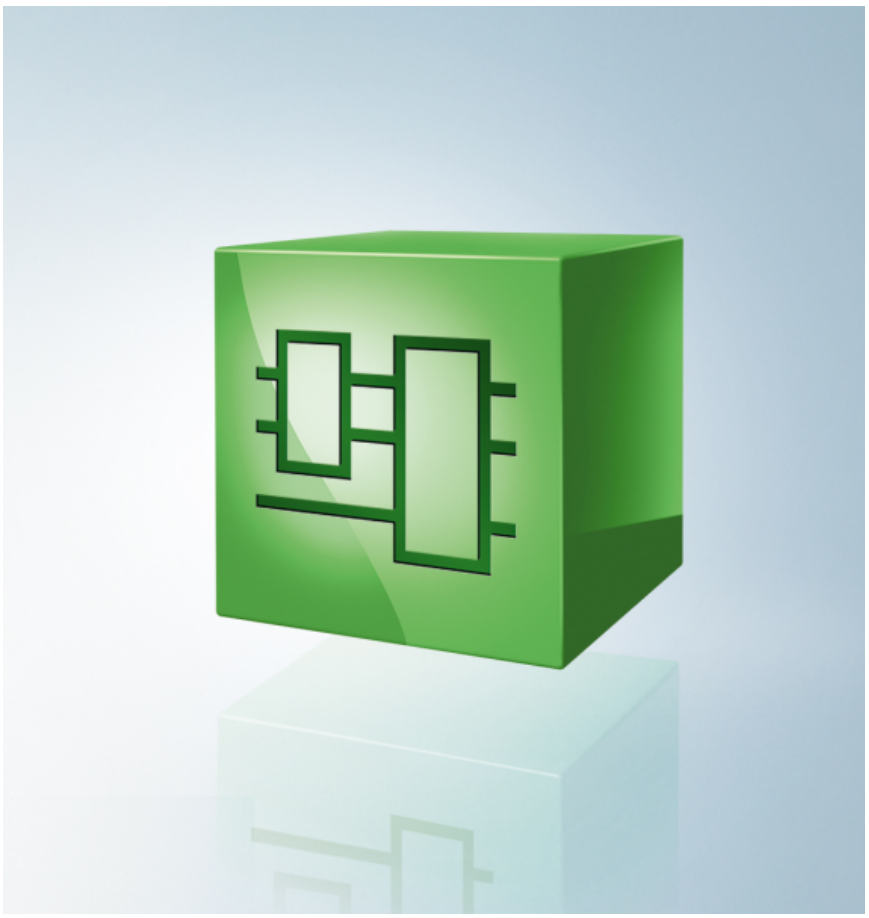


取扱説明書

PLCライブラリ: Tc3_Module

TwinCAT 3

バージョン: 1.2
日付: 2020-04-02

BECKHOFF

目次

1 序文

1.1 取扱説明書に関する注記

この説明は対応する国内規格を熟知した、トレーニングを受けた制御、オートメーションエンジニアリングの専門技術者のみの使用を対象としています。
コンポーネントのインストールとコミッショニングの際には、取扱説明書および以下の注意事項と説明に従うことが重要です。
技術者には各設置およびコミッショニングのそれぞれの時点で、発行された取扱説明書を使用する義務があります。

本製品を使用するうえでの責任者は、本製品の用途および使用方法が、関連するすべての法律、法規、ガイドラインおよび規格を含む、安全に関するすべての要件を満たしていることを確認してください。

免責事項

この取扱説明書の記載内容は、 一般的な製品説明および性能を記載したものであり、場合により記載通りに動作しないことがあります。
製品の情報・仕様は予告なく変更されます。
この説明書に記載されているデータ、図および説明に基づいて、すでに納品されている製品の変更を要求することはできません。

商標

Beckhoff®、TwinCAT®、EtherCAT®、EtherCAT G®、EtherCAT G10®、EtherCAT P®、Safety over EtherCAT®、TwinSAFE®、XFC®、XTS®およびXPlanar®は、Beckhoff Automation GmbHの登録商標です。
この取扱説明書で使用されているその他の名称は商標である可能性があり、第三者が独自の目的のために使用すると所有者の権利を侵害する可能性があります。

特許出願

EtherCAT Technologyについては、欧州特許 EP1590927、EP1789857、
EP1456722およびEP2137893、ドイツ特許DE102015105702
に記載されていますが、これらに限定されるものではありません。

EtherCAT®は、Beckhoff Automation GmbH（ドイツ)によりライセンスを受けた登録商標および特許技術です。

著作権

1.2 安全に関する指示事項

安全に関する注意事項

この取扱説明書に記載された安全に関する指示や注意事項はよくお読みになり、必ず指示に従ってください。

納入仕様

すべての製品は、用途に適した特定のハードウェア構成およびソフトウェア構成を有する状態で供給されます。ハードウェアまたはソフトウェアに取扱説明書に記載されている以外の変更を加えることは許可されていません。許可されていない変更を加えると、Beckhoff Automation GmbH & Co. KGの保証の対象外となります。

使用者の資格

この説明書は関連する国内法規を熟知した、制御およびオートメーションエンジニアリングの専門家の使用を目的としています。

安全記号の説明

この取扱説明書では、安全に関する指示や注意事項とともに以下の安全記号を使用します。安全に関する指示事項はよくお読みになり、必ず指示に従ってください。

⚠ 危険

重大な人的傷害の危険

この記号が付いた安全に関する注意事項に従わないと、人命および健康に直ちに危害を及ぼします。

⚠ 警告

人的傷害の危険

この記号が付いた安全に関する注意事項に従わないと、人命および健康に危険を及ぼします。

⚠ 注意

人的傷害の恐れ

この記号が付いた安全に関する注意事項に従わないと、人命および健康に危険を及ぼす恐れがあります。

注記

物的損害と環境汚染

この記号が付いた安全に関する注意事項に従わないと、物的損害と環境汚染をもたらす恐れがあります。

ヒントまたはアドバイス

この記号が示す情報により、さらに理解が深まります。

2 概要

PLCライブラリTc3_Moduleは、TcCOM通信に使用されます。

システム要件

ターゲットシステム	WinXP、WES、Win7、WES7、WEC7 IPCまたはCX、(x86、x64、ARM)
最小TwinCATバージョン	3.1.4020.0
最小TwinCATレベル	TC1200 TC3 PLC

3 ファンクションブロック

PLCライブラリTc3_Moduleは、TcCOM経由でモジュールからモジュールへの通信を行うためのファンクションブロックを提供します。このモジュールとは、TwinCATシステムのコンポーネント、C++オブジェクト、Matlabオブジェクト、またはPLC内のオブジェクト等になります。

3.1 TcBaseModuleRegistered

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

説明

このオブジェクトを継承することにより、ファンクションブロックによりTcCOMオブジェクトを作成することができます。オブジェクトはオブジェクトサーバに自動的に登録され、OP状態になります。

このTcCOMオブジェクトをどうのように作成し、TwinCATシステムでグローバルに使用するかについては、example [} 27] [▶ 28]で説明しています。

TcBaseModule BaseクラスはITcUnknownインターフェイスの拡張であるITComObjectインターフェイスを実装します。

ITComObjectインターフェイス

ITComObjectインターフェイスは、すべてのTwinCATモジュールに実装されます。これによりステートマシンに関する機能が使用可能になり、TwinCATシステムどうしでの情報のやり取りが可能になります。

3.1.1 TcAddRef

TcAddRef()メソッドは、参照カウンタをインクリメントし、新しい値を返します。

インターフェイス

パラメータ

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

戻り値

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

結果の参照カウント値が返されます。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.2 TcGetObjectId

メソッドTcGetObjectIdは、与えられたOTCID参照を使用してオブジェクトIDを保存します。

インターフェイス

パラメータ

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

objId: OTCID値への参照

戻り値

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

OTCID確認の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.3 TcGetObjectName

メソッドTcGetObjectNameは、与えられた長さのバッファにオブジェクト名を保存します。

インターフェイス

パラメータ

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

objName: 設定する名前

nameLen: ライトする名前の最大長

戻り値

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

名前ID確認の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.4 TcGetObjPara

TcGetObjParaメソッドは、PTCIDを使用して識別されるオブジェクトパラメータの確認を行います。

インターフェイス

パラメータ

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

pid: オブジェクトパラメータのパラメータID

nData: データの最大長

pData: データへのポインタ

pgp: 将来の拡張用に予約済み、NULLを渡す

戻り値

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

オブジェクトパラメータの確認の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.5 TcGetObjState

TcGetObjStateメソッドは、現在のオブジェクトの状態の確認を行います。

インターフェイス

パラメータ

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

pState: 状態へのポインタ

戻り値

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

状態確認の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.6 TcQueryInterface

実装されたインターフェイスへの参照のIID経由の確認。

インターフェイス

パラメータ

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

iid: インターフェイスID

pipItf: インターフェイスポインタへのポインタ。リクエストされるインターフェイスタイプが対応するインスタンスで使用可能である場合に設定されます。

戻り値

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

インターフェイス確認の成功に関する情報を提供します。
リクエストされるインターフェイスが使用可能でない場合、メソッドはADS_E_NOINTERFACEを返します。

インターフェイスポインタのリリースが必要

すべての参照は、明示的にリリースする必要があります。使用後にインターフェイスポインタのリリースを実行するために、FW SafeRelease [▶ 18]を使用することを推奨します。参照のリリースは、多くの場合、オブジェクトのデストラクタに実装されています。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.7 TcRelease

TcRelease()メソッドは、参照カウンタをデクリメントし、新しい値を返します。参照カウンタが0の場合、オブジェクトは自身を削除します。

インターフェイス

パラメータ

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

戻り値

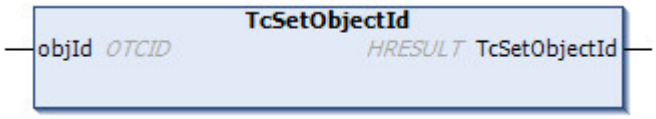
```
VAR_OUTPUT
    TcRelease : UDINT;
END_VAR
```

結果の参照カウント値が返されます。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.8 TcSetObjId



TcSetObjectIdメソッドは、与えられたOTCIDにオブジェクトのオブジェクトIDを設定します。

インターフェイス

パラメータ

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

objId: 設定するOTCID

戻り値

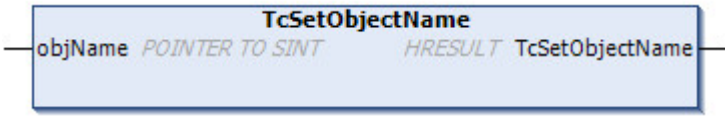
```
VAR_OUTPUT
    TcSetObjId : HRESULT;
END_VAR
```

ID変更の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.9 TcSetObjectName



TcSetObjectNameメソッドは、オブジェクトのオブジェクト名を設定します。

インターフェイス

パラメータ

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

objName: 設定するオブジェクトの名前

戻り値

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

名前変更の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.10 TcSetObjPara

TcSetObjParaメソッドは、PTCIDを使用して識別されるオブジェクトパラメータを設定します。

インターフェイス

パラメータ

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

pid: オブジェクトパラメータのパラメータID

nData: データの最大長

pData: データへのポインタ

pgp: 将来の拡張用に予約済み、NULLを渡す

戻り値

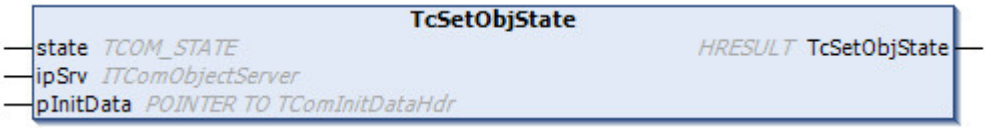
```
VAR_OUTPUT
    TcSetObjPara : HRESULT;
END_VAR
```

パラメータ変更の成功に関する情報を提供します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

3.1.11 TcSetObjState



TcSetObjStateメソッドは、特定の状態への遷移を初期化します。

インターフェイス

パラメータ

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

state: 新しい状態を表示します

ipSrv: オブジェクトの説明

pInitData: パラメータのリストへのポインタ(オプション)

戻り値:

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

状態の変更の成功に関する情報を提供します。

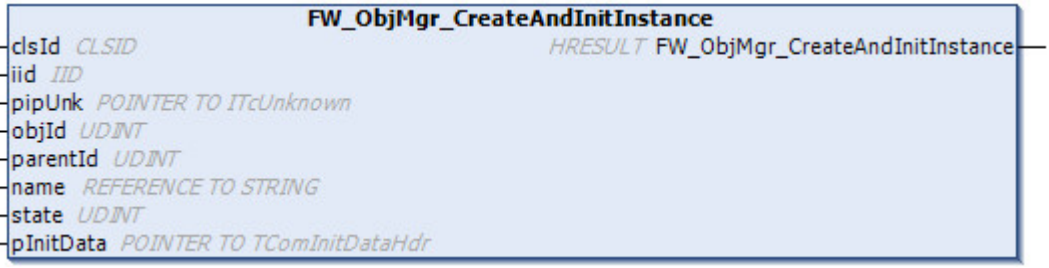
要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4 ファンクション

PLCライブラリTc3_Moduleは、TcCOM経由でモジュールからモジュールへの通信を行うための機能を提供します。このモジュールとは、TwinCATシステムのコンポーネント、C++オブジェクト、Matlabオブジェクト、またはPLC内のオブジェクト等になります。

4.1 FW_ObjMgr_CreateAndInitInstance



このファンクションは、クラスIDを使用して指定されたクラスのインスタンスを生成し、同時に、このオブジェクトへのインターフェイスポインタを返します。またオブジェクト名とオブジェクトの状態の指定に加えて、初期パラメータも指定することができます。

インターフェイス

パラメータ

```
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
```

clsId: 作成するオブジェクトのクラスを指定します。

iid: インターフェイスポインタが参照するインターフェイスIDを指定します。

pipUnk: 作成されたオブジェクトへのインターフェイスポインタを返します。

objId: 新規作成されたオブジェクトのオブジェクトIDを指定します。グローバル定数OTCID_CreateNewIdをここに入力すると、新しいオブジェクトIDが内部的に生成されます。

parentId: 親オブジェクトのオブジェクトID((オプション)。このファンクションが呼び出されるPLCインスタンスのオブジェクトIDをここで指定することができます。(TwinCAT_SystemInfoVarList._AppInfo.ObjId)

name: 新規作成されたオブジェクトに関連付けるオブジェクト名を指定します。

state: 新規作成されたオブジェクトの状態を指定します。一般的にOperational (TCOM_STATE.TCOM_STATE_OP)が指定されます。

pInitData: 初期パラメータへのポインタ(オプション)

戻り値

```
VAR_OUTPUT
    FW_ObjMgr_CreateAndInitInstance : HRESULT;
END_VAR
```

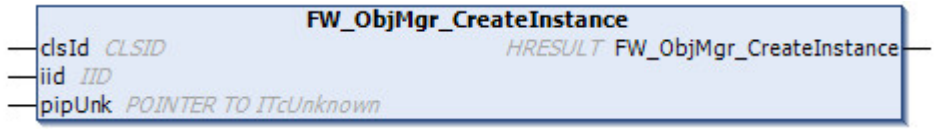
オブジェクトの削除が必要

生成されたオブジェクトは、明示的に削除する必要があります。.Netのようなガベージ コレクタはありません。生成されたインスタンスを遅くともインスタンスを生成したオブジェクトのデストラクタで削除するために、FW_ObjMgr_DeleteInstance [▶ 16]を使用することを推奨します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.2 FW_ObjMgr_CreateInstance



このファンクションは、クラスIDを使用して指定されたクラスのインスタンスを生成し、同時に、このオブジェクトへのインターフェイスポインタを返します。

インターフェイス

パラメータ

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

clsId: 作成するオブジェクトのクラスを指定します。

iid: インターフェイスポインタが参照するインターフェイスIDを指定します。

pipUnk: 作成されたオブジェクトへのインターフェイスポインタを返します。

戻り値

```
VAR_OUTPUT
    FW_ObjMgr_CreateInstance : HRESULT;
END_VAR
```

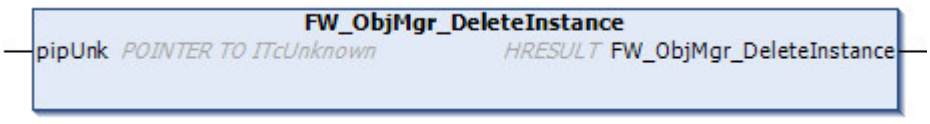
オブジェクトの削除が必要

生成されたオブジェクトは、明示的に削除する必要があります。.Netのようなガベージ コレクタはありません。生成されたインスタンスを遅くともインスタンスを生成したオブジェクトのデストラクタで削除するために、FW_ObjMgr_DeleteInstance [▶ 16]を使用することを推奨します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.3 FW_ObjMgr_DeleteInstance



このファンクションは、オブジェクトをInit状態に置きその後に、ITcUnknown.TcRelease()と同じようにオブジェクトの参照カウンタをデクリメントします、同時にインターフェイスポインタが0に設定されます。

インターフェイス

パラメータ

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

pipUnk: オブジェクトへのインターフェイスポインタのアドレスを指定します。インターフェイスポインタがヌルポインタでないかが内部的にチェックされます。

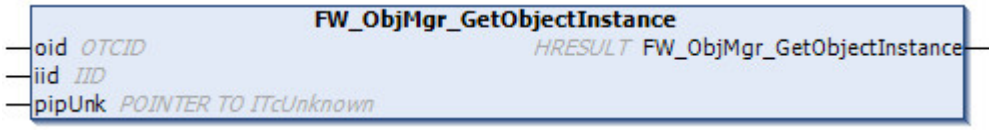
戻り値

```
VAR_OUTPUT
    FW_ObjMgr_DeleteInstance : HRESULT;
END_VAR
```

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.4 FW_ObjMgr_GetObjectInstance



このファンクションは、オブジェクトIDを使用して指定されるオブジェクトインスタンスへのインターフェイスポインタを返します。

インターフェイス

パラメータ

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

oid: オブジェクトID

iid: インターフェイスポインタが参照するインターフェイスIDを指定します。

pipUnk: 作成されたオブジェクトへのインターフェイスポインタを返します。

戻り値

```
VAR_OUTPUT
    FW_ObjMgr_GetObjectInstance : HRESULT;
END_VAR
```

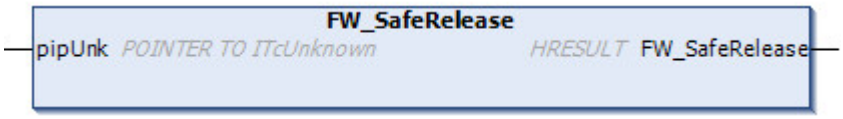
インターフェイスポインタのリリースが必要

すべての参照は、明示的にリリースする必要があります。使用後にインターフェイスポインタのリリースを実行するために、FW_SafeRelease [▶ 18]を使用することを推奨します。参照のリリースは、多くの場合、オブジェクトのデストラクタに実装されています。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.5 FW_SafeRelease



このファンクションは、ITcUnknown.TcRelease()と同じように、オブジェクトの参照カウンタをデクリメントし、それと同時にインターフェイスポインタを0に設定します。

インターフェイス

パラメータ

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

pipUnk: オブジェクトへのインターフェイスポインタのアドレスを指定します。インターフェイスポインタがヌルポインタでないかが内部的にチェックされます。

戻り値

```
VAR_OUTPUT
    FW_SafeRelease : HRESULT;
END_VAR
```

サンプル

例えば、このファンクションはオブジェクトファミリのデストラクタで呼び出すことができます。デストラクタは他のオブジェクトへのインターフェイスポインタを保持しています。

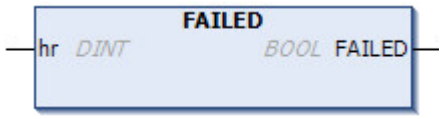
```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied
 afterwards (online change).
END_VAR
-------------------------------------------------------------------------
```

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

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.6 FAILED



タイプHRESULTのエラーコードまたはステータスコードは、このファンクションで無効性をチェックされます。

インターフェイス

パラメータ

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

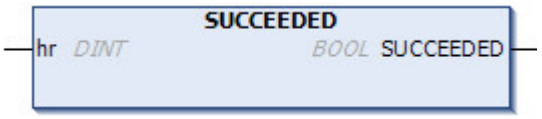
戻り値

```
VAR_OUTPUT
    FAILED : BOOL;
END_VAR
```

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.7 SUCCEEDED



タイプHRESULTのエラーコードまたはステータスコードは、このファンクションで有効性をチェックされます。

インターフェイス

パラメータ

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

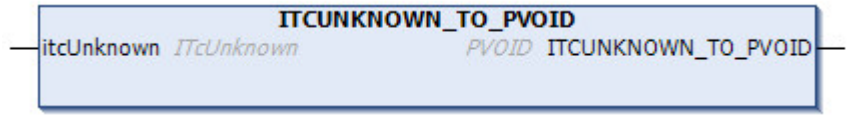
戻り値

```
VAR_OUTPUT
    SUCCEEDED : BOOL;
END_VAR
```

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.8 ITCUNKNOWN_TO_PVOID



この変換関数は、タイプITcUnknownのインターフェイスポインタをVOIDへのポインタに変換します。

インターフェイス

パラメータ

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

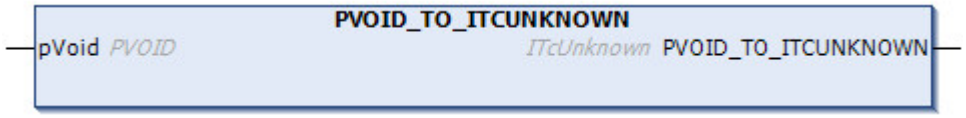
戻り値

```
VAR_OUTPUT
    ITCUNKNOWN_TO_PVOID : PVOID
END_VAR
```

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.9 PVOID_TO_ITCUKNOWN



この変換関数は、VOIDへのポインタをタイプITcUnknownのインターフェイスポインタに変換します。

インターフェイス

パラメータ

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

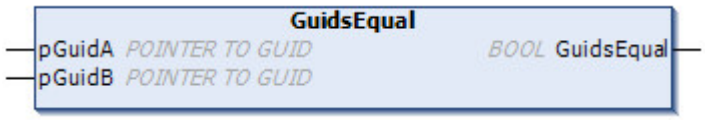
戻り値

```
VAR_OUTPUT
    PVOID_TO_ITCUKNOWN : ITcUknown;
END_VAR
```

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

4.10 GuidsEqual



ファンクションGuidsEqualは、2つのGUIDオブジェクトの相互の同等性をチェックします。

インターフェイス

パラメータ:

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

pGuidA: GUIDオブジェクトへのポインタ

pGuidB: GUIDオブジェクトへのポインタ

戻り値:

```
VAR_OUTPUT
    GuidsEqual : BOOL;
END_VAR
```

このメソッドは、両方の引数が等しいときにTRUEを返します。

要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

5 グローバル定数

5.1 GVL

```
VAR_GLOBAL CONSTANT GVL
    S_OK : DINT := 0;
    OTCID_CreateNewId : OTCID := 16#FFFFFFFF;
    OTCID_FirstFreeId : OTCID := 16#71010000;
    OTCID_LastFreeId  : OTCID := 16#710FFFFF;
END_VAR
```

名前	タイプ	値	使用	意味
S_OK	DINT	0		この定数は、HRESULTステータスコードでエラーフリー処理を指定するために使用できます。
OTCID_CreateNewId	OTCID	16#FFFFFFFF	FW_ObjMgr_CreateAndInitInstance [▶ 15]	この定数は、新しいオブジェクトIDを生成するために使用されます。
OTCID_FirstFreeId	OTCID	16#71010000		
OTCID_LastFreeId	OTCID	16#710FFFFF		

5.2 Global_Version

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

stLibVersion_Tc3_Module: Tc3_モジュールライブラリのバージョン情報(タイプ: ST_LibVersion)。

6 エラーコード

ファンクションとメソッドの戻り値は、タイプHRESULTによって出力されます。

HRESULTの上位ワード	エラーコードのグループ
16#9811	Adsエラーコード

6.1 ADSリターンコード

エラーコード: 0x000 [▶ 23]..., 0x500 [▶ 24]..., 0x700 [▶ 25]..., 0x1000 [▶ 27]...

HRESULT

HRESULT形式で出力された場合、ADSリターンコードの先頭に上位ワード16#9811 が付けられます。次に例えば、16#9811_0006として、エラーコード 'Destination port not found' (宛先ポートが見つかりません)が出力されます。

グローバルエラーコード

Hex	Dec	説明
0x0	0	エラーなし
0x1	1	内部エラー
0x2	2	Rtimeなし
0x3	3	ロックされたメモリの割り当てエラー
0x4	4	メールボックスエラーを挿入
0x5	5	間違ったHMSGの受信
0x6	6	ターゲットポートが見つかりません
0x7	7	ターゲットマシンが見つかりません
0x8	8	不明なコマンドID
0x9	9	無効なタスクID
0xA	10	IOなし
0xB	11	不明なADSコマンド
0xC	12	Win 32エラー
0xD	13	ポートが接続されていません
0xE	14	ADSの長さが無効
0xF	15	無効なAMS Net ID
0x10	16	インストールレベルが低い
0x11	17	デバッグが使用不可
0x12	18	ポートが無効
0x13	19	ポートは既に接続されています
0x14	20	ADS Sync Win32のエラー
0x15	21	ADS Syncのタイムアウト
0x16	22	ADS Sync AMSのエラー
0x17	23	ADS Syncにインデックスマップがありません
0x18	24	無効なADSポート
0x19	25	メモリなし
0x1A	26	TCP送信エラー
0x1B	27	ホストに到達できません
0x1C	28	無効なAMSフラグメント

ルータのエラーコード

Hex	Dec	名前	説明
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	ロックされたメモリは割り当てることができません
0x501	1281	ROUTERERR_RESIZEMEMORY	ルータメモリのサイズを変更できませんでした
0x502	1282	ROUTERERR_MAILBOXFULL	メールボックスが可能なメッセージの最大数に達しています。現在送信されたメッセージは拒否されました。
0x503	1283	ROUTERERR_DEBUGBOXFULL	メールボックスが可能なメッセージの最大数に達しています。 送信されたメッセージはデバッグモニタに表示されません。
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	不明なポートタイプ
0x505	1285	ROUTERERR_NOTINITIALIZED	ルータが初期化されていません
0x506	1286	ROUTERERR_PORTALREADYINUSE	要求されたポート番号は既に割り当てられています
0x507	1287	ROUTERERR_NOTREGISTERED	ポートが登録されていません
0x508	1288	ROUTERERR_NOMOREQUEUES	ポートの最大数に達しています
0x509	1289	ROUTERERR_INVALIDPORT	無効なポート
0x50A	1290	ROUTERERR_NOTACTIVATED	TwinCATルータが有効でありません

一般的なADSエラーコード

Hex	Dec	名前	説明
0x700	1792	ADSERR_DEVICE_ERROR	一般的なデバイスエラー
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	サービスはサーバではサポートされていません
0x702	1794	ADSERR_DEVICE_INVALIDGRP	無効なインデックスグループ
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	無効なインデックスオフセット
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	読み取り/書き込みが許可されていません
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	パラメータのサイズが正しくありません
0x706	1798	ADSERR_DEVICE_INVALIDDATA	無効なパラメータ値
0x707	1799	ADSERR_DEVICE_NOTREADY	デバイスがレディー状態ではありません
0x708	1800	ADSERR_DEVICE_BUSY	デバイスがビジーです
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	無効なコンテキスト(Windowsでは必要)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	メモリ不足
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	無効なパラメータ値
0x70C	1804	ADSERR_DEVICE_NOTFOUND	見つかりません(ファイル...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	コマンドまたはファイルの構文エラー
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	オブジェクトが一致しません
0x70F	1807	ADSERR_DEVICE_EXISTS	オブジェクトは既に存在しています
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	シンボルが見つかりません
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVAL	シンボルのバージョンが無効です
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	サーバが無効な状態です
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransModeはサポートされていません
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	通知ハンドルが無効です
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	通知クライアントが登録されていません
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	通知ハンドルはこれ以上ありません
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	時計のサイズが大きすぎます
0x718	1816	ADSERR_DEVICE_NOTINIT	デバイスが初期化されていません
0x719	1817	ADSERR_DEVICE_TIMEOUT	デバイスのタイムアウト
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	クエリーインターフェイスの障害
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	間違ったインターフェイスの要求
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	クラスIDが無効です
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	オブジェクトIDが無効です
0x71E	1822	ADSERR_DEVICE_PENDING	要求が保留されています
0x71F	1823	ADSERR_DEVICE_ABORTED	要求が中止されました
0x720	1824	ADSERR_DEVICE_WARNING	信号警告
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	無効な配列インデックス
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	シンボルが有効でありません
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	アクセスが拒否されました
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	ライセンスが見つかりません
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	ライセンスの期限が切れています
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	ライセンスを超えています
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	ライセンスが無効です
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	ライセンスのシステムIDが無効です
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	ライセンスの期限がありません
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	次のライセンス発行時間
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOLONG	ライセンスの期間が長すぎます
0x72c	1836	ADSERR_DEVICE_EXCEPTION	システム起動時に例外が発生しました
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	ライセンスファイルを2回読込みました
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	無効なシグネチャ
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	公開鍵の証明書
0x740	1856	ADSERR_CLIENT_ERROR	エラークラス〈クライアントエラー〉
0x741	1857	ADSERR_CLIENT_INVALIDPARM	サービスでの無効なパラメータ
0x742	1858	ADSERR_CLIENT_LISTEMPTY	ポーリングリストが空です
0x743	1859	ADSERR_CLIENT_VARUSED	var接続が既に使用されています
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	呼び出しIDが使用中です
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	タイムアウトが経過しました

Hex	Dec	名前	説明
0x746	1862	ADSERR_CLIENT_W32ERROR	win32補助システムのエラー
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	無効なクライアントタイムアウト値
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	adsポートが開いていません
0x750	1872	ADSERR_CLIENT_NOAMSADDR	ads syncの内部エラー
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	ハッシュテーブルのオーバーフロー
0x752	1874	ADSERR_CLIENT_ADDHASH	ハッシュにキーが見つかりません
0x753	1875	ADSERR_CLIENT_REMOVEHASH	キャッシュにシンボルはこれ以上ありません
0x754	1876	ADSERR_CLIENT_NOMORESYM	無効な応答を受信しました
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	syncポートがロックされています

RTimeエラーコード

Hex	Dec	名前	説明
0x1000	4096	RTERR_INTERNAL	TwinCATリアルタイムシステムでの致命的な内部エラー
0x1001	4097	RTERR_BADTIMERPERIODS	タイマ値が有効でありません
0x1002	4098	RTERR_INVALIDTASKPTR	タスクポインタに無効な値0が設定されています
0x1003	4099	RTERR_INVALIDSTACKPTR	タスクスタックポインタに無効な値0が設定されています
0x1004	4100	RTERR_PRIOEXISTS	要求タスクの優先順位は既に割り当て済みです
0x1005	4101	RTERR_NOMORETCB	使用可能なTCB(タスク制御ブロック)はこれ以上ありません。TCBの最大数は64です。
0x1006	4102	RTERR_NOMORESEMAS	使用可能なセマフォはこれ以上ありません。セマフォの最大数は64です。
0x1007	4103	RTERR_NOMOREQUEUES	使用可能なキューはこれ以上ありません。キューの最大数は64です。
0x100D	4109	RTERR_EXTIRQALREADYDEF	外部同期割り込みが既に適用されています
0x100E	4110	RTERR_EXTIRQNOTDEF	外部同期割り込みは適用されていません
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	外部同期割り込みの適用に失敗しました
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	間違ったコンテキストでのサービス機能の呼び出し
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x拡張はサポートされていません
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x拡張はシステムBIOSでは有効でありません
0x1019	4121	RTERR_VMXCONTROLSMISSING	Intel VT-x拡張でファンクションが見つかりません
0x101A	4122	RTERR_VMXENABLEFAILS	Intel VT-xの有効化に失敗しました

TCP Winsockのエラーコード

Hex	Dec	説明
0x274D	10061	接続された側が一定の時間後に適切に応答しなかったために接続に失敗したか、 または接続されたホストが応答に失敗したために確立された接続でエラーになりました。
0x2751	10065	ターゲットマシンが明確に拒否しているため、接続できませんでした。このエラーは通常、別のホストでアクティブになっていないサービス(サーバアプリケーションが存在しないサービス)に接続しようとした場合に発生します。
0x274C	10060	ホストへのルートがありません。 到達できないホストに対してソケット操作が試行されました。
		その他のWinsockエラーコード: Win32エラーコード

7 サンプル

TcCOM Sample01 [▶ 28]は、2つのPLC間でTcCOM通信がどのように行われるかを示しているサンプルです。サンプルでは1つのPLCのプロセス機能が他のPLCから直接呼び出されてます。

TcCOM Sample02 [▶ 38]は、PLCアプリケーションがTwinCAT C++クラスの既存のインスタンスの機能を使用するサンプルです。C++（またはMatlab）で書かれた別個のアルゴリズムをPLCで簡単に使用することができることを示しています。
ターゲットシステムで既存のTwinCAT C++ドライバを使用するにはTwinCAT C++ライセンスが必要ですが、ターゲットシステムまたは開発用コンピュータでのC++開発環境には必要ありません。

TcCOM Sample03サンプル [▶ 43]は、PLCアプリケーションが同時にC++クラスのインスタンスを生成することによってTwinCAT C++クラスの機能をどのように使用するかを示しています。前のサンプルと比べ、このサンプルは柔軟性が向上しています。

追加のプログラミング例は、TwinCAT 3 C++のドキュメンテーションにあります。例えば、PLCプラグラムのC++で書かれたアルゴリズム呼び出しのための追加のオプションについて記述されています(Sample11)。TcCOM_Sample02とは対照的に、PLC、.C++それぞれのインターフェイスメソッドを実装したラッパーモジュールがプログラミングされています。そのため、このバリアントはより複雑になっています。PLCアプリケーションで機能を呼び出すインターフェイスポインタの使用を断念しなければならない場合は、このバリアントが実行するためのオプションになります。

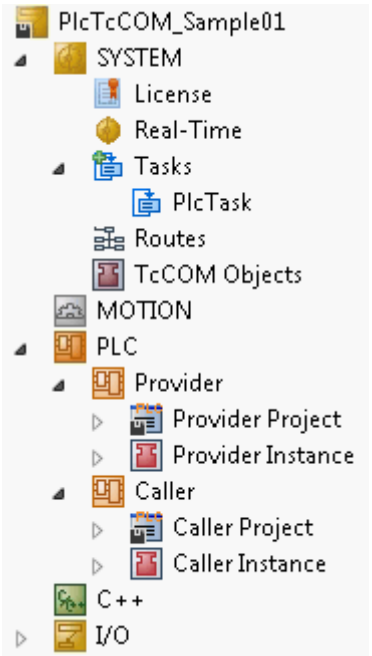
他にTwinCAT 3 C++のドキュメンテーションでは、TwinCAT C++モジュールがTcCOMインターフェイスによってどのようにPLCのファンクションブロックのメソッドを呼び出すかが示されているサンプルもあります(Sample13)。

7.1 TcCOM_Sample01_PlcToPlc

このサンプルでは、2つのPLC間のTcCOM通信について記述されています。

最初のPLC(このサンプルでは「提供側」とも呼ばれる)のファンクションブロックによって提供される機能は、2番目のPLC(このサンプルでは「呼び出し側」とも呼ばれる)から呼び出されます。このためには、ファンクションブロックまたはそのプログラムコードをコピーする必要はありません。その代わりに、プログラムは最初のPLCのオブジェクトインスタンスに直接働きかけます。

2つのPLCは、1つのTwinCATランタイム内にある必要があります。これに関連して、ファンクションブロックは、グローバルに定義されたインターフェイス経由でのシステム全体のメソッドを提供します、またそれ自体がTcCOMオブジェクトを表します。すべてのTcCOMオブジェクトと同様に、そのようなファンクションブロックも、「TcCOMオブジェクト」ノードとして実行時は登録されます。



この手順は、以下のサブチャプターに説明されています。

1. 1つ目のPLCでグローバルに機能を提供するFBを作成 [▶ 29]
2. 2つ目のPLCで単純なプロキシとしてこの機能を提供するFBを作成 [▶ 34]
3. サンプルプロジェクトの実行 [▶ 37]

サンプルのダウンロード: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/zip/2343046667.zip

マルチタスク(マルチスレッド)を使用する場合のレース条件

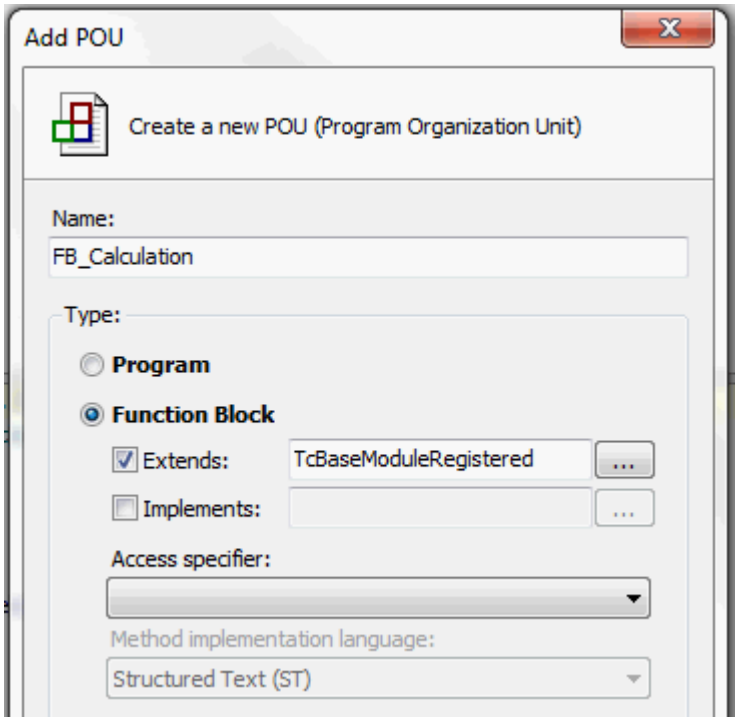
グローバルに機能を提供するファンクションブロックは、最初のPLCでインスタンス化されます。これは、他のファンクションブロックと同じようにそこで使用できます。加えて、別のPLCから(または、例えば、C++モジュールから)使用される場合は、提供されるメソッドがスレッドセーフであることを確認してください。これは、システム構成に応じて、さまざまな呼び出しが異なるタスクコンテキストから同時に発生し、または相互に割り込みが発生する可能性があるためです。この場合、メソッドは最初のPLCのファンクションブロックまたはグローバル変数のメンバ変数にアクセスしてはいけません。必要である場合は、同時アクセスを行わないようにしてください。Tc2_SystemライブラリのファンクションTestAndSet()使用してチェックしてください。

システム要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64、ARM	Tc3_Module

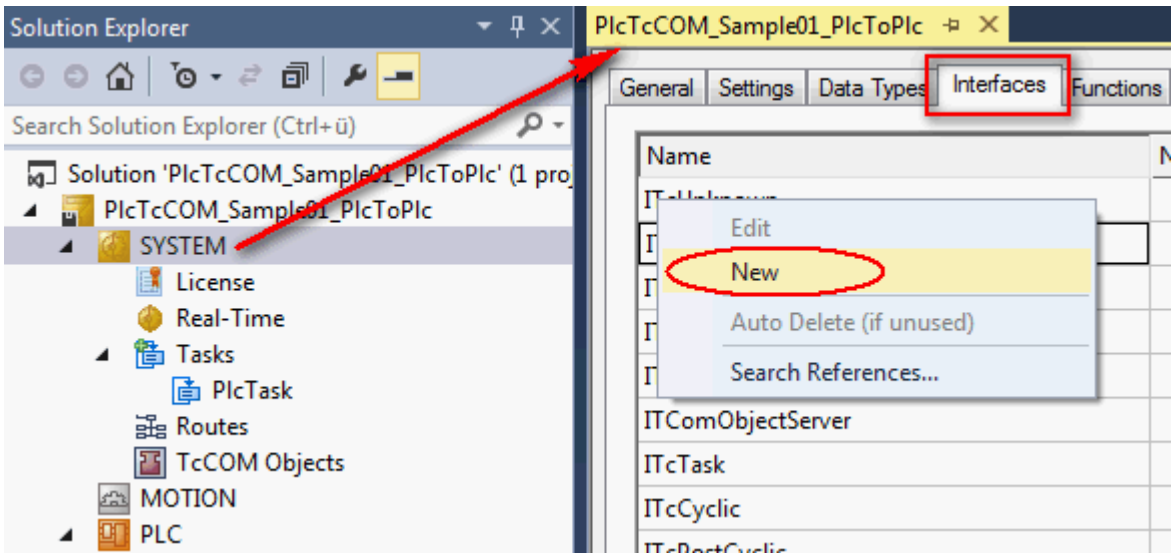
7.1.1 最初のPLCで機能をグローバルに提供するFBを作成

1. PLCを作成し、新しいファンクションブロック(FB) (ここでは: FB_Calculation)を準備します。TcBaseModuleRegistered [▶ 8]クラスからファンクションブロックを導出し、このファンクションブロックのインスタンスが同じPLCで使用可能であるだけでなく、2番目のPLCからの到達できるようにします。
   ヒント: 別の方法として、既存のPLCのFBを修正することもできます。

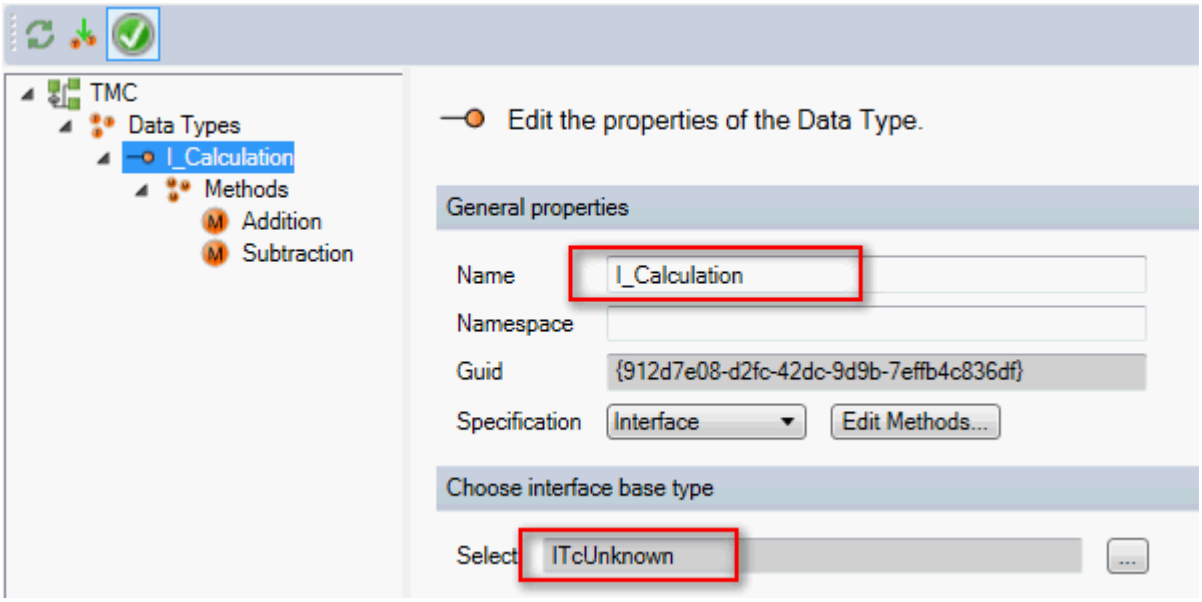


2. ファンクションブロックは、メソッドを使用することによりその機能を提供する必要があります。これらはグローバルインターフェイスで定義されています。グローバルインターフェイスのタイプはシステム全体で適用され、プログラミング言語に関わりなく認識されます。グローバルインターフェイスを作成するには、システムプロパティの[Interface]タブのコンテキストメニューを開き、オプション[New]を選択します。

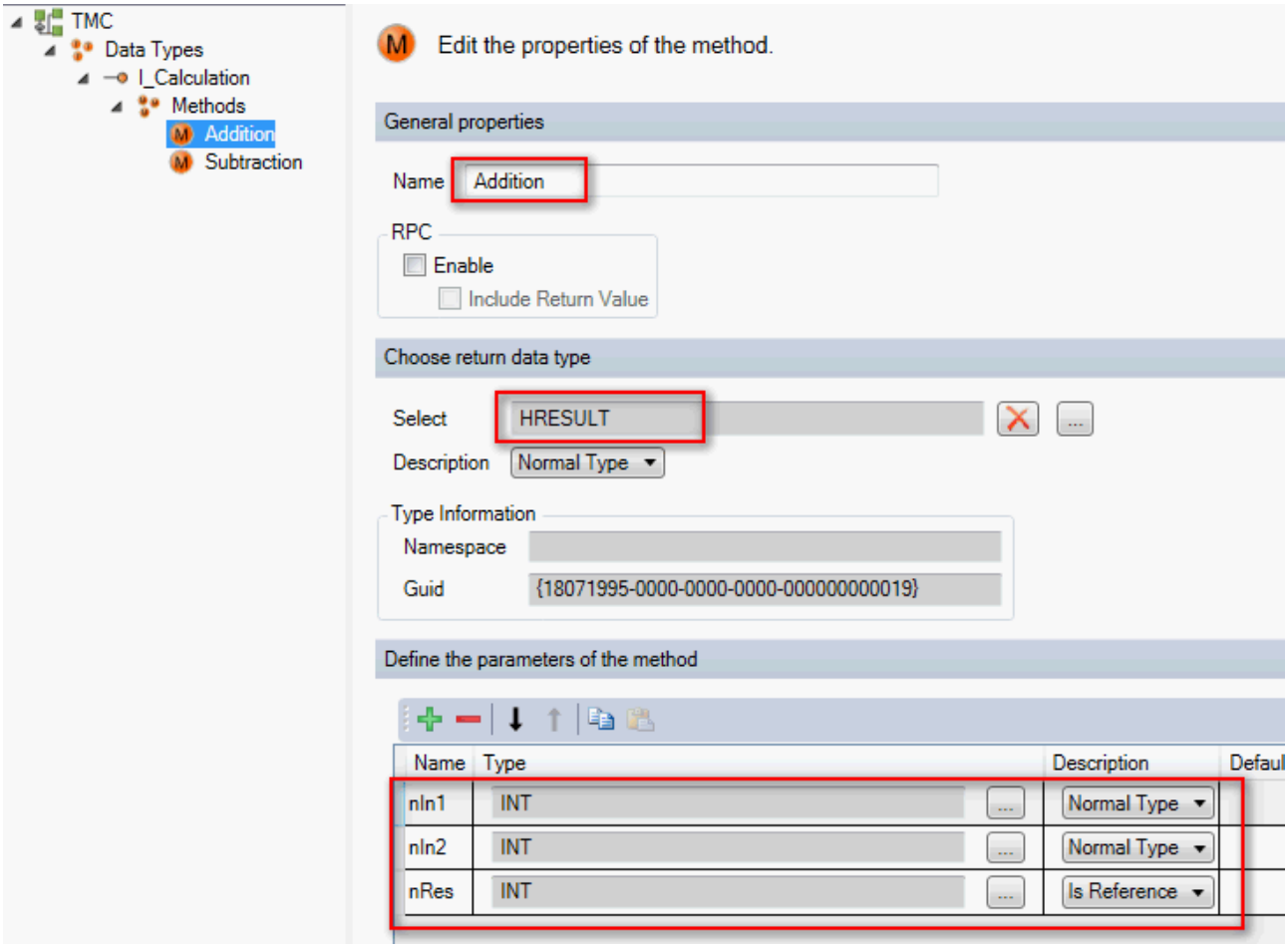
   ⇨ TMCエディタが開き、そこでグローバルインターフェイスを作成します。



3. 名前(ここでは: I_Calculation)を指定し、必要なメソッドを追加します。インターフェイスは、TwinCAT TcCOMモジュールコンセプトを満たすためにITcUnknownに自動で選択されます。



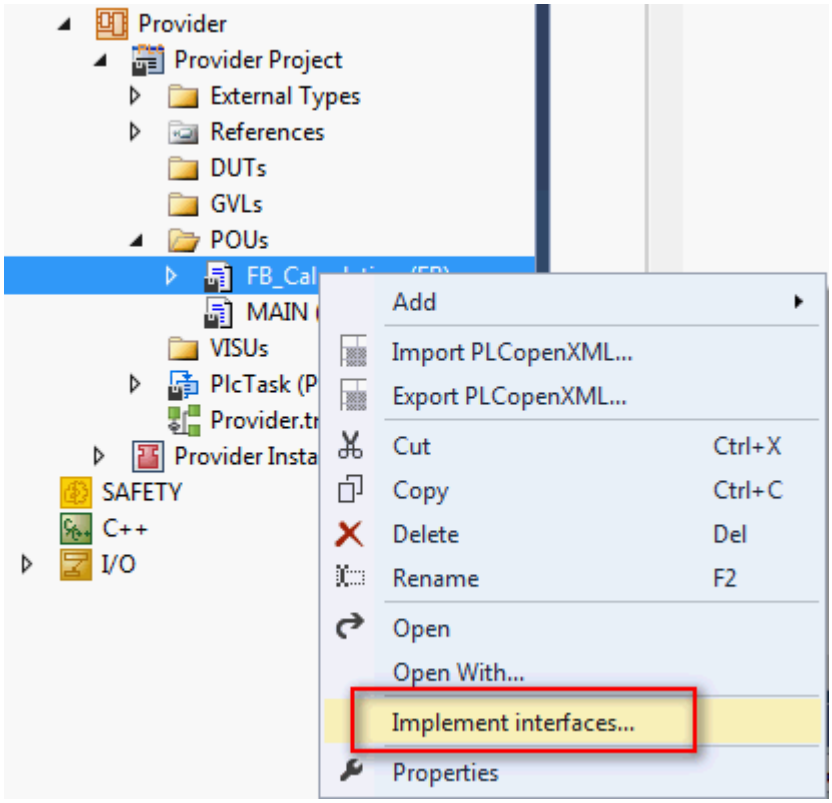
4. 同様にメソッドの名前を指定し(ここでは: Addition()とSubtraction())、戻りデータ型としてHRESULTを選択します。この戻りタイプは、このタイプのTcCOM通信が実装される場合は必須です。
5. 最後にメソッドパラメータを指定してから、TMCエディタを閉じます。



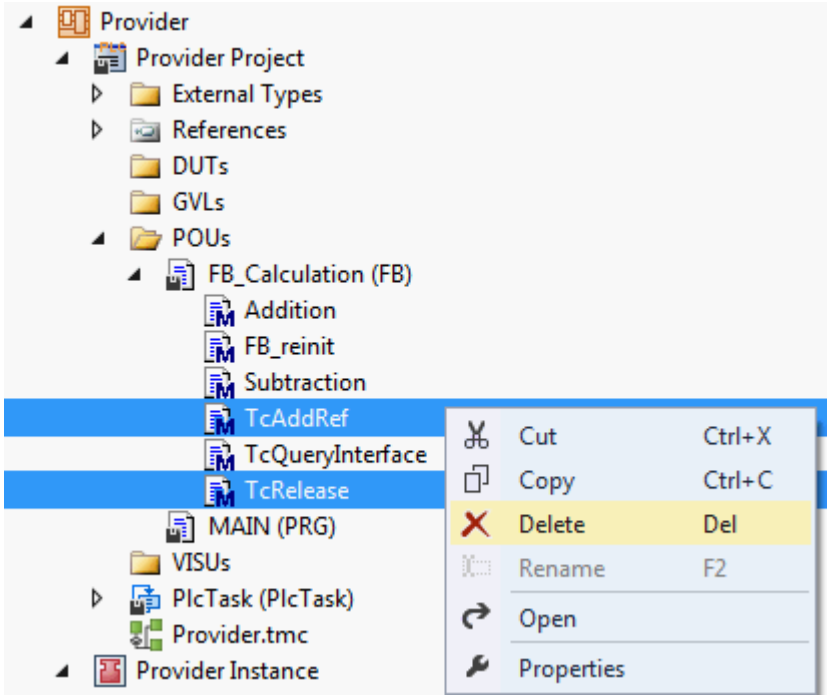
6. これで、I_Calculationインターフェイスが FB_Calculationファンクションブロックに実装され、c++_compatible属性が追加されます。

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

7. このインターフェイスに属するメソッドを取得するために、ファンクションブロックのコンテキストメニューで[Implement interfaces...]オプションを選択します。



8. ベースクラスの既存の実装が使用されるため、2つのメソッドTcAddRef() およびTcRelease()を削除します。



9. FB_Calculationファンクションブロックに FB_reinit()メソッドを作成し、基本実装を呼び出します。これにより、ベースクラスのFB_reinit()メソッドがオンラインでの変更時に実行されます。これは必須です。

FB_Calculation.FB_reinit

```
METHOD FB_reinit : BOOL
VAR_INPUT
END_VAR


```

```
SUPER^.FB_reinit();

```

10. インターフェイスITcUnknown [▶ 67]のTcQueryInterface()メソッドを実装します。このメソッドを介して、他のTwinCATコンポーネントがこのファンクションブロックのインスタンスへのインターフェイスポインタを取得し、それによりメソッドの呼び出しを起動することが可能になります。ファンクションブロックまたはそのベースクラスがiid(インターフェイスID)でリクエストしたインターフェイスを返答した場合に、TcQueryInterfaceの呼び出し成功になります。この場合、渡されたインターフェイスポインタは、タイプが変更されたファンクションブロックのアドレスに割り当てられ、参照カウンタがTcAddRef()によりインクリメントされます。

FB_Calculation.TcQueryInterface

```
{attribute 'object_name' := 'TcQueryInterface'}
{attribute 'c++_compatible'}
{attribute 'signature_flag' := '33554688'}
{attribute 'pack_mode' := '4'}
{attribute 'show'}
{attribute 'minimal_input_size' := '4'}
{attribute 'checksuperglobal'}
METHOD TcQueryInterface : HRESULT
VAR_INPUT
    iid : REFERENCE TO IID;
    pipItf  : POINTER TO PVOID;
END_VAR

VAR
    ipCalc : I_Calculation;
END_VAR
```

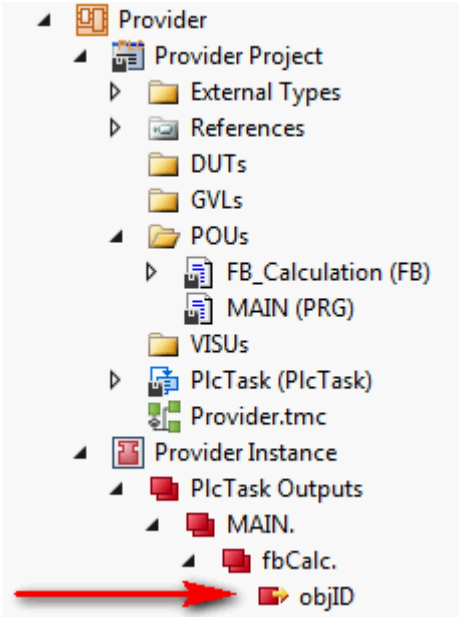
```
IF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_I_Calculation)) THEN
    ipCalc := THIS^; // cast to interface pointer
    pipItf^ := ITCUNKNOWN_TO_PVOID(ipCalc);
    TcAddRef();
    TcQueryInterface := S_OK;
ELSE
    TcQueryInterface := SUPER^.TcQueryInterface(iid, pipItf);
END_IF

```

11. 2つのメソッドAddition()およびSubtraction()nRes := nIn1 + nIn2およびnRes := nIn1 - nIn2を作成、対応するコードを入力して、機能を追加します。
12. このファンクションブロックの1つまたは複数のインスタンスをMAINプログラムモジュールまたはグローバル変数リストに追加します。

    ⇨ 1つ目のPLCでの実装が完了しました。

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

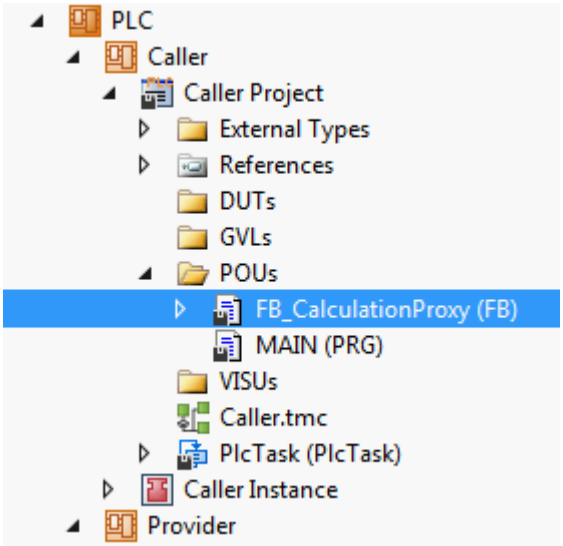
⇨ PLCをコンパイルした後、FB_Calculationのインスタンスを表すTcCOMオブジェクトのオブジェクトIDが、プロセスイメージの出力として使用可能になります。



7.1.2 同様に、2つ目のPLCで単純なプロキシとしてこの機能を提供するFBを作成

1. PLCを作成し、新しいファンクションブロックをそこに追加します。

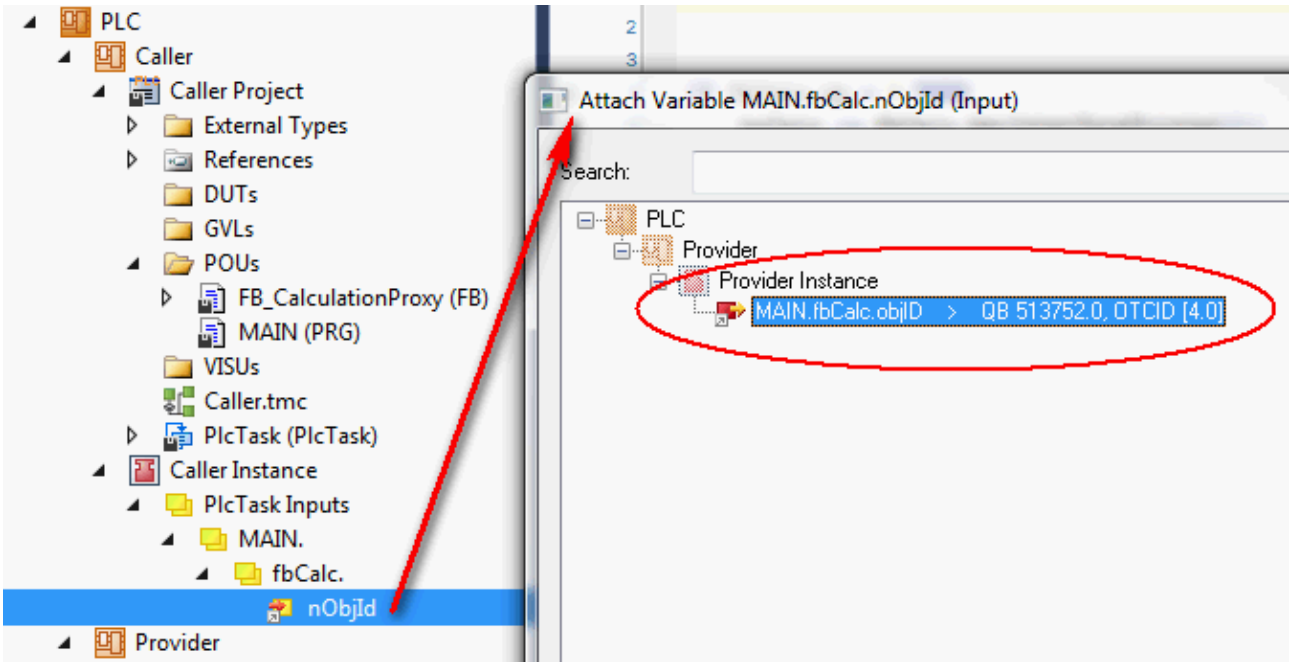
   ⇨ このプロキシファンクションブロックは、1つ目のPLCでプログラミングされた機能を提供します。これは、グローバルインターフェイスI_Calculationのタイプのインターフェイスポインタを介して行われます。



2. ファンクションブロックの宣言部分で、出力としてグローバルインターフェイスへのインターフェイスポインタを宣言します。このインターフェイスポインタは後に機能を外部に提供します。

```
FB_CalculationProxy
1  FUNCTION_BLOCK FB_CalculationProxy
2  VAR_OUTPUT
3      ip : I_Calculation;
4  END_VAR
5
6  VAR
7      {attribute 'displaymode':='hex'}
8      nObjId AT%I* : OTCID;    // Instance configured to be retrieved
9      iid : IID := TC_GLOBAL_IID_LIST.IID_I_Calculation;
10 END_VAR
11
```

3. さらに、ローカルメンバ変数としてオブジェクトIDとインターフェイスIDを作成します。
インターフェイスIDはグローバルリストを介して既に使用可能なので、オブジェクトIDはプロセスイメージのリンクを介して割り当てられます。



4. PLCプロキシファンクションブロックを実装します。最初にGetInterfacePointer()メソッドをファンクションブロックに追加します。
インターフェイスポインタは、FW_ObjMgr_GetObjectInstance() [▶ 17]ファンクションの助けを借りて、指定されたTcCOMオブジェクトの指定されたインターフェイスに取得されます。これは、オブジェクト IDが有効であり、インターフェイスポインタが既に割り当てられていない場合にのみ実行されます。オブジェクト自体は、参照カウンタをインクリメントします。

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. 使用された参照を再度使用するためにはリリースが必須となります。この目的で、ファンクションブロックのFB_exitデストラクタでFW_SafeRelease()ファンクションを呼び出します。

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
```

⇨ これによって、プロキシファンクションブロックの実装が完了します。

6. アプリケーションでプロキシファンクションブロックFB_CalculationProxyをインスタンス化し、そのメソッドGetInterfacePointer()を呼び出して、有効なインターフェイスポインタを取得します。プロキシブロックのインスタンスは、インターフェイス経由で提供されるメソッドを呼び出すために、アプリケーション内で宣言されます。呼び出し自体は、ファンクションブロックの出力として定義されたインターフェイスポインタを全面的に引き継ぎます。一般的なポインタと同様、事前のヌルチェックを行う必要があります。その後にメソッドを直接呼び出すができるようになり、またIntellisense経由でも呼び出すことができます。

MAIN*

```
PROGRAM MAIN
VAR
    fbCalc : FB_CalculationProxy;
    hrCalc : HRESULT;
    a : INT := 10;
    b : INT := 7;
    nSum : INT; // a + b
    nDiff : INT; // a - b
END_VAR

IF fbCalc.ip = 0 THEN
    hrCalc := fbCalc.GetInterfacePointer();
END_IF
IF fbCalc.ip <> 0 THEN
    hrCalc := fbCalc.ip.Addition(a,b,nSum);
    hrCalc := fbCalc.ip.Subtraction(a,b,nDiff);
END_IF
```

⇨ サンプルはテストの準備ができています。

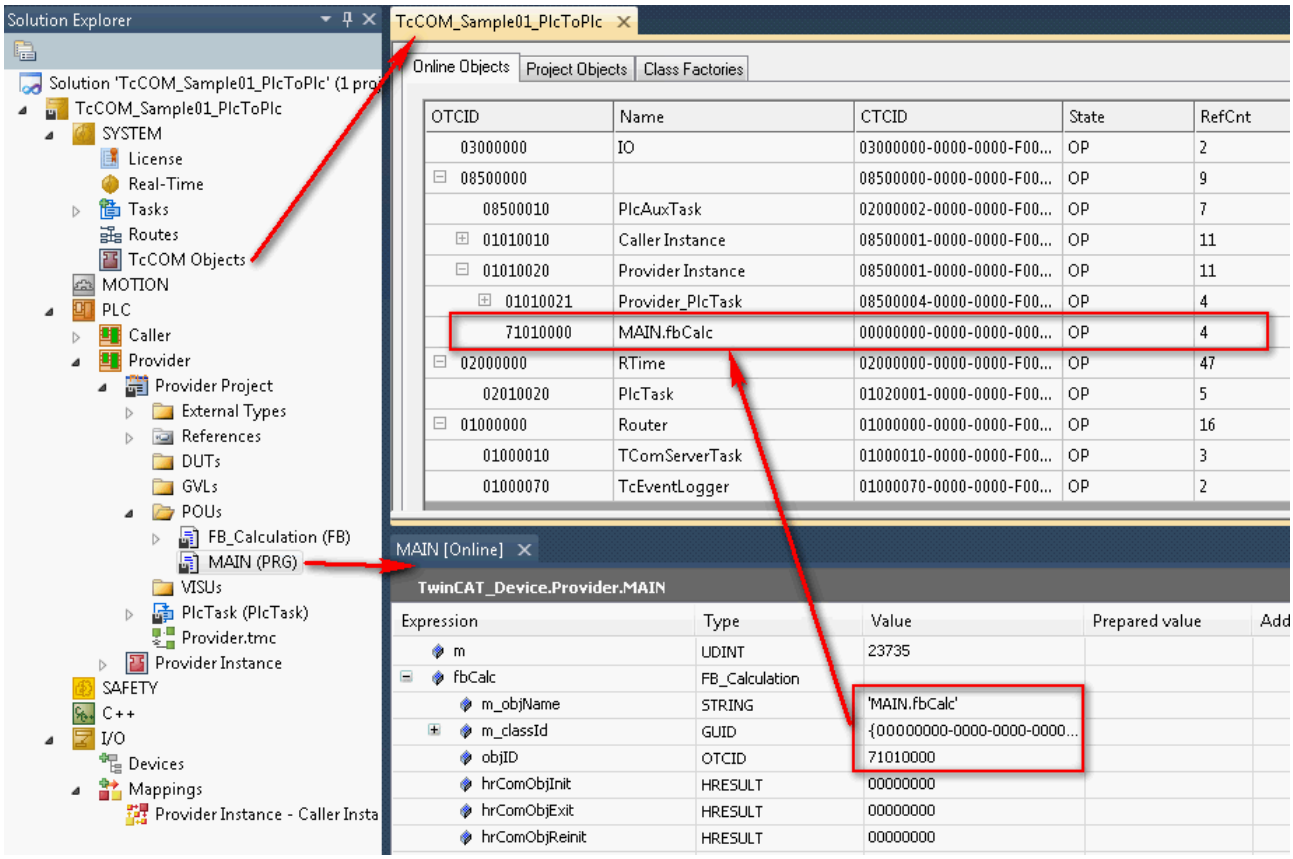
不適切な順序

この実装では、2つのPLCのどちらを先に開始するかによっては不適切なケースもあります。

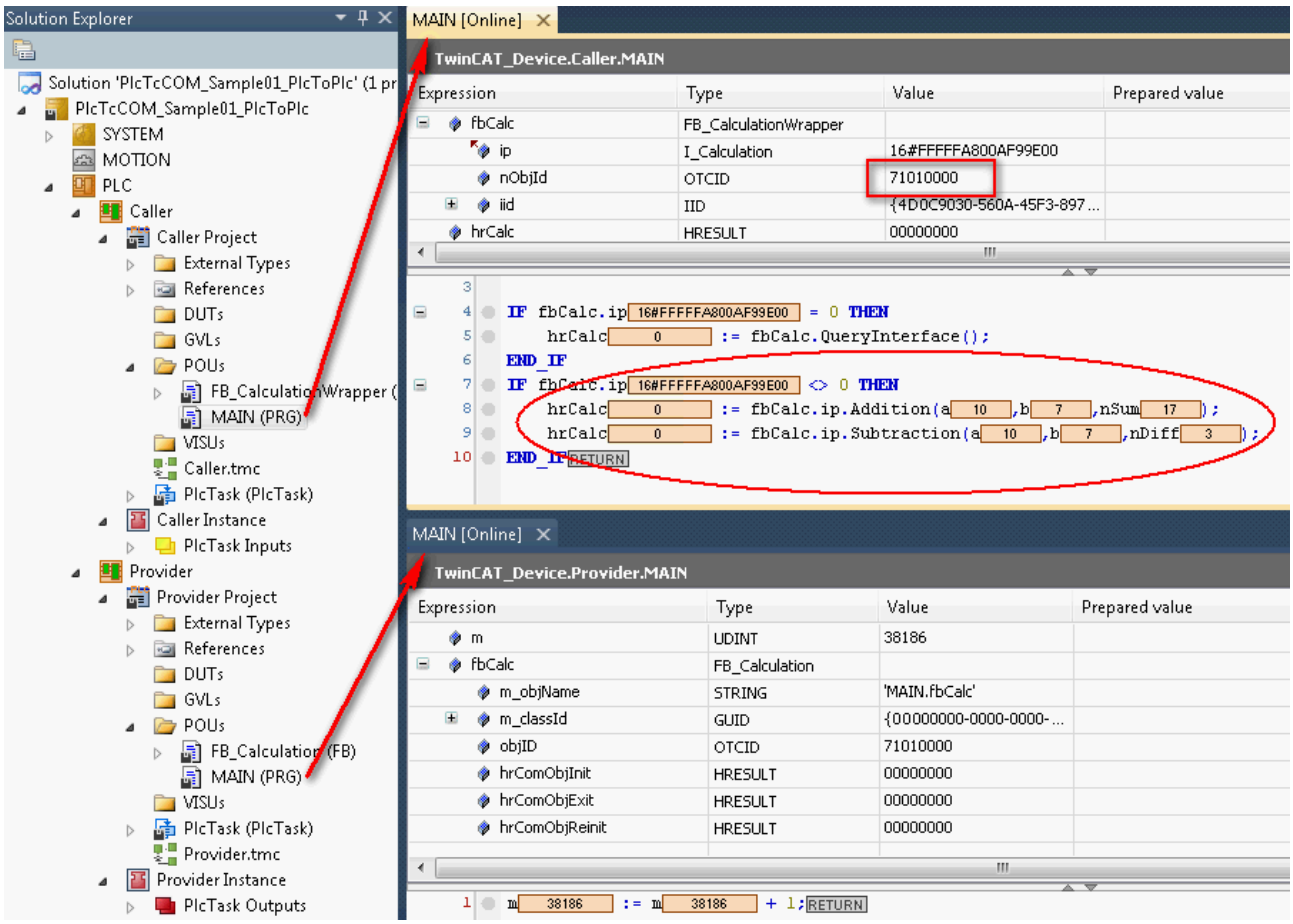
7.1.3 サンプルプロジェクトの実行

1. ターゲットシステムを選択し、プロジェクトをコンパイルします。
2. TwinCATの構成を有効にし、ログインを実行して両方のPLCを開始します。

   ⇨ PLCアプリケーション「提供側」のオンラインビューで、C++オブジェクトの生成されたオブジェクトIDが、PLCファンクションブロックFB_Calculationで確認できます。プロジェクトノード“TcCOM Objects”は、生成されたオブジェクトをそのオブジェクトIDと選択された名前とともにリストに登録します。



   ⇨ PLCアプリケーション「呼び出し側」のオンラインビューでは、プロキシファンクションブロックがプロセスイメージを介して同じオブジェクトIDを割り当てられています。インターフェイスポインタは有効な値をもち、メソッドが実行されます。



7.2 TcCOM_Sample02_PlcToCpp

このサンプルでは、PLCとC++との間のTcCOM通信が説明されています。接続の中で、PLCアプリケーションはTwinCAT C++クラスの既存のインスタンスの機能を使用します。このように、C++で書かれた自身のアルゴリズムをPLCで簡単に使用することができます。ターゲットシステムにおいて既存のTwinCAT C++ドライバを使用する場合はTwinCAT C++ライセンスが必要になりますが、
ターゲットシステムや開発や開発環境にはC++開発環境は必要ありません。

ビルド済みのC++ドライバは、インターフェイスがTMC記述ファイルに保存され、それによってPLCで認識される1つまたは複数のクラスを提供します。

この手順は、以下のサブチャプターに説明されています。

1. TwinCAT TcCOMオブジェクトとしてTwinCAT++クラスをインスタンス化 [▶ 39]
2. PLCでFBを作成、PLCは単純なラッパとしてC++オブジェクトの機能を提供 [▶ 39]
3. サンプルプロジェクトの実行 [▶ 42]

サンプルのダウンロード: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/zip/2343048971.zip

システム要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64	Tc3_Module

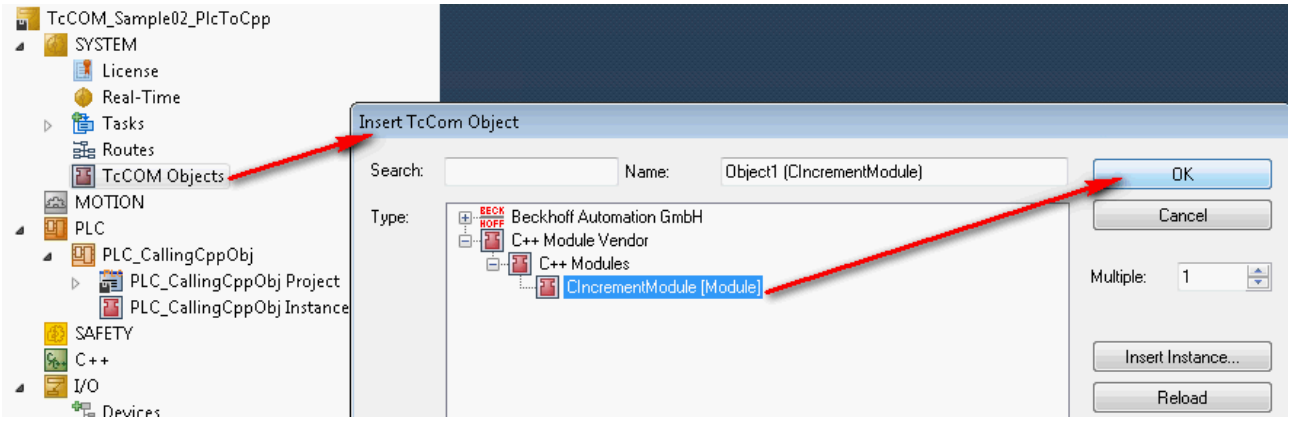
7.2.1 TwinCAT TcCOMオブジェクトとしてTwinCAT++クラスをインスタンス化

TwinCAT C++ドライバがターゲットシステムで使用可能である必要があります。TwinCATはこの目的のためにデプロイメントを提供し、コンポーネントは開発用コンピュータ上に適切に保存するだけで済みます。

既存のTwinCAT C++ドライバとそのTMC記述ファイルもドライバのアーカイブとして使用可能です。このアーカイブ(IncrementerCpp.zip)を以下のフォルダに解凍します:
C:¥TwinCAT¥3.1¥CustomConfig¥Modules¥IncrementerCpp¥

TwinCATのデプロイメントは、構成のアクティベーション後にファイルをターゲットシステムの以下のフォルダにコピーします:
C:¥TwinCAT¥3.1¥Driver¥AutoInstall¥

1. TwinCATプロジェクトを開くか、またはプロジェクトを新規作成します。
2. ソリューションエクスプローラーの“TcCOM Objects”下に、クラスCIncrementModuleのインスタンスを追加します。

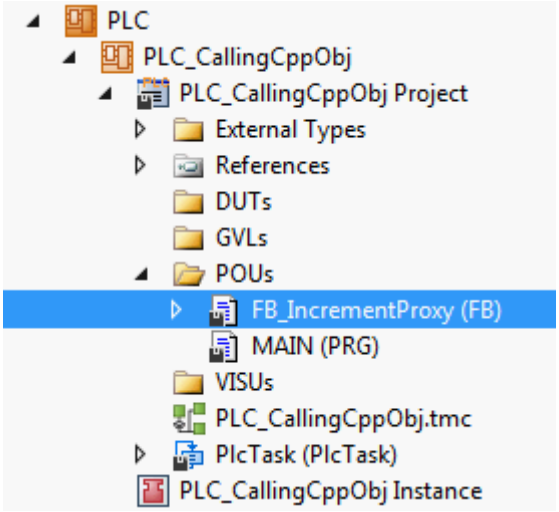


C++ドライバの作成

TwinCAT C++のドキュメンテーションには、TwinCAT用のC++ドライバの作成方法に関する詳細な説明があります。
前述のドライバアーカイブを作成するには、ドライブの作成時に最後のステップとしてC++プロジェクトのコンテキストメニューから[Publish TwinCAT Modules]を選択します。

7.2.2 PLCで単純なプロキシとしてC++オブジェクトの機能を提供するFBを作成する

1. PLCを作成し、新しいファンクションブロックをそこに追加します。

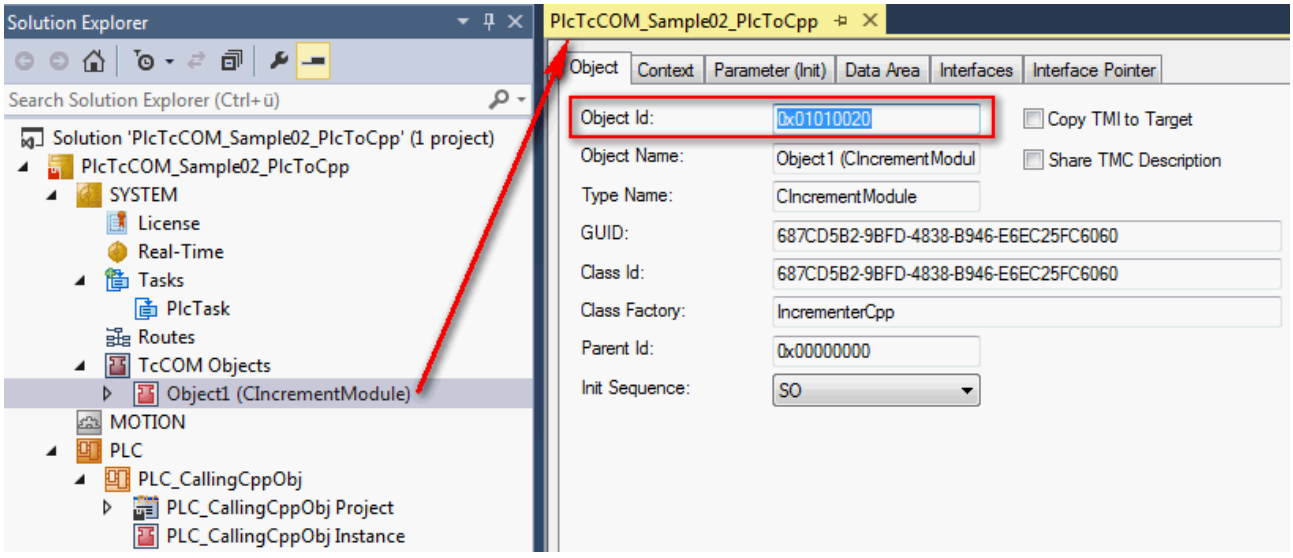


⇨ このプロキシブロックは、C++でプログラミングされた機能を提供します。これは、C++クラスで定義されTMC記述ファイルによりPLCで認識されるインターフェイスポインタを介して実行できます。

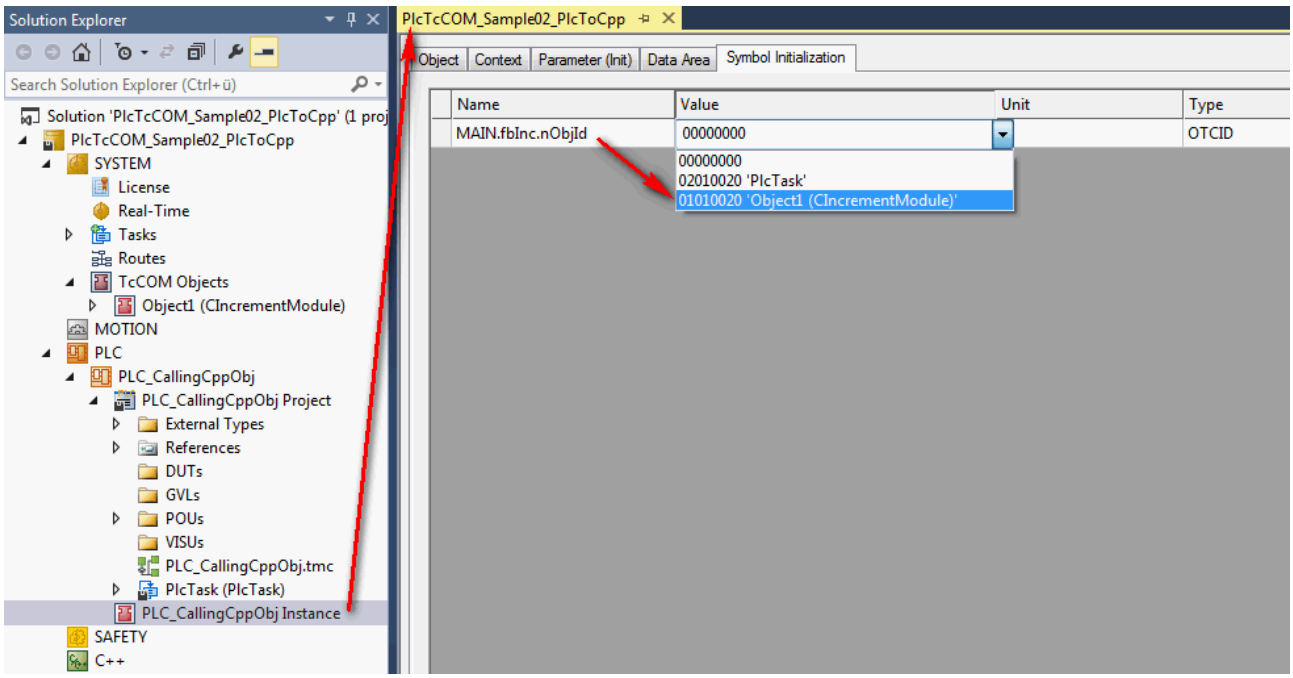
2. ファンクションブロックの宣言部分で、出力としてインターフェイスへのインターフェイスポインタを宣言します。このインターフェイスポインタは後に機能を外部に提供します。
3. ローカルメンバ変数としてオブジェクトIDとインターフェイスIDを作成します。
インターフェイスIDがグローバルリストを介して既に使用可能であるのに対して、オブジェクトIDはTwinCATのシンボル初期化を介して割り当てられます。TcInitSymbol属性によって、外部シンボル初期化用リストに変数が表示されるようになります。作成されたC++オブジェクトのオブジェクトIDを割り当てる必要があります。

```
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
```

⇨ オブジェクトIDは、TcCOM Objectsノード下のオブジェクトの選択に表示されます。



⇨ TcInitSymbol属性が使用された場合でも、シンボル初期化のリストは、[Symbol Initialization]タブのPLCインスタンスのノードに配置されます。ここでドロップダウンを使用することにより、既存のオブジェクトIDを変数のシンボル名に割り当てます。この値は、PLCランタイムの前に定義できるようにPLCがダウンロードされたときに割り当てられます。新しいシンボル初期化または変更は、新しいPLCのダウンロードとともに状況に応じて入力されます。



ヒント: 別の方法として、オブジェクトIDの引き渡しは、最初のサンプル(TcCOM Sample01 PlcToPlc [▶ 28])で実装したようにプロセスイメージとのリンクを使用することによっても実装できます。

4. PLCプロキシブロックを実装します。
   最初に、FB_initコンストラクタメソッドをファンクションブロックに追加します。オンラインでの変更という目的よりむしろファンクションブロックの初期化の時に、指定されたTcCOMオブジェクトの指定されたポインタに対するインターフェイスポインタは、ファンクション FW_ObjMgr_GetObjectInstance() [▶ 17]により取得されます。この時に、オブジェクト自体は参照カウンタをインクリメントします。

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
```

5. 使用された参照を再度使用するためにはリリースが必須となります。この目的で、ファンクションブロックのFB_exitデストラクタでFW_SafeRelease()ファンクション [▶ 18]を呼び出します。

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
```

⇨ これによって、プロキシファンクションブロックの実装が完了します。

6. インターフェイス経由で提供されるメソッドを呼び出すために、プロキシブロックのインスタンスをアプリケーション内で宣言します。
呼び出し自体は、ファンクションブロックの出力として定義されたインターフェイスポインタを全面的に引き継ぎます。一般的なポインタと同様、事前のヌルチェックを行う必要があります。その後にメソッドを直接呼び出すができるようになり、またIntellisense経由でも呼び出すことができます。

MAIN*

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

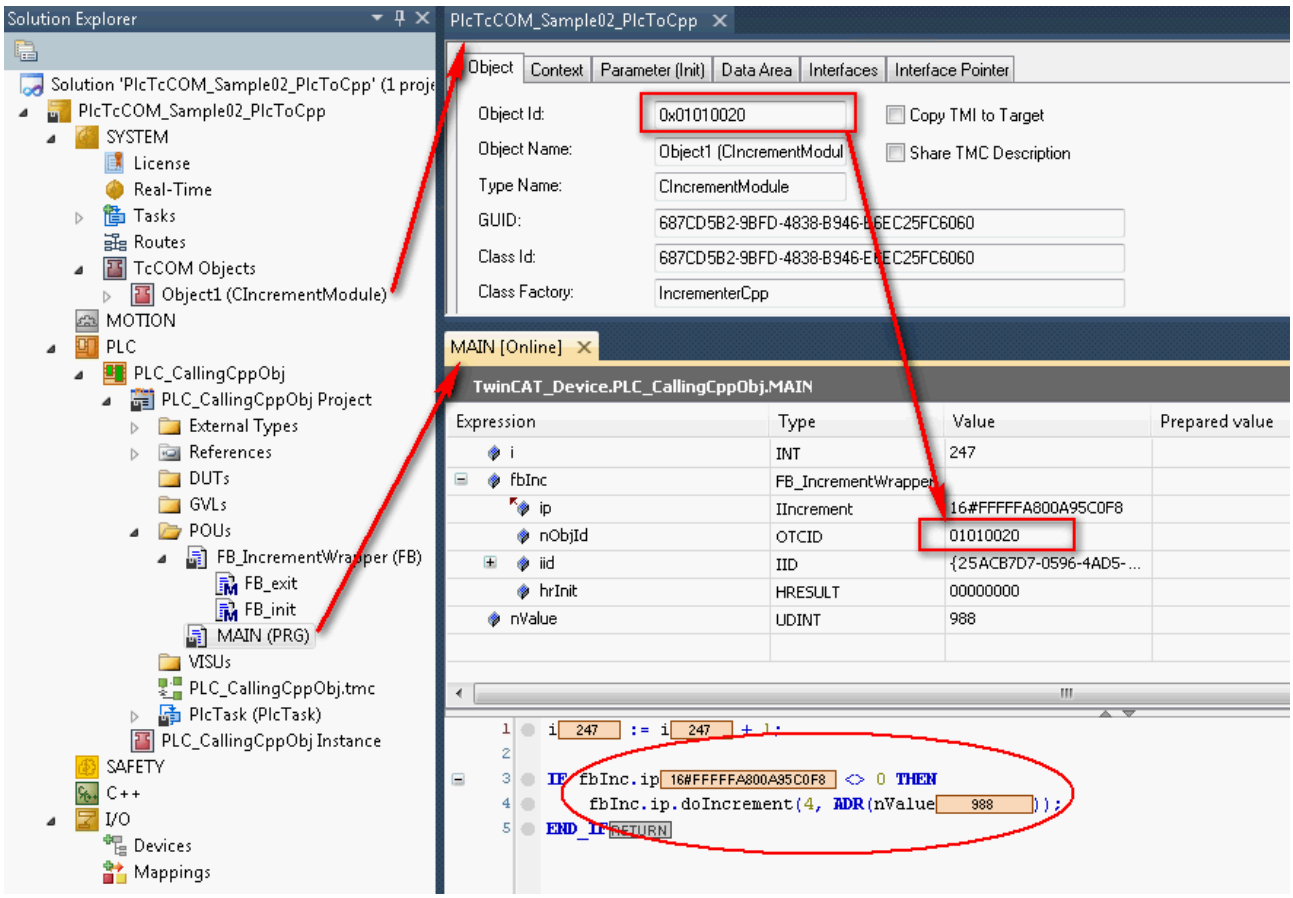
```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(4, ADR(nValue));
END_IF
```

⇨ サンプルのテスト準備が完了です

7.2.3 サンプルのテスト準備が完了です

1. ターゲットシステムを選択し、プロジェクトをコンパイルします。
2. TwinCATの構成を有効にし、ログインを実行してPLCを開始します。

⇨ PLCアプリケーションのオンラインビューで、C++オブジェクトの割り当てられたオブジェクトIDが、PLCプロキシファンクションブロックで確認できます。インターフェイスポインタは有効な値をもち、メソッドが実行されます。



7.3 TcCOM_Sample03_PlcCreatesCpp

Sample02と同様、この例では、PLCとC++との間のTcCOM通信が説明されています。この目的で、PLCアプリケーションはTwinCAT C++クラスの機能を使用します。C++クラスの必要なインスタンスは、この例ではPLC自体によって作成されます。このように、C++で書かれた自身のアルゴリズムをPLCで簡単に使用することができます。
ターゲットシステムでTwinCAT C++ライセンスが必要な既存のTwinCAT C++ドライバを使用する場合でも、ターゲットシステムまたは開発用コンピュータではC++開発環境は必要ありません。

ビルド済みのC++ドライバは、インターフェイスがTMC記述ファイルに保存され、それによってPLCで認識される1つまたは複数のクラスを提供します。

この手順は、以下のサブチャプターに説明されています。

1. TwinCAT C++ドライバおよびそのクラスの提供 [▶ 44]
2. C++オブジェクトを作成し、その機能を提供するPLCでFBを作成 [▶ 44]
3. サンプルプロジェクトの実行 [▶ 47]

サンプルのダウンロード: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/zip/2343051531.zip

システム要件

TwinCATバージョン	ハードウェア	必要ライブラリ
TwinCAT 3.1、ビルド4020	x86、x64	Tc3_Module

7.3.1 TwinCAT C++ドライバおよびそのクラスの提供

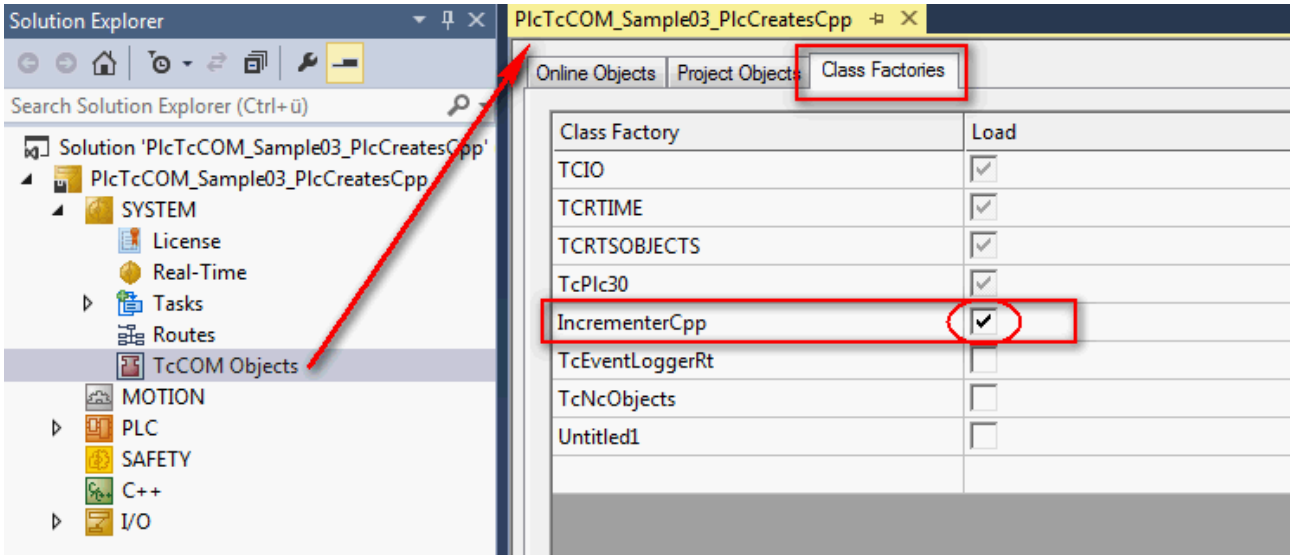
TwinCAT C++ドライバがターゲットシステムで使用可能である必要があります。TwinCATはこの目的のためにデプロイメントを提供し、コンポーネントは開発用コンピュータ上に適切に保存するだけで済みます。

既存のTwinCAT C++ドライバとそのTMC記述ファイルもドライバのアーカイブとして使用可能です。このアーカイブ(IncrementerCpp.zip)を以下のフォルダに解凍します:
C:¥TwinCAT¥3.1¥CustomConfig¥Modules¥IncrementerCpp¥

TwinCATのデプロイメントは、構成のアクティベーション後にファイルをターゲットシステムの以下のフォルダにコピーします:
C:¥TwinCAT¥3.1¥Driver¥AutoInstall¥

1. TwinCATプロジェクトを開くか、またはプロジェクトを新規作成します。
2. TcCOM Objectsノード下にある[Class Factories]タブのソリューションで必要なC++ドライバを選択します。

⇨ これにより、TwinCATの起動時にドライバがターゲットシステムにロードされます。さらに、この選択は記述されたデプロイメントに提供されます。



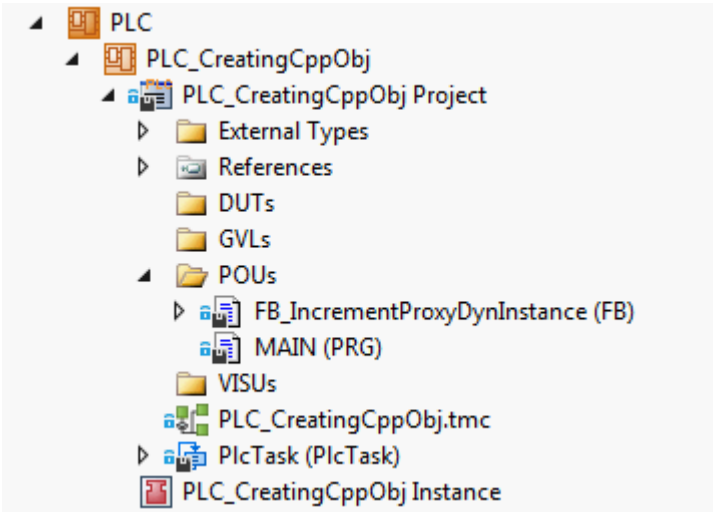
C++ドライバの作成

TwinCAT C++のドキュメンテーションには、TwinCAT用のC++ドライバの作成方法に関する詳細な説明があります。
Sample03では、クラスの動的なインスタンス化が想定されるためTwinCAT C++ドライバは、[TwinCAT Module Class for RT Context]として定義される必要があることに注意してください。C++ Wizardは、この目的のために専用のテンプレートを提供します。
加えて、このサンプルは、TcCOMinitializationデータとTcCOMパラメータを使用せずに管理を行うTwinCAT C++クラスを使用します。

7.3.2 C++オブジェクトを作成し、その機能を提供するPLCでFBを作成

1. PLCを作成し、新しいファンクションブロックをそこに追加します。

⇨ このプロキシブロックは、C++でプログラミングされた機能を提供します。これは、C++によって定義されTMC記述ファイルによりPLCで認識されるインターフェイスポインタを介して実行されます。



2. ファンクションブロックの宣言部分で、出力として(IIncrement)インターフェイスへのインターフェイスポインタを宣言します。このインターフェイスポインタは後に機能を外部に提供します。

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
```

```
1
```

3. クラスIDとインターフェイスIDをメンバ変数として作成します。
インターフェイスIDはグローバルリストを介して既に使用可能であるのに対し、まだ認識されていないクラスIDは他の手段によって決定されます。関連するC++ドライバのTMC記述ファイルを開くと、対応するGUIDが見つかります。

```
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. FB_initコンストラクタメソッドをPLCプロキシファンクションブロックに追加します。
この場合、オンラインでの変更ではなくむしろファンクションブロックの初期化であり、新しいTcCOMオブジェクト(指定されたクラスのクラスインスタンス)が作成され、指定されたインターフェイスへのインターフェイスポインタが取得されます。このプロセスで、使用される FW_ObjMgr_CreateAndInitInstance()ファンクション [▶ 15]は、TcCOMオブジェクトの名前と目的状態も与えられます。この2つのパラメータは、FB_initメソッドの入力パラメータとしてここで宣言され、これによってプロキシファンクションブロックで指定できます。インスタンス化されるTwinCAT C++クラスは、TcCOM初期化データとTcCOMパラメータを使用せずに管理を行います。
この関数呼び出しの場合、オブジェクト自体が参照カウンタをインクリメントします。

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. 使用された参照を再度使用するためにはリリースが必須となります。この目的で、ファンクションブロックのFB_exitデストラクタでFW_ObjMgr_DeleteInstance() [▶ 16]ファンクションを呼び出します。

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
```

⇨ これによって、プロキシファンクションブロックの実装が完了します。

6. インターフェイス経由で提供されるメソッドを呼び出すために、プロキシブロックのインスタンスをアプリケーション内で宣言します。呼び出し自体は、ファンクションブロックの出力として定義されたインターフェイスポインタを全面的に引き継ぎます。一般的なポインタと同様、事前のヌルチェックを行う必要があります。その後にメソッドを直接呼び出すができるようになり、またIntellisense経由でも呼び出すことができます。

MAIN*

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

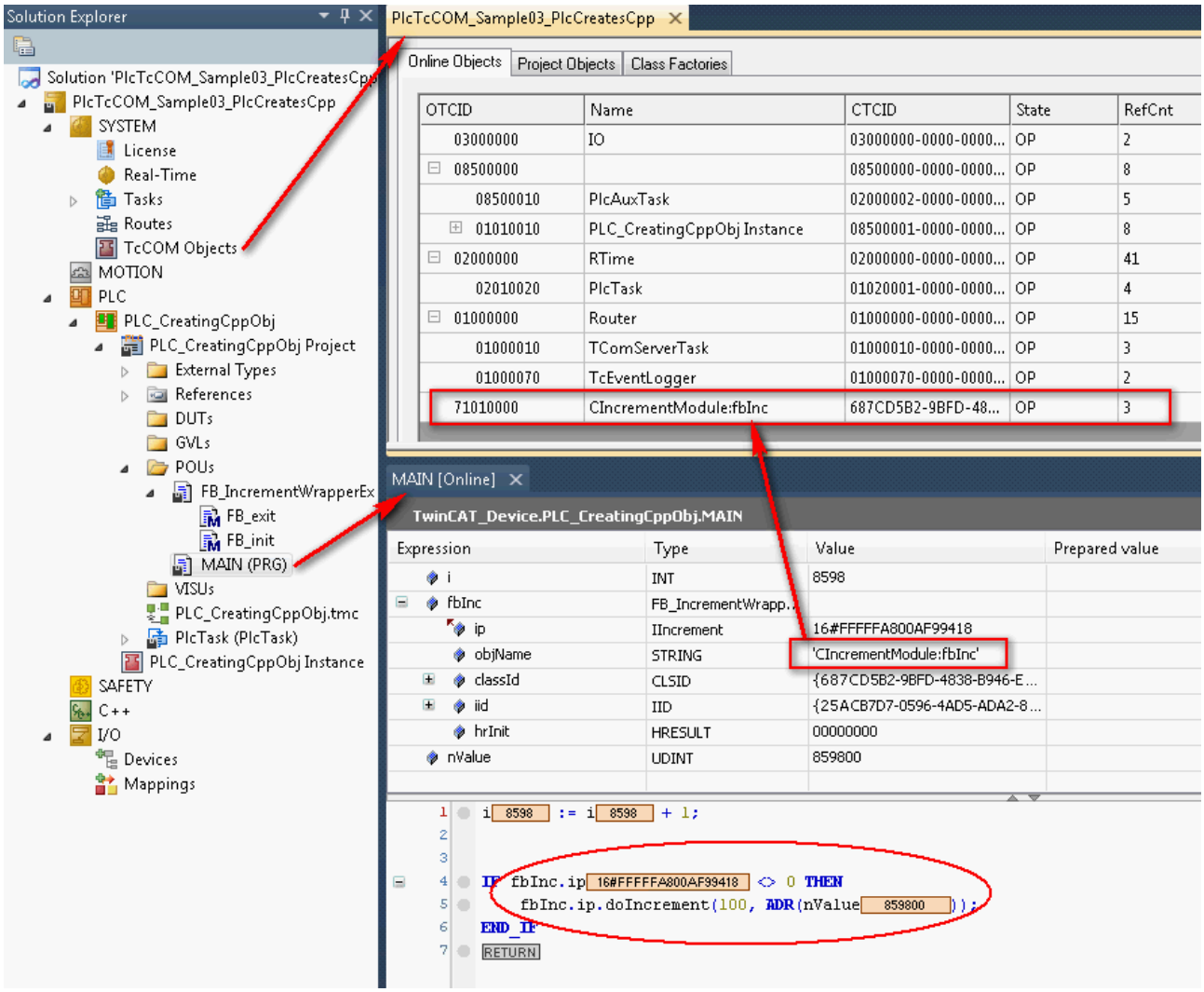
```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(100, ADR(nValue));
END_IF
```

⇨ サンプルはテストの準備ができています。

7.3.3 サンプルプロジェクトの実行

1. ターゲットシステムを選択し、プロジェクトをコンパイルします。
2. TwinCATの構成を有効にし、ログインを実行してPLCを開始します。

⇨ PLCアプリケーションのオンラインビューで、PLCプロキシファンクションブロックで必要なTcCOMオブジェクトの名前が確認できます。プロジェクトノードTcCOM Objectsは、生成されたオブジェクトを生成されたIDと必要な名前とともにリストに保持します。インターフェイスポインタは有効な値をもち、メソッドが実行されます。



7.4 TcCOM_Sample13_CppToPlc

説明

このサンプルは、メソッド呼び出しを使用することにより、C++モジュールからPLCのファンクションブロックへの通信を提供します。この目的で、PLCによってC++モジュールにより使用されるTcCOMインターフェイスが定義されます。

プロセスの提供側であるPLCページは、TcCOM Sample 01 [▶ 28]のプロジェクトを参照します。このサンプルでは、PLCはPLC通信の後に検討されます。ここでは、呼び出しがC++で提供され、同じインターフェイスを使用します。ここでは、呼び出しがC++で提供され、同じインターフェイスを使用します。

サブチャプター「サンプルの実装」にサンプルの説明があります。

サンプルのダウンロード: TcCOM Sample13 CppToPlc.zip

システム要件

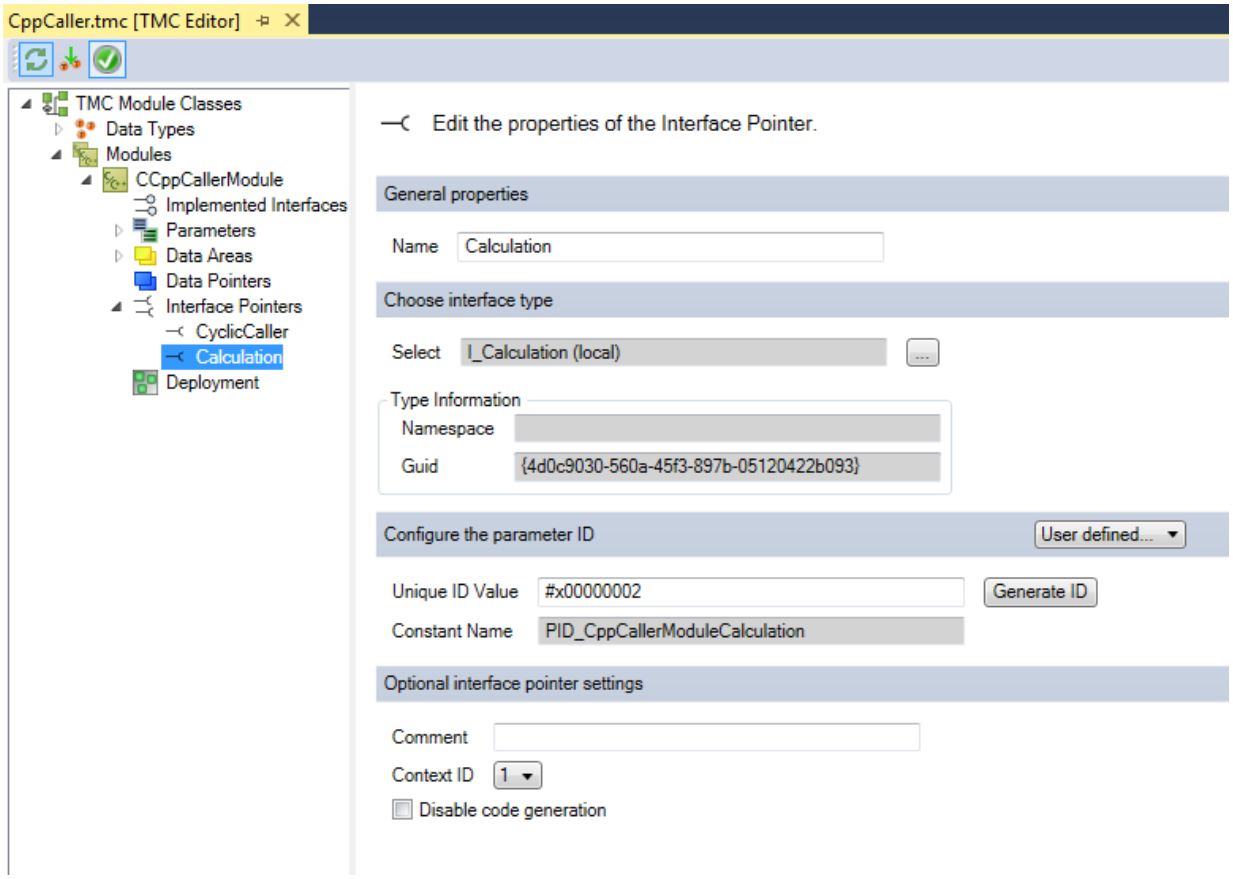
TwinCATバージョン	ハードウェア	リンクする必要があるPLCライブラリ
TwinCAT 3.1、ビルド4020	x86、x64	Tc3_Module

7.4.1 サンプルの実装

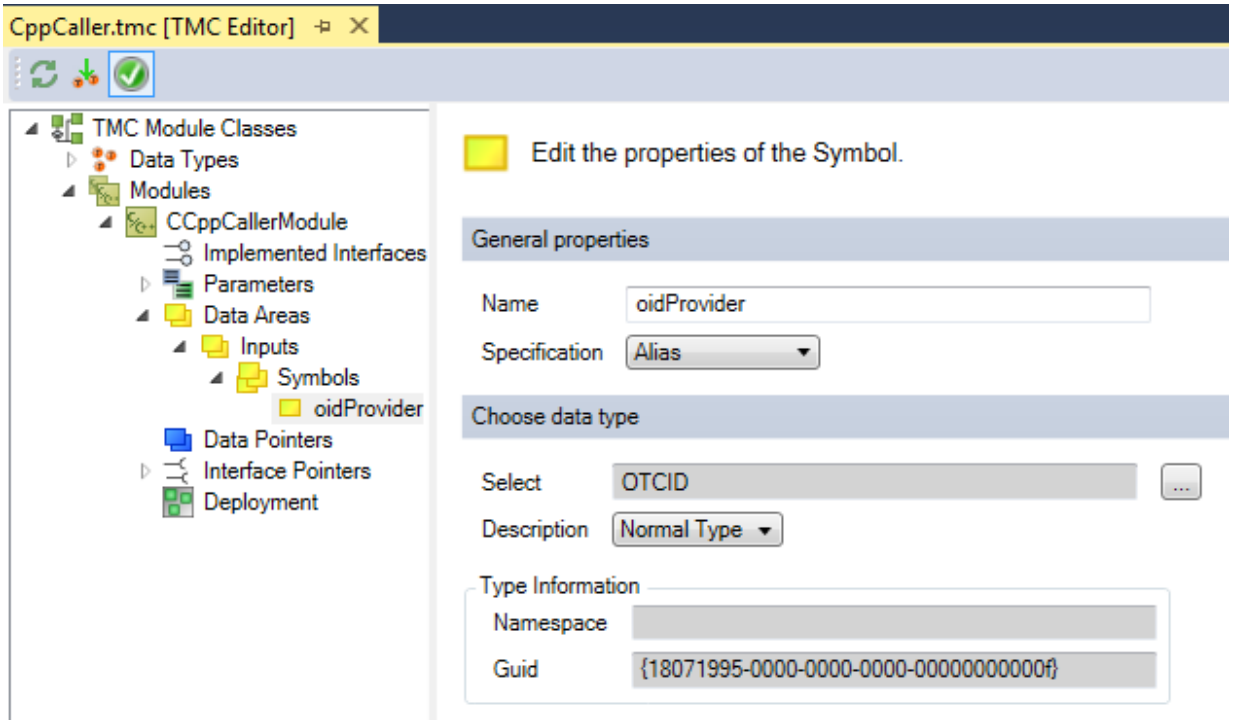
TcCOM Sample 01 [▶ 28]によって、PLCページを参照してください。そこでTcCOMモジュールとして登録されたファンクションブロックは、出力変数として割り当てられたオブジェクトIDを提供します。
このファンクションブロックの提供されたインターフェイスをアクセス可能にすることが、C++モジュールの役割です。

✓ サイクルIOモジュールを使用するC++プロジェクトが想定されています。

1. TMCエディタで、Calculationnの名前をもつタイプI_Calculationのインターフェイスポインタを作成します。後にこのポインタを介してアクセスを行います。

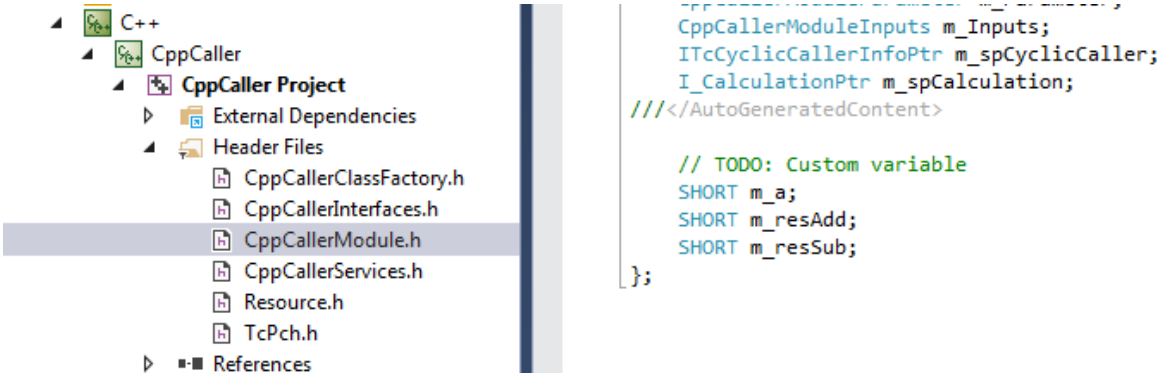


2. データ領域入力は、モジュールウィザードによってタイプInput-Destinationで既に作成されています。ここではTMCエディタで、oidProviderという名前でタイプOTCIDの入力を作成します。この入力を介して、オブジェクトIDは後にPLCからリンクされます。

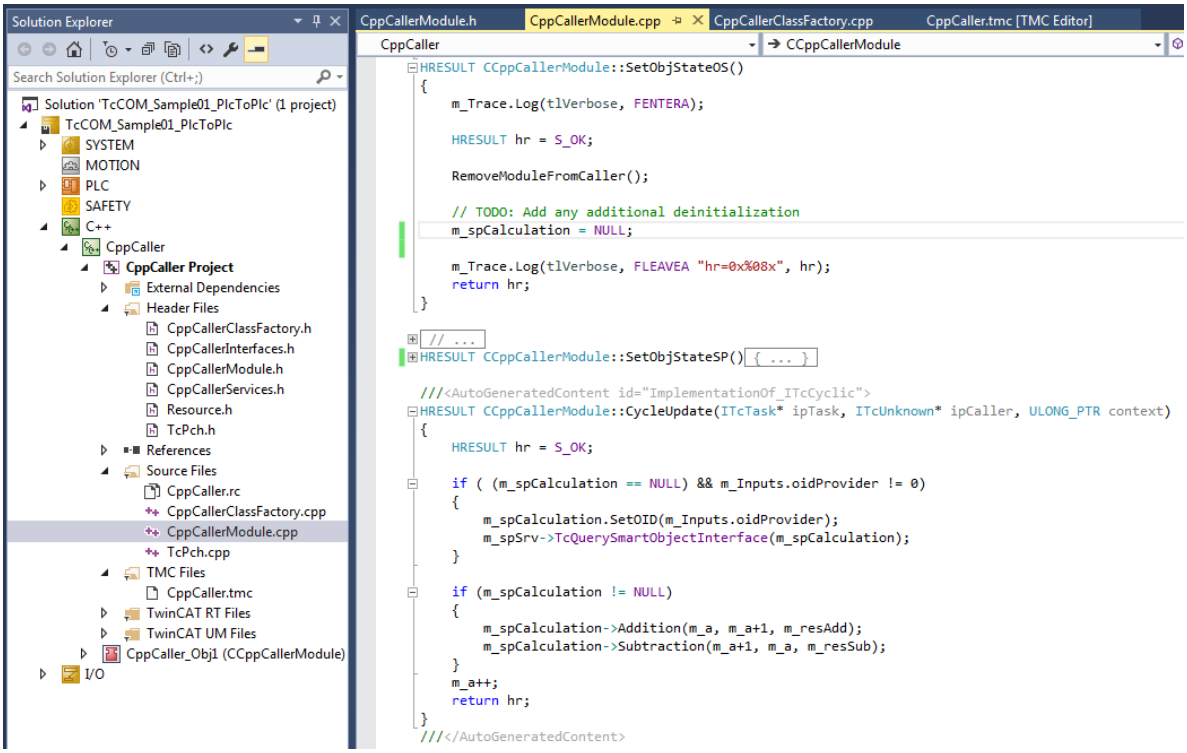


3. その他のすべてのシンボルはこのサンプルに関連がないので、削除して構いません。

⇨ TMCコードジェネレータは、コードを設定にしたがって生成します。
モジュールのヘッダに、一部の変数が後でメソッドの呼び出しを実行するために作成されます。

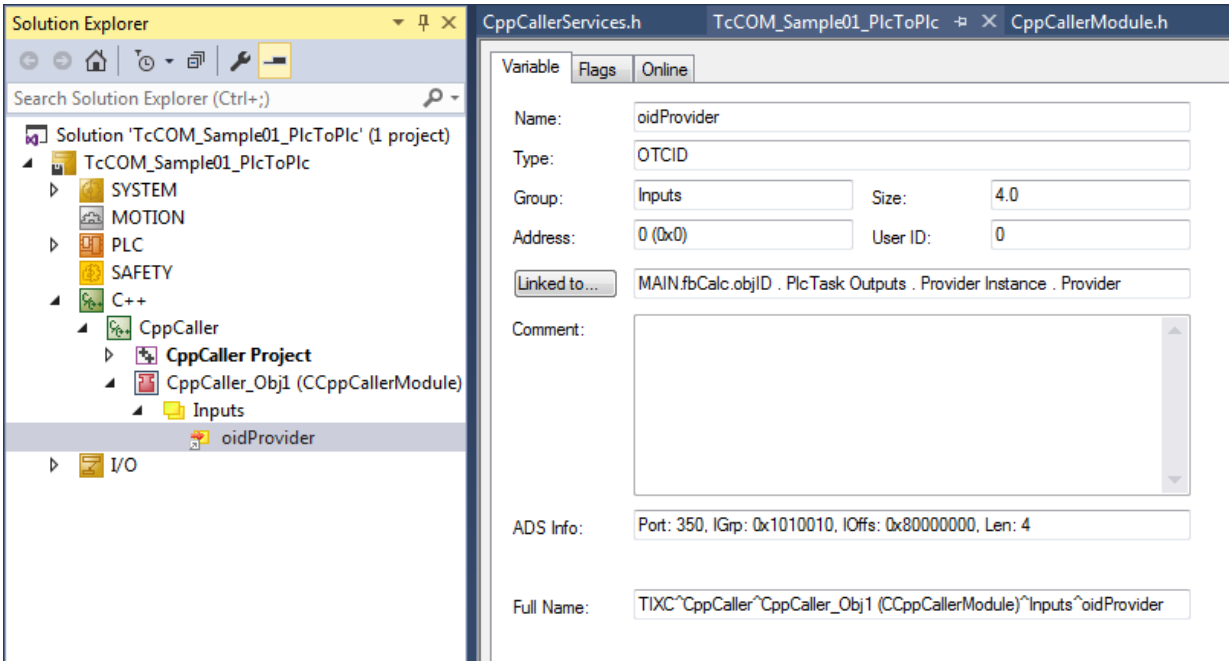


CycleUpdate()のモジュールの実際のコードでは、PLCから送信されたオブジェクトIDを使用してインターフェイスポインタが設定されます。PLCは最初にファンクションブロックを提供する必要があり、そしてCycleUpdate()で行われ、そのためリアルタイムで実行されることが重要となります。いったんこの状態になると、メソッドを呼び出すことができます。



加えて、上記で見たように、インターフェイスポインタはプログラムがシャットダウンするときにクリアされます。これは、SetObjStateOSメソッドで行われます。

4. ここで、C++プロジェクトをビルドします。
5. モジュールのインスタンスを作成します。
6. C++モジュールの入力をPLCの出力に接続します。



⇨ プロジェクトを開始することができます。PLCの実行中は、C++インスタンスへのMappingを通じて、OIDが認識されます。いったんこの状態になると、メソッドを呼び出すことができます。

8 付録

8.1 TcCOMテクノロジ

TwinCATモジュールコンセプトは、最新のマシンのモジュール化のためのコアエレメントの1つです。この章では、モジュール型コンセプトとモジュールの使用法について説明します。

8.1.1 TwinCATコンポーネントオブジェクトモデル(TcCOM)のコンセプト

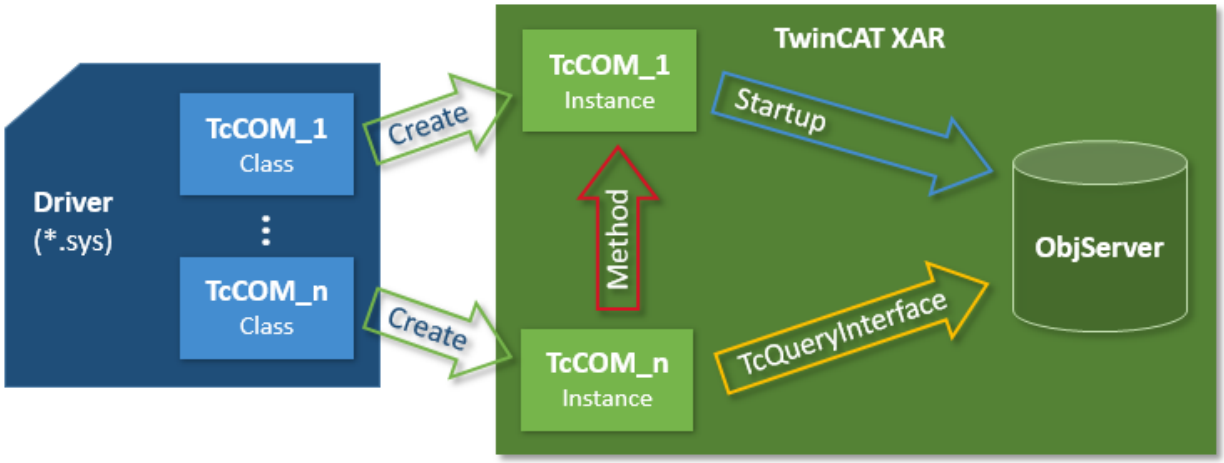
TwinCATコンポーネントオブジェクトモデル(COM)は、モジュールの特性と動作を定義します。Microsoft Windowsの「コンポーネントオブジェクトモデル(COM)」から導出されたモデルは、独立して開発されたコンパイル済みの各種ソフトウェアコンポーネントが相互に協調して動作する方法を記述しています。これを可能にするため、正確に定義されたモジュールの動作モードとインターフェイスの遵守を定義する必要があり、それによってこれらの相互作用が可能になります。そのようなインターフェイスは、例えば、異なるメーカ製のモジュール間の相互作用を容易にするためにも理想的です。

ある程度まで、TcCOMはCOM(Microsoft Windowsのコンポーネントオブジェクトモデル)に基づいていますが、COMのサブセットが使用されています。ただし、COMと比較すると、TcCOMにはCOMにはない追加定義が含まれています(例えば、ステートマシンモジュールなど)。

TcCOMモジュールの概要と用途

この入門的概要は、個々のトピックを分かりやすく説明するためのものです。

1つまたは複数のTcCOMモジュールがドライバに統合されています。このドライバは、MSVCコンパイラを使用してTwinCATエンジニアリングによって作成されています。モジュールとインターフェイスは、TMC (TwinCAT Module Class)ファイルに記述されています。ドライバとそのTMCファイルは、エンジニアリングシステム間で交換や結合が可能です。



これらのモジュールのインスタンスは、エンジニアリング機能を使用して作成されています。インスタンスはTMIファイルに関連付けられています。インスタンスはパラメータ化され、相互におよび他のモジュールとリンクしてIOを構築することができます。対応する構成がターゲットシステムに送信され、そこで実行されます。

対応するモジュールが開始され、それがTwinCAT ObjectServerに登録されます。TwinCAT XARも、プロセスイメージを提供します。モジュールは、特定のインターフェイスに関する他のオブジェクトへの参照をTwinCAT ObjectServerに確認することができます。そのような参照が使用可能な場合は、モジュールインスタンス上でインターフェイスメソッドを呼び出すことができます。

以下のセクションで、個々のトピックについて具体的に説明します。

IDの管理

種類の異なるIDが、モジュール同士およびモジュール内の相互作用に使用されています。TcCOMは、GUID(128ビット)と32ビットのlong intを使用しています。

TcCOMは、以下を使用します。

- GUIDに対して: ModulID、ClassID、およびInterfaceID
- 32ビットのlong intが使用されるID: ParameterID、ObjectID、ContextID、CategoryID

インターフェイス

COMやTcCOMでの重要となるコンポーネントはインターフェイスです。

インターフェイスは、特定のタスクを実行するために結合された一連のメソッドを定義します。インターフェイスは、固有のID (InterfaceID)をによって参照され、インターフェイスが変更されない限り、このIDは決して変更してはいけません。このIDにより、モジュールは他のモジュールと協調できるかどうかを決定することが可能になります。同時に、インターフェイスが明確に定義されている場合は、開発プロセスを独立して行うことができます。そのため、インターフェイスを変更するとIDが異なる状態が発生します。TcCOMコンセプトは、 InterfaceIDが他の(以前の)InterfaceID(TMC記述/TMCエディタでは「非表示」)の上に重ねることができるように設計されています。このようにすることで、インターフェイスの両方のバージョンが使用可能になり、他方では、どちらが最新のInterfaceIDであるかを常に明確にすることができます。同じコンセプトは、データ型にも存在します。

TcCOM自体は、一部のケース(ITComObjectなど)で規定されている一連のインターフェイス全体を既に定義していますが、ほとんどの場合はオプションです。多くのインターフェイスは、特定の適用分野でのみ意味をもちます。別のインターフェイスは非常に汎用的で、しばしば再利用が可能です。ユーザ定義インターフェイスに関する規定があるため、例えば、2つのサードパーティ製モジュールも相互に連携することができます。

- すべてのインターフェイスは、基本インターフェイスItcUnknownから導出されています。ItcUnknownは、COMの対応するインターフェイスと同じように、モジュールの他のインターフェイスを確認するための基本的サービス(TcQueryInterface)や、モジュールの寿命を管理するためのサービス(TcAddRefおよびTcRelease)を提供します。
- ITComObjectインターフェイスは、各モジュールで実装される必要があり、モジュールの名前、ObjectID、親のObjectID、パラメータ、およびステートマシンにアクセスするためのメソッドを含んでいます。

次のようないくつかの汎用インターフェイスが多くのモジュールで使用されています。

- ITcCyclicはモジュールに実装され、周期的に呼び出されます("CycleUpdate")。このモジュールは、周期的呼び出しを取得するためにTwinCATタスクのITcCyclicCallerインターフェイス経由で登録することができます。
- ITcADIインターフェイスは、モジュールのデータ領域にアクセスするために使用することができます。
- ITcWatchSourceはデフォルトで実装され、ADSデバイスの通知や他の機能を容易にします。
- ITcTaskインターフェイスは、リアルタイムシステムのタスクに実装され、サイクルタイムに関する情報や、優先順位およびその他のタスク情報を提供します。
- ITComObjectServerインターフェイスは、ObjectServerに実装され、すべてのモジュールによって参照されます。

一連の汎用インターフェイス全体は、既に定義済みです。汎用インターフェイスには、モジュールの交換と再利用をサポートするという利点があります。ユーザ定義インターフェイスは、適切な汎用インターフェイスが使用可能でない場合にのみ定義してください。

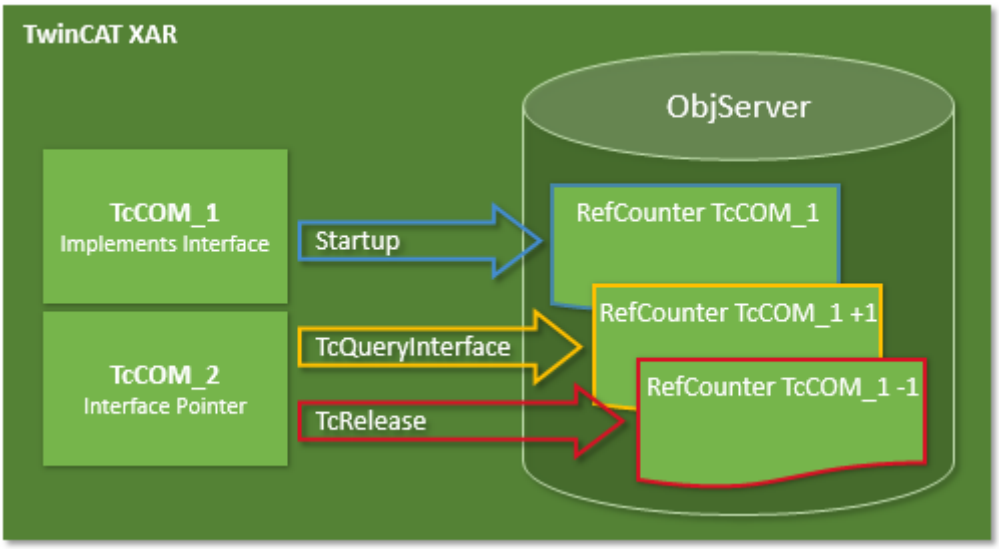
Class Factories

"Class Factories"は、C++でモジュールを作成するために使用されます。すべてのモジュールには、共通のClass Factoryをもつドライバが含まれます。Class Factoryが、ObjectServerに登録されると、特定のモジュールクラスの開発のためのサービスが可能になります。モジュールクラスは、モジュールの固有の

ClassIDによって識別されます。ObjectServerが(構成の初期化データに基づき、またはランタイム時に他のモジュールを通じて)新しいモジュールをリクエストすると、モジュールはClassIDに基づき適切なClass Factoryを選択し、ITcClassFactoryインターフェイス経由でモジュールの作成をトリガします。

モジュールのサービス寿命

COMと同様に、モジュールのサービス寿命は参照カウンタ(RefCounter)によって決定されます。参照カウンタは、モジュールインターフェイスが確認されるたびにインクリメントされます。インターフェイスがリリースされると、このカウンタはデクリメントされます。インターフェイスは、モジュールがObjectServer(ITComObjectインターフェイス)にログインしたときにも確認されるため、参照カウンタは少なくとも1になります。このカウンタはログアウト時にデクリメントされます。カウンタが0に達すると(通常はObjectServerからのログアウト後に)、モジュールは自動的に自身を削除します。もう1つのモジュールが既に参照を保持している(インターフェイスポインタをもっている)場合は、モジュールは継続して存続し、インターフェイスポインタはこのポインタがリリースされるまで有効のままになります。



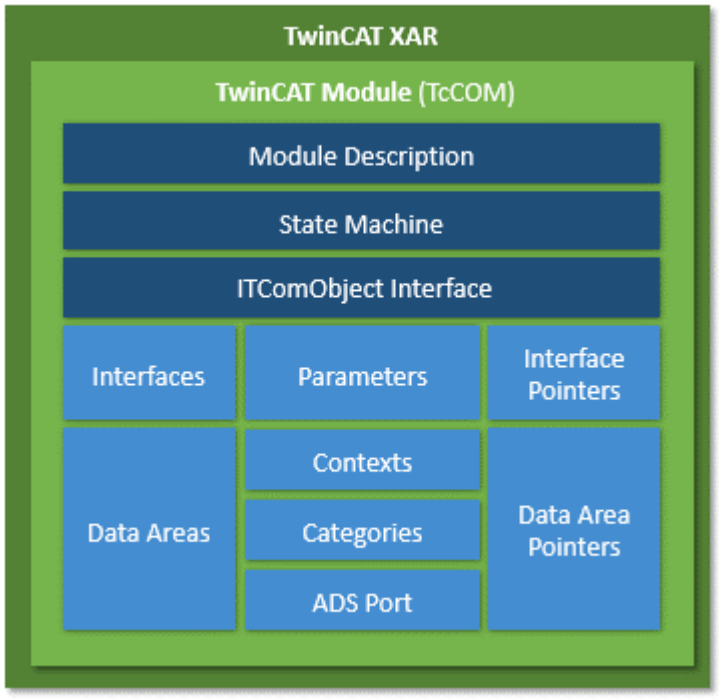
8.1.1.1 TwinCATモジュールのプロパティ

TcCOMモジュールには数多くの正式に定義され規定されたオプションのプロパティがあります。これらのプロパティは十分に正規化され、互換性のあるアプリケーションを実現しています。各モジュールにはモジュール記述が含まれ、モジュールプロパティがここに記述されています。モジュールプロパティは、モジュールとそれらの相互関係を構成するために使用されます。

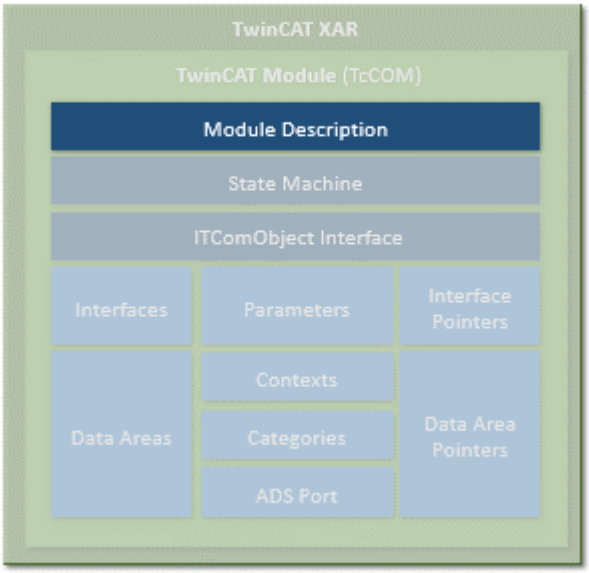
モジュールがTwinCATランタイムでインスタンス化されている場合、モジュールは自身を中央システムインスタンスであるObjectServerに登録します。これによってモジュールは他のモジュールや汎用ツールからも到達およびパラメータ設定が可能になります。モジュールは個別にコンパイルでき、そのため個別に開発、テストおよびアップデートすることができます。モジュールは、ローパスフィルタなどの基本機能のみを含むなど、非常に単純に構成することができます。あるいは、内部的に非常に複雑で、マシンサブアセンブリの制御システム全体を含む場合もあります。

モジュールには非常に多くのアプリケーションがあり、オートメーションシステムのすべてのタスクをモジュールで指定することができます。そのため、各種のモジュールを明確に区別することはできません。モジュールは主として、リアルタイムタスクやフィールドバスドライバまたはPLCランタイムシステム、およびマシンユニット制御用のユーザ固有またはアプリケーション固有のアルゴリズムなどのオートメーションシステムの基本機能を表します。

下記の図は、一般的なTwinCATモジュールとそのメインプロパティを示しています。濃い青のブロックは所定のプロパティを定義し、薄い青のブロックはオプションのプロパティを定義しています。



モジュールの説明



各TcCOMモジュールには、いくつかの一般的記述パラメータがあります。これには、一義的にモジュールクラスを参照するClassIDが含まれます。このモジュールは対応するClassFactoryによってインスタンス化されます。各モジュールインスタンスは、TwinCATランタイムで固有のObjectIDをもちます。加えて、論理的な親を参照する親ObjectIDがあります。

以下に説明するモジュールの記述、ステートマシンおよびパラメータは、ITComObjectインターフェイス(「インターフェイス」を参照)を介して到達することができます。

クラス記述ファイル(*.tmc)

モジュールクラスは、クラス記述ファイル(TwinCATモジュールクラス; *.tmc)に記述されています。

これらのファイルは、他の人がモジュールを使用し組み込むことができるように、モジュールのプロパティおよびインターフェイスを記述するために開発者によって使用されます。一般的情報(ベンダーデータ、モジュールクラスIDなど)に加えて、オプションのモジュールプロパティも記述されます。

- サポートされるカテゴリ
- 実装済みのインターフェイス
- 対応するシンボルをもつデータ領域
- パラメータ
- インターフェイスポインタ
- データポインタ(設定可能)

システムコンフィギュレータは、構成へのモジュールインスタンスの追加やパラメータの指定、他のモジュールとのリンクの設定の基礎として主にクラス記述ファイルを使用します。

また、このファイルにはモジュールのすべてのデータ型の記述も含まれ、それらの記述はコンフィギュレータによって全般的なデータ型に適用されます。このようにすることで、システムに存在するTMC記述のすべてのインターフェイスをすべてのモジュールが使用することができます。

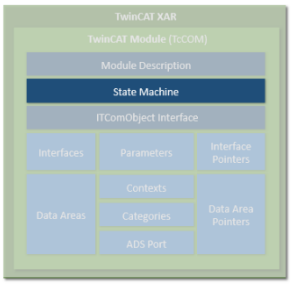
複数のモジュールが含まれるより複雑な構成もクラス記述ファイルに記述することができ、これらの構成は事前設定され特定のアプリケーションにリンクされます。そのため、内部的に多くのサブモジュールで構成される複雑な装置のモジュールは、開発段階で1つのまとまりとして定義および事前設定することができます。

インスタンス記述ファイル(*.tmi)

特定のモジュールのインスタンスは、インスタンス記述ファイル(TwinCATモジュールインスタンス; *.tmi)で記述されます。インスタンス記述ファイルはクラス記述ファイルに類似した形式に基づきますが、クラス記述ファイルとは対照的に、プロジェクト内の特別なモジュールインスタンスに関するパラメータやインターフェイスポインタなどの具体的な仕様を既に含んでいます。

インスタンス記述ファイルは、特定のプロジェクト向けにクラス記述のインスタンスが作成されたときにTwinCATエンジニアリング(XAE)によって作成されます。この記述ファイルは、主として構成に関係するすべてのツール間でデータをやり取りするために使用されます。ただし、インスタンス記述ファイルは、例えば、特別にパラメータ設定されたモジュールを新しいプロジェクトで再利用する場合など、プロジェクト間でも使用できます。

ステートマシン

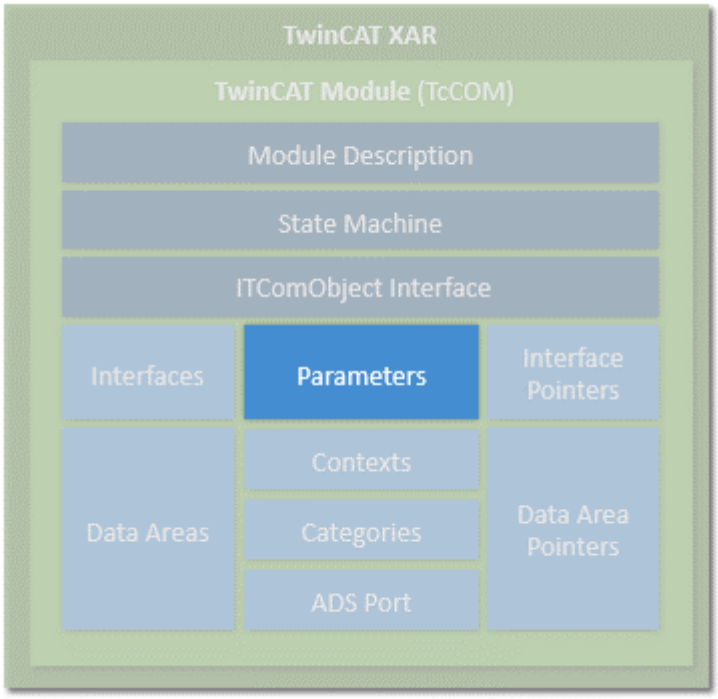


各モジュールは1つのステートマシンを含み、モジュールの初期化状態とこの状態を外部から変更する手段を記述します。ステートマシンは、モジュールの開始と停止中に発生する状態を記述します。これは、他のモジュールと連動したモジュールの作成、パラメータ設定および作成に関係します。

アプリケーション固有の状態(フィールドバスまたはドライバなどの)は、自身のステートマシンで記述することができます。TcCOMモジュールのステートマシンは、INIT、PREOP、SAFEOPおよびOPの状態を定義します。状態の名称はEtherCATフィールドバスのものと同じですが、実際の状態は異なります。TcCOMモジュールがEtherCAT用のフィールドバスドライバを実装する場合、2つのステートマシン(モジュールおよびフィールドバスステートマシン)を含み、これらはシーケンシャルに渡されます。モジュールステートマシンは、フィールドバスステートマシンが開始する前に、運転状態(OP)に達している必要があります。

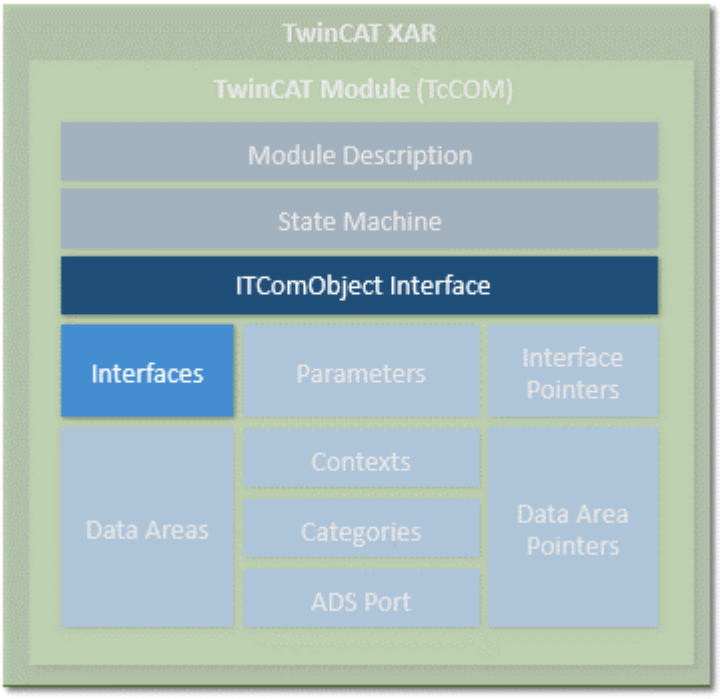
ステートマシンの詳細は、個別に記述 [▶ 60]されています。

パラメータ



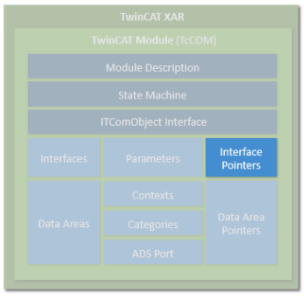
モジュールはパラメータをもつことができ、パラメータは初期化中またはその後のランタイム時(OP状態)にリードまたはライトできます。各パラメータは、パラメータIDによって指定されます。パラメータIDの一意性は、グローバル、制限付きグローバル、またはモジュール固有にすることができます。詳細については、「IDの管理」セクションを参照してください。パラメータIDに加えて、パラメータには現在のデータが含まれています。データ型はパラメータによって異なり、個々のパラメータIDごとに一義的に定義されています。

インターフェイス



インターフェイスは、定義済みのメソッド(ファンクション)のセットで構成され、他のモジュールとの接点となるモジュールを提供します。インターフェイスは、前述したように固有のIDで特徴づけられます。モジュールは少なくとも1つのITComObjectインターフェイスをサポートする必要があり、これに加えて必要な数のインターフェイスを含むことができます。インターフェイス参照は、対応するインターフェイスIDを指定したメソッド"TcQueryInterface"を呼び出すことにより確認することができます。

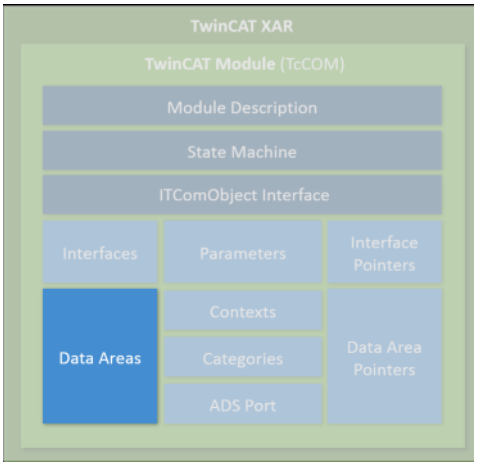
インターフェイスポインタ



インターフェイスポインタは、それが指し示すインターフェイスと同じように動作します。モジュールが他のモジュールのインターフェイスを使用したい場合、対応するインターフェイスタイプのインターフェイスポインタをもつ必要があり、このポインタが他のモジュールを指し示す必要があります。それによって他のモジュールのメソッドを使用することができます。

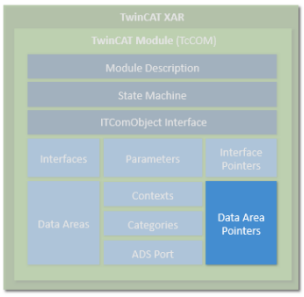
インターフェイスポインタは通常、ステートマシンの起動時に設定されます。INITからPREOP(IP)への遷移中には、モジュールは対応するインターフェイスをもつ他のモジュールのオブジェクトIDを受け取り、PREOPからSAFEOP (PS)またはSAFEOPからOP (SO)への遷移中には、他のモジュールのインスタンスがObjectServerで検索され、対応するインターフェイスがMethod Queryインターフェイスによって設定されます。SAFEOPからPREOP (SP)またはOPからSAFEOP (OS)などの反対方向の状態遷移中には、インターフェイスを再び有効にする必要があります。

データ領域



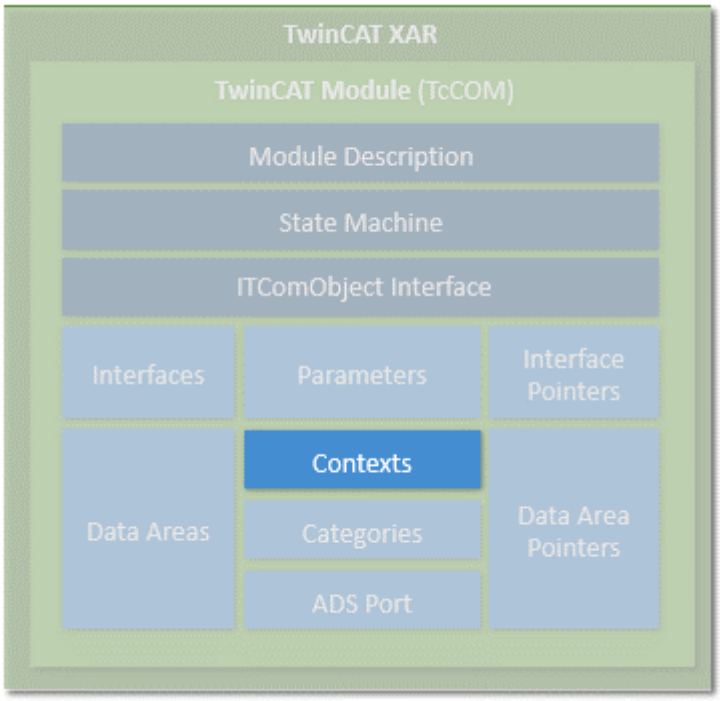
モジュールにはデータ領域を含むことができ、環境によって(他のモジュールまたはTwinCATのI/Oエリアなどによって)使用されます。これらのデータ領域は、任意のデータを含むことができます。これらはしばしばプロセスイメージデータ(入出力)に使用されます。データ領域の構造は、モジュールのデバイスディスクリプションに定義されています。モジュールに他のモジュールからアクセス可能にしたいデータ領域が含まれる場合、データへのアクセスを有効にするITcADIインターフェイスを実装します。データ領域にはシンボル情報が含まれ、個々のデータ領域の構造が詳細に記述されています。

データ領域ポインタ



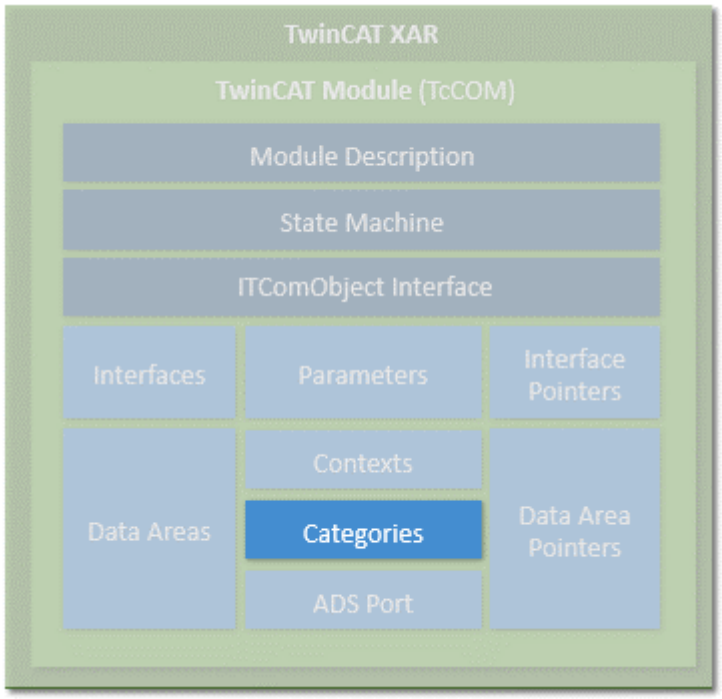
モジュールが他のモジュールのデータ領域にアクセスしたい場合、データ領域ポインタを含むことができます。これらのポインタは通常、ステートマシンの初期化中に他のモジュールのデータ領域またはデータ領域セクションに対して設定されます。アクセスはメモリ領域に直接行われるため、必要に応じてアクセス操作を防止できる対応する保護メカニズムを実装する必要があります。多くの場合、対応するインターフェイスを使用することを推奨します。

コンテキスト



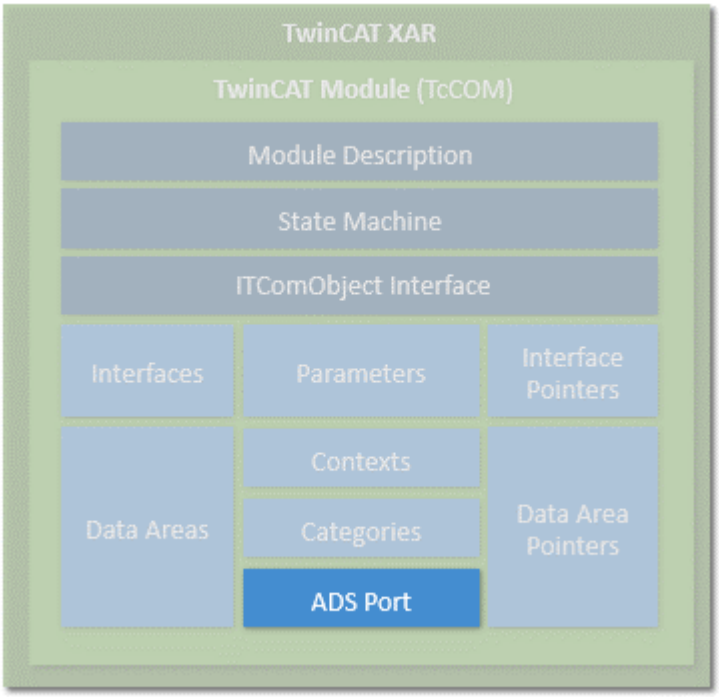
コンテキストはリアルタイムタスクの一環と見なされます。コンテキストは、例えば、モジュールの設定に必要となります。単純なモジュールは通常、シングルタイムコンテキストで動作するため、詳細な指定は必要ありません。一部の他のモジュールは、複数のコンテキストでアクティブになる場合があります(例えば、EtherCATマスタは複数の独立したリアルタイムタスクをサポートし、または制御ループは別のサイクルタイムの下位層の制御ループを処理することができます)。モジュールに複数の時間依存コンテキストが含まれている場合は、モジュール記述にこれを指定する必要があります。

カテゴリ



モジュールは、インターフェイスITComObjectCategoryを実装することによってカテゴリを提供することができます。カテゴリはObjectServerによって列挙され、カテゴリに自身を関連付けるためにこのサーバを使用するオブジェクトは、ObjectServer(ITComObjectEnumPtr)によって確認することができます。

ADS



ObjectServerに入力された各モジュールは、ADS経由で到達することができます。ObjectServerは、例えば、パラメータのリード/ライトや、ステートマシンへのアクセスのために、モジュールのITComObjectインターフェイスを使用します。加えて、専用のADSポートを実装することができ、このポートを通じて専用のADSコマンドを受け取ることができます。

システムモジュール

加えて、TwinCATランタイムは多くのモジュールを提供し、これにより基本的なランタイムサービスが他のモジュールから利用可能になります。これらのシステムモジュールは、固定された一定のObjectIDをもち、このIDを通じて他のモジュールからアクセスできます。そのようなシステムモジュールの一例がリアルタイムシステムであり、リアルタイムタスクの生成などの基本的なリアルタイムシステムサービスをITcRTimeインターフェイス経由で使用可能にします。また、ADSルータもシステムモジュールとして実装され、他のモジュールがそのADSポートをここに登録することができます。

モジュールの作成

モジュールは、C++とIEC 61131-3の両方で作成できます。オブジェクト指向のTwinCAT PLCの拡張がこの目的で使用されます。両方の環境で作成されたモジュールは、純粋なC++モジュールと同じ方法でインターフェイス経由で連携することができます。オブジェクト指向の拡張は、C++と同じインターフェイスを使用可能にします。

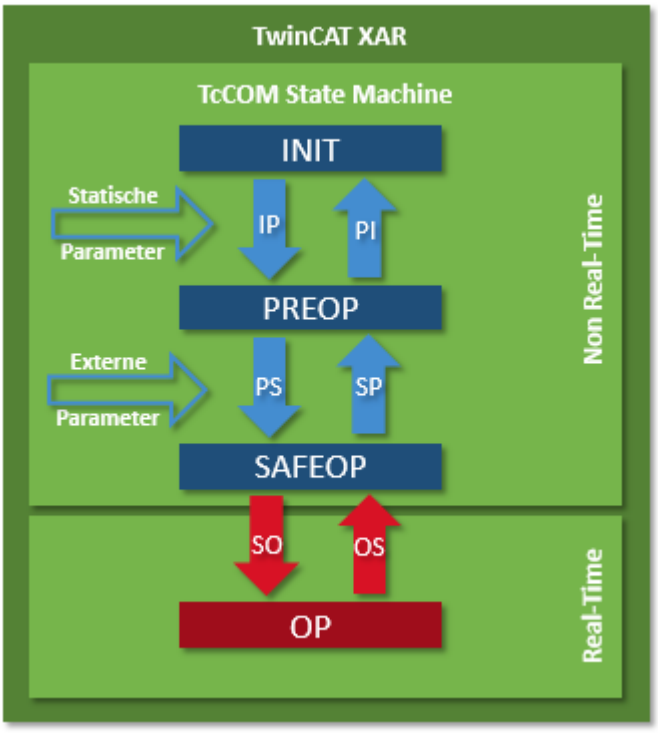
また、PLCモジュールもObjectServer経由で登録でき、したがってこれを経由して到達することができます。PLCモジュールは、複雑さに応じてさまざまなものがありますが、小規模なフィルタモジュールのみ生成する場合も、PLCプラグラム全体を1つのモジュールにパックする場合も違いはありません。オートメーションにおいて、各PLCプラグラムはTwinCATモジュールの意味の範囲内で1つのモジュールになります。従来の各PLCプラグラムは自動的にモジュールにパックされ、自身をObjectServerや1つまたは複数のタスクモジュールに登録します。PLCモジュールのプロセスデータへのアクセス(フィールドバスドライバとのMappingなど)も、定義済みのデータ領域とITcADIを介して制御されます。

TwinCATモジュールとしてPLCプラグラムの各部を明示的に定義する限り、この動作の透過性が保たれ、PLCプログラマから隠蔽されるため、適切な柔軟性をもって使用することができます。

8.1.1.2 TwinCATモジュールステートマシン

状態(INIT、PREOP、SAFEOP、OP)に加えて、対応する状態遷移があり、この範囲内で汎用またはモジュール固有の動作を実行する必要があり、また実行することができます。ステートマシンの設計は非常にシンプルです。いかなる場合も、次または前のステップしか存在しません。

これには次の状態遷移が含まれます: INITからPREOPへ(IP)、PREOPからSAFEOPへ(PS)、およびSAFEOPからOPへ(SO)。反対方向では次の状態遷移が含まれます: OPからSAFEOPへ(OS)、SAFEOPからPREOPへ(SP)、およびPREOPからINITへ(PI)。SAFEOP状態までの(SAFEOP状態を含む)すべての状態および状態遷移は、非リアルタイムコンテキストで行われます。SAFEOPからOPへの遷移、OP状態、およびOPからSAFEOPへの遷移は、リアルタイムコンテキストで行われます。この使い分けは、リソースが割り当てまたは有効になっている場合、またはモジュールが他のモジュールを登録または登録解除する場合に該当します。



状態: INIT

INIT状態は、仮想的な状態です。モジュールを作成すると直ちに、IP状態遷移が実行され、モジュールはINITからPREOPに変わります。インストールとIP状態遷移は常に一緒に起こるため、モジュールはINIT状態にとどまることは決してありません。モジュールを削除する場合のみ、短い時間のみINIT状態のままになります。

遷移: INITからPREOP (IP)へ

IP状態遷移の間に、モジュールは固有のObjectIDでObjectServerに登録します。初期化パラメータ(オブジェクトの作成時にこのパラメータも割り付けられます)がモジュールに送信されます。この遷移中は、他のモジュールが既に存在し、ObjectServerに登録されているかどうかが不明であるため、モジュールは他のモジュールとの接続を確立することができません。モジュールがリソース(メモリなど)を必要とする場合には、リソースは状態遷移中に割り当てられます。すべての割り当て済みリソースは、PREOPからINITへの(PI)遷移中に再びリリースする必要があります。

状態: PREOP

PREOP状態では、モジュールの作成は完了し、モジュールは通常、完全にパラメータ設定されています。ただし、その他のパラメータがPREOPからSAFEOPへの遷移中に追加されることはあります。モジュールはObjectServerに登録されますが、他のモジュールとの接続はまだ作成されていません。

遷移: PREOPからSAFEOPへ(PS)

この状態遷移では、モジュールは他のモジュールとの接続を確立することができます。この目的で通常、モジュールは特に初期化データをもつ他のモジュールのObjectIDを受け取り、このデータがObjectServer経由のこれらのモジュールとの実際の接続に変換されます。

この遷移は、一般的にコンフィギュレータにしたがってシステムによって、または別のモジュール(親モジュールなど)によってトリガすることができます。この状態遷移中に、その他のパラメータを送信することができます。例えば、親モジュールは自身のパラメータを子モジュールに送信することができます。

状態: SAFEOP

モジュールはまだ非リアルタイムコンテキストにあり、システムまたは他のモジュールによってOP状態に切り替えられるのを待機しています。

遷移: SAFEOPからOPへ(SO)

SAFEOPからOPへの遷移、OP状態、およびOPからSAFEOPへの遷移は、リアルタイムコンテキストで行われます。この状態では、システムリソースは割り当てられません。一方、リソースは他のモジュールからリクエストできるようになり、タスク中の周期的呼び出しを取得するためなどに、モジュールを他のモジュールに登録できます。

この遷移は、長時間実行するタスクには使用しないでください。例えば、ファイル操作はPS遷移中に実行してください。

状態: OP

OP状態では、モジュールは動作を開始し、TwinCATシステムの意味合いで完全にアクティブになります。

遷移: OPからSAFEOPへ(OS)

この状態遷移は、リアルタイムコンテキストで行われます。SO遷移のすべての動作が逆になり、SO遷移中にリクエストされたすべてのリソースは再びリリースされます。

遷移: SAFEOPからPREOPへ(SP)

PS遷移のすべての動作が逆になり、PS遷移中にリクエストされたすべてのリソースは再びリリースされます。

遷移: PREOPからINITへ(PI)

IP遷移のすべての動作が逆になり、IP遷移中にリクエストされたすべてのリソースは再びリリースされます。モジュールはObjectServerからサインオフし、通常、自身を削除します(「サービス寿命」を参照)。

8.2 インターフェイス

8.2.1 インターフェイスITComObject

ITComObjectインターフェイスは、すべてのTwinCATモジュールに実装されています。これにより、基本機能が使用可能になります。

シンタックス

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

方法

名前	説明
TcGetObjectId(OTCID& objId) [▶ 63]	与えられたOTCID参照を使用して、オブジェクトIDを保存します。
TcSetObjectId [▶ 63]	与えられたOTCIDにオブジェクトのオブジェクトIDを設定します。
TcGetObjectName [▶ 63]	与えられた長さのバッファにオブジェクト名を保存します。
TcSetObjectName [▶ 64]	与えられたCHAR*にオブジェクトのオブジェクト名を設定します。
TcSetObjState [▶ 64]	事前定義された状態への遷移を初期化します。
TcGetObjState [▶ 65]	オブジェクトの現在の状態の確認を行います。
TcGetObjPara [▶ 65]	PTCIDで識別されるオブジェクトパラメータを確認します。
TcSetObjPara [▶ 65]	PTCIDで識別されるオブジェクトパラメータを設定します。
TcGetParentObjId [▶ 66]	与えられたOTCID参照を使用して親オブジェクトIDを保存します。
TcSetParentObject [▶ 66]	親オブジェクトIDを与えられたOTCIDに設定します。

コメント

ITComObjectインターフェイスは、すべてのTwinCATモジュールに実装されています。これによりステートマシンに関する機能が使用可能になり、TwinCATシステムとの情報のやり取りが可能になります。

8.2.1.1 メソッドITcComObject:TcGetObjectId(OTCID& objId)

このメソッドは、与えられたOTCID参照を使用してオブジェクトIDを保存します。

シンタックス

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

パラメータ

objId: (タイプ: OTCID&) OTCID値への参照

戻り値

OTCID取得の成功を示します

説明

このメソッドは、与えられたOTCID参照を使用して、オブジェクトIDを保存します。

8.2.1.2 メソッドITcComObject:TcSetObjectId

メソッドTcSetObjectIdは、与えられたOTCIDにオブジェクトのオブジェクトIDを設定します。

シンタックス

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

パラメータ

objId: (タイプ: OTCID) OTCID、設定する必要があります。

戻り値

常にS_OKを返すことを推奨します。現在、戻り値はTwinCATタスクによって無視されます。

説明

ID変更の成功を示します

8.2.1.3 メソッドITcComObject:TcGetObjectName

メソッドTcGetObjectNameは、オブジェクト名を与えられた長さのバッファに保存します。

シンタックス

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

パラメータ

objName: （タイプ: CHAR*） 名前、設定する必要があります。

nameLen: （タイプ: ULONG） ライトの最大長。

戻り値

名前取得の成功を示します

説明

メソッドTcGetObjectNameは、オブジェクト名を与えられた長さのバッファに保存します。

8.2.1.4 メソッドITcComObject:TcSetObjectName

メソッドTcSetObjectNameは、与えられたCHAR*にオブジェクトのオブジェクト名を設定します。

シンタックス

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

パラメータ

objName: （タイプ: CHAR*） 設定するオブジェクトの名前

戻り値

名前変更の成功を示します

説明

メソッドTcSetObjectNameは、与えられたCHAR*にオブジェクトのオブジェクト名を設定します。

8.2.1.5 メソッドITcComObject:TcSetObjState

メソッドTcSetObjStateは、特定の状態への遷移を初期化します。

シンタックス

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

パラメータ

state: （タイプ: TCOM_STATE） 新しい状態を示します

ipSrv: （タイプ: ITComObjectServer*） オブジェクトを処理します

pInitData: （タイプ: PTComInitDataHdr） パラメータのリストのポインタ(オプション)
リストを反復する方法の例は、マクロIMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATAを参照してください。

戻り値

状態の変更の成功を示します

説明

メソッドTcSetObjStateは、特定の状態への遷移を初期化します。

8.2.1.6 メソッドITcComObject:TcGetObjState

メソッドTcGetObjStateは、オブジェクトの現在の状態を取得します。

シンタックス

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

パラメータ

pState:（タイプ: TCOM_STATE*）状態へのポインタ

戻り値

状態取得の成功を示します

説明

メソッドTcGetObjStateは、オブジェクトの現在の状態を取得します。

8.2.1.7 メソッドITcComObject:TcGetObjPara

メソッドTcGetObjParaは、PTCIDによって識別されるオブジェクトパラメータを取得します。

シンタックス

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

パラメータ

pid:（タイプ: PTCID）オブジェクトパラメータのパラメータID

nData:（タイプ: ULONG&）データの最大長

pData:（タイプ: PVOID&）データへのポインタ

pgp:（タイプ: PTCGP）将来の拡張用に予約済み、NULLを渡す

戻り値

オブジェクトパラメータ取得の成功を示します

説明

メソッドTcGetObjParaは、PTCIDによって識別されるオブジェクトパラメータを取得します。

8.2.1.8 メソッドITcComObject:TcSetObjPara

メソッドTcSetObjParaは、PTCIDによって識別されるオブジェクトパラメータを設定します。

シンタックス

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

パラメータ

pid: (タイプ: PTCID) オブジェクトパラメータのパラメータID

nData: (タイプ: ULONG) データの最大長

pData: (タイプ: PVOID) データへのポインタ

pgp: (タイプ: PTCGP) 将来の拡張用に予約済み、NULLを渡す

戻り値

オブジェクトパラメータ取得の成功を示します

説明

メソッドTcSetObjParaは、PTCIDによって識別されるオブジェクトパラメータを設定します。

8.2.1.9 メソッドITcComObject:TcGetParentObjId

メソッドTcGetParentObjIdは、与えられたOTCID参照を使用して、親オブジェクトIDを保存します。

シンタックス

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

パラメータ

objId: (タイプ: OTCID&) OTCID値への参照

戻り値

parentObjId取得の成功を示します

説明

メソッドTcGetParentObjIdは、与えられたOTCID参照を使用して、親オブジェクトIDを保存します。

8.2.1.10 メソッドITcComObject:TcSetParentObjId

メソッドTcSetParentObjIdは、与えられたOTCID参照を使用して、親オブジェクトIDを設定します。

シンタックス

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

パラメータ

objId: (タイプ: OTCID) OTCID値への参照

戻り値

常にS_OKを返すことを推奨します。現在、戻り値はTwinCATタスクによって無視されます。

説明

メソッドTcSetParentObjIdは、与えられたOTCID参照を使用して、親オブジェクトIDを設定します。

8.2.2 インターフェイスITcUnknown

ITcUnknownは、参照のカウントやさらに特定されたインターフェイスへの参照の確認を定義します。

シンタックス

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

宣言ファイル: TcInterfaces.h

インクルードが必要: -

方法

名前	説明
TcAddRef [▶ 67]	参照カウンタをインクリメントします。
TcQueryInterface [▶ 67]	実装されたインターフェイスへの参照のIID経由の確認
TcRelease [▶ 69]	参照カウンタをデクリメントします。

注意

すべてのTcCOMインターフェイスは、ITcUnknownから直接的または間接的に導出されます。その結果、すべてのTcCOMモジュールクラスはITComObjectから導出されているため、ITcUnknownを実装しています。

ITcUnknownのデフォルトの実装は、オブジェクトの最後の参照がリリースされた場合はオブジェクトを削除します。そのため、TcRelease()が呼び出された後に、インターフェイスポインタを逆参照してはいけません。

8.2.2.1 メソッドITcUnknown:TcAddRef

このメソッドは参照カウンタをインクリメントします。

シンタックス

```
ULONG TcAddRef( )
```

戻り値

結果の参照カウント値。

説明

参照カウンタをインクリメントし、新しい値を返します。

8.2.2.2 メソッドITcUnknown:TcQueryInterface

インターフェイスID (IID)によって与えられるインターフェイスに関するインターフェイスポインタの確認。

シンタックス

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

iid: (タイプ: RITCID) インターフェイスIID

pipItf: (PPVOIDタイプ) インターフェイスポインタへのポインタ。リクエストされるインターフェイスタイプが対応するインスタンスで使用可能である場合に設定されます。

戻り値

戻り値S_OKは成功を示します。

リクエストされるインターフェイスが使用可能でない場合、このメソッドはADS_E_NOINTERFACEを返します。

説明

IIDによる実装されたインターフェイス参照。インターフェイスポインタを初期化および保持するには、スマートポインタの使用を推奨します。

バリアント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;
}
}
```

スマートポインタに関連付けられているインターフェイスIDは、TcQueryInterfaceのパラメータとして使用することができます。演算子“&”は、スマートポインタの内部インターフェイスポインタメンバへのポインタを返します。バリアント1は、TcQueryInterfaceが成功を示す場合にインターフェイスポインタが初期化されることを想定しています。スコープが残されている場合は、スマートポインタspObjのデストラクタが参照をリリースします。

バリアント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;
}
```

インターフェイスポインタIPをスマートポインタspObjに割り当てる場合は、参照するインスタンスIPのIID_ITComObjectによってメソッドTcQueryInterfaceが暗黙に呼び出されます。これによりコードは短縮されますが、TcQueryInterfaceの元の戻りコードが失われます。

8.2.2.3 メソッドITcUnknown:TcRelease

このメソッドは参照カウンタをデクリメントします。

シンタックス

```
ULONG TcRelease( )
```

戻り値

結果の参照カウント値。

説明

参照カウンタをデクリメントし、新しい値を返します。

参照カウンタが0の場合、オブジェクトは自身を削除します。