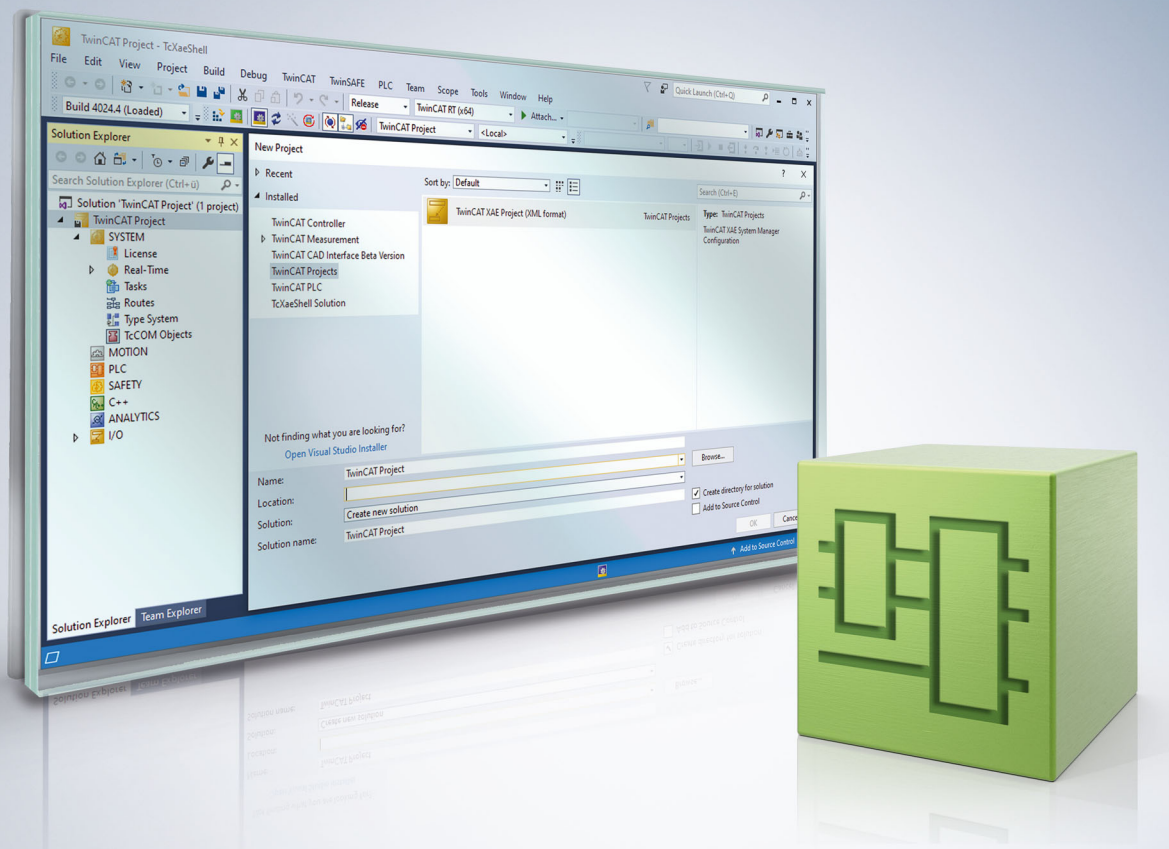
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC-Bibliothek: Tc3_Module



13.06.2023 | Version: 1.4.0

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 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

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.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

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

Die SPS Bibliothek Tc3_Module wird zur TcCOM-Kommunikation genutzt.

Systemvoraussetzung

Target System	WinXP, WES, Win7, WES7, WEC7 IPC or CX, (x86, x64, ARM)
Min. TwinCAT-Version	3.1.4020.0
Min. TwinCAT-Level	TC1200 TC3 PLC

3 Funktionsbausteine

Die SPS Bibliothek Tc3_Module bietet Funktionsbausteine, um über TcCOM von Modul zu Modul zu kommunizieren. Bei einem Modul kann es sich um eine TwinCAT-Systemkomponente handeln, um ein C++ Objekt, um ein Matlab Objekt oder auch um Objekte in der SPS.

3.1 TcBaseModuleRegistered

```
FUNCTION_BLOCK TcBaseModuleRegistered EXTENDS TcBaseModule
VAR
END_VAR
```

Beschreibung

Wenn von diesem Objekt geerbt wird, kann aus einem Funktionsbaustein ein TcCOM-Objekt erstellt werden. Das Objekt wird automatisch beim Objektserver registriert und in den OP-Zustand hochgefahren. Die eigene Objekt ID wird als Prozessabbild-Variable bereitgestellt. Methoden, welche zusätzlich implementiert werden und über dieses Objekt angeboten werden sollen, müssen einen Rückgabewert vom Typ HRESULT haben und threadsicher implementiert sein. Weitere Informationen finden Sie im Kapitel 'Multitask-Datenzugriffs-Synchronisation in der SPS'. Wie Sie dieses TcCOM-Objekt erstellen und im TwinCAT-System global nutzen können, wird in einem Beispiel [▶ 37] detailliert erläutert. Die Basisklasse TcBaseModule implementiert die Schnittstelle ITComObject, welche wiederum die Schnittstelle ITcUnknown erweitert.

ITComObject Interface

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt Funktionalitäten zur Verfügung bezüglich der Zustandsmaschine und Informationen vom/an das TwinCAT-System.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.1 TcAddRef

Die Methode TcAddRef() inkrementiert den Referenzzähler und gibt den neuen Wert zurück.

Rückgabewert

```
VAR_OUTPUT
    TcAddRef : UDINT;
END_VAR
```

Name	Typ	Beschreibung
TcAddRef	UDINT	Der resultierende Referenzzählwert wird zurückgegeben.

Eingänge

```
VAR_INPUT
    (*none*)
END_VAR
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.2 TcGetObjectId

Die Methode TcGetObjectId speichert die Objekt-ID mit Hilfe der gegebenen OTCID-Referenz.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjectId : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjectId	HRESULT	Informiert über Erfolg der OTCID-Abfrage.

Eingänge

```
VAR_INPUT
    objId : REFERENCE TO OTCID;
END_VAR
```

Name	Typ	Beschreibung
objId	REFERENCE TO OTCID	Referenz auf OTCID-Wert

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.3 TcGetObjectName

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjectName: DINT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjectName	DINT	Informiert über Erfolg der Namen-Abfrage.

Eingänge

```
VAR_INPUT
    objName : POINTER TO SINT;
    nameLen : UDINT;
END_VAR
```

Name	Typ	Beschreibung
objName	POINTER TO SINT	Der zu setzende Name
nameLen	UDINT	Die maximale, zu schreibende Länge des Namens

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.4 TcGetObjPara

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjPara : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjPara	HRESULT	Informiert über Erfolg der Objektparameterabfrage.

Eingänge

```
VAR_INPUT
    pid   : PTCID;
    nData : REFERENCE TO UDINT;
    pData : REFERENCE TO PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Typ	Beschreibung
pid	PTCID	Parameter-ID des Objektparameters
nData	REFERENCE TO UDINT	Maximale Länge der Daten
pData	REFERENCE TO PVOID	Zeiger auf die Daten
Pgp	PTCGP	Für zukünftige Erweiterung vorbehalten. NULL weitergeben.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.5 TcGetObjState

Die Methode TcGetObjState fragt den aktuellen Zustand des Objekts ab.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjState : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjState	HRESULT	Informiert über Erfolg der Zustandsabfrage.

Eingänge

```
VAR_INPUT
    pState : POINTER TO TCOM_STATE;
END_VAR
```

Name	Typ	Beschreibung
pState	POINTER TO TCOM_STATE	Zeiger auf den Zustand

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.6 TcQueryInterface

Die Methode fragt die Referenz an einer implementierten Schnittstelle über der ID ab.

Rückgabewert

```
VAR_OUTPUT
    TcQueryInterface : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcQueryInterface	HRESULT	Informiert über Erfolg der Schnittstellenabfrage. Wenn die verlangte Schnittstelle nicht verfügbar ist, gibt die Methode ADS_E_NOINTERFACE zurück.

Eingänge

```
VAR_INPUT
    iid    : REFERENCE TO IID;
    pipItf : POINTER TO PVOID;
END_VAR
```

Name	Typ	Beschreibung
iid	REFERENCE TO IID	Schnittstelle ID
pipItf	POINTER TO PVOID	Zeiger auf Schnittstellenzeiger. Wird gesetzt, wenn der verlangte Schnittstellentyp von der entsprechenden Instanz verfügbar ist.

Notwendige Freigabe der Schnittstellenzeiger

Sie müssen alle Referenzen explizit wieder freigeben. Wir empfehlen, FW_SafeRelease [▶ 26] zu verwenden, um nach der Verwendung eine Freigabe des Schnittstellenzeigers durchzuführen. Häufig wird die Freigabe der Referenzen im Destruktor des Objektes implementiert.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.7 TcRelease

Die Methode TcRelease() dekrementiert den Referenzzähler und gibt den neuen Wert zurück. Wenn der Referenzzähler 0 wird, löscht sich das Objekt selbst.

Rückgabewert

```
VAR_OUTPUT
    TcRelease : UDINT;
END_VAR
```

Name	Typ	Beschreibung
TcRelease	UDINT	Der resultierende Referenzzählwert wird zurückgegeben.

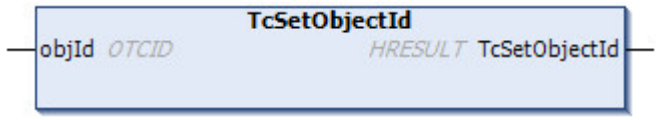
Eingänge

```
VAR_INPUT
    (*none*)
END_VAR
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.8 TcSetObjId



Die Methode TcSetObjectId setzt die Objekt-ID des Objekts auf die gegebene OTCID.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjId : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjId	HRESULT	Informiert über Erfolg der ID-Änderung.

Eingänge

```
VAR_INPUT
    objId : OTCID;
END_VAR
```

Name	Typ	Beschreibung
objId	OTCID	Die zu setzende OTCID

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.9 TcSetObjectName

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjectName : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjectName	HRESULT	Informiert über den Erfolg der Namen-Änderung.

Eingänge

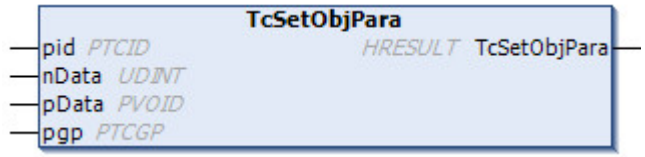
```
VAR_INPUT
    objName : POINTER TO SINT;
END_VAR
```

Name	Typ	Beschreibung
objName	POINTER TO SINT	Der zu setzende Name des Objekts

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.10 TcSetObjPara



Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjPara : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjPara	HRESULT	Informiert über Erfolg der Parameter-Änderung.

Eingänge

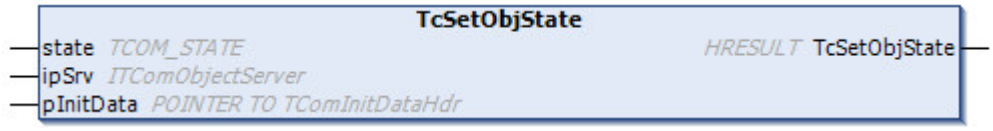
```
VAR_INPUT
    pid   : PTCID;
    nData : UDINT;
    pData : PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Typ	Beschreibung
pid	PTCID	Parameter-ID des Objektparameters
nData	UDINT	Maximale Länge der Daten
pData	PVOID	Zeiger auf die Daten
pgp	PTCGPkl	Für zukünftige Erweiterung vorbehalten, NULL weitergeben.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.11 TcSetObjState



Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjState : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjState	HRESULT	Informiert über Erfolg der Zustandsänderung.

Eingänge

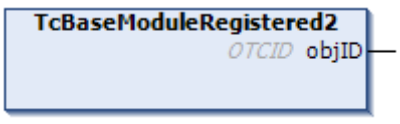
```
VAR_INPUT
    state     : TCOM_STATE;
    ipSrv     : ITComObjServer;
    pInitData : POINTER TO TComInitDataHdr;
END_VAR
```

Name	Typ	Beschreibung
state	TCOM_STATE	Stellt den neuen Zustand dar.
ipSrv	ITComObjServer	Objektbeschreibung
pInitData	POINTER TO TComInitDataHdr	Zeigt auf eine Liste von Parametern (optional).

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2 TcBaseModuleRegistered2



```
FUNCTION_BLOCK TcBaseModuleRegistered2 EXTENDS TcBaseModule
VAR_OUTPUT
    objID : OTCID;
END_VAR
```

Beschreibung

Wenn von diesem Objekt geerbt wird, kann aus einem Funktionsbaustein ein TcCOM-Objekt erstellt werden. Das Objekt wird automatisch beim Objektserver registriert und in den OP-Zustand hochgefahren. Die eigene Objekt ID wird am Ausgang bereitgestellt.

Methoden, welche zusätzlich implementiert werden und über dieses Objekt angeboten werden sollen, müssen einen Rückgabewert vom Typ HRESULT haben und threadsicher implementiert sein. Weitere Informationen hierzu finden Sie im Kapitel 'Multitask-Datenzugriffs-Synchronisation in der SPS'. Wie Sie dieses TcCOM-Objekt erstellen und im TwinCAT-System global nutzen können, wird in einem Beispiel zu TcBaseModuleRegistered [▶ 37] detailliert erläutert. Die Basisklasse TcBaseModule implementiert die Schnittstelle ITComObject, welche wiederum die Schnittstelle ITcUnknown erweitert.

ITComObject Interface

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt Funktionalitäten zur Verfügung bezüglich der Zustandsmaschine und Informationen vom/an das TwinCAT-System.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024	x86, x64, ARM	Tc3_Module >= v3.3.23.0

3.2.1 TcAddRef

Die Methode TcAddRef() inkrementiert den Referenzzähler und gibt den neuen Wert zurück.

Rückgabewert

```
VAR_OUTPUT
    TcAddRef : UDINT;
END_VAR
```

Name	Typ	Beschreibung
TcAddRef	UDINT	Der resultierende Referenzzählwert wird zurückgegeben.

Eingänge

```
VAR_INPUT
    (*none*)
END_VAR
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.2 TcGetObjectId

Die Methode TcGetObjectId speichert die Objekt-ID mit Hilfe der gegebenen OTCID-Referenz.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjectId : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjectId	HRESULT	Informiert über Erfolg der OTCID-Abfrage.

Eingänge

```
VAR_INPUT
    objId : REFERENCE TO OTCID;
END_VAR
```

Name	Typ	Beschreibung
objId	REFERENCE TO OTCID	Referenz auf OTCID-Wert

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.3 TcGetObjectName

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjectName: DINT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjectName	DINT	Informiert über Erfolg der Namen-Abfrage.

Eingänge

```
VAR_INPUT
    objName : POINTER TO SINT;
    nameLen : UDINT;
END_VAR
```

Name	Typ	Beschreibung
objName	POINTER TO SINT	Der zu setzende Name
nameLen	UDINT	Die maximale, zu schreibende Länge des Namens

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.4 TcGetObjPara

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjPara : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjPara	HRESULT	Informiert über Erfolg der Objektparameterabfrage.

Eingänge

```
VAR_INPUT
    pid   : PTCID;
    nData : REFERENCE TO UDINT;
    pData : REFERENCE TO PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Typ	Beschreibung
pid	PTCID	Parameter-ID des Objektparameters
nData	REFERENCE TO UDINT	Maximale Länge der Daten
pData	REFERENCE TO PVOID	Zeiger auf die Daten
Pgp	PTCGP	Für zukünftige Erweiterung vorbehalten. NULL weitergeben.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.5 TcGetObjState

Die Methode TcGetObjState fragt den aktuellen Zustand des Objekts ab.

Rückgabewert

```
VAR_OUTPUT
    TcGetObjState : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcGetObjState	HRESULT	Informiert über Erfolg der Zustandsabfrage.

Eingänge

```
VAR_INPUT
    pState : POINTER TO TCOM_STATE;
END_VAR
```

Name	Typ	Beschreibung
pState	POINTER TO TCOM_STATE	Zeiger auf den Zustand

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.6 TcQueryInterface

Die Methode fragt die Referenz an einer implementierten Schnittstelle über der ID ab.

Rückgabewert

```
VAR_OUTPUT
    TcQueryInterface : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcQueryInterface	HRESULT	Informiert über Erfolg der Schnittstellenabfrage. Wenn die verlangte Schnittstelle nicht verfügbar ist, gibt die Methode ADS_E_NOINTERFACE zurück.

Eingänge

```
VAR_INPUT
    iid    : REFERENCE TO IID;
    pipItf : POINTER TO PVOID;
END_VAR
```

Name	Typ	Beschreibung
iid	REFERENCE TO IID	Schnittstelle ID
pipItf	POINTER TO PVOID	Zeiger auf Schnittstellenzeiger. Wird gesetzt, wenn der verlangte Schnittstellentyp von der entsprechenden Instanz verfügbar ist.

Notwendige Freigabe der Schnittstellenzeiger

Sie müssen alle Referenzen explizit wieder freigeben. Wir empfehlen, FW_SafeRelease [▶ 26] zu verwenden, um nach der Verwendung eine Freigabe des Schnittstellenzeigers durchzuführen. Häufig wird die Freigabe der Referenzen im Destruktor des Objektes implementiert.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.7 TcRelease

Die Methode TcRelease() dekrementiert den Referenzzähler und gibt den neuen Wert zurück. Wenn der Referenzzähler 0 wird, löscht sich das Objekt selbst.

Rückgabewert

```
VAR_OUTPUT
    TcRelease : UDINT;
END_VAR
```

Name	Typ	Beschreibung
TcRelease	UDINT	Der resultierende Referenzzählwert wird zurückgegeben.

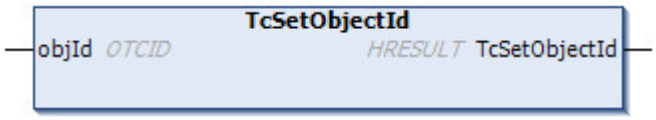
Eingänge

```
VAR_INPUT
    (*none*)
END_VAR
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.8 TcSetObjId



Die Methode TcSetObjectId setzt die Objekt-ID des Objekts auf die gegebene OTCID.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjId : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjId	HRESULT	Informiert über Erfolg der ID-Änderung.

Eingänge

```
VAR_INPUT
    objId : OTCID;
END_VAR
```

Name	Typ	Beschreibung
objId	OTCID	Die zu setzende OTCID

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.9 TcSetObjectName

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjectName : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjectName	HRESULT	Informiert über den Erfolg der Namen-Änderung.

Eingänge

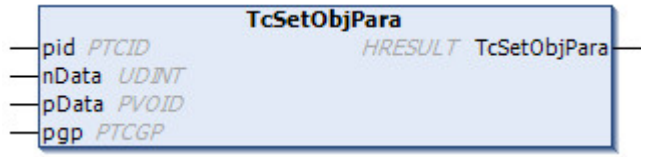
```
VAR_INPUT
    objName : POINTER TO SINT;
END_VAR
```

Name	Typ	Beschreibung
objName	POINTER TO SINT	Der zu setzende Name des Objekts

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.10 TcSetObjPara



Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjPara : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjPara	HRESULT	Informiert über Erfolg der Parameter-Änderung.

Eingänge

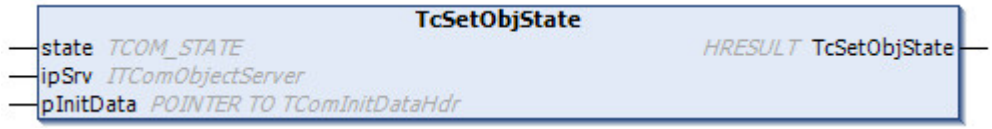
```
VAR_INPUT
    pid   : PTCID;
    nData : UDINT;
    pData : PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Typ	Beschreibung
pid	PTCID	Parameter-ID des Objektparameters
nData	UDINT	Maximale Länge der Daten
pData	PVOID	Zeiger auf die Daten
pgp	PTCGPkl	Für zukünftige Erweiterung vorbehalten, NULL weitergeben.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.11 TcSetObjState



Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

Rückgabewert

```
VAR_OUTPUT
    TcSetObjState : HRESULT;
END_VAR
```

Name	Typ	Beschreibung
TcSetObjState	HRESULT	Informiert über Erfolg der Zustandsänderung.

Eingänge

```
VAR_INPUT
    state     : TCOM_STATE;
    ipSrv     : ITComObjServer;
    pInitData : POINTER TO TComInitDataHdr;
END_VAR
```

Name	Typ	Beschreibung
state	TCOM_STATE	Stellt den neuen Zustand dar.
ipSrv	ITComObjServer	Objektbeschreibung
pInitData	POINTER TO TComInitDataHdr	Zeigt auf eine Liste von Parametern (optional).

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4 Funktionen

Die SPS Bibliothek Tc3_Module bietet Funktionen, um über TcCOM von Modul zu Modul zu kommunizieren. Bei einem Modul kann es sich um eine TwinCAT-Systemkomponente handeln, um ein C++ Objekt, um ein Matlab Objekt oder auch um Objekte in der SPS.

4.1 FW_ObjMgr_CreateAndInitInstance

Diese Funktion erzeugt eine Instanz der mittels Class-ID spezifizierten Klasse und liefert zugleich einen Schnittstellenzeiger auf dieses Objekt. Zudem können Objektname und Zustand, in den das Objekt versetzt werden soll, sowie optional auch Initialisierungsparameter angegeben werden.

Rückgabewert

```
FW_ObjMgr_CreateAndInitInstance : HRESULT;
```

Name	Typ	Beschreibung
FW_ObjMgr_CreateAndInitInstance	HRESULT	Liefert S_OK, wenn der Funktionsaufruf erfolgreich war.

Eingänge

```
VAR_INPUT
    clsId     : CLSID;
    iid       : IID;
    pipUnk    : POINTER TO ITcUnknown;
    objId     : UDINT;
    parentId  : UDINT;
    name      : REFERENCE TO STRING;
    state     : UDINT;
    pInitData : POINTER TO TComInitDataHdr;
END_VAR
```

Name	Typ	Beschreibung
clsId	CLSID	Spezifiziert die Klasse, von welcher ein Objekt angelegt werden soll.
iid	IID	Spezifiziert die Schnittstellen-ID, zu welcher ein Schnittstellenzeiger referenziert werden soll.
pipUnk	POINTER TO ITcUnknown	Liefert den Schnittstellenzeiger auf das erstellte Objekt.
objId	UDINT	Spezifiziert die Objekt-ID für das neu erstellte Objekt. Wird hier die globale Konstante OTCID_CreateNewId eingegeben, so wird intern eine neue Objekt-ID generiert.
parentId	UDINT	Objekt-ID des Elternobjektes (optional) Hier kann die Objekt-ID der SPS-Instanz angegeben werden, aus welcher diese Funktion aufgerufen wird. (TwinCAT_SystemInfoVarList._AppInfo.ObjId).
name	REFERENCE TO STRING	Spezifiziert den Objektnamen, welcher für das neu erstellte Objekt vergeben werden soll.
State	UDINT	Spezifiziert den Zustand, in den das neu erstellte Objekt versetzt werden soll. Typischerweise wird Operational (TCOM_STATE.TCOM_STATE_OP) angegeben.
pInitData	POINTER TO TComInitDataHdr	Zeiger auf Initialisierungsparameter (optional)

Notwendiges Löschen des Objektes

Ein erzeugtes Objekt muss explizit wieder gelöscht werden. Es gibt keinen Garbage-Collector wie in .Net. Es wird empfohlen, FW_ObjMgr_DeleteInstance [▶ 25] zu verwenden, um spätestens im Destruktor des Objektes, welches die Instanz angelegt hat, die erzeugte Instanz zu löschen.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.2 FW_ObjMgr_CreateInstance

Diese Funktion erzeugt eine Instanz der mittels Class-ID spezifizierten Klasse und liefert zugleich einen Schnittstellenzeiger auf dieses Objekt.

Rückgabewert

```
FW_ObjMgr_CreateInstance : HRESULT;
```

Name	Typ	Beschreibung
FW_ObjMgr_CreateInstance	HRESULT	Liefert S_OK, wenn der Funktionsaufruf erfolgreich war.

Eingänge

```
VAR_INPUT
    clsId  : CLSID;
    iid    : IID;
    pipUnk : POINTER TO ITcUnknown;
END_VAR
```

Name	Typ	Beschreibung
clsId	CLSID	Spezifiziert die Klasse, von welcher ein Objekt angelegt werden soll.
iid	IID	Spezifiziert die Schnittstellen-ID, zu welcher ein Schnittstellenzeiger referenziert werden soll.
pipUnk	POINTER TO ITcUnknown	Liefert den Schnittstellenzeiger auf das erstellte Objekt.

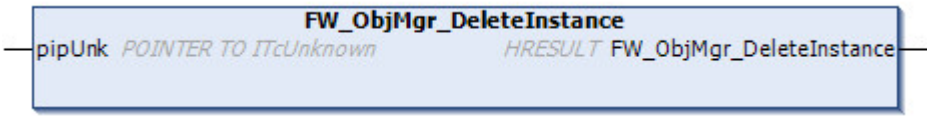
Notwendiges Löschen eines Objekts

Ein erzeugtes Objekt muss explizit wieder gelöscht werden. Es gibt keinen Garbage-Collector wie in .Net. Wir empfehlen, FW_ObjMgr_DeleteInstance [▶ 25] zu verwenden, um spätestens im Destruktor des Objektes, welches die Instanz angelegt hat, die erzeugte Instanz zu löschen.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.3 FW_ObjMgr_DeleteInstance



Diese Funktion versetzt das Objekt in den Init-Zustand. Daraufhin wird der Referenzzähler des Objektes dekrementiert, analog zu ITcUnknown.TcRelease(), und der Schnittstellenzeiger zugleich auf Null gesetzt.

Rückgabewert

```
FW_ObjMgr_DeleteInstance : HRESULT;
```

Name	Typ	Beschreibung
FW_ObjMgr_DeleteInstance	HRESULT	Liefert S_OK, wenn der Funktionsaufruf erfolgreich war.

Eingänge

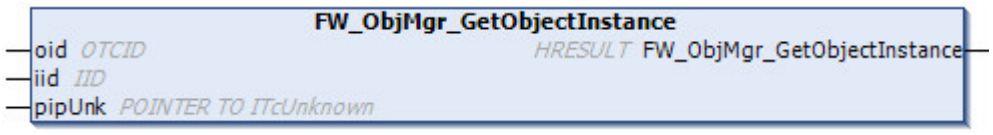
```
VAR_INPUT
    pipUnk : POINTER TO ITcUnknown;
END_VAR
```

Name	Typ	Beschreibung
pipUnk	POINTER TO ITcUnknown	Spezifiziert die Adresse des Schnittstellenzeigers auf das Objekt. Der Schnittstellenzeiger wird intern auf Nullpointer geprüft.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.4 FW_ObjMgr_GetObjectInstance



Diese Funktion liefert einen Schnittstellenzeiger auf eine mittels Objekt-ID spezifizierte Objektinstanz.

Rückgabewert

```
FW_ObjMgr_GetObjectInstance : HRESULT;
```

Name	Typ	Beschreibung
FW_ObjMgr_GetObjectInstance	HRESULT	Liefert S_OK, wenn der Funktionsaufruf erfolgreich war.

Eingänge

```
VAR_INPUT
    oid    : OTCID;  (*OID of object*)
    iid    : IID;  (*requested interface*)
    pipUnk : POINTER TO ITcUnknown;
END_VAR
```

Name	Typ	Beschreibung
oid	OTCID	Objekt-ID
iid	IID	Spezifiziert die Schnittstellen-ID, zu welcher ein Schnittstellenzeiger referenziert werden soll.
pipUnk	POINTER TO ITcUnknown	Liefert den Schnittstellenzeiger auf das erstellte Objekt.

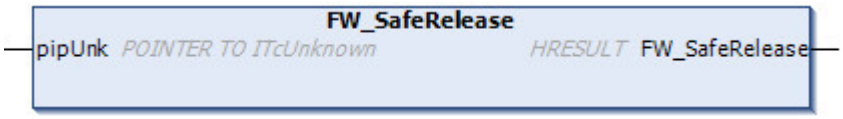
Notwendige Freigabe der Schnittstellenzeiger

Alle Referenzen müssen explizit wieder freigegeben werden. Es wird empfohlen, FW_SafeRelease [▶ 26] zu verwenden, um nach der Verwendung eine Freigabe des Schnittstellenzeigers durchzuführen. Häufig wird die Freigabe der Referenzen im Destruktor des Objektes implementiert.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.5 FW_SafeRelease



Diese Funktion dekrementiert den Referenzzähler des Objekts, analog zu ITcUnknown.TcRelease(), und setzt den Schnittstellenzeiger zugleich auf null.

Rückgabewert

```
FW_SafeRelease : HRESULT;
```

Name	Typ	Beschreibung
FW_SafeRelease	HRESULT	Liefert S_OK, wenn der Funktionsaufruf erfolgreich war.

Eingänge

```
VAR_INPUT
    pipUnk : POINTER TO ITcUknown;
END_VAR
```

Name	Typ	Beschreibung
pipUnk	POINTER TO ITcUknown	Spezifiziert die Adresse des Schnittstellenzeigers auf das Objekt. Der Schnittstellenzeiger wird intern auf Nullpointer geprüft.

Beispiel

Diese Funktion kann beispielweise im Destruktor des Objekts aufgerufen werden, welches einen Interfacepointer auf ein anderes Objekt hält.

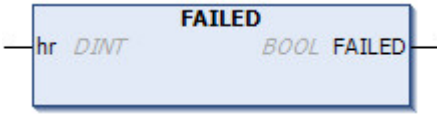
```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied
 afterwards (online change).
END_VAR
-------------------------------------------------------------------------
```

```
IF NOT bInCopyCode THEN // no online change
    FW_SafeRelease(ADR(ipItf));
END_IF
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.6 FAILED



Fehlercodes bzw. Statuscodes vom Typ HRESULT werden mit dieser Funktion auf Ungültigkeit geprüft.

Rückgabewert

```
FAILED : BOOL;
```

Name	Typ	Beschreibung
FAILED	BOOL	Liefert TRUE, wenn ein Fehler vorliegt.

Eingänge

```
VAR_INPUT
    hr : DINT;
END_VAR
```

Name	Typ	Beschreibung
hr	DINT	Angabe des zu überprüfenden Fehlercodes bzw. Statuscodes vom Typ HRESULT.

HRESULT

Der Typ HRESULT hat die Besonderheit, dass Fehler durch negative Werte repräsentiert werden. Warnungen oder Informationen können optional mittels positiver Werte ausgegeben werden.

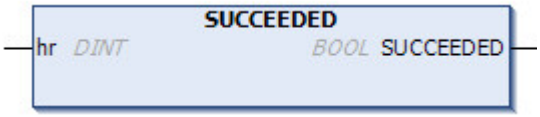
Deklaration	Fehlerbereich	Kein Fehler	Meldung/Info	Prüffunktionen

hrErrorCode : HRESULT;	<0	>=0	>0	IF SUCCEEDED(hrErrorCode) THEN
				...
				END_IF
				IF FAILED(hrErrorCode) THEN
				...
				END_IF

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.7 SUCCEEDED



Fehlercodes bzw. Statuscodes vom Typ HRESULT werden mit dieser Funktion auf Gültigkeit geprüft.

Rückgabewert

```
SUCCEEDED : BOOL;
```

Name	Typ	Beschreibung
SUCCEEDED	BOOL	Liefert TRUE, wenn kein Fehler.

Eingänge

```
VAR_INPUT
    hr : DINT;
END_VAR
```

Name	Typ	Beschreibung
hr	DINT	Angabe des zu überprüfenden Fehlercodes bzw. Statuscodes vom Typ HRESULT.

HRESULT

Der Typ HRESULT hat die Besonderheit, dass Fehler durch negative Werte repräsentiert werden. Warnungen oder Informationen können optional mittels positiver Werte ausgegeben werden.

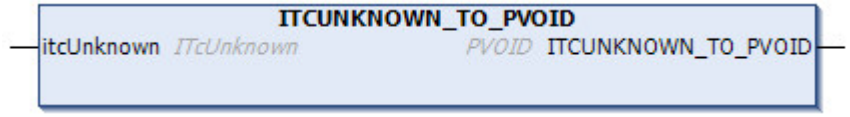
Deklaration	Fehlerbereich	Kein Fehler	Meldung/Info	Prüffunktionen

hrErrorCode : HRESULT;	<0	>=0	>0	IF SUCCEEDED(hrErrorCode) THEN
				...
				END_IF
				IF FAILED(hrErrorCode) THEN
				...
				END_IF

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.8 ITCUNKNOWN_TO_PVOID



Diese Konvertierungsfunktion konvertiert einen Interfacepointer vom Typ ITcUnknown zu einem Pointer to VOID.

Rückgabewert

```
ITCUNKNOWN_TO_PVOID : PVOID
```

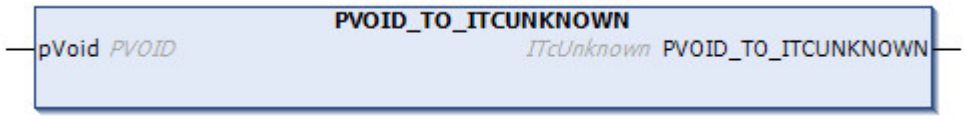
Eingänge

```
VAR_INPUT
    itcUnknown : ITcUknown;
END_VAR
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.9 PVOID_TO_ITCUNKNOWN



Diese Konvertierungsfunktion konvertiert einen Pointer to VOID zu einem Interfacepointer vom Typ ITcUnknown.

Rückgabewert

```
PVOID_TO_ITCUNKNOWN : ITcUnknown;
```

Eingänge

```
VAR_INPUT
   pVoid : PVOID;
END_VAR
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.10 GuidsEqual

Die Funktion GuidsEqual prüft zwei GUID-Objekte auf ihre Gleichheit zueinander.

Rückgabewert:

```
GuidsEqual : BOOL;
```

Name	Typ	Beschreibung
GuidsEqual	BOOL	Die Methode liefert TRUE, wenn beide Argumente gleich sind.

Eingänge

```
VAR_INPUT
    pGuidA : POINTER TO GUID;
    pGuidB : POINTER TO GUID;
END_VAR
```

Name	Typ	Beschreibung
pGuidA	POINTER TO GUID	Zeiger auf GUID-Objekt
pGuidB	POINTER TO GUID	Zeiger auf GUID-Objekt

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

5 Globale Konstanten

5.1 GVL

```
VAR_GLOBAL CONSTANT GVL
    S_OK                : HRESULT := 0;
    S_FALSE             : HRESULT := 1;
    S_PENDING           : HRESULT := 16#203;
    S_WATCHDOG_TIMEOUT  : HRESULT := 16#256;
    OTCID_CreateNewId   : OTCID := 16#FFFFFFFF;
    OTCID_FirstFreeId   : OTCID := 16#71010000;
    OTCID_LastFreeId    : OTCID := 16#710FFFFF;
    NULL : PVOID := 0;
END_VAR
```

Name	Typ	Wert	Verwendung	Bedeutung
S_OK	HRESULT	0		Diese Konstante kann genutzt werden, um eine fehlerfreie Abarbeitung in einem HRESULT Statuscode zu kennzeichnen.
S_FALSE	HRESULT	1		Diese Konstante signalisiert eine erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
S_PENDING	HRESULT	16#203		Diese Konstante signalisiert eine erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
S_WATCHDOG_TIMEOUT	HRESULT	16#256		Diese Konstante signalisiert eine erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat. Je nach Funktion wurde dabei die gewünschte Verarbeitung abgebrochen.
OTCID_CreateNewId	OTCID	16#FFFFFFFF	FW_ObjMgr_CreateAndInitInstance [▶ 23]	Diese Konstante wird genutzt, um eine neue Objekt-ID generieren zu lassen.
OTCID_FirstFreeId	OTCID	16#71010000		
OTCID_LastFreeId	OTCID	16#710FFFFF		
NULL	PVOID	0		NULL Zeiger

5.2 Global_Version

Alle Bibliotheken haben eine bestimmte Version. Diese Version ist u. a. im SPS-Bibliotheks-Repository zu sehen. Eine globale Konstante enthält die Information über die Bibliotheksversion:

```
VAR_GLOBAL CONSTANT
   stLibVersion_Tc3_Module : ST_LibVersion;
END_VAR
```

Name	Typ	Beschreibung
stLibVersion_Tc3_Module	ST_LibVersion	Versionsinformation der Tc3_Module-Bibliothek

Um zu sehen, ob die Version, die Sie haben, auch die Version ist, die Sie brauchen, benutzen Sie die Funktion F_CmpLibVersion (definiert in der Tc2_System SPS Bibliothek).

6 Fehlercodes

Die Rückgabewerte der Funktionen und Methoden werden vom Typ HRESULT ausgegeben.

HighWord des HRESULT	Gruppe der Fehlercodes
16#9811	Ads Fehlercodes

6.1 ADS Return Codes

Gruppierung der Fehlercodes: 0x000 [▶ 32]..., 0x500 [▶ 32]..., 0x700 [▶ 33]..., 0x1000 [▶ 35]...

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x9811 0000	ERR_NOERROR	Kein Fehler.
0x1	1	0x9811 0001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x9811 0002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x9811 0003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x9811 0004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x9811 0005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x9811 0006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x9811 0007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x9811 0008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x9811 0009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811 000A	ERR_NOIO	Kein IO.
0xB	11	0x9811 000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811 000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811 000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811 000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811 000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x9811 0010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x9811 0011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x9811 0012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x9811 0013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x9811 0014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x9811 0015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x9811 0016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x9811 0017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x9811 0018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x9811 0019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811 001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811 001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811 001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811 001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811 001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x9811 0500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x9811 0501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x9811 0502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x9811 0503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x9811 0504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x9811 0505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x9811 0506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x9811 0507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x9811 0508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x9811 0509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811 050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811 050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811 050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811 050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x9811 0700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x9811 0701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x9811 0702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x9811 0703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x9811 0704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x9811 0705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x9811 0706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x9811 0707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x9811 0708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x9811 0709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811 070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811 070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811 070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811 070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811 070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811 070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x9811 0710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x9811 0711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x9811 0712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x9811 0713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x9811 0714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x9811 0715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x9811 0716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x9811 0717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x9811 0718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x9811 0719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811 071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811 071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811 071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811 071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811 071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811 071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x9811 0720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x9811 0721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x9811 0722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x9811 0723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x9811 0724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x9811 0725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x9811 0726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x9811 0727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x9811 0728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x9811 0729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811 072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811 072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811 072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811 072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811 072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811 072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x9811 0730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x9811 0731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x9811 0732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x9811 0733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x9811 0734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.

Hex	Dec	HRESULT	Name	Beschreibung
0x735	1845	0x9811 0735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x9811 0736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x9811 0737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x9811 0738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x9811 0739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x9811 0740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x9811 0741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x9811 0742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x9811 0743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x9811 0744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x9811 0745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x9811 0746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x9811 0747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x9811 0748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x9811 0749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x9811 0750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x9811 0751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x9811 0752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x9811 0753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x9811 0754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x9811 0755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x9811 1000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x9811 1001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x9811 1002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x9811 1003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x9811 1004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x9811 1005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x9811 1006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x9811 1007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811 100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811 100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811 100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x9811 1010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x9811 1017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x9811 1018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x9811 1019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811 101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

7 Beispiele

Im Beispiel TcCOM_Sample01 [▶ 37] wird dargestellt, wie eine TcCOM-Kommunikation zwischen zwei SPS stattfinden kann. Dabei werden aus der einen SPS heraus Funktionalitäten der anderen SPS direkt aufgerufen.

Im Beispiel TcCOM_Sample02 [▶ 47] wird dargestellt, wie eine SPS-Applikation Funktionalitäten einer existierenden Instanz einer TwinCAT C++ Klasse nutzen kann. Eigene in C++ (oder Matlab) geschriebene Algorithmen lassen sich so leicht in der SPS verwenden.
Bei Verwendung eines existierenden TwinCAT C++ Moduls bedarf es zwar der TwinCAT C++ Lizenz auf dem Zielsystem, eine C++ Entwicklungsumgebung muss jedoch weder auf dem Zielsystem noch auf dem Entwicklungsrechner vorhanden sein.

Im Beispiel TcCOM_Sample03 [▶ 51] wird dargestellt, wie eine SPS-Applikation Funktionalitäten einer TwinCAT C++ Klasse nutzt, indem zugleich eine Instanz der C++ Klasse erzeugt wird. Dies kann im Vergleich zum vorherigen Sample eine erhöhte Flexibilität bieten.

Weitere Programmierbeispiele finden Sie in der Dokumentation zu TwinCAT 3 C++. Beispielsweise wird dort eine weitere Möglichkeit beschrieben, aus einem SPS-Programm heraus einen in C++ geschriebenen Algorithmus aufzurufen (Sample11). Im Unterschied zu TcCOM_Sample02 wird hier ein Wrapperbaustein programmiert, der jede Interfacemethode selbst implementiert. Deshalb ist diese Variante etwas aufwändiger. Falls Sie aber in der SPS-Applikation aus Anwendergründen zwingend auf Schnittstellenzeiger beim Aufruf der Funktionalitäten verzichten müssen, so bietet diese Variante eine Möglichkeit dazu.

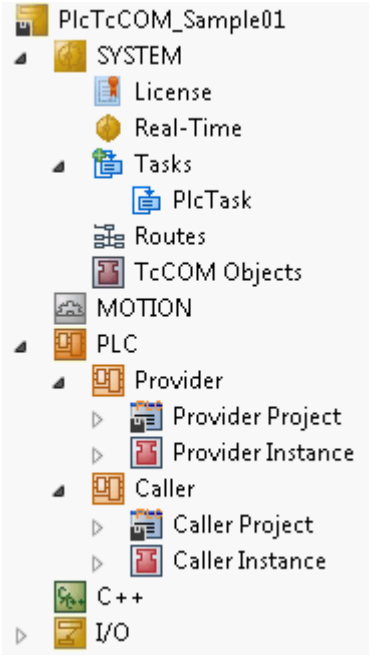
Ein anderes Beispiel in der Dokumentation zu TwinCAT 3 C++ zeigt, wie ein TwinCAT C++ Modul per TcCOM Interface eine Methode eines Funktionsbausteins der PLC aufruft (Sample13).

7.1 TcCOM_Sample01_PlcToPlc

Dieses Beispiel beschreibt eine TcCOM-Kommunikation zwischen zwei SPS.

Funktionalitäten, die von einem Funktionsbaustein in der ersten SPS (im Beispiel auch „Provider“ genannt) bereitgestellt werden, werden aus der zweiten SPS (im Beispiel auch „Caller“ genannt) heraus aufgerufen. Dazu muss der Funktionsbaustein oder dessen Programmcode nicht kopiert werden, sondern es wird direkt mit der Objektinstanz, die sich in der ersten SPS befindet, gearbeitet.

Beide SPS müssen sich in einer TwinCAT-Laufzeit befinden. Ein Funktionsbaustein bietet hierbei seine Methoden über eine global definierte Schnittstelle systemweit an und stellt selbst ein TcCOM-Objekt dar. Wie jedes TcCOM-Objekt wird auch ein solcher Funktionsbaustein zur Laufzeit im Knoten **TcCOM Objects** gelistet.



Die Vorgehensweise wird in folgenden Unterkapiteln erläutert:

- Erstellen eines FBs in der ersten SPS, der seine Funktionalität global bereitstellt [▶ 38]
- Erstellen eines FBs in der zweiten SPS, der als einfacher Proxy diese Funktionalität dort ebenfalls anbietet [▶ 42]
- Ausführung des Beispielprojektes [▶ 45]

Download des Beispiels: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc3_Module/Resources/2343046667.zip

Race Conditions bei Multi-Tasking (Multi-Threading)-Verwendung

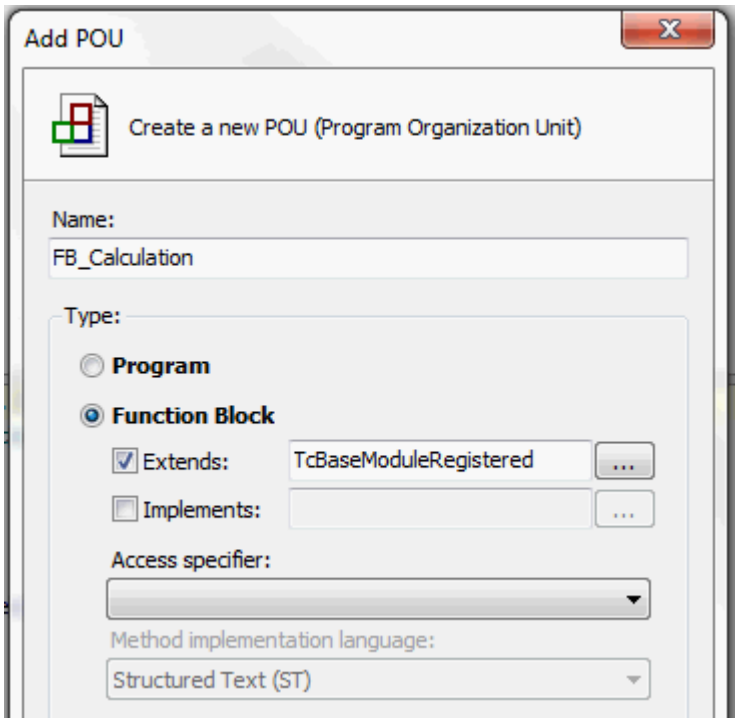
Der Funktionsbaustein, der seine Funktionalität global zur Verfügung stellt, wird in der ersten SPS instanziiert. Dort kann er wie jeder Funktionsbaustein verwendet werden. Wenn er außerdem aus einer anderen SPS (oder bspw. einem C++ Modul) verwendet wird, achten Sie darauf, dass die angebotenen Methoden thread-sicher sind, da die verschiedenen Aufrufe je nach Systemkonfiguration zeitgleich aus unterschiedlichen Taskkontexten erfolgen oder sich gegenseitig unterbrechen könnten. In diesem Fall dürfen die Methoden nicht auf Membervariablen des Funktionsbausteins oder globale Variablen der ersten SPS zugreifen. Sollte dies zwingend notwendig sein, beugen Sie einem zeitgleichen Zugriff vor. Beachten Sie die Funktion TestAndSet() aus der Tc2_System Bibliothek.

Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

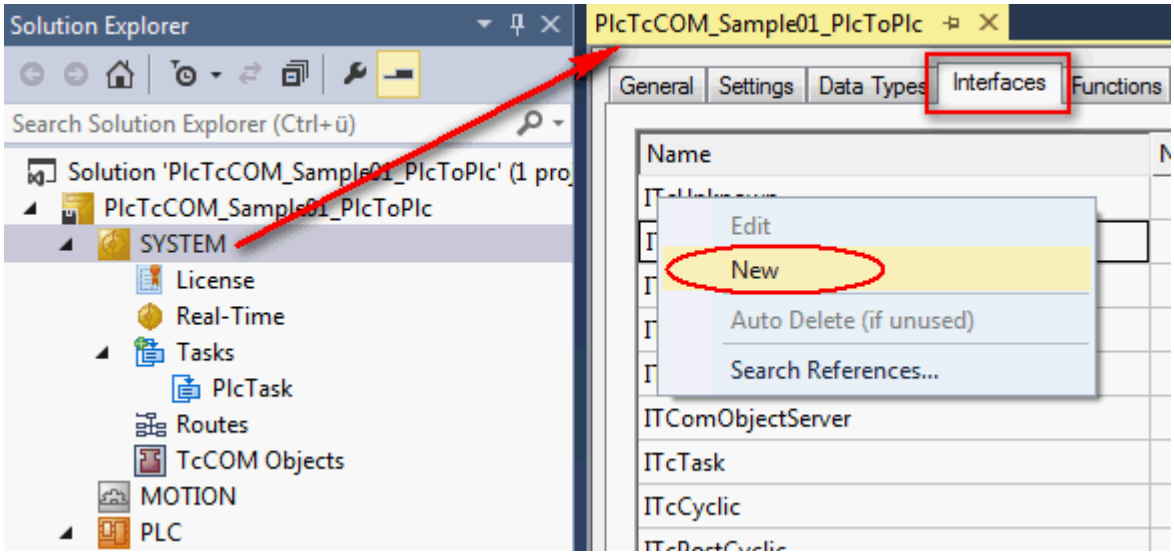
7.1.1 Erstellen eines FBs in der ersten SPS, welcher seine Funktionalität global bereitstellt

1. Legen Sie eine SPS an und erstellen Sie einen neuen Funktionsbaustein (FB) (hier: FB_Calculation). Leiten Sie den Funktionsbaustein von der Klasse TcBaseModuleRegistered [▶ 9] ab, damit eine Instanz dieses Funktionsbausteins nicht nur in der gleichen SPS verfügbar, sondern auch aus einer zweiten SPS heraus erreichbar ist.
   Hinweis: Alternativ können Sie auch einen FB in einer bestehenden SPS modifizieren.

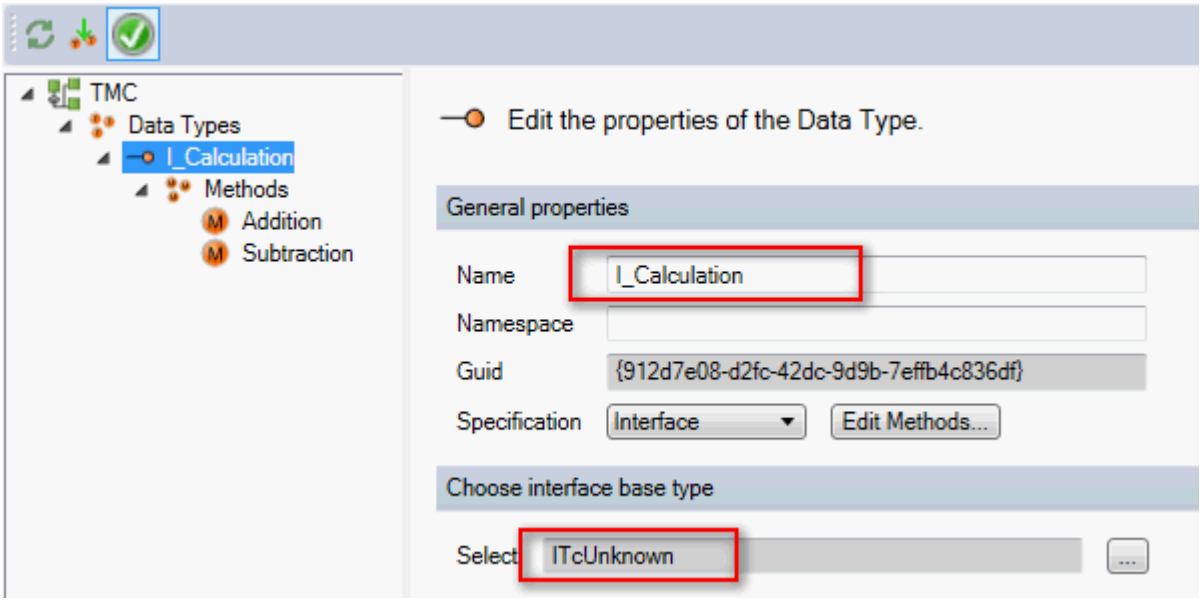


2. Der Funktionsbaustein muss seine Funktionalität mittels Methoden anbieten. Diese werden in einer globalen Schnittstelle definiert, deren Typ systemweit und programmiersprachenunabhängig bekannt ist. Um ein globales Interface anzulegen, öffnen Sie im Reiter „Interface“ der Systemeigenschaften das Kontextmenü und wählen Sie die Option „New“ aus.

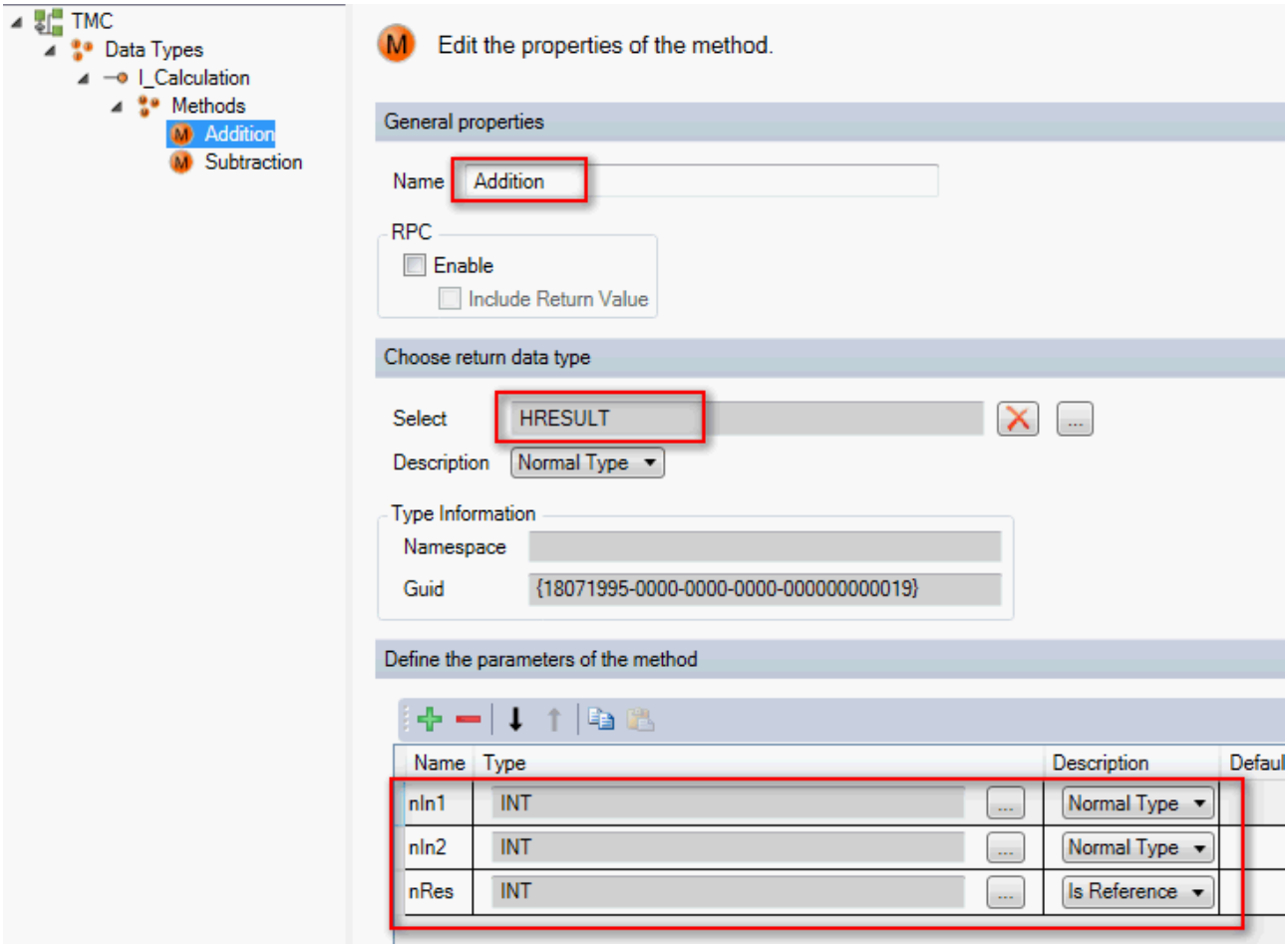
   ⇨ Es öffnet sich der TMC Editor, welcher Sie darin unterstützt ein globales Interface anzulegen.



3. Spezifizieren Sie den Namen (hier: I_Calculation) und fügen Sie die gewünschten Methoden an. Das Interface wird automatisch von ITcUnknown abgeleitet, um dem TwinCAT TcCOM-Modulkonzept gerecht zu werden.



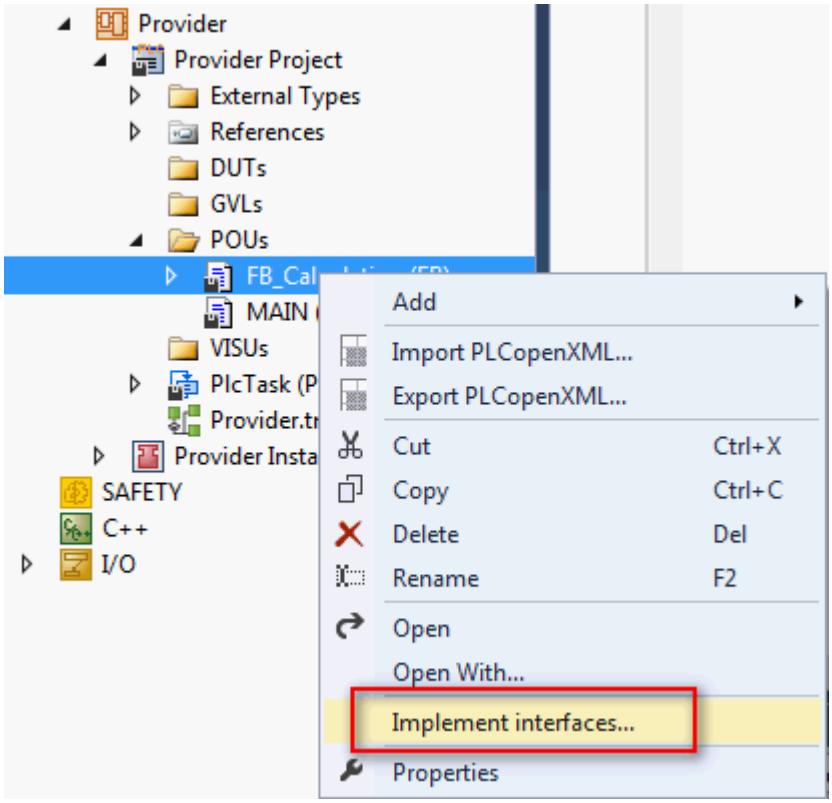
4. Geben Sie analog den Namen der Methoden an (hier: Addition() und Subtraction()) und wählen Sie als Rückgabedatentyp HRESULT. Dieser Rückgabetyp ist zwingend vorgeschrieben, wenn diese Art der TcCOM-Kommunikation implementiert werden soll.
5. Spezifizieren Sie zuletzt die Methodenparameter und schließen dann den TMC Editor.



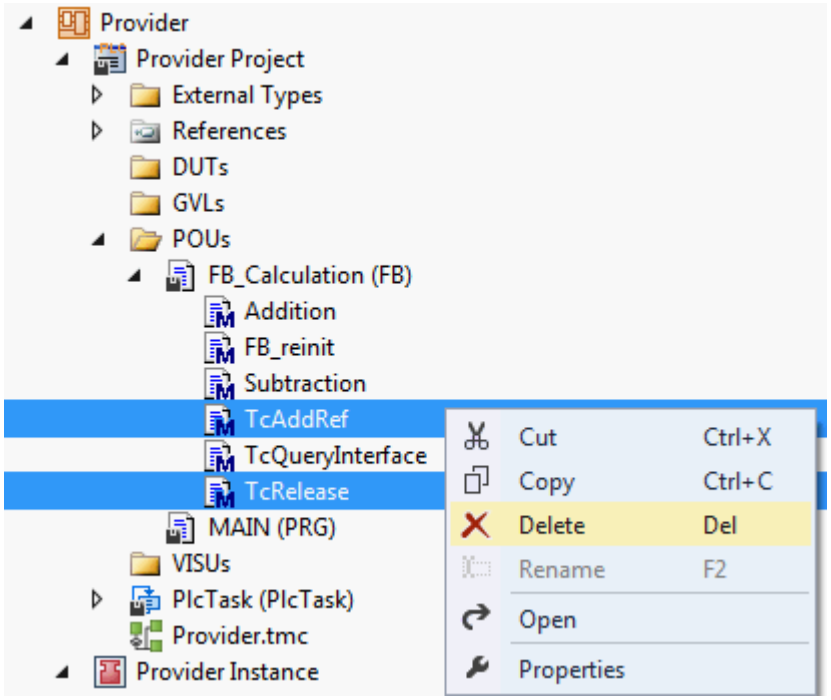
6. Implementieren Sie nun im Funktionsbaustein FB_Calculation die Schnittstelle I_Calculation und fügen Sie das Attribut c++_compatible an.

```
4   {attribute 'c++_compatible'}
5   FUNCTION_BLOCK FB_Calculation EXTENDS TcBaseModuleRegistered IMPLEMENTS I_Calculation
6
7   VAR
8   END_VAR
9
```

7. Wählen Sie im Kontextmenü des Funktionsbausteins die Option „Implement interfaces...“ aus, um die zu dieser Schnittstelle gehörenden Methoden zu erhalten.



8. Löschen Sie die beiden Methoden TcAddRef() und TcRelease(), weil hiervon die bereits vorhandene Implementierung der Basisklasse verwendet werden soll.



9. Legen Sie für den Funktionsbaustein FB_Calculation die Methode FB_reinit() an und rufen Sie die Basisimplementierung auf. Hierdurch wird gewährleistet, dass die Methode FB_reinit() der Basisklasse beim Online Change durchlaufen wird. Dies ist zwingend notwendig.

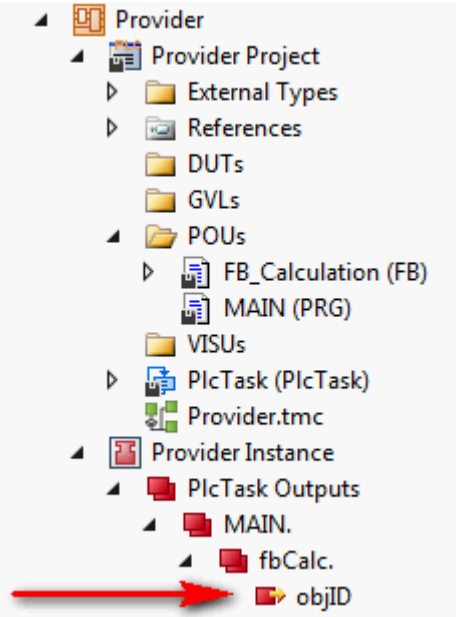
```
FB_Calculation.FB_reinit
1  METHOD FB_reinit : BOOL
2  VAR_INPUT
3  END_VAR
4

1  SUPER^.FB_reinit();
2
```

10. Implementieren Sie die Methode TcQueryInterface() der Schnittstelle ITcUnknown [▶ 74]. Über diese Methode ist es anderen TwinCAT Komponenten möglich, einen Schnittstellenzeiger auf eine Instanz dieses Funktionsbausteines zu erhalten und damit Methodenaufrufe zu tätigen. Der Aufruf von TcQueryInterface ist erfolgreich, wenn der Funktionsbaustein oder seine Basisklasse die mittels iid (Interface-ID) angefragte Schnittstelle bereitstellt. Für diesen Fall wird dem übergebenen Schnittstellenzeiger die Adresse auf den Funktionsbaustein typgewandelt zugewiesen und der Referenzzähler mittels TcAddRef() erhöht.
11. Füllen Sie die beiden Methoden Addition() und Subtraction() mit entsprechendem Code, um die Funktionalität zu erbringen: `nRes := nIn1 + nIn2` und `nRes := nIn1 - nIn2`
12. Fügen Sie eine oder mehrere Instanzen dieses Funktionsbausteins im Programmbaustein MAIN oder in einer globalen Variablenliste hinzu.

    ⇨ Die Implementierung in der ersten SPS ist vollständig.

```
MAIN*
1  PROGRAM MAIN
2  VAR
3      m : UDINT;
4
5      fbCalc : FB_Calculation('MAIN.fbCalc');
6  END_VAR
7
```

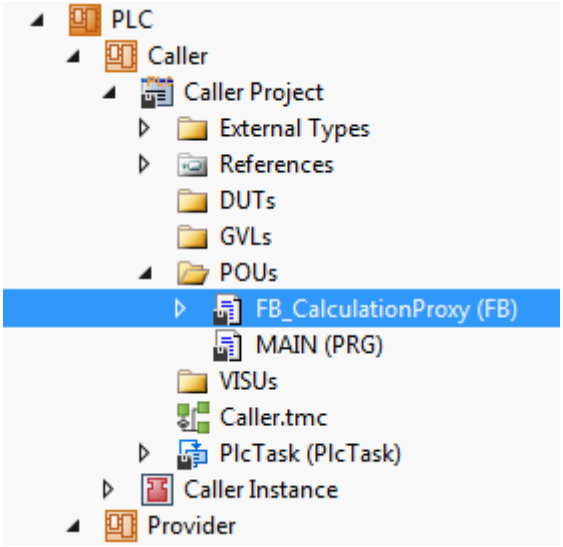
⇨ Nach dem Kompilieren der SPS ist im Prozessabbild die Objekt-ID des TcCOM-Objektes, welches die Instanz von FB_Calculation representiert, als Ausgang verfügbar.



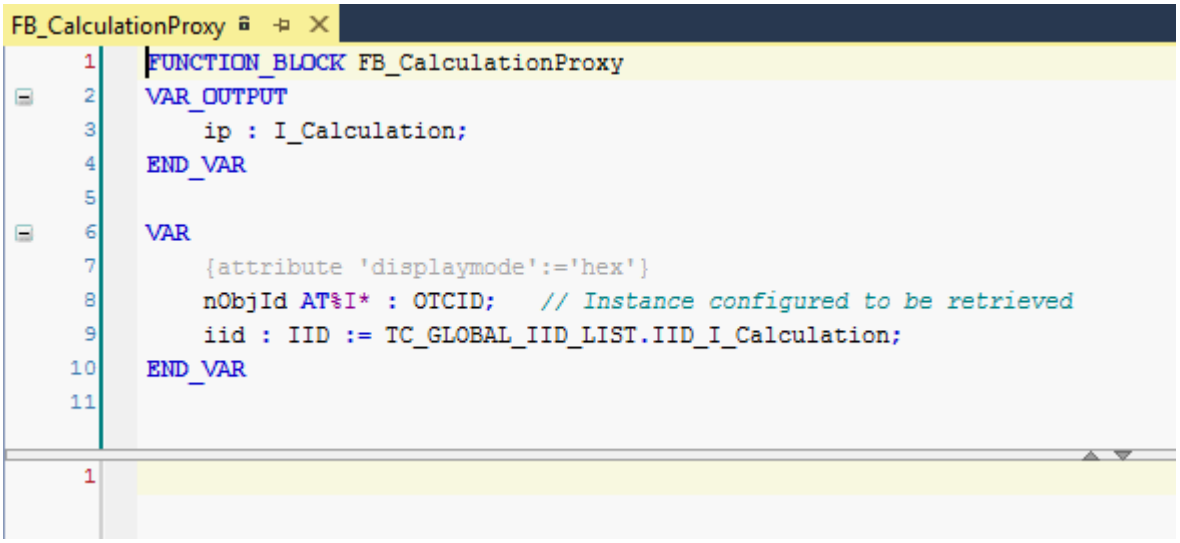
7.1.2 Erstellen eines FBs in der zweiten SPS, welcher als einfacher Proxy diese Funktionalität dort ebenfalls anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.

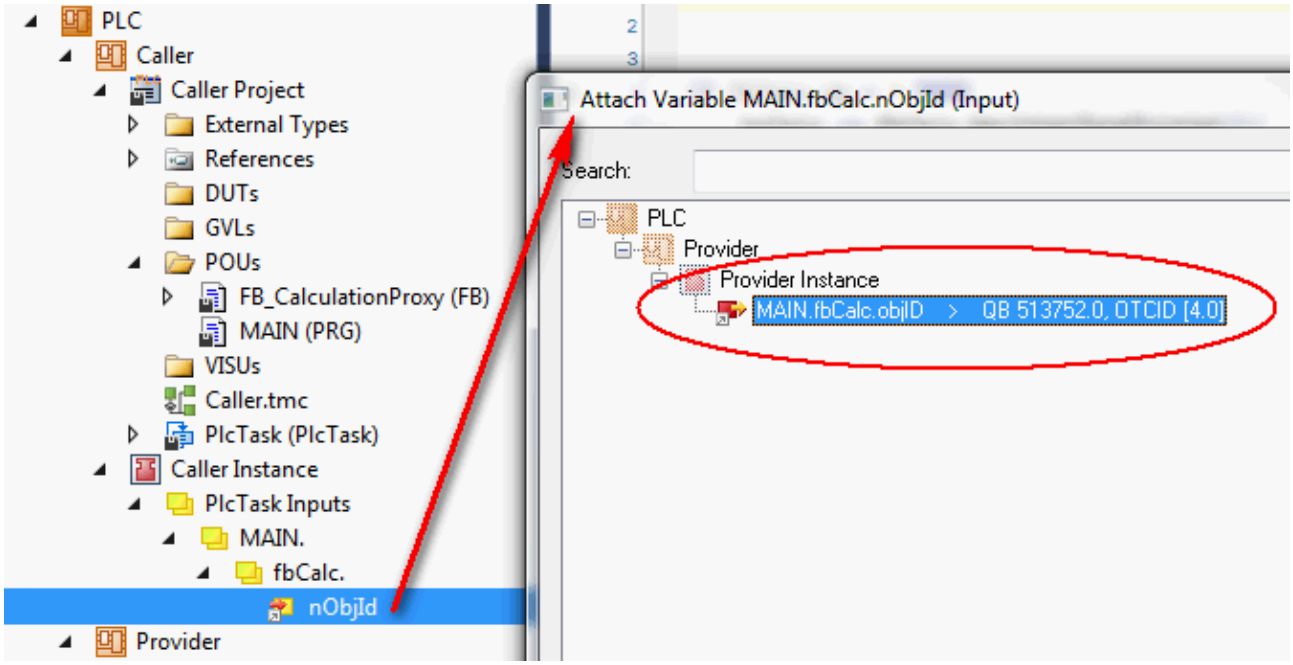
⇨ Dieser Proxy-Baustein soll die Funktionalität bereitstellen, welche in der ersten SPS programmiert wurde. Dies kann er über einen Schnittstellenzeiger vom Typ der globalen Schnittstelle I_Calculation.



2. Deklarieren Sie im Deklarationsteil des Funktionsbaustein als Ausgang einen Schnittstellenzeiger auf die globale Schnittstelle, welche später die Funktionalität nach außen bereitstellt.



3. Legen Sie zudem die Objekt-ID und die Schnittstellen-ID als lokale Membervariablen an.
Während die Schnittstellen-ID über eine globale Liste bereits verfügbar ist, wird die Objekt-ID über eine Verknüpfung im Prozessabbild zugewiesen.



4. Implementieren Sie den SPS Proxy-Baustein. Zuerst fügen Sie dem Baustein die Methode GetInterfacePointer() hinzu.
   Der Schnittstellenzeiger wird auf die spezifizierte Schnittstelle des spezifizierten TcCOM-Objektes mit Hilfe der Funktion FW_ObjMgr_GetObjectInstance() [▶ 25] geholt. Dies wird nur ausgeführt, wenn die Objekt-ID gültig und der Schnittstellenzeiger nicht bereits zugewiesen ist. Das Objekt selbst zählt einen Referenzzähler hoch.

FB_CalculationProxy.GetInterfacePointer

```
METHOD GetInterfacePointer : HRESULT
VAR
END_VAR

IF nObjID <> 0 THEN
    IF (ip = 0) THEN // only get interface pointer if it is not already existing
        GetInterfacePointer := FW_ObjMgr_GetObjectInstance(oid:=nObjID, iid:=iid, pipUnk:=ADR(ip));
    ELSE
        GetInterfacePointer := E_HRESULTAdsErr.EXISTS;
    END_IF
ELSE
    GetInterfacePointer := E_HRESULTAdsErr.INVALIDOBJID;
END_IF
```

5. Es ist zwingend notwendig die verwendete Referenz wieder freizugeben. Rufen Sie hierzu die Funktion FW_SafeRelease() im FB_exit Destruktor des Bausteines auf.

FB_CalculationProxy.FB_exit | FB_CalculationProxy.GetInterfacePointer

```
{attribute 'hide'}
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is c
END_VAR

IF NOT bInCopyCode THEN // if not online change
    FW_SafeRelease(ADR(ip));
END_IF
```

⇨ Damit ist die Implementierung des Proxy-Funktionsbausteines bereits abgeschlossen.

6. Instanziieren Sie in der Applikation den Proxy-Funktionsbaustein FB_CalculationProxy und rufen Sie dessen Methode GetInterfacePointer() auf, um einen gültigen Schnittstellenzeiger zu erhalten.
   Zum Aufruf der über die Schnittstelle bereitgestellten Methoden wird in der Applikation eine Instanz des Proxy-Bausteines deklariert. Die Aufrufe selbst finden alle über den als Ausgang des Bausteines definierten Schnittstellenzeiger statt. Wie bei Zeigern typisch muss eine Überprüfung auf Null vorausgehen. Daraufhin können die Methoden, auch mittels Intellisense, direkt aufgerufen werden.

MAIN*

```
1  PROGRAM MAIN
2  VAR
3      fbCalc : FB_CalculationProxy;
4      hrCalc : HRESULT;
5      a : INT := 10;
6      b : INT := 7;
7      nSum : INT; // a + b
8      nDiff : INT; // a - b
9  END_VAR
10
```

```
1  IF fbCalc.ip = 0 THEN
2      hrCalc := fbCalc.GetInterfacePointer();
3  END_IF
4  IF fbCalc.ip <> 0 THEN
5      hrCalc := fbCalc.ip.Addition(a,b,nSum);
6      hrCalc := fbCalc.ip.Subtraction(a,b,nDiff);
7  END_IF
8
```

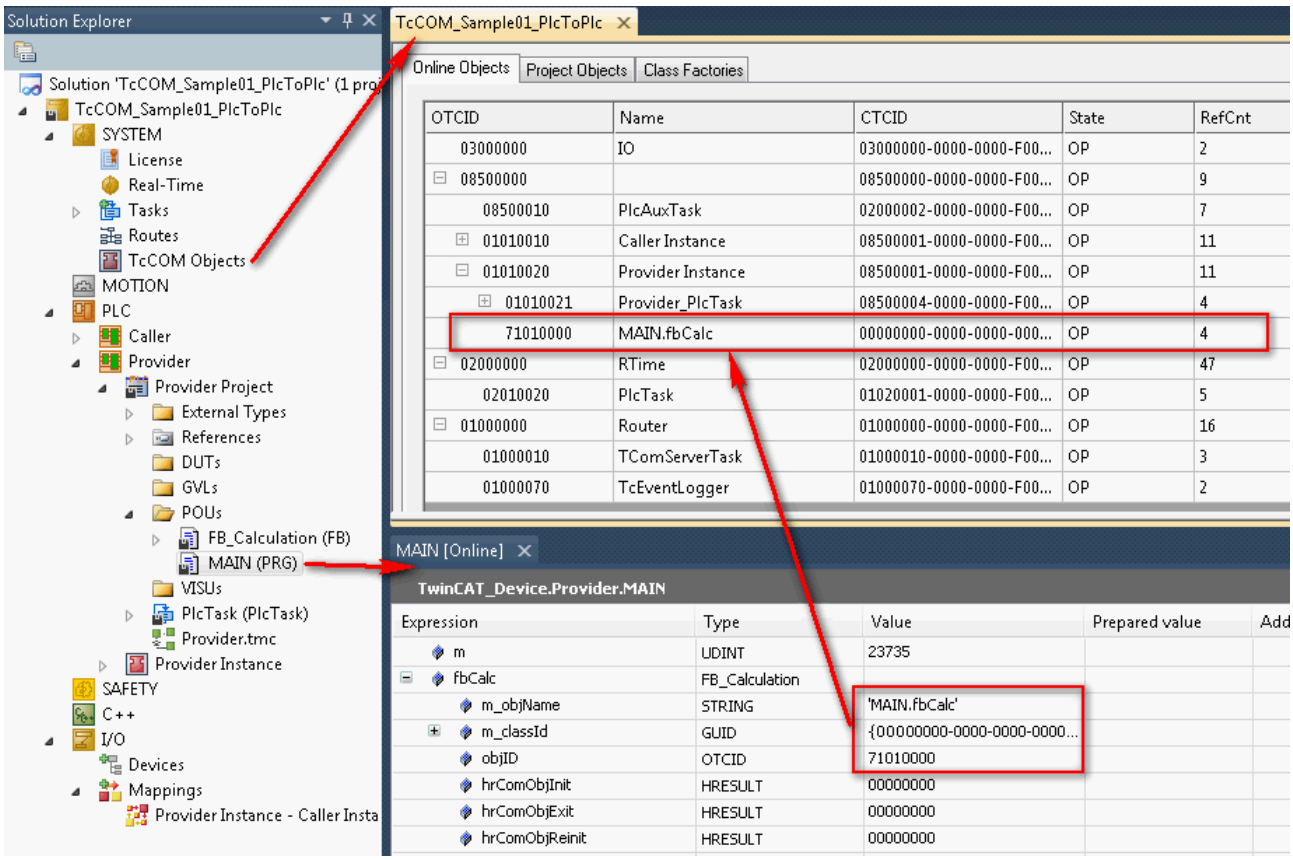
⇨ Das Beispiel ist bereit zum Test.

Reihenfolge irrelevant

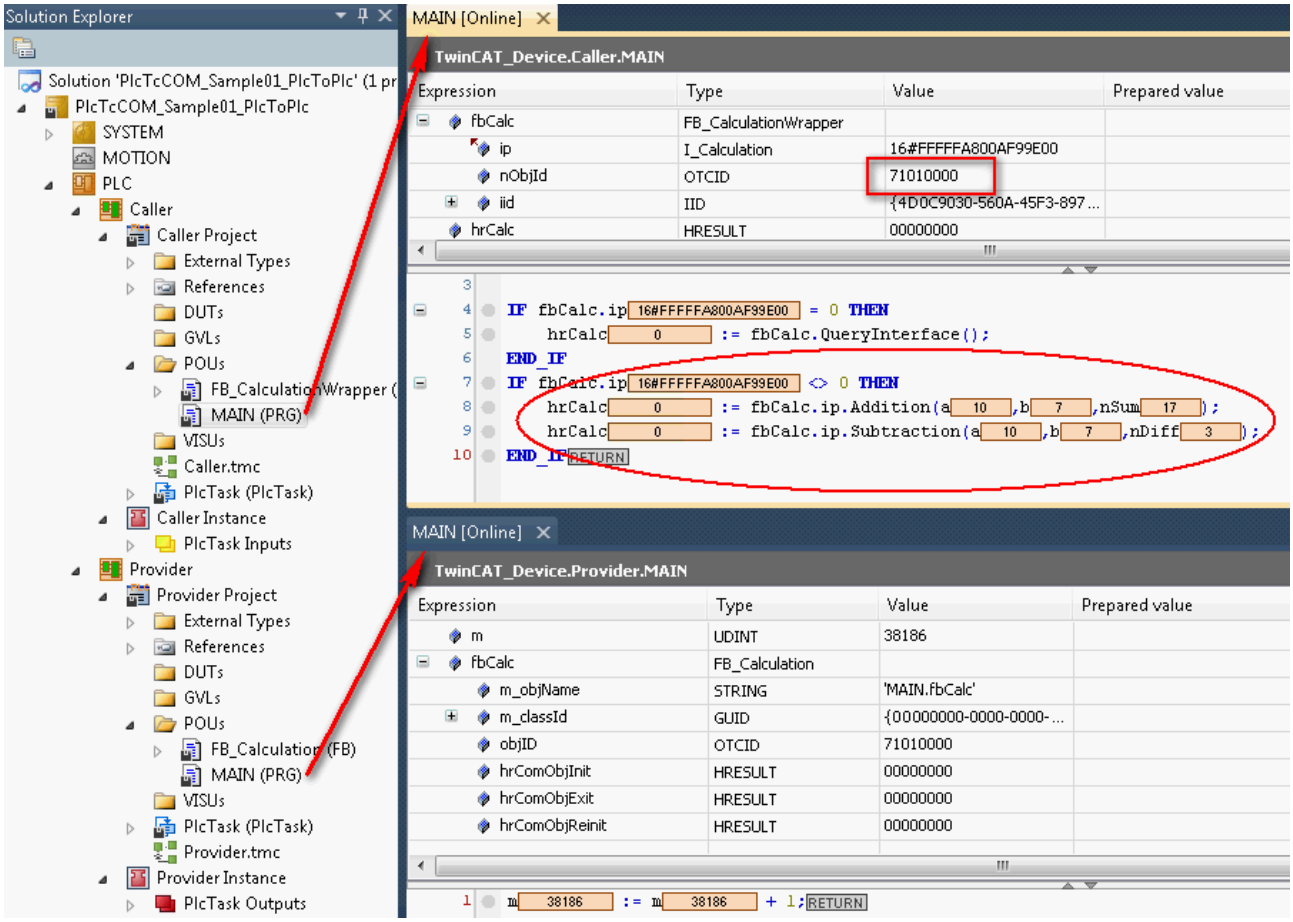
In welcher Reihenfolge die zwei SPS später starten, ist bei dieser Implementierung irrelevant.

7.1.3 Ausführung des Beispielprojektes

1. Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
2. Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten beider SPSn aus.
   ⇨ In der Onlineansicht der SPS-Applikation „Provider“ ist die generierte Objekt-ID des C++ Objektes im SPS Baustein FB_Calculation ersichtlich. Der Projektknoten „TcCOM Objekte“ führt das erzeugte Objekt mit dieser Objekt-ID und dem gewählten Namen in seiner Liste.



⇨ In der Onlineansicht der SPS Applikation „Caller“ hat der Proxy-Funktionsbaustein die gleiche Objekt-ID über das Prozessabbild zugewiesen bekommen. Der Schnittstellenzeiger hat einen gültigen Wert und die Methoden werden ausgeführt.



7.2 TcCOM_Sample02_PlcToCpp

Dieses Beispiel beschreibt eine TcCOM-Kommunikation zwischen SPS und C++. Hierbei nutzt eine SPS-Applikation Funktionalitäten einer existierenden Instanz einer TwinCAT C++ Klasse. Eigene in C++ geschriebene Algorithmen lassen sich so leicht in der SPS verwenden.
Bei Verwendung eines existierenden TwinCAT C++ Treibers bedarf es zwar der TwinCAT C++ Lizenz auf dem Zielsystem, eine C++ Entwicklungsumgebung muss jedoch weder auf dem Zielsystem noch auf dem Entwicklungsrechner vorhanden sein.

Ein bereits gebauter C++ Treiber stellt eine oder mehrere Klassen zur Verfügung, deren Schnittstellen in der TMC-Beschreibungsdatei hinterlegt und somit in der SPS bekannt sind.

Die Vorgehensweise wird in folgenden Unterkapiteln erläutert:

1. Instanziieren einer TwinCAT C++ Klasse als TwinCAT TcCOM Objekt [▶ 47]

2. Erstellen eines FBs in der SPS, welcher als einfacher Wrapper die Funktionalität des C++ Objektes anbietet [▶ 48]

3. Ausführung des Beispielprojektes [▶ 50]

Download des Beispiels: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc3_Module/Resources/2343048971.zip

Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64	Tc3_Module

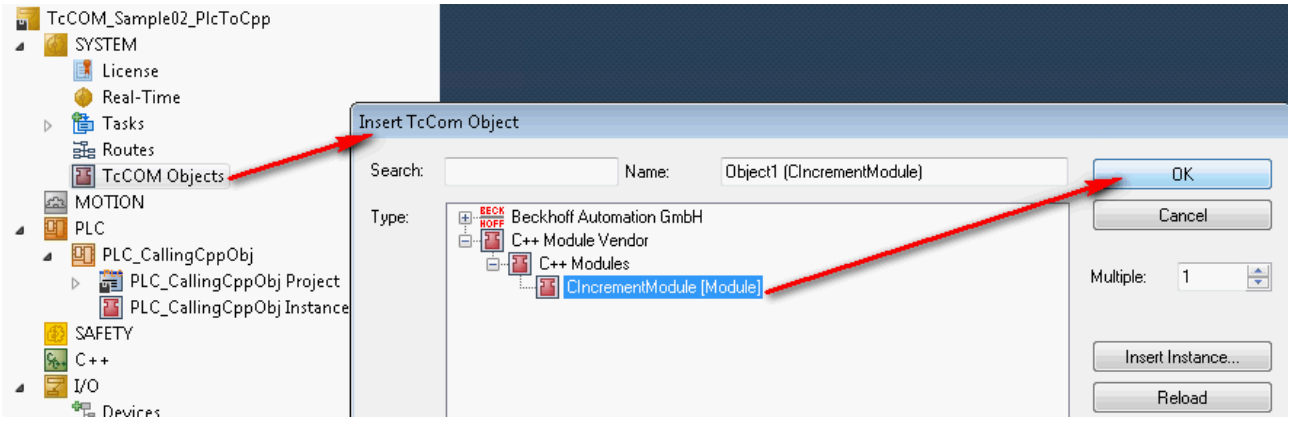
7.2.1 Instanziieren einer TwinCAT C++ Klasse als TwinCAT TcCOM Objekt

Der TwinCAT C++ Treiber muss auf dem Zielsystem zur Verfügung stehen. TwinCAT bietet hierfür ein Deployment, sodass die Komponenten nur passend auf dem Entwicklungsrechner abgelegt sein müssen.

Der existierende TwinCAT C++ Treiber sowie dessen TMC-Beschreibungsdatei(en) stehen als Treiberarchiv zur Verfügung. Dieses Archiv (IncrementerCpp.zip) wird in folgenden Ordner entpackt:
C:\TwinCAT\3.1\CustomConfig\Modules\IncrementerCpp

Das TwinCAT Deployment kopiert die Datei(en) später beim Aktivieren einer Konfiguration in folgenden Ordner auf dem Zielsystem:
C:\TwinCAT\3.1\Driver\AutoInstall

1. Öffnen Sie ein TwinCAT Projekt oder legen Sie ein neues Projekt an.
2. Fügen Sie in der Solution unter dem Knotenpunkt **TcCOM Objekte** eine Instanz der Klasse CIncrementModule hinzu.

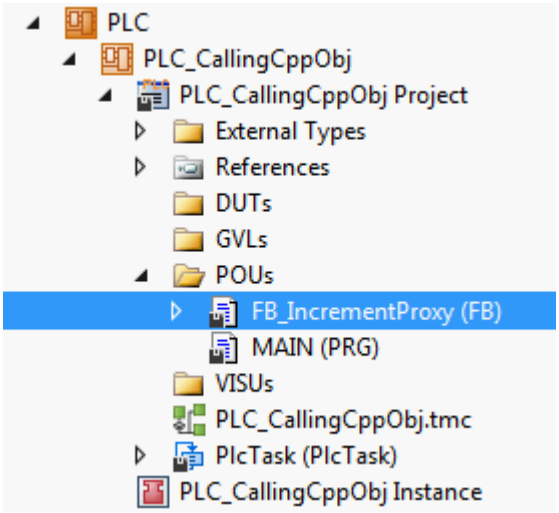


Erstellung des C++ Treibers

In der Dokumentation zu TwinCAT C++ wird ausführlich erläutert, wie C++ Treiber für TwinCAT erstellt werden.
Um das oben erwähnte Treiberarchiv zu erstellen, wird bei der Treibererstellung als letzter Schritt **Publish TwinCAT Modules** aus dem C++ Projektkontext gewählt.

7.2.2 Erstellen eines FBs in der SPS, der als einfacher Proxy die Funktionalität des C++ Objektes anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.
   Dieser Proxy-Baustein soll die Funktionalität bereitstellen, die in C++ programmiert wurde. Dies kann er über einen Schnittstellenzeiger, der von der C++ Klasse definiert wurde und aufgrund der TMC-Beschreibungsdatei in der SPS bekannt ist.

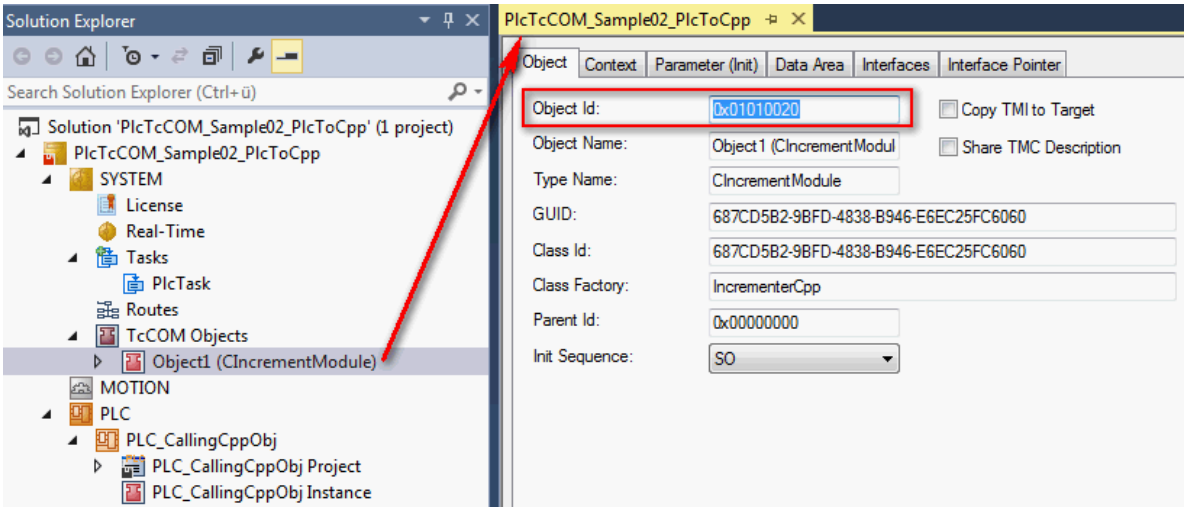


2. Deklarieren Sie im Deklarationsteil des Funktionsbausteins als Ausgang einen Schnittstellenzeiger auf die Schnittstelle, der später die Funktionalität nach außen bereitstellt.
3. Legen Sie die Objekt-ID und die Schnittstellen-ID als lokale Membervariablen an.
   Während die Schnittstellen-ID über eine globale Liste bereits verfügbar ist, wird die Objekt-ID über die TwinCAT-Symbol-Initialisierung zugewiesen. Das Attribut TcInitSymbol sorgt dafür, dass die Variable in einer Liste auftaucht, die der externen Symbolinitialisierung dient. Zugewiesen werden soll die Objekt-ID des angelegten C++ Objektes.

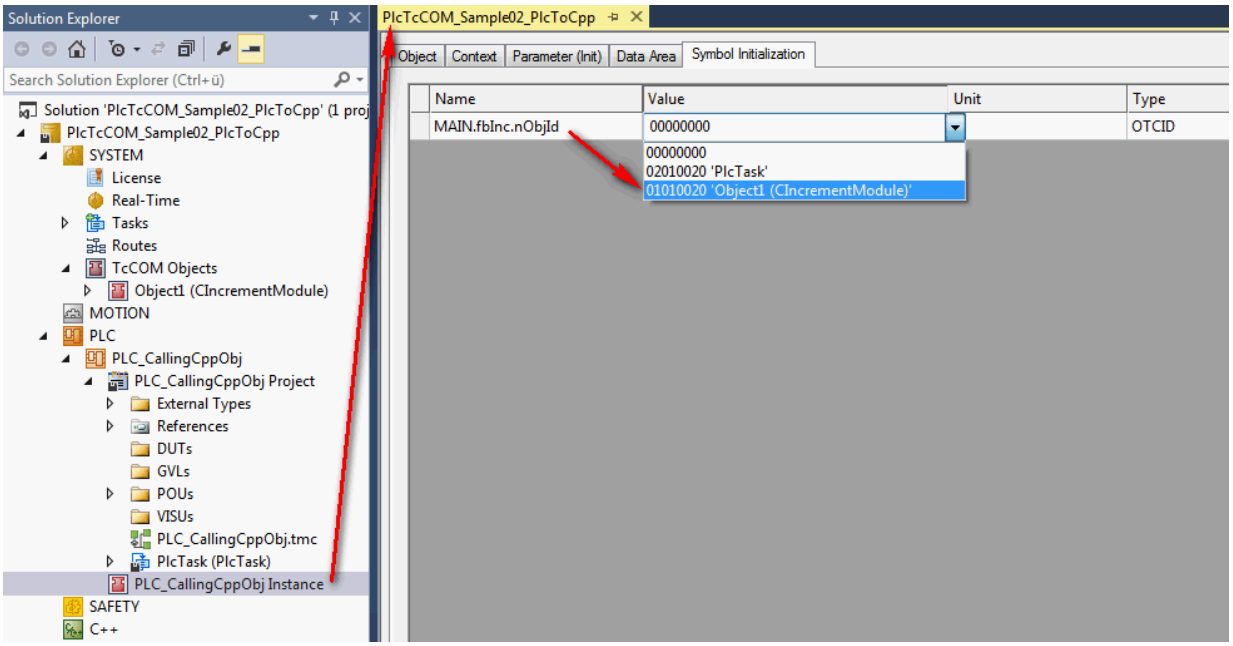
FB_IncrementProxy

```
1  FUNCTION_BLOCK FB_IncrementProxy
2  VAR_OUTPUT
3      ip : IIncrement;
4  END_VAR
5
6  VAR
7      {attribute 'TcInitSymbol'}
8      {attribute 'displaymode':='hex'}
9      nObjId : OTCID;     // Instance configured to be retrieved
10     iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
11     hrInit : HRESULT;
12 END_VAR
13
```

⇨ Die Objekt-ID wird bei Anwahl des Objektes unter dem Knoten **TcCOM-Objekte** angezeigt. Die Liste der Symbol-Initialisierungen befindet sich - sofern das Attribut TcInitSymbolverwendet wurde - im Knotenpunkt der SPS-Instanz im Reiter **Symbol Initialisierung**.



4. Weisen Sie hier dem Symbolnamen der Variablen per Dropdown eine vorhandene Objekt-ID zu. Beim Download der SPS wird dieser Wert zugewiesen, um so bereits vor der SPS-Laufzeit festgelegt zu sein. Neue Symbolinitialisierungen oder Änderungen werden demnach mit einem neuen Download der SPS eingespielt.



Die Übergabe der Objekt-ID könnte alternativ auch per Prozessabbildverknüpfung wie im ersten Beispiel implementiert werden (TcCOM_Sample01_PlcToPlc [▶ 37]).

5. Implementieren Sie den SPS Proxy-Baustein.
Zuerst wird dem Baustein die FB_init Konstruktormethode hinzugefügt. Für den Fall, dass es sich nicht um einen OnlineChange sondern um die Initialisierung des Bausteins handelt, wird der Schnittstellenzeiger auf die spezifizierte Schnittstelle des spezifizierten TcCOM-Objektes mit Hilfe der

Funktion FW_ObjMgr_GetObjectInstance() [▶ 25] geholt. Hierbei zählt das Objekt selbst einen Referenzzähler hoch.

```
FB_IncrementProxy.FB_init
{attribute 'hide'}
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
END_VAR

IF NOT bInCopyCode THEN // if not online change
    IF nObjID <> 0 THEN
        hrInit := FW_ObjMgr_GetObjectInstance(oid:=nObjID, iid:=iid, pipUnk:=ADR(ip));
    ELSE
        hrInit := E_HRESULTAdsErr.INVALIDOBJID;
    END_IF
END_IF
```

6. Es ist zwingend notwendig, die verwendete Referenz wieder freizugeben. Rufen Sie hierzu die Funktion FW_SafeRelease() [▶ 26] im FB_exit Destruktor des Bausteins auf.

```
FB_IncrementProxy.FB_exit | FB_IncrementProxy.FB_init
{attribute 'hide'}
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for
END_VAR

IF NOT bInCopyCode THEN // if not online change
    FW_SafeRelease(ADR(ip));
END_IF
```

⇨ Damit ist die Implementierung des Proxy-FunktionsBausteins bereits abgeschlossen.

7. Deklarieren Sie zum Aufruf der über die Schnittstelle bereitgestellten Methoden in der Applikation eine Instanz des Proxy-Bausteins.
   Die Aufrufe selbst finden alle über den als Ausgang des Bausteins definierten Schnittstellenzeiger statt. Wie bei Zeigern typisch, muss eine Überprüfung auf Null vorrausgehen. Daraufhin können die Methoden, auch mittels Intellisense, direkt aufgerufen werden.

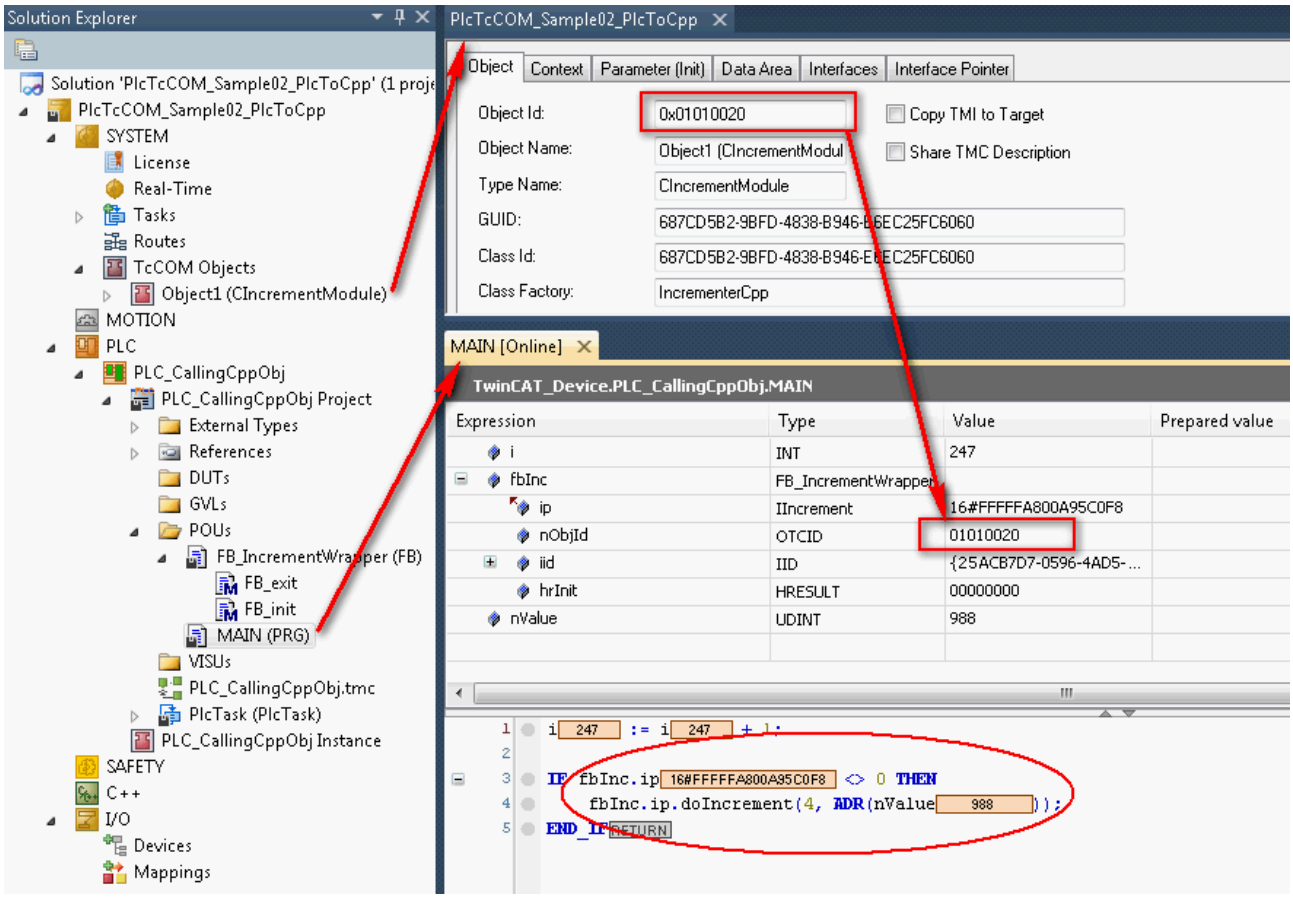
```
MAIN*
PROGRAM MAIN
VAR
    fbInc : FB_IncrementProxy;
    nValue : UDINT;
END_VAR

IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(4, ADR(nValue));
END_IF
```

⇨ Das Beispiel ist bereit zum Test.

7.2.3 Ausführung des Beispielprojektes

1. Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
2. Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten der SPS aus.

⇨ In der Onlineansicht der SPS-Applikation ist die zugewiesene Objekt-ID des C++ Objektes im SPS Proxy-Bausteins ersichtlich. Der Schnittstellenzeiger hat einen gültigen Wert und die Methode wird ausgeführt.



7.3 TcCOM_Sample03_PlcCreatesCpp

Dieses Beispiel beschreibt, ebenso wie Sample02, eine TcCOM-Kommunikation zwischen SPS und C++. Hierbei nutzt eine SPS Applikation Funktionalitäten einer TwinCAT C++ Klasse. Die benötigten Instanzen dieser C++ Klasse werden in diesem Beispiel von der SPS selbst angelegt. Eigene in C++ geschriebene Algorithmen lassen sich so leicht in der SPS verwenden.
Bei Verwendung eines existierenden TwinCAT C++ Treibers bedarf es zwar der TwinCAT C++ Lizenz auf dem Zielsystem, eine C++ Entwicklungsumgebung muss jedoch weder auf dem Zielsystem noch auf dem Entwicklunsgrechner vorhanden sein.

Ein bereits gebauter C++ Treiber stellt eine oder mehrere Klassen zur Verfügung, deren Schnittstellen in der TMC-Beschreibungsdatei hinterlegt und somit in der SPS bekannt sind.

Die Vorgehensweise wird in folgenden Unterkapiteln erläutert:

1. Bereitstellen eines TwinCAT C++ Treibers und seiner Klassen [▶ 52]

2. Erstellen eines FBs in der SPS, welcher das C++ Objekt anlegt und dessen Funktionalität anbietet [▶ 53]

3. Ausführung des Beispielprojektes [▶ 55]

Download des Beispiels: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc3_Module/Resources/2343051531.zip

Systemvoraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64	Tc3_Module

7.3.1 Bereitstellen eines TwinCAT C++ Treibers und seiner Klassen

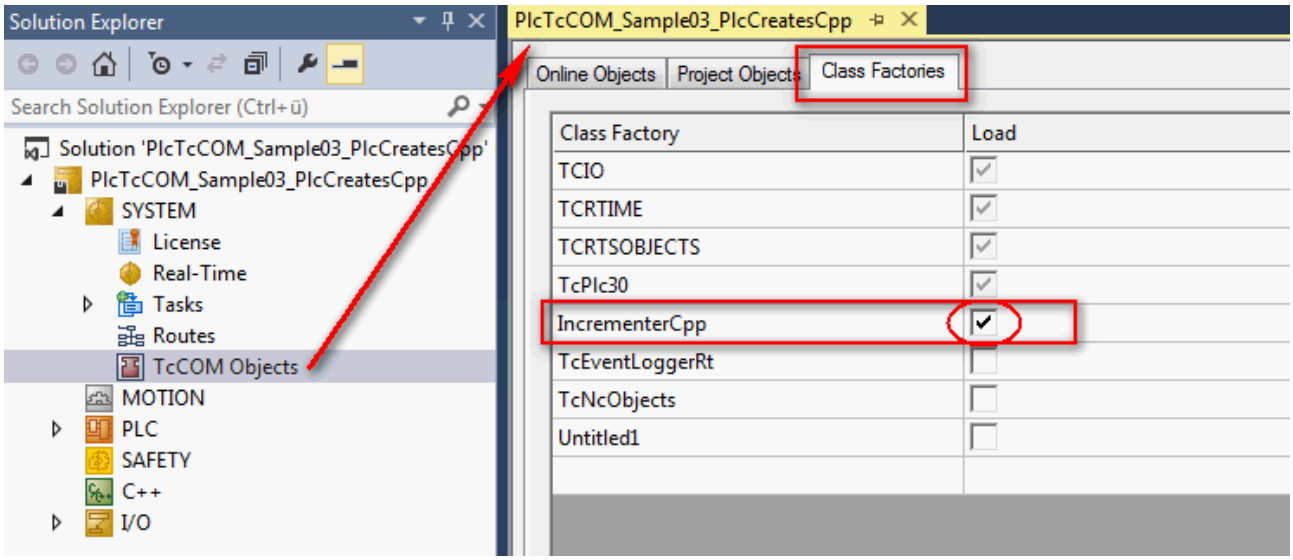
Der TwinCAT C++ Treiber muss auf dem Zielsystem zur Verfügung stehen. TwinCAT bietet hierfür ein Deployment, so dass die Komponenten nur passend auf dem Entwicklungsrechner abgelegt sein müssen.

Der existierende TwinCAT C++ Treiber sowie dessen TMC-Beschreibungsdatei(en) stehen als Treiberarchiv zur Verfügung. Dieses Archiv (IncrementerCpp.zip) wird in folgenden Ordner entpackt:
C:\TwinCAT\3.1\CustomConfig\Modules\IncrementerCpp

Das TwinCAT Deployment kopiert die Datei(en) später beim Aktivieren einer Konfiguration in folgenden Ordner auf dem Zielsystem:
C:\TwinCAT\3.1\Driver\AutoInstall

1. Öffnen Sie ein TwinCAT-Projekt oder legen Sie ein neues Projekt an.
2. Wählen Sie in der Solution unter dem Knotenpunkt **TcCom-Objekte** im Reiter **Class Factories** den benötigten C++ Treiber aus.

⇨ So wird sichergestellt, dass der Treiber beim Starten von TwinCAT auf dem Zielsystem geladen wird. Zudem sorgt diese Auswahl für das beschriebene Deployment.

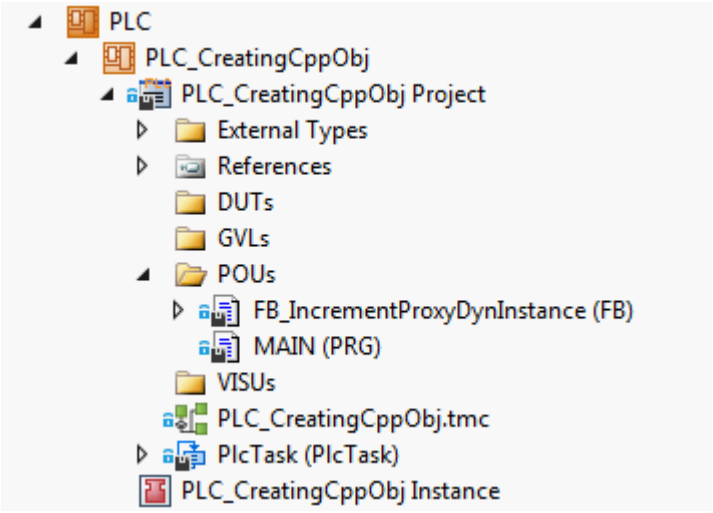


Erstellung des C++ Treibers

In der Dokumentation zu TwinCAT C++ wird ausführlich erläutert, wie C++ Treiber für TwinCAT erstellt werden.
Für Sample03 ist zu beachten, dass TwinCAT C++ Treiber, deren Klassen dynamisch instanziiert werden sollen, als „TwinCAT Module Class for RT Context“ definiert sein müssen. Der C++ Wizard bietet hierfür ein spezielles Template an.
Des Weiteren verwendet dieses Beispiel eine TwinCAT C++ Klasse, die ohne TcCOM-Initialisierungsdaten und ohne TcCOM-Parameter auskommt.

7.3.2 Erstellen eines FBs in der SPS, der das C++ Objekt anlegt und dessen Funktionalität anbietet

1. Legen Sie eine SPS an und fügen Sie dort einen neuen Funktionsbaustein an.
   Dieser Proxy-Baustein soll die Funktionalität bereitstellen, die in C++ programmiert wurde. Dies kann er über einen Schnittstellenpointer, der von der C++ Klasse definiert wurde und aufgrund der TMC-Beschreibungsdatei in der SPS bekannt ist.



2. Deklarieren Sie im Deklarationsteil des Funktionsbausteins als Ausgang einen Schnittstellenzeiger auf die Schnittstelle (IIncrement), die später die Funktionalität nach außen bereitstellt.

```
FB_IncrementProxyDynInstance*
1  FUNCTION_BLOCK FB_IncrementProxyDynInstance
2  VAR_OUTPUT
3      ip : IIncrement;
4  END_VAR
5
6  VAR
7      classId : CLSID := STRING_TO_GUID('687cd5b2-9bfd-4838-b946-e6ec25fc6060');
8      iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
9      hrInit : HRESULT;
10 END_VAR
11
```

3. Legen Sie die Klassen-ID und die Schnittstellen-ID als Membervariablen an.
   Während die Schnittstellen-ID über eine globale Liste bereits verfügbar ist, wird die Klassen-ID - sofern sie noch nicht bekannt sein sollte - über einen anderen Weg ermittelt. Wenn Sie die TMC-Beschreibungsdatei des zugehörigen C++ Treibers öffnen, finden Sie die entsprechende GUID dort vor.

```
13  <Modules>
14    <Module GUID="{687cd5b2-9bfd-4838-b946-e6ec25fc6060}" Group="C++">
15      <Name>CIncrementModule</Name>
16      <CLSID ClassFactory="IncrementerCpp">{687cd5b2-9bfd-4838-b946-e6ec25fc6060}</CLSID>
17      <Licenses>
18        <License>
```

4. Fügen Sie dem SPS Proxy-Baustein die FB_init Konstruktormethode hinzu.
   Für den Fall, dass es sich nicht um einen Online Change sondern um die Initialisierung des Bausteins handelt, wird ein neues TcCOM-Objekt (Klasseninstanz der spezifizierten Klasse) angelegt und der Schnittstellenzeiger auf die spezifizierte Schnittstelle geholt. Dabei wird der verwendeten Funktion FW_ObjMgr_CreateAndInitInstance() [▶ 23] auch der Name und der Zielzustand des TcCOM-Objektes mitgegeben. Diese zwei Parameter werden hier als Eingangsparameter der FB_init Methode deklariert, weshalb sie bei Instanziierung des Proxy-Bausteins anzugeben sind. Die zu instanziierende

TwinCAT C++ Klasse kommt ohne TcCOM-Initialisierungsdaten und ohne TcCOM-Parameter aus. Bei diesem Funktionsaufruf zählt das Objekt selbst einen Referenzzähler hoch.

FB_IncrementProxyDynInstance.FB_init

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online change)

    sObjName : STRING;      // object name to be set for this instance (optional)
    eObjState : TCOM_STATE; // target object state (usually TCOM_STATE.TCOM_STATE_OP)
END_VAR
```

```
IF NOT bInCopyCode THEN // if not online change
    objName := sObjName;
    hrInit := FW_ObjMgr_CreateAndInitInstance( clsId       := classId,
                                               iid         := iid,
                                               pipUnk      := ADR(ip),
                                               objId       := OTCID_CreateNewId,
                                               parentId    := TwinCAT_SystemInfoVarList._AppInfo.ObjId, //
                                               name        := sObjName,
                                               state       := eObjState,
                                               pInitData   := 0 );
END_IF
```

5. Es ist zwingend notwendig, die verwendete Referenz wieder freizugeben und das Objekt zu löschen, sofern es nicht mehr verwendet wird. Rufen Sie hierzu die Funktion FW_ObjMgr_DeleteInstance() [▶ 25] im FB_exit Destruktor des Bausteins auf.

FB_IncrementProxyDynInstance.FB_exit | FB_IncrementProxyDynInstance.FB_init

```
{attribute 'hide'}
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instan
END_VAR
```

```
IF NOT bInCopyCode THEN // if not online change
    FW_ObjMgr_DeleteInstance(ADR(ip));
END_IF
```

⇨ Damit ist die Implementierung des Proxy-Funktionsbausteins bereits abgeschlossen.

6. Deklarieren Sie zum Aufruf der über die Schnittstelle bereitgestellten Methoden in der Applikation eine Instanz des Proxy-Bausteins. Die Aufrufe selbst finden alle über den als Ausgang des Bausteins definierten Schnittstellenzeiger statt. Wie bei Zeigern typisch, muss eine Überprüfung auf Null vorrausgehen. Daraufhin können die Methoden, auch mittels Intellisense, direkt aufgerufen werden.

MAIN*

```
PROGRAM MAIN
VAR
    fbInc : FB_IncrementProxyDynInstance(   sObjName:='CIncrementModule:fbInc',
                                            eObjState:=TCOM_STATE.TCOM_STATE_OP);
    nValue : UDINT;
END_VAR
```

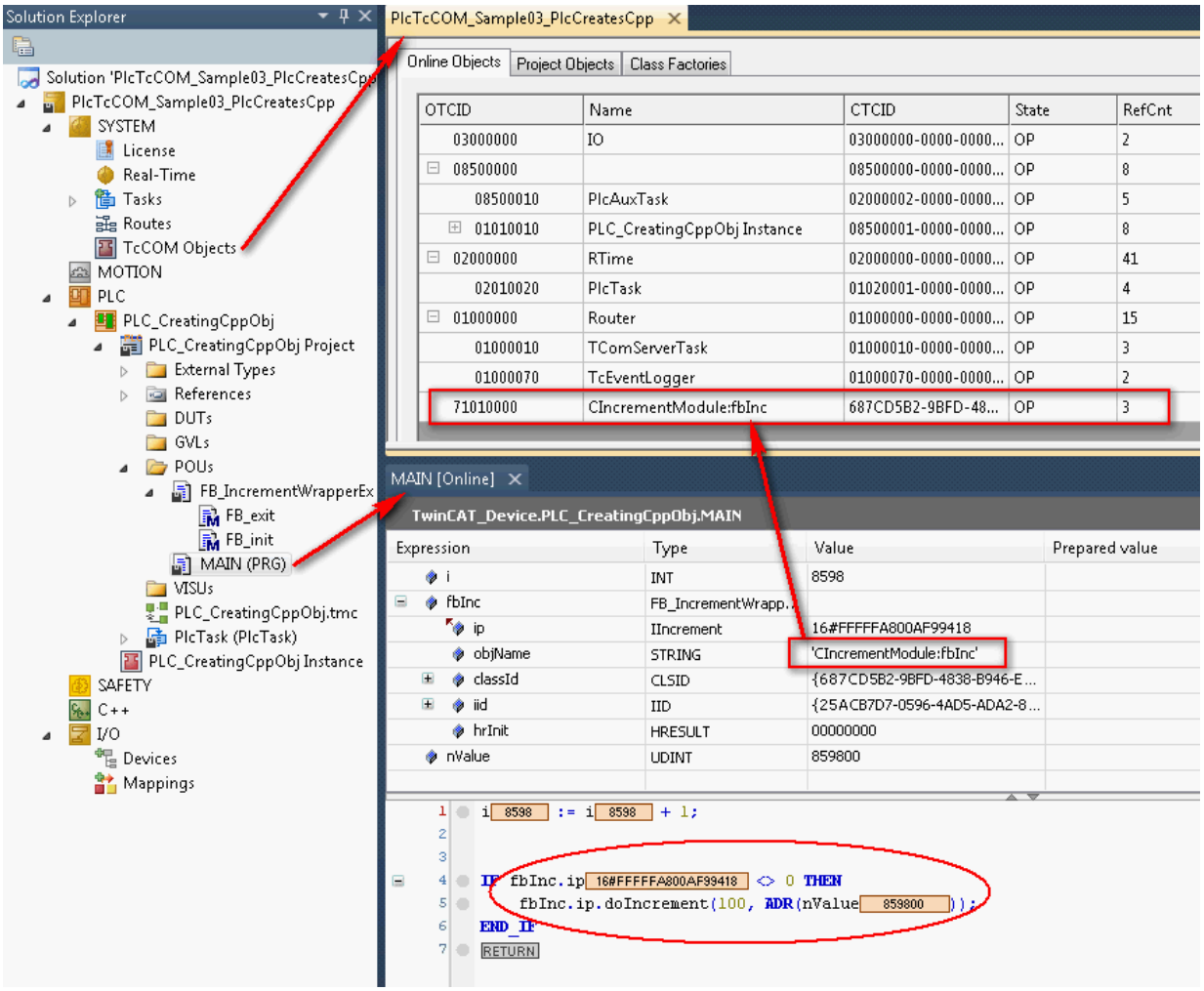
```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(100, ADR(nValue));
END_IF
```

⇨ Das Beispiel ist bereit zum Test.

7.3.3 Ausführung des Beispielprojektes

1. Wählen Sie das Zielsystem aus und kompilieren Sie das Projekt.
2. Aktivieren Sie die TwinCAT-Konfiguration und führen Sie ein Log-In sowie ein Starten der SPS aus.

⇨ In der Onlineansicht der SPS-Applikation ist der gewünschte TcCOM-Objektname im SPS-Proxy-Baustein ersichtlich. Der Projektknoten **TcCOM-Objekte** führt das erzeugte Objekt mit der generierten ID und dem gewünschten Namen in seiner Liste. Der Schnittstellenzeiger hat einen gültigen Wert und die Methode wird ausgeführt.



7.4 TcCOM_Sample13_CppToPlc

Beschreibung

Dieses Beispiel stellt die Kommunikation von einem C++ Modul zu einem Funktionsbaustein einer SPS mittels Methodenaufruf dar. Hierfür wird ein TcCOM-Interface definiert, welches von der SPS angeboten wird und von dem C++ Modul genutzt wird.

Die SPS-Seite als Provider entspricht dabei dem entsprechenden Projekt des Beispiels TcCOM Sample 01 [▶ 37], wo eine SPS nach SPS Kommunikation betrachtet wird. Hier wird nun ein Caller in C++ bereitgestellt, der das gleiche Interface nutzt.

Die Erläuterung des Beispiels finden Sie im Unterkapitel „Implementierung des Beispiels“.

Download des Beispiels: TcCOM_Sample13_CppToPlc.zip

Systemvoraussetzungen

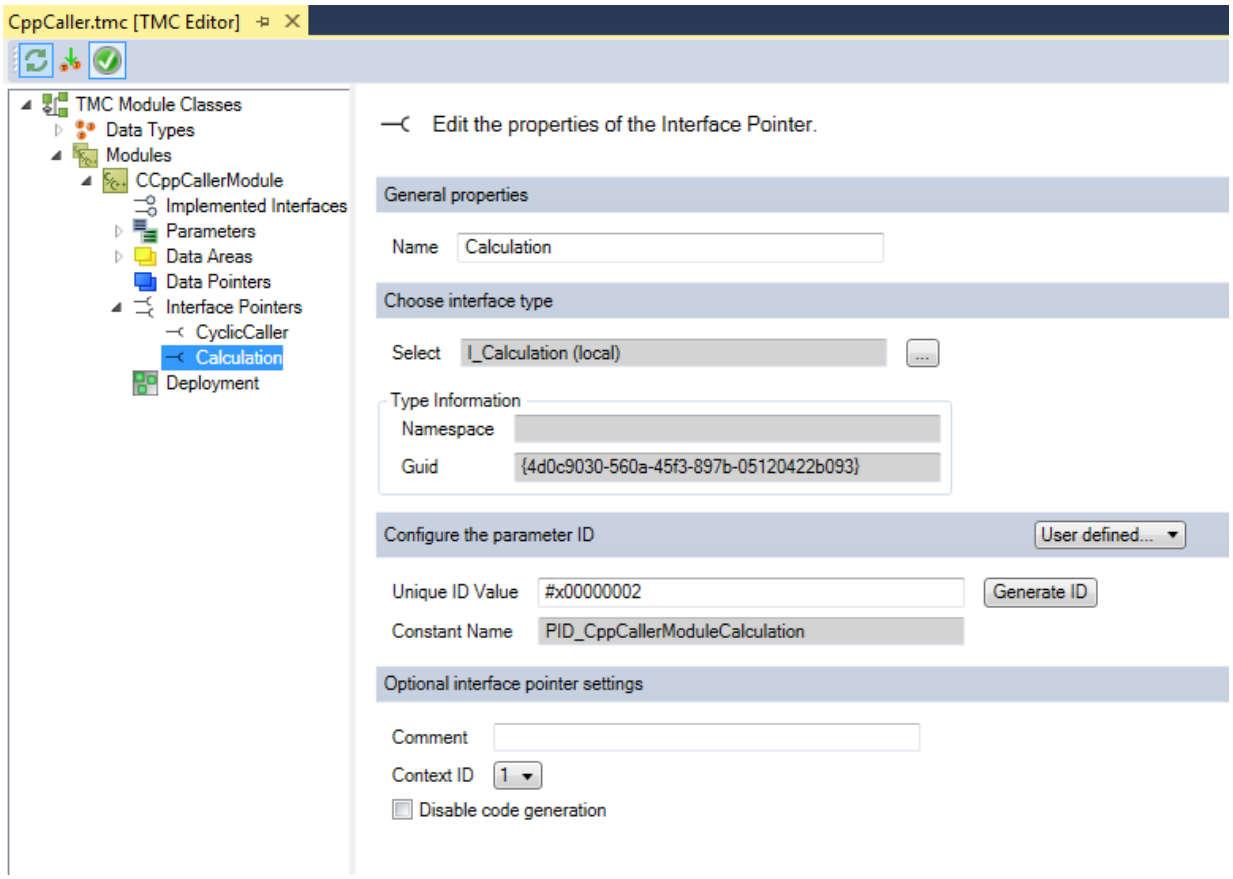
TwinCAT Version	Hardware	Einzubindende SPS Bibliotheken
TwinCAT 3.1, Buid 4020	x86, x64	Tc3_Module

7.4.1 Implementierung des Beispiels

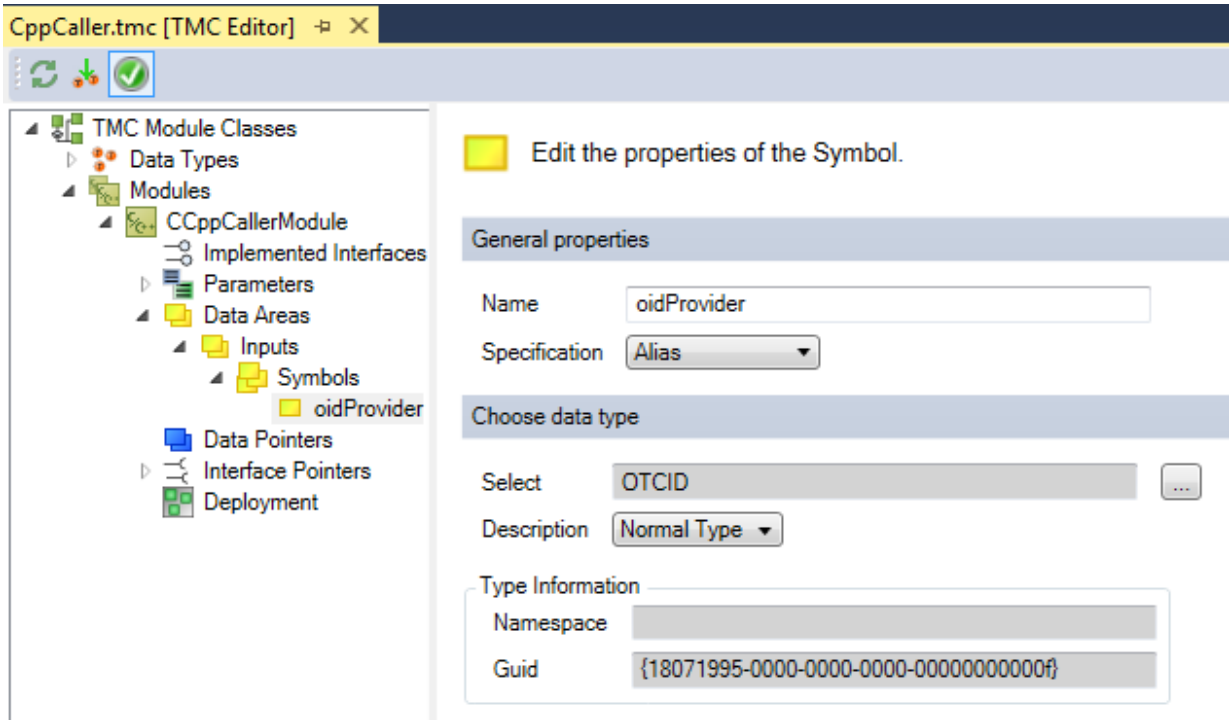
Die SPS-Seite wird von TcCOM Sample 01 [▶ 37] übernommen. Der dort als TcCOM-Modul registrierte Funktionsbaustein bietet die ihm zugeteilte Objekt-ID als Ausgangsvariable an.
Aufgabe des C++ Moduls ist es, das angebotene Interface dieses Funktionsbausteins zugreifbar zu machen.

✓ Es wird von einem C++ Projekt mit einem Cycle IO-Modul ausgegangen.

1. Legen Sie im TMC Editor einen Interface-Pointer mit dem Namen Calculation vom Typ I_Calculation an. Über diesen erfolgt später der Zugriff.

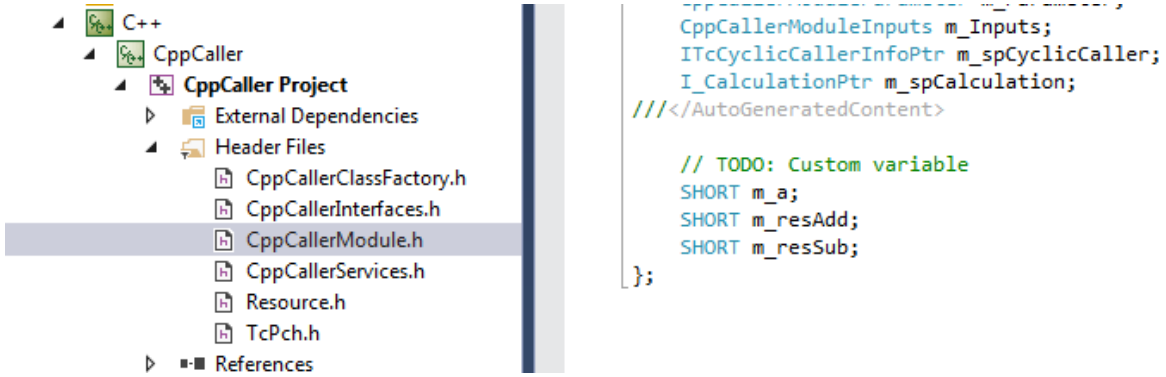


2. Die Data Area Inputs wurde mit dem Typ Input-Destination vom Modul-Wizard bereits erstellt. Hier legen Sie im TMC Editor einen Eingang mit dem Namen oidProvider vom Typ OTCID an. Über diesen wird später die Objekt-ID aus der SPS verknüpft.

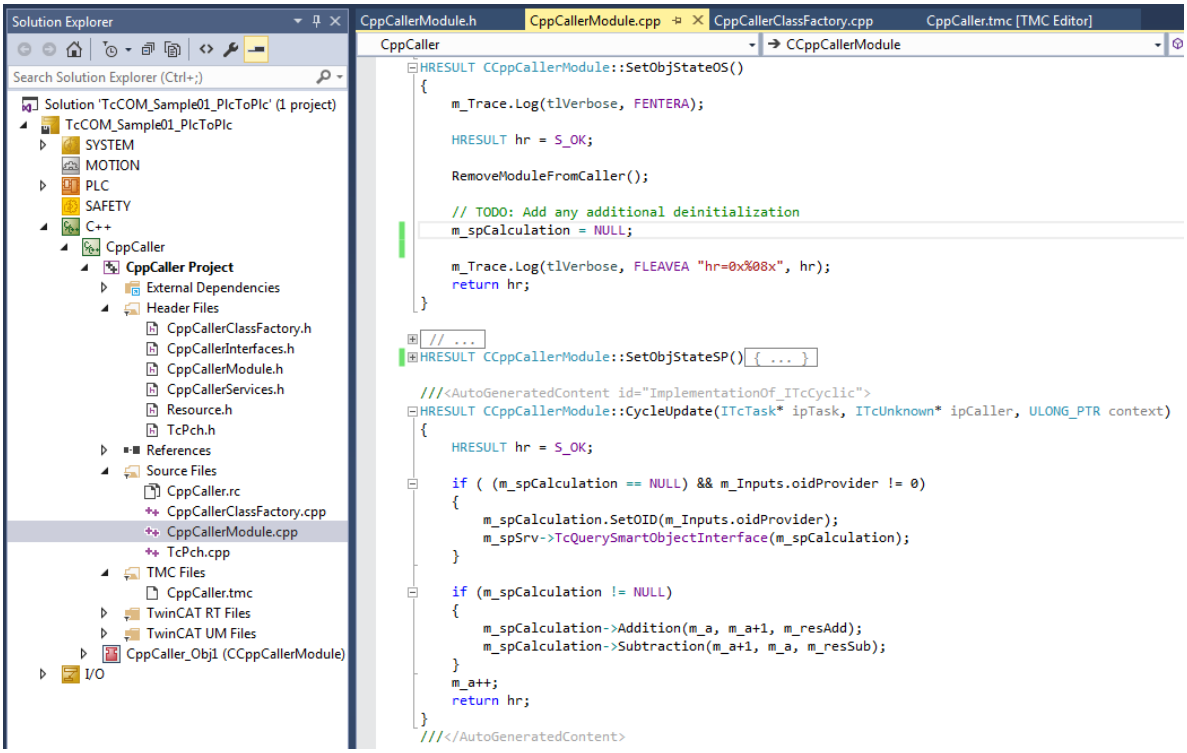


3. Alle weiteren Symbole sind für das Beispiel nicht relevant und können gelöscht werden.

   ⇨ Der TMC-Code-Generator bereitet den Code entsprechend vor:
   In dem Header des Moduls werden einige Variablen angelegt, um die Methoden-Aufrufe später durchzuführen.

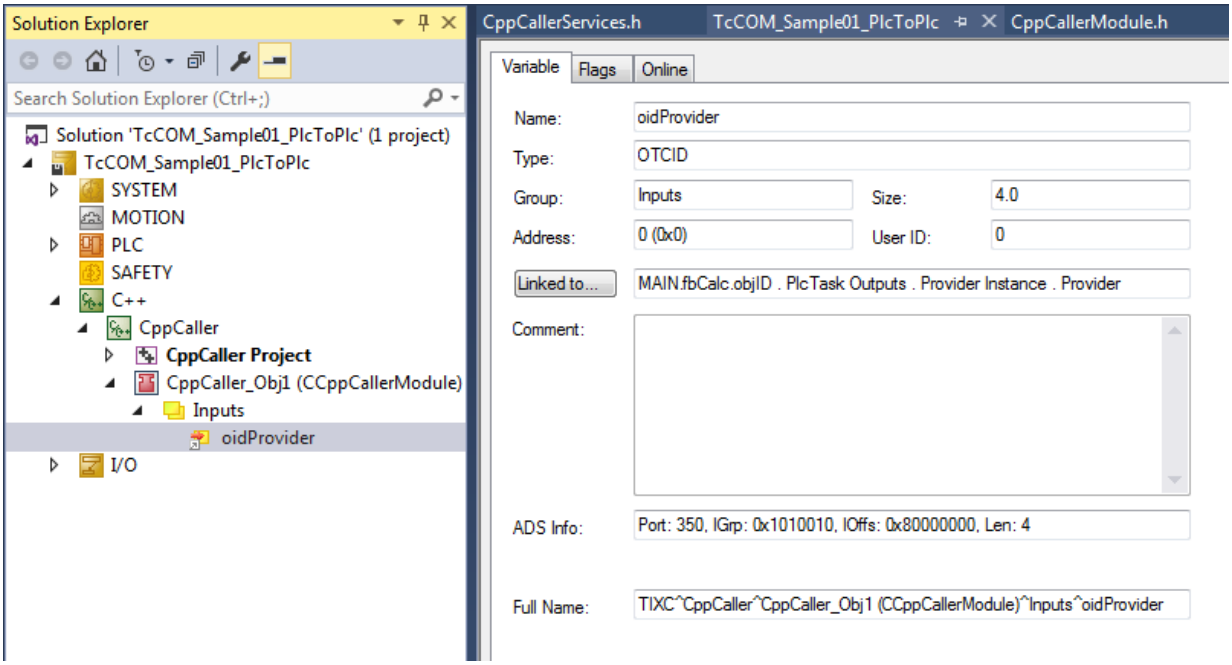


   Im eigentlichen Code des Moduls wird im CycleUpdate() der Interface-Pointer anhand der von der SPS übermittelten Objekt-ID gesetzt. Es ist wichtig, dass dieses im CycleUpdate() und damit im Echtzeitkontext geschieht, da die SPS erst den Baustein bereitstellen muss.
   Ist dies einmalig erfolgt, können die Methoden aufgerufen werden.



Zusätzlich muss, wie oben zu sehen ist, der Interface-Pointer beim Herunterfahren aufgeräumt werden. Dies geschieht in der SetObjStateOS Methode.

4. Bauen Sie das C++ Projekt nun einmal.
5. Legen Sie eine Instanz des Moduls an.
6. Verbinden Sie den Eingang des C++ Moduls mit dem Ausgang der SPS.



⇨ Das Projekt kann gestartet werden. Wenn die SPS läuft, wird die OID durch das Mapping an die C++ Instanz bekannt gemacht. Sobald dies erfolgt ist, können die Methoden aufgerufen werden.

8 Anhang

8.1 TcCOM Technologie

Das TwinCAT-Modulkonzept ist eines der Kernelemente für die Modularisierung moderner Maschinen. Dieses Kapitel beschreibt das Modulkonzept und den Umgang mit Modulen.

8.1.1 Das TwinCAT Component Object Model (TcCOM) Konzept

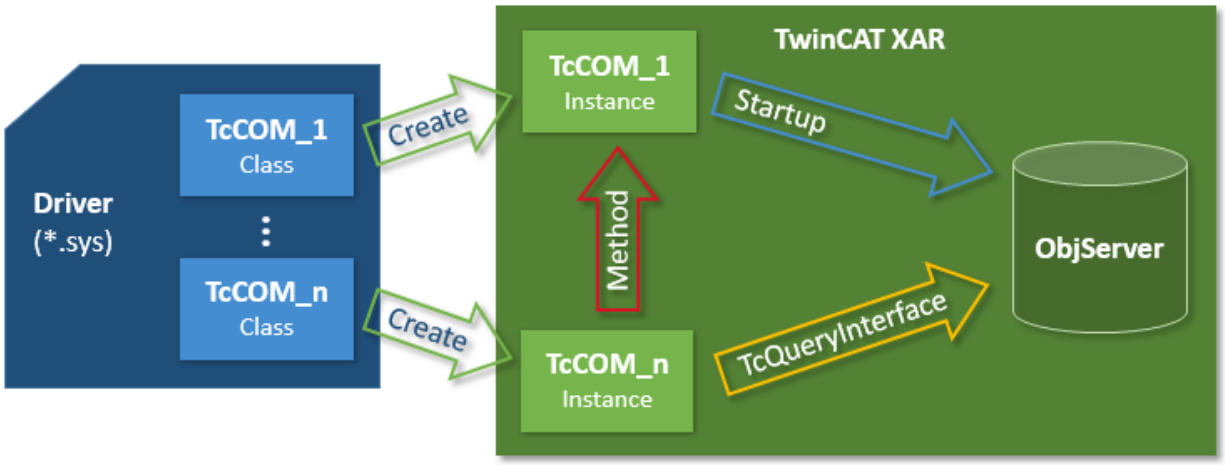
Das TwinCAT Component Object Model definiert die Eigenschaften und das Verhalten der Module. Das vom „Component Object Model“ (COM) von Microsoft Windows abgeleitete Modell beschreibt die Art und Weise, wie verschiedene Software-Komponenten, die unabhängig voneinander entwickelt und kompiliert wurden, zusammen arbeiten können. Damit das möglich ist, müssen ein genau definierter Verhaltensmodus und die Beachtung von Schnittstellen des Moduls definiert werden, so dass sie interagieren können. Eine solche Schnittstelle ist damit beispielsweise auch ideal um Module unterschiedlicher Hersteller in Interaktion zu setzen.

TcCOM macht Anleihen bei COM (Component Object Model aus der Microsoft Windows-Welt), wobei lediglich eine Untermenge von COM verwendet wird. Allerdings enthält TcCOM im Vergleich zu COM zusätzliche Definitionen, die über COM hinausgehen, z.B. die Modul Zustandsmaschine.

Überblick und Verwendung von TcCOM Modulen

Einleitend hier ein Überblick, der die einzelnen Themen leichter verständlich machen soll.

Ein oder mehrere TcCOM Module werden in einem Treiber zusammengefasst. Dieser Treiber wird vom TwinCAT Engineering mittels des MSVC Compilers erstellt. Die Module und Schnittstellen werden in einer TMC (TwinCAT Module Class) Datei beschrieben. Die Treiber mit ihrer TMC Datei können nun zwischen den Engineering Systemen ausgetauscht und kombiniert werden.



Mittels des Engineerings werden nun Instanzen von diesen Modulen erstellt. Zu diesen gibt es eine TMI Datei. Die Instanzen können parametriert und untereinander mit anderen Modulen oder zu dem IO verknüpft werden. Eine entsprechende Konfiguration wird auf das Zielsystem übertragen und dort ausgeführt.

Dabei werden entsprechende Module gestartet, die sich beim TwinCAT-ObjectServer anmelden. Das TwinCAT XAR sorgt auch für die Bereitstellung der Prozessabbilder. Module können den TwinCAT-ObjectServer nach einer Referenz auf ein anderes Objekt in Bezug auf eine bestimmte Schnittstelle fragen. Wenn sie eine solche Referenz erhalten, können sie die Methoden der Schnittstelle auf der Modulinstanz aufrufen.

Die folgenden Abschnitte konkretisieren die einzelnen Themengebiete.

ID Management

Es werden verschiedene Typen von IDs für die Interaktion der Module untereinander und auch innerhalb der Module verwendet. TcCOM verwendet GUIDs (128 Bit) sowie 32 Bit lange Ganzzahlen.

TcCOM verwendet

- GUIDs für: ModulIDs, ClassIDs und InterfaceIDs.
- 32 Bit lange Ganzzahlen werden verwendet für: ParameterIDs, ObjectIDs, ContextIDs, CategoryID.

Schnittstellen

Eine wichtige Komponente von COM, und damit auch von TcCOM, sind Schnittstellen.

Schnittstellen definieren einen Satz Methoden, die zusammengefasst werden, um eine bestimmte Aufgabe zu erfüllen. Eine Schnittstelle wird mit einer eindeutigen ID (InterfaceID) referenziert, die bei gleichbleibender Schnittstelle niemals geändert werden darf. Dank dieser ID können verschiedene Module feststellen, ob sie mit anderen Modulen zusammenarbeiten können. Gleichzeitig kann der Entwicklungsprozess unabhängig erfolgen, wenn die Schnittstellen fest definiert sind. Änderungen an Schnittstellen führen also zu unterschiedlichen IDs. Im TcCOM Konzept ist deswegen vorgesehen, dass InterfaceIDs andere (ältere) InterfaceIDs überlagern können („Hides“ in der TMC Beschreibung / im TMC Editor). Auf diese Weise stehen zum einen beide Varianten des Interfaces bereit, zum anderen wird aber auch immer klar, welches die aktuellste InterfaceID ist. Das gleiche Konzept gibt es auch für die Datentypen.

TcCOM selber definiert bereits eine ganze Reihe an Schnittstellen, die in manchen Fällen vorgeschrieben (z.B. ITComObject), aber in den meisten Fällen optional sind. Viele Schnittstellen machen nur in bestimmten Anwendungsbereichen Sinn. Andere Schnittstellen sind dermaßen allgemein, dass sie häufig wiederverwendet werden können. Kunden-definierte Schnittstellen sind vorgesehen, so dass z.B. zwei Module eines Herstellers in der Lage sind, miteinander in Verbindung zu treten.

- Alle Schnittstellen werden von der grundlegenden Schnittstelle ITcUnknown abgeleitet, die, wie die entsprechende Schnittstelle von COM, die grundlegenden Dienste zur Abfrage von anderen Schnittstellen des Moduls (TcQueryInterface) und zur Steuerung der Lebensdauer des Moduls (TcAddRef und TcRelease) bereitstellt.
- Die ITComObject-Schnittstelle, die von jedem Modul implementiert sein muss, enthält Methoden um auf den Namen, die ObjectID, die ObjectID des Parent, die Parameter und die Zustandsmaschine des Moduls zuzugreifen.

Einige allgemeine Schnittstellen werden von vielen Modulen verwendet:

- ITcCyclic wird von Modulen, die zyklische aufgerufen werden sollen („CycleUpdate“), implementiert. Mit Hilfe des ITcCyclicCaller Interfaces einer TwinCAT-Task kann sich das Modul anmelden um zyklische Aufrufe zu erhalten.
- Durch die ITcADI-Schnittstelle kann auf Datenbereiche eines Moduls zugegriffen werden.
- ITcWatchSource wird standardmäßig implementiert und erlaubt unter anderem ADS-DeviceNotifcations zu erhalten.
- Die ITcTask-Schnittstelle, die von den Tasks des Echtzeitsystems implementiert wird, stellen Informationen bezüglich der Zykluszeit, der Priorität und anderer Informationen zur Task zur Verfügung.
- Die Schnittstelle ITComObjectServer wird vom ObjectServer implementiert und von allen Modulen referenziert.

Es wurden bereits eine ganze Reihe allgemeiner Schnittstellen definiert. Allgemeine Schnittstellen haben den Vorteil, dass deren Verwendung den Austausch und die Wiederverwertung von Modulen unterstützt. Nur dann, wenn keine geeigneten allgemeinen Schnittstellen bestehen, sollten eigene Schnittstellen definiert werden.

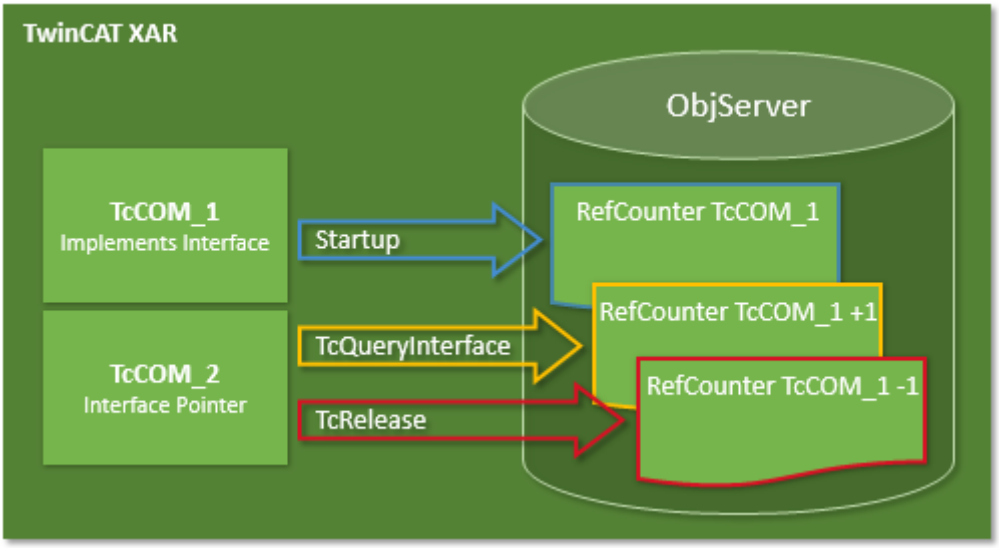
Class Factories

Für die Erzeugung von in C++ Modulen werden sogenannte „Class Factories“ verwendet. Alle Module, die in einem Treiber sind, besitzen eine gemeinsame Class Factory. Die Class Factory meldet sich einmal beim ObjectServer an und bietet ihre Dienste für die Erstellung bestimmter Modulklassen an. Die Modulklassen sind durch die eindeutige ClassID des Moduls gekennzeichnet. Wenn der ObjectServer ein neues Modul

anfordert (auf Grundlage der Initialisierungsdaten des Konfigurators oder durch andere Module während der Laufzeit), wählt das Modul die richtige Class Factory auf Grundlage der ClassID aus und veranlasst die Erzeugung des Moduls über seine ITcClassFactory-Schnittstelle.

Modul-Lebensdauer

Auf ähnliche Weise wie im Falle von COM wird die Lebensdauer eines Moduls über einen Verweiszähler (RefCounter) bestimmt. Jedes Mal, wenn eine Schnittstelle eines Moduls abgefragt wird, wird der Verweiszähler inkrementiert. Er wird wieder dekrementiert, wenn die Schnittstelle freigegeben wird. Eine Schnittstelle wird auch bei der Anmeldung eines Moduls beim ObjectServer abgefragt, (die ITComObject-Schnittstelle), so dass der Verweiszähler zumindest auf eins steht. Bei der Abmeldung wird er erneut dekrementiert. Wenn der Zähler den Wert 0 erreicht, löscht das Modul sich selbst automatisch - normalerweise nach der Abmeldung beim ObjectServer. Wenn aber bereits ein anderes Modul einen Verweis hält (einen Schnittstellenzeiger besitzt), dann besteht das Modul weiter - und der Schnittstellenzeiger bleibt so lange gültig, bis dieser Zeiger auch freigegeben wird.



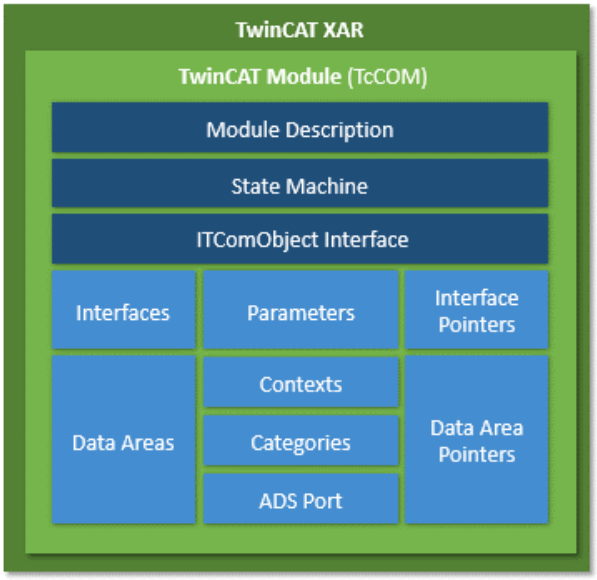
8.1.1.1 TwinCAT-Modul Eigenschaften

Ein TcCOM-Modul hat eine Reihe formal definierter, vorgeschriebener und optionaler Eigenschaften. Die Eigenschaften sind so weit formalisiert, dass eine Verwendung untereinander möglich ist. Jedes Modul hat eine Modulbeschreibung, die die Eigenschaften des Moduls beschreibt. Diese werden für eine Konfiguration der Module und deren Beziehungen untereinander verwendet.

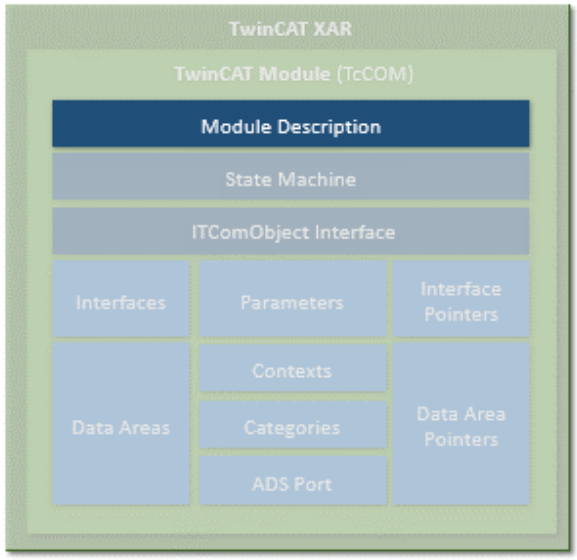
Wenn ein Modul in der TwinCAT Runtime instanziiert wird, dann meldet es sich selber bei einer zentralen Systeminstanz, dem ObjectServer, an. Dadurch wird es für andere Module und auch für allgemeine Tools erreichbar und parametrierbar. Module können unabhängig voneinander kompiliert und demzufolge auch entwickelt, getestet und aktualisiert werden. Module können sehr einfach konzipiert sein, z.B. nur eine kleine Funktion wie einen Tiefpassfilter enthalten. Sie können aber auch intern sehr komplex sein und z.B. das gesamte Steuerungssystem einer Maschinenunterbaugruppe beinhalten.

Es gibt sehr viele Anwendungen für Module; alle Aufgaben eines Automatisierungssystems können in Module spezifiziert werden. Demzufolge wird nicht unterschieden, ob das Modul primär die Grundfunktionen eines Automatisierungssystems darstellt, wie Echtzeit-Tasks, Feldbus-Treiber oder ein SPS-Laufzeitsystem, oder eher benutzer- und anwendungsspezifische Algorithmen für die Steuerung oder Regelung einer Maschineneinheit.

Die Abbildung unten zeigt ein gewöhnliches TwinCAT-Modul mit seinen wichtigsten Eigenschaften. Die dunkelblauen Blöcke definieren vorgeschriebene, die hellblauen Blöcke optionale Eigenschaften.



Modulbeschreibung



Jedes TcCOM Modul hat einige allgemeine Beschreibungsparameter. Dazu gehört eine ClassID, die die Klasse des Moduls eindeutig referenziert. Sie wird durch die passende ClassFactory instanziiert. Jede Instanz eines Moduls hat eine ObjectID, die in der TwinCAT Runtime eindeutig ist. Darüber hinaus gibt es eine Parent-ObjectID, die auf einen möglichen logischen Parent verweist.

Modulbeschreibung, Zustandsmaschine und Parameter des unten beschriebenen Moduls können über die ITComObject-Schnittstelle erreicht werden (Siehe „Schnittstellen“).

Klassenbeschreibungsdateien (*.tmc)

Die Klassen der Module werden in den Klassenbeschreibungsdateien (TwinCAT Module Class; *.tmc) beschrieben.

In dieser Datei beschreibt der Entwickler Eigenschaften und Schnittstellen des Moduls, so dass andere es nutzen und einbetten können. Neben den allgemeinen Informationen (Herstellerangaben, Klassen-ID des Moduls, usw.) werden optionale Eigenschaften des Moduls beschrieben.

- unterstützte Kategorien
- implementierte Schnittstellen
- Datenbereiche mit entsprechenden Symbolen
- Parameter

- Schnittstellenzeiger
- Datenzeiger, die gesetzt werden können

Die Klassenbeschreibungsdateien werden vom Konfigurator des Systems in erster Linie als Grundlage für die Einbindung einer Instanz des Moduls in die Konfiguration, zum Festlegen der Parameter und zwecks Konfiguration der Verbindungen mit anderen Modulen verwendet.

Sie enthalten zudem die Beschreibung aller Datentypen in den Modulen, die dann vom Konfigurator in sein allgemeines Datentypsystem aufgenommen werden. Damit werden also alle Schnittstellen der im System vorhandenen TMC Beschreibungen für alle Module nutzbar.

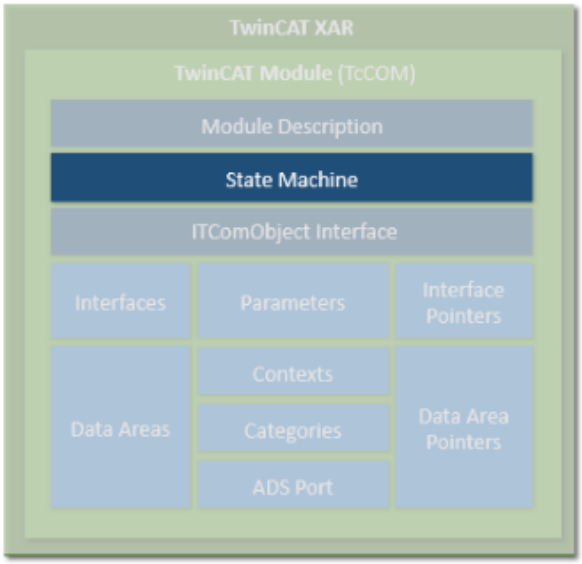
Auch komplexere Konfigurationen mehrerer Module können in den Klassenbeschreibungsdateien beschrieben werden, die für eine spezifische Anwendung vorkonfiguriert und verbunden sind. Demzufolge kann ein Modul für eine komplexe Maschineneinheit, die intern aus einer Reihe von Untermodulen besteht, bereits im Verlauf der Entwicklungsphase als ein Ganzes definiert und vorkonfiguriert werden.

Instanzenbeschreibungsdateien (*.tmi)

Ein Instanz eines bestimmten Moduls wird in der Instanzenbeschreibungsdatei (TwinCAT Module Instance; *.tmi) beschrieben. Die Instanzbeschreibungen sind durch ein ähnliches Format beschrieben, enthalten entgegen den Klassenbeschreibungsdateien aber bereits konkrete Festlegungen der Parameter, Schnittstellenzeiger usw. für die besondere Instanz des Moduls innerhalb eines Projekts.

Die Instanzenbeschreibungsdateien werden vom TwinCAT Engineering (XAE) erstellt, wenn eine Instanz einer Klassenbeschreibung für ein konkretes Projekt erstellt wird. Sie dienen hauptsächlich dem Datenaustausch zwischen allen an der Konfiguration beteiligten Tools. Allerdings können die Instanzenbeschreibungen auch projektübergreifend genutzt werden, wenn z.B. ein besonders parametriertes Modul in einem neuen Projekt erneut verwendet werden soll.

Zustandsmaschine

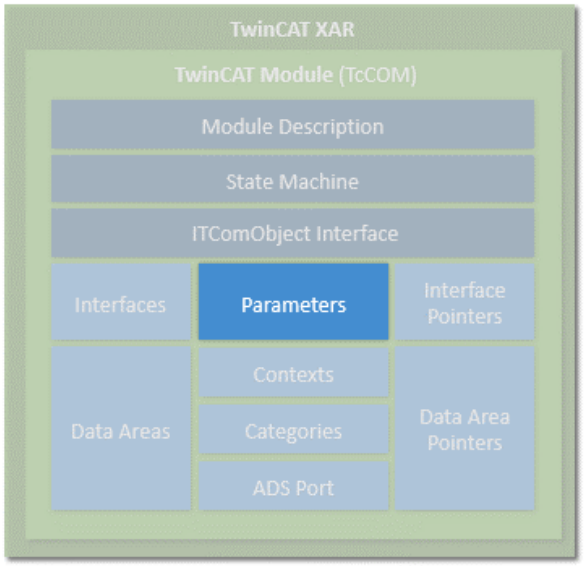


Jedes Modul enthält eine Zustandsmaschine, die den Initialisierungszustand des Moduls und die Mittel, mit denen dieser Zustand von außen verändert werden kann, beschreibt. Diese Zustandsmaschine beschreibt die Zustände, die beim Starten und Beenden des Moduls durchlaufen werden. Dieses betrifft die Modulerzeugung, -parametrierung und Herstellung der Verbindung mit den anderen Modulen.

Anwendungsspezifische Zustände (z.B. von Feldbus oder Treiber) können in ihren eigenen Zustandsmaschinen beschrieben werden. Die Zustandsmaschine der TcCOM-Module definiert die Zustände INIT, PREOP, SAFEOP und OP. Auch wenn es dieselben Zustandsbezeichnungen sind wie unter EtherCAT-Feldbus sind es doch nicht die gleichen Zustände. Wenn das TcCOM-Modul einen Feldbustreiber für EtherCAT implementiert, hat es zwei Zustandsmaschinen (Modul- und Feldbuszustandsmaschine), die nacheinander durchlaufen werden. Die Modulzustandsmaschine muss den Betriebszustand (OP) erreicht haben, bevor die Feldbuszustandsmaschine überhaupt starten kann.

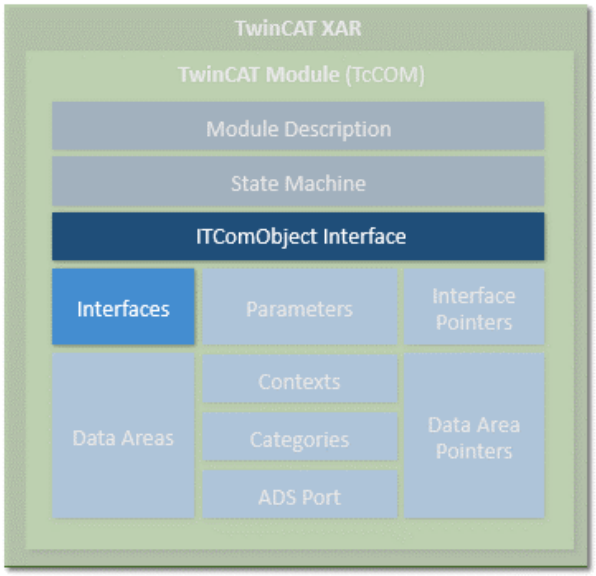
Die Zustandsmaschine ist im Detail separat beschrieben [▶ 68].

Parameter



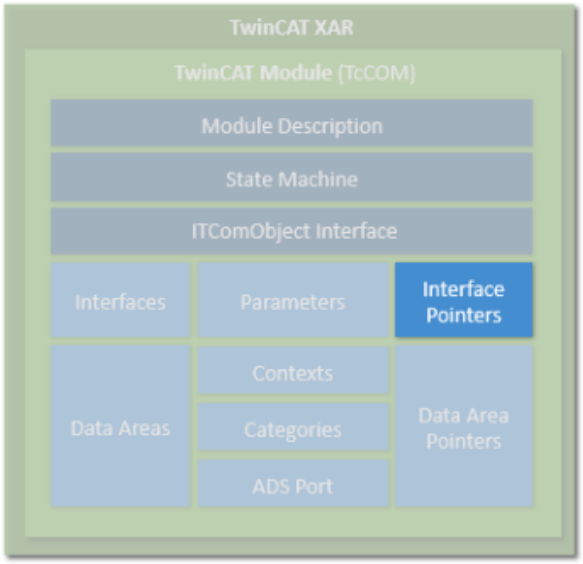
Module können Parameter haben, die während der Initialisierung oder später während der Laufzeit (OP-Zustand) gelesen oder geschrieben werden können. Jeder Parameter ist durch eine Parameter-ID gekennzeichnet. Die Eindeutigkeit der Parameter-ID kann global, eingeschränkt global oder modulspezifisch sein. Mehr hierzu im Abschnitt „ID Management“. Neben der Parameter-ID enthält der Parameter die aktuellen Daten; der Datentyp hängt vom Parameter ab und ist für die jeweilige Parameter-ID eindeutig definiert.

Schnittstellen



Schnittstellen bestehen aus einem definierten Satz an Methoden (Funktionen), die Module anbieten und über die sie z.B. von anderen Modulen kontaktiert werden können. Schnittstellen sind durch eine eindeutige ID charakterisiert, wie oben beschrieben. Ein Modul muss mindestens die ITComObject-Schnittstelle unterstützen, kann aber daneben so viele Schnittstellen wie gewünscht beinhalten. Eine Schnittstellen-Referenz kann durch Aufruf der Methode „TcQueryInterface“ mit Angabe der entsprechenden Schnittstellen-ID abgefragt werden.

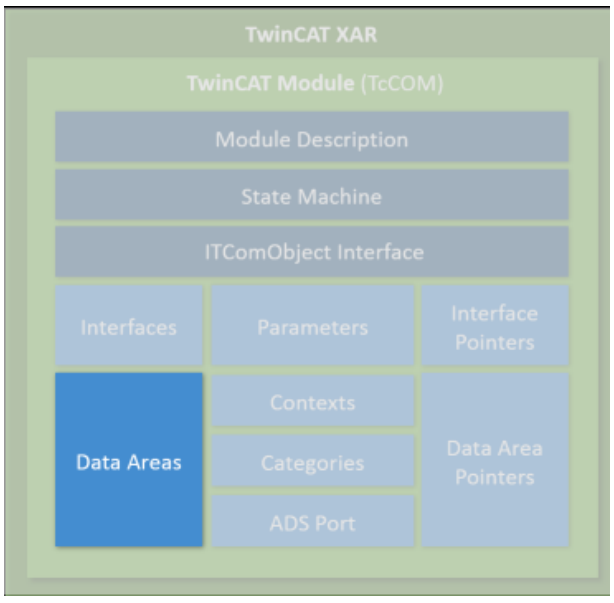
Schnittstellenzeiger



Schnittstellenzeiger verhalten sich wie das Gegenstück von Schnittstellen. Wenn ein Modul eine Schnittstelle eines anderen Moduls nutzen möchte, muss es einen Schnittstellenzeiger des entsprechenden Schnittstellentyps haben und dafür sorgen, dass dieser auf das andere Modul zeigt. Danach können die Methoden des anderen Moduls verwendet werden.

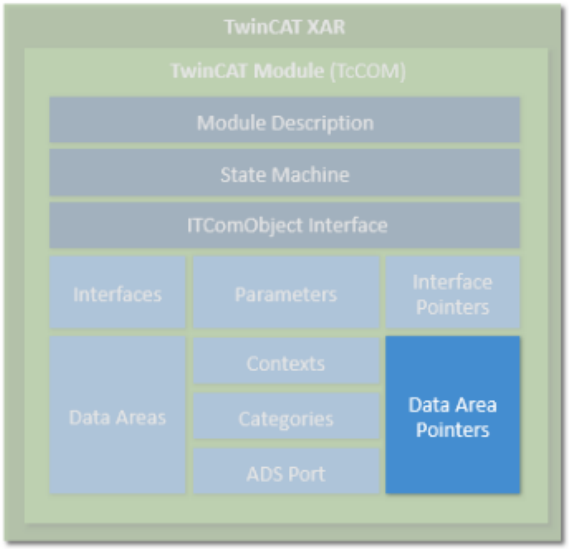
Schnittstellenzeiger werden normalerweise beim Start der Zustandsmaschine gesetzt. Beim Übergang von INIT zu PREOP (IP) empfängt das Modul die Objekt-ID des anderen Moduls mit der entsprechenden Schnittstelle, beim Übergang PREOP zu SAFEOP (PS) oder SAFEOP zu OP (SO) wird die Instanz des anderen Moduls mit dem ObjectServer durchsucht und die entsprechende Schnittstelle mit der Methode Query Interface gesetzt. Beim Zustandsübergang in die entgegengesetzte Richtung, von SAFEOP zu PREOP (SP) oder OP zu SAFEOP (OS) muss die Schnittstelle wieder freigegeben werden.

Datenbereiche



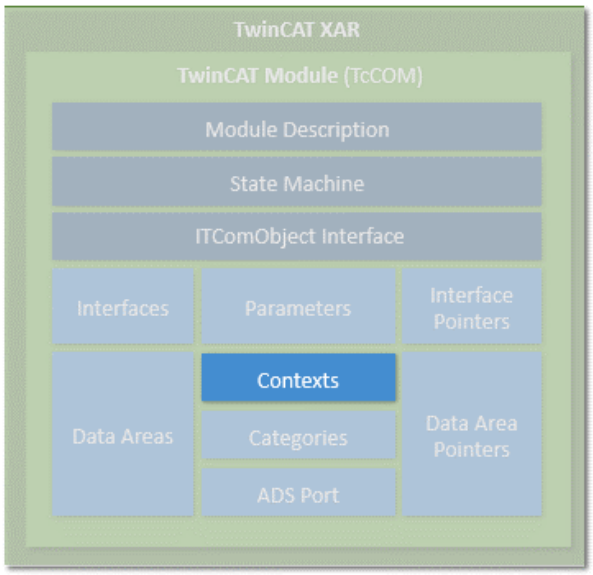
Module können Datenbereiche enthalten, die von der Umgebung verwendet werden können (z.B. von anderen Modulen oder zum IO Bereich von TwinCAT). Diese Datenbereiche können beliebige Daten enthalten. Sie werden oft für Prozessabbilddaten (Ein- und Ausgänge) genutzt. Die Struktur der Datenbereiche wird in der Gerätebeschreibung des Moduls definiert. Wenn ein Modul Datenbereiche hat, die es für andere zugänglich machen möchte, implementiert es die ITcADI-Schnittstelle, die den Zugriff auf die Daten ermöglicht. Datenbereiche können Symbolinformationen enthalten, die die Struktur des jeweiligen Datenbereichs im Einzelnen beschreiben.

Datenbereichszeiger



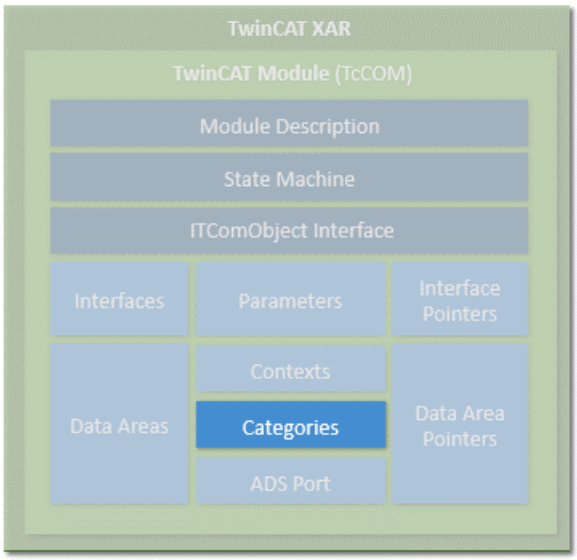
Wenn ein Modul auf den Datenbereich anderer Module zugreifen möchte, kann es Datenbereichszeiger enthalten. Diese werden normalerweise während der Initialisierung der Zustandsmaschine auf Datenbereiche oder Datenbereichsabschnitte anderer Module gesetzt. Es handelt sich dabei um einen direkten Zugriff auf den Speicherbereich, so dass bei Bedarf entsprechende Schutzmechanismen für konkurrierende Zugriffe eingesetzt werden müssen. Häufig bietet sich deshalb besser an, eine entsprechende Schnittstelle zu nutzen.

Kontext



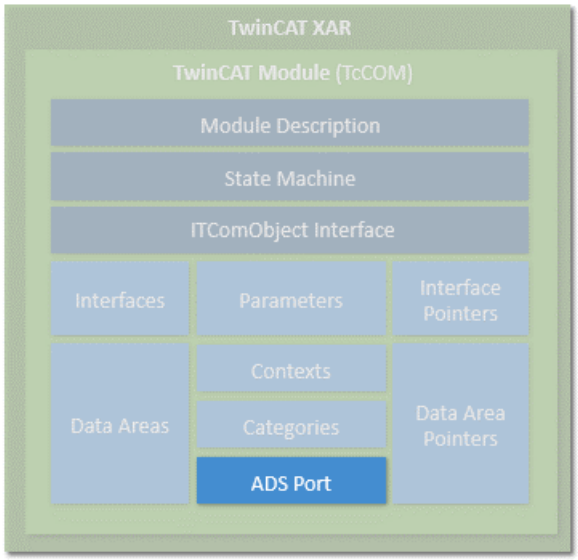
In diesem Zusammenhang ist Kontext als Echtzeit-Task Kontext zu verstehen. Kontexte werden u.a. für die Konfiguration der Module benötigt. Einfache Module arbeiten normalerweise in einem einzigen Zeitkontext, demzufolge muss er nicht näher spezifiziert werden. Andere Module können teilweise in mehreren Kontexten aktiv sein (z.B. ein EtherCAT Master kann mehrere unabhängige Echtzeit Tasks unterstützten, oder eine Regelschleife kann Regelschleifen der darunterliegenden Ebene in anderer Zykluszeit abarbeiten). Wenn ein Modul mehr als einen zeitabhängigen Kontext hat, muss das in der Modulbeschreibung angegeben werden.

Kategorien



Module können Kategorien anbieten indem sie das Interface ITComObjectCategory implementieren. Kategorien werden vom ObjectServer enumeriert und Objekte, die sich hierrüber Kategorien zuordnen, können durch den ObjectServer (ITComObjectEnumPtr) abgefragt werden.

ADS



Jedes Modul, das beim ObjectServer eingetragen ist, kann per ADS erreicht werden. Der ObjectServer nutzt dabei die ITComObject Schnittstelle der Module, um z.B. Parameter zu lesen oder zu schreiben oder auch um auf die Zustandsmaschine zuzugreifen. Es gibt zusätzlich die Möglichkeit der Implementierung eines eigenes ADS Ports, über den eigene ADS Kommandos empfangen werden können.

Systemmodul

Die TwinCAT Laufzeit stellt darüber hinaus eine Reihe sogenannter Systemmodule zur Verfügung, die die grundlegenden Dienste der Laufzeit für andere Module zur Verfügung stellen. Diese Systemmodule haben eine feste, konstante ObjectID, über welche die anderen Module auf sie zugreifen können. Ein Beispiel eines solchen Systemmoduls ist das Echtzeitsystem, das die grundlegenden Dienste des Echtzeitsystems, d.h. die Generierung von Echtzeit-Tasks, via ITcRTime-Schnittstelle zur Verfügung stellt. Auch der ADS Router ist als Systemmodul implementiert, so dass andere Module an dieser Stelle ihren ADS Port anmelden können.

Erstellung von Modulen

Module können sowohl in C++ als auch in IEC 61131-3 erstellt werden. Die objektorientierten Erweiterungen der TwinCAT PLC werden hierzu verwendet. Module aus den beiden Welten können über Schnittstellen auf die gleiche Weise wie reine C++ Module untereinander interagieren. Mit Hilfe der objektorientierten Erweiterung werden die gleichen Schnittstellen bereitgestellt wie in C++.

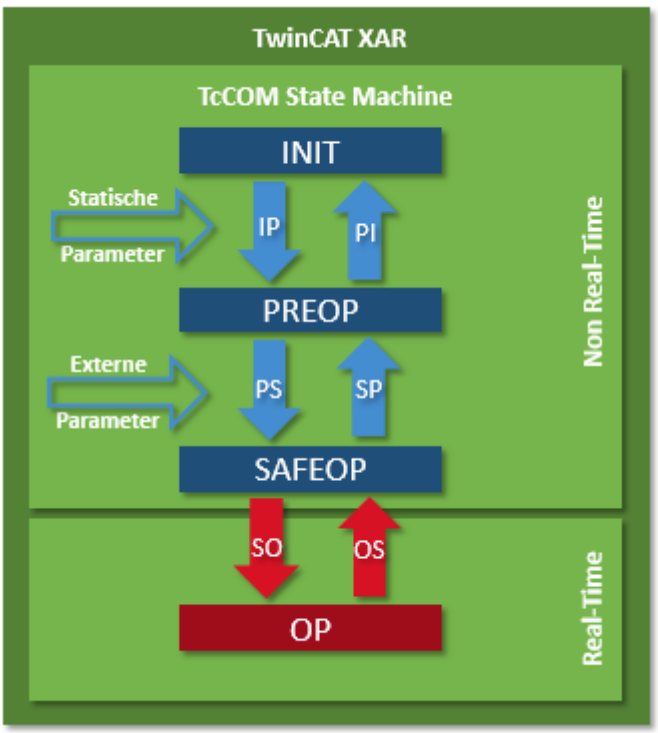
Die SPS-Module melden sich ebenfalls selber beim ObjectServer an und sind demzufolge über ihn erreichbar. Die Komplexität von SPS-Modulen ist unterschiedlich. Es macht keinen Unterschied, ob nur ein kleines Filtermodul generiert oder ein komplettes SPS-Programm in ein Modul hineingepackt wird. Jedes SPS-Programm ist aufgrund der Automation ein Modul im Sinne der TwinCAT-Module. Jedes klassische SPS-Programm wird automatisch in ein Modul gepackt und meldet sich selber beim ObjectServer und bei einem oder mehreren Task-Modulen an. Der Zugriff auf die Prozessdaten eines SPS-Moduls (z.B. Mapping in Bezug auf einen Feldbus-Treiber) wird ebenfalls über die definierten Datenbereiche und ITcADI gesteuert.

Dieses Verhalten bleibt transparent und unsichtbar für den SPS-Programmierer, so lange bis er beschließt ausdrücklich Teile des SPS-Programms als TwinCAT-Module zu definieren, damit er diese mit geeigneter Flexibilität nutzen kann.

8.1.1.2 TwinCAT-Modul Zustandsmaschine

Neben den Zuständen (INIT, PREOP, SAFEOP und OP) gibt es entsprechende Zustandsübergänge, innerhalb derer allgemeine oder modulspezifische Aktionen auszuführen sind oder ausgeführt werden können. Die Zustandsmaschine ist sehr einfach konzipiert; in jedem Falle gibt es nur Übergänge zum nächsten oder vorherigen Schritt.

Hieraus ergeben sich die Zustandsübergänge: INIT zu PREOP (IP), PREOP zu SAFEOP (PS) und SAFEOP zu OP (SO). In umgekehrter Richtung bestehen die folgenden Zustandsübergänge: OP zu SAFEOP (OS), SAFEOP zu PREOP (SP) und PREOP zu INIT (PI). Einschließlich bis zum SAFEOP-Zustand finden alle Zustände und Zustandsübergänge innerhalb des Nicht-Echtzeitkontextes statt. Nur der Übergang SAFEOP zu OP, der Zustand OP und der Übergang OP zu SAFEOP finden im Echtzeitkontext statt. Diese Differenzierung hat einen Einfluss, wenn Ressourcen alloziert oder freigegeben werden, oder wenn Module sich bei anderen Modulen an- oder abmelden.



Zustand: INIT

Der INIT-Zustand ist nur ein virtueller Zustand. Sofort nach Erstellung eines Moduls wechselt das Modul von INIT zu PREOP, d.h. der IP-Zustandsübergang wird ausgeführt. Die Instanziierung und der IP-Zustandsübergang erfolgen immer zusammen, so dass das Modul nie im INIT-Zustand bleibt. Nur beim Entfernen des Moduls, bleibt es kurzzeitig im INIT-Zustand.

Transition: INIT zu PREOP (IP)

Während des IP-Zustandsübergangs meldet sich das Modul mit seiner eindeutigen ObjectID beim ObjectServer an. Die Initialisierungsparameter, die auch während der Objekterstellung alloziert werden, werden an das Modul weitergegeben. Bei diesem Übergang kann das Modul keine Verbindung zu anderen Modulen herstellen, weil nicht sicher ist, ob die anderen Module bereits bestehen und beim ObjectServer angemeldet sind. Wenn das Modul Systemressourcen benötigt (z.B. Speicherplatz), können diese während des Zustandsübergangs alloziert werden. Alle allozierten Ressourcen müssen dann entsprechend beim Übergang PREOP zu INIT (PI) wieder freigegeben werden.

Zustand: PREOP

Im PREOP-Zustand ist das Modul vollständig erstellt und auch normalerweise vollständig parametriert, auch wenn möglicherweise beim Übergang von PREOP zu SAFEOP weitere Parameter hinzukommen. Das Modul ist im ObjectServer angemeldet, es wurden aber noch keine Verbindungen mit anderen Modulen herstellt.

Transition: PREOP zu SAFEOP (PS)

In diesem Zustandsübergang kann das Modul Verbindungen mit anderen Modulen herstellen. Zu diesem Zweck hat es normalerweise, neben anderen Dingen, ObjectIDs von anderen Modulen mit den Initialisierungsdaten erhalten, die nun über den ObjectServer in reale Verbindungen mit diesen Modulen umgewandelt werden.

Der Übergang kann sowohl im Allgemeinen vom System, gemäß dem Konfigurator, als auch von einem anderen Modul (z.B. dem Parent-Modul) veranlasst werden. Im Verlauf dieses Zustandsübergangs können auch weitere Parameter übergeben werden. So kann z.B. das Parent-Modul eigene Parameter an das Child-Modul übergeben.

Zustand: SAFEOP

Das Modul ist noch im Nicht-Echtzeitkontext und wartet darauf, vom System oder von anderen Modulen in den OP-Zustand geschaltet zu werden.

Transition: SAFEOP zu OP (SO)

Sowohl der Zustandsübergang von SAFEOP zu OP, als auch der Zustand OP, als auch der Übergang von OP zu SAFEOP finden im Echtzeitkontext statt! Ressourcen des Systems dürfen nicht mehr alloziert werden. Auf der anderen Seite können nun Ressourcen von anderen Modulen angefordert werden und Module können sich bei anderen Modulen anmelden, z.B. im Verlauf von Tasks, um einen zyklischen Aufruf zu erhalten.

Diese Transition sollte nicht für langlaufende Aufgaben verwendet werden. So sollten z.B. Dateioperationen schon im PS ausgeführt werden.

Zustand: OP

Im OP-Zustand nimmt das Modul seine Arbeit auf und ist im Sinne des TwinCAT-Systems voll aktiv.

Transition: OP zu SAFEOP (OS)

Dieser Zustandsübergang findet im Echtzeitkontext statt. Alle Aktionen aus dem SO-Übergang werden umgekehrt und alle beim SO-Übergang angeforderten Ressourcen werden wieder freigegeben.

Transition: SAFEOP zu PREOP (SP)

Alle Aktionen vom PS-Übergang werden umgekehrt und alle beim PS-Übergang angeforderten Ressourcen werden wieder freigegeben.

Transition: PREOP zu INIT (PI)

Alle Aktionen vom IP-Übergang werden umgekehrt und alle beim IP-Übergang angeforderten Ressourcen werden wieder freigegeben. Das Modul meldet sich beim ObjectServer ab und löscht sich normalerweise selbst (siehe „Lebensdauer“).

8.2 Schnittstellen

8.2.1 Schnittstelle ITComObject

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt grundlegende Funktionalitäten zur Verfügung.

Syntax

```
TCOM_DECL_INTERFACE("00000012-0000-0000-e000-000000000064", ITComObject)
struct __declspec(novtable) ITComObject: public ITcUnknown
```

Methoden

Name	Beschreibung
TcGetObjectId(OTCID& objId) [▶ 70]	Speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.
TcSetObjectId [▶ 71]	Setzt die Objekt-ID des Objekts auf die gegebene OTCID.
TcGetObjectName [▶ 71]	Speichert den Objektnamen im Puffer mit der gegebenen Länge.
TcSetObjectName [▶ 71]	Setzt den Objektnamen des Objekts auf gegebenen CHAR*.
TcSetObjState [▶ 72]	Initialisiert einen Übergang zu einem vorgegebenen Zustand.
TcGetObjState [▶ 72]	Fragt den aktuellen Zustands des Objekts ab.
TcGetObjPara [▶ 73]	Fragt einen mit seiner PTCID identifizierten Objektparameter ab.
TcSetObjPara [▶ 73]	Setzt einen mit seiner PTCID identifizierten Objektparameter.
TcGetParentObjId [▶ 74]	Speichert die Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.
TcSetParentObjId [▶ 74]	Setzt die Parent-Objekt-ID auf die gegebene OTCID.

Anmerkungen

Die ITComObject Schnittstelle wird von jedem TwinCAT-Modul implementiert. Sie stellt Funktionalitäten zur Verfügung bezüglich der Zustandsmaschine und Informationen vom/an das TwinCAT-System.

8.2.1.1 Methode ITcComObject:TcGetObjectId

Die Methode speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
HRESULT TcGetObjectId( OTCID& objId )
```

Parameter

objId: (Typ: OTCID&) Referenz auf OTCID-Wert.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode speichert Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

8.2.1.2 Methode ITcComObject:TcSetObjectId

Die Methode TcSetObjectId setzt die Objekt-ID des Objekts auf die gegebene OTCID.

Syntax

```
HRESULT TcSetObjectId( OTCID objId )
```

Parameter

objId: (Typ: OTCID) Die zu setzende OTCID.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

Zeigt den Erfolg der ID-Änderung an.

8.2.1.3 Methode ITcComObject:TcGetObjectName

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

Syntax

```
HRESULT TcGetObjectName( CHAR* objName, ULONG nameLen );
```

Parameter

objName: (Typ: CHAR*) der zu setzende Name.

nameLen: (Typ: ULONG) die maximale, zu schreibende Länge.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

8.2.1.4 Methode ITcComObject:TcSetObjectName

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR*.

Syntax

```
HRESULT TcSetObjectName( CHAR* objName )
```

Parameter

objName: (Typ: CHAR*) der zu setzende Name des Objekts.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR*.

8.2.1.5 Methode ITcComObject:TcSetObjState

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

Syntax

```
HRESULT TcSetObjState(TCOM_STATE state, ITComObjectServer* ipSrv, PTComInitDataHdr pInitData);
```

Parameter

state: (Typ: TCOM_STATE) stellt den neuen Zustand dar.

ipSrv: (Typ: ITComObjectServer*) ObjServer, der das Objekt handhabt.

pInitData: (Typ: PTComInitDataHdr) zeigt auf eine Liste von Parametern (optional), siehe Makro IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA als Beispiel, wie die Liste iteriert werden kann.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

8.2.1.6 Methode ITcComObject:TcGetObjState

Die Methode TcGetObjState fragt den aktuellen Zustands des Objekts ab.

Syntax

```
HRESULT TcGetObjState(TCOM_STATE* pState)
```

Parameter

pState: (Typ: TCOM_STATE*) Zeiger auf den Zustand.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcGetObjState fragt den aktuellen Zustand des Objekts ab.

8.2.1.7 Methode ITcComObject:TcGetObjPara

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

Syntax

```
HRESULT TcGetObjPara(PTCID pid, ULONG& nData, PVOID& pData, PTCGP pgp=0)
```

Parameter

pid: (Typ: PTCID) Parameter-ID des Objektparameters.

nData: (Typ: ULONG&) max. Länge der Daten.

pData: (Typ: PVOID&) Zeiger auf die Daten.

pgp: (Typ: PTCGP) für zukünftige Erweiterung vorbehalten, NULL weitergeben.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

8.2.1.8 Methode ITcComObject:TcSetObjPara

Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

Syntax

```
HRESULT TcSetObjPara(PTCID pid, ULONG nData, PVOID pData, PTCGP pgp=0)
```

Parameter

pid: (Typ: PTCID) Parameter-ID des Objektparameters.

nData: (Typ: ULONG) max. Länge der Daten.

pData: (Typ: PVOID) Zeiger auf die Daten.

pgp: (Typ: PTCGP) für zukünftige Erweiterung vorbehalten, NULL weitergeben.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

8.2.1.9 Methode ITcComObject:TcGetParentObjId

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
HRESULT TcGetParentObjId( OTCID& objId )
```

Parameter

objId: (Typ: OTCID&) Referenz auf OTCID-Wert.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Beschreibung

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

8.2.1.10 Methode ITcComObject:TcSetParentObjId

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
HRESULT TcSetParentObjId( OTCID objId )
```

Parameter

objId: (Typ: OTCID) Referenz auf OTCID-Wert.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

8.2.2 Schnittstelle ITcUnknown

ITcUnknown definiert die Referenzzählung, sowie das Abfragen einer Referenz auf eine spezifischere Schnittstelle.

Syntax

```
TCOM_DECL_INTERFACE("00000001-0000-0000-e000-000000000064", ITcUnknown)
```

Deklariert in: TcInterfaces.h

Benötigtes include: -

Methoden

Name	Beschreibung
TcAddRef [▶ 75]	Inkrementiert den Referenzzähler.
TcQueryInterface [▶ 75]	Abfrage der Referenz an eine implementierte Schnittstelle über der IID.
TcRelease [▶ 76]	Dekrementiert den Referenzzähler.

Anmerkungen

Jede TcCOM Schnittstelle ist direkt oder indirekt von ITcUnknown abgeleitet. Demzufolge implementiert jede TcCOM Modulklasse ITcUnknown, weil sie von ITComObject abgeleitet ist.

Die standardmäßige Implementierung von ITcUnknown sorgt dafür, dass das Objekt nach Freigabe der letzten Referenz gelöscht wird. Aus diesem Grunde darf ein Schnittstellenzeiger nach dem Aufruf von TcRelease() nicht dereferenziert werden.

8.2.2.1 Methode ITcUnknown:TcAddRef

Diese Methode inkrementiert den Referenzzähler.

Syntax

```
ULONG TcAddRef( )
```

Rückgabewert

Daraus resultierender Referenzzählwert.

Beschreibung

Inkrementiert den Referenzzähler und gibt den neuen Wert zurück.

8.2.2.2 Methode ITcUnknown:TcQueryInterface

Abfrage eines Schnittstellenzeigers in Bezug auf eine Schnittstelle, die per Interface ID (IID) gegeben ist.

Syntax

```
HRESULT TcQueryInterface(RITCID iid, PPVOID pipItf )
```

iid: (Typ: RITCID) Schnittstelle IID.

pipItf: (Typ PPVOID) Zeiger auf Schnittstellenzeiger. Wird gesetzt, wenn der verlangte Schnittstellentyp von der entsprechenden Instanz verfügbar ist.

Rückgabewert

Bei Erfolg wird S_OK („0“) oder ein anderer positiver Wert zurückgegeben, vgl. Rückgabewerte. Erweiterte Meldungen beziehen sich dabei insbesondere auf die Spalte HRESULT in ADS Return Codes [▶ 32].

Wenn die verlangte Schnittstelle nicht verfügbar ist, gibt die Methode ADSERR_DEVICE_NOINTERFACE zurück.

Beschreibung

Abfrage der Referenz an eine implementierte Schnittstelle über der IID. Es wird empfohlen, Smart Pointer zu verwenden, um Schnittstellenzeiger zu initialisieren und zu halten.

Variante 1:

```
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
HRESULT hr = S_OK;
if (ip != NULL)
```

```
{
ITComObjectPtr spObj;
hr = ip->TcQueryInterface(spObj.GetIID(), &spObj);
if (SUCCEEDED(hr))
{
hr = spObj->TcGetObjPara(PID_TcTraceLevel, &tl, sizeof(tl));
}
return hr;
}
}
```

Die mit dem Smart Pointer verbundene Schnittstellen-ID kann in TcQueryInterface als Parameter verwendet werden. Der Operator „&“ wird den Zeiger auf die interne Schnittstellen-Zeiger-Membervariable des Smart Pointers zurückgeben. Variante 1 geht davon aus, dass der Schnittstellenzeiger initialisiert ist, wenn TcQueryInterface Erfolg anzeigt. Wenn der Bereich bleibt, dann gibt der Destructor des Smart Pointers spObj die Referenz frei.

Variante 2:

```
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
HRESULT hr = S_OK;
ITComObjectPtr spObj = ip;
if (spObj != NULL)
{
spObj->TcGetObjParam(PID_TcTraceLevel, &tl);
}
else
{
hr = ADS_E_NOINTERFACE;
}
return hr;
}
```

Wenn der Schnittstellenzeiger ip dem Smart Pointer spObj zugewiesen wird, dann wird die TcQueryInterface-Methode implizit aufgerufen mit IID_ITComObject auf der Instanz, auf die ip verweist. Dies führt zu einem kürzeren Code, aber der ursprüngliche Return-Code von TcQueryInterface geht verloren.

8.2.2.3 Methode ITcUnknown:TcRelease

Diese Methode dekrementiert den Referenzzähler.

Syntax

```
ULONG TcRelease( )
```

Rückgabewert

Daraus resultierender Referenzzählwert.

Beschreibung

Dekrementiert den Referenzzähler und gibt den neuen Wert zurück.

Wenn der Referenzzähler 0 wird, löscht das Objekt sich selber.

Mehr Informationen:
www.beckhoff.de/te1000

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