS-Routen über das Automation Interface kann mit Hilfe der ConsumeXml()-Methode der ITcSmTreeItem-Schnittstelle erledigt werden. Allerdings sollte man mit der zugrunde liegenden XML-Struktur vertraut sein, bevor man Routen zu einem entfernten Zielgerät hinzufügt.

XML-Struktur

Der folgende Code-Ausschnitt stellt ein Beispiel für XML-Strukturen für das Hinzufügen von Routen zu einem entfernten Zielgerät dar. Beachten Sie, dass Sie entweder die IP-Adresse oder den Hostnamen des entfernten Zielgeräts angeben können.

Dieser Code-Ausschnitt fügt eine reguläre Route zu einem entfernten Zielgerät hinzu.

Code-Ausschnitt (XML):

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

Dieser Code-Ausschnitt fügt eine Projektroute zu einem entfernten Zielgerät hinzu.

Code-Ausschnitt (XML):

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

Der folgende Code-Ausschnitt verwendet den Hostnamen anstatt der IP-Adresse.

Code-Ausschnitt (XML):

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

Und für Projektrouten.

Code-Ausschnitt (XML):

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

Denken Sie daran, dass die XML-Struktur für reguläre und Projektrouten gleichzeitig benutzt werden kann.

Der folgende Code-Ausschnitt erstellt eine ADS-Route zu einem entfernten Gerät, das durch seine IP-Adresse (10.1.128.217) und AMS NetId (10.1.128.217.1.1) spezifiziert wurde.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

```
$xmlString ="<TreeItem><ItemName>Route Settings</ItemName><PathName>TIRR</
PathName><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList><AddRoute><RemoteName>RouteName</RemoteName><RemoteNetId>10.1.128.217.1.1</
RemoteNetId><RemoteIpAddr>10.1.128.217</RemoteIpAddr><UserName>Administrator</UserName><Password>1</
```

```
Password><NoEncryption></NoEncryption></AddRoute></RoutePrj></TreeItem>"
$routes = $systemManager.LookupTreeItem("TIRR")
$routes.ConsumeXml($xmlString)
```

4.3.4.2 Durchführung einer ADS Broadcast-Suche

Zum Auslösen einer TwinCAT Broadcast-Suche und Auffinden eines unbekannten entfernten ADS-Geräts können die ConsumeXml() und ProduceXml() Methoden aus der ITcSmTreeItem Schnittstelle verwendet werden.

Allgemeine Broadcast-Suche

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die Variable „result“ enthält nun eine XML-Darstellung aller gefundenen ADS-Geräte im Netzwerk. Zur Auswahl eines spezifischen Geräts aus dieser Liste kann der reguläre .NET Mechanismus für XML-Umgang verwendet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Diese Informationen können dann verwendet werden, um eine Route zu diesem ADS-Gerät hinzuzufügen, wie dies in einem separaten Artikel beschrieben ist.

Direkte Broadcast-Suche

Erfordert mindestens TwinCAT 3.1 Build 4020.10 oder höher.

Zur Ausführung einer Broadcast-Suche mit einem gegebenen Hostnamen oder einer IP-Adresse kann die folgende XML-Struktur in ConsumeXml() verwendet werden.

XML – Suche nach Hostnamen:

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

XML – Suche nach IP-Adresse:

```
<TreeItem>
  <RoutePrj>
```

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

Ein nachfolgendes ProduceXml() gibt den gefundenen Host wie folgt aus:

XML – Gefundener Host:

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

4.3.5 SPS

4.3.5.1 Zugriff auf, Erstellung von und Umgang mit SPS-Projekten

In diesem Kapitel wird die Erstellung von, der Zugriff auf und der Umgang mit SPS-Projekten ausführlich erklärt. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über SPS-Projekte
- Erstellung von und Umgang mit SPS-Projekten
- Bestehende SPS-Projekte öffnen
- Verschachtelte (nested) Projekte und Projektinstanzen
- Das SPS-Projekt als Bibliothek speichern
- Umgang mit den Online-Funktionalitäten (Login, StartPlc, StopPlc)
- Bootprojektoptionen einstellen
- Projekt und/oder Lösung als Archiv speichern
- Aufruf CheckAllObjects()

Allgemeine Informationen über SPS-Projekte

SPS-Projekte werden mit Hilfe ihrer sogenannten Projekt-Templates spezifiziert. Derzeit stellt TwinCAT zwei Templates zur Verfügung, die mittels einer Template-Datei im TwinCAT-Verzeichnis beschrieben sind. Die folgende Tabelle zeigt, welche SPS-Templates verfügbar sind, und die entsprechende Template-Datei:

Template-Name	Template-Datei
Standardmäßiges SPS- Template	C:\TwinCAT\3.x\Components\Plc\PlcTemplate\Plc Templates\Standard PLC Template.plcproj
Leeres SPS- Template	C:\TwinCAT\3.x\Components\Plc\PlcTemplate\Plc Templates\Empty PLC Template.plcproj

Erstellung von und Umgang mit SPS-Projekten

Um ein neues SPS-Projekt mit Hilfe des Automation Interface zu erstellen, müssen Sie zum SPS-Knoten navigieren und dann die CreateChild()-Methode mit der entsprechenden Vorlagendatei als Parameter ausführen.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Bitte beachten

Stellen Sie bei der Verwendung von Standard-SPS-Vorlagen, wie sie von Beckhoff bereitgestellt werden, sicher, dass Sie nur den Vorlagennamen statt des gesamten Pfads, z. B. „Standard-SPS-Vorlage“, verwenden.

Alle nachfolgenden Operationen, wie Erstellung von und Umgang mit POUs, sowie entsprechende Code-Eingabe werden in einem getrennten Artikel beschrieben.

Nachdem das SPS-Projekt erstellt ist, kann es weiter verwendet werden, indem es in die spezielle Schnittstelle ITcPlcIECProject [▶ 168] umgewandelt wird, die mehr Funktionalitäten und den Zugriff auf die projektspezifischen Attribute gewährt:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Das "iecProject"-Objekt kann jetzt dazu verwendet werden, um auf die Methoden der ITcPlcIECProject-Schnittstelle zuzugreifen, um z.B. das SPS-Projekt als eine SPS-Bibliothek abzuspeichern.

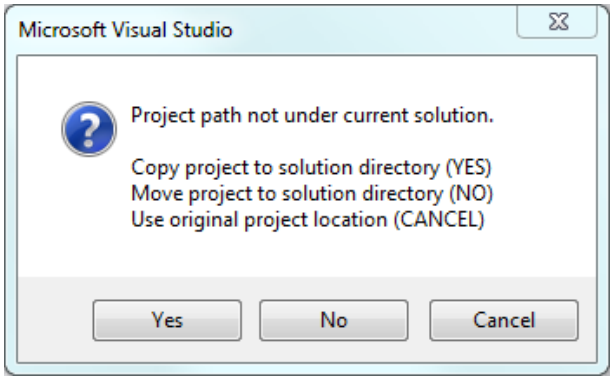
Bestehende SPS-Projekte öffnen

Zum Öffnen eines bestehenden SPS-Projekts mit Hilfe von Automation Interface, müssen Sie zum SPS-Knoten navigieren und dann die CreateChild()-Methode mit dem Pfad der entsprechenden SPS-Projektdatei als Parameter ausführen.

Sie können drei verschiedene Werte als SubType verwenden:

- 0: Projekt zum Lösungsverzeichnis kopieren
- 1: Projekt zum Lösungsverzeichnis verschieben
- 2: Original-Projektspeicherort verwenden (falls verwendet, verwenden Sie bitte "" als Projektnamensparameter)

Grundsätzlich repräsentieren diese Werte die Funktionalitäten (Ja, Nein, Abbrechen) von der folgenden MessageBox in TwinCAT XAE:



Anstelle der Vorlagendatei müssen Sie den Pfad zum SPS-Projekt (bzw. dessen plcproj Datei) verwenden, das hinzugefügt werden muss. Alternativ können Sie auch ein SPS-Projektarchiv (tpzip Datei) verwenden.

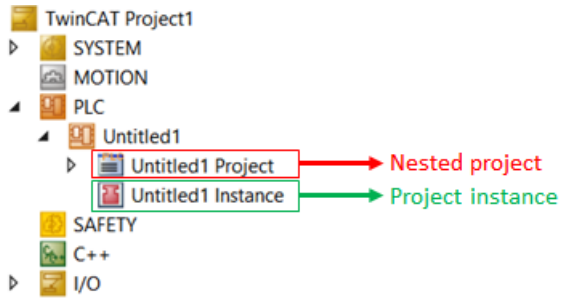
Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

TwinCAT SPS-Projekte bestehen aus zwei verschiedenen Bereichen – dem sogenannten „nested Projekt“ und der Projektinstanz. Das nested Projekt (Tree Item-Subtyp 56) enthält den Quellcode des SPS-Programms, wohingegen die Projektinstanz die deklarierten Ein- und Ausgangsvariablen des SPS-Programms enthält.



Der folgende Code-Ausschnitt zeigt einen gemeinsamen Weg, um im Allgemeinen auf beide Tree Items zuzugreifen, wenn der vollständige Pfadname nicht bekannt ist.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Bitte beachten

Ein Minimum TwinCAT 3.1 Build 4018 ist erforderlich, um auf die Schnittstelle ITcProjectRoot zuzugreifen.

Das SPS-Projekt als Bibliothek speichern

Um ein SPS-Projekt als eine SPS-Bibliothek abzuspeichern, müssen Sie die Methode ITcPlcIECProject [▶ 168]::SaveAsLibrary() [▶ 171] verwenden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Der zweite Parameter bestimmt, ob die Bibliothek, nachdem sie als Datei gespeichert wurde, in das standardmäßige Repository installiert wird.

Umgang mit den Online-Funktionalitäten (Login, StartPlc, StopPlc, ResetCold, ResetOrigin)

Erforderliche Version: TwinCAT 3.1 Build 4010 und höher

Das Automation Interface bietet Ihnen zudem Online-SPS-Funktionen, um sich z.B. in eine SPS-Laufzeit einzuloggen und das SPS-Programm zu starten/stoppen/zurückzusetzen. Auf diese Features kann man über die ITcSmTreeItem [▶ 127]::ProduceXml() [▶ 158] und ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] Methoden zugreifen. Diese Funktionen können auf einen ITcPlcIECProject [▶ 168]-Knoten angewendet werden.

XML-Struktur:

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

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

Die folgende Tabelle beschreibt jeden XML-Knoten im Einzelnen:

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

Bitte beachten: Um Befehle wie ResetOriginCmd benutzen zu können, müssen Sie vorher einen LoginCmd ausführen – ähnlich wie TwinCAT XAE.

Bootprojektoptionen einstellen

Der folgende Code-Ausschnitt zeigt, wie die ITcPlcProject Schnittstelle verwendet wird, um Bootprojektoptionen für ein SPS-Projekt einzustellen.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Projekt und/oder Lösung als Archiv speichern

Zum Speichern der gesamten TwinCAT Lösung in einer ZIP-kompatiblen Archivdatei (*.tszip) kann die ITcSysManager9 Schnittstelle verwendet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Zum erneuten Laden einer zuvor gespeicherten TSZIP-Datei kann die DTE Methode AddFromTemplate() verwendet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Zum Speichern eines spezifischen SPS-Projekts in einer ZIP-kompatiblen Archivdatei (*.tpzip) kann die Methode ITcSmTreeItem::ExportChild() verwendet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Zum erneuten Laden einer zuvor gespeicherten TSZIP-Datei kann die ITcSmTreeItem::CreateChild() Methode verwendet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Aufruf CheckAllObjects()

Zum Aufruf der CheckAllObjects() Methode auf einem Nested SPS-Projekt können Sie die entsprechende Methode verwenden, die in der Schnittstelle ITcPlcIECProject2 verfügbar ist.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.3.5.2 Zugriff auf, Erstellung von und Umgang mit SPS-Bibliotheken und -Platzhaltern

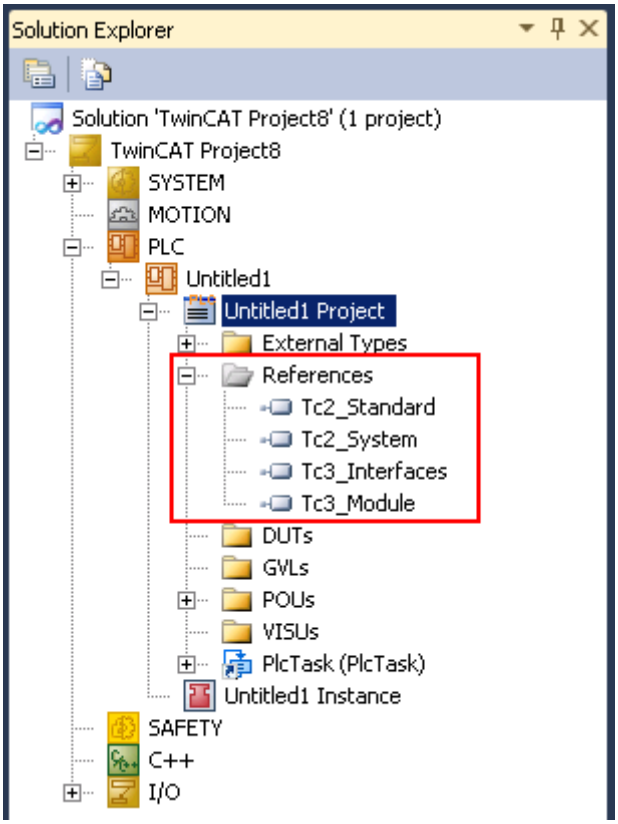
In diesem Kapitel wird der Zugriff auf und der Umgang mit SPS-Bibliotheken und SPS-Platzhaltern ausführlich erläutert. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über SPS-Bibliotheken und -Platzhalter
- Durch Referenzen navigieren
- Referenzen hinzufügen
- Referenzen entfernen
- Verfügbare Bibliotheken durchsuchen
- Platzhalterversion einfrieren
- Mit Repositories arbeiten

Allgemeine Informationen über SPS-Bibliotheken und -Platzhalter

In TwinCAT 3 gibt es zwei Bibliotheks-Objekttypen: Bibliotheken und Platzhalter. Weitere Informationen über beide Typen finden Sie in der TwinCAT 3-Dokumentation über Bibliotheksverwaltung.

In einem TwinCAT 3 SPS-Projekt werden die Referenzen zu Bibliotheken und Platzhaltern als untergeordnete Tree Items zum Referenzenknoten unterhalb des entsprechenden SPS-Projekts hinzugefügt. Bei Auswahl der Vorlage "Standard PLC Project" werden dem Projekt standardmäßig bestimmte Bibliotheken und Platzhalter hinzugefügt, z.B. Tc2_Standard, Tc2_System, ...



Bei Verwendung des Automation Interface können Sie mit Hilfe der Methode ITcSysManager [▶ 118]::LookupTreeItem() [▶ 127] einfach durch den Referenzenknoten navigieren.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Damit Sie dieses Objekt korrekt handhaben können, muss es einer der ITcPlcLibraryManager-Schnittstelle entsprechenden Typumwandlung unterzogen werden.

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

Achten Sie darauf, dass dieser Schritt in Windows Powershell nicht erforderlich ist.

Durch Referenzen navigieren

Sie können alle Referenzen durchlaufen unter Verwendung der ITcPlcLibraryManager [▶ 171]::References Eigenschaft. Diese Eigenschaft gibt eine ITcPlcReferences-Sammlung zurück von entweder Bibliotheksobjekten (dargestellt durch ITcPlcLibrary [▶ 178]) oder Platzhalterobjekten (dargestellt durch ITcPlcPlaceholderRef).

Code-Ausschnitt (C#):

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

Das Objekt libRef; das für die Iteration verwendet wird, ist vom Typ ITcPlcLibRef. Dies ist eine gemeinsame Basisklasse für ITcPlcLibrary und ITcPlcPlaceholderRef-Objekte. Um mit einer dieser spezifischen Klassen arbeiten zu können, müssen wir das Objekt einer entsprechenden Typumwandlung unterziehen, wie oben im Code-Ausschnitt gezeigt.

Code-Ausschnitt (Powershell):

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

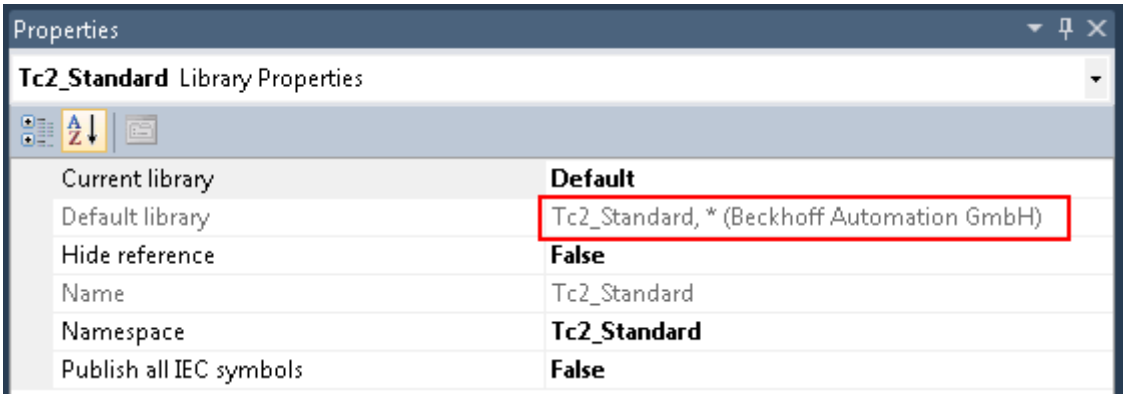
Referenzen hinzufügen

Die Klasse ITcPlcLibraryManager [▶ 171] bietet zwei Methoden, mit denen einem SPS-Projekt eine Bibliotheks- oder Platzhalterreferenz hinzugefügt werden kann: AddLibrary() und AddPlaceholder().

Eine Bibliothek kann auf zwei verschiedene Weisen hinzugefügt werden:

- Entweder mittels Angabe des Namens, der Version und des Verteilers der Bibliothek
- oder mittels des Anzeigenamens der Bibliothek
- oder (im Falle eines Platzhalters) mit dem Platzhalternamen.

Der Anzeigename einer Bibliothek kann im Eigenschaftenfenster der Bibliothek oder des Platzhalters bestimmt werden:



Bibliotheken hinzufügen:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Ein Platzhalter kann auf zwei verschiedene Weisen hinzugefügt werden:

- Entweder mittels Angabe des Platzhalternamens, des Namens, der Version und des Verteilers der Bibliothek
- oder unter Verwendung des Platzhalternamens, wenn der Platzhalter bereits existiert

Platzhalter hinzufügen:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Bitte beachten: Beim Hinzufügen eines neuen Platzhalters bestimmen die Parameter der AddPlaceholder()-Methode dessen standardmäßige Auflösung. Um die effektive Auflösung festzulegen, einfach die Methode ITcPlcLibraryManager [▶ 171]::SetEffectiveResoltion() verwenden.

Referenzen entfernen

Um eine Referenz zu entfernen, einfach die Methode ITcPlcLibraryManager [▶ 171]::RemoveReference() verwenden. Weil diese Methode ITcPlcLibRef-Elemente bearbeitet (welche die Basisklasse für ITcPlcLibrary und ITcPlcPlaceholderRef-Objekte ist), können Sie diese Methode für beide, Bibliotheks- und Platzhalterreferenzen verwenden.

Bibliotheksreferenzen können entweder unter Angabe ihres Namens, ihrer Version und ihres Verteilers oder unter Angabe ihres Anzeigenamens entfernt werden.

Platzhalterreferenzen können unter Angabe ihres Platzhalternamens entfernt werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Verfügbare Bibliotheken durchsuchen

Um das System nach allen verfügbaren SPS-Bibliotheken zu durchsuchen, benutzen Sie einfach die Methode ITcPlcLibraryManager [▶ 171]::ScanLibraries() [▶ 176]. Diese Methode gibt eine ITcPlcReferences-Sammlung von Bibliotheken zurück (Typ ITcPlcLibrary).

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Platzhalterversion einfrieren

Es ist möglich, die verwendete Version eines Platzhalters auf eine bestimmte Version einzufrieren. Dies kann mit der Methode ITcPlcLibraryManager [▶ 171]::FreezePlaceholder() erzielt werden. Diese Methode wird auf ein Objekt aufgerufen, das auf den Referenzenknoten zeigt.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Bitte beachten: Die Version wird mit der effektiven Auflösung eingefroren. Wenn die effektive Auflösung auf "*" zeigt, dann wird die neueste Version im System verwendet.

Mit Repositories arbeiten

Das Automation Interface bietet eine Methode für den Umgang mit SPS-Bibliotheks-Repositories. Ein standardmäßiges Repository ist Teil einer jeden TwinCAT-Installierung. Um zusätzliche Repositories zu erstellen, können Sie die Methode ITcPlcLibraryManager [▶ 171]::InsertRepository() [▶ 174] verwenden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Beim Installieren einer neuen Bibliothek in das System, muss die Bibliothek Teil eines Repository sein. Diese Einfügung kann mit Hilfe der Methode ITcPlcLibraryManager [▶ 171]::InstallLibrary() [▶ 174] vorgenommen werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Um eine Bibliothek aus dem Repository zu entfernen, verwenden Sie die Methode ITcPlcLibraryManager [▶ 171]::UninstallLibrary() [▶ 177].

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Um ein Repository zu entfernen, verwenden Sie die Methode ITcPlcLibraryManager [▶ 171]::RemoveRepository() [▶ 175].

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.3.5.3 Zugriff, Erstellung und Umgang mit SPS-POUs

In diesem Kapitel wird ausführlich beschrieben, wie auf SPS-Objekte, z.B. POUs, Methoden, Transitionen, Eigenschaften und auf deren entsprechenden Implementierungs- und Deklarationsbereiche für die Handhabung von SPS-Code zugegriffen werden kann. Auch der Import/Export von SPS-Objekten in PLCopen XML wird behandelt. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über SPS-Objekte
- Zugriff auf Implementierungs- / Deklarationsbereich einer POU
- Auf Sub-POUs (Aktionen, Eigenschaften, ...) zugreifen
- SPS-Objekte erstellen
- SPS-Zugriffsbezeichner
- PLCopen XML Import/Export
- Bestehende POUs (Vorlagen) importieren

Allgemeine Informationen über SPS-Objekte

Auch wenn jedes Tree Item als vom Typ ITcSmTreeItem [▶ 127] betrachtet wird, müssen bestimmte Elemente in eine speziellere Schnittstelle umgewandelt werden, damit der Zugriff auf alle ihre Methoden und Eigenschaften möglich wird, zum Beispiel POUs, die in die Schnittstelle ITcPlcPou [▶ 166] umgewandelt werden müssen, um Zugang zu deren einmaligen Methoden und Eigenschaften zu erlangen.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

In diesem Beispiel wird die POU *MAIN* im SPS-Projekt erzeugt. Das Objekt *programPou* referenziert diese POU und kann nun dazu verwendet werden, um auf spezifische Methoden und Eigenschaften der POU mit Hilfe der entsprechenden Methode/Eigenschaft der ITcPlcPou-Schnittstelle zuzugreifen.

Zugriff auf Implementierungs- / Deklarationsbereich einer POU

Sie können auch Lese-/Schreibzugriff auf entweder den Implementierungs- oder den Deklarationsbereich einer POU erlangen, mit Hilfe der Schnittstellen ITcPlcImplementation [▶ 167] oder ITcPlcDeclaration [▶ 167], wie im folgenden Beispiel gezeigt.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Beim Vergleichen der beiden Schnittstellen werden Sie feststellen, dass sich die zugänglichen Eigenschaften in beiden Schnittstellen voneinander unterscheiden. So bietet z.B. die ITcPlcImplementation-Schnittstelle eine Eigenschaft "Sprache", welche ITcPlcDeclaration nicht bietet. Der Grund hierfür ist, dass gemäß IEC der Deklarationsbereich nicht auf eine reale Programmiersprache basiert (z.B. ST), so dass diese Eigenschaft keinen Sinn ergäbe, wenn sie im Deklarationsbereich verwendet würde.

Auf Sub-POUs (Aktionen, Eigenschaften, ...) zugreifen

POUs können mehrere Unterelemente haben, wie Methoden, Transitionen, Aktionen und Eigenschaften. Es ist wichtig zu verstehen, dass nicht jedes Unterelement einer POU auch beides, einen Implementierungs- und einen Deklarationsbereich, aufweist. Aktionen und Transitionen z.B. haben nur einen Implementierungsbereich. Demzufolge ist die Umwandlung in eine dieser erwähnten Schnittstellen nur gültig, wenn das entsprechende Unterobjekt diesen Bereich aufweist. Die folgende Tabelle gibt einen Überblick darüber, welche Bereiche in den verschiedenen POU-Typen verfügbar sind.

Tree item	Typ	Deklarationsbereich	Implementierungsbereich
Programm	POU	Ja	Ja
Funktion	POU	Ja	Ja
Funktionsbaustein	POU	Ja	Ja
Aktion	POU	Nein	Ja
Methode	POU	Ja	Ja
Eigenschaft (Abrufen/ Setzen)	POU	Ja	Ja
Transition	POU	Nein	Ja
Enum	DUT	Ja	Nein
Struct	DUT	Ja	Nein
Union	DUT	Ja	Nein
Alias	DUT	Ja	Nein
Schnittstelle	Schnittstelle	Ja	Nein
Eigenschaft (Abrufen/ Setzen)	Schnittstelle	Ja	Ja
Globale Variablen	GVL	Ja	Nein

SPS-Objekte erstellen

Die Erstellung von SPS-Objekten ist einfach und kann mit der Methode CreateChild() [▶ 160] der ITcSmTreeItem [▶ 127]-Schnittstelle vorgenommen werden. Die Parameter dieser Methode müssen je nach POU-Typ unterschiedlich interpretiert werden. In der folgenden Tabelle finden Sie die notwendigen Informationen, um unterschiedliche POU-Typen zu erstellen. Achten Sie darauf, dass der vInfo-Parameter gegebenenfalls eine Zeichenkette ist, wenn mehr als ein Parameter benötigt wird. In [...] wird auf jede Arrayposition verwiesen, und darauf, ob sie optional ist oder nicht.

Tree Item	Typ	Parameter "nSubType"	Parameter "vInfo"
Programm	POU	602	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Kann zur Ableitung verwendet werden (Schlüsselwörter “Extends" oder "Implements"). Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld das Schlüsselwort "Extends" spezifiziert. **[2, optional]:** Name der Schnittstelle oder POU, die zu erweitern/implementieren ist (obligatorisch, wenn [1] verwendet wird). Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld die zu erweiternde Bibliothek spezifiziert. **[3, optional]:** Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld das Schlüsselwort "Implements" spezifiziert. **[4, optional]:** Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld die Schnittstelle spezifiziert, die für die Ableitung verwendet wird.
Funktion	POU	603	**[0]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. **[1]:** Rückgabetyp der Funktion. Kann ein SPS-Datentyp sein, z.B. DINT, BOOL, ...
Funktionsbaustein	POU	604	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Kann zur Ableitung verwendet werden (Schlüsselwörter “Extends" oder "Implements"). Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld das Schlüsselwort "Extends" spezifiziert. **[2, optional]:** Name der Schnittstelle oder POU, die zu erweitern/implementieren ist (obligatorisch, wenn [1] verwendet wird). Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld die zu erweiternde Bibliothek spezifiziert. **[3, optional]:** Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld das Schlüsselwort "Implements" spezifiziert. **[4, optional]:** Wenn die Schlüsselwörter "Extends" UND "Implements" verwendet werden sollen, wird in diesem Feld die Schnittstelle spezifiziert, die für die Ableitung verwendet wird.
Aktion	POU	608	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Kann gegebenenfalls PLCopen XML-Zeichenkette mit Code für die Aktion beinhalten
Methode	POU	609	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Rückgabetyp **[2, optional]:** Zugriffsbezeichner, z.B. PUBLIC. PUBLIC wird standardmäßig verwendet. **[3, optional]:** Kann gegebenenfalls PLCopen XML-Zeichenkette mit Code für die Aktion beinhalten

Tree Item	Typ	Parameter "nSubType"	Parameter "vInfo"
Eigenschaft	POU	611	**[0]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert **[1]:** Rückgabetyp **[2, optional]:** Zugriffsbezeichner, z.B. PUBLIC. PUBLIC wird standardmäßig verwendet.
Eigenschaft Abrufen	POU	613	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Zugriffsbezeichner, z.B. PUBLIC. PUBLIC wird standardmäßig verwendet. **[2, optional]:** Kann gegebenenfalls PLCopen XML-Zeichenkette mit Code für die Aktion beinhalten
Eigenschaft Setzen	POU	614	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Zugriffsbezeichner, z.B. PUBLIC. PUBLIC wird standardmäßig verwendet. **[2, optional]:** Kann gegebenenfalls PLCopen XML-Zeichenkette mit Code für die Aktion beinhalten
Transition	POU	616	**[0, optional]:** IEC-Programmiersprache, wie durch IECLANGUAGETYPES [▶ 166] definiert. ST (Strukturierter Text) wird standardmäßig verwendet. **[1, optional]:** Kann gegebenenfalls PLCopen XML-Zeichenkette mit Code für die Aktion beinhalten
Enum	DUT	605	**[0, optional]:** Kann gegebenenfalls Deklarationstext beinhalten
Struct	DUT	606	**[0, optional]:** Kann gegebenenfalls Deklarationstext beinhalten
Union	DUT	607	**[0, optional]:** Kann gegebenenfalls Deklarationstext beinhalten
Alias	DUT	623	**[0, optional]:** Kann gegebenenfalls Deklarationstext beinhalten
Schnittstelle	Schnittstelle	618	**[0, optional]:** Typ erweitern
Eigenschaft	Schnittstelle	612	**[0]:** Rückgabetyp
Eigenschaft Abrufen	Schnittstelle	654	Kein vInfo-Parameter erforderlich, "Null" kann verwendet werden.
Eigenschaft Setzen	Schnittstelle	655	Kein vInfo-Parameter erforderlich, "Null" kann verwendet werden.
Methode	Schnittstelle	610	**[0]:** Rückgabetyp
Globale Variablen	GVL	615	**[0, optional]:** Kann gegebenenfalls Deklarationstext beinhalten
SPS-Ordner	Ordner	601	Kein vInfo-Parameter erforderlich, "Null" kann verwendet werden.
Parameterliste	PL	629	**[0, optional]:** Kann gegebenenfalls Deklarationstext beinhalten
UML Klassendiagramm	POU	631	**---**

Beispiel: Der folgende Code-Ausschnitt zeigt, wie der vInfo-Parameter für die Erstellung eines Funktionsbausteins "FB_Test" verwendet werden kann, mit den Schlüsselwörtern “extends” und/oder “implements”, in Abhängigkeit des Werts der booleschen Variablen *bExtends* und *bImplements*. Die erweiterte Bibliothek ist *ADSRDSTATE* und die implementierte Schnittstelle ist *ITcADI*.

Code-Ausschnitt (C#):

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

if (bExtends && bImplements)
{
  vInfo= new string[5];
}
else
{
  if (bExtends || bImplements)
  {
    vInfo= new string[3];
  }
  else
  {
    vInfo= new string[1];
  }
}
vInfo[0] = language.AsString();
if (bExtends && bImplements)
{
  vInfo[1] = "Extends";
  vInfo[2] = "ADSRDSTATE";
  vInfo[3] = "Implements";
  vInfo[4] = "ITcADI";
}
else
{
  if (bExtends)
  {
    vInfo[1] = "Extends";
    vInfo[2] = "ADSRDSTATE";
  }
  else
  {
    if (bImplements)
    {
      vInfo[1] = "Implements";
      vInfo[2] = "ITcADI";
    }
  }
}

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

Bitte beachten

Wenn der Parameter vInfo nur einen Wert beinhaltet (mit [0] in Tabelle oben gekennzeichnet), dann sollten Sie kein Array der Größe 1, sondern einfach eine normale Variable erstellen. Beispiel: Bei der Verwendung von nSubType 618 (Schnittstelle) sollten Sie vInfo folgendermaßen verwenden.

Code-Ausschnitt (C#):

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

SPS-Zugriffsbezeichner

Die folgenden Zugriffsbezeichner sind gültig und können als vInfo-Parameter verwendet werden, wie oben in der Tabelle gezeigt: PUBLIC, PRIVATE, PROTECTED, INTERNAL.

PLCopen XML Import/Export

Sie können auch SPS-Elemente in PLCopen XML über die ITcPlcIECProject [▶ 168]-Schnittstelle importieren/exportieren. Methoden und Eigenschaften dieser Schnittstelle können nur direkt auf einen SPS-Projektknoten ausgeführt werden, z.B.:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Dieser Code-Ausschnitt setzt voraus, dass bereits der TwinCAT-Konfiguration ein SPS-Projekt hinzugefügt wurde, das über das Objekt *plcProject* referenziert wird. Diese Referenz wird in das Objekt *importexport* vom Typ ITcPlcIECProject umgewandelt, welches dann für den Export der POUs POUProgram und POUFunctionBlock (beide befinden sich im SPS-Ordner "MyPOUs") in eine XML-Datei und deren anschließenden erneuten Import verwendet wird.

Die verfügbaren Optionen für den Import sind: Kein (0), Umbenennen (1), Ersetzen (2), Überspringen (3)

Bestehende POUs (Vorlagen) importieren

SPS-Vorlagen sind auf zwei Arten verfügbar: Sie können entweder das komplette SPS-Projekt als Ganzes verwenden oder jede POU einzeln als Vorlage. Das Letztere wird in diesem Kapitel behandelt. Einer der Gründe dafür, warum sich ein Entwickler für einzelne POUs als Vorlagen entscheiden kann, besteht darin, dass es einfacher ist, einen Pool von bestehenden Funktionalitäten aufzubauen und in separaten POUs zusammenzufassen, z. B. verschiedene Funktionsblöcke, die verschiedene Sortieralgorithmen abdecken. Bei der Projekterstellung wählt der Entwickler einfach die Funktionalität aus, die er in seinem TwinCAT-Projekt benötigt und greift auf das Vorlagetool zu, um die entsprechende POU abzurufen und diese in das SPS-Projekt zu importieren.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Bitte beachten: Die POU Vorlagedatei kann nicht nur eine .TcPou Datei sein, sondern auch die entsprechenden Dateien für DUTs und/oder GVLs. Das vorstehende Beispiel gibt zwei übliche Wege wieder, um POU Vorlagedateien zu importieren. Der erste besteht darin, eine einzige Datei zu importieren, der zweite, mehrere Dateien auf einmal zu importieren, indem die Dateipfade in einem String-Array gespeichert werden und dieses String-Array als vInfo Parameter von CreateChild() verwendet wird.

4.3.6 I/O

4.3.6.1 Erstellung von und Umgang mit EtherCAT-Teilnehmern

In diesem Artikel wird beschrieben, wie eine EtherCAT-Topologie mit Hilfe des TwinCAT-Automation Interface aufgebaut wird. Er besteht aus folgenden Themen:

- Allgemeine Informationen
- Einen EtherCAT-Teilnehmer erstellen
- EtherCAT-Boxen erstellen und in Topologie einfügen
- EtherCAT-Klemmen erstellen und in Topologie einfügen
- Ausnahmen zum ItemSubType 9099

- Den “Vorherigen Anschluss” einer EtherCAT-Klemme ändern
- Einer HotConnect-Gruppe EtherCAT-Slaves hinzufügen
- Hinzufügen von EtherCAT SyncUnits
- Umgang mit EtherCAT Abzweigdosen (CU1128)

Alle diese Themen behandeln den Fall einer **Offline**-Konfiguration, d.h. die realen Adressen aller Geräte sind zum Zeitpunkt der Konfiguration nicht bekannt. Im letzten Kapitel dieses Artikels wird daher auch erläutert, wie Sie:

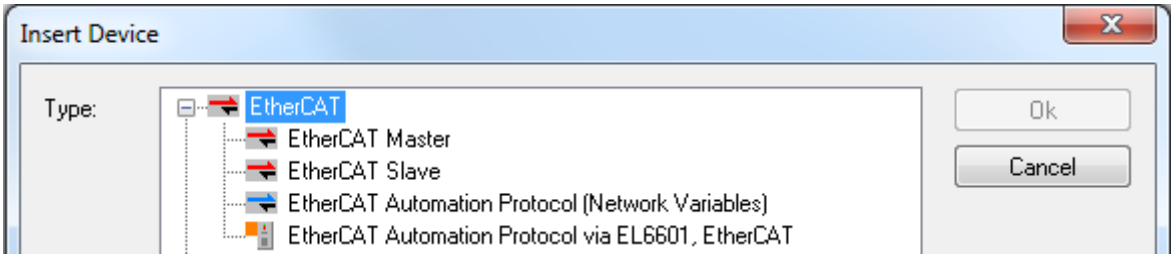
- Die Konfiguration aktivieren

Allgemeine Informationen

Diese Dokumentation sollte Ihnen einen Überblick darüber verschaffen, wie EtherCAT-Teilnehmer und deren entsprechende Topologie über das Automation Interface erzeugt und gehandhabt werden. Für ein tieferes Verständnis der Arbeitsweise von EtherCAT und wie es im Allgemeinen in TwinCAT System Manager/XAE integriert ist, lesen Sie bitte die EtherCAT-Systemdokumentation und das entsprechende Kapitel EtherCAT-Konfiguration in TwinCAT. EtherCAT-Boxen und -Klemmen, die mit einem EtherCAT-Master verbunden sind, werden auf die gleiche Art und Weise erstellt, wie in einem getrennten Artikel über E-Bus-Subtypen [▶ 132] erläutert wird.

Einen EtherCAT-Teilnehmer erstellen

Der erste Schritt bei der Erstellung einer TwinCAT EtherCAT-Konfiguration besteht in der Erstellung eines EtherCAT-Teilnehmers, die möglicherweise die Erstellung eines EtherCAT-Masters, -Slaves oder eines Automatisierungsprotokolls beinhaltet (z.B. um Netzwerkvariablen zu verwenden, wie dies in einem getrennten Artikel [▶ 77] beschrieben wird). Um einen EtherCAT-Master/Slave zu erstellen, verwenden Sie einfach die ITcSmTreeItem [▶ 160]::CreateChild() [▶ 160]-Methode mit dem entsprechenden Parameter für SubType (EtherCAT Master = 111, EtherCAT Slave = 130).



EtherCAT-Master – Code-Ausschnitt (C#)

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

EtherCAT-Master – Code-Ausschnitt (Powershell)

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

EtherCAT-Slave – Code-Ausschnitt (C#)

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

EtherCAT-Slave – Code-Ausschnitt (Powershell)

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

EtherCAT-Boxen erstellen

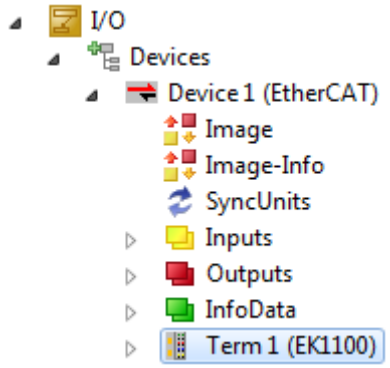
Der zweite Schritt beinhaltet die Erstellung von EtherCAT-Boxen, zum Beispiel einen EK1100 EtherCAT-Koppler. Wie im Artikel über E-Bus SubTypes [▶ 132] erklärt, verwenden alle untergeordneten Tree Items (es gibt wenige Ausnahmen, siehe unten) eines EtherCAT-Masters einen gemeinsamen SubType (9099) und sie werden über die Produkt-Revision identifiziert, die als vInfo-Parameter der Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] übergeben werden muss.

Code-Ausschnitt (C#)

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

Code-Ausschnitt (Powershell)

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



Bitte beachten: Zusätzlich zu einer vollständigen Produkt-Revision können Sie auch einen Platzhalter verwenden. Wenn Sie nur "EK1100" als vInfo spezifizieren, erfasst Automation Interface automatisch die neueste Revisionsnummer und verwendet diese. Beispiel:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

EtherCAT-Klemmen erstellen und in Topologie einfügen

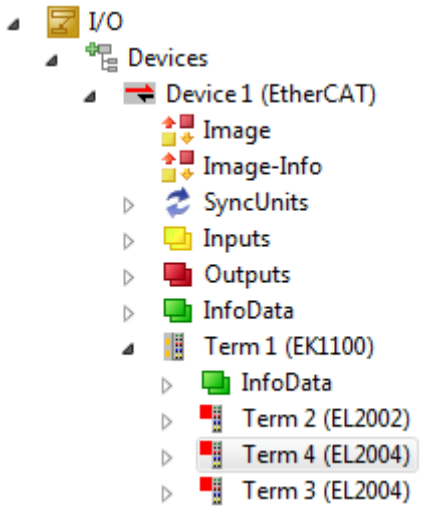
Die Erstellung von EtherCAT-Klemmen basiert auf den gleichen Konzepten wie die Erstellung von EtherCAT-Boxen. Alle Klemmen teilen ebenfalls einen gemeinsamen SubType und werden anhand der Produkt-Revision identifiziert, die als vInfo-Parameter der Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] übergeben werden muss. Der Parameter bstrBefore bietet Ihnen die Möglichkeit, die Position zu bestimmen, an welcher die Klemme in die Konfiguration eingefügt wird.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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



Bitte beachten: Sollten bei Verwendung des bstrBefore-Parameters Probleme auftreten, versuchen Sie bitte die Klemme auf einer “Geräteebene” einzufügen, zum Beispiel:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die neue Klemme wird dann vor "Term 3 (EL2004)" unter der zuletzt eingefügten EtherCAT-Box eingefügt.

Bitte beachten: Zusätzlich zu einer vollständigen Produkt-Revision können Sie auch einen Platzhalter verwenden. Wenn Sie nur "EL2004" als vInfo spezifizieren, erfasst Automation Interface automatisch die neueste Revisionsnummer und verwendet diese. Beispiel:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Ausnahmen zum ItemSubType 9099

Es gibt wenige Ausnahmen zum ItemSubType 9099, z.B. die RS232-Klemmen EP6002 (ItemSubType 9101) und EL600X (ItemSubType 9101). Die folgende Tabelle gibt einen Überblick über alle Ausnahmen und deren entsprechenden ItemSubType.

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

Den “Previous Port” einer EtherCAT-Klemme ändern

Der vorherige Anschluss einer EtherCAT-Klemme bestimmt die Position der Klemme innerhalb der EtherCAT-Topologie.

In TwinCAT XAE schließt das Dropdown-Listenfeld automatisch alle verfügbaren Previous Ports ein. Um diese Einstellung über das Automation Interface zu konfigurieren, können Sie die Methoden ITcSmTreeItem [▶ 127]::ProduceXml() [▶ 158] und ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] für die Bearbeitung der XML-Beschreibung [▶ 25] der entsprechenden EtherCAT-Klemme verwenden. Die XML-Beschreibung beinhaltet hierzu einen oder mehrere XML-Knoten <PreviousPort>, wobei dessen (deren) Attribut "Selected=1" festlegt, welcher vorherige Anschluss derzeit ausgewählt ist. Jedes vorherige Anschlussgerät wird mittels seiner physikalischen Adresse identifiziert.

Beispiel (XML-Beschreibung)

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

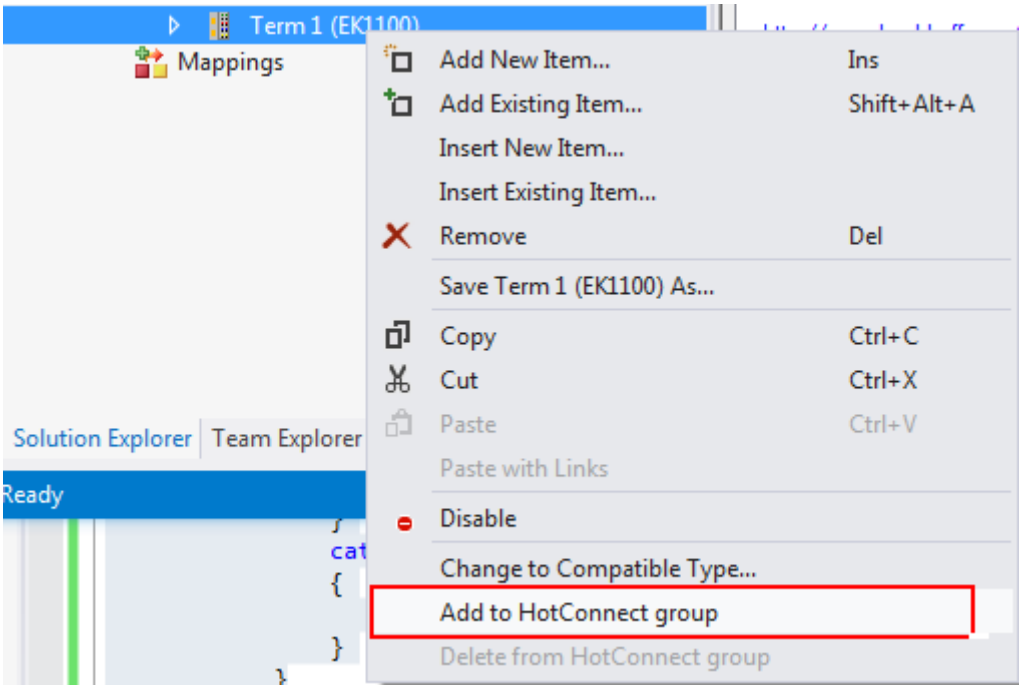
Wenn Sie den vorherigen Anschluss ändern möchten, müssen sie die <PhysAddr> des gewünschten Geräts kennen, welche ebenfalls über die XML-Beschreibung ermittelt werden kann.

Beim Einfügen von Childs in eine EtherCAT-Konfiguration kann der Parameter bstrBefore der ImportChild- oder CreateChild-Methode verwendet werden, um das vorherige Element beim Aufruf der ImportChild- oder CreateChild-Methode auf einem Slave anzugeben. Auf einem EtherCAT-Master muss diese Einstellung über die XML-Beschreibung von oben angegeben werden.

Einer HotConnect-Gruppe EtherCAT-Slaves hinzufügen

Mit Hilfe von EtherCAT HotConnect können vorkonfigurierte Abschnitte dem Datenverkehr vor dem Start oder während des Betriebs des Systems hinzugefügt oder aus ihm entfernt werden. Weitere Informationen über EtherCAT HotConnect finden Sie in unserer EtherCAT-Systemdokumentation.

In TwinCAT XAE kann einer HotConnect-Gruppe ein EtherCAT-Slave hinzugefügt werden, indem die entsprechende Option im Kontextmenü des Geräts angeklickt wird.



ImTwinCAT-Automation Interface kann das gleiche mittels Verwendung der folgenden XML-Struktur auf dem EtherCAT-Slave erzielt werden:

Beispiel (XML-Beschreibung):

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

Bitte beachten:

- Das <GroupMemberCnt>-Tag muss die genaue Anzahl Klemmen plus das Gerät selber angeben.

Einstellung von EtherCAT SyncUnits

EtherCAT SyncUnits können über ITcSmTreeItem::ConsumeXml() und durch Verwendung der folgenden XML-Beschreibung eingestellt werden.

Beispiel (XML-Beschreibung):

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

Umgang mit EtherCAT Abzweigdosen (CU1128)

EtherCAT Abzweigdosen können wie alle anderen Tree Items behandelt werden. Der folgende Beispielcode zeigt nachstehend, wie eine CU1128-Box und anschließend zwei EK1100-Boxen hinzugefügt werden:

Code-Ausschnitt (C#)

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

Code-Ausschnitt (Powershell)

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

Die EtherCAT-Konfiguration aktivieren

Um eine erstellte TwinCAT-Konfiguration über das Automation Interface zu aktivieren, kann die Methode ITcSysManager [▶ 118]::ActivateConfiguration() [▶ 121] verwendet werden. Allerdings wurde in den vorigen Kapiteln lediglich erläutert, wie eine Offline-Konfiguration zu erstellen ist, d.h. dass alle erstellten Geräte noch keine reale Adresse haben, z.B. der EtherCAT-Master wurde noch nicht mit einer physikalischen Netzwerkschnittstellenkarte verbunden. Vor der Aktivierung der Konfiguration müssen Sie deshalb jedes Gerät mit Online-Adressen konfigurieren. Um die realen Adressen zu bestimmen, können Sie ein ScanDevices auf dem Online-System ausführen, die realen Adressen über die XML-Beschreibung (ITcSmTreeItem [▶ 127]::ProduceXml() [▶ 158]) ermitteln und dann die Adressinformation über ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] in die erstellte (Offline-) Konfiguration importieren. Es gibt zwei „Wie...“ Beispiele, die Ihnen bei genau dieser Art der Konfiguration helfen können:

- Scan Devices [▶ 95] über Automation Interface

4.3.6.2 Erstellung von und Umgang mit Netzwerkvariablen

In diesem Kapitel wird die Erstellung von und der Umgang mit Netzwerkvariablen ausführlich erklärt. Die folgende Liste enthält alle Kapitel dieses Artikels:

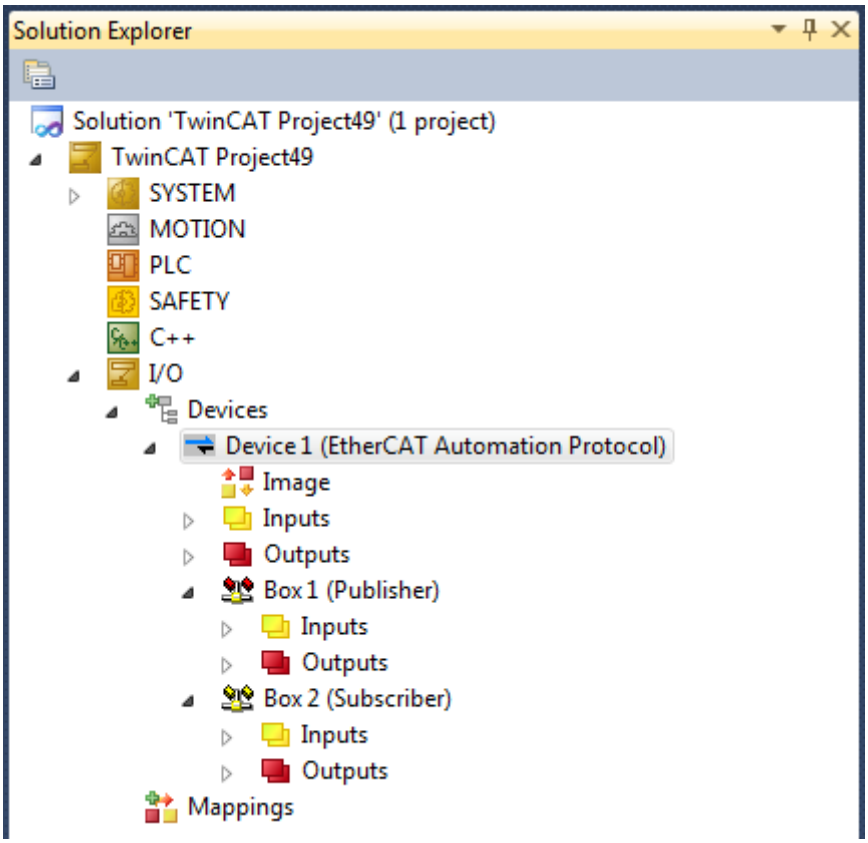
- Allgemeine Informationen über Netzwerkvariablen
- Ein EtherCAT Automation Protocol-Gerät erstellen
- Eine Publisher-Box erstellen
- Eine Subscriber-Box erstellen
- Parameter für eine Publisher-/Subscriber-Box setzen
- Publisher-Variablen erstellen

- Subscriber-Variablen erstellen
- Publisher-/Subscriber-Variablen verknüpfen
- Publisher-/Subscriber-Variablen-IDs lesen

Beachten Sie, dass der Scripting Container [▶ 182] ein ausführliches Beispiel bezüglich der Erstellung und Konfiguration von Netzwerkvariablen mit dem Automation Interface beinhaltet.

Allgemeine Informationen über Netzwerkvariablen

Netzwerkvariablen können dazu verwendet werden, Daten zwischen zwei TwinCAT-Geräten über ein IP-basiertes Netzwerk auszutauschen. Ein Gerät deklariert Variablen als „Publisher“ (Sender) und das andere Gerät empfängt Variablenwerte als „Subsriber“. Aus diesem Grund sprechen wir auch von Publisher-/Subscriber-Variablen. TwinCAT bietet Ihnen die Flexibilität, Netzwerkvariablen direkt innerhalb eines TwinCAT-Projekts zu konfigurieren, so dass Sie diese zu Ihrer SPS oder I/O mappen können.



Netzwerkvariablen verwenden das EtherCAT Automation Protocol-Gerät für die Kommunikation über das lokale Netzwerk. Aus diesem Grunde müssen Sie dieses Gerät hinzufügen, bevor Sie eine Publisher- und/oder Subscriber-Box zusammen mit den entsprechenden Variablen konfigurieren können.

Weitere Informationen über Netzwerkvariablen und wie Sie diese in TwinCAT konfigurieren können, finden Sie hier.

Ein EtherCAT Automation Protocol-Gerät erstellen

Um das EtherCAT Automation Protocol-Gerät zu erstellen, können Sie die Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] zusammen mit dem entsprechenden SubType dieses Geräts (112) verwenden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Eine Publisher-Box erstellen

Die Publisher-Box ist der Container für Publisher-Variablen, die z.B. festlegen, welcher Kommunikationsmustertyp für die darin enthaltenen Variablen zu verwenden ist (Unicast, Multicast, Broadcast). Für das Hinzufügen einer Publisher-Box verwenden Sie einfach erneut die Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] zusammen mit dem entsprechenden SubType (9051).

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Der kleine Code-Ausschnitt fügt eine Publisher-Box zum zuvor erzeugten EtherCAT Automation Protocol-Gerät hinzu. Um das Kommunikationsmuster dieser Box zu konfigurieren, müssen Sie deren Einstellungen mit Hilfe der Methode ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] anpassen. Dies wird ausführlicher im EtherCAT Automation Protocol-Beispiel des Scripting Container [▶ 182] oder weiter unten auf dieser Seite beschrieben.

Eine Subscriber-Box erstellen

Die Subscriber-Box ist der Container für Subscriber-Variablen, die z.B. festlegen, welcher Kommunikationsmustertyp für die darin enthaltenen Variablen zu verwenden ist (Unicast, Multicast, Broadcast). Für das Hinzufügen einer Publisher-Box verwenden Sie einfach erneut die Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] zusammen mit dem entsprechenden SubType (9052).

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Der kleine Code-Ausschnitt fügt eine Subscriber-Box zum zuvor erzeugten EtherCAT Automation Protocol-Gerät hinzu. Um das Kommunikationsmuster dieser Box zu konfigurieren, müssen Sie deren Einstellungen mit Hilfe der Methode ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] anpassen. Dies wird ausführlicher im EtherCAT Automation Protocol-Beispiel des Scripting Container [▶ 182] oder weiter unten auf dieser Seite beschrieben.

Publisher-Variablen erstellen

Nachdem Sie erfolgreich eine Publisher-Box hinzugefügt haben, können Sie nun Publisher-Variablen in diese Box hinzufügen, indem Sie die Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] verwenden zusammen mit den benötigten Parametern für SubType (0) und vInfo, welche den Datentyp der Publisher-Variablen festlegen, z.B. "BOOL".

Code-Ausschnitt (C#):

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

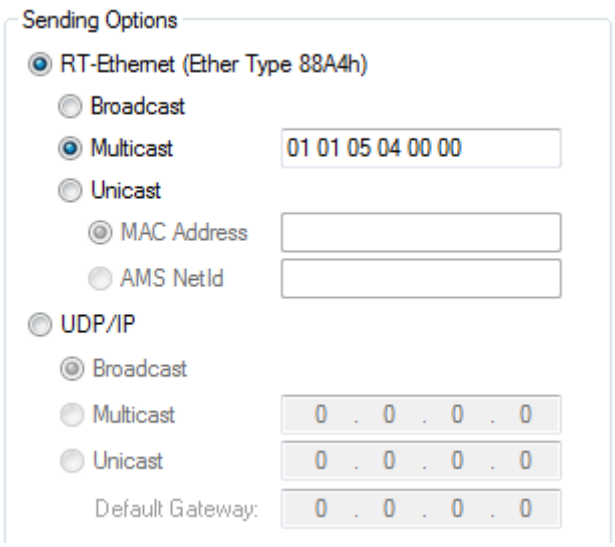
Code-Ausschnitt (Powershell):

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

Siehe auch das Scripting Container [▶ 182]-Beispiel "EtherCAT Automation Protocol" für weitere Informationen.

Parameter für eine Publisher-/Subscriber-Box setzen

Es müssen zwei Kommunikationsmuster auf einer Publisher- und/oder Subscriber-Box konfiguriert werden: **RT-Ethernet** oder **UDP/IP** . Der folgende Screenshot zeigt die entsprechende Konfigurationsregisterkarte von TwinCAT XAE.



Ausführlichere Informationen bezüglich dieser Optionen finden Sie hier.

Um die Box für **RT-Ethernet** zu konfigurieren, müssen Sie die Methode ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] zum Importieren der folgenden XML-Struktur verwenden:

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

Die folgende Tabelle zeigt, wie die **fett** markierten Knoten entsprechend dem gewünschten Kommunikationsmuster angepasst werden müssen.

RT-Ethernet Kommunikations-muster	<PublisherNetId>	<MacAddress>
Broadcast	0.0.0.0.0.0	FF FF FF FF FF FF
Multicast	0.0.0.0.0.0	Muss eine Multicast MAC Adresse enthalten, siehe hier für weitere Informationen.
Unicast - MAC Adresse	0.0.0.0.0.0	Muss eine Unicast MAC Adresse enthalten, siehe hier für weitere Informationen.
Unicast - AmsNetId	Enthält AmsNetId von Zielrechner.	00 00 00 00 00 00

Importieren Sie folgende XML-Struktur, wenn Sie **UDP/IP** verwenden möchten:

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

Die folgende Tabelle zeigt, wie die **fett** markierten Knoten entsprechend dem gewünschten Kommunikationsmuster angepasst werden müssen. Auf dem Knoten <Udp> muss das Attribut “Aktiviert” auf “true” gesetzt werden.

RT-Ethernet Kommunikations-muster	<Address>	<Gateway>
Broadcast	255.255.255.255	0.0.0.0
Multicast	Muss eine Multicast IP-Adresse enthalten. Siehe hier für weitere Informationen.	Enthält gegebenenfalls ein standardmäßiges Gateway, an das die Pakete für’s Routing gesendet werden müssen. Andernfalls auf 0.0.0.0 lassen
Unicast	Muss eine (Unicast) IP-Adresse enthalten. Siehe hier für weitere Informationen.	Enthält gegebenenfalls ein standardmäßiges Gateway, an das die Pakete für’s Routing gesendet werden müssen. Andernfalls auf 0.0.0.0 lassen

Subscriber-Variablen erstellen

Nachdem Sie erfolgreich eine Publisher-Box hinzugefügt haben, können Sie nun Publisher-Variablen in diese Box hinzufügen, indem Sie die Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] verwenden zusammen mit den benötigten Parametern für SubType (0) und vInfo, welche den Datentyp der Publisher-Variablen festlegen, z.B. "BOOL".

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Siehe auch das Scripting Container [▶ 182]-Beispiel "EtherCAT Automation Protocol" für weitere Informationen.

Publisher-/Subscriber-Variablen verknüpfen

Zum Verknüpfen von Publisher-/Subscriber-Variablen mit SPS-Variablen benutzen Sie einfach die Methode ITcSysManager [▶ 118]::Linkvariables() [▶ 122].

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Siehe auch das Scripting Container [▶ 182]-Beispiel "EtherCAT Automation Protocol" für weitere Informationen.

Publisher-/Subscriber-Variablen-IDs lesen

Der folgende Code-Ausschnitt liest alle Variablen-IDs aus einer Publisher-Box und speichert sie in das List-Array "ids". Dies kann auch für Subscriber-Variablen verwendet werden.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.3.6.3 Erstellung von und Umgang mit Profinet-Geräten

In diesem Artikel wird beschrieben, wie Profinet E/A-Geräte mit Hilfe des TwinCAT-Automation Interface erstellt und gehandhabt werden. Die folgenden Themen werden dabei behandelt:

- Erstellung von Profinet-Geräten
- Hinzufügen von Profinet-Boxen
- Hinzufügen von Profinet-Modulen
- Hinzufügen von Profinet-Submodulen

Erstellung von Profinet-Geräten

Zur Erstellung von Profinet E/A-Geräten (Controller/Geräte) kann die Methode ITcSmTreeItem::CreateChild() verwendet werden. Der Subtyp stellt den tatsächlichen Typ ein, der hinzugefügt werden sollte.

Name	Subtyp
Profinet-Controller (RT)	113
Profinet-Controller CCAT (RT)	140
Profinet-Controller EL6631 (RT, EtherCAT)	119
Profinet-Controller EL6632 (RT + IRT, EtherCAT)	126
Profinet-Gerät (RT)	115
Profinet-Gerät CCAT (RT)	142
Profinet-Gerät CCAT (RT + IRT)	143
Profinet-Gerät EL6631 (RT, EtherCAT)	118

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Hinzufügen von Profinet-Boxen

Zur Erstellung von Boxen unterhalb eines Profinet-Geräts kann die Methode ITcSmTreeItem::CreateChild() verwendet werden. Der Subtyp hängt von der Box ab, die hinzugefügt werden soll. Die folgende Tabelle gibt einen Überblick über die möglichen Werte:

Name	Subtyp
BK9102	9125
EK9300	9128
EL6631	9130

Zudem sind einige Kenntnisse über die entsprechende Profinet-GSD-Datei erforderlich, um den vInfo-Parameter korrekt zu nutzen. Der vInfo-Parameter ist aus der folgenden Syntax zusammengesetzt: PathToGSDfile#ModuleIdentNumber#BoxFlags#DAPNumber

Die ModuleIdentNumber kann aus der GSD-Datei heraus bestimmt werden, z. B. über die XML-Struktur <ProfileBody><ApplicationProcess><DeviceAccessPointList><DeviceAccessPointItem ModuleIdentNumber>. Die ModuleIdentNumber ist in der Regel einzigartig. Wenn nicht, spezifiziert die DAPNumber die Position in der DeviceAccessPointList.

Es werden z.Zt. folgende BoxFlags interpretiert:

Name	Wert	Beschreibung
GENERATE_NAME_FROM_PAB	0x0004	Der Profinet-Name wird über das Prozessabbild erstellt
GET_STATIONNAME	0x0400	Es wird der Profinet-Name von TC config verwendet (Strukturelementname)
SET_NOT_IP_TO_OS	0x4000	Nur CE: Profinet-IP wird in OS nicht registriert

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Hinzufügen von Profinet-Modulen

Profinet-Module werden unterhalb des API-Knotens einer Profinet-Box erstellt. Der API-Knoten wird automatisch erstellt, wenn der TwinCAT-Konfiguration eine Profinet-Box hinzugefügt wird. Zum Hinzufügen von Profinet-Modulen kann die Methode ITcSmTreeItem::CreateChild() verwendet werden. Der Subtyp bestimmt die Position der Module innerhalb der <ProfileBody><ApplicationProcess><ModuleList> XML-Struktur der entsprechenden GSD-Datei.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Hinzufügen von Profinet-Submodulen

Profinet-Submodule können unterhalb eines sogenannten Profinet modularen Moduls hinzugefügt werden. Die Handhabung ähnelt sehr dem Hinzufügen von normalen Profinet-Modulen. Der Subtyp bestimmt die Position der Submodule innerhalb der <ProfileBody><ApplicationProcess><ModuleList> XML-Struktur der entsprechenden GSD-Datei.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

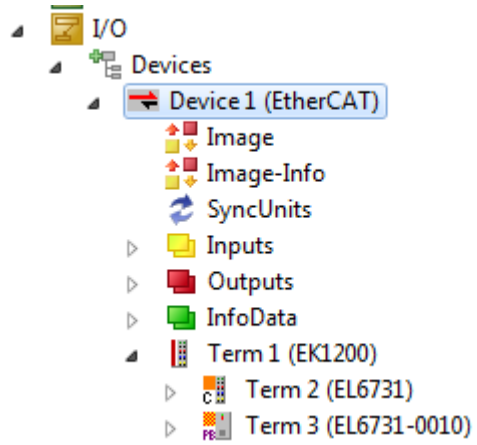
4.3.6.4 Erstellung von und Umgang mit Profibus-Geräten

In diesem Artikel wird beschrieben, wie Profibus-Master- und –Slave-Geräte mit Hilfe des TwinCAT-Automation Interface erstellt und gehandhabt werden. Er setzt sich aus folgenden Schwerpunkten zusammen:

- Einen Profibus-Master erstellen und hinzufügen
- Geeignetes Gerät (EL6731, FC310x, usw.) suchen und konfigurieren
- Einen Profibus-Slave erstellen und hinzufügen
- Geeignetes Slave-Gerät (EL6731-0010, usw.) suchen und konfigurieren
- Änderung der Feldbus-Adresse

Einen Profibus-Master erstellen und hinzufügen

1. Um ein Profibus-Master-Gerät zu erstellen, öffnen Sie eine neue oder bestehende TwinCat-Konfiguration
2. Alle Geräte scannen. (Diese Aktionen können auch über das Automation Interface durchgeführt werden.)



3. Erstellen Sie ein Systemmanager-Objekt und navigieren Sie zu den Geräten

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Verwenden Sie die ITcSmTreeItem:CreateChild Methode, um einen Profibus-Master hinzuzufügen.

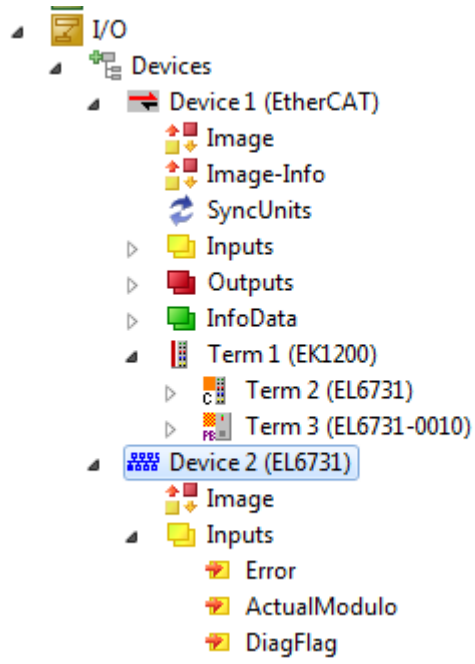
Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

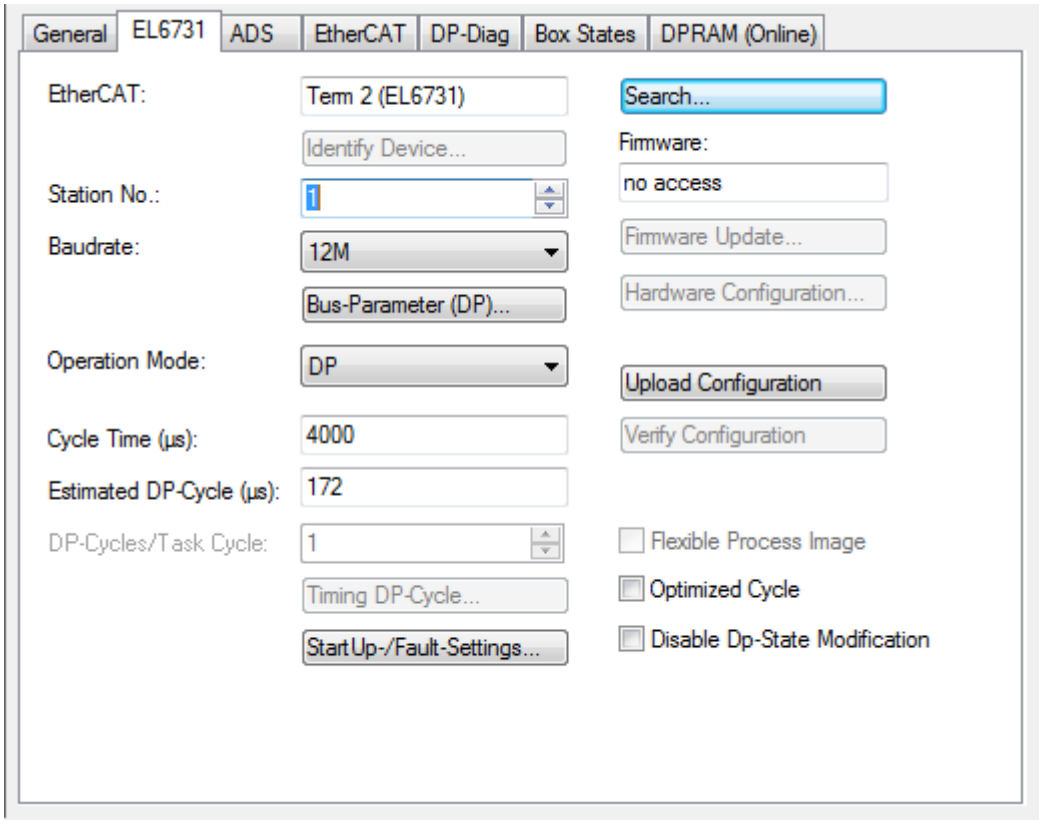
```
$profi_master = $io.CreateChild("Device 2 (EL6731) ", "86", "", $null)
```

Für weitere Profibus-Master geben Sie den korrekten, hier aufgelisteten ItemSubtype ein. Dadurch wird das neue Gerät wie im Screenshot dargestellt hinzugefügt:



Ein Profibus-Master-Gerät in der Liste suchen und anfordern:

Der neu hinzugefügte Profibus-Master muss konfiguriert werden, was normalerweise in TwinCAT mittels Drücken der Suchen-Taste und Auswahl des korrekten Geräts aus der Liste erledigt wird.



Dies kann über Automation Interface gemacht werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

ITcSmTreeItem5:ResourcesCount zeigt die Anzahl kompatibler Profibus-Master-Geräte an, und ITcSmTreeItem5:ClaimResources nimmt den Index des CANOpen-Geräts, das als Master konfiguriert werden soll.

Einen Profibus-Slave erstellen und hinzufügen

Der aktuellen Konfiguration kann ein Profibus-Slave folgendermaßen hinzugefügt werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Einen Profibus Slave suchen und anfordern

Wie im Falle von Profibus-Master kann die Anzahl von Profibus-Slaves durch folgenden Code publiziert werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die letzte Zeile im Code bestimmt das EL6731-0010 Gerät als Profibus-Slave, ähnlich wie im Dialogfeld, das in der TwinCAT-Bedieneroberfläche erscheint.

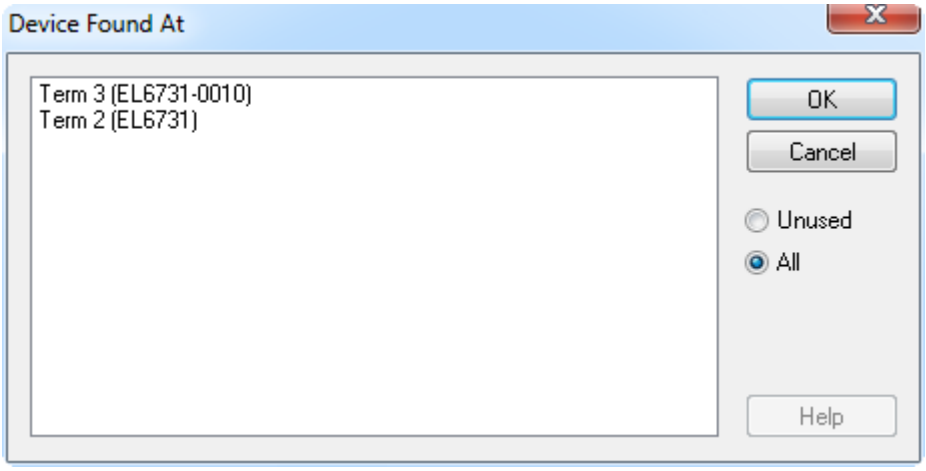


Abb. 2:

Änderung der Feldbus-Adresse

Um die Feldbusadresse (Station-Nr.) einer Profibus-Box in einer Konfiguration zu ändern, muss eine TwinCAT System Manager-Instanz erstellt und die Konfiguration geöffnet werden. Die LookupTreeItem [▶ 124]-Methode der ITcSysManager [▶ 118]-Schnittstelle gibt einen ITcSmTreeItem [▶ 127]-Schnittstellenzeiger zurück, der vom Tree Item, das über seinen Pfadnamen [▶ 10] referenziert wird, implementiert ist. Diese Schnittstelle beinhaltet eine ConsumeXml [▶ 159]-Methode des Tree Items.

Ablauf

Die Vorgehensweise für die Erstellung der ITcSysManager [▶ 118]-Schnittstelle (die '*sysMan*'-Instanz hier) wird im Kapitel Auf TwinCAT-Konfigurationen zugreifen [▶ 20] beschrieben. Diese Schnittstelle verfügt über eine LookupTreeItem [▶ 124]-Methode, die einen ITcSmTreeItem [▶ 127]-Zeiger auf ein, mit seinem Pfadnamen [▶ 10] angegebenes Tree Item zurückgibt. Zum Ändern der Feldbusadresse (Station-Nr.) einer Profibus-Box "*TIID^Device 1 (FC310x)^Box 1 (BK3100)*" in 44, kann der folgende Code verwendet werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

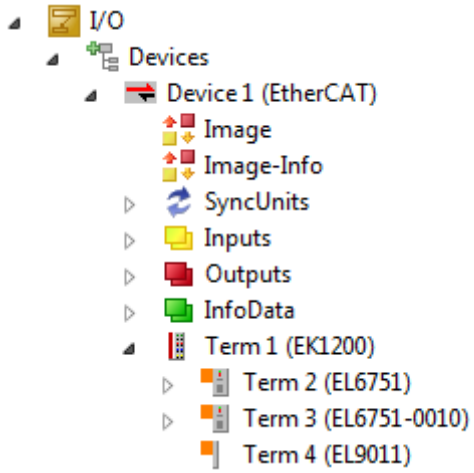
4.3.6.5 Erstellung von und Umgang mit CANOpen-Geräten

In diesem Artikel wird beschrieben, wie CANOpen-Master- und –Slave-Geräte mit Hilfe des TwinCAT-Automation Interface erstellt und gehandhabt werden. Er setzt sich aus folgenden Schwerpunkten zusammen:

- Einen CANOpen-Master erstellen und hinzufügen
- Geeignete Geräte (EL6751, FC510x usw.) suchen und konfigurieren
- Einen CANOpen-Slave erstellen und hinzufügen
- Geeignete Geräte (EL6751-0010 usw.) suchen und konfigurieren
- DBC-Dateien über Automation Interface importieren

Einen CANOpen-Master erstellen und hinzufügen

Zum Erstellen eines CANOpen-Master-Geräts öffnen Sie eine neue oder bestehende TwinCat-Konfiguration und scannen alle Geräte. Denken Sie daran, dass diese Aktionen auch über das Automation Interface durchgeführt werden können.



Erstellen Sie ein Systemmanager-Objekt und navigieren Sie zu den Geräten.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Verwenden Sie die ITcSmTreeItem:CreateChild Methode, um einen CANOpen-Master hinzuzufügen:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Geben Sie hier für andere CANOpen-Master-Geräte den korrekten hier [▶ 134] aufgelisteten ItemSubtype an. Dadurch wird das neue Gerät wie im Screenshot dargestellt hinzugefügt:

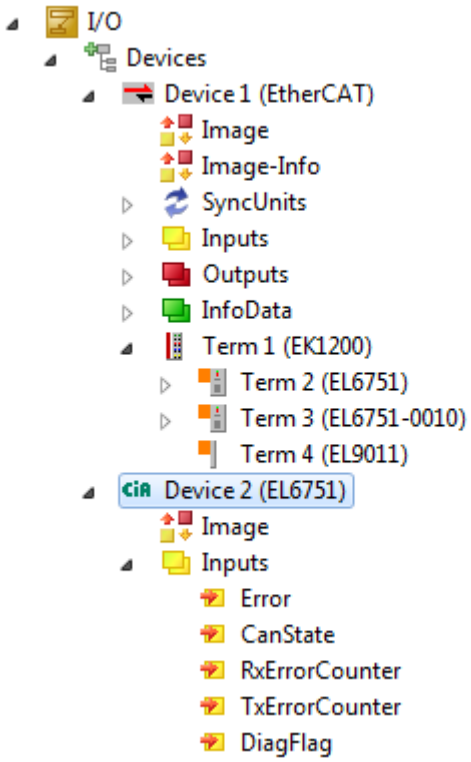


Abb. 3: CANOpen-Master hinzugefügt

Ein CANOpen-Master-Gerät in der Liste suchen und anfordern:

Der neu hinzugefügte CANOpen-Master muss konfiguriert werden, was normalerweise in TwinCAT mittels Drücken der Suchen-Taste und Auswahl des korrekten Geräts aus der Liste erledigt wird.

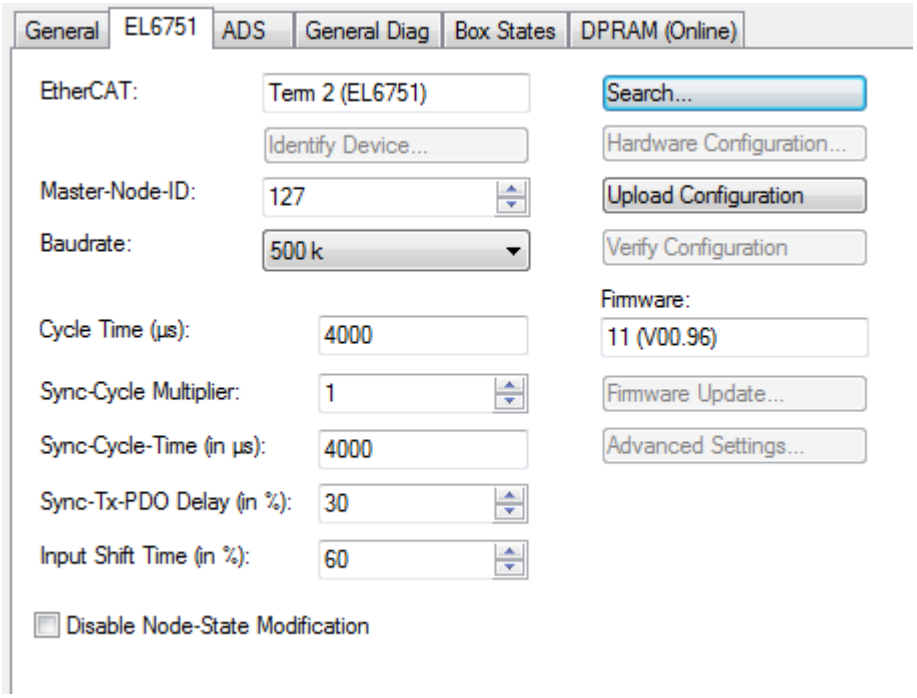


Abb. 4: CANOpen Master über Search-Taste

Dies kann über Automation Interface gemacht werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

ITcSmTreeItem5:ResourceCount zeigt die Anzahl kompatibler CANOpen-Master-Geräte an, und ITcSmTreeItem5:ClaimResources nimmt den Index des CANOpen-Geräts, das als Master konfiguriert werden soll.

Einen CANOpen-Slave erstellen und hinzufügen

Der aktuellen Konfiguration kann ein CANOpen-Slave folgendermaßen hinzugefügt werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Einen CANOpen-Slave suchen und anfordern

Wie im Fall von CANOpen-Master kann eine Liste von CANOpen-Slaves durch folgenden Code publiziert werden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die letzte Zeile im Code bestimmt das EL6731-0010 Gerät als CANOpen-Slave, ähnlich wie im Dialogfeld, das in der TwinCAT-Bedieneroberfläche erscheint.

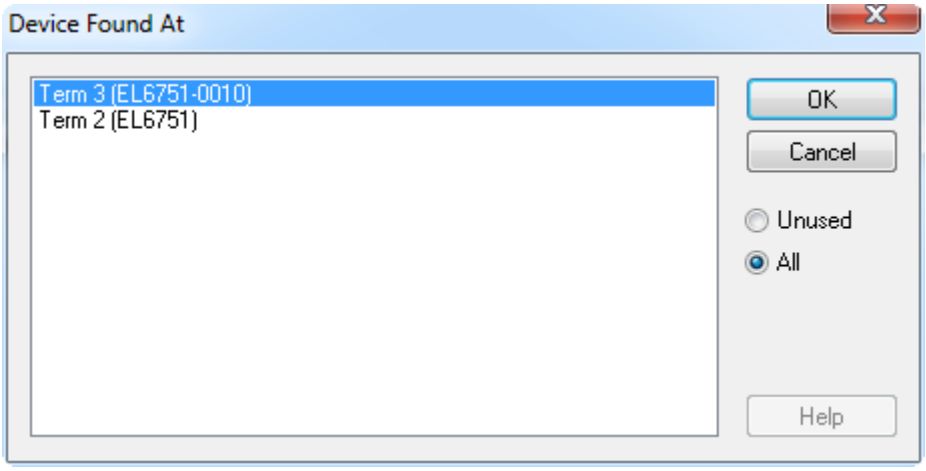


Abb. 5: CANOpen bestimmt Ressourcen über das Dialogfeld

DBC-Dateien über Automation Interface importieren

Erforderliche Version: TwinCAT 3.1 Build 4018 oder höher

TwinCAT ermöglicht den Import der DBC-Datei über ein Dialogfeld, wie dies im Screenshot wiedergegeben ist. Mit Rechtsklick auf einen CANOpen-Master, z. B. EL6751, und Auswahl von “Import dbc-file” fordert ein Dialogfeld den Benutzer auf, die DBC-Datei zu durchsuchen. Durch Klicken auf OK wird die CANOpen-Konfiguration automatisch geladen.

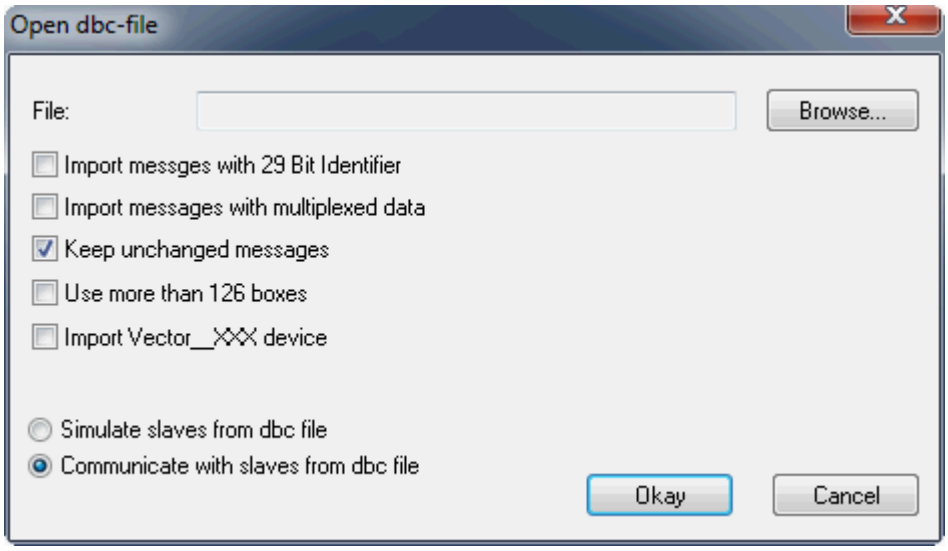


Abb. 6: CANOpen DBC-Datei importieren

Automation Interface unterstützt ebenfalls den DBC-Dateiimport. Hierzu muss zum CANOpen-Master-Tree Item navigiert und die XML-Datei über `ITcSmTreeItem:ProduceXml()` exportiert werden. In der XML-Datei den Pfad zu der zu importierenden DBC-Datei hinzufügen, zusammen mit den weiteren Eigenschaften, die Bestandteil des vorstehend wiedergegebenen Dialogfelds sind.

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

```
<ExtBoxesSupport>false</ExtBoxesSupport>
<VectorXXXSupport>false</VectorXXXSupport>
<CommunicateWithSlavesFromDbcFile>true</CommunicateWithSlavesFromDbcFile>
</ImportDbcFile>
</CanOpenMaster>
<Fcxxxx>
<CalculateEquiTimes>0</CalculateEquiTimes>
<NodeId>127</NodeId>
<Baudrate>500 k</Baudrate>
<DisableNodeStateModification>false</DisableNodeStateModification>
</Fcxxxx>
</DeviceDef>
```

Die modifizierte XML-Datei über `ITcSmTreeImte:ConsumeXml()` zurück in die TwinCAT-Konfiguration importieren. Die Konfiguration wird nun unter CANOpen-Master geladen.

4.3.6.6 Erstellung von und Umgang mit Devicenet-Geräten

Wie jede andere I/O-Komponente können Devicenet-Geräte über das TwinCAT Automation Interface unter Verwendung der CreateChild() Methode der ITcSmTreeItem Schnittstelle hinzugefügt werden. Der SubType spezifiziert das Gerät, das hinzugefügt werden soll.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Die folgende Tabelle gibt einen Überblick über alle Devicenet I/O-Geräte und deren entsprechenden SubTypes. Sollte ein Gerät fehlen, können Sie immer selbst den SubType herausfinden, indem Sie dem Artikel über die XML-Beschreibung eines Tree Items [▶ 25] folgen.

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

Devicenet-Boxen können ebenfalls über CreateChild() angeschlossen werden. Der SubType spezifiziert die Box, die hinzugefügt werden soll.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

Verwenden Sie zum Hinzufügen einer Devicenet-Box einfach den folgenden Code-Ausschnitt. Der vInfo Parameter spezifiziert den Datentyp der Variable.

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

4.3.6.7 Creating and handling AX5000 devices

This documentation article describes how to create AX5000 devices and and modify their mailbox startup list as well as their power management, PDO and motor condfiguration. All of these settings can be modifies by using the XML description [▶ 25] of the AX5000 TreeItem. This article consists of the following chapters:

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

Creating an AX5000 I/O device

AX5000 devices can be created by using the CreateChild() method. Please consult the documentation article Erstellung von und Umgang mit EtherCAT-Teilnehmern [▶ 71] for more information.

Code Snippet (C#)

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

Code Snippet (Powershell)

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

Overview XML description

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

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

Mailbox startup list

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

Import of startup list

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

PDO configuration

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

AdsInfo

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

Import of PDO configuration

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

Parametrization and template generation

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

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

Available tools and samples

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

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

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

CreateTcIoAx5000_Templates

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

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

CreateTcIoAx5000_Config

This Windows Powershell scripts demonstrates how to build a full AX5000 configuration out of different template files. These template files provide the settings for the PDO configuration as well as the mailbox startup list with their init commands for power management and motor configuration. The script output is a

full AX5000 XML description, which can be imported on a AX5000 TreeItem either by using the menu entry “TwinCAT -> SelectedItem -> Import XML description” or by using the Automation Interface method ConsumeXml().

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

Mappings

Process data mapping can be performed via regular Automation Interface mechanisms. Please consult the article Erstellung von und Umgang mit Variablenzuordnungen [▶ 42] for more information.

4.3.6.8 Nach Geräten und Boxen suchen

Bei der Erstellung einer neuen Konfiguration ist es häufig notwendig, die TwinCAT XAE-Konfiguration der tatsächlich verfügbaren Hardware anzupassen. Eine Möglichkeit, dies zu tun, besteht darin eine völlig neue TwinCAT XAE-Konfiguration zu starten und dann die folgenden Schritte durchzuführen:

- Erstellen einer neuen TwinCAT XAE-Konfiguration
- Festlegen der Adresse des Zielsystems
- Scannen der verfügbaren Geräte
- Hinzufügen und Parametrisierung der zu verwendenden Geräte
- Scannen und Einfügen von Boxen für jedes Gerät

Ablauf

Die Vorgehensweise für die Erstellung der ITcSysManager [▶ 118]-Schnittstelle (die '*systemManager*'-Instanz hier) wird im Kapitel Auf TwinCAT-Konfigurationen zugreifen [▶ 20] beschrieben. Diese Schnittstelle verfügt über eine LookupTreeItem [▶ 124]-Methode, die einen ITcSmTreeItem [▶ 127] Zeiger auf ein, mit seinem Pfadnamen [▶ 10] angegebenes Tree Item zurückgibt.

Code-Ausschnitt (C#):

```
ITcSysManager3 systemManager = null;

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

```
    {
      device.ConsumeXml(xml);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Warning: {0}",ex.Message);
    }
    foreach (ITcSmTreeItem box in device)
    {
      Console.WriteLine(box.Name);
    }
  }
}
```

4.3.6.9 I/O-Geräte aktivieren und deaktivieren

Um ein Tree Item in einer Konfiguration zu aktivieren/deaktivieren muss eine TwinCAT System Manager-Instanz erstellt und die Konfiguration geöffnet werden. Die LookupTreeItem [▶ 124]-Methode der ITcSysManager [▶ 118]-Schnittstelle gibt einen ITcSmTreeItem [▶ 127]-Schnittstellenzeiger zurück, der vom Tree Item, das über seinen Pfadnamen [▶ 10] referenziert wird, implementiert ist. Diese Schnittstelle beinhaltet eine Disabled [▶ 127]-Eigenschaft des Tree Items.

Ablauf

Die Vorgehensweise für die Erstellung der ITcSysManager [▶ 118]-Schnittstelle (die '*sysMan*'-Instanz hier) wird im Kapitel Auf TwinCAT-Konfigurationen zugreifen [▶ 20] beschrieben. Diese Schnittstelle verfügt über eine LookupTreeItem [▶ 124]-Methode, die einen ITcSmTreeItem [▶ 127]-Zeiger auf ein, mit seinem Pfadnamen [▶ 10] angegebenes Tree Item zurückgibt. Zum Aktivieren/Deaktivieren des Tree Items "*TIID^EtherCAT Master*" können die folgenden Code-Ausschnitte verwendet werden.

Beispiel (CSharp):

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

Beachten Sie, dass Sie für dieses Beispiel Ihrem Projekt sowohl eine Referenz auf "Microsoft Developer Environment 10.0", als auch auf "Beckhoff TCatSysManager Bibliothek 1.1" hinzufügen müssen.

Beispiel (PowerShell):

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

Beispiel (VBScript):

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

4.3.7 TcCOM

4.3.7.1 Erstellung von und Umgang mit TcCOM Modulen

In diesem Kapitel wird beschrieben, wie bestehende TcCOM-Module einer bestehenden TwinCAT-Konfiguration hinzugefügt werden und diese parametrisiert werden. Die folgenden Themen werden in diesem Kapitel kurz behandelt:

- Erwerb einer Referenz zu “TcCOM Objects” Knoten
- TcCOM-Module hinzufügen

- Durch hinzugefügte TcCOM-Module iterieren
- CreateSymbol Flag für Parameter einstellen
- CreateSymbol Flag für Datenbereiche einstellen
- Kontext (Aufgaben) einstellen
- Variablen verknüpfen

Erwerb einer Referenz zu "TcCOM Objects" Knoten

In einer TwinCAT-Konfiguration befindet sich der "TcCOM Objects" Knoten unter "SYSTEM^TcCOM Objects". Daher können Sie eine Referenz zu diesem Knoten erfassen, indem Sie die Methode ITcSysManager::LookupTreeItem() auf folgende Weise verwenden:

Code-Ausschnitt (C#):

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

Code-Ausschnitt (Powershell):

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

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

Adding existing TcCOM modules

Der vorstehende Code geht davon aus, dass bereits ein SystemManager Objekt in Ihrem AI-Code vorhanden ist.

TcCOM-Module hinzufügen

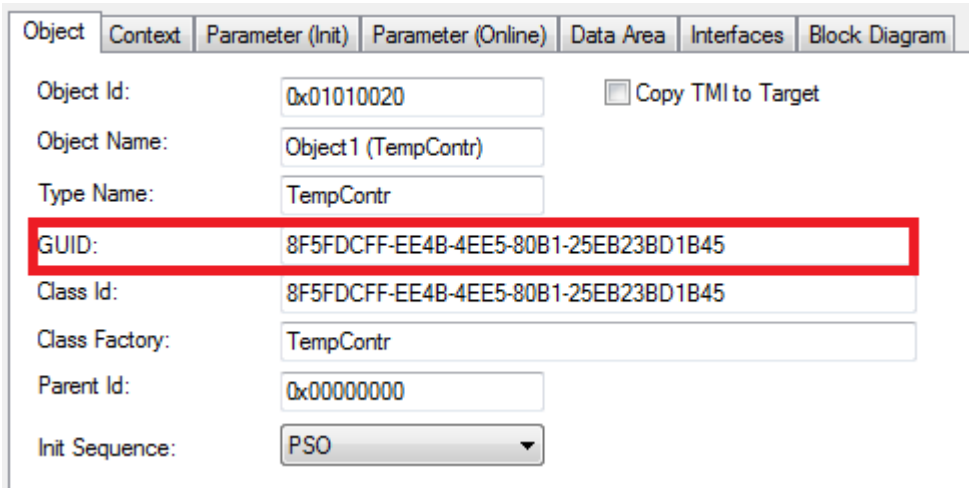
Um bestehende TcCOM-Module zu Ihrer TwinCAT-Konfiguration hinzufügen, müssen diese Module für TwinCAT erkennbar sein. Dies kann durch eine der folgenden Vorgehensweisen erreicht werden:

- TcCOM-Module zum Ordner %TWINCAT3.XDIR"\CustomConfig\Modules\ kopieren
- %TWINCAT3.XDIR"\Config\Io\TcModuleFolders.xml bearbeiten, um einen Pfad zu einem Ordner Ihrer Wahl hinzuzufügen und die Module innerhalb dieses Ordners zu platzieren

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

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

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



Alternativ können Sie die GUID über die TMC-Datei des TcCOM-Moduls bestimmen.

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

```
    </Module>
  </Modules>
</TcModuleClass>
```

Angenommen, wir verfügen bereits über ein TcCOM-Modul, das in TwinCAT registriert ist und für TwinCAT erkennbar ist. Wir möchten nun dieses TcCOM-Modul, das den GUID {8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45} hat, zur TwinCAT-Konfiguration hinzufügen. Dies kann wie folgt geschehen:

Code-Ausschnitt (C#):

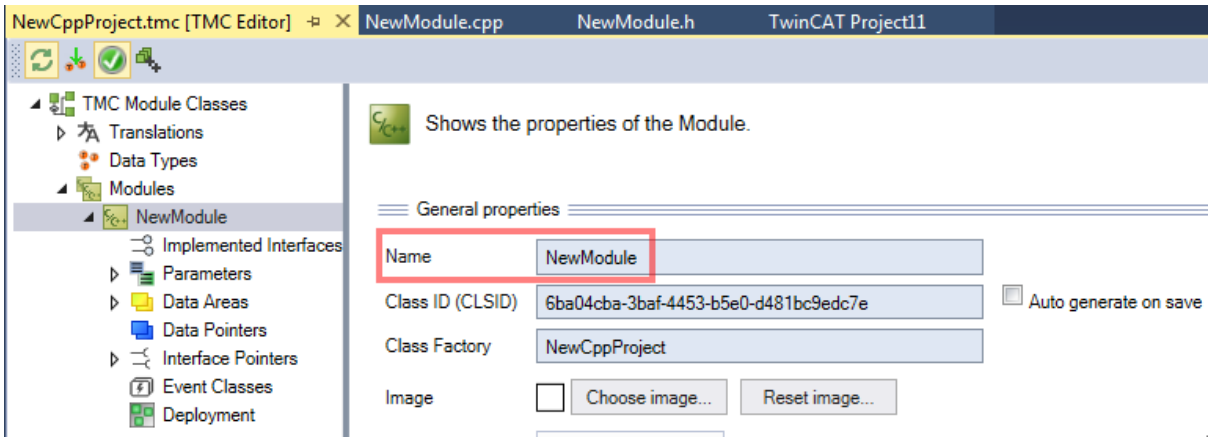
```
Dictionary<string,Guid> tcomModuleTable = new Dictionary<string,Guid>();
tcomModuleTable.Add("TempContr",Guid.Parse("{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}"));
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 0, "",
tcomModuleTable["TempContr"]);
```

Code-Ausschnitt (Powershell):

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

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

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



- This can be done by the following way:

Code Snippet (C#):

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

Code Snippet (Powershell):

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

Durch hinzugefügte TcCOM-Module iterieren

Zur Iteration durch alle hinzugefügte TcCOM-Modulinstanzen können Sie die ITcModuleManager2 Schnittstelle verwenden. Der folgende Code-Ausschnitt zeigt, wie Sie diese Schnittstelle verwenden.

Code-Ausschnitt (C#):

```
ITcModuleManager2 moduleManager = (ITcModuleManager2)systemManager.GetModuleManager();
foreach (ITcModuleManager2 moduleInstance in moduleManager)
{
  string moduleType = moduleInstance.ModuleTypeName;
  string instanceName = moduleInstance.ModuleInstanceName;
  Guid classId = moduleInstance.ClassID;
  uint objId = moduleInstance.oid;
  uint parentObjId = moduleInstance.ParentOID;
}
```

Code-Ausschnitt (Powershell):

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

Denken Sie daran, dass jedes Modulobjekt ebenfalls als ein ITcSmTreeItem interpretiert werden kann, daher wäre die folgende Typumwandlung gültig:

Code-Ausschnitt (C#):

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

Bitte beachten: Powershell verwendet standardmäßig dynamische Datentypen.

CreateSymbol Flag für Parameter einstellen

Das CreateSymbol (CS) Flag für Parameter eines TcCOM-Moduls kann über ihre XML-Beschreibung eingestellt werden. Der folgende Code-Ausschnitt zeigt, wie das CS Flag für den Parameter „CallBy“ aktiviert wird.

Code-Ausschnitt (C#):

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

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

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

createSymbol.Value = "true";

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

Code-Ausschnitt (Powershell):

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

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

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

$callByParameter.CreateSymbol = "true"
```

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

CreateSymbol Flag für Datenbereiche einstellen

Das CreateSymbol (CS) Flag für Datenbereiche eines TcCOM-Moduls kann über ihre XML-Beschreibung eingestellt werden. Der folgende Code-Ausschnitt zeigt, wie das CS Flag für den Datenbereich „Input“ aktiviert wird. Es sei darauf hingewiesen, dass dieses Verfahren dem für Parameter sehr ähnlich ist.

Code-Ausschnitt (C#):

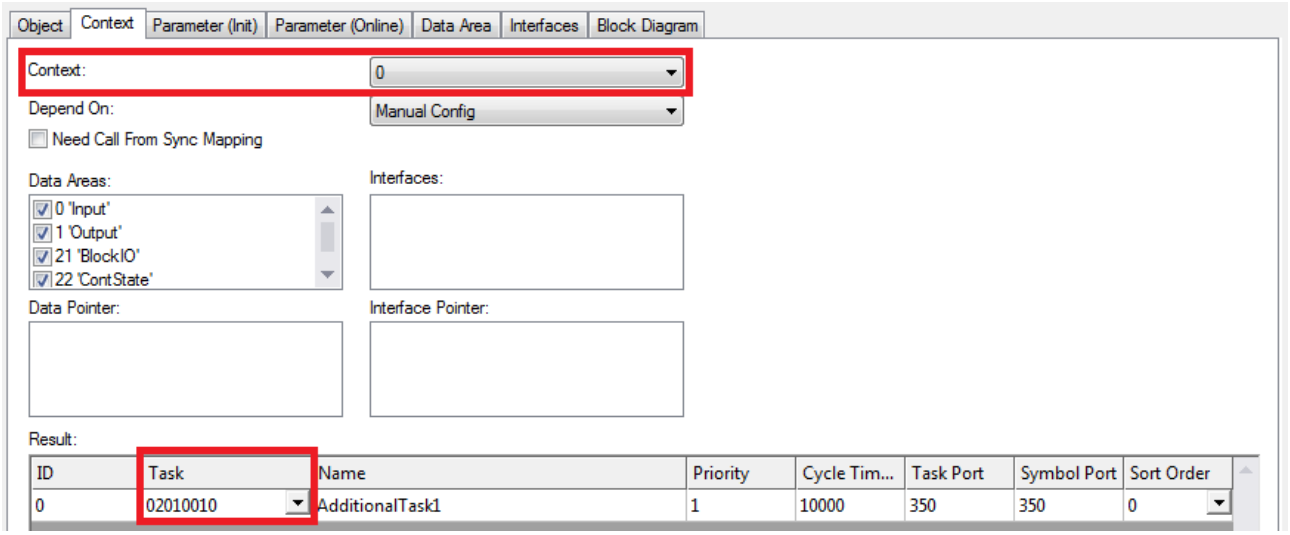
```
bool activateCS = true;
// First step: Read all Data Areas of a TcCOM module instance
string tempControllerXml = tempController.ProduceXml();
XmlDocument tempControllerDoc = new XmlDocument();
tempControllerDoc.LoadXml(tempControllerXml);
XmlNode sourceDataAreas = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/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

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

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

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

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

Code-Ausschnitt (Powershell):

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

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

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

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

Kontext (Aufgaben) einstellen

Jede TcCOM-Modulinstanz muss in einem spezifischen Kontext (Aufgabe) laufen. Dies kann über die ITcModuleInstance2::SetModuleContext() Methode erfolgen. Diese Methode erwartet zwei Parameter: ContextId und TaskObjectId. Beide sind äquivalent zu den entsprechenden Parametern in TwinCAT XAE:



Denken Sie daran, dass die TaskObjectId in TwinCAT XAE hexadezimal wiedergegeben wird.

Code-Ausschnitt (C#):

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

Sie können die TaskObjectId über die XML-Beschreibung der entsprechenden Aufgabe bestimmen, z. B.:

Code-Ausschnitt (C#):

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

Variablen verknüpfen

Die Verknüpfung von Variablen einer TcCOM-Modulinstanz zu SPS/IO oder anderen TcCOM-Modulen kann vorgenommen werden, indem die regulären Mechanismen des Automation Interface verwendet werden, z. B. ITcSysManager::LinkVariables().

4.3.8 C++

4.3.8.1 Erstellung von und Umgang mit C++ Projekten und Modulen

In diesem Kapitel wird die Erstellung von, der Zugriff auf und der Umgang mit TwinCAT C++ Projekten ausführlich erklärt. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über C++ Projekte
- Neue C++ Projekte erstellen
- Neues Modul innerhalb eines C++ Projekts erstellen
- Bestehende C++ Projekte öffnen
- Modulinstanzen erstellen
- TMC Code Generator aufrufen
- Publish Modules Befehl aufrufen

- C++ Projekteigenschaften einstellen
- Projekt aufbauen

Allgemeine Informationen über C++ Projekte

C++ Projekte werden durch ihre sogenannten Projektvorlagen spezifiziert, die vom „TwinCAT C++ Projekt-Assistenten“ verwendet werden. Innerhalb eines Projekts können verschiedene Module durch Modulvorlagen spezifiziert werden, die vom „TwinCAT Klassenassistenten“ verwendet werden.

TwinCAT-definierte Vorlagen sind im Abschnitt C++ / Assistenten dokumentiert.

Der Kunde kann eigene Vorlagen definieren, was in dem entsprechenden Unterabschnitt C++ Abschnitt / Assistenten dokumentiert ist.

C++ Projekte erstellen

Um ein neues C++ Projekt mit Hilfe des Automation Interface zu erstellen, müssen Sie zum C++ Knoten navigieren und dann die CreateChild()-Methode mit entsprechenden Vorlagendatei als Parameter ausführen.

Code-Ausschnitt (C#):

```
ITcSmTreeItem cpp = systemManager.LookupTreeItem("TIXC");
ITcSmTreeItem cppProject = cpp.CreateChild("NewCppProject", 0, "", pathToTemplateFile);
```

Code-Ausschnitt (Powershell):

```
$cpp = $systemManager.LookupTreeItem("TIXC")
$newProject = $cpp.CreateChild("NewCppProject", 0, "", $pathToTemplateFile)
```

Zur Instanziierung eines Treiber-Projekts verwenden Sie "TcVersionedDriverWizard" als `pathToTemplateFile`.

Neues Modul innerhalb eines C++ Projekts erstellen

Innerhalb eines C++ Projekts wird ein TwinCAT Modul-Assistent verwendet, damit der Assistent aus einer Vorlage ein Modul erstellt.

Code-Ausschnitt (C#):

```
ITcSmTreeItem cppModule = cppProject.CreateChild("NewModule", 1, "", pathToTemplateFile);
```

Code-Ausschnitt (Powershell):

```
$cppModule = $cppProject.CreateChild("NewModule", 0, "", $pathToTemplateFile);
```

Als Beispiel zur Instanziierung eines Cyclic IO-Modulprojekts verwenden Sie "TcModuleCyclicCallerWizard" als `pathToTemplateFile`.

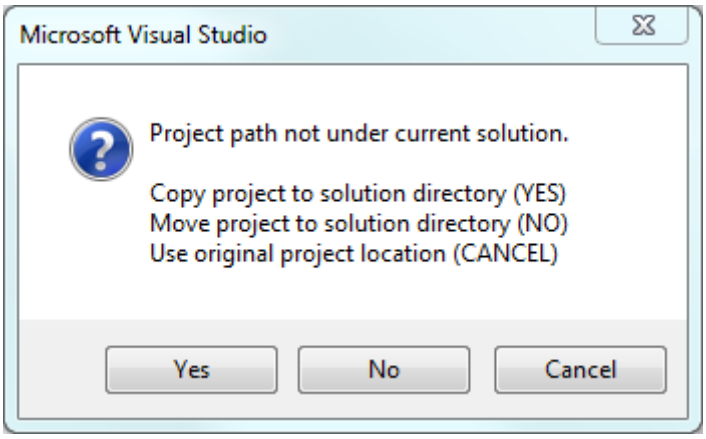
Bestehende C++ Projekte öffnen

Zum Öffnen eines bestehenden C++ Projekts mit Hilfe von Automation Interface, müssen Sie zum C++ Knoten navigieren und dann die CreateChild()-Methode mit dem Pfad der entsprechenden C++ Projektdatei als Parameter ausführen.

Sie können drei verschiedene Werte als SubType verwenden:

- 0: Projekt zum Solution-Verzeichnis kopieren
- 1: Projekt zum Solution-Verzeichnis verschieben
- 2: Verwenden Sie den Original-Projektspeicherort (spezifizieren Sie “” als NameOfProject Parameter)

Grundsätzlich repräsentieren diese Werte die Funktionalitäten (Ja, Nein, Abbrechen) von der folgenden MessageBox in TwinCAT XAE:



Anstelle der Vorlagendatei müssen Sie den Pfad zum C++ Projekt (bzw. dessen vcxproj Datei) verwenden, das hinzugefügt werden muss. Alternativ können Sie auch ein C++ Projektarchiv (tczip Datei) verwenden.

Code-Ausschnitt (C#):

```
ITcSmTreeItem cpp = systemManager.LookupTreeItem("TIXC");
ITcSmTreeItem newProject = cpp.CreateChild("NameOfProject", 1, "", pathToProjectOrTczipFile);
```

Code-Ausschnitt (Powershell):

```
$cpp = $systemManager.LookupTreeItem("TIXC")
$newProject = $cpp.CreateChild("NameOfProject", 1, "", $pathToProjectOrTczipFile)
```

Achten Sie darauf, dass C++ Projekte nicht umbenannt werden können, somit muss der Original-Projektname spezifiziert werden. (vergl. Umbenennen von TwinCAT-C++ Projekten)

Modulinstanzen erstellen

TcCOM Module können am System -> TcCOM Modulknoten erstellt werden. Siehe dort diesbezügliche Dokumentation [▶ 96].

Das gleiche Verfahren kann auch für den C++ Projektknoten gelten, um die TcCOM Instanzen an der Stelle hinzuzufügen (`$newProject` im Code oben auf dieser Seite).

TMC Code Generator aufrufen

Der TMC-Code-Generator kann aufgerufen werden, um den C++ Code nach den Änderungen der TMC-Datei oder dem C++ Projekt zu erstellen.

Code-Ausschnitt (C#):

```
string startTmcCodeGenerator = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<StartTmcCodeGenerator>
<Active>true</Active>
</StartTmcCodeGenerator>
</Methods>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(startTmcCodeGenerator);
```

Code-Ausschnitt (Powershell):

```
$startTmcCodeGenerator = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<StartTmcCodeGenerator>
<Active>true</Active>
</StartTmcCodeGenerator>
</Methods>
```

```
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($startTmcCodeGenerator)
```

Publish Modules Befehl aufrufen

Die Veröffentlichung umfasst den Aufbau des Projekts für alle Plattformen. Das kompilierte Modul wird für den Export bereitgestellt, wie dies im Abschnitt Modulumgang bei C++ beschrieben ist.

Code-Ausschnitt (C#):

```
string publishModules = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<PublishModules>
<Active>true</Active>
</PublishModules>
</Methods>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(publishModules);
```

Code-Ausschnitt (Powershell):

```
$publishModules = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<PublishModules>
<Active>true</Active>
</PublishModules>
</Methods>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($publishModules)
```

C++ Projekteigenschaften einstellen

C++ Projekte bieten verschiedene Optionen für den Aufbau- und Bereitstellungsprozess.
Diese sind über das Automation Interface einstellbar.

Code-Ausschnitt (C#):

```
string projProps = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<BootProjectEncryption>Target</BootProjectEncryption>
<TargetArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</TargetArchiveSettings>
<FileArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</FileArchiveSettings>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(projProps);
```

Code-Ausschnitt (Powershell):

```
$projProps = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<BootProjectEncryption>Target</BootProjectEncryption>
<TargetArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</TargetArchiveSettings>
<FileArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</FileArchiveSettings>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($projProps)
```

Für die BootProjectEncryption sind die Werte „None“ und „Target“ gültig.
Beide anderen Einstellungen sind „false“ und „true“ Werte.

Projekt aufbauen

Zum Aufbau des Projekts oder der Solution können Sie die entsprechenden Klassen und Methoden der Visual Studio API verwenden, die hier [▶ 31] dokumentiert sind.

4.3.9 Measurement

4.3.9.1 Erstellung von und Umgang mit TwinCAT Measurement-Projekten

Das TwinCAT Automation Interface stellt Methoden und Eigenschaften zur Verfügung, um TwinCAT Measurement-Projekte zu erzeugen und darauf zuzugreifen. Im folgenden Kapitel wird beschrieben, wie einige Grundaufgaben mit solch einem TwinCAT-Projekt gelöst werden können und umfasst Informationen zu folgenden Themen:

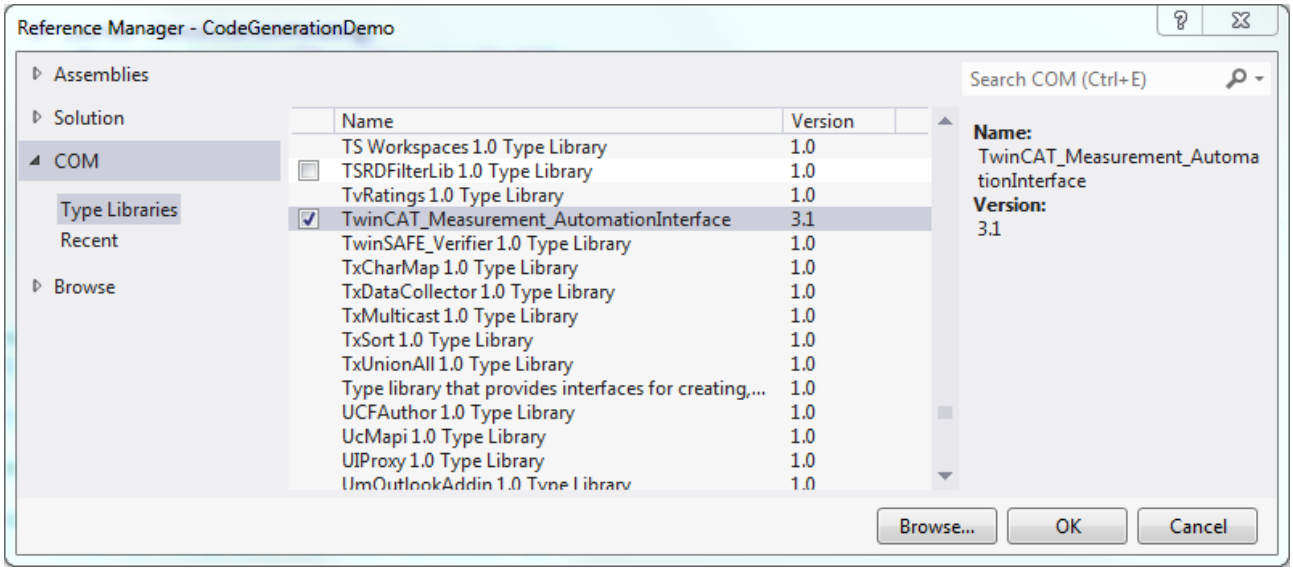
- Anforderungen
- Erstellung eines TwinCAT Measurement-Projekts
- Erstellung einer TwinCAT Scope-Konfiguration
- Erstellung von, Zugriff auf und Umgang mit Diagrammen
- Erstellung von, Zugriff auf und Umgang mit Achsen
- Erstellung von, Zugriff auf und Umgang mit Kanälen
- Aufzeichnungen starten und stoppen
- Weitere Auskünfte zu TwinCAT Measurement finden Sie auf der entsprechenden Internetseite in unserem Informationssystem.

Anforderungen

Die Integration von TwinCAT Measurement-Projekten über das Automation Interface ist ab TwinCAT 3.1 Build 4013 möglich.

Ein Verweis auf die folgenden COM-Schnittstellen ist als Vorbedingung für die Verwendung der Scope AI Funktionalitäten erforderlich:

- TwinCAT Measurement Automation Interface (als COM-Bibliothek registriert)
- TwinCAT Scope2 Kommunikation (.NET-Assembly: C:\Windows\Microsoft.NET\assembly\GAC_MSIL\TwinCAT.Scope2.Communications\v4.0_3.1.3121.0__180016cd49e5e8c3\)



Erstellung eines TwinCAT Measurement-Projekts

TwinCAT Measurement ist ein globaler "Container" der eine oder mehrere Measurement-Projekte hosten kann, z.B. eine TwinCAT Scope-Konfiguration. Ähnlich wie eine reguläre TwinCAT-Konfiguration wird jedes Projekt zu allererst mit Hilfe einer Template-Datei beschrieben. Diese Template-Datei wird beim Hinzufügen eines neuen Projekts zur Solution verwendet, was mit Hilfe eines Aufrufs der AddFromTemplate()-Methode aus dem Visual Studio DTE-Objekt vorgenommen werden kann. Beachten Sie, dass diese Vorgehensweise beim Hinzufügen eines regulären TwinCAT-Projekts die gleiche ist. Der nachfolgende Code-Ausschnitt setzt voraus, dass Sie bereits eine DTE-Instanz erworben und eine Visual Studio Solution erzeugt haben, wie in unserem Artikel über Zugriff auf TwinCAT-Konfigurationen gezeigt. Ein Verweis auf die DTE-Instanz wird im Objekt "dte" gespeichert.

Code-Ausschnitt (C#):

```
EnvDTE.Project scopeProject = dte.Solution.AddFromTemplate(template, destination, name);
```

[template]: Die standardmäßigen Template-Dateien sind unter C:\TwinCAT\Functions\TE130X-Scope-View\Templates\Projects\ gespeichert und haben den Dateityp "tcmproj".

[destination]: Pfad, wo die neue Konfiguration auf der Festplatte gespeichert werden soll.

[name]: Name für die neue Konfiguration, wie sie in TwinCAT XAE angezeigt werden soll.

Erstellung einer TwinCAT Scope-Konfiguration

Ein TwinCAT Scope-Projekt steht für eine Aufzeichnungskonfiguration. Das bedeutet, dass alle in das Projekt eingefügten Elemente den gleichen Aufzeichnungseinstellungen unterliegen. Sie können ein Scope-Projekt über das Automation Interface hinzufügen, indem Sie, wie oben beschrieben, das entsprechende "TwinCAT Scope Project"-Template beim Hinzufügen eines Projekts mithilfe der AddFromTemplate()-Methode spezifizieren.

Erstellung von, Zugriff auf und Umgang mit Diagrammen

In einer Scope-Konfiguration können mehrere Diagramme parallel bestehen. Um einem bestehenden Scope-Projekt Diagramme hinzuzufügen, verwenden Sie einfach die CreateChild()-Methode von der IMeasurementScope-Schnittstelle. Der folgende Code-Ausschnitt setzt voraus, dass bereits ein TwinCAT Measurement-Projekt erstellt wurde und dass ein Verweis auf dieses Projekt im Objekt "scopeProject" gespeichert ist.

Code-Ausschnitt (C#):

```
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).ShowControl();
EnvDTE.ProjectItem newChart;
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).CreateChild(out newChart);
IMeasurementScope newChartObj = (IMeasurementScope)newChart.Object;
newChartObj.ChangeName("NC Chart");
```

Das Objekt "newChartObj" speichert nun einen Verweis auf das neu hinzugefügte Diagramm. Beachten Sie, dass Sie zu diesem Zeitpunkt weiterhin das ursprüngliche Objekt "newChart" für spätere Zwecke, z.B. Einstellen der Diagrammeigenschaften, benötigen.

In TwinCAT XAE werden die Diagrammeigenschaften über das Visual Studio-Eigenschaftenfenster festgelegt. Das Objekt "newChart" ermöglicht den Zugriff auf diese Eigenschaften über dessen Auflistung "Properties". Der folgende Code-Ausschnitt läuft durch alle Eigenschaften eines Diagramms und setzt die Eigenschaft "StackedYAxis" auf "true".

Code-Ausschnitt (C#):

```
foreach (EnvDTE.Property prop in newChart.Properties)
{
If (prop.Name = "StackedYAxis")
prop.Value = true;
}
```

Erstellung von, Zugriff auf und Umgang mit Achsen

Der folgende Code-Ausschnitt zeigt, wie Achsen hinzugefügt werden können.

Code Snippet (C#):

```
EnvDTE.ProjectItem newAxis;
newChartObj.CreateChild(out newAxis);
IMeasurementScope newAxisObj = (IMeasurementScope)newAxis.Object;
newAxisObj.ChangeName(“Axis 1”);
```

Erstellung von, Zugriff auf und Umgang mit Kanälen

Ein Scope-Kanal beschreibt eine Verbindung zu einem Laufzeitsymbol, z.B. einer SPS-Variable. Um Kanäle über das Automation Interface hinzuzufügen, können Sie die CreateChild()-Methode von der IMeasurementScope-Schnittstelle verwenden.

Code-Ausschnitt (C#):

```
EnvDTE.ProjectItem newChannel;
newAxisObj.CreateChild(out newChannel);
IMeasurementScope newChannelObj = (IMeasurementScope)newChannel.Object;
newChannelObj.ChangeName("Signals.Rectangle");
```

Das Objekt "newChannelObj" speichert nun einen Verweis auf den neu hinzugefügten Kanal. Beachten Sie, dass Sie zu diesem Zeitpunkt weiterhin das ursprüngliche Objekt "newChannel" für spätere Zwecke, z.B. Einstellen der Kanaleigenschaften, benötigen.

In TwinCAT XAE werden die Kanaleigenschaften über das Visual Studio-Eigenschaftenfenster festgelegt. Das Objekt "newChannel" ermöglicht den Zugriff auf diese Eigenschaften über dessen Auflistung "Properties". Der folgende Code-Ausschnitt läuft durch alle Eigenschaften eines Kanals und legt mehrere Eigenschaften fest.

Code-Ausschnitt (C#):

```
foreach (EnvDTE.Property prop in newChannel.Properties)
{
switch (prop.Name)
{
case "TargetPorts":
prop.Value = "851";
break;

case "Symbolbased":
prop.Value = true;
break;

case "SymbolDataType":
prop.Value = TwinCAT.Scope2.Communications.Scope2DataType.BIT;
break;

case "SymbolName":
prop.Value = "MAIN.bSignal";
break;

case "SampleState":
prop.Value = 1;
break;

case "LineWidth":
prop.Value = 3;
}
}
```

Wie Sie sehen können, definiert die Aufzählung Scope2DataType mehrere Datentypen, die bei der Konfiguration eines Kanals unterstützt werden. Bei der Erstellung dieses Dokuments, ist diese Aufzählung folgendermaßen definiert:

Code-Ausschnitt (C#):

```
public enum Scope2DataType
{
VOID = 0,
BIT = 1,
INT16 = 2,
INT32 = 3,
INT64 = 4,
INT8 = 5,
REAL32 = 6,
REAL64 = 7,
UINT16 = 8,
UINT32 = 9,
```

```
UINT64 = 10,
UINT8 = 11,
BIT_ARRAY_8 = 12,
}
```

Bitte verwenden Sie einen DLL-Explorer (z.B. Visual Studio) für eine jüngere und aktuellere Liste.

Aufzeichnungen starten und stoppen

Zum Starten/Stoppen einer konfigurierten Scope-Aufzeichnung können die entsprechenden Methoden der IMeasurementScope-Schnittstelle verwendet werden:

Code-Ausschnitt (C#):

```
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).StartRecord()
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).StopRecord();
```

4.3.9.2 Erstellung von und Umgang mit TwinCAT Analytics-Projekten

Das TwinCAT Automation Interface stellt Methoden und Eigenschaften zur Verfügung, um TwinCAT Measurement-Projekte zu erzeugen und darauf zuzugreifen. Im folgenden Kapitel wird beschrieben, wie einige Grundaufgaben mit einem TwinCAT-Projekt gelöst werden können und umfasst Informationen zu folgenden Themen:

- Anforderungen
- Erstellung eines TwinCAT Measurement-Projekts
- Erstellung von, Zugriff auf und Umgang mit einer TwinCAT Analytics-Konfiguration
- Erstellung von, Zugriff auf und Umgang mit Netzwerken
- Erstellung von, Zugriff auf und Umgang mit Funktionen
- Analyse starten und stoppen
- Weitere Auskünfte zu TwinCAT Measurement finden Sie auf der entsprechenden Internetseite in unserem Informationssystem.

Erstellung eines TwinCAT Measurement-Projekts

TwinCAT Measurement ist ein globaler "Container" der eine oder mehrere Measurement-Projekte hosten kann, z. B. eine TwinCAT Scope-Konfiguration. Ähnlich wie eine reguläre TwinCAT-Konfiguration wird jedes Projekt zuerst mit Hilfe einer Template-Datei beschrieben. Diese Template-Datei wird beim Hinzufügen eines neuen Projekts zur Solution verwendet, was mithilfe eines Aufrufs der AddFromTemplate()-Methode aus dem Visual Studio DTE-Objekt vorgenommen werden kann. Beachten Sie, dass diese Vorgehensweise beim Hinzufügen eines regulären TwinCAT-Projekts die gleiche ist. Der nachfolgende Code-Ausschnitt setzt voraus, dass Sie bereits eine DTE-Instanz erworben und eine Visual Studio Solution erzeugt haben, wie in unserem Artikel über Zugriff auf TwinCAT-Konfigurationen gezeigt. Ein Verweis auf die DTE-Instanz wird im Objekt "dte" gespeichert.

```
EnvDTE.Project scopeProject = dte.Solution.AddFromTemplate(template, destination, name);
```

Erstellung von, Zugriff auf und Umgang mit einer TwinCAT Analytics-Konfiguration

Ein TwinCAT Analytics -Projekt steht für eine Analysekonfiguration. Das bedeutet, dass alle in das Projekt eingefügten Elemente den gleichen Analyseeinstellungen unterliegen. Sie können ein Analytics -Projekt über das Automation Interface hinzufügen, indem Sie, wie oben beschrieben, das entsprechende "TwinCAT Analytics Project"-Template beim Hinzufügen eines Projekts mithilfe der AddFromTemplate()-Methode spezifizieren.

Das dadurch hinzugefügte EnvDTE.Project Element kann dann auf das Interface IMeasurementAnalyticsProject abgebildet werden, welches folgende Methoden bereitstellt:

int StartAnalytics()	Startet den Analyseprozess.
int StopAnalytics();	Stoppt den Analyseprozess.
int AddReferencedScope();	Fügt eine Instanz eines TwinCAT Scope Projektes der Projektmappe hinzu und verknüpft die Analytics Elemente. Die TwinCAT Scope Instanz mit dem Interface IMeasurementScope per Automation Interface angepasst werden.
int ChangeName(string name);	Methode zum Ändern des Projektnamens.
int ShowControl();	Bringt den Microsoft Visual Studio© Editor in den Vordergrund zur Darstellung des Analytics Projektes.
int CloseControl();	Schließt den Editor.
int GetAvailableModules(out Hashtable modules);	Befüllt eine Hashtabelle mit den Daten der bereitgestellten Module der Analytics Engine. Die Hashtabelle könnte wie folgt aussehen: {02040109-0000-0000-f000-000000000064} "Min Max Avg 1Ch" {02040103-0000-0000-f000-000000000064} "Edge Counter OnOff 2Ch" {02040102-0000-0000-f000-000000000064} "Edge Counter OnOff 1Ch" {02040101-0000-0000-f000-000000000064} "Edge Counter 1Ch" Die GUIDs sind für das spätere Erzeugen den Analyse-Funktionen notwendig.
int AddNetwork(out ProjectItem item, string name = "");	Fügt ein Netzwerk dem Analytics-Projekt hinzu und gibt die Instanz des EnvDTE.ProjectItem aus mit welchen weitergearbeitet werden kann. Das Objekt EnvDTE.ProjectItem.Object kann auf die Schnittstelle IMeasurementAnalayticsNetwork abgebildet werden.

Erstellung von, Zugriff auf und Umgang mit Netzwerken

Ein TwinCAT Analytics Netzwerk stellt die Ebene dar, in welcher Funktionsbausteine angelegt, verwaltet und visualisiert werden.

Ein Netzwerk kann mehrere Instanzen von verschiedenen Funktionen, sowie weitere Netzwerke als sogenannte Sub-Netzwerke beinhalten.

Jedes Netzwerk wird innerhalb einer Microsoft Visual Studio© Editor Instanz dargestellt, sodass die internen Docking-Mechanismen genutzt werden können, um gleichzeitig verschiedene Netzwerke darzustellen.

Das Interface IMeasurementAnalayticsNetwork definiert folgende Methoden:

int ShowControl();	Bringt den Microsoft Visual Studio© Editor in den Vordergrund zur Darstellung des Analytics Netzwerkes.
int CloseControl();	Schließt den Editor.
int AddFunction(out ProjectItem item, Guid guid, string name = "");	Fügt eine Analyse-Funktion dem Netzwerk hinzu. Die Funktion wird über die zugehörige GUID bestimmt, welche über die Interface-Methode IMeasurementAnalyticsProject.GetAvailableModules(out Hashtable modules); ausgewählt werden kann. Die Instanz von EnvDTE.ProjectItem.Object kann auf die Schnittstelle IMeasurementAnalayticsFunction abgebildet werden.
int AddNetwork(out ProjectItem item, string name = "");	Das Objekt EnvDTE.ProjectItem.Object kann auf die Schnittstelle IMeasurementAnalayticsNetwork abgebildet werden.
int ChangeName(string name);	Ändert den Namen des Netzwerkes.
int AddNetworkTemplate(out ProjectItem item, string path);	Fügt ein Netzwerk aus einer Vorlage hinzu. Die Vorlage muss auf Dateiebene angegeben werden. Das Objekt EnvDTE.ProjectItem.Object kann auf die Schnittstelle IMeasurementAnalayticsNetwork abgebildet werden.

Erstellung von, Zugriff auf und Umgang mit Funktionen

Eine TwinCAT Analytics Function stellt die Ebene der Analysefunktion dar.

Jede Funktion hat eigene Eingangs- und Ausgangsvariablen sowie diverse Konfigurationsparameter.

Das Interface IMeasurementAnalayticsFunction definiert folgende Methoden:

int ShowControl();	Bringt den Microsoft Visual Studio© Editor in den Vordergrund zur Darstellung des Analytics Projektes.
int CloseControl();	Schließt den Editor.
int ChangeName(string name);	Ändert den Namen der Funktion.
int SetInputVariable(string input, int inputIndex);	Setzt die Eingangsvariable an den angegebenen Index der Funktion auf den Input-String. Der Input-String muss in der XML-Formatierung vorliegen als Beispiel aus dem TargetBrowser. TargetBrowserExportInfo Um den XML formatierten String zu erhalten kann man aus dem TargetBrowser einfach die gewünschte Variable in einen Texteditor per Drag&Drop verschieben. Bitte beachten Sie, dass dazu der TextEditor als Administrator gestartet sein muss, falls der TargetBrowser Host (z.B. Visual Studio) auch als Administrator gestartet wurde.

Um die Parameter der Funktion anzupassen wird kein Interface benötigt.

Hierzu können Sie einfach die EnvDTE-Properties Liste des EnvDTE.ProjectItem.Properties Objektes nach dem EnvDTE.Property durchsuchen und dessen Value setzen.

4.3.9.3 Erstellen von und Umgang mit Analytics Logger und Stream Helper

Das TwinCAT Automation Interface stellt Methoden und Eigenschaften zur Verfügung, um TwinCAT Analytics Logger und Stream Helper zu erzeugen und auf sie zuzugreifen. Alle Funktionen werden nachfolgend mit C# Code-Beispielen beschrieben.

• Anlegen und Löschen eines Data Loggers

• Anlegen und Löschen eines Stream Helpers

• Parametrieren eines bereits angelegten Data Loggers

• Parametrieren von Streams

• Auswählen von Symbolen, die gelogged werden sollen.

Alle diese Vorgänge sind auf die jeweiligen Knoten der Analytics-Konfiguration, des Analytics Data Loggers, Streams und Stream Helpers in einem TwinCAT XAE Projekt bezogen.

4.3.9.3.1 Anlegen und Löschen eines DataLoggers

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics Configuration Node
ITcSmTreeItem analyticsConfig = sysManager.LookupTreeItem("TIAN");

analyticsConfig.CreateChild("MyDataLoggerName", 1, null, null);

analyticsConfig.DeleteChild("MyDataLoggerName");
```

4.3.9.3.2 Erstellen und Löschen eines StreamHelpers

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics Configuration Node
ITcSmTreeItem analyticsConfig = sysManager.LookupTreeItem("TIAN");

analyticsConfig.CreateChild("MyStreamHelperName", 0, null, null);

analyticsConfig.DeleteChild("MyStreamHelperName_Obj1 (StreamHelper)");
```

Beim Löschen ist der Zusatz „_Obj1 (StreamHelper)“, der dem in CreateChild übergebenen Namen angehängt wird, zu beachten.

4.3.9.3.3 Parametrieren eines bereits angelegten DataLoggers

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
```

```
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics DataLogger Node
ITcSmTreeItem dataLogger = sysManager.LookupTreeItem("TIAN^MyDataLoggerName");

string sXmlDoc = dataLogger.ProduceXml();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(sXmlDoc);

XmlElement elemSetParam = xmlDoc.CreateElement("SetParameter");

XmlAttribute attrParamName = xmlDoc.CreateAttribute("name");
attrParamName.Value = "Data Format";

XmlAttribute attrParamValue = xmlDoc.CreateAttribute("value");
attrParamValue.Value = "ANALYTICS_FORMAT_FILE";
```

```
elemSetParam.Attributes.Append(attrParamName);
elemSetParam.Attributes.Append(attrParamValue);
xmlDoc.DocumentElement.AppendChild(elemSetParam);

string sConsumeXml = xmlDoc.OuterXml;
dataLogger.ConsumeXml(sConsumeXml);
```

Der Codeausschnitt zeigt die Verwendung des Produce-Consume-XML Mechanismus, bei dem der XML-Text, der den Projektknoten beschreibt, eingelesen, verändert und wieder geschrieben werden kann, woraufhin Aktionen seitens des System Managers folgen können. Der eingelesene XML-Text beinhaltet unter anderem eine Auflistung der Parameter mit Namen und aktuellem Wert. Der Name entspricht genau der Bezeichnung des Parameters unter dem **Parameter (Init)**-Reiter des DataLogger-Projektknotens. Das trifft auch für die jeweiligen Werte zu.

4.3.9.3.4 Parametrieren eines Streams

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics DataLogger Node
ITcSmTreeItem stream = sysManager.LookupTreeItem("TIAN^MyDataLoggerName^PlcStream1");

string sXmlDoc = stream.ProduceXml();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(sXmlDoc);

XmlElement elemSetParam = xmlDoc.CreateElement("SetParameter");

XmlAttribute attrParamName = xmlDoc.CreateAttribute("name");
attrParamName.Value = "Max ADS Buffer";

XmlAttribute attrParamValue = xmlDoc.CreateAttribute("value");
attrParamValue.Value = "23";

elemSetParam.Attributes.Append(attrParamName);
elemSetParam.Attributes.Append(attrParamValue);
xmlDoc.DocumentElement.AppendChild(elemSetParam);

string sConsumeXml = xmlDoc.OuterXml;
dataLogger.ConsumeXml(sConsumeXml);
```

Der Codeausschnitt zeigt die Verwendung des Produce-Consume-XML Mechanismus, bei dem der XML-Text, der den Projektknoten beschreibt, eingelesen, verändert und wieder geschrieben werden kann, woraufhin Aktionen seitens des System Managers folgen können. Der eingelesene XML-Text beinhaltet unter anderem eine Auflistung der Parameter mit Namen und aktuellem Wert. Der Name entspricht genau der Bezeichnung des Parameters unter dem **Data Handling**-Reiter des Stream-Projektknotens. Das trifft auch für die jeweiligen Werte zu.

4.3.9.3.5 Auswählen von Symbolen

```
EnvDTE.Solution solution = dte.Solution;
EnvDTE.Project xaeProject = null;

// Get xaeProject
…
ITcSysManager sysManager = (ITcSysManager)xaeProject.Object;

// Navigate to the Analytics DataLogger Node
ITcSmTreeItem stream = sysManager.LookupTreeItem("TIAN^MyDataLoggerName^PlcStream1");

string sXmlDoc = stream.ProduceXml();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(sXmlDoc);

XmlElement elemSetSymbol = xmlDoc.CreateElement("SetSymbol");
```

```
XmlAttribute attrSymbolName = xmlDoc.CreateAttribute("name");
attrSymbolName.Value = "MAIN.stTestStructSimple.nMember1";

XmlAttribute attrEnable = xmlDoc.CreateAttribute("value");
attrEnable.Value = "true";

elemSetSymbol.Attributes.Append(attrSymbolName);
elemSetSymbol.Attributes.Append(attrEnable);

xmlDoc.DocumentElement.AppendChild(elemSetSymbol);

string sConsumeXml = xmlDoc.OuterXml;
stream.ConsumeXml(sConsumeXml);
```

4.3.10 Motion

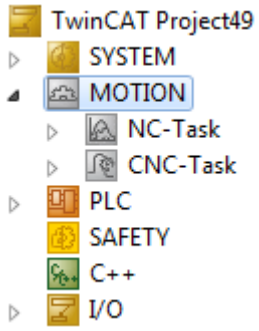
4.3.10.1 Erstellung von und Umgang mit Motion-Projekten

In diesem Kapitel wird die Erstellung und der Umgang mit Motion-Projekten ausführlich erklärt. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über TwinCAT 3 Motion
- Ein NC-Task erstellen
- Achsen erstellen
- Achsen parametrisieren

Allgemeine Informationen über TwinCAT Motion

TwinCAT 3 Motion besteht aus den drei folgenden Komponenten: NC-I, NC-PTP und CNC. Es ist daher eine Zusammenstellung von Funktionsgruppen, die für die Steuerung und Regelung von Achsen oder von synchronisierten Achsgruppen verwendet wird. Weitere Informationen über TwinCAT Motion finden Sie in TwinCAT 3 Motion Dokumentation.



Ein NC-Task erstellen

Der erste Schritt bei der Erstellung eines TwinCAT 3 Motion-Projekts ist die Erstellung eines sogenannten NC-Tasks. Im TwinCAT-Automation Interface können Sie dieses Task einfach mittels Aufruf der Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] und Verwendung des SubType Parameter 1 erstellen, wie in folgendem Code-Ausschnitt gezeigt.

Code-Ausschnitt (C#):

```
ITcSmTreeItem ncConfig = systemManager.LookupTreeItem("TINC");
ncConfig.CreateChild("NC-Task", 1);
```

Code-Ausschnitt (Powershell):

```
$ncConfig = $systemManager.LookupTreeItem("TINC")
$ncConfig.CreateChild("NC-Task", 1)
```

Achsen erstellen

Da ein NC-Task standardmäßig keine Achsen beinhaltet, müssen Sie diese hinzufügen, erneut unter Verwendung der CreateChild()-Methode, nur diesmal auf einer Referenz zu einem Achsknoten unterhalb des NC-Task.

Code-Ausschnitt (C#):

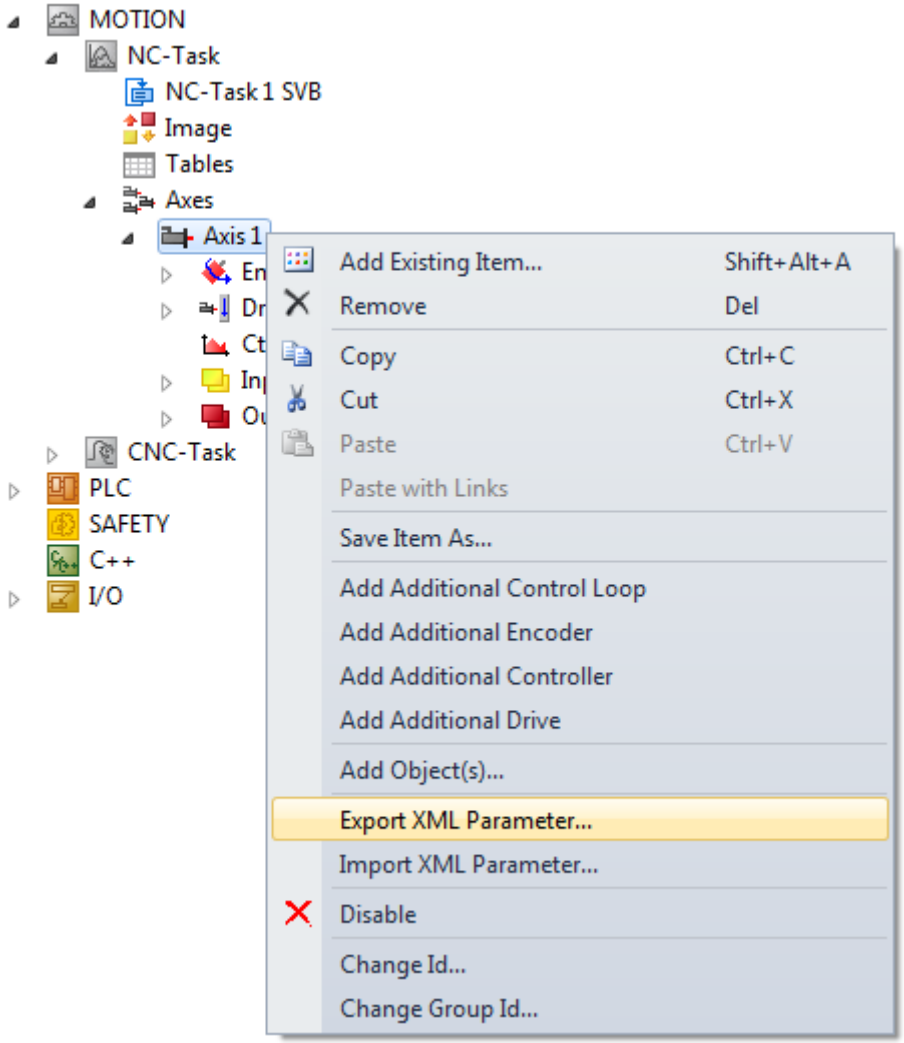
```
ITcSmTreeItem axes = systemManager.LookupTreeItem("TINC^NC-Task^Axes");
axes.CreateChild("Axis 1", 1);
```

Code-Ausschnitt (Powershell):

```
$axes = $systemManager.LookupTreeItem("TINC^NC-Task^Axes")
$axes.CreateChild("Axis 1", 1)
```

Achsen parametrisieren

Achsen können über ihre XML-Beschreibung [▶ 25] parametrisiert werden. Um eine beispielhafte XML-Beschreibung zu erhalten, können Sie eine Achse manuell in TwinCAT XAE hinzufügen und den entsprechenden Eintrag im Kontextmenü verwenden oder die Achse über das Automation Interface hinzufügen und die Methode ITcSmTreeItem [▶ 127]::ProduceXml() [▶ 158] verwenden, um sie zu erhalten.



Code-Ausschnitt (C#):

```
ITcSmTreeItem axis = systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1");
string xmlDescription = axis.ProduceXml();
```

Code-Ausschnitt (Powershell):

```
$axis = $systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1")
$xmlDescription = $axis.ProduceXml()
```

Sie können diese XML-Beschreibung Ihren Bedürfnissen entsprechend anpassen und dann erneut mit der Methode ITcSmTreeItem [▶ 127]::ConsumeXml() [▶ 159] importieren, um Ihre Achse zu parametrisieren.

Code-Ausschnitt (C#):

```
ITcSmTreeItem axis = systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1");
axis.ConsumeXml(xmlDescription);
```

Code-Ausschnitt (Powershell):

```
$axis = $systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1")
$axis.ConsumeXml($xmlDescription)
```

4.3.11 Sicherheit

4.3.11.1 Erstellung von und Umgang mit Sicherheitsprojekten

In diesem Kapitel wird die Erstellung von, der Zugriff auf und der Umgang mit Sicherheitsprojekten ausführlich erklärt. Die folgende Liste enthält alle Kapitel dieses Artikels:

- Allgemeine Informationen über Sicherheitsprojekte
- Bestehende Sicherheitsprojekte öffnen

Allgemeine Informationen über Sicherheitsprojekte

- Das TwinCAT Automation Interface ermöglicht den Import bestehender TwinCAT-Sicherheitsprojekte in die TwinCAT-Konfiguration. Zu diesem Zweck können Benutzer entweder die entsprechende *.splcproj Datei oder das Containerformat *.tfzip als Quellvorlage verwenden.

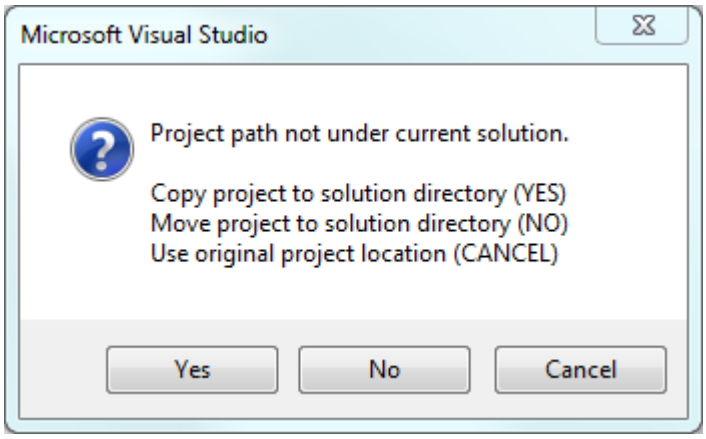
Bestehende Sicherheitsprojekte öffnen

Zum Öffnen eines bestehenden Sicherheitsprojekts mit Hilfe von Automation Interface, müssen Sie zum Sicherheitsknoten navigieren und dann die CreateChild()-Methode mit dem Pfad der entsprechenden Sicherheitsprojektdatei als Parameter ausführen.

Sie können drei verschiedene Werte als SubType verwenden:

- 0: Projekt zum Lösungsverzeichnis kopieren
- 1: Projekt zum Lösungsverzeichnis verschieben
- 2: Original-Projektspeicherort verwenden (falls verwendet, verwenden Sie bitte "" als Projektnamensparameter)

Grundsätzlich repräsentieren diese Werte die Funktionalitäten (Ja, Nein, Abbrechen) von der folgenden MessageBox in TwinCAT XAE:



Sie können entweder den Pfad zum Sicherheitsprojekt (bzw. dessen *.splcproj Datei), das hinzugefügt werden muss, verwenden oder Sie können die Sicherheitsprojekt-Archivdatei (*.tfzip) verwenden.

Code-Ausschnitt (C#):

```
ITcSmTreeItem safety = systemManager.LookupTreeItem("TISC");
ITcSmTreeItem newProject = safety.CreateChild("NameOfProject", 0, null, pathToProjectOrTfzipFile);
```

Code-Ausschnitt (Powershell):

```
$safety = $systemManager.LookupTreeItem("TISC")
$newProject = $safety.CreateChild("NameOfProject", 0, "", pathToProjectOrTfzipFile)
```

5 API

5.1 Referenz

Dieses Kapitel dokumentiert alle Klassen und Methoden des TwinCAT-Automation Interface. Die bereitgestellten Schnittstellen können in zwei unterschiedlichen „Ebenen“ unterteilt werden, wobei die Schnittstellen der höheren Ebene die primären – und demzufolge für die grundlegende Interaktion mit dem Automation Interface zuständigen – Schnittstellen bilden. Beachten Sie, dass diese Differenzierung nur einem logischen Standpunkt geschuldet ist, um besser zu verstehen, welche Schnittstellen die wichtigsten und welche Schnittstellen von zweitrangiger Bedeutung sind.

Schnittstellen der Ebene 1

Wie bereits in unserer Einleitung [▶ 10] erwähnt, gibt es nur zwei Hauptschnittstellen, die für die Navigation und Referenzierung von Tree items in einer TwinCAT-Konfiguration verwendet werden.

Hauptklasse	Beschreibung	Verfügbar seit
ITcSysManager [▶ 118]	Basisklasse für die Erstellung und Parametrisierung einer TwinCAT-Konfiguration	TwinCAT 2.11
ITcSmTreeItem [▶ 127]	Stellt ein Tree item innerhalb einer TwinCAT-Konfiguration dar	TwinCAT 2.11

Schnittstellen der Ebene 2

Diese Schnittstellen werden als "Helferklassen" betrachtet, die immer im Verbund mit Klassen der Ebene 1 verwendet werden, um z.B. ein ITcSmTreeItem-Objekt in einen spezifischeren Tree item-typ umzuwandeln, zum Beispiel eine POU (ITcPlcPou) oder eine verknüpfte Task (ITcTaskReference).

Helferklasse	Beschreibung	Verfügbar seit
ITcPlcLibraryManager [▶ 171]	Definiert Methoden und Eigenschaften für die SPS-Bibliotheksverwaltung	TwinCAT 3.1
ITcPlcPou [▶ 166]	Definiert Methoden und Eigenschaften für den Umgang mit SPS-POUs	TwinCAT 3.1
ITcPlcDeclaration [▶ 167]	Definiert Methoden zum Lesen/Schreiben des Deklarationsbereichs einer SPS-POU	TwinCAT 3.1
ITcPlcImplementation [▶ 167]	Definiert Methoden zum Lesen/Schreiben des Implementierungsbereichs einer SPS-POU	TwinCAT 3.1
ITcPlcProject [▶ 164]	Definiert Methoden und Eigenschaften bezüglich eines SPS-Projekts, z.B. Einstellung des Projekts als Boot-Projekt.	TwinCAT 3.1
ITcPlcIECProject [▶ 168]	Definiert Methoden, die für den Import/Export von SPS-Projekten in/aus PLCopen XML und für deren Installierung als SPS-Bibliothek benötigt werden.	TwinCAT 3.1
ITcPlcTaskReference [▶ 181]	Definiert Methoden und Eigenschaften, um das SPS-Projekt mit einer Task zu verknüpfen.	TwinCAT 3.1
ITcPlcLibrary [▶ 178]	Helferklasse, die eine einzelne SPS-Bibliothek darstellt	TwinCAT 3.1
ITcPlcLibraries [▶ 178]	Helferklasse, die eine Sammlung von SPS-Bibliotheken darstellt	TwinCAT 3.1
ITCPlcReferences [▶ 177]	Helferklasse, die eine Sammlung von ITcPlcLibRef-Objekten darstellt (und daher Referenzen in einem SPS-Projekt)	TwinCAT 3.1
ITcPlcLibRef [▶ 179]	Helferklasse, die eine Basisklasse für ITcPlcLibrary und ITcPlcPlaceholderRef-Objekte bildet	TwinCAT 3.1
ITcPlcPlaceholderRef [▶ 179]	Helferklasse, die einen einzelnen SPS-Platzhalter darstellt	TwinCAT 3.1
ITcPlcLibRepository [▶ 179]	Helferklasse, die ein einzelnes SPS-Bibliotheks-Repository darstellt	TwinCAT 3.1
ITcPlcLibRepositories [▶ 180]	Helferklasse, die eine Sammlung von SPS-Bibliotheks-Repositories darstellt	TwinCAT 3.1

5.2 ITcSysManager

5.2.1 ITcSysManager

ITcSysManager ist die Hauptschnittstelle des TwinCAT Automation Interface. Diese Schnittstelle ermöglicht grundlegende Operationen zwecks Konfiguration von TwinCAT 3 XAE und besteht aus mehreren Methoden um dies zu tun. Im Laufe der Jahre wurde die ITcSysManager-Schnittstelle mit weiteren Funktionalitäten erweitert, um dem Kunden den Zugriff auf alle Funktionen des Automation Interface zu vereinfachen. Aufgrund von Einschränkungen in den COM-Objektmodellen mussten diese neuen Funktionen als getrennte Schnittstellen dem Automation Interface hinzugefügt werden. Aus diesem Grunde wurden diese Eigenschaften, jedes Mal wenn ein neuer Eigenschaftensatz hinzugefügt wurde, in einer neuen Schnittstelle zusammengefasst, die dann ITcSysManagerX genannt wurde, wobei X eine Zahl ist, die bei jedem Hinzufügen einer neuen Schnittstelle inkrementiert wurde. Die folgenden Tabellen erläutern, welche Methoden Bestandteil der ITcSysManager-Schnittstelle sind, und welche jeweils einer neuen “;Eigenschaftensatz”;-Schnittstelle hinzugefügt wurden.

Methoden

ITcSysManager-Methoden	Beschreibung	Verfügbar seit
NewConfiguration [▶ 120]	Erzeugt eine neue Konfiguration	TwinCAT 2.11
OpenConfiguration [▶ 120]	Lädt eine zuvor erzeugte Konfigurationsdatei (WSM-Datei).	TwinCAT 2.11
SaveConfiguration [▶ 121]	Speichert die Konfiguration in eine Datei mit angegebenem Namen oder mit dem aktuellen Namen.	TwinCAT 2.11
ActivateConfiguration [▶ 121]	Aktiviert die Konfiguration (genau wie „;Save To Registry“; (In Registry speichern))	TwinCAT 2.11
LookupTreeItem [▶ 124]	Sucht nach einem Konfigurationselement (Element im Baum) nach Namen und gibt eine ITcSmTreeItem [▶ 127]-Schnittstelle zurück.	TwinCAT 2.11
StartRestartTwinCAT [▶ 122]	Startet oder startet erneut das TwinCAT-System	TwinCAT 2.11
IsTwinCATStarted [▶ 121]	Prüft, ob das TwinCAT-System bereits läuft.	TwinCAT 2.11
LinkVariables [▶ 122]	Verknüpft zwei Variablen, die mit Namen angegeben werden	TwinCAT 2.11
UnlinkVariables [▶ 123]	Löscht die Verknüpfung zwischen zwei mit Namen bezeichneter Variablen oder alle Verknüpfungen zu einer Variablen.	TwinCAT 2.11

ITcSysManager2-Methoden	Beschreibung	Verfügbar seit
SetTargetNetId [▶ 124]	Legt die Ziel-NetId der aktuell geöffneten TwinCAT-Konfiguration fest.	TwinCAT 2.11
GetTargetNetId [▶ 124]	Ruft die Ziel-NetId der aktuell geöffneten TwinCAT-Konfiguration ab.	TwinCAT 2.11
GetLastErrorMessages [▶ 124]	Ruft die letzten Fehlermeldungen ab, die im TwinCAT-Subsystem aufgetreten sind.	TwinCAT 2.11

ITcSysManager3-Methoden	Beschreibung	Verfügbar seit
LookupTreeItemById [▶ 125]	Sucht nach einem Tree Item mit spezifizierter Element-Id.	TwinCAT 2.11
ProduceMappingInfo [▶ 126]	Erstellt eine Xml-Beschreibung des eigentlichen Konfigurations-Mappings.	TwinCAT 3.1
ClearMappingInfo	Löscht die Mapping-Informationen.	TwinCAT 2.11

Kommentare

Die ITcSysManager-Schnittstelle enthält zwei Methoden für das Navigieren innerhalb von TwinCAT XAE: ITcSysManager::LookupTreeItem [▶ 124] und ITcSysManager3::LookupTreeItemById [▶ 125]. Eine ausführliche Erläuterung bezüglich dem Browsen von TwinCAT XAE finden Sie im Kapitel Tree item-Browsingmodelle [▶ 23].

Achtung: Die drei Methoden ITcSysManager::NewConfiguration [▶ 120], ITcSysManager::OpenConfiguration [▶ 120] und ITcsSysManager::SaveConfiguration [▶ 121] sind ausschließlich im Kompatibilitätsmodus [▶ 20] verfügbar. Sie im Standardmodus aufzurufen löst eine E_NOTSUPPORTED-Ausnahme aus.

Die **ITcSysmanager-** und ITcSmTreeItem [▶ 127]-Schnittstellen ermöglichen uneingeschränkten Zugriff auf eine TwinCAT-Konfiguration. Im Abschnitt „;Vorgehensweise...“; dieser Dokumentation finden Sie eine lange (aber unvollständige) Liste von Beispielen, wie eine TwinCAT-Konfiguration automatisch bearbeitet werden kann.

Versionsinformationen

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 2.11 unterstützt.

5.2.2 ITcSysManager::NewConfiguration

Die NewConfiguration()-Methode erzeugt eine neue TwinCAT-Konfigurationsdatei.

```
HRESULT NewConfiguration();
```

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_ACCESSDENIED	das aktuelle Dokument in der System Manager-Instanz ist gesperrt. Dies ist der Fall, wenn mindestens eine Referenz zum System Manager-Objekt oder eines der Tree Items geöffnet ist.
E_FAIL	die Funktion schlägt fehl.

Kommentare

Achtung: Die drei Methoden ITcSysManager::NewConfiguration [▶ 120], ITcSysManager::OpenConfiguration [▶ 120] und ITcsSysManager::SaveConfiguration [▶ 121] sind ausschließlich im Kompatibilitätsmodus [▶ 20] verfügbar. Sie im Standardmodus aufzurufen löst eine E_NOTSUPPORTED-Ausnahme aus.

5.2.3 ITcSysManager::OpenConfiguration

Die OpenConfiguration()-Methode lädt eine zuvor erzeugte TwinCAT-Konfigurationsdatei.

```
HRESULT OpenConfiguration(BSTRbstrFile);
```

Parameter

bstrFile	[in, defaultvalue(L"")] enthält den Dateipfad der zu ladenden Konfigurationsdatei oder eine leere Zeichenkette, wenn eine neue Konfiguration erzeugt werden soll. Die derzeit laufende Konfiguration eines Zielgeräts kann auch unter Verwendung von „;CURRENTCONFIG“; gelesen werden.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_ACCESSDENIED	das aktuelle Dokument in der System Manager-Instanz ist gesperrt. Dies ist der Fall, wenn mindestens eine Referenz zum System Manager-Objekt oder eines der Tree Items geöffnet ist.
E_INVALIDARG	Der angegebene Dateipfad verweist nicht auf eine gültige Konfigurationsdatei.

Kommentare

Achtung: Die drei Methoden ITcSysManager::NewConfiguration [▶ 120], ITcSysManager::OpenConfiguration [▶ 120] und ITcsSysManager::SaveConfiguration [▶ 121] sind ausschließlich im Kompatibilitätsmodus [▶ 20] verfügbar. Sie im Standardmodus aufzurufen löst eine E_NOTSUPPORTED-Ausnahme aus.

5.2.4 ITcSysManager::SaveConfiguration

Die SaveConfiguration()-Methode speichert eine TwinCAT-Konfiguration in eine Datei mit angegebenem Namen.

```
HRESULT SaveConfiguration(BSTRbstrFile);
```

Parameter

bstrFile	[in, defaultvalue(L"")] enthält den Dateipfad, unter dem die Konfigurationsdatei gespeichert werden soll. Wenn *bstrFile* eine leere Zeichenkette ist, dann wird der aktuelle Dateiname verwendet.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_INVALIDARG	der angegebene Dateipfad ist ungültig.

Kommentare

Achtung: Die drei Methoden ITcSysManager::NewConfiguration [▶ 120], ITcSysManager::OpenConfiguration [▶ 120] und ITcsSysManager::SaveConfiguration [▶ 121] sind ausschließlich im Kompatibilitätsmodus [▶ 20] verfügbar. Sie im Standardmodus aufzurufen löst eine E_NOTSUPPORTED-Ausnahme aus.

5.2.5 ITcSysManager::ActivateConfiguration

Die ActivateConfiguration()-Methode aktiviert die TwinCAT-Konfiguration. Anschließend muss das TwinCAT-System gestartet oder neu gestartet werden, um die Konfiguration physisch zu aktivieren.

```
HRESULT ActivateConfiguration();
```

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_FAIL	die Funktion schlägt fehl.

5.2.6 ITcSysManager::IsTwinCATStarted

Die IsTwinCATStarted()-Methode prüft, ob das TwinCAT-System bereits ausgeführt wird.

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

Parameter

pStarted	[out, retval] Zeigt auf den Speicherort eines boolschen Werts, der das Ergebnis erhält.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.

5.2.7 ITcSysManager::StartRestartTwinCAT

Die StartRestartTwinCAT()-Methode startet oder startet erneut das TwinCAT-System. Wenn TwinCAT bereits gestartet wurde, führt die Funktion einen Neustart aus, wenn TwinCAT gestoppt ist, führt sie einen Start aus.

```
HRESULT StartRestartTwinCAT();
```

Rückgabewerte

Voraussetzungen

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_FAIL	TwinCAT konnte nicht gestartet werden.

5.2.8 ITcSysManager::LinkVariables

Die LinkVariables()-Methode verknüpft zwei Variablen, die mit ihrem Namen spezifiziert werden. Die beiden mit ihrem jeweiligen Pfadnamen bezeichneten Variablen werden verknüpft. Die Syntax der Pfadnamen muss der in ITcSysManager::LookupTreeItem [▶ 124] beschriebenen Syntax entsprechen. Die gleichen Kurzformen können verwendet werden.

```
HRESULT LinkVariables(BSTRbstrV1, BSTRbstrV2, longoffs1, longoffs2, longsize);
```

Parameter

bstrV1	[in] Pfadname der ersten Variablen. Der vollständige Pfadname ist erforderlich und jeder Zweig muss durch ein Zirkumflex ‚;^' oder Tabulatorzeichen getrennt sein.
bstrV2	[in] Pfadname der zweiten Variablen. Der vollständige Pfadname ist erforderlich und jeder Zweig muss durch ein Zirkumflex ‚;^' oder Tabulatorzeichen getrennt sein.
offs1	[in, defaultvalue(0)] Bit-Offset der ersten Variablen (wird verwendet, wenn die beiden Variablen unterschiedlicher Größe sind oder wenn nicht die ganze Variable verknüpft werden soll).
offs2	[in, defaultvalue(0)] Bit-Offset der zweiten Variablen.
Größe	[in, defaultvalue(0)] Anzahl der Bits, die verknüpft werden sollen. Wenn *Größe* gleich 0 ist, dann wird der Kleinstwert der Variablengröße von Variable 1 und 2 verwendet.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
TSM_E_ITEMNOTFOUND (0x98510001)	ein oder beide Pfadname(n) verweist(verweisen) nicht auf ein bestehendes Tree Item.
TSM_E_INVALIDITEMTYPE (0x98510002)	ein oder beide Tree item(s) ist (sind) keine Variable.
TSM_E_MISMATCHINGITEMS (0x98510004)	die beiden Variablen können nicht miteinander verknüpft werden. Möglicherweise haben Sie versucht, den Ausgang von einer Task mit einem Ausgang von einer anderen Task, oder einen Ausgang von einer Task mit dem Eingang eines Geräts, oder zwei Variablen vom gleichen Eigentümer zu verbinden.
E_INVALIDARG	die Werte von *offs1*, *offs2* und/oder *Größe* passen nicht zu den Variablen.

5.2.9 ITcSysManager::UnlinkVariables

Die UnlinkVariables()-Methode hebt die Verknüpfung von zwei Variablen auf, die mit ihren Namen angegeben werden, oder hebt alle Verknüpfungen der ersten Variablen auf, wenn der Name *bstrV2* der zweiten Variablen leer ist. Die Verknüpfung der beiden mit ihrem Pfadnamen bezeichneten Variablen wird aufgehoben. Die Syntax der Pfadnamen muss der in ITcSysManager::LookupTreeItem [▶ 124] beschriebenen Syntax entsprechen. Die gleichen Kurzformen können verwendet werden.

```
HRESULT UnlinkVariables(BSTRbstrV1, BSTRbstrV2);
```

Parameter

bstrV1	[in] Pfadname der ersten Variablen. Der vollständige Pfadname ist erforderlich und jeder Zweig muss durch ein Zirkumflex ,;^’; oder Tabulatorzeichen getrennt sein.
bstrV2	[in, defaultvalue(L"")] Pfadname der zweiten Variablen. Wenn gesetzt, dann ist der vollständige Pfadname erforderlich und jeder Zweig muss durch ein Zirkumflex ,;^’; oder Tabulatorzeichen getrennt sein.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
S_FALSE	es besteht keine Verknüpfung zwischen den beiden Variablen.
TSM_E_ITEMNOTFOUND (0x98510001)	ein oder beide Pfadname(n) verweist(verweisen) nicht auf ein bestehendes Tree Item.
TSM_E_INVALIDITEMTYPE (0x98510002)	ein oder beide Tree item(s) ist (sind) keine Variable.
TSM_E_CORRUPTEDLINK (0x98510005)	die Verknüpfung zwischen den beiden Variablen kann nicht aufgehoben werden.

Kommentare

Wenn *bstrV2* eine leere Zeichenkette ist, dann werden alle Verknüpfungen der durch *bstrV1* gegebenen Variablen aufgehoben. Wenn *bstrV2* keine leere Zeichenkette ist, dann wird nur die zwischen den beiden Variablen bestehende Verknüpfung aufgehoben.

5.2.10 ITcSysManager2::GetTargetNetId

Die GetTargetNetId()-Methode gibt die NetId des aktuellen TwinCAT-Systems zurück.

```
HRESULT GetTargetNetId();
```

Parameter

Keine

Rückgabewerte

STRING	Gibt Ziel-NetId zurück.

5.2.11 ITcSysManager2::SetTargetNetId

Die SetTargetNetId()-Methode legt die NetId des aktuellen TwinCAT-Systems fest.

```
HRESULT SetTargetNetId(STRING netId);
```

Parameter

netId	ist die Ziel-NetId.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.

5.2.12 ITcSysManager2::GetLastErrorMessages

Die GetLastErrorMessages()-Methode gibt die letzten Fehlermeldungen zurück.

```
HRESULT GetLastErrorMessages();
```

Parameter

Keine

Rückgabewerte

STRING	Gibt letzte Fehlermeldungen zurück.

5.2.13 ITcSysManager::LookupTreeItem

Die LookupTreeItem()-Methode gibt einen *ITcTreeItem*-Zeiger eines Tree Items zurück, das mit seinem vollständigen Pfadnamen angegeben wird.

```
HRESULT LookupTreeItem(BSTRbstrItem, ITcSmTreeItem**pipItem);
```

Parameter

bstrItem	[in] Pfadname des von Ihnen gesuchten Tree Items. Der vollständige Pfadname ist erforderlich und jeder Zweig muss durch ein Zirkumflex ‚^' oder Tabulatorzeichen getrennt sein. Eine Liste der Kurzformen der wichtigsten Tree Items wird unten angegeben.
pipItem	[out, retval] zeigt auf den Speicherort eines ITcSmTreeItem [▶ 127]-Schnittstellenzeigers bei Rückgabe. Der Schnittstellenzeiger ermöglicht den Zugriff auf spezifische, zum Tree Item gehörige Methoden.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
TSM_E_ITEMNOTFOUND (0x98510001)	der Pfadname verweist nicht auf ein bestehendes Tree Item.

Kurzformen

Auf die wichtigsten Tree Item, die in jeder Konfigurationsdatei bestehen, kann über Kurzformen zugegriffen werden. Diese Kurzformen sind sprachenneutral und erfordern weniger Speicherplatz:

```
"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"
```

Beispiel (C++):

```
ITcSmTreeItem* ipItem;

BSTR bstrItem = L"TIID^Device 1 (C1220)";

if ( SUCCEEDED(spTsm->LookupTreeItem( bstrItem, &ipItem ))
)

{

// do anything with ipItem


ipItem->Release();

}
```

Beispiel (VB): Dim ipItem As ITcSmTreeItem

set ipItem = spTsm.LookupTreeItem("TIID^Device 1 (C1220)")

' do anything with ipItem

Kommentare

5.2.14 ITcSysManager3::LookupTreeItemById

Die LookupTreeItemById()-Methode gibt einen *ITcTreeItem*-Zeiger eines Tree Items zurück, das mit seinem vollständigen Pfadnamen angegeben wird.

```
HRESULT LookupTreeItemById(longitemType, longitemId, ITcSmTreeItem**pipItem);
```

Parameter

itemType	[in] Elementtyp des gesuchten TreeItem.
itemId	[in] ID von TreeItem
pipItem	[out, retval] zeigt auf den Speicherort eines ITcSmTreeItem [▶ 127]-Schnittstellenzeigers bei Rückgabe. Der Schnittstellenzeiger ermöglicht den Zugriff auf spezifische, zum Tree Item gehörige Methoden.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
TSM_E_ITEMNOTFOUND (0x98510001)	die *itemType itemId*-Kombination stellt kein gültiges Tree Item dar.

5.2.15 ITcSysManager3::ProduceMappingInfo

Generiert eine XML-Struktur, welche alle derzeit konfigurierten Mappings enthält, z.B. zwischen SPS und I/O.

```
HRESULT ProduceMappingInfo();
```

Parameter

none

Rückgabewerte

STRING: Liefert die XML-Struktur mit allen konfigurierten Mappings. Das folgende Code Snippet zeigt ein Beispiel für diese Struktur:

```
<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>
```

Dieses Beispiel zeigt Mappings zwischen SPS <--> I/O und TcCOM (C++) <--> I/O.

5.2.16 ITcSysManager3::ConsumeMappingInfo

Importiert eine XML-Struktur, welche die Mapping-Informationen für ein Projekt enthält.

```
HRESULT ConsumeMappingInfo(BSTR bstrXml);
```

Parameter

bstrXml

[in]: String mit XML-Struktur. Die XML Mapping Informationen können durch Aufruf der Methode ITcSysManager3::ProduceMappingInfo() [▶ 126] ermittelt werden.

5.3 ITcSmTreeItem

5.3.1 ITcSmTreeItem

Jedes Tree item in einer TwinCAT XAE Konfiguration wird mit Hilfe einer Instanz der *ITcSmTreeItem*-Klasse dargestellt, welche verschiedene Interaktionen mit dem Tree Item ermöglicht.

Ein Tree Item dieser Schnittstelle wird z.B. beim Aufruf der ITCatSysManager::LookupTreeItem-Methode zurückgegeben, welches dann zum Navigieren durch die Baumstruktur verwendet wird.

Eigenschaften

ITcSmTreeItem-Properties	Typ	Zugriff	Beschreibung
Name	BSTR	RW	Name des Tree Items
Comment	BSTR	RW	Kommentar
Disabled	BOOL	RW	Abrufen / Setzen des Zustands eines Tree Items, welcher einem der folgenden Aufzählwerte entsprechen kann: • SMDS_NOT_DISABLED (Element ist aktiviert) • SMDS_DISABLED (Element ist deaktiviert) • SMDS_PARENT_DISABLED (**schreibgeschützt**, gesetzt, wenn eines seiner übergeordneten Elemente deaktiviert ist)
PathName	BSTR	R	Pfad des Tree Items in TwinCAT XAE. Die Zweige werden durch '^' getrennt. Der Pfadname kann in anderen Methoden, wie z.B. ITCatSysManager::LookupTreeItem [▶ 124] verwendet werden. Beachten Sie, dass diese Eigenschaft ein Tree Item in TwinCAT XAE eindeutig identifiziert.
ItemType	ENUM	R	Kategorisierung eines Tree Items, z.B. Geräte, Boxen, SPS, ... Wie mittels Item Types [▶ 128] festgelegt.
ItemSubType	LONG	RW	Subtyp [▶ 131] eines Tree Items.
Parent	ITcSmTreeItem*	R	Zeiger auf das übergeordnete Tree Item.
ChildCount	LONG	R	Anzahl untergeordneter Elemente. Diese Eigenschaft zählt lediglich untergeordnete Hauptelemente des Tree Items (z.B. Boxen sind untergeordnete Hauptelemente von einem Gerät, aber nicht das Geräteprozessabbild). Um auf alle untergeordneten Elemente zuzugreifen, verwenden Sie die Eigenschaft _NewEnum.
Child(LONG n)	ITcSmTreeItem*	R	ITcSmTreeItem Zeiger auf das n-te untergeordnete Element
VarCount((LONG x)	LONG	R	Anzahl der zum Tree Item zugehörigen Variablen. x = 0 Zählung der Eingangsvariablen, x = 1 Ausgangsvariablen
Var(LONG x, LONG n)	ITcSmTreeItem*	R	ITcSmTreeItem Zeiger auf die n-te Variable. x = 0 verwendet die Eingangsvariablen, x = 1 Ausgangsvariablen
_NewEnum	IUnknown* (IEnumVariant*)	R	Gibt eine Aufzählungsschnittstelle zurück, die **alle** untergeordneten Tree Items des aktuellen Tree Items aufzählt. Diese Eigenschaft kann z.B. bei einer For-Each Anweisung verwendet werden.

Methoden

ITcSmTreeItem-Methoden	Beschreibung	Verfügbar seit
CreateChild [▶ 160]	Erzeugt ein untergeordnetes Tree Item.	TwinCAT 2.11
DeleteChild [▶ 162]	Löscht ein untergeordnetes Tree Item.	TwinCAT 2.11
ImportChild [▶ 163]	Importiert ein untergeordnetes Tree Item aus der Zwischenablage oder einer zuvor exportierten Datei.	TwinCAT 2.11
ExportChild [▶ 163]	Exportiert ein untergeordnetes Tree Item in die Zwischenablage oder in eine Datei.	TwinCAT 2.11
ProduceXml [▶ 158]	Gibt eine Zeichenkette mit der XML-Darstellung eines Elements zurück, mitsamt dessen elementspezifischen Daten und Parametern.	TwinCAT 2.11
ConsumeXml [▶ 159]	Verwendet eine Zeichenkette mit der XML-Darstellung eines Tree Items, mitsamt dessen elementspezifischen Daten und Parametern.	TwinCAT 2.11
GetLastXmlError [▶ 164]	Ruft die Fehlermeldung des letzten gescheiterten ConsumeXml()-Aufrufs ab.	TwinCAT 2.11
LookupChild [▶ 164]	Suche nach einem untergeordneten Element mit dem angegebenen relativen Pfad.	TwinCAT 2.11

ITcSmTreeItem2-Methoden	Beschreibung	Verfügbar seit
ResoucesCount	Nur für internen Gebrauch	TwinCAT 2.11
ChangeChildSubType	Ändert den Subtyp von ITcSmTreeItem.	TwinCAT 2.11
ClaimResources	Nur für internen Gebrauch	TwinCAT 2.11

Versionsinformationen

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 2.11 unterstützt.

5.3.2 ITcSmTreeItem Item Types

Jedes Tree Item im TwinCAT System Manager / TwinCAT XAE wird in verschiedene **Gruppen kategorisiert** , z.B. Geräte, Boxen, Tasks, ... Sie können den Elementtyp eines Tree Items überprüfen, indem sie es manuell in den TwinCAT System Manager oder XAE hinzufügen und anschließend seine XML-Beschreibung über den entsprechenden Menüeintrag exportieren.

- **TwinCAT System Manager:** Aktionen --> XML-Beschreibung exportieren
- **TwinCAT XAE:** TwinCAT --> Ausgewähltes Element --> XML-Beschreibung exportieren

```
<TreeItem>
    <ItemName>Device 1 (EtherCAT)</ItemName>
    <PathName>TIID^Device 1 (EtherCAT)</PathName>
    <ItemType>2</ItemType>
    <ItemId>1</ItemId>
    <ObjectId>#x03010010</ObjectId>
    <ItemSubType>111</ItemSubType>
```

In der sich daraus ergebenden XML-Datei wird der Elementtyp durch den <ItemType>-Knoten dargestellt.

Allgemeine Elementtypen

Elementtyp	Tag	Beschreibung
0	TREEITEMTYPE_UNKNOWN	---
1	TREEITEMTYPE_TASK	---
9	TREEITEMTYPE_IECPRJ	---
10	TREEITEMTYPE_CNCPRJ	---
11	TREEITEMTYPE_GSDMOD	Modul eines Profibus-GSD-Geräts
12	TREEITEMTYPE_CDL	---
13	TREEITEMTYPE_IECLZS	---
14	TREEITEMTYPE_LZSGRP	---
15	TREEITEMTYPE_IODEF	---
16	TREEITEMTYPE_ADDTASKS	---
17	TREEITEMTYPE_DEVICEGRP	---
18	TREEITEMTYPE_MAPGRP	---
30	TREEITEMTYPE_CANPDO	---
31	TREEITEMTYPE_RTIMESET	---
32	TREEITEMTYPE_BCPLC_VARS	---
33	TREEITEMTYPE_FILENAME	---
34	TREEITEMTYPE_DNETCONNECT	---
37	TREEITEMTYPE_FLBCMD	---
43	TREEITEMTYPE_EIPCONNECTION	---
44	TREEITEMTYPE_PNIOAPI	---
45	TREEITEMTYPE_PNIOMOD	---
46	TREEITEMTYPE_PNIOSUBMOD	---
47	TREEITEMTYPE_ETHERNETPROTOCOL	---
200	TREEITEMTYPE_CAMDEF	---
201	TREEITEMTYPE_CAMGROUP	---
202	TREEITEMTYPE_CAM	---
203	TREEITEMTYPE_CAMENCODER	---
204	TREEITEMTYPE_CAMTOOLGRP	---
205	TREEITEMTYPE_CAMTOOL	---
300	TREEITEMTYPE_LINEDEF	---
400	TREEITEMTYPE_ISGDEF	---
401	TREEITEMTYPE_ISGCHANNEL	---
402	TREEITEMTYPE_ISGAGROUP	---
403	TREEITEMTYPE_ISGAXIS	---
500	TREEITEMTYPE_RTSCONFIG	---
501	TREEITEMTYPE_RTSAPP	---
502	TREEITEMTYPE_RTSAPPTASK	---
503	TREEITEMTYPE_RTSADI	---
504	TREEITEMTYPE_CPPCONFIG	---
505	TREEITEMTYPE_SPLCCONFIG	---

I/O-Elementtypen

Elementtyp	Tag	Beschreibung
2	TREEITEMTYPE_DEVICE	E/A-Gerät
3	TREEITEMTYPE_IMAGE	Prozessabbild
4	TREEITEMTYPE_MAPPING	---
5	TREEITEMTYPE_BOX	I/O-Box (z.B. "BK2000", untergeordnetes Element eines E/A-Gerät)
6	TREEITEMTYPE_TERM	I/O-Klemme (untergeordnetes Element von Klemmenkoppler (Box))
7	TREEITEMTYPE_VAR	Variable
8	TREEITEMTYPE_VARGRP	Variablengruppe (z.B. "Eingänge")
35	TREEITEMTYPE_NVPUBLISHERVAR	---
36	TREEITEMTYPE_NVSUBSCRIBERVAR	---

SPS-Elementtypen

Elementtyp	Tag	Beschreibung
600	TREEITEMTYPE_PLCAPP	SPS-Anwendung (SPS-Stammobjekt) [1]
601	TREEITEMTYPE_PLCFOLDER	SPS-Ordner [1]
602	TREEITEMTYPE_PLCPOUPROG	POU-Programm [1]
603	TREEITEMTYPE_PLCPOUFUNC	POU-Funktion [1]
604	TREEITEMTYPE_PLCPOUFB	POU-Funktionsbaustein [1]
605	TREEITEMTYPE_PLCDUTENUM	DUT-Aufzählungsdatentyp [1]
606	TREEITEMTYPE_PLCDUTSTRUCT	DUT-Strukturdatentyp [1]
607	TREEITEMTYPE_PLCDUTUNION	DUT-Union-Datentyp [1]
608	TREEITEMTYPE_PLCACTION	SPS-Aktion [1]
609	TREEITEMTYPE_PLCMETHOD	SPS-Methode [1]
610	TREEITEMTYPE_PLCITFMETH	SPS-Schnittstellenmethode [1]
611	TREEITEMTYPE_PLCPROP	SPS-Eigenschaft [1]
612	TREEITEMTYPE_PLCITFPROP	SPS-Schnittstelleneigenschaft [1]
613	TREEITEMTYPE_PLCPROPGET	SPS-Eigenschaften-Getter [1]
614	TREEITEMTYPE_PLCPROPSET	SPS-Eigenschaften-Setter [1]
615	TREEITEMTYPE_PLCGVL	GVL (Globale Variablenliste) [1]
616	TREEITEMTYPE_PLCTRANS	SPS-Übergang [1]
617	TREEITEMTYPE_PLCLIBMAN	SPS-Bibliotheksverwalter [1]
618	TREEITEMTYPE_PLCITF	SPS-Schnittstelle [1]
619	TREEITEMTYPE_PLCVISOBJ	SPS visuelles Objekt[1]
620	TREEITEMTYPE_PLCVISMAN	SPS visueller Manager [1]
621	TREEITEMTYPE_PLCTASK	SPS-Taskobjekt [1]
622	TREEITEMTYPE_PLCPROGREF	SPS-Programmreferenz [1]
623	TREEITEMTYPE_PLCDUTALIAS	DUT-Alias
624	TREEITEMTYPE_PLCEXTDATATYPECONT	SPS externer Datentypcontainer [1]
625	TREEITEMTYPE_PLCTMCDESCRIPTION	SPS TMC-Beschreibungsdatei [1]
654	TREEITEMTYPE_PLCITFPROPGET	SPS-Schnittstelleneigenschaften-Getter
655	TREEITEMTYPE_PLCITFPROPSET	SPS-Schnittstelleneigenschaften-Setter

NC-Elementtypen

Elementtyp	Tag	Beschreibung
19	TREEITEMTYPE_NCDEF	---
20	TREEITEMTYPE_NCAXISES	---
21	TREEITEMTYPE_NCCHANNEL	NC-Kanal
22	TREEITEMTYPE_NCAXIS	NC-Achse
23	TREEITEMTYPE_NCENCODER	---
24	TREEITEMTYPE_NCDRIVE	---
25	TREEITEMTYPE_NCCONTROLLER	---
26	TREEITEMTYPE_NCGROUP	---
27	TREEITEMTYPE_NCINTERPRETER	---
40	TREEITEMTYPE_NCTABLEGRP	---
41	TREEITEMTYPE_NCTABLE	---
42	TREEITEMTYPE_NCTABLESLAVE	---

Voraussetzungen

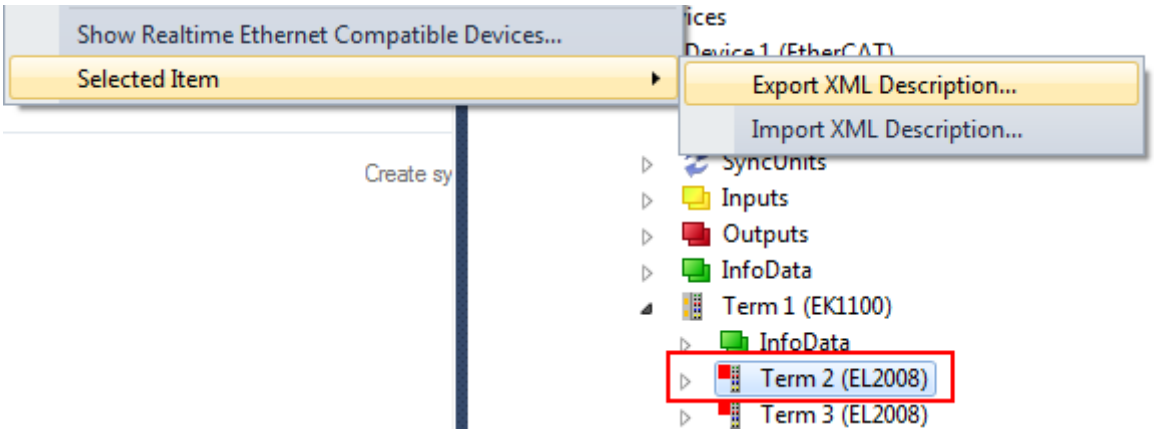
Anmerkungen	
[1]	erfordert TwinCAT 3.1

5.3.3 Tree item sub types

5.3.3.1 ITcSmTreeItem Element-Subtypen

Element-Subtypen legen fest, welche Art **Gerät** , **Box** oder **Klemme** verwendet wird, z.B. ein Subtyp 2408 steht für eine KL2408 Digitalausgangsklemme. Sie können den Subtyp eines Elements überprüfen, indem sie es manuell in den TwinCAT System Manager oder XAE hinzufügen und anschließend seine XML-Beschreibung über den entsprechenden Menüeintrag exportieren.

- **TwinCAT System Manager:** Aktionen --> XML-Beschreibung exportieren
- **TwinCAT XAE:** TwinCAT --> Ausgewähltes Element --> XML-Beschreibung exportieren



In der sich daraus ergebenden XML-Datei wird der Subtyp durch den <ItemSubType>-Knoten dargestellt. Im obigen Beispiel haben wir die XML-Beschreibung einer EL2008-Klemme exportiert. Die XML-Datei zeigt an, dass diese Klemme vom Subtyp 9099 ist.

```
<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>
```

In den folgenden Tabellen finden Sie einen Überblick über einige der verfügbaren Subtypen. Wenn Ihre Gerät nicht aufgelistet ist, gehen Sie bitte wie oben beschrieben vor, um den Subtyp Ihres spezifischen Geräts zu ermitteln.

Verknüpfungen:

- Geräte [▶ 134]
- Boxen [▶ 141]
- Klemmen: E-Bus [▶ 132] (ELxxxx)
- Klemmen: K-Bus-Digitaleingang [▶ 146] (KL1xxx)
- Klemmen: K-Bus-Digitalausgang [▶ 148] (KL2xxx)
- Klemmen: K-Bus-Analogeingang [▶ 151] (KL3xxx)
- Klemmen: K-Bus-Analogausgang [▶ 153] (KL4xxx)
- Klemmen: K-Bus-Lagemessung [▶ 153] (KL5xxx)
- Klemmen: K-Bus-Kommunikation [▶ 154] (KL6xxx)
- Klemmen: K-Bus-Stromversorgung [▶ 155] (KL8xxx)
- Klemmen: K-Bus-Sicherheit [▶ 157] (KLx90x)
- Klemmen: K-Bus-System [▶ 156] (KL9xxx)

5.3.3.2 ITcSmTreeItem Element-Subtypen: E-Bus

Aufgrund ihrer Architektur werden E-Bus-**Boxen** , **-Klemmen** und **-Module** anders als ihre K-Bus-Gegenstücke behandelt, z.B. bei der Erstellung mit Hilfe der CreateChild() [▶ 160]-Methode. Während jede K-Bus-Klemme entsprechend ihrem spezifischen Subtyp spezifiziert wird, werden E-Bus-Klemmen über einen gemeinsamen Subtyp erkannt und dann über deren "**Produkt-Revision** " spezifiziert, die als vInfo-Parameter in der CreateChild()-Methode verwendet wird.

Subtyp	Beschreibung
9099	Allgemeiner Subtyp aller EtherCAT-Klemmen. Im Falle von CreateChild() [▶ 160] wird eine spezifische Klemme über den vInfo-Parameter definiert.

Es gibt einige wenige Ausnahmen zu dieser Regel, z.B. für RS232-Klemmen. Die folgende Tabelle gibt einen Überblick über diese Ausnahmen:

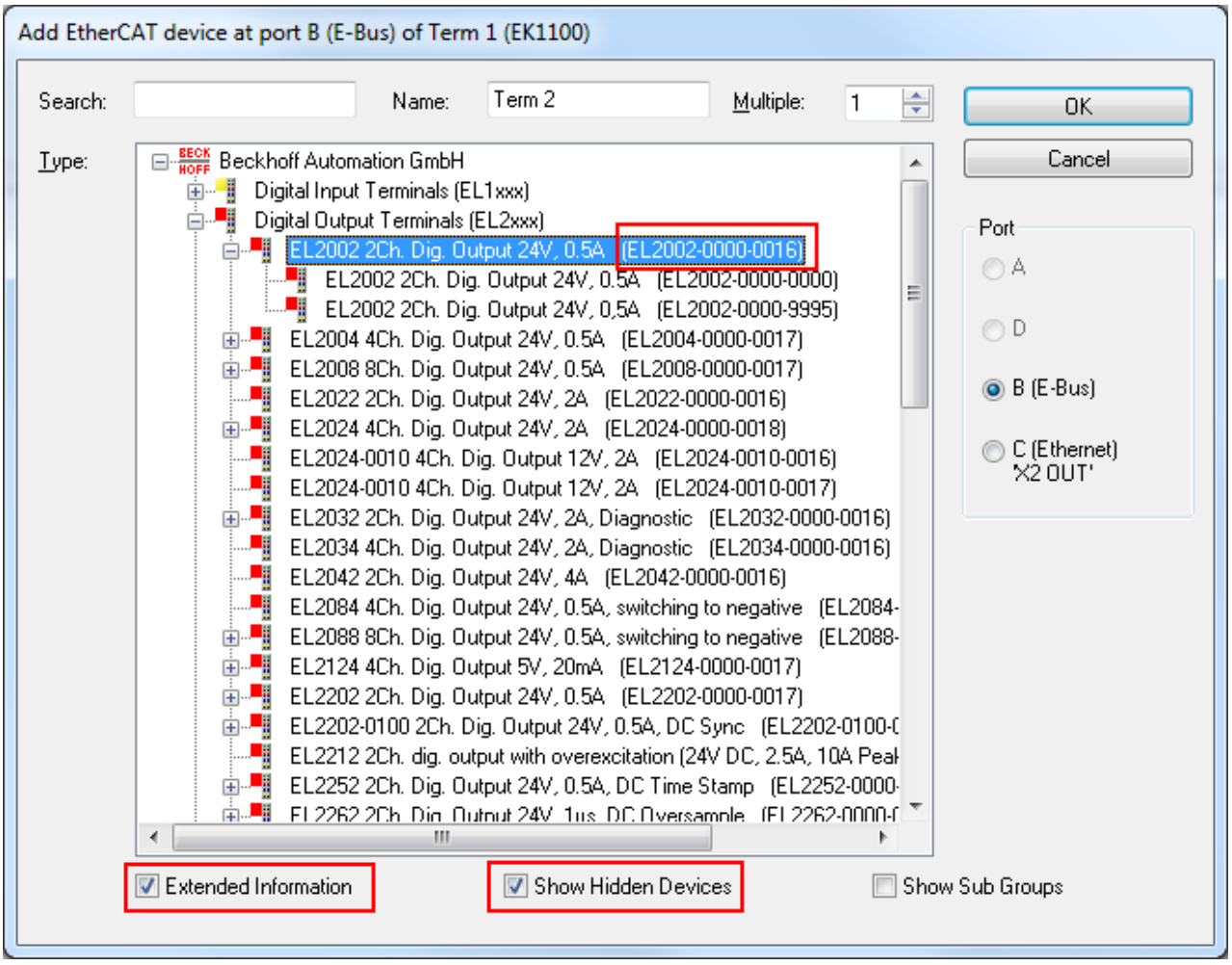
I/O	ItemSubType	Beschreibung
EP6002	9101	RS232 / RS422 / RS485-Schnittstellenklemme
EL6001	9101	RS232-Schnittstellenklemme
EL6002	9101	RS232-Schnittstellenklemme (2-Kanal)
EL6021	9103	RS422 / RS485-Schnittstellenklemme
EL6022	9103	RS422 / RS485-Schnittstellenklemme (2-Kanal)

Code-Ausschnitt (C#)

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^EtherCAT Master^EK1100");
ek1100.CreateChild("EL2002 - 1", 9099, "", "EL2002-0000-0016");
```

Produkt-Revision

Jede(s) E-Bus Box/Klemme/Modul hat ihre (seine) eigene Produkt-Revision, die Sie entweder über den Export der XML-Beschreibung [▶ 25] oder im Dialog "Gerät hinzufügen" in TwinCAT XAE einsehen können.



Zum Beispiel hat die EL2002-Digitalausgangsklemme die Produkt-Revision EL2002-0000-0016, was Sie auch anhand ihrer XML-Beschreibung feststellen können:

```
<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>
```

Um die XML-Beschreibung eines Tree Items zu erhalten, gehen Sie folgendermaßen vor:

- **TwinCAT 2:** Fügen Sie das Element zum System Manager hinzu, wählen es aus und wählen anschließend den Menüpunkt “Aktionen”--> "XML-Beschreibung exportieren"
- **TwinCAT 3:** Fügen Sie das Element zu XAE hinzu, wählen es aus und wählen anschließend den Menüpunkt "TwinCAT" --> "Ausgewähltes Element " --> "XML-Beschreibung exportieren"

5.3.3.3 Geräte

5.3.3.3.1 ITcSmTreeItem Element-Subtypen: Geräte

Geräte: Verschiedene

Subtyp	Tag	Beschreibung
0	IODEVICETYPE_UNKNOWN	---
6	IODEVICETYPE_BKHFPC	Beckhoff Industrie-PC C2001
9	IODEVICETYPE_LPTPORT	LPT-Port
10	IODEVICETYPE_DPRAM	Allgemeine DPRAM
11	IODEVICETYPE_COMPORT	COM-Port
18	IODEVICETYPE_FCXXXX	Beckhoff-Feldbuskarte
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 oder 816 Chip Master oder Slave (PCI)
53	IODEVICETYPE_BKHFAH2000	Beckhoff AH2000 (Hydraulik Backplane)
55	IODEVICETYPE_AH2000MC	Beckhoff-AH2000 mit Profibus-MC

Geräte: Beckhoff CX-Geräte

Subtyp	Tag	Beschreibung
120	IODEVICETYPE_CX5000	CX5000
135	IODEVICETYPE_CX8000	CX8000
105	IODEVICETYPE_CX9000_BK	CX9000
65	IODEVICETYPE_CX1100_BK	CX1100
124	IODEVICETYPE_CCAT	Beckhoff CCAT-Adapter

Geräte: Beckhoff CP-Geräte

Subtyp	Tag	Beschreibung
14	IODEVICETYPE_BKHFCP2030	Beckhoff CP2030 (Panel-Link)
31	IODEVICETYPE_BKHFCP9030	Beckhoff CP9030 (Panel-Link mit USV, ISA)
52	IODEVICETYPE_BKHFCP9040	Beckhoff CP9040 (CP-PC)
54	IODEVICETYPE_BKHFCP9035	Beckhoff CP9035 (Panel-Link mit USV, PCI)
116	IODEVICETYPE_BKHFCP6608	Beckhoff CP6608(IXP-PC)

Geräte: Beckhoff BC/BX-Geräte

Subtyp	Tag	Beschreibung
77	IODEVICETYPE_BX_BK	BX Klemmenbus Interface
78	IODEVICETYPE_BX_M510	BX SSB-Master
103	IODEVICETYPE_BC8150	BCXX50 serieller Slave
104	IODEVICETYPE_BX9000	BX9000 Ethernet-Slave
107	IODEVICETYPE_BC9050	BC9050 Ethernet-Slave
108	IODEVICETYPE_BC9120	BC9120 Ethernet-Slave
110	IODEVICETYPE_BC9020	BC9020 Ethernet-Slave

Geräte: Beckhoff EtherCAT

Subtyp	Tag	Beschreibung
94	IODEVICETYPE_ETHERCAT	Veraltet: EtherCAT-Master. Verwenden Sie stattdessen „;111“;.
111	IODEVICETYPE_ETHERCATPROT	EtherCAT-Master
130	IODEVICETYPE_ETHERCATSLV	EtherCAT-Slave
106	IODEVICETYPE_EL6601	EtherCAT-Automatisierungsprotokoll über EL6601
112	IODEVICETYPE_ETHERNETNVPROT	EtherCAT-Automatisierungsprotokoll (Netzwerkvariablen)
144	IODEVICETYPE_ETHERCATSIMULATION	EtherCAT-Simulation

Geräte: Beckhoff Lightbus Master/Slave

Subtyp	Tag	Beschreibung
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-Schnittstellenkarte C1220
2	IODEVICETYPE_C1200	Beckhoff Lightbus ISA-Schnittstellenkarte C1200

Geräte: Beckhoff Profibus Master/Slave

Subtyp	Tag	Beschreibung
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-Karte
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

Geräte: Beckhoff CANopen Master/Slave

Subtyp	Tag	Beschreibung
71	IODEVICETYPE_CX1500_M510	PC104 CANopen-Master
72	IODEVICETYPE_CX1500_B510	PC104 CANopen-Slave
39	IODEVICETYPE_FC5100	Beckhoff-CanOpen-PCI-Karte
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

Geräte: Beckhoff DeviceNet Master/Slave

Subtyp	Tag	Beschreibung
73	IODEVICETYPE_CX1500_M520	PC104 DeviceNet-Master
74	IODEVICETYPE_CX1500_B520	PC104 DeviceNet-Slave
41	IODEVICETYPE_FC5200	Beckhoff-DeviceNet-PCI-Karte
59	IODEVICETYPE_FC5200MON	Beckhoff-DeviceNet-Monitor-PCI-Karte
62	IODEVICETYPE_FC5200SLV	Beckhoff-DeviceNet-PCI-Karte als Slave
82	IODEVICETYPE_BX_B520	BX DeviceNet-Slave
85	IODEVICETYPE_BC5250	BCxx50 DeviceNet-Slave
88	IODEVICETYPE_EL6752	Beckhoff-DeviceNet-EtherCAT-Klemme
99	IODEVICETYPE_EL6752SLV	Beckhoff-DeviceNet-Slave-EtherCAT-Klemme

Geräte: Beckhoff Sercos Master/Slave

Subtyp	Tag	Beschreibung
75	IODEVICETYPE_CX1500_M750	PC104 Sercos-Master
76	IODEVICETYPE_CX1500_B750	PC104 Sercos-Slave
48	IODEVICETYPE_FC7500	Beckhoff-SERCOS-PCI-Karte

Geräte: Ethernet

Subtyp	Tag	Beschreibung
66	IODEVICETYPE_ENETRTMP	Ethernet Echtzeit-Miniport
109	IODEVICETYPE_ENETADAPTER	Real-Time-Ethernet-Adapter (Multiple Protocol Handler)
138	---	Echtzeit-Ethernet-Protokoll (BK90xx, AX2000-B900)
45	IODEVICETYPE_ETHERNET	Virtuelle Ethernet-Schnittstelle

Geräte: USB

Subtyp	Tag	Beschreibung
57	IODEVICETYPE_USB	Virtuelle USB-Schnittstelle
125	---	Virtuelle USB-Schnittstelle (Remote)

Geräte: Hilscher

Subtyp	Tag	Beschreibung
4	IODEVICETYPE_CIF30DPM	ISA ProfiBus-Master 2 kByte (Hilscher-Karte)
5	IODEVICETYPE_CIF40IBSM	ISA Interbus-S-Master 2 kByte (Hilscher-Karte)
12	IODEVICETYPE_CIF30CAN	ISA CANopen-Master (Hilscher-Karte)
13	IODEVICETYPE_CIF30PB	ISA ProfiBus-Master 8 kByte (Hilscher-Karte)
16	IODEVICETYPE_CIF30IBM	ISA Interbus-S-Master (Hilscher-Karte)
17	IODEVICETYPE_CIF30DNM	ISA DeviceNet-Master (Hilscher-Karte)
19	IODEVICETYPE_CIF50PB	PCI ProfiBus-Master 8 kByte (Hilscher-Karte)
20	IODEVICETYPE_CIF50IBM	PCI Interbus-S-Master (Hilscher-Karte)
21	IODEVICETYPE_CIF50DNM	PCI DeviceNet-Master (Hilscher-Karte)
22	IODEVICETYPE_CIF50CAN	PCI CANopen-Master (Hilscher-Karte)
23	IODEVICETYPE_CIF60PB	PCMCIA ProfiBus-Master (Hilscher-Karte)
24	IODEVICETYPE_CIF60DNM	PCMCIA DeviceNet-Master (Hilscher-Karte)
25	IODEVICETYPE_CIF60CAN	PCMCIA CANopen-Master (Hilscher-Karte)
26	IODEVICETYPE_CIF104DP	PC104 ProfiBus-Master 2 kByte (Hilscher-Karte)
27	IODEVICETYPE_C104PB	PC104 ProfiBus-Master 8 kByte (Hilscher-Karte)
28	IODEVICETYPE_C104IBM	PC104 Interbus-S-Master 2 kByte (Hilscher-Karte)
29	IODEVICETYPE_C104CAN	PC104 CANopen-Master (Hilscher-Karte)
30	IODEVICETYPE_C104DNM	PC104 DeviceNet-Master (Hilscher-Karte)
35	IODEVICETYPE_CIF60IBM	PCMCIA Interbus-S-Master (Hilscher-Karte)
49	IODEVICETYPE_CIF30IBS	ISA Interbus-S-Slave (Hilscher-Karte)
50	IODEVICETYPE_CIF50IBS	PCI Interbus-S-Slave (Hilscher-Karte)
51	IODEVICETYPE_C104IBS	PC104 Interbus-S-Slave (Hilscher-Karte)
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)

Subtyp	Tag	Beschreibung
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)

Geräte: Profinet / Profibus

Subtyp	Tag	Beschreibung
3	IODEVICETYPE_SPC3	ProfiBus Slave (Siemens-Karte)
7	IODEVICETYPE_CP5412A2	ProfiBus-Master (Siemens-Karte)
34	IODEVICETYPE_CP5613	PCI ProfiBus-Master (Siemens-Karte)
113	IODEVICETYPE_PROFINETIOCONTROLLER	PROFINET Master
115	IODEVICETYPE_PROFINETIODEVICE	PROFINET Slave

Geräte: Indramat

Subtyp	Tag	Beschreibung
8	IODEVICETYPE_SERCANSISA	Sercos Master (Indramat)

Geräte: Phoenix

Subtyp	Tag	Beschreibung
15	IODEVICETYPE_IBSSCIT	Interbus-S-Master (Phoenix-Karte)
47	IODEVICETYPE_IBSSCRIRTLK	Interbus-S-Master mit Slave-Teil auf LWL-Basis (Phoenix-Karte)
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 Kupferbasis (Phoenix-Karte)

5.3.3.4 Boxen

5.3.3.4.1 ITcSmTreeItem Element-Subtypen: Boxen

Subtyp	Tag	Beschreibung
0	BOXTYPE_UNKNOWN	---
1	BOXTYPE_BK2000	BK2000 Lightbus-Koppler
2	BOXTYPE_M1400	M1400 Lightbus digitales Ein-/Ausgangsmodul
3	BOXTYPE_M2400	M2400 Lightbus Ein-/Ausgangsmodul
4	BOXTYPE_M3120_1	M3120 Lightbus Inkremental-Encoder-Interface
5	BOXTYPE_M3120_2	M3120 Lightbus Inkremental-Encoder-Interface
6	BOXTYPE_M3120_3	M3120 Lightbus Inkremental-Encoder-Interface
7	BOXTYPE_M3120_4	M3120 Lightbus Inkremental-Encoder-Interface
8	BOXTYPE_M3000	M3000 Absolut-/Inkrementalgeber
9	BOXTYPE_C1120	---
10	BOXTYPE_BK2010	BK2010 Lightbus-Koppler
11	BOXTYPE_AX2000	---
12	BOXTYPE_M2510	---
20	BOXTYPE_BK2020	BK2020 Lightbus-Koppler
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	---

Subtyp	Tag	Beschreibung
1030	BOXTYPE_IPXB3	---
1031	BOXTYPE_ILXB3	---
1032	BOXTYPE_ILXC3	---
1040	BOXTYPE_TSMBOX_310	---
1041	BOXTYPE_BX3100	---
1042	BOXTYPE_CX1500_B310	---
1043	BOXTYPE_FC310X_SLAVE	---
1044	BOXTYPE_EL6731_SLAVE	---
1051	BOXTYPE_AX2000_B310	---
1100	BOXTYPE_TCPBDPSLAVE	---
1101	BOXTYPE_TCFDLAGAG	---
1102	BOXTYPE_TCMPI	---
1103	BOXTYPE_TCPBMCSLAVE	---
1104	BOXTYPE_TCPBMCSLAVE2	---
1105	BOXTYPE_TCPBMCSLAVE3	---
1106	BOXTYPE_TCPBMCSLAVE4	---
1107	BOXTYPE_TCPBMCSLAVE5	---
1108	BOXTYPE_TCPBMCSLAVE6	---
1109	BOXTYPE_TCPBMCSLAVE7	---
1110	BOXTYPE_TCPBMCSLAVE8	---
1111	BOXTYPE_TCPBMONSLAVE	---
2001	BOXTYPE_BK4000	---
2002	BOXTYPE_IBS_GENERIC	---
2003	BOXTYPE_IBS_BK	---
2004	BOXTYPE_BK4010	---
2005	BOXTYPE_BK4500	---
2006	BOXTYPE_BK4510	---
2007	BOXTYPE_IBS_SLAVEBOX	---
2008	BOXTYPE_BC4000	---
2009	BOXTYPE_BK4020	---
2020	BOXTYPE_CP2020	---
2030	BOXTYPE_IPXB4	---
2031	BOXTYPE_ILXB4	---
2032	BOXTYPE_ILXC4	---
3001	BOXTYPE_SERCOSAXIS	---
3011	BOXTYPE_BK7500	---
3012	BOXTYPE_BK7510	---
3013	BOXTYPE_BK7520	---
3021	BOXTYPE_SERCOSMASTERBOX	---
3031	BOXTYPE_SERCOSSLAVEBOX	---
4001	BOXTYPE_BK8100	BK8100 Buskoppler
4002	BOXTYPE_BK8110	BK8110 Buskoppler
4003	BOXTYPE_BK8000	BK8000 Buskoppler
4004	BOXTYPE_BK8010	BK8010 Buskoppler
4005	BOXTYPE_CP9040	CP9040 PCB
4011	BOXTYPE_BC8000	BC8000 Busklemmen Controller
4012	BOXTYPE_BC8100	BC8100 Busklemmen Controller
4030	BOXTYPE_IPXB80	---

Subtyp	Tag	Beschreibung
4031	BOXTYPE_ILXB80	---
4032	BOXTYPE_ILXC80	---
4040	BOXTYPE_IPXB81	---
4041	BOXTYPE_ILXB81	---
4042	BOXTYPE_ILXC81	---
4050	BOXTYPE_BC8150	BC8150 Busklemmen Controller
5001	BOXTYPE_BK5100	BK5100 Buskoppler
5002	BOXTYPE_BK5110	BK5110 Buskoppler
5003	BOXTYPE_CANNODE	---
5004	BOXTYPE_BK5120	BK5120 Buskoppler
5005	BOXTYPE_LC5100	---
5006	BOXTYPE_CANDRIVE	---
5007	BOXTYPE_AX2000_B510	---
5008	BOXTYPE_BK5150	BK5150 Buskoppler
5009	BOXTYPE_BK5151	BK5151 Buskoppler
5011	BOXTYPE_BC5150	BC5150 Busklemmen Controller
5030	BOXTYPE_IPXB51	---
5031	BOXTYPE_ILXB51	---
5032	BOXTYPE_ILXC51	---
5040	BOXTYPE_TSMBOX_510	---
5041	BOXTYPE_BX5100	BX5100 CANopen Busklemmen Controller
5042	BOXTYPE_CX1500_B510	---
5043	BOXTYPE_FC510XSLV	---
5050	BOXTYPE_TCCANSLAVE	---
5051	BOXTYPE_CANQUEUE	CAN-Schnittstelle
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 Buskoppler
9002	BOXTYPE_BK9100	BK9100 Ethernet TCP/IP Buskoppler
9005	BOXTYPE_BK9050	BK9050 Ethernet TCP/IP Buskoppler
9011	BOXTYPE_BC9000	BC9000 Ethernet TCP/IP Busklemmen Controller
9012	BOXTYPE_BC9100	BC9100 Ethernet TCP/IP Busklemmen Controller
9013	BOXTYPE_BX9000	BX9000 Ethernet TCP/IP Busklemmen Controller
9014	BOXTYPE_BX9000SLV	---
9015	BOXTYPE_BC9050	BC9050 Ethernet TCP/IP Busklemmen Controller
9016	BOXTYPE_BC9050SLV	BC9050 Ethernet TCP/IP Busklemmen Controller Slave
9017	BOXTYPE_BC9120	BC9120 Ethernet TCP/IP Busklemmen Controller
9018	BOXTYPE_BC9120SLV	BC9120 Ethernet TCP/IP Busklemmen Controller Slave
9019	BOXTYPE_BC9020	BC9020 Ethernet TCP/IP Busklemmen Controller

Subtyp	Tag	Beschreibung
9020	BOXTYPE_BC9020SLV	BC9020 Ethernet TCP/IP Busklemmen Controller Slave
9030	BOXTYPE_IPXB9	---
9031	BOXTYPE_ILXB9	---
9032	BOXTYPE_ILXC9	---
9041	BOXTYPE_REMOTETASK	---
9051	BOXTYPE_NV_PUB	Netzwerk-Publisher.
9052	BOXTYPE_NV_SUB	Netzwerk-Subscriber.
9053	BOXTYPE_NV_PUBVAR	Publisher-Variable.
9054	BOXTYPE_NV_SUBVAR	Subscriber-Variable.
9055	BOXTYPE_NV_PUBDATA	Publisher-Datenobjekt.
9056	BOXTYPE_NV_SUBDATA	Subscriber-Datenobjekt.
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 Geräteschnittstelle
9124	BOXTYPE_PNIO_DRIVE	---
9125	BOXTYPE_PNIOBK9103	---
9126	BOXTYPE_PNIOILB903	---
9132	BOXTYPE_BK9053	BK9053 K-Bus-Koppler, 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	---

5.3.3.5 Klemmen

5.3.3.5.1 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Digitaleingang (KL1)

Subtyp	Beschreibung
1002	KL1002 2-Kanal-Digitaleingangsklemme
1012	KL1012 2-Kanal-Digitaleingangsklemme
1032	KL1032 2-Kanal-Digitaleingangsklemme
1052	KL1052 2-Kanal-Digitaleingangsklemme
1104	KL1104 4-Kanal-Digitaleingangsklemme
1114	KL1114 4-Kanal-Digitaleingangsklemme
1124	KL1124 4-Kanal-Digitaleingangsklemme
1154	KL1154 4-Kanal-Digitaleingangsklemme
1164	KL1164 4-Kanal-Digitaleingangsklemme
1184	KL1184 4-Kanal-Digitaleingangsklemme
1194	KL1194 4-Kanal-Digitaleingangsklemme
1212	KL1212 2-Kanal-Digitaleingangsklemme
1232	KL1232 2-Kanal-Digitaleingangsklemme
1302	KL1302 2-Kanal-Digitaleingangsklemme
1304	KL1304 4-Kanal-Digitaleingangsklemme
1312	KL1312 2-Kanal-Digitaleingangsklemme
1314	KL1314 4-Kanal-Digitaleingangsklemme
1352	KL1352 2-Kanal-Digitaleingangsklemme
1362	KL1362 2-Kanal-Digitaleingangsklemme
1382	KL1382 2-Kanal-Digitaleingangsklemme
1402	KL1402 2-Kanal-Digitaleingangsklemme
1404	KL1404 4-Kanal-Digitaleingangsklemme
1408	KL1408 8-Kanal-Digitaleingangsklemme
1412	KL1412 2-Kanal-Digitaleingangsklemme
1414	KL1414 4-Kanal-Digitaleingangsklemme
1418	KL1418 8-Kanal-Digitaleingangsklemme
1434	KL1434 4-Kanal-Digitaleingangsklemme
1488	KL1488 8-Kanal-Digitaleingangsklemme
1498	KL1498 8-Kanal-Digitaleingangsklemme
1501	KL1501 1-Kanal-Digitaleingangsklemme
1512	KL1512 2-Kanal-Digitaleingangsklemme
1702	KL1702 2-Kanal-Digitaleingangsklemme
1712	KL1712 2-Kanal-Digitaleingangsklemme
16778928	KL1712-0060 2-Kanal-Digitaleingangsklemme
1722	KL1722 2-Kanal-Digitaleingangsklemme
1804	KL1804 4-Kanal-Digitaleingangsklemme
1808	KL1808 8-Kanal-Digitaleingangsklemme
1809	KL1809 16-Kanal-Digitaleingangsklemme
1814	KL1814 4-Kanal-Digitaleingangsklemme
1819	KL1819 16-Kanal-Digitaleingangsklemme
1859	KL1859 8-Kanal-Digitaleingangsklemme
1862	KL1862 16-Kanal-Digitaleingangsklemme
16779078	KL1862-0010 16-Kanal-Digitaleingangsklemme
1872	KL1872 16-Kanal-Digitaleingangsklemme
1889	KL1889 16-Kanal-Digitaleingangsklemme
1202	KL1202 2-Kanal-Digitaleingangsklemme

5.3.3.5.2 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Digitalausgang (KL2)

Subtyp	Beschreibung
2408	KL2408 8-Kanal-Digitalausgangsklemme
2012	KL2012 2-Kanal-Digitalausgangsklemme
2022	KL2022 2-Kanal-Digitalausgangsklemme
2032	KL2032 2-Kanal-Digitalausgangsklemme
2114	KL2114 4-Kanal-Digitalausgangsklemme
2124	KL2124 4-Kanal-Digitalausgangsklemme
2134	KL2134 4-Kanal-Digitalausgangsklemme
2184	KL2184 4-Kanal-Digitalausgangsklemme
2212	KL2212 2-Kanal-Digitalausgangsklemme
2404	KL2404 4-Kanal-Digitalausgangsklemme
2212	KL2212 2-Kanal-Digitalausgangsklemme
2404	KL2404 4-Kanal-Digitalausgangsklemme
2408	KL2408 8-Kanal-Digitalausgangsklemme
2284	KL2284 4-Kanal-Digitalausgangsklemme
2424	KL2424 4-Kanal-Digitalausgangsklemme
2442	KL2442 2-Kanal-Digitalausgangsklemme
2488	KL2488 8-Kanal-Digitalausgangsklemme
2502	KL2502 2-Kanal-PWM-Ausgansgklemme
2512	KL2512 2-Kanal-PWM-Ausgangsklemme
2521	KL2521 1-Kanal-Pulse-Train-Ausgangsklemme
2531	KL2531 1-Kanal-Schrittmotorklemme
16779747	KL2531-1000 1-Kanal-Schrittmotorklemme
2532	KL2532 2-Kanal-DC-Motor-Verstärkerausgangsklemme
2535	KL2535 2-Kanal-PWM-Verstärkerausgangklemme
2541	KL2541 1-Kanal-Schrittmotorklemme
16779757	KL2541-1000 1-Kanal-Schrittmotorklemme
2542	KL2542 2-Kanal-DC-Motor-Verstärkerklemme
2545	KL2545 2-Kanal-PWM-Verstärkerausgangklemme
2552	KL2552 2-Kanal-DC-Motor-Verstärkerklemme
2602	KL2602 2-Kanal-Ausgangsrelaisklemme
2612	KL2612 2-Kanal-Ausgangsrelaisklemme
2622	KL2622 2-Kanal-Ausgangsrelaisklemme
2631	KL2631 1-Kanal-Leistungsausgangsrelaisklemme
2641	KL2641 1-Kanal-Leistungsausgangsrelaisklemme
2652	KL2652 2-Kanal-Leistungsausgangsrelaisklemme
2701	KL2701 1-Kanal-Solid-State-Lastrelaisklemme
2702	KL2702 2-Kanal-Solid-State-Ausgangsrelaisklemme
16779918	KL2702-0002 2-Kanal-Solid-State-Ausgangsrelaisklemme
33557134	KL2702-0020 2-Kanal-Solid-State-Ausgangsrelaisklemme
2712	KL2712 2-Kanal-Triac-Ausgangsklemme
2722	KL2722 2-Kanal-Triac-Ausgangsklemme
2732	KL2732 2-Kanal-Triac-Ausgangsklemme
2744	KL2744 4-Kanal-Solid-State-Ausgangsrelais
2751	KL2751 1-Kanal-Universal-Dimmerklemme
33557183	KL2751-1200 1-Kanal-Universal-Dimmerklemme
2761	KL2761 1-Kanal-Universal-Dimmerklemme
2784	KL2784 4-Kanal-Ausgangsklemme
2791	KL2791 1-Kanal-Drehzahlstellerklemme

Subtyp	Beschreibung
33557223	KL2791-1200 1-Kanal-Drehzahlstellerklemme
2794	KL2794 4-Kanal-Ausgangsklemme
2808	KL2808 8-Kanal-Ausgangsklemme
2809	KL2809 16-Kanal-Ausgangsklemme
2872	KL2872 16-Kanal-Ausgangsklemme
2889	KL2889 16-Kanal-Ausgangsklemme

5.3.3.5.3 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Analogeingang (KL3)

Subtyp	Beschreibung
3001	KL3001 1-Kanal-Analogeingangsklemme
3002	KL3002 3-Kanal-Analogeingangsklemme
3011	KL3011 1-Kanal-Analogeingangsklemme
3012	KL3012 2-Kanal-Analogeingangsklemme
3021	KL3021 1-Kanal-Analogeingangsklemme
3022	KL3022 2-Kanal-Analogeingangsklemme
3041	KL3041 1-Kanal-Analogeingangsklemme
3042	KL3042 2-Kanal-Analogeingangsklemme
3044	KL3044 4-Kanal-Analogeingangsklemme
3051	KL3051 1-Kanal-Analogeingangsklemme
3052	KL3052 2-Kanal-Analogeingangsklemme
3054	KL3054 4-Kanal-Analogeingangsklemme
3061	KL3061 1-Kanal-Analogeingangsklemme
3062	KL3062 2-Kanal-Analogeingangsklemme
3064	KL3064 4-Kanal-Analogeingangsklemme
3102	KL3102 2-Kanal-Analogeingangsklemme
3112	KL3112 2-Kanal-Analogeingangsklemme
3122	KL3122 2-Kanal-Analogeingangsklemme
3132	KL3132 2-Kanal-Analogeingangsklemme
3142	KL3142 2-Kanal-Analogeingangsklemme
3152	KL3152 2-Kanal-Analogeingangsklemme
3158	KL3158 8-Kanal-Analogeingangsklemme
3162	KL3162 2-Kanal-Analogeingangsklemme
3172	KL3172 2-Kanal-Analogeingangsklemme
33557604	KL3172-0500 2-Kanal-Analogeingangsklemme
67112036	KL3172-1000 2-Kanal-Analogeingangsklemme
3182	KL3182 2-Kanal-Analogeingangsklemme
3201	KL3201 1-Kanal-Analogeingangsklemme
3202	KL3202 2-Kanal-Analogeingangsklemme
3204	KL3204 4-Kanal-Analogeingangsklemme
33557640	KL3208-0010 8-Kanal-Analogeingangsklemme
3222	KL3222 2-Kanal-Analogeingangsklemme
3228	KL3228 8-Kanal-Analogeingangsklemme
3302	KL3302 2-Kanal-Analogeingangsklemme
3311	KL3311 1-Kanal-Analogeingangsklemme
3312	KL3312 2-Kanal-Analogeingangsklemme
3314	KL3314 4-Kanal-Analogeingangsklemme
3351	KL3351 1-Kanal-Widerstandsbrückenklemme
50334999	KL3351-0001 1-Kanal-Widerstandsbrückenklemme
3356	KL3356 Präzise 1-Kanal-Widerstandsbrückenklemme
3361	KL3361 1-Kanal-Oszilloskopklemme
3362	KL3362 2-Kanal-Oszilloskopklemme
3403	KL3403 3-Phasen-Leistungsmessklemme
3404	KL3404 4-Kanal-Analogeingangsklemme
3408	KL3408 8-Kanal-Analogeingangsklemme
3444	KL3444 4-Kanal-Analogeingangsklemme
3448	KL3448 8-Kanal-Analogeingangsklemme
3454	KL3454 4-Kanal-Analogeingangsklemme

Subtyp	Beschreibung
3458	KL3458 8-Kanal-Analogeingangsklemme
3464	KL3464 4-Kanal-Analogeingangsklemme
3468	KL3468 8-Kanal-Analogeingangsklemme

5.3.3.5.4 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Analogausgang (KL4)

Subtyp	Beschreibung
4001	KL4001 1-Kanal-Analogausgangsklemme
4002	KL4002 2-Kanal-Analogausgangsklemme
4004	KL4004 4-Kanal-Analogausgangsklemme
4011	KL4011 1-Kanal-Analogausgangsklemme
4012	KL4012 2-Kanal-Analogausgangsklemme
4021	KL4021 1-Kanal-Analogausgangsklemme
4022	KL4022 2-Kanal-Analogausgangsklemme
4031	KL4031 1-Kanal-Analogausgangsklemme
4032	KL4032 2-Kanal-Analogausgangsklemme
4034	KL4034 4-Kanal-Analogausgangsklemme
4112	KL4112 2-Kanal-Analogausgangsklemme
4122	KL4122 2-Kanal-Analogausgangsklemme
4132	KL4132 2-Kanal-Analogausgangsklemme
4404	KL4404 4-Kanal-Analogausgangsklemme
4408	KL4408 8-Kanal-Analogausgangsklemme
4414	KL4414 4-Kanal-Analogausgangsklemme
4418	KL4418 8-Kanal-Analogausgangsklemme
4424	KL4424 4-Kanal-Analogausgangsklemme
4428	KL4428 8-Kanal-Analogausgangsklemme
4434	KL4434 4-Kanal-Analogausgangsklemme
4438	KL4438 8-Kanal-Analogausgangsklemme
4494	KL4494 2-Kanal-Analogausgangsklemme

5.3.3.5.5 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Messung (KL5)

Subtyp	Beschreibung
5001	KL5001 1-Kanal-SSI-Encoderklemme
5051	KL5051 1-Kanal-Bi-SSI-Encoderklemme
16782267	KL5051-0010 1-Kanal-Bi-SSI-Encoderklemme
5101	KL5101 5V-Inkremental-Encoder-Klemme
5111	KL5111 24V-Inkremental-Encoder-Klemme
5121	KL5121 4-Kanal-Streckensteuerungsklemme
5151	KL5151 1-Kanal-Inkremental-Encoder-Klemme
33559583	KL5151-0021 1-Kanal-Inkremental-Encoder-Klemme
16782367	KL5151-0050 2-Kanal-Inkremental-Encoder-Klemme
5152	KL5152 2-Kanal-Inkremental-Encoder-Klemme

5.3.3.5.6 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Kommunikation (KL6)

Subtyp	Beschreibung
16783217	KL6001-Schnittstelle RS232C-Klemme
50337649	KL6001-Schnittstelle RS232-Klemme
16783227	KL6011-Schnittstelle TTY-Klemme
50337659	KL6011-Schnittstelle TTY-Klemme
16783237	KL6021-Schnittstelle RS422/485-Klemme
50337669	KL6021-Schnittstelle RS485-Klemme
100669317	KL6021-Schnittstelle RS485-Klemme
100669327	KL6031-Schnittstelle RS232-Klemme
50337679	KL6031-Schnittstelle RS232-Klemme
100669337	KL6041-Schnittstelle RS485-Klemme
50337689	KL6041-Schnittstelle RS485-Klemme
16783257	KL6041-0100-Schnittstelle RS485-Klemme
6051	KL6051-Datenaustauschklemme
335550521	KL6201 ASI-Masterklemme
369104953	KL6201 ASI-Masterklemme
402659385	KL6201 ASI-Masterklemme
335550531	KL6211 ASI-Masterklemme
369104963	KL6211 ASI-Masterklemme
402659395	KL6211 ASI-Masterklemme
6224	KL6224 I/O-Link Masterklemme
16783440	KL6224 I/O-Link Masterklemme
33560656	KL6224 I/O-Link Masterklemme
50337872	KL6224 I/O-Link Masterklemme
33560733	KL6301 EIB-Klemme
33560833	KL6401 LON-Klemme
33561013	KL6581 EnOcean-Klemme
33561203	KL6771 MP-Bus Masterklemme
6781	KL6781 M-Bus-Schnittstellenklemme
6811	KL6811 DALI-Masterklemme
6821	KL6821 eDALI-Masterklemme

5.3.3.5.7 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus Power (KL8)

Subtyp	Beschreibung
33562433	KL8001 1-Kanal-Powerklemme
8001	KL8001 3-Kanal-Powerklemme
8519	KL8519 16 Digitaleingangsklemme
8524	KL8524 2x4 Digitalausgangsklemme
8528	KL8528 8 Digitalausgangsklemme
8548	KL8548 8-Kanal-Analogausgangsklemme
8610	KL8610 1-Kanal-Adapterklemme KL8601
16785826	KL8610 2-Kanal-Adapterklemme KL8601
33563042	KL8610 3-Kanal-Adapterklemme KL8601
50340258	KL8610 4-Kanal-Adapterklemme KL8601
67117474	KL8610 5-Kanal-Adapterklemme KL8601
83894690	KL8610 6-Kanal-Adapterklemme KL8601
100671906	KL8610 7-Kanal-Adapterklemme KL8601
117449122	KL8610 8-Kanal-Adapterklemme KL8601

5.3.3.5.8 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-System (KL9)

Subtyp	Beschreibung
9010	KL9010 Endklemme
9020	KL9020 Busverlängerungs-Endklemme
9050	KL9050 Busverlängerungs-Kopplerklemme
9060	KL9060 Adapterklemme
9070	KL9070 Schirmklemme
9080	KL9080 Trennklemme
9100	KL9100 Netzteilklemme
9110	KL9110 Netzteilklemme
9150	KL9150 Netzteilklemme
9160	KL9160 Netzteilklemme
9180	KL9180 Potenzialverteilklemme
9181	KL9181 Potenzialverteilklemme
9182	KL9182 Potenzialverteilklemme
9183	KL9183 Potenzialverteilklemme
9184	KL9184 Potenzialverteilklemme
9185	KL9185 Potenzialverteilklemme
9186	KL9186 Potenzialverteilklemme
9187	KL9187 Potenzialverteilklemme
9188	KL9188 Potenzialverteilklemme
9189	KL9189 Potenzialverteilklemme
9190	KL9190 Einspeiseklemme
9195	KL9195 Schirmklemme
9200	KL9200 Netzteilklemme
9210	KL9210 Netzteilklemme
9250	KL9250 Netzteilklemme
9260	KL9260 Netzteilklemme
9290	KL9290 Netzteilklemme
9300	KL9300 4-Kanal-Dioden-Array-Klemme
9301	KL9301 7-Kanal-Dioden-Array-Klemme
9302	KL9302 7-Kanal-Dioden-Array-Klemme
9309	KL9309-Schnittstellenklemme für KL85xx
9400	KL9400 K-Bus-Netzteilklemme
9505	KL9505 Netzteilklemme
167781665	KL9505-0010 Netzteilklemme
9508	KL9508 Netzteilklemme
167781668	KL9508-0010 Netzteilklemme
9510	KL9510 Netzteilklemme
167781670	KL9510-0010 Netzteilklemme
9512	KL9512 Netzteilklemme
167781672	KL9512-0010 Netzteilklemme
9515	KL9515 Netzteilklemme
167781675	KL9515-0010 Netzteilklemme
9528	KL9528 Netzteilklemme
9540	KL9540 Surgefilter-Feldversorgungsklemme
9550	KL9550 Surgefiltersystem- und Feldversorgungsklemme
9560	KL9560 Netzteilklemme
9570	KL9570 Puffer-Kondensator-Klemme

5.3.3.5.9 ITcSmTreeItem Element-Subtypen: Klemmen: K-Bus-Safety (KLx90x)

Subtyp	Beschreibung
1904	KL1904 4-Kanal-Safety-Eingangsklemme
1908	KL1908 8-Kanal-Safety-Eingangsklemme
2904	KL2904 4-Kanal-Safety-Ausgangsklemme
6904	KL6904 Safety-Logic-Klemme (7 TwinSAFE-Anschlüsse)
16784120	KL6904 Safety-Logic-Klemme (15 TwinSAFE-Anschlüsse)

5.3.3.6 Module

5.3.3.6.1 ITcSmTreeItem Element-Subtypen: Module: K-Bus-Digitaleingang (KM1)

Subtyp	Beschreibung
838861802	KM1002 16-Kanal-Digitaleingangsmodul
838861804	KM1004 32-Kanal-Digitaleingangsmodul
838861808	KM1008 64-Kanal-Digitaleingangsmodul
838861812	KM1012 16-Kanal-Digitaleingangsmodul
838861814	KM1014 32-Kanal-Digitaleingangsmodul
838861818	KM1018 64-Kanal-Digitaleingangsmodul
838862444	KM1644 4-Kanal-Digitaleingangsmodul

5.3.3.6.2 ITcSmTreeItem Element-Subtypen: Module: K-Bus-Digitalausgang (KM2)

Subtyp	Beschreibung
838862802	KM2002 16-Kanal-Digitalausgangsmodul
838862804	KM2004 32-Kanal-Digitalausgangsmodul
838862808	KM2008 64-Kanal-Digitalausgangsmodul
838862822	KM2022 16-Kanal-Digitalausgangsmodul
838862842	KM2042 16-Kanal-Digitalausgangsmodul
838863404	KM2604 4-Kanal-Digitalausgangsrelaismodul
838863414	KM2614 4-Kanal-Digitalausgangsrelaismodul
838863442	KM2642 2-Kanal-Digitalleistungsausgangsrelaismodul
838863452	KM2652 2-Kanal-Digitalleistungsausgangsrelaismodul
16779990	KM2774 4-Kanal-Jalousien-Steuerungsklemme
2774	KM2774-1001 4-Kanal-Jalousien-Steuerungsklemme

5.3.3.6.3 ITcSmTreeItem Element-Subtypen: Module: K-Bus-Analogeingang (KM3)

Subtyp	Beschreibung
838864501	KM3701 1-Kanal-Differenzdruckmessung
872418933	KM3701-0340 1-Kanal-Differenzdruckmessung
838864502	KM3702 2-Kanal-Absolutdruckmessung
838864512	KM3712 2-Kanal-Absolutdruckmessung

5.3.3.6.4 ITcSmTreeItem Element-Subtypen: Module: K-Bus-Analogausgang (KM4)

Subtyp	Beschreibung
838865402	KM4602 2-Kanal-Analogausgangsklemme

5.3.3.6.5 ITcSmTreeItem Element-Subtypen: Module: K-Bus-Kommunikation (KM6)

Subtyp	Beschreibung
6551	KM6551 IEEE802.15.4 Klemme
838867463	KM6663 Ethernet-Switchklemme

5.3.4 ITcSmTreeItem::ProduceXml

Die ProduceXml()-Methode gibt eine XML-Zeichenkette mit elementspezifischen Informationen und Parametern zurück.

```
HRESULT ProduceXml(VARIANT_BOOLbRecursive, BSTR* pXML);
```

Parameter

bRecursive	[in, defaultvalue(0)] Optionaler Parameter für späteren Gebrauch.
pXML	[in] enthält die XML-Darstellung der elementspezifischen Daten und Parameter.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_POINTER	pXML-Zeiger ist ungültig.

Kommentare

Die folgende XML-Zeichenkette ist ein unvollständiges Beispiel der resultierenden Informationen, wenn das Tree Item ein E/A-Gerät vom Typ *SERCOS Master/Slave FC750x* ist. Diese Zeichenkette kann als Eingabe für die IXMLDOMDocument::loadXML-Methode zwecks Erstellung eines XML DOM-Dokuments verwendet werden.

```
<?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>
```

Siehe auch

ITcSmTreeItem::ConsumeXML [▶ 159]

5.3.5 ITcSmTreeItem::ConsumeXml

Die ConsumeXml()-Methode verwendet ein BSTR, der eine XML-Darstellung mit elementspezifischen Daten und aktualisierten Parametern enthält. Diese Methode wird für die Änderung von Elementparametern verwendet, die nicht direkt über die ITcSmTreeItem-Schnittstelle zugänglich sind.

```
HRESULT ConsumeXml(BSTRbstrXML);
```

Parameter

bstrXML	[in] Zeichenkette mit der XML-Darstellung der elementspezifischen Parameter

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_FAIL	die *bstrXML* Zeichenkette enthält kein gültiges XML-Dokument.

Kommentare

Dieses Dokument kann nur die spezifischen Parameter enthalten, die zu verändern sind. Die Dokumentstruktur muss dem elementspezifischen XML-Baum entsprechen, die nicht zu verändernden Parameter können ausgelassen werden. Das folgende Dokument stellt ein kleinstmögliches Beispiel dar,

das für die Änderung des spezifischen Parameters *CheckNumberBoxes* des Elements verwendet werden kann (in diesem Falle eine C1220 Feldbuskarte). Wenn die im Dokument enthaltenen Parameter dem Element nicht bekannt sind, werden sie ignoriert.

```
<TreeItem><DeviceDef><DevC1220Def><CheckNumberBoxes>0</CheckNumberBoxes></
DevC1220Def></DeviceDef></TreeItem>
```

Der Parametersatz eines spezifischen Tree Items wird im XML-Schemadokument definiert, das mit TwinCAT geliefert wird. Der Parameter eines spezifischen Elements kann auch mittels Aufruf der ITcSmTreeItem::ProduceXML [▶ 158]-Methode ausgewertet werden. Die sich daraus ergebende XML-Zeichenkette enthält alle Parameter dieses Elements, aber nicht alle können verändert werden. Die XML-Zeichenkette kann eine beliebige Anzahl XML-Elemente, in beliebiger hierarchischer Ordnung, enthalten, die den Anforderungen des XML-Schemas entsprechen. Es ist zulässig, nur einen Parameter auf einmal zu ändern (wie im Beispiel oben), einen Parametersatz auf einmal zu ändern oder den vollständigen Paramtersatz zu liefern, den ITcSmTreeItem::ProduceXML [▶ 158] zurückgibt (normalerweise mit den gleichen geänderten Parametern).

Es gibt einige spezielle XML-Elemente, die keinem Parameter entsprechen, und die eine Funktion „;ausführen“;. Ein Beispiel ist das <Rescan>-Element eines SPS-Projekttree items. Die Zeichenkette:

```
<TreeItem><PlcDef><ReScan>1</ReScan></PlcDef></TreeItem>
```

als Parameter von **ConsumeXml** veranlasst den System Manager unter TwinCAT 2 das SPS-Projekt erneut zu scannen (wie ein manuell veranlasstes erneutes Scannen mittels Drücken der "Rescan" Taste auf einem SPS-Projekt). Die verfügbaren Parameter und Funktionen werden in der XML-Schemadatei dokumentiert.

Siehe auch

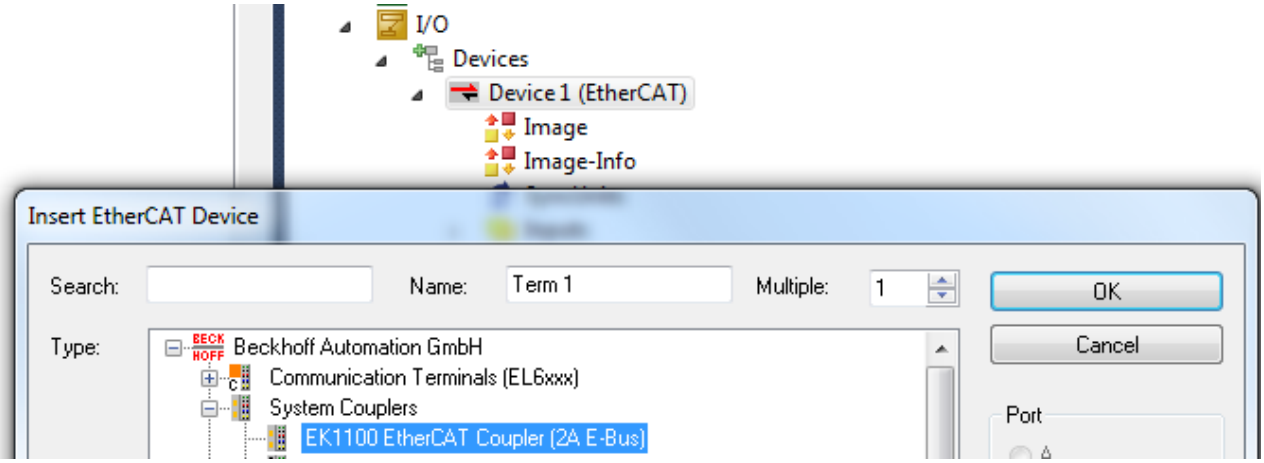
ITcSmTreeItem::ProduceXML [▶ 158]

5.3.6 ITcSmTreeItem::CreateChild

Erzeugt ein untergeordnetes Tree Item auf einem untergeordneten Knoten. Das untergeordnete Element wird dabei durch seinen Subtyp [▶ 131] spezifiziert.

```
HRESULT CreateChild(BSTRbstrName,long nSubType, BSTRbstrBefore, VARIANT vInfo,
ITcSmTreeItem**pipItem);
```

Das folgende Beispiel erläutert dieses Verhalten genauer.



In diesem Beispiel wird einem EtherCAT-Master-Gerät (Gerät 1) in TwinCAT XAE ein EK1100-Koppler hinzugefügt. Im TwinCAT Automation Interface könnte dies mit der CreateChild()-Methode durchgeführt werden. Der übergeordnete Knoten (Gerät 1 EtherCAT) ist vom Typ '2' (TREEITEMTYPE_DEVICE). Der Subtyp spezifiziert das untergeordnete Tree Item und so auch den EK1100, der vom Subtyp '9099' (BOXTYPE_EXXXXX) ist.

Parameter

bstrName	[in] Elementname des neuen untergeordneten Tree Item.
nSubtype:	[in] Subtyp [▶ 131] des neuen untergeordneten Tree Items. Der verwendbare Subtyp hängt ab vom Tree Item Type [▶ 128] (der Kategorie) des übergeordneten ´Tree Items . So kann z.B. ein SPS-Funktionsblock nur dem Elementtyp PLCFOLDER, und nicht einem DEVICE, hinzugefügt werden.
bstrBefore	[in, defaultvalue("")] Wenn gesetzt, enthält der Parameter den Namen eines anderen untergeordneten Tree Items, vor dem das neue Element einzufügen ist.
vInfo	[in, optional] Ein optionaler Parameter mit zusätzlichen Informationen für die Erstellung , welche abhängt vom Subtypen [▶ 131]. Die verschiedenen Abhängigkeiten werden in der Tabelle unten aufgelistet.
pipItem	[out, retval] Zeigt auf den Speicherort eines ITcSmTreeItem [▶ 127]-Schnittstellenzeigers, der das Ergebnis erhält.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_POINTER	die Adresse des Rückgabezeigers ist ungültig
TSM_E_INVALIDITEMSUBTYPE (0x98510003)	der angegebene Subtyp ist ungültig.
TSM_E_ITEMNOTFOUND (0x98510001)	Das Element *bstrBefore* wurde nicht gefunden.

Optionaler vInfo Parameter

Abhängig vom Element-Sub typ des neuen untergeordneten Tree Items, bestimmte Zusatzinformationen sind möglicherweise erforderlich für die Erstellung des untergeordneten Tree Items. Diese Information kann über den *vInfo*-Parameter geliefert werden.

Untergeordnetes Element: Element-Subtyp	vInfo-Parameter
E-Bus Box / Klemme / Modul (Element-Subtyp 9099)	Enthält das Identity-Objekt (CoE 1018h, VendorId, ProductCode und optional RevisionNo und SerialNo) der EtherCAT Box. Der Typ der Variante muss ein Array von LONG (VT_I4|VT_ARRAY, 2-4 Elemente) sein. Enthält alternativ einen BSTR im folgenden Format (wobei %X = Wert in hexadezimaler Notation, z.B. "V00000002_P044c2c52_R00000000"): Insbesondere für Beckhoff E-Bus Klemmen / Module bitte entsprechenden E-Bus Artikel [▶ 132] lesen.
Interbus-Box. (Element-Subtyp 2002)	Enthält den IdentCode und LengthCode der Interbus-Box. Der Typ der Variante muss ein vorzeichenloses Short (VT_I2) sein, wobei das Low-Byte den IdentCode und das High-Byte den LengthCode enthält.
AX2000 (Element-Subtyp 5007) CANDrive (Element-Subtyp 5006)	Enthält optional einen boolschen Wert (VT_BOOL) und wenn er gesetzt ist, wird eine zusätzliche Variable für den “Schleppfehler” erzeugt.
DeviceNET (Element-Subtyp 5203) TcDNSSlave (Element-Subtyp 5250) CX1500 (Element-Subtyp 1042) FC520X Slave (Element-Subtyp 1043)	Enthält optional den Dateipfad einer EDS-Datei (VT_BSTR).
BK3000 und alle übrigen PROFIBUS-Box-Typen	Enthält optional den Dateipfad einer GSD-Datei (VT_BSTR).
Variable	Enthält optional die Bit-Adresse der neuen Variablen. Der Typ der Variante kann ein SHORT oder ein LONG (VT_I2 oder VT_I4) sein.
SPS-POU Funktionsblock (Element-Subtyp 604)	Enthält IEC-Sprachentyp als Integer, wie durch IECLANGUAGETYPES [▶ 166] definiert.
SPS-POU Funktion (Element-Subtyp 603)	Enthält ein string[2]-Array, das folgende Werte hält: • array[0] = IEC-Sprachentyp als Integer, wie durch IECLANGUAGETYPES [▶ 166] definiert. • array[1] = Rückgabetyp der Funktion. Kann jeder beliebiger SPS-Datentyp sein, z.B. DINT, BOOL, ...

5.3.7 ITcSmTreeItem::DeleteChild

Löscht ein untergeordnetes Element.

```
HRESULT DeleteChild(BSTRbstrName);
```

Parameter

bstrName	[in] Elementname des zu löschenden untergeordneten Tree Items.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
E_ACCESSDENIED	es ist nicht erlaubt, das untergeordnete Tree Item zu löschen.
TSM_E_ITEMNOTFOUND (0x98510001)	Das Element *bstrBefore* wurde nicht gefunden.

5.3.8 ITcSmTreeItem::ImportChild

Importiert ein untergeordnetes Tree Item aus der Zwischenablage oder einer zuvor exportierten Datei.

```
HRESULT ImportChild(BSTRbstrFile, BSTRbstrBefore, VARIANT_BOOLbReconnect, BSTRbstrName,
ITcSmTreeItem**pipItem);
```

Parameter

bstrFile	[in, defaultvalue(L"")] Name der Datei, aus der das neue untergeordnete Element importiert wird. Wird kein Dateiname angegeben (leere Zeichenkette), dann wird das untergeordnete Element aus der Zwischenablage importiert.
bstrBefore	[in, defaultvalue(L"")] Wenn gesetzt, enthält der Parameter den Namen eines anderen untergeordneten Tree Items, vor dem das neue Element einzufügen ist. Wenn nicht gesetzt, dann wird das untergeordnete Element am Ende angefügt.
bReconnect	[in, defaultvalue(VARIANT_TRUE)] Ein optionales Flag, das den System Manager anweist, zu versuchen, die Variablen aus dem importierten Element mit den anderen Variablen in der Konfiguration (nach Namen) erneut zu verbinden.
bstrName	[in, defaultvalue(L"")] Wenn gesetzt, wird der Name des untergeordneten Elements mit seinem Namen in der Importdatei überschrieben.
pipItem	[out, retval] Zeigt auf den Speicherort eines ITcSmTreeItem [▶ 127]-Schnittstellenzeigers, der das Ergebnis erhält.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
NTE_NOT_FOUND (0x80090011)	die Datei kann nicht gefunden / geöffnet werden.
NTE_BAD_SIGNATURE (0x80090006)	die Datei enthält kein gültiges Tree item.
TSM_E_MISMATCHINGITEMS (0x98510004)	das in der Datei enthaltene Element ist kein gültiges untergeordnetes Tree Item.

5.3.9 ITcSmTreeItem::ExportChild

Exportiert ein untergeordnetes Tree Item in die Zwischenablage oder in eine Datei.

```
HRESULT ExportChild(BSTRbstrName, BSTRbstrFile);
```

Parameter

bstrName	[in] Name des zu exportierenden untergeordneten Elements.
bstrFile	[in, defaultvalue("")] Name der Datei, in die das untergeordnete Element exportiert wird. Wird kein Dateiname angegeben (leere Zeichenkette), dann wird das untergeordnete Element in die Zwischenablage exportiert.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
TSM_E_ITEMNOTFOUND (0x98510001)	Das Element *bstrName* wurde nicht gefunden.

5.3.10 ITcSmTreeItem::LookupChild

Gibt einen *ITcTreeItem*-Zeiger eines nachfolgenden untergeordneten Tree Items zurück, das mit relativem Pfadnamen bestimmt ist.

```
HRESULT LookupChild(BSTRbstrName, ITcSmTreeItem**pipItem);
```

Parameter

bstrName	[in] relativer Pfad des von Ihnen gesuchten Tree Items. Der relative Pfadname ist erforderlich und jeder Zweig muss durch ein Zirkumflex ‚;^' oder Tabulatorzeichen getrennt sein.
pipItem	[out, retval] zeigt auf den Speicherort eines ITcSmTreeItem [▶ 127]-Schnittstellenzeigers bei Rückgabe. Der Schnittstellenzeiger ermöglicht den Zugriff auf spezifische, zum Tree Item gehörige Methoden.

Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.
TSM_E_ITEMNOTFOUND (0x98510001)	der Pfadname verweist nicht auf ein bestehendes Tree Item.

5.3.11 ITcSmTreeItem::GetLastXmlError

Ruft den Elementpfad / die Fehlermeldung des letzten gescheiterten ConsumeXml [▶ 159]-Aufrufs ab.

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

Parameter

pXML	[out, retval] Fehlermeldung.

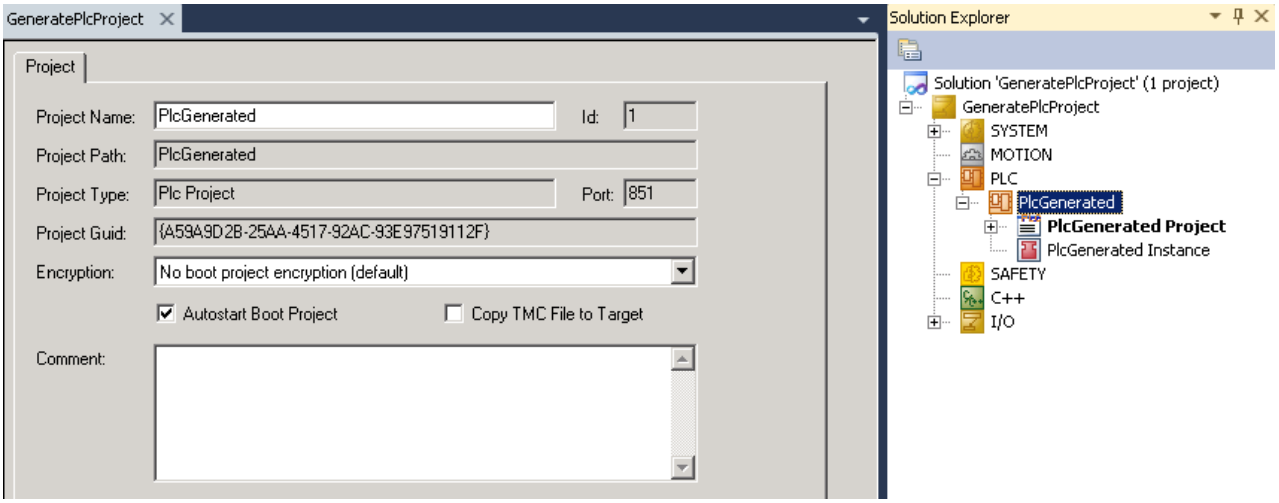
Rückgabewerte

S_OK	Funktion hat Wert erfolgreich zurückgegeben.

5.4 ITcPlcProject

5.4.1 ITcPlcProject

Entwickler können mit Hilfe der Klasse ITcPlcProject die Eigenschaften für ein SPS-Projekt festlegen. Sie zielt normalerweise auf den Stammknoten eines SPS-Projekts, wie in Abbildung unten gezeigt.



Der folgende C# Code-Ausschnitt zeigt ein Beispiel, wie diese Klasse innerhalb des Automation Interface-Codes verwendet werden kann:

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

Bitte beachten: Wenn Sie ein SPS-Projekt **kompilieren** möchten, dann verwenden Sie bitte die Kompilerfunktionalitäten vom Visual Studio COM-ObjektEnvDTE, wie in vielen unserer Beispiele [▶ 182].gezeigt .

Methoden

ITcPlcProject-Methoden	Beschreibung	Verfügbar seit
GenerateBootProject() [▶ 165]	Gleiche Wirkung wie „;Activate Boot Project“; (Bootprojekt aktivieren) Menüpunkt im TwinCAT XAE-Kontextmenü	TwinCAT 3.1

Eigenschaften

ITcPlcProject-Properties	Get / Set	Beschreibung	Verfügbar seit
BootProjectAutoStart	Ja / Ja	Gleiche Wirkung wie die Checkbox „;Autostart Boot Project“; (Bootprojekt automatisch starten) im oben gezeigten Dialog	TwinCAT 3.1
BootProjectEncryption	Ja / Ja	Gleiche Wirkung wie das Dropdown-Listenfeld „;Encryption“; (Verschlüsselung) im oben gezeigten Dialog	TwinCAT 3.1
TmcFileCopy	Ja / Ja	Gleiche Wirkung wie die Checkbox „;Copy TMC File to Target“; (TMC-Datei in Ziel kopieren) im oben gezeigten Dialog	TwinCAT 3.1

Versionsinformationen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.4.2 ITcPlcProject::GenerateBootProject

Aktiviert oder deaktiviert das SPS-Projekt als Bootprojekt, je nachdem, welchen bool-Parameter Sie angeben.

```
HRESULT GenerateBootProject(VARIANT_BOOL bActivate);
```

Parameter

bActivate	[in, optional, defaultvalue(-1)] Bestimmt, ob das Bootprojekt aktiviert werden soll

5.5 ITcPlcPou

5.5.1 ITcPlcPou

Für den Umgang mit POUs und deren Inhalt innerhalb eines TwinCAT 3 Projekts können die Schnittstellen ITcPlcPou, ITcPlcDeclaration [▶ 167] und ITcPlcImplementation [▶ 167] verwendet werden. Wenn Sie z.B. einen neuen Funktionsblock für ein SPS-Projekt erstellen und mit Code füllen möchten, können Sie diese Schnittstellen hierzu verwenden. Weitere Informationen finden Sie in unserem Best-Practice-Artikel „Umgang und Zugriff auf SPS-Objekten“;.

Eigenschaften

ITcPlcPou-Properties	Get / Set	Beschreibung	Verfügbar seit
DocumentXml	Ja / Ja	Abrufen / Setzen des SPS-Codes einer POU in XML Format (**nicht** PLCopen XML)	TwinCAT 3.1
ReturnType	Ja / Nein	Rückgabetyp der POU, z.B. der Rückgabetyp einer Funktion. Kann jeder beliebiger, der SPS bekannter Datentyp sein, z.B. BOOL, DINT, ... Der Rückgabetyp wird bei der Erstellung der POU über die CreateChild() [▶ 160]-Methode festgelegt.	TwinCAT 3.1

Versionsinformationen

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

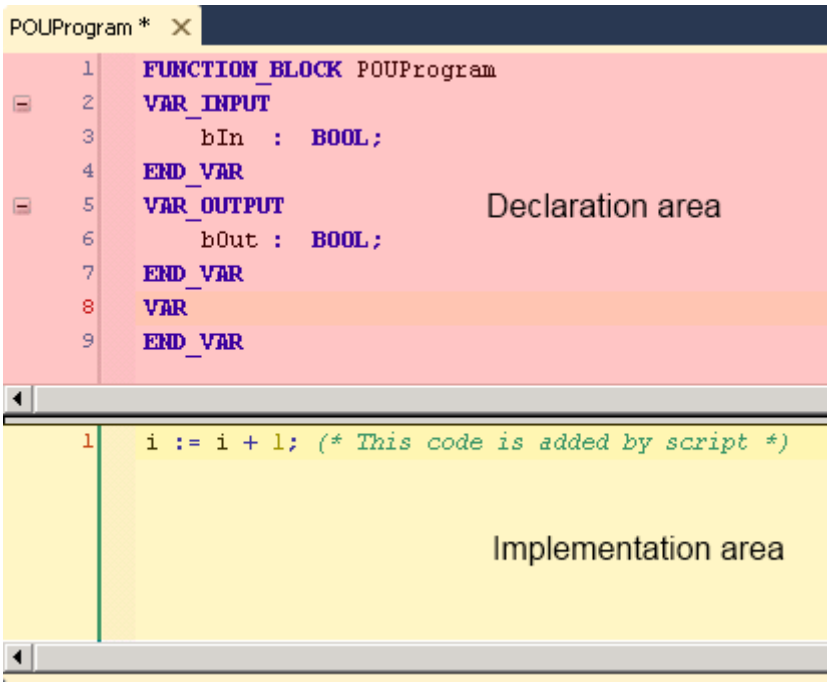
5.5.2 IECLanguageTypes

Das Enum IECLanguageTypes definiert verschiedene Programmiersprachen gemäß der IEC-Norm und kann bei der Erstellung einer neuen POU mit Hilfe der Methode ITcSmTreeItem [▶ 127]::CreateChild() [▶ 160] verwendet werden.

Eintrag	Wert	Beschreibung
IECLANGUAGE_NONE	0	---
IECLANGUAGE_ST	1	Strukturierter Text
IECLANGUAGE_IL	2	Anweisungsliste –; AWL
IECLANGUAGE_SFC	3	Ablaufsprache –; AS
IECLANGUAGE_FBD	4	Funktionsplan
IECLANGUAGE_CFC	5	Continuous Function Chart
IECLANGUAGE_LD	6	Kontaktplan

5.6 ITcPlcDeclaration

Die Schnittstelle ITcPlcDeclaration ermöglicht den Zugang zum Deklarationsbereich der SPS-POUs und deren Unterknoten (wie Aktionen, Methoden, ...). Siehe auch den "Best Practices"-Artikel „;Umgang mit SPS-Objekten“; für weitere Informationen über die Verwendung dieser Schnittstelle.



Eigenschaften (in Vtable-Reihenfolge)

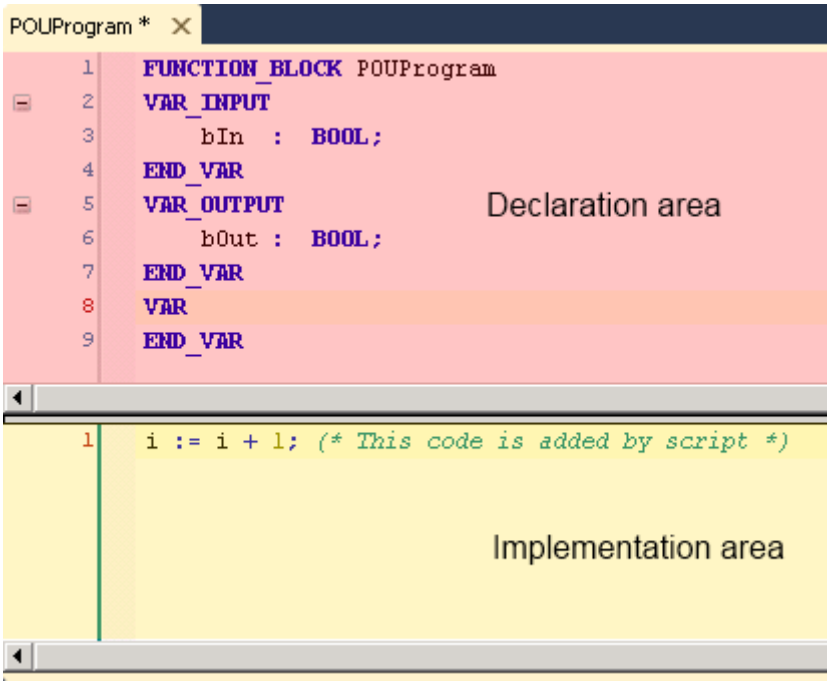
ITcPlcDeclaration Properties	Get / Set	Beschreibung	Verfügbar seit
DeclarationText	Ja / Ja	Stellt den Deklarationsbereich des Elements dar und setzt dessen Inhalt in Klartext oder ruft ihn ab.	TwinCAT 3.1

Versionsinformationen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.7 ITcPlcImplementation

Die Schnittstelle ITcPlcImplementation ermöglicht den Zugang zum Implementierungsbereich der SPS-POUs und deren Unterknoten (wie Aktionen, Methoden, ...). Siehe auch den Best-Practice-Artikel „;Umgang mit SPS-Objekten“; für weitere Informationen über die Verwendung dieser Schnittstelle.



Eigenschaften (in Vtable-Reihenfolge)

ITcPlcImplementation-Properties	Get / Set	Beschreibung	Verfügbar seit
ImplementationText	Ja / Ja	Stellt den Implementierungsbereich des Elements dar und setzt dessen Inhalt in Klartext oder ruft ihn ab.	TwinCAT 3.1
ImplementationXml	Ja / Ja	Abrufen / Setzen des Inhalts in XML-Format (**nicht** PLCopen XML)	TwinCAT 3.1
Language	Ja / Nein	Ruft die im Implementierungsbereich verwendete IEC-Sprache ab	TwinCAT 3.1

Versionsinformationen

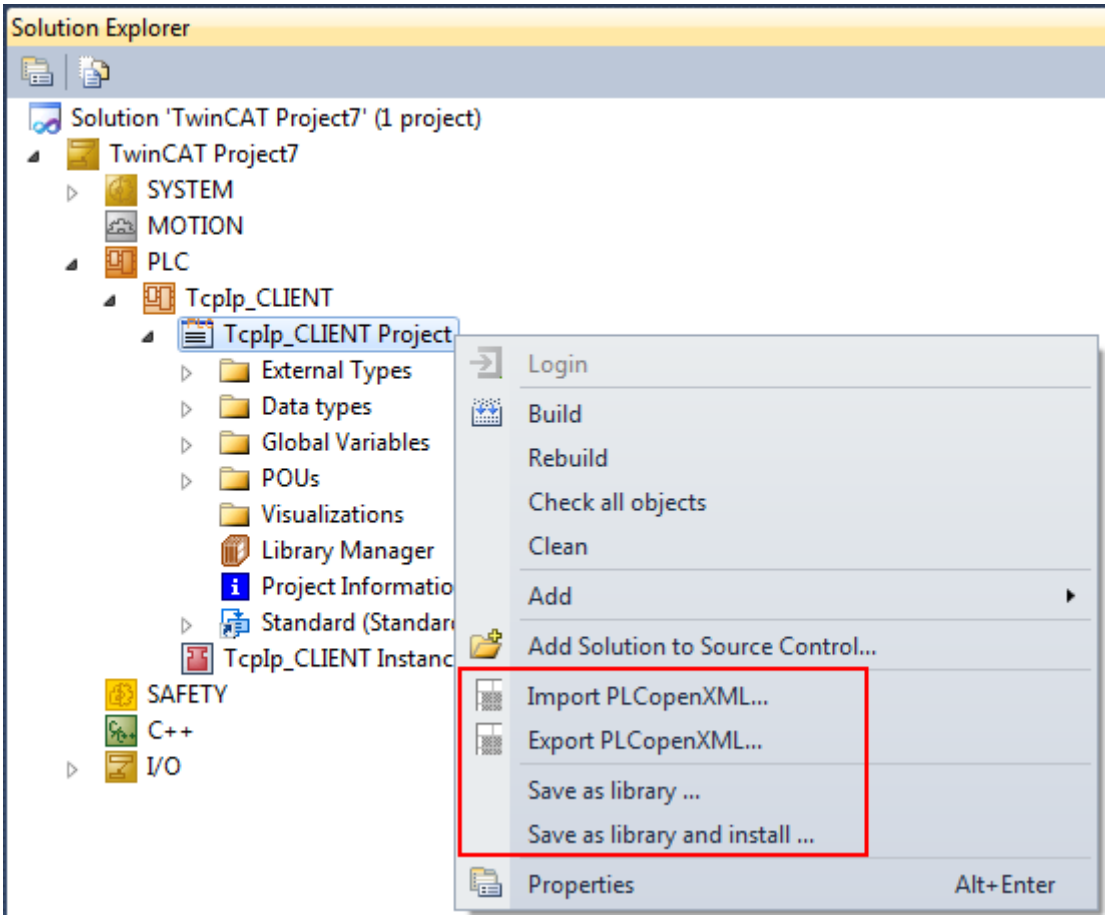
Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.8 ITcPlcIECProject

5.8.1 ITcPlcIECProject

Die ITcPlcIECProject-Schnittstelle stellt Methoden zur Verfügung, die den Import und Export von dem PlcOpen XML-Standard entsprechenden SPS-Projekten oder das Abspeichern von SPS-Projekten in Form einer SPS-Bibliothek ermöglichen. Im Vergleich zur grafischen TwinCAT XAE Benutzeroberfläche bietet diese Schnittstelle die folgenden vier Optionen:



Methoden (in Vtable-Reihenfolge)

ITcPlcIECProject-Methoden	Beschreibung	Verfügbar seit
PlcOpenExport() [▶ 170]	Exportiert die spezifizierten Tree Items mitsamt deren Inhalt in eine PlcOpen-konforme XML-Datei	TwinCAT 3.1
PlcOpenImport() [▶ 170]	Importiert eine PlcOpen-konforme XML-Datei mitsamt Inhalt	TwinCAT 3.1
SaveAsLibrary() [▶ 171]	Speichert das ausgewählte SPS-Projekt als SPS-Bibliothek und installiert diese gegebenenfalls.	TwinCAT 3.1

Versionsinformationen

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.8.2 PlcImportOptions

Dieses Enum enthält die folgenden Optionen und wird beim Import einer PlcOpen-konformen XML-Datei über die Methode ITcPlcIECProject [▶ 168]::PlcOpenImport() [▶ 170] verwendet.

Element	Beschreibung	Unterstützt von
PLCIMPORTOPTIONS_NONE	---	TwinCAT 3.1
PLCIMPORTOPTIONS_RENAME	In dem Fall, in dem das importierte Element bereits im SPS-Projekt existiert, wird es automatisch umbenannt	TwinCAT 3.1
PLCIMPORTOPTIONS_REPLACE	In dem Fall, in dem das importierte Element bereits im SPS-Projekt existiert, wird das bestehende Element ersetzt	TwinCAT 3.1
PLCIMPORTOPTIONS_SKIP	In dem Fall, in dem das importierte Element bereits im SPS-Projekt existiert, wird es übersprungen	TwinCAT 3.1

5.8.3 ITcPlcIECProject::PlcOpenExport

Exportiert die spezifizierten Tree Items mitsamt deren Inhalt in eine PlcOpen-konforme XML-Datei. Ein Pfad zur XML-Datei und zu den Tree Items wird in einem Parameter dieser Methode angegeben.

```
HRESULT PlcOpenExport(BSTR bstrFile, BSTR bstrSelection);
```

Parameters

bStrFile

[in] : Pfad zur XML-Datei

bstrSelection

[in] : Auswahl der zu exportierenden Tree Items. Die Elemente werden durch ein Semikolon getrennt, z.B. "POUs.FBItem1;POUs.FBItem2"

Rückgabewerte

S_OK	Elemente wurden erfolgreich in PlcOpen XML-Datei exportiert.

5.8.4 ITcPlcIECProject::PlcOpenImport

Importiert eine PlcOpen-konforme XML-Datei mitsamt dessen Inhalt. Ein Pfad zur XML-Datei wird der Methode über einen Parameter übergeben.

```
HRESULT PlcOpenImport(BSTR bstrFile, int options, BSTR bstrSelection, VARIANT_BOOL
bFolderStructure);
```

Parameter

bstrFile

[in] : Pfad zur XML-Datei.

options

[in, optional, defaultvalue(0)] : Optionen für den Import gemäß Enum PLCIMPORTOPTIONS [▶ 169].

bstrSelection

[in, optional, defaultvalue("")] : Wählt die aus der XML-Struktur zu importierenden Elemente aus

bFolderStructure

[in, optional, defaultvalue(-1)] : Wenn auf true (-1) gesetzt, dann bleibt die Ordnerstruktur der importierten Objekte unverändert, falls diese Information in der XML-Datei verfügbar ist

Rückgabewerte

S_OK	PlcOpen XML-Datei wurde erfolgreich importiert.

5.8.5 ITcPlcIECProject::SaveAsLibrary

Speichert das SPS-Projekt als SPS-Bibliothek und installiert diese bei Bedarf anschließend.

```
HRESULT SaveAsLibrary(BSTR bstrLibraryPath, VARIANT_BOOL binstall);
```

Parameter

bStrLibraryPath

[in] : Pfad zum Ort, an dem die SPS-Bibliothek gespeichert werden soll

binstall

[in] : auf „;true“; gesetzt, wenn SPS-Bibliothek auch installiert werden soll

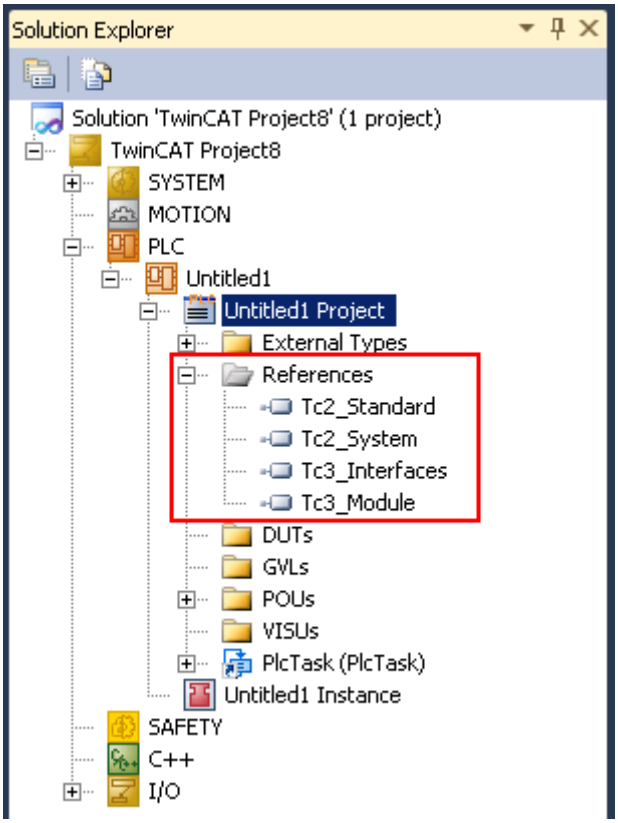
Rückgabewerte

S_OK	SPS-Projekt wurde erfolgreich als SPS-Bibliothek gespeichert.

5.9 ITcPlcLibraryManager

5.9.1 ITcPlcLibraryManager

Die ITcPlcLibraryManager-Schnittstelle erweitert das Automation Interface indem der Zugriff auf SPS-Bibliotheken eines TwinCAT 3 SPS-Projekts oder auf SPS-Repositories eines TwinCAT-Systems ermöglicht wird.



Methoden (in Vtable-Reihenfolge)

ITcPlcLibraryManager-Methoden	Beschreibung	Verfügbar seit
AddLibrary() [▶ 173]	Fügt eine Bibliothek zu einem SPS-Projekt hinzu.	TwinCAT 3.1
AddPlaceholder() [▶ 174]	Fügt einen Platzhalter zu einem SPS-Projekt hinzu.	TwinCAT 3.1
InsertRepository() [▶ 174]	Erstellt ein Repository, das einen logischen Container für mehrere Bibliotheken darstellt.	TwinCAT 3.1
InstallLibrary() [▶ 174]	Installiert eine Bibliothek in einem Repository.	TwinCAT 3.1
MoveRepository() [▶ 175]	Ändert die Position eines Repositories innerhalb der Repositorypfadliste.	TwinCAT 3.1
RemoveReference() [▶ 175]	Entfernt Bibliothek aus dem SPS-Projekt.	TwinCAT 3.1
RemoveRepository() [▶ 175]	Entfernt ein Repository.	TwinCAT 3.1
ScanLibraries() [▶ 176]	Gibt eine Liste **aller** in einem System gefundenen Bibliotheken zurück. Das zurückgegebene Objekt ist vom Typ ITcPlcLibraries [▶ 178], welches eine Sammlung von ITcPlcLibrary [▶ 178]-Objekten ist.	TwinCAT 3.1
SetEffectiveResolution() [▶ 176]	Legt die effektive Auflösung eines Platzhalters fest	TwinCAT 3.1
UninstallLibrary() [▶ 177]	Deinstalliert eine Bibliothek aus einem Repository.	TwinCAT 3.1

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibraryManager-Properties	Get / Set	Beschreibung	Verfügbar seit
References	Ja / Nein	Ruft ein Objekt vom Typ ITcPlcReferences [▶ 177] ab, das eine Sammlung von ITcPlcLibrary [▶ 178] oder ITcPlcPlaceholderRef-Objekten ist. Repräsentiert eine Liste aller dem SPS-Projekt hinzugefügter Referenzen.	TwinCAT 3.1
Repositories	Ja / Nein	Ruft ein Objekt vom Typ ITcPlcLibRepositories [▶ 180] ab, das eine Sammlung von ITcPlcLibRepository [▶ 179]-Objekten ist. Repräsentiert eine Liste aller derzeit konfigurierten Repositories.	TwinCAT 3.1

Versionsinformationen

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.9.2 ITcPlcLibraryManager::AddLibrary

Fügt eine Bibliothek zum SPS-Projekt hinzu. Eine Bibliothek kann entweder mit Hilfe ihrer Attribute (Name, Version, Unternehmen) oder ihres Anzeigentextes hinzugefügt werden.

```
HRESULT AddLibrary(BSTR bstrLibName, BSTR bstrVersion, BSTR bstrCompany);
```

Parameter

bstrLibName	[in] Bibliotheksname.
bstrVersion	[in, optional, defaultvalue("")] Bibliotheksversion.
bstrCompany	[in, optional, defaultvalue("")] Unternehmen, das die Bibliothek erstellt hat.

Rückgabewerte

S_OK	Bibliothek wurde erfolgreich hinzugefügt.

Kommentare

Zwei geläufige Methoden für das Hinzufügen einer SPS-Bibliothek sind:

- libraryManager.AddLibrary("Tc2_MDP", "3.2.0.0", "Beckhoff Automation GmbH"); // Hinzufügen der Bibliothek über ihre Attribute
- libraryManager.AddLibrary("Tc2_MDP, 3.2.0.0 (Beckhoff Automation GmbH)"); // Hinzufügen der Bibliothek mit Hilfe ihres Anzeigenamens

Wobei librayManager ein Objekt vom Typ ITcPlcLibraryManager ist.

5.9.3 ITcPlcLibraryManager::AddPlaceholder

Fügt einen Platzhalter zum SPS-Projekt hinzu. Ein Platzhalter kann entweder mit Hilfe seiner Attribute (Platzhaltername, Bibliotheksname, Bibliotheksverion, Bibliotheksverteiler) oder mit Hilfe seines Namens, wenn der Platzhalter bereits existiert, hinzugefügt werden.

```
HRESULT AddPlaceholder(BSTR bstrPlaceholderName, BSTR bstrDefaultLibName, BSTR bstrDefaultVersion,
BSTR bstrDefaultDistributor);
```

Parameter

bstrPlaceholderName	[in] Platzhaltername.
bstrDefaultLibName	[in] Standardbibliotheksname, auf den der Platzhalter zeigt.
bstrVersion	[in, optional, defaultvalue("")] Standardbibliotheksversion.
bstrCompany	[in, optional, defaultvalue("")] Unternehmen, das die Bibliothek erstellt hat.

Rückgabewerte

S_OK	Platzhalter wurde erfolgreich hinzugefügt.

5.9.4 ITcPlcLibraryManager::InsertRepository

Fügt ein neues Bibliotheks-Repository an angegebener Position hinzu. Die Position stellt den Index dar, bei welchem das Repository in der Repository-Liste in TwinCAT 3 positioniert ist. Das Repository wird mit dem Index und mit Hilfe seines Namens und dem Verzeichnispfad im Dateisystem identifiziert.

```
HRESULT InsertRepository(BSTR bstrName, BSTR rootFolder, int iIndex);
```

Parameter

bstrName	[in] Repository-Name.
rootFolder	[in] Pfad zum Repository (Dateisystem).
iIndex	[in] Zeigt an, an welcher Position das Repository sich in der Repository-Liste befindet.

Rückgabewerte

S_OK	Repository wurde erfolgreich eingefügt

5.9.5 ITcPlcLibraryManager::InstallLibrary

Installiert eine Bibliothek in ein bestehendes Bibliotheks-Repository.

```
HRESULT InstallLibrary(BSTR bstrRepositoryName, BSTR bstrLibPath, VARIANT_BOOL bOverwrite);
```

Parameter

bstrRepositoryName	[in] : Name des Repository, in das die Bibliothek einzufügen ist
bstrLibPath	[in] : Pfad zur Bibliothek
bOverwrite	[in] : Wenn bereits eine andere Bibliothek besteht, setzen Sie diesen Parameter um letztere zu überschreiben

Rückgabewerte

S_OK	Bibliothek wurde erfolgreich installiert.

5.9.6 ITcPlcLibraryManager::MoveRepository

Verschiebt das Repository an eine andere Position in der Repository-Liste. Die Position wird mit Hilfe ihres Index gekennzeichnet, der bei 0 beginnt.

```
HRESULT MoveRepository(BSTR bstrRepositoryName, int iNewIndex);
```

Parameter

bstrRepositoryName	[in] : Name des Repository
iNewIndex	[in] : Index des Repository

5.9.7 ITcPlcLibraryManager::RemoveReference

Entfernt eine Referenz aus dem eigentlichen SPS-Projekt. Eine Referenz ist entweder eine Bibliothek oder ein Platzhalter.

Bitte beachten: Im Falle einer Bibliothek wird hiermit nur die Referenz, aber nicht die Bibliotheksdatei selbst aus dem Repository entfernt. Hierzu muss die Methode UninstallLibrary() [▶ 177] verwendet werden.

Ähnlich wie im Falle der Methode AddLibrary() [▶ 173], kann eine Bibliothek entweder mit ihren Attributen (Name, Version, Unternehmen) oder mit dem Anzeigetext spezifiziert werden.

```
HRESULT RemoveReference(BSTR bstrLibName, BSTR bstrVersion, BSTR bstrCompany);
```

Parameter

bstrLibName	[in] : Name der Bibliothek
bstrVersion	[in, optional, defaultvalue("")] :
bstrCompany	[in, optional, defaultvalue("")]

Kommentare

Zwei geläufige Methoden für das Entfernen einer SPS-Bibliothek sind:

- libraryManager.RemoveReference("Tc2_MDP", "3.2.0.0", "Beckhoff Automation GmbH"); // Entfernen einer Bibliothek mit Hilfe ihrer Attribute
- libraryManager.RemoveReference("Tc2_MDP, 3.2.0.0 (Beckhoff Automation GmbH)"); // Entfernen einer Bibliothek mit Hilfe ihres Anzeigenamens

Wobei librayManager ein Objekt vom Typ ITcPlcLibraryManager ist.

5.9.8 ITcPlcLibraryManager::RemoveRepository

Entfernt ein Bibliotheks-Repository. Ein Repository wird mit Hilfe seines eindeutigen Namens spezifiziert.

```
HRESULT RemoveRepository(BSTR bstrName);
```

Parameter

bstrName

[in] Name des Repository.

5.9.9 ITcPlcLibraryManager::ScanLibraries

Gibt eine Collection aller eingetragenen Bibliotheken in allen Repositories zurück.

```
HRESULT ScanLibraries(ITcPlcLibraries** ppEnumLibraries);
```

Parameter

ppEnumLibraries	[out, retval] Gibt ein Objekt vom Typ ITcPlcLibraries [▶ 178] zurück, das eine Sammlung von ITcPlcLibrary [▶ 178]-Objekten ist.

Kommentare

Diese Methode bietet die gleiche Funktionalität wie der Import der folgenden XML-Struktur über ConsumeXml() [▶ 159] unter dem "Library Manager" Tree Item:

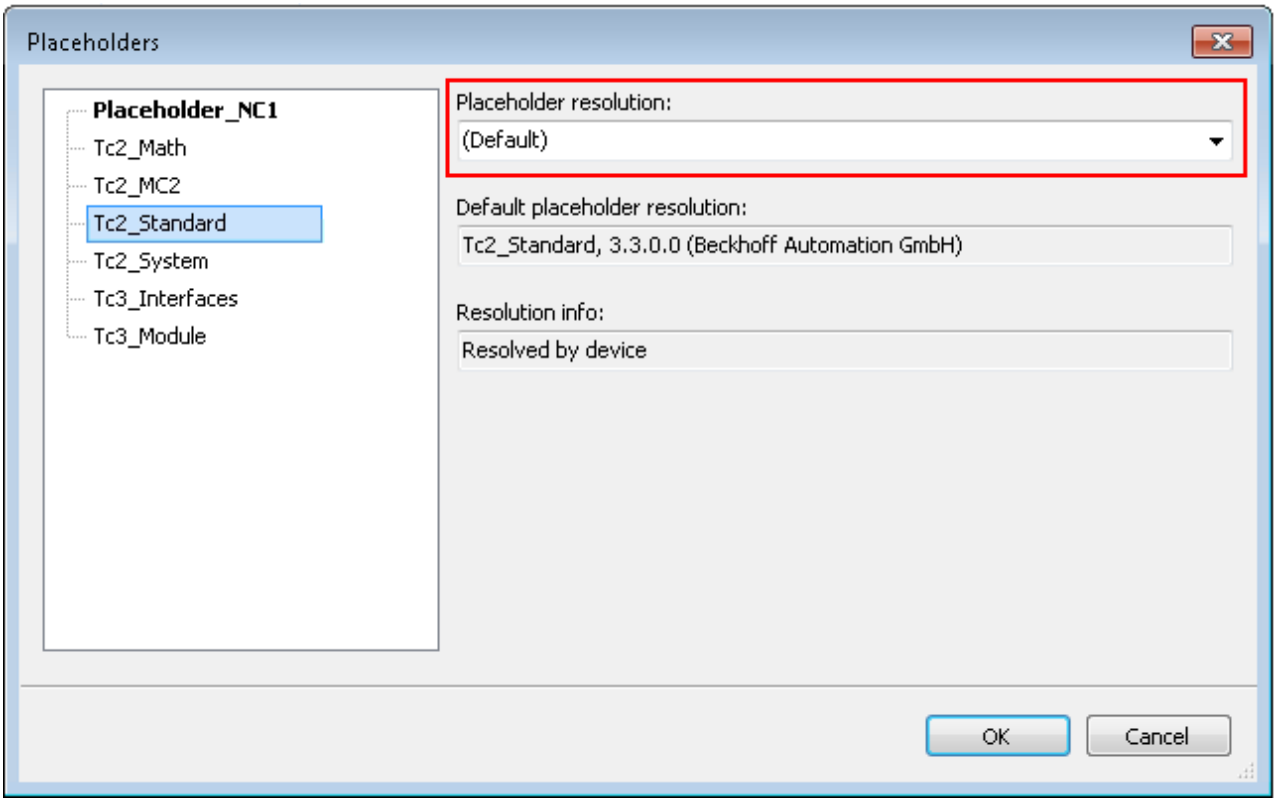
```
<TreeItem>
<PlcLibDef>
    <ScanLibraries>
     <Active>true</Active>
    </ScanLibraries>
</PlcLibDef>
</TreeItem>
```

5.9.10 ITcPlcLibraryManager::SetEffectiveResolution

Legt die effektive Auflösung, sprich die effektive Bibliothek, eines Platzhalters fest.

```
HRESULT SetEffectiveResolution(BSTR bstrPlaceholderName, BSTR strLibName, BSTR bstrVersion, BSTR
bstrDistributor);
```

In TwinCAT XAE kann die effektive Auflösung über das Platzhalterkonfigurationsfenster festgelegt werden.



Bitte beachten: Die Standardauflösung eines Platzhalters wird beim Hinzufügen des Platzhalters über ITcPlcLibraryManager [▶ 171]::AddPlaceholder() [▶ 174] festgelegt.

Parameter

bstrPlaceholderName	[in] Name des Platzhalters, für den die effektive Auflösung festzulegen ist.
bstrLibName	[in] Bibliotheksname für effektive Auflösung.
bstrVersion	[in, optional, defaultvalue("")] Bibliotheksversion.
bstrDistributor	[in, optional, defaultvalue("")] Unternehmen, das die Bibliothek erstellt hat.

Rückgabewerte

S_OK	Effektive Auflösung wurde erfolgreich festgelegt.

5.9.11 ITcPlcLibraryManager::UninstallLibrary

Deinstalliert eine Bibliothek aus einem Repository.

```
HRESULT UninstallLibrary(BSTR bstrRepositoryName, BSTR bstrLibraryName, BSTR bstrVersion, BSTR bstrDistributor);
```

Parameter

bstrRepositoryName	[in] : Name des Repository
bstrLibraryName	[in] : Name der Bibliothek
bstrVersion	[in, optional] : Version der Bibliothek
bstrDistributor	[in, optional] : Verteiler der Bibliothek

5.10 ITcPlcReferences

ITcPlcReferences stellt eine Collection von ITcPlcLibRef [▶ 179]-Objekten dar, die z.B. bei Verwendung der Eigenschaft ITcPlcLibraryManager [▶ 171]::References zurückgegeben wird.

Methoden (in Vtable-Reihenfolge)

ITcPlcLibRepositories-Methoden	Beschreibung	Verfügbar seit
get_Item() [▶ 180]	Gibt ein Element vom Typ ITcPlcLibRef [▶ 179] zurück, das sich an einem bestimmten Ort befindet.	TwinCAT 3.1

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibRepositories-Properties	Get / Set	Beschreibung	Verfügbar seit
Count	Ja / Nein	Gibt die Anzahl der in der Collection enthaltenen ITcPlcLibRef [▶ 179]-Objekte zurück.	TwinCAT 3.1

Versionsinformationen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.11 ITcPlcLibrary

Repräsentiert eine einzelne SPS-Bibliothek bei Verwendung mit einer ITcPlcLibraries [▶ 178]-Collection und der Methode ITcPlcLibraryManager [▶ 171]::ScanLibraries() [▶ 176] oder der Eigenschaft ITcPlcLibraryManager [▶ 171]::References.

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibrary-Properties	Get / Set	Beschreibung	Verfügbar seit
DisplayName	Ja / Nein	Name anzeigen um Bibliothek in Bibliothekenliste zu identifizieren	TwinCAT 3.1
Distributor	Ja / Nein	Bibliothekersteller	TwinCAT 3.1
Name	Ja / Nein	Bibliotheksname	TwinCAT 3.1
Version	Ja / Nein	Bibliotheksversion	TwinCAT 3.1

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.12 ITcPlcLibraries

5.12.1 ITcPlcLibraries

ITcPlcLibraries repräsentiert eine Sammlung von ITcPlcLibrary [▶ 178]-Objekten, zum Beispiel bei Verwendung der Methode ITcPlcLibraryManager [▶ 171]::ScanLibraries() [▶ 176] oder der Eigenschaft ITcPlcLibraryManager [▶ 171]::Libraries.

Methoden (in Vtable-Reihenfolge)

ITcPlcLibraries-Methoden	Beschreibung	Verfügbar seit
get_Item() [▶ 179]	Gibt ein Element vom Typ ITcPlcLibrary [▶ 178] zurück, das sich an einem bestimmten Ort befindet.	TwinCAT 3.1

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibraries-Properties	Get / Set	Beschreibung	Verfügbar seit
Count	Ja / Nein	Gibt die Anzahl der in der Sammlung enthaltenen ITcPlcLibrary [▶ 178]-Objekte zurück	TwinCAT 3.1

Versionsinformationen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.12.2 ITcPlcLibraries::get_Item

Gibt das an der angegebenen Position befindliche ITcPlcLibrary-Objekt zurück.

```
HRESULT AddLibrary(long n, ITcPlcLibrary** pipType);
```

Parameter

n	[in] Position von Element in Liste.
pipType	[out, retval] Gibt Objekt vom Typ ITcPlcLibrary zurück

5.13 ITcPlcLibRef

ITcPlcLibRef stellt eine Basisklasse für ITcPlcLibrary [▶ 178] oder ITcPlcPlaceholderRef [▶ 179]-Objekte dar.

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibRepositories-Properties	Get / Set	Beschreibung	Verfügbar seit
Name	Ja / Nein	Gibt den Namen des ITcPlcLibRef-Objekts zurück	TwinCAT 3.1

Versionsinformationen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.14 ITcPlcPlaceholderRef

Stellt einen einzelnen SPS-Platzhalter dar, bei Verwendung mit der Collection ITcPlcReferences [▶ 177] und der Methode ITcPlcLibraryManager [▶ 171]::ScanLibraries() [▶ 176] oder der Eigenschaft ITcPlcLibraryManager [▶ 171]::References.

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibrary-Properties	Get / Set	Beschreibung	Verfügbar seit
DisplayName	Ja / Nein	Name anzeigen um Bibliothek in Bibliothekenliste zu identifizieren	TwinCAT 3.1
Distributor	Ja / Nein	Bibliothekersteller	TwinCAT 3.1
Name	Ja / Nein	Bibliotheksname	TwinCAT 3.1
Version	Ja / Nein	Bibliotheksversion	TwinCAT 3.1

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.15 ITcPlcLibRepository

Die ITcPlcLibRepository-Schnittstelle repräsentiert ein einzelnes Repository, zum Beispiel bei Verwendung mit der Collection ITcPlcLibRepositories [▶ 180] und der Eigenschaft ITcPlcLibraryManager [▶ 171]::Repositories.

Eigenschaften

ITcPlcLibRepository-Properties	Get / Set	Beschreibung	Verfügbar seit
Folder	Ja / Nein	Pfad zum Repository (Dateisystem)	TwinCAT 3.1
Name	Ja / Nein	Repository-Name.	TwinCAT 3.1

Voraussetzungen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.16 ITcPlcLibRepositories

5.16.1 ITcPlcLibRepositories

ITcPlcLibraries stellt eine Sammlung von ITcPlcLibRepository [▶ 179]-Objekten dar, zum Beispiel bei Verwendung der Eigenschaft ITcPlcLibraryManager [▶ 171]::Repositories.

Methoden (in Vtable-Reihenfolge)

ITcPlcLibRepositories-Methoden	Beschreibung	Verfügbar seit
get_Item() [▶ 180]	Gibt ein Element vom Typ ITcPlcLibRepository [▶ 179] zurück, das sich an einem bestimmten Ort befindet.	TwinCAT 3.1

Eigenschaften (in Vtable-Reihenfolge)

ITcPlcLibRepositories-Properties	Get / Set	Beschreibung	Verfügbar seit
Count	Ja / Nein	Gibt die Anzahl der in der Sammlung enthaltenen ITcPlcLibRepository [▶ 179]-Objekte zurück.	TwinCAT 3.1

Versionsinformationen

Erforderliche TwinCAT Version
Diese Schnittstelle wird ab TwinCAT 3.1 unterstützt.

5.16.2 ITcPlcLibRepositories::get_Item

Gibt das an der angegebenen Position befindliche ITcPlcLibRepository-Objekt zurück.

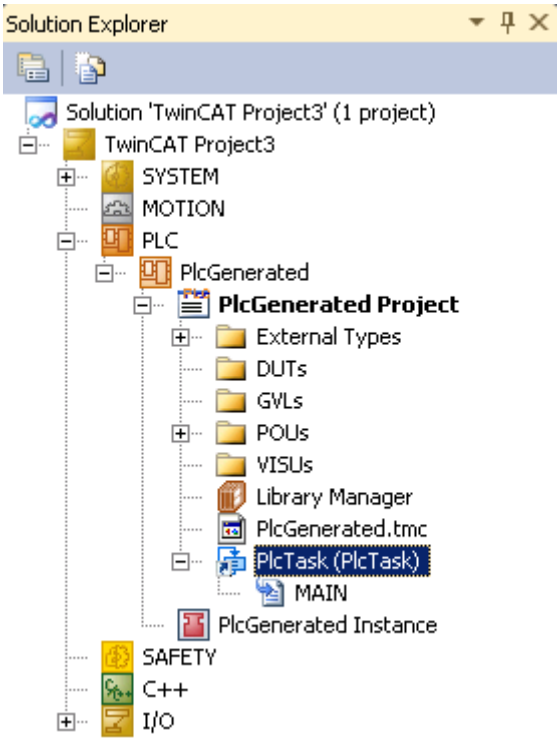
```
HRESULT AddLibrary(long n, ITcPlcLibRepository** ppRepo);
```

Parameter

n	[in] Position von Element in Liste.
pipType	[out, retval] Gibt Objekt vom Typ ITcPlcLibRepository zurück

5.17 ITcPlcTaskReference

Dank der ITcPlcTaskReference-Schnittstelle können Programmierer die derzeit verknüpfte Task eines SPS-Projekts abrufen oder setzen. Dies entspricht dem folgenden TwinCAT XAE-Eintrag:



Eigenschaften (in Vtable-Reihenfolge)

ITcPlcTaskReference-Properties	Get / Set	Beschreibung	Verfügbar seit
LinkedTask	Ja / Ja	Abrufen oder Setzen der verknüpften Task eines SPS-Projekts. Beim Festlegen einer neuen verknüpften Task wird die Task als Zeichenkette, welche den Pfad zum Task in der TwinCAT XAE-Baumstruktur enthält, festgelegt.	TwinCAT 3.1

6 Beispiele

6.1 Beispiel Downloads:

Die meisten Beispiele und Anleitungen stehen in unserem Abschnitt „Best Practices“ zur Verfügung, um diese schnell wiederzufinden. Dieser Download-Abschnitt bietet ebenfalls einsatzbereite Beispiele als Binär- oder Quellcode.

Wichtige Hinweise:

- Alle C# Beispiele basieren auf einem Visual Studio Projekt mit mindestens .NET 4.0.
- Alle C#-Beispiele beinhalten eine Referenz auf die "TwinCAT XAE Base" Typbibliothek in Version 2.1. In Abhängigkeit der installierten TwinCAT-Version muss die Referenz zu dieser Bibliothek gegebenenfalls aktualisiert werden, siehe Installierungsartikel [▶ 18].
- Achten Sie bitte darauf, dass alle Visual Studio Plugin Beispiele nur unter Visual Studio 2010 und 2012 funktionieren. Zur Entwicklung von Add-Ins für Visual Studio 2013 empfiehlt Microsoft die Verwendung der VSPackage Erweiterung, wie dies auf der entsprechenden MSDN-Website über Creating Add-ins and Wizards beschrieben ist.

Beispiel Nr.	Beschreibung	Programmier- / Scriptsprache	Ab TwinCAT Version	Download
1	Scripting Container [▶ 182]	C#	Abhängig von Skript	ScriptingContainer nur Binaries ScriptingContainer Quelle
2	CodeGenerationDemo [▶ 186]	C#	3.1 Build 4014.0	CodeGenerationDemo nur Binaries CodeGenerationDemo Quelle
3	Visual Studio Plugin PlcVersionInfo [▶ 188]	C#	3.1 Build 4016.6	PluginSample_PlcVersionInfo.zip

6.2 Scripting Container

6.2.1 Scripting Container

Der Scripting Contrainer (download hier [▶ 182]) ist eine C# WPF-Anwendung, die eine Sammlung aller verfügbaren Automation Interface-Beispiele darstellt. Die meisten dieser Beispiele stehen auch als eigenständige Beispieldownloads zur Verfügung. Allerdings werden neue Bespiele zunächst im Scripting Container veröffentlicht, bevor sie als getrenntes Beispiel erstellt werden. Aus diesem Grunde empfehlen wir Ihnen, dass Sie sich mit dieser Anwendung vertraut machen und sie regelmäßig auf neue Beispiele hin überprüfen.

Dieser Artikel beschreibt den allgemeinen Aufbau der Scripting Container-Anwendung und besteht aus den folgenden Themen:

- Grundlegender Aufbau
- Erstes Setup

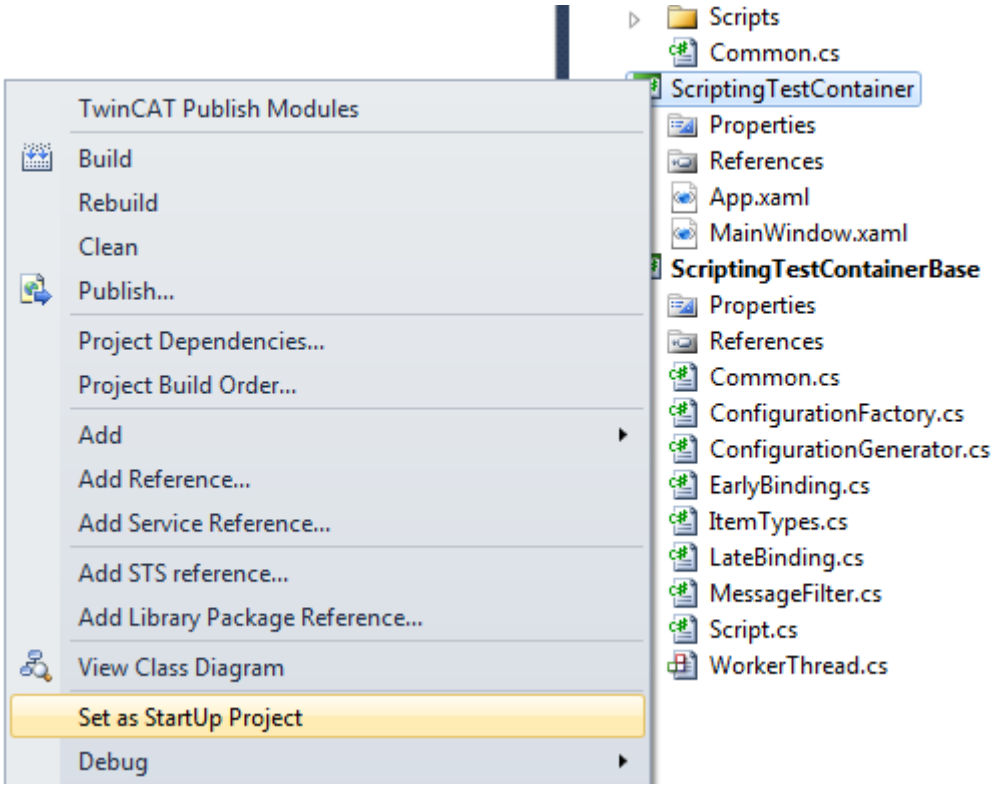
Grundlegender Aufbau

Wie bereits oben erwähnt besteht der ScriptingContainer aus einer Sammlung von Automation Interface-Beispielen, die durch verschiedene Projekte (jedes mit einer eigenen UI) im ScriptingContainer dargestellt sind. Die nachfolgende Tabelle gibt einen Überblick über alle verfügbaren Projekte und den Verknüpfungen zu den entsprechenden Dokumentationsartikeln.

Projektname	Beschreibung
CodeGenerationDemo [▶ 186]	Implementiert Automation Interface-Code, der drei unterschiedliche TwinCAT-Konfigurationen aus einer XML-Datei liest.
CopyProjectDemo	Kopiert I/Os und die Achsenkonfiguration aus einer bestehenden TwinCAT-Konfiguration in eine neue Konfiguration.
ScriptingTestContainer [▶ 183]	Stellt eine Sammlung verfügbarer Automation Interface-Beispiele bereit, die von einer grafischen Bedieneroberfläche ausgeführt werden können.

Erstes Setup

Legen Sie beim ersten Öffnen des ScriptingContainer die gewünschte Bedieneroberfläche als Startprojekt fest, z.B. "ScriptingTestContainer" indem Sie dieses Projekt rechtsklicken und "Set as StartUp Project" auswählen. Dadurch wird sichergestellt, dass die richtige grafische Benutzeroberfläche beim Starten der Anwendung geladen wird.



Sie können nun die Anwendung starten, indem Sie das Menü„;Debug“;öffnen und auf „;Start Debugging“; klicken. Wenn Sie die Anwendung auf diese Weise starten, können Sie Breakpoints festlegen oder den Code schrittweise ausführen, um das ausgeführte Skript zu begutachten.

6.2.2 Projekte

6.2.2.1 Scripting Container: ScriptingTestContainer

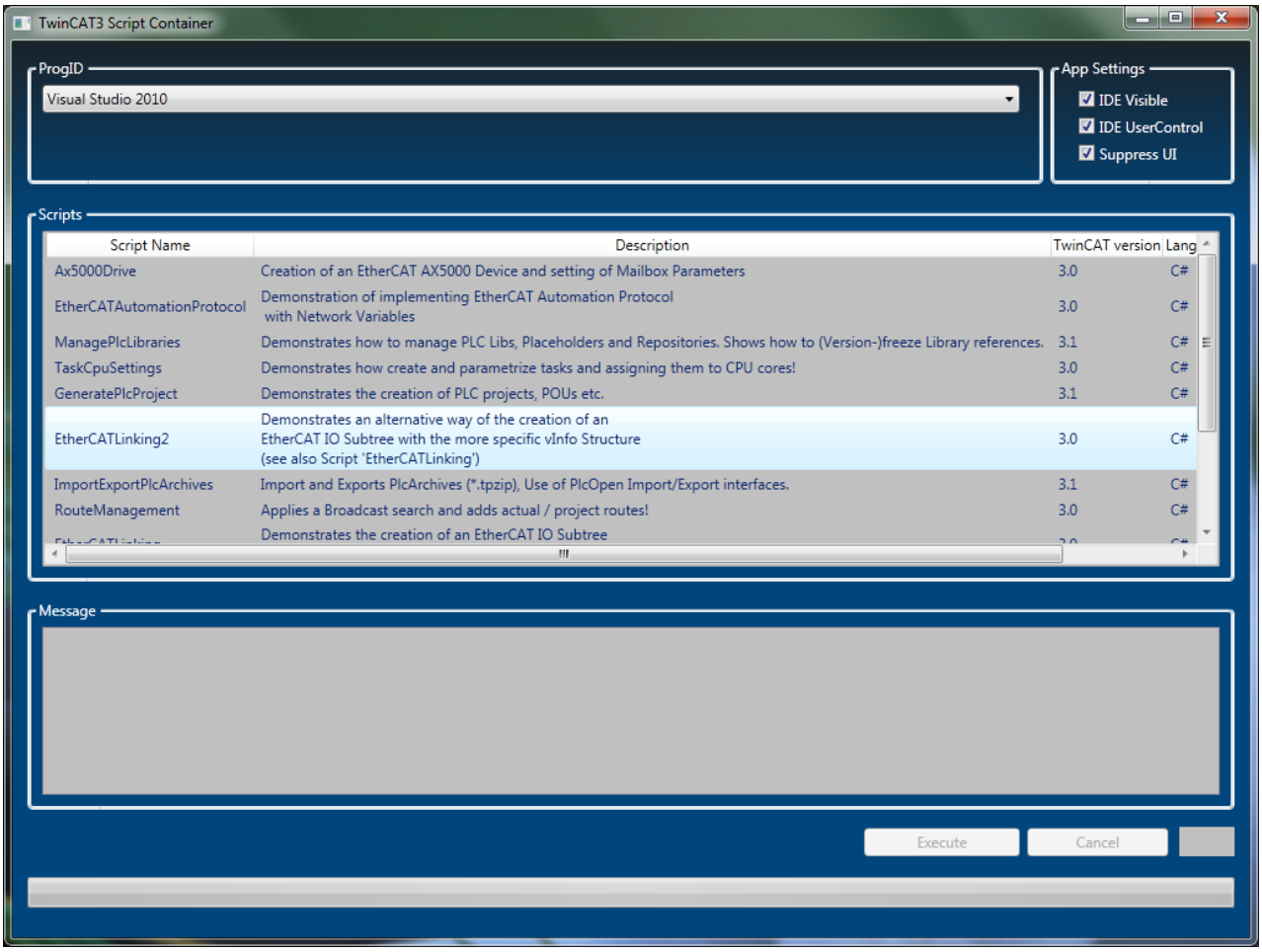
Dieser Artikel beschreibt den grundlegenden Aufbau des ScriptingTestContainer-Projekts und besteht aus den folgenden Themen:

- Die grafische Benutzeroberfläche (GUI)
- Früh und spät gebundene Skriptbeispiele
- Speicherort von Beispielskripts (<31>„Wo ist der Automation Interface-Code???”</31>)
- Hinter den Kulissen: Klassenaufbau
- Hinter den Kulissen: Methodenaufbau

- Hinter den Kulissen: Eigene Beispiele implementieren

Die grafische Benutzeroberfläche (GUI)

Nachdem Sie die Scripting Container-Anwendung gestartet haben, sieht ihre grafische Benutzeroberfläche wie folgt aus:



Im Dropdown-Listenfeld oben im Fenster können Sie die Version von Visual Studio [▶ 30] auswählen, die Sie für die Erstellung der TwinCAT-Konfiguration verwenden möchten.

Es stehen viele Beispielskripts zur Verfügung, die Sie in der Tabelle in der Mitte des Fensters auswählen können.

Um ein Skript auszuführen, wählen Sie es einfach aus der Tabelle aus und klicken Sie dann auf „Ausführen“. Im Verlauf der Code-Ausführung blendet das Skript Statusinformationen im Nachrichtenaufzeichnungsfenster unterhalb der Skripttabelle ein.

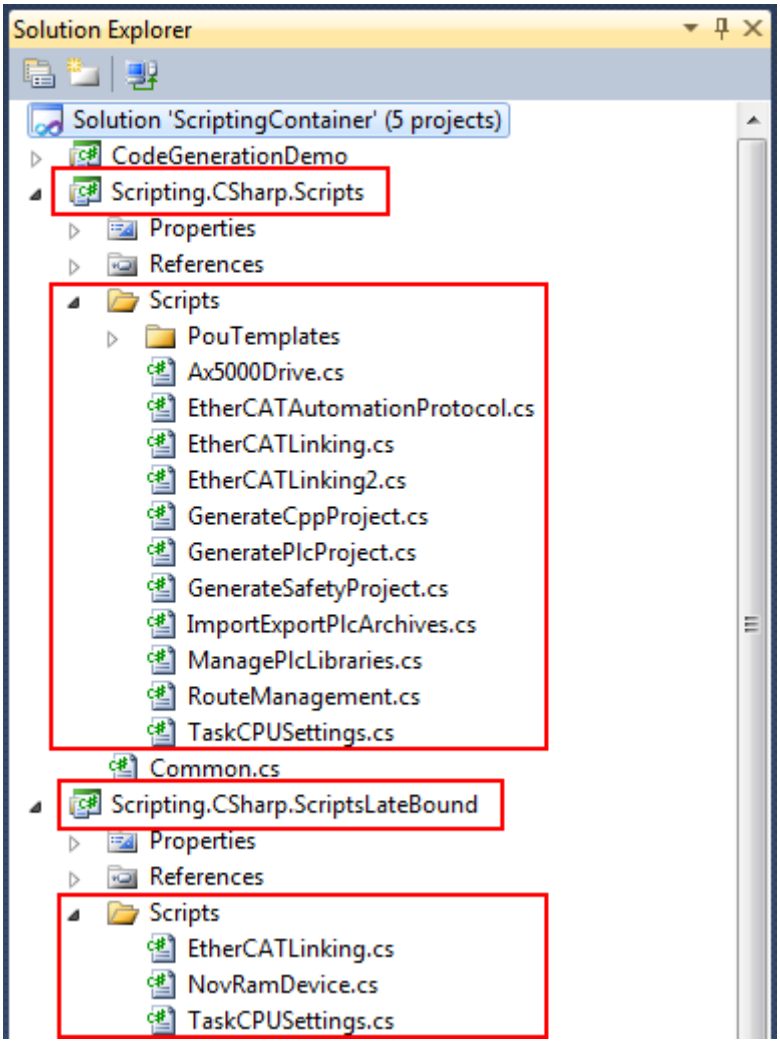
Um das Anzeigeverhalten von Visual Studio zu ändern, können Sie die Checkboxen oben rechts im Fenster umschalten. Standardmäßig wird Visual Studio während der Skriptausführung eingeblendet und nicht nach Ende des Skripts geschlossen.

Früh und spät gebundene Skriptbeispiele

Diese Anwendung enthält sowohl früh, als auch spät gebundene Skriptbeispiele. Spät gebundene Beispiele machen sich den .NET-Datentyp "dynamic" zunutze und bestimmen den Datentyp eines Objekts während der Laufzeit, während früh gebundene Skripts den einem Objekt entsprechenden Datentyp im Verlauf der Erstellung eines Objekts verwenden. Beide Möglichkeiten haben ihre Vor- und Nachteile und es hängt vom Entwickler und vom Projekt ab, welche Typisierung bevorzugt wird.

Speicherort von Beispielskripts („Wo ist der Automation Interface-Code???")

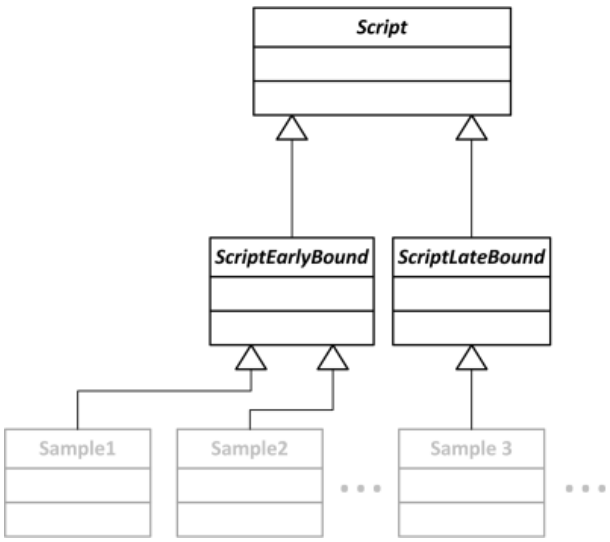
Alle Skriptbeispiele – früh und spät gebundene – befinden sich in einen eigenen Projekt-Container innerhalb der Visual Studio Solution. **Scripting.CSharp.Scripts** für früh gebundenen und **Scripting.CSharp.ScriptsLateBound** für spät gebundenen Beispielcode.



Jede Skriptdatei implementiert eine OnExecute()-Methode, in welcher der Skriptcode des TwinCAT-Automation Interface implementiert ist.

Hinter den Kulissen: Klassenaufbau

Jedes Codebeispiel wird durch eine eigene Klasse dargestellt, die entweder von der abstrakten Klasse "**ScriptEarlyBound**" oder von der Klasse "**ScriptLateBound**" abgeleitet ist, je nach der im Beispiel verwendeten Datentypisierung. Der Unterschied zwischen diesen beiden Klassen besteht darin, dass die ScriptEarlyBound Klasse die statische Typisierung für die DTE- und Solution-Objekte verwendet. Die ScriptLateBound Klasse benutzt stattdessen die dynamische Typisierung. Das folgende UML-Diagramm erläutert die Klassenhierarchie im Einzelnen. Die grauen Klassen stellen die eigentlichen Beispielskripts dar, die entweder von der ScriptEarlyBound-Klasse (wenn statische Typisierung verwendet werden soll) oder von der ScriptLateBound-Klasse (wenn dynamische Typisierung verwendet werden soll) abgeleitet sind.



Die abstrakte Klasse "Script" definiert Methoden und Attribute, die beiden Klassen, ScriptEarlyBound und ScriptLateBound, gemein sind.

Hinter den Kulissen: Methodenaufbau

Jede Beispielklasse enthält drei Methoden, die im Scripting Container für die Ausführung des Automation Interface-Codes verwendet werden. In folgender Tabelle wird deren Bedeutung beschrieben.

Signatur der abgeleiteten Methode	Beschreibung
OnInitialize (dynamic dte, dynamic solution, IWorker worker)	Die OnInitialize()-Methode wird normalerweise für den Automation Interface-Code verwendet, der eine neueTwinCAT XAE-Konfiguration öffnet oder vorbereitet.
OnCleanUp (IWorker worker)	Die OnCleanUp()-Methode kann zum Aufräumen der TwinCAT XAE-Konfiguration im Anschluss an die Code-Ausführung verwendet werden.
OnExecute (IWorker worker)	Diese Methode führt den eigentlichen Automation Interface-Skriptcode aus.

Vergleichen Sie auch die Implementierung von bestehenden Beispielskripts, um die Arbeitsweise jeder Methode besser zu verstehen.

Hinter den Kulissen: Eigene Beispiele implementieren

Es ist sehr einfach, eigene Automation Interface-Beispiele in den Scripting Container zu implementieren. Um eigenen Automation Interface-Code zu implementieren, muss ein Entwickler lediglich entscheiden, welche Bindung er verwenden möchte und anschließend eine neue Klasse, die von einer der Hauptklassen (ScriptEarlyBound oder ScriptLateBound) abgeleitet ist, implementieren, und dann die abgeleiteten Methoden implementieren.

6.3 CodeGenerationDemo

Der Scripting Container (download hier [▶ 182]) ist eine C# WPF-Anwendung, die eine Sammlung aller verfügbaren Automation Interface-Beispiele darstellt. Die meisten dieser Beispiele stehen auch als eigenständige Beispieldownloads zur Verfügung. Allerdings werden neue Beispiele zunächst im Scripting Container veröffentlicht, bevor sie als getrenntes Beispiel erstellt werden. Aus diesem Grunde empfehlen wir Ihnen, dass Sie sich mit dieser Anwendung vertraut machen und sie regelmäßig auf neue Beispiele hin überprüfen.

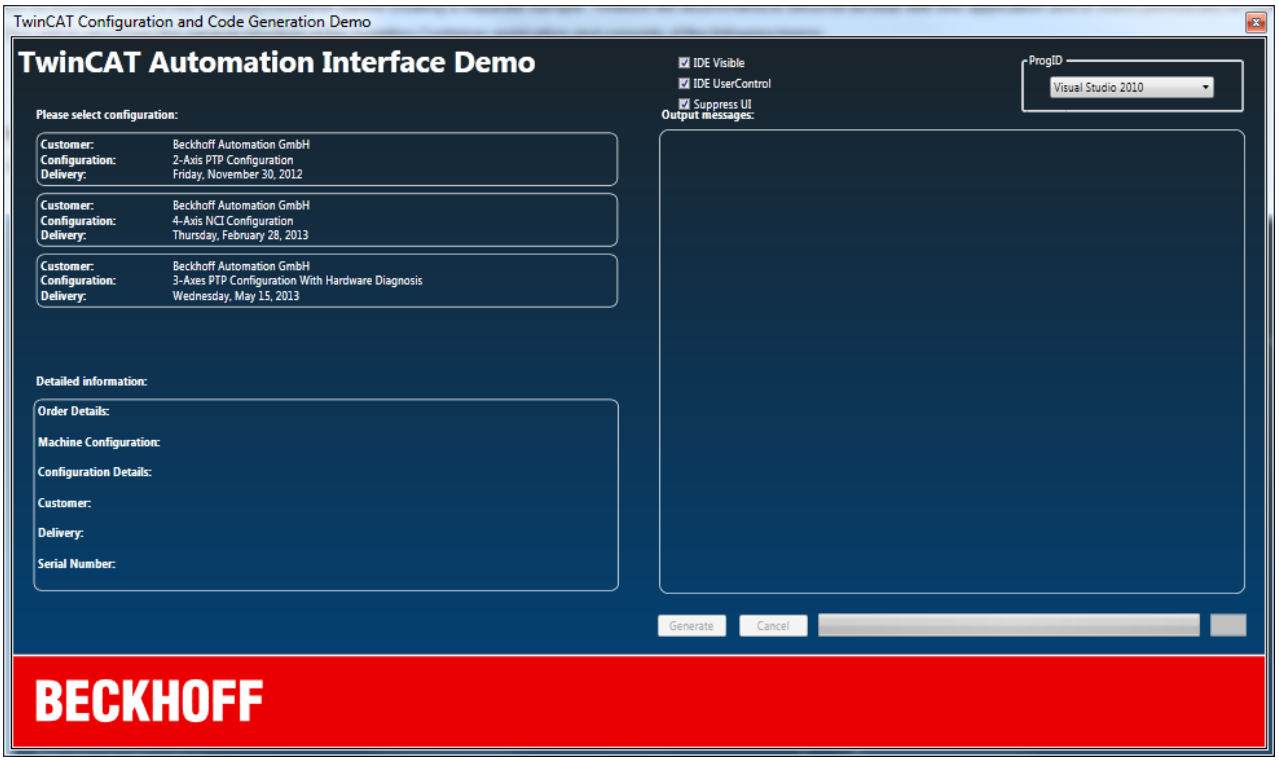
Dieser Artikel beschreibt den allgemeinen Aufbau der Scripting Container-Anwendung und besteht aus den folgenden Themen:

- Die grafische Benutzeroberfläche (GUI)

- Speicherort von Beispielskripts („Wo ist der Automation Interface-Code???")
- Speicherort von Daten („Woher stammen alle diese Daten?")

Die grafische Benutzeroberfläche (GUI)

Nachdem Sie die CodeGenerationDemo-Anwendung gestartet haben, sieht ihre grafische Benutzeroberfläche wie folgt aus:



Auf der linken Seite können Sie aus drei verschiedenen TwinCAT-Konfigurationen auswählen, jede mit ihrer eigenen I/O, Achse und SPS-Konfiguration

Im Dropdown-Listenfeld oben rechts können Sie die Visual Studio-Version auswählen, welche der Skript-Code für die Erstellung der Konfiguration verwenden soll, für den Fall, dass auf Ihrem System verschiedene Visual Studio-Versionen installiert sind.

Um die Erstellung der Konfiguration zu starten, wählen Sie einfach eine Konfiguration und drücken „Generate" (Erstellen).

Speicherort von Beispielskripts („Wo ist der Automation Interface-Code???")

Der eigentliche Automation Interface-Code befindet sich in den folgenden Klassen:

- CodeGenerationScript.cs
- ConfigurationScriptA.cs
- ConfigurationScriptB.cs
- ConfigurationScriptC.cs

Entsprechend der ausgewählten Konfiguration instanziiert die CodeGenerationDemo-Anwendung eine der drei ConfigurationScriptX.cs-Klassen, die alle von CodeGenerationScript.cs abgeleitet sind.

Jede Klasse stellt entsprechende Methoden für die Erstellung der SPS-, Achsen- oder I/O-Konfiguration zur Verfügung.

Speicherort von Daten („Woher stammen alle diese Daten?“)

Jede Konfiguration hat ihre eigenen I/O-, Achsen- und SPS-Einstellungen. Wie Sie wohl bereits festgestellt haben, ist diese Konfiguration nicht fest im Automation Interface-Code programmiert (wie in anderen Automation Interface-Beispielen). Im Falle dieses Beispiels sind diese Einstellungen in einer XML-Datei gespeichert, die sich unter "CodeGenerationDemo\Data\Orders.xml" befindet. Diese XML-Datei spezifiziert zwei wichtige XML-Unterstrukturen, die für die Speicherung der TwinCAT-Einstellungen verwendet werden.

Hauptbeschreibung einer Konfiguration:

```
<MachineOrders>
<Order id="1">
...
</Order>
</MachineOrders>
```

Diese Struktur definiert eine Konfiguration und legt ihre globalen beschreibenden Eigenschaften, wie Name, Beschreibung, ... fest. . Diese Eigenschaften werden auf der grafischen Benutzeroberfläche unter der Konfigurationsauswahl gezeigt.

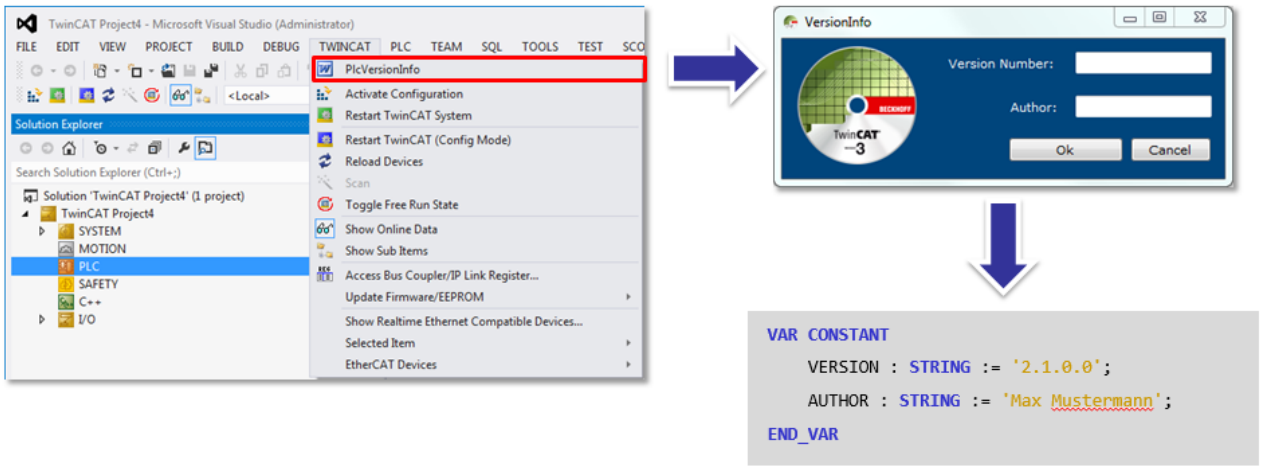
Eigentliche Konfiguration mit einer Referenz auf ihre Beschreibung:

```
<AvailableConfigurations>
<Configuration id="..." name="...">
...
</Configuration>
</AvailableConfigurations>
```

Diese Struktur spezifiziert die I/Os, Achsen, SPS-Bibliotheken und Verknüpfungen.

6.4 Visual Studio Plugin - PlcVersionInfo

Das PlcVersionInfo Beispiel zeigt, wie das TwinCAT Automation Interface in einem Visual Studio Plugin zu verwenden ist. Der Hauptzweck besteht darin, zu zeigen, wie ein einfaches Tool erzeugt werden kann, mit dessen Hilfe die SPS Entwicklung mit mehr Funktionalitäten erweitert werden kann. Mit Hilfe der kleinen Benutzerschnittstelle können Versions- und Autoreninformationen für einen SPS Funktionsbaustein eingegeben werden. Beim Ausführen des Beispiels wird diese Information in einem VAR_CONSTANT Bereich in allen Funktionsbausteinen innerhalb eines SPS-Projekts abgelegt. Auch wenn dieses Beispiel ziemlich einfach erscheinen mag, zeigt es doch klar, wie leistungsstark Visual Studio Plugins sein können und wie sie die Entwicklungsarbeit des Nutzers erheblich vereinfachen können.



Weitere Informationen über Visual Studio Plugins finden Sie auf der MSDN Internetseite.

Basiskonzept

Wie bereits erwähnt startet das Tool eine Iteration über alle SPS-Baumelemente eines SPS-Projekts. Wenn das aktuelle Baumelement entweder ein Funktionsbaustein, eine Funktion oder ein Programm ist, dann wird der Deklarationstext des Baumelements bestimmt und mit einem VAR_CONSTANT Block erweitert.

Anmerkung bezüglich Ausführung und Debugging von Visual Studio Plugins

Visual Studio Plugins können sich in verschiedenen Ordner befinden, so dass Visual Studio diese erkennt. Diese Dokumentation setzt voraus, dass das Plugin nur für ein bestimmtes Benutzerkonto zugänglich ist. Dadurch wird das Plugin in einen Ordner innerhalb des Benutzerprofils abgelegt. Weitere Informationen finden Sie auf der MSDN Internetseite bezüglich der Add-In Registrierung.

Um das Plugin ohne Debugging auszuführen, kopieren Sie die kompilierte DLL und die .AddIn Datei in das Verzeichnis c:\Users\username\Documents\Visual Studio 201x\Addins\ und starten Visual Studio neu.

Wenn Sie das Plugin mit Debugging ausführen möchten, dann führen Sie einfach den Debugger aus, wenn Sie sich in der Plugin Lösung befinden. Wenn dies Probleme aufwirft, dann können Sie auch die .AddIn Datei in das Addins Verzeichnis (das oben erwähnt ist) kopieren und die Datei mit einem Texteditor Ihrer Wahl bearbeiten, um den <Assembly> Knoten in den Pfad der kompilierten Debug DLL zu verändern. Dann starten Sie eine Visual Studio Instanz und verbinden Ihren Debugger mit dieser Instanz/diesem Prozess.

Das Visual Studio Plugin wird im TwinCAT Menü im Visual Studio verfügbar gemacht.

7 Anhang

7.1 Sonstige Fehlercodes

Die folgenden Fehlercodes sind die HRESULT-Werte der Automation Interface Methoden, wie in der API-Referenz [▶ 117] beschrieben.

```
typedefenum TCSYSMANAGERHRESULTS
{
    [helpstring("ITcSmTreeItem not found!" (ITcSmTreeItem nicht gefunden!))]
    TSM_E_ITEMNOTFOUND         = 0x98510001,
    [helpstring("Invalid Item Type!" (Ungültiger Elementtyp!))]
    TSM_E_INVALIDITEMTYPE          = 0x98510002,
    [helpstring("Invalid SubItem Type!" (Ungültiger Unterelementtyp!))]
    TSM_E_INVALIDITEMSUBTYPE        = 0x98510003,
    [helpstring("Mismatching Items!" (Nicht übereinstimmende Elemente!))]
    TSM_E_MISMATCHINGITEMS          = 0x98510004,
    [helpstring("Corrupted Link" (Fehlerhafte Verknüpfung))]
    TSM_E_CORRUPTEDLINK         = 0x98510005,
    [helpstring("Item still referenced!" (Element immer noch referenziert!))]
    TSM_E_ITEMREFERENCED          = 0x98510006,
    [helpstring("Item already deleted!" (Element bereits gelöscht!))]
    TSM_E_ITEMDELETED         = 0x98510007,
    [helpstring("XML Error" (XML-Fehler))]
    TSM_E_XMLERROR             = 0x98510008,
} TCSYSMANAGERHRESULTS;
```

Beachten Sie, dass die folgende COM-Fehlerliste nur einen Ausschnitt darstellt und nicht vollständig ist:

Fehler	Beschreibung
RPC_E_CALL_REJECTED	Der COM-Server hat die Anfrage verworfen. Lesen Sie bitte unseren Artikel über die Implementierung eines eigenen Nachrichtenfilters [▶ 27] um diesen Fehler zu umgehen.
Rückgabewert von Methode CoRegisterMessageFilter() <> 0	Eine mögliche Ursache dieses Fehlers kann sein, dass der COM-Nachrichtenfilter auf ein MTA-Apartment angewandt wurde. Nachrichtenfilter können nur auf STA-Threads angewandt werden. Siehe Nachrichtenfilter [▶ 27] Artikel, um mehr über Nachrichtenfilter, einschließlich STA und MTA zu erfahren.
Fehlermeldung "A reference to Beckhoff TwinCAT XAE Base 2.0 Type Library could not be added" (Eine Referenz zu Beckhoff TwinCAT XAE Base 2.0 Typbibliothek konnte nicht hinzugefügt werden) beim Referenzieren der Typbibliothek in Visual Studio.	Die Typbibliothek ist nicht ordnungsgemäß registriert. Bitte registrieren Sie die Typbibliothek erneut, indem Sie den folgenden Befehl ausführen: *C:\Windows\Microsoft .NET\Framework\v4.0.xxxxx\regtlibv12.exe C:\TwinCAT3\Components\Base\TCatSysManager.tlb* xxxxxx ist die Version der derzeit installierten Version von .NET Framework 4.0.

Mehr Informationen:
www.beckhoff.de/te1000/

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de