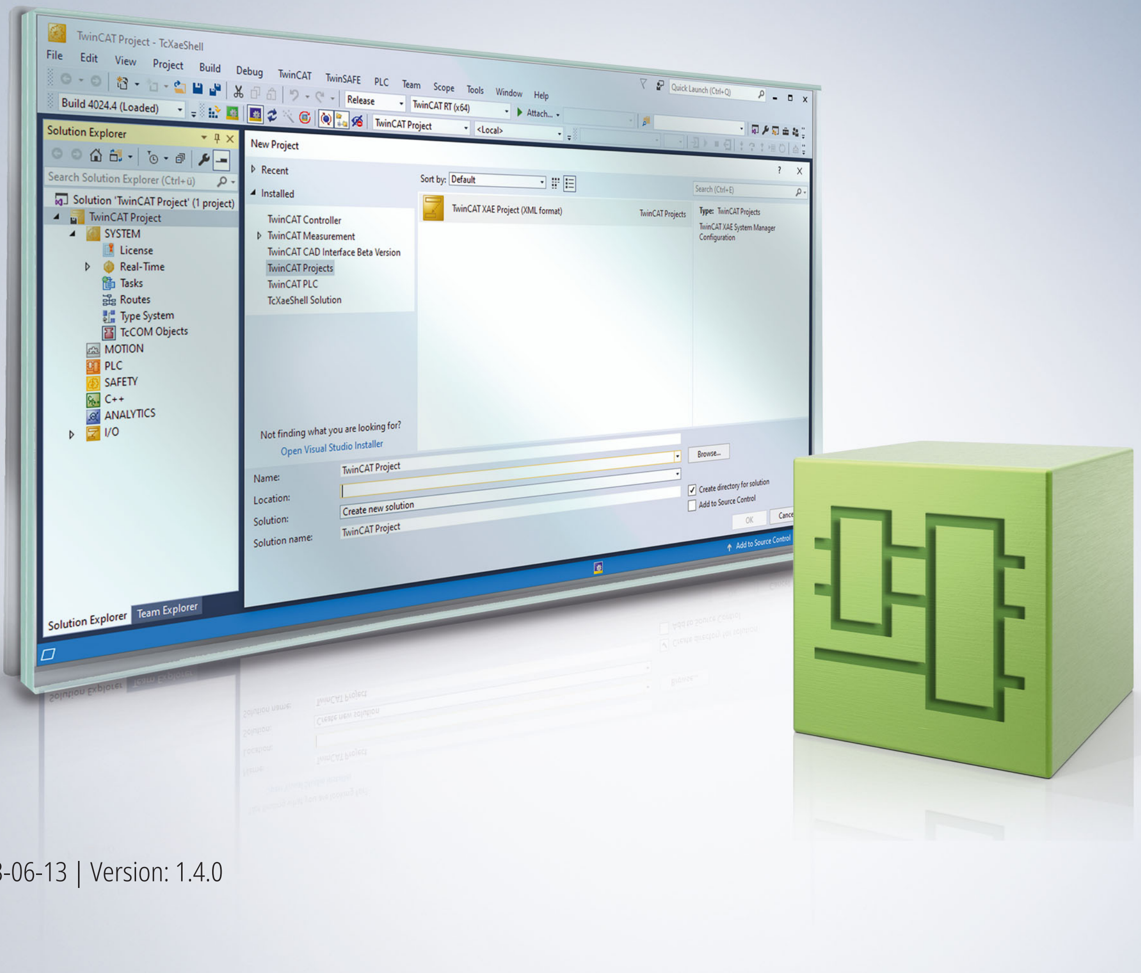
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC Library: Tc3_Module


TwinCAT Project - TcXaeShell
Solution Explorer
New Project
TwinCAT XAE Project (XML format)

2023-06-13 | Version: 1.4.0

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 For your safety

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Signal words

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

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

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

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

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

1.3 Notes on information security

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

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

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

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

2 Introduction

The PLC library Tc3_Module is used for TcCOM communication.

System requirements

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 Function blocks

The PLC library Tc3_Module offers function blocks in order to communicate from module to module via TcCOM. The module can be a TwinCAT system component, a C++ object, a Matlab object or also objects in the PLC.

3.1 TcBaseModuleRegistered

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

Description

If something is inherited from this object, a TcCOM object can be created from a function block. The object is automatically registered at the object server and ramped up to OP state. The own object ID is provided as a process image variable. Methods which are additionally implemented and are to be offered via this object must have a return value of the type HRESULT and must be implemented in a thread-safe manner. For more information, refer to the chapter 'Multi-task data access synchronization in the PLC'. How to create this TcCOM object and use it globally in the TwinCAT system is explained in detail in an example [▶ 36]. The TcBaseModule base class implements the ITComObject interface, which in turn expands the ITcUnknown interface.

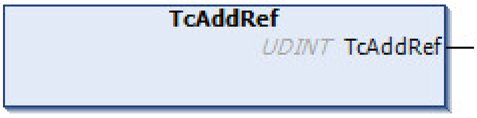
ITComObject Interface

The ITComObject interface is implemented by every TwinCAT module. It makes functionalities available regarding the state machine and Information from/to the TwinCAT system.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.1 TcAddRef



The TcAddRef() method increments the reference counter and returns the new value.

Return value

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

Name	Type	Description
TcAddRef	UDINT	The resulting reference count value is returned.

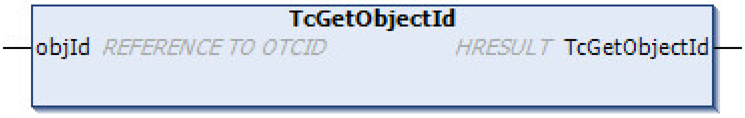
Inputs

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.2 TcGetObjectId



The method TcGetObjectId saves the object ID with the help of the given OTCID reference.

Return value

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

Name	Type	Description
TcGetObjectId	HRESULT	Gives information about success of the OTCID query.

Inputs

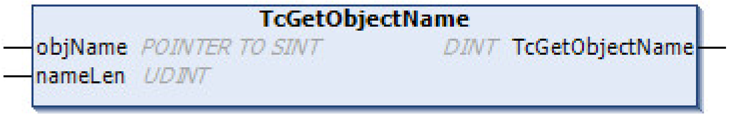
```
VAR_INPUT
    objId : REFERENCE TO OTCID;
END_VAR
```

Name	Type	Description
objId	REFERENCE TO OTCID	Reference to OTCID value

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.3 TcGetObjectName



The method TcGetObjectName saves the object names in the buffer with the given length.

Return value

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

Name	Type	Description
TcGetObjectName	DINT	Gives information about success of the name query.

Inputs

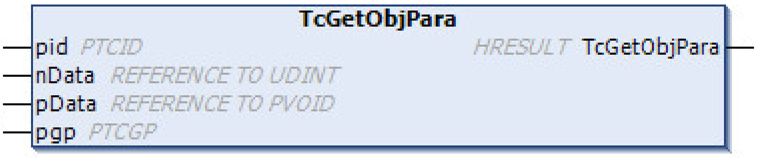
```
VAR_INPUT
    objName : POINTER TO SINT;
    nameLen : UDINT;
END_VAR
```

Name	Type	Description
objName	POINTER TO SINT	The name to be set
nameLen	UDINT	The maximum length of the name to be written

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.4 TcGetObjPara



The TcGetObjPara method queries an object parameter identified by means of its PTCID.

Return value

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

Name	Type	Description
TcGetObjPara	HRESULT	Gives information about success of the object parameter query.

Inputs

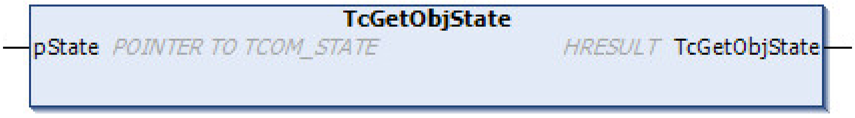
```
VAR_INPUT
    pid   : PTCID;
    nData : REFERENCE TO UDINT;
    pData : REFERENCE TO PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Type	Description
pid	PTCID	Parameter-ID of the object parameter
nData	REFERENCE TO UDINT	Maximum length of the data
pData	REFERENCE TO PVOID	Pointer to the data
Pgp	PTCGP	Reserved for future expansion. Pass NULL.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.5 TcGetObjState



The TcGetObjState method queries the current state of the object.

Return value

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

Name	Type	Description
TcGetObjState	HRESULT	Gives information about success of the state query.

Inputs

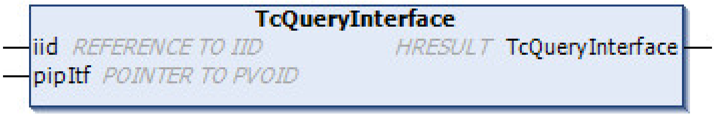
```
VAR_INPUT
    pState : POINTER TO TCOM_STATE;
END_VAR
```

Name	Type	Description
pState	POINTER TO TCOM_STA TE	Pointer to the state

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.6 TcQueryInterface



The method queries the reference at an implemented interface over the ID.

Return value

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

Name	Type	Description
TcQueryInterface	HRESULT	Informs about success of the interface query. If the requested interface is not available, the method returns ADS_E_NOINTERFACE.

Inputs

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

Name	Type	Description
iid	REFERENCE TO IID	Interface ID
pipItf	POINTER TO PVOID	Pointer to interface pointer. Is set when the requested interface type is available from the corresponding instance.

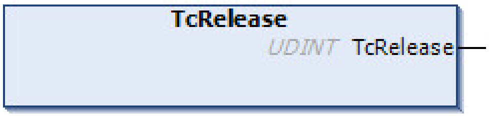
Necessary release of the interface pointers

You must explicitly release all references again. We recommend to use FW_SafeRelease [▶ 26] in order to perform a release of the interface pointer after use. Frequently the release of the references is implemented in the destructor of the object.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.7 TcRelease



The TcRelease() method decrements the reference counter and returns the new value. If the reference counter is 0, the object deletes itself.

Return value

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

Name	Type	Description
TcRelease	UDINT	The resulting reference count value is returned.

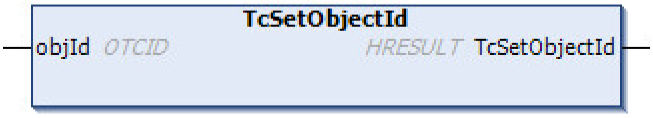
Inputs

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.8 TcSetObjId



The TcSetObjectId method sets the object ID of the object to the given OTCID.

Return value

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

Name	Type	Description
TcSetObjId	HRESULT	Gives information about success of the ID change.

Inputs

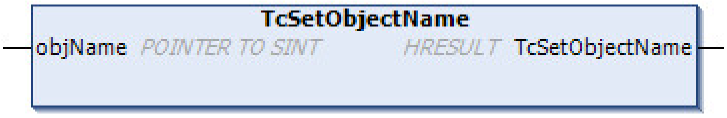
```
VAR_INPUT
    objId : OTCID;
END_VAR
```

Name	Type	Description
objId	OTCID	The OTCID to be set

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.9 TcSetObjectName



The TcSetObjectName method sets the object name of the object.

Return value

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

Name	Type	Description
TcSetObjectName	HRESULT	Gives information about the success of the name change.

Inputs

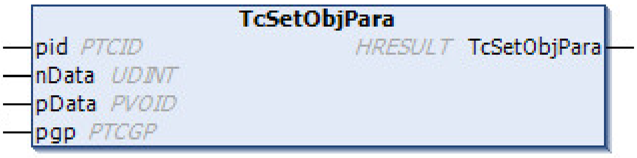
```
VAR_INPUT
    objName : POINTER TO SINT;
END_VAR
```

Name	Type	Description
objName	POINTER TO SINT	The name to be set of the object

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.10 TcSetObjPara



The TcSetObjPara method sets an object parameter identified by means of its PTCID.

Return value

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

Name	Type	Description
TcSetObjPara	HRESULT	Gives information about success of the parameter change.

Inputs

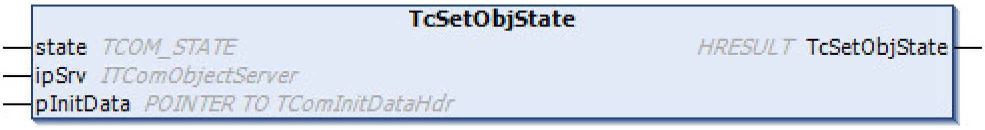
```
VAR_INPUT
    pid   : PTCID;
    nData : UDINT;
    pData : PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Type	Description
pid	PTCID	Parameter-ID of the object parameter
nData	UDINT	Maximum length of the data
pData	PVOID	Pointer to the data
pgp	PTCGPkl	Reserved for future expansion, pass NULL.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.1.11 TcSetObjState



The TcSetObjState method initializes a transition to the given state.

Return value

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

Name	Type	Description
TcSetObjState	HRESULT	Gives information about success of the state change.

Inputs

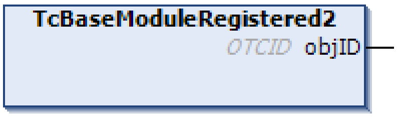
```
VAR_INPUT
    state     : TCOM_STATE;
    ipSrv     : ITComObjServer;
    pInitData : POINTER TO TComInitDataHdr;
END_VAR
```

Name	Type	Description
state	TCOM_STATE	Displays the new state.
ipSrv	ITComObjServer	Object description
pInitData	POINTER TO TComInitDataHdr	Points to a list of parameters (optional).

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2 TcBaseModuleRegistered2



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

Description

If something is inherited from this object, a TcCOM object can be created from a function block. The object is automatically registered at the object server and ramped up to OP state. The own object ID is provided at the output.

Methods which are additionally implemented and are to be offered via this object must have a return value of the type HRESULT and must be implemented in a thread-safe manner. For more information, refer to chapter 'Multi-task data access synchronization in the PLC'. How to create this TcCOM object and use it globally in the TwinCAT system is explained in detail in an example for TcBaseModuleRegistered [▶ 36]. The TcBaseModule base class implements the ITComObject interface, which in turn expands the ITcUnknown interface.

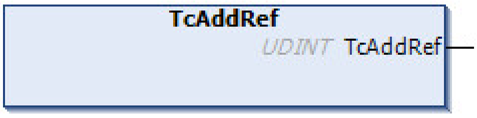
ITComObject Interface

The ITComObject interface is implemented by every TwinCAT module. It makes functionalities available regarding the state machine and Information from/to the TwinCAT system.

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024	x86, x64, ARM	Tc3_Module >= v3.3.23.0

3.2.1 TcAddRef



The TcAddRef() method increments the reference counter and returns the new value.

Return value

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

Name	Type	Description
TcAddRef	UDINT	The resulting reference count value is returned.

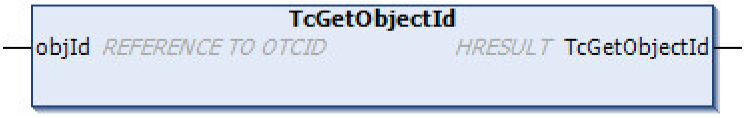
Inputs

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.2 TcGetObjectId



The method TcGetObjectId saves the object ID with the help of the given OTCID reference.

Return value

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

Name	Type	Description
TcGetObjectId	HRESULT	Gives information about success of the OTCID query.

Inputs

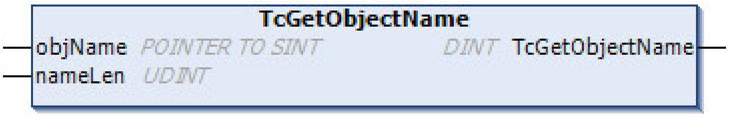
```
VAR_INPUT
    objId : REFERENCE TO OTCID;
END_VAR
```

Name	Type	Description
objId	REFERENCE TO OTCID	Reference to OTCID value

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.3 TcGetObjectName



The method TcGetObjectName saves the object names in the buffer with the given length.

Return value

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

Name	Type	Description
TcGetObjectName	DINT	Gives information about success of the name query.

Inputs

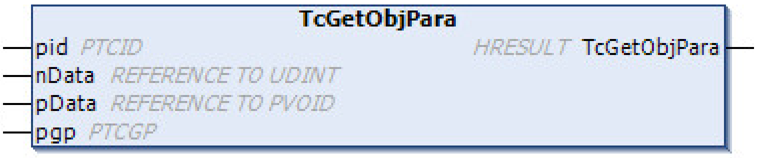
```
VAR_INPUT
    objName : POINTER TO SINT;
    nameLen : UDINT;
END_VAR
```

Name	Type	Description
objName	POINTER TO SINT	The name to be set
nameLen	UDINT	The maximum length of the name to be written

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.4 TcGetObjPara



The TcGetObjPara method queries an object parameter identified by means of its PTCID.

Return value

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

Name	Type	Description
TcGetObjPara	HRESULT	Gives information about success of the object parameter query.

Inputs

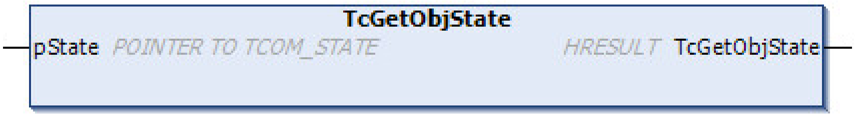
```
VAR_INPUT
    pid   : PTCID;
    nData : REFERENCE TO UDINT;
    pData : REFERENCE TO PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Type	Description
pid	PTCID	Parameter-ID of the object parameter
nData	REFERENCE TO UDINT	Maximum length of the data
pData	REFERENCE TO PVOID	Pointer to the data
Pgp	PTCGP	Reserved for future expansion. Pass NULL.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.5 TcGetObjState



The TcGetObjState method queries the current state of the object.

Return value

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

Name	Type	Description
TcGetObjState	HRESULT	Gives information about success of the state query.

Inputs

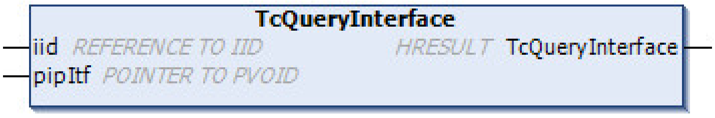
```
VAR_INPUT
    pState : POINTER TO TCOM_STATE;
END_VAR
```

Name	Type	Description
pState	POINTER TO TCOM_STATE	Pointer to the state

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.6 TcQueryInterface



The method queries the reference at an implemented interface over the ID.

Return value

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

Name	Type	Description
TcQueryInterface	HRESULT	Informs about success of the interface query. If the requested interface is not available, the method returns ADS_E_NOINTERFACE.

Inputs

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

Name	Type	Description
iid	REFERENCE TO IID	Interface ID
pipItf	POINTER TO PVOID	Pointer to interface pointer. Is set when the requested interface type is available from the corresponding instance.

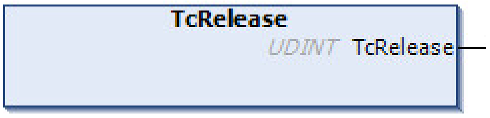
Necessary release of the interface pointers

You must explicitly release all references again. We recommend to use FW_SafeRelease [▶ 26] in order to perform a release of the interface pointer after use. Frequently the release of the references is implemented in the destructor of the object.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.7 TcRelease



The TcRelease() method decrements the reference counter and returns the new value. If the reference counter is 0, the object deletes itself.

Return value

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

Name	Type	Description
TcRelease	UDINT	The resulting reference count value is returned.

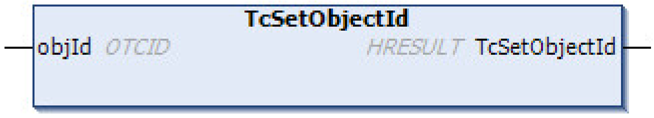
Inputs

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.8 TcSetObjId



The TcSetObjectId method sets the object ID of the object to the given OTCID.

Return value

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

Name	Type	Description
TcSetObjId	HRESULT	Gives information about success of the ID change.

Inputs

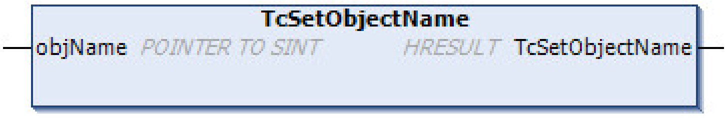
```
VAR_INPUT
    objId : OTCID;
END_VAR
```

Name	Type	Description
objId	OTCID	The OTCID to be set

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.9 TcSetObjectName



The TcSetObjectName method sets the object name of the object.

Return value

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

Name	Type	Description
TcSetObjectName	HRESULT	Gives information about the success of the name change.

Inputs

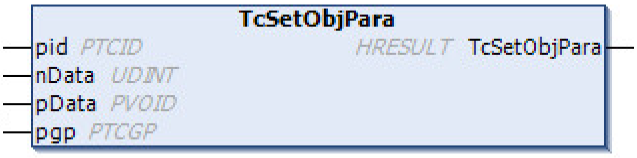
```
VAR_INPUT
    objName : POINTER TO SINT;
END_VAR
```

Name	Type	Description
objName	POINTER TO SINT	The name to be set of the object

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.10 TcSetObjPara



The TcSetObjPara method sets an object parameter identified by means of its PTCID.

Return value

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

Name	Type	Description
TcSetObjPara	HRESULT	Gives information about success of the parameter change.

Inputs

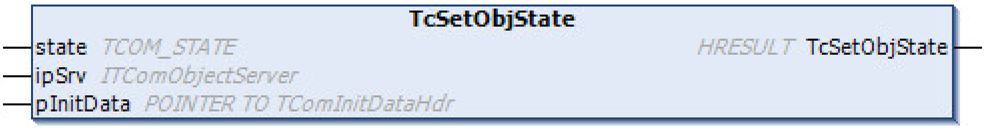
```
VAR_INPUT
    pid   : PTCID;
    nData : UDINT;
    pData : PVOID;
    pgp   : PTCGP;
END_VAR
```

Name	Type	Description
pid	PTCID	Parameter-ID of the object parameter
nData	UDINT	Maximum length of the data
pData	PVOID	Pointer to the data
pgp	PTCGPkl	Reserved for future expansion, pass NULL.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

3.2.11 TcSetObjState



The TcSetObjState method initializes a transition to the given state.

Return value

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

Name	Type	Description
TcSetObjState	HRESULT	Gives information about success of the state change.

Inputs

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

Name	Type	Description
state	TCOM_STATE	Displays the new state.
ipSrv	ITComObjServer	Object description
pInitData	POINTER TO TComInitDataHdr	Points to a list of parameters (optional).

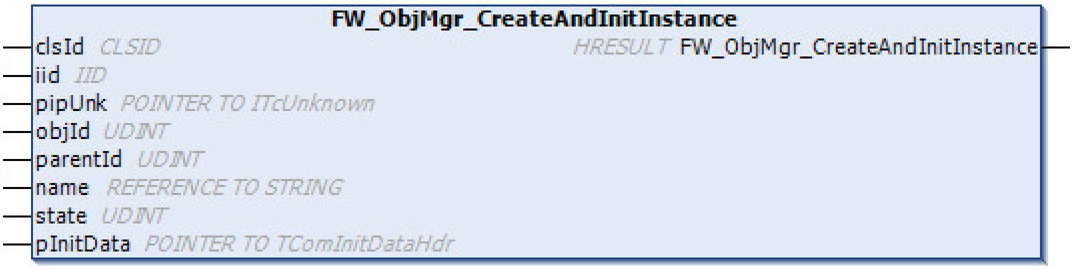
Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4 Functions

The PLC library Tc3_Module offers functions, in order to communicate from module to module via TcCOM. The module can be a TwinCAT system component, a C++ object, a Matlab object or also objects in the PLC.

4.1 FW_ObjMgr_CreateAndInitInstance


FW_ObjMgr_CreateAndInitInstance
clsId CLSID
iid IID
pipUnk POINTER TO ITcUnknown
objId UDINT
parentId UDINT
name REFERENCE TO STRING
state UDINT
pInitData POINTER TO TComInitDataHdr
HRESULT FW_ObjMgr_CreateAndInitInstance

This function generates an instance of the class specified by means of Class-ID and at the same time returns an interface pointer to this object. In addition the object name and state into which the object is to be put, as well as optionally also initialization parameters can be specified.

Return value

```
FW_ObjMgr_CreateAndInitInstance : HRESULT;
```

Name	Type	Description
FW_ObjMgr_CreateAndInitInstance	HRESULT	Returns S_OK if the function call was successful.

Inputs

```
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	Type	Description
clsId	CLSID	Specifies the class from which an object should be created.
iid	IID	Specifies the interface ID to which an interface pointer should be referenced.
pipUnk	POINTER TO ITcUnknown	Returns the interface pointer to the created object.
objId	UDINT	Specifies the object ID for the newly created object. If the global constant OTCID_CreateNewId is entered here a new object ID is generated internally.
parentId	UDINT	Object ID of the parent object (optional) Here the object ID of the PLC instance can be specified from which this function is called. (TwinCAT_SystemInfoVarList._AppInfo.ObjId).
name	REFERENCE TO STRING	Specifies the object name which should be assigned for the newly created object.
State	UDINT	Specifies the state into which the newly created object should be put. Typically Operational (TCOM_STATE.TCOM_STATE_OP) is specified.
pInitData	POINTER TO TComInitDataHdr	Pointer to initialization parameter (optional)

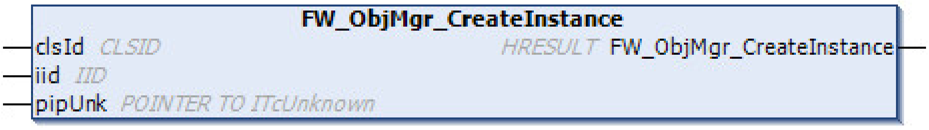
Necessary deletion of the object

A generated object must be explicitly deleted again. There is no Garbage-Collector as in .Net. It is recommended to use FW_ObjMgr_DeleteInstance [▶ 25], in order to delete the generated instance at the latest in the destructor of the object which created the instance.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.2 FW_ObjMgr_CreateInstance


FW_ObjMgr_CreateInstance
clsId CLSID
HRESULT FW_ObjMgr_CreateInstance
iid IID
pipUnk POINTER TO ITcUnknown

This function generates an instance of the class specified by means of Class-ID and at the same time returns an interface pointer to this object.

Return value

```
FW_ObjMgr_CreateInstance : HRESULT;
```

Name	Type	Description
FW_ObjMgr_CreateInstance	HRESULT	Returns S_OK if the function call was successful.

Inputs

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

Name	Type	Description
clsId	CLSID	Specifies the class from which an object should be created.
iid	IID	Specifies the interface ID to which an interface pointer should be referenced.
pipUnk	POINTER TO ITcUnknown	Returns the interface pointer to the created object.

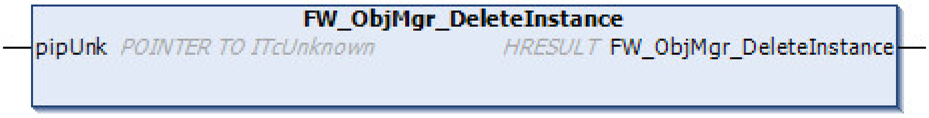
Necessary deletion of an object

A generated object must be explicitly deleted again. There is no Garbage-Collector as in .Net. We recommend to use FW_ObjMgr_DeleteInstance [▶ 25], in order to delete the generated instance at the latest in the destructor of the object which created the instance.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.3 FW_ObjMgr_DeleteInstance


FW_ObjMgr_DeleteInstance
pipUnk POINTER TO ITcUnknown
HRESULT FW_ObjMgr_DeleteInstance

This function puts the object in the Init state. After that the reference counter of the object is decremented, analogous to ITcUnknown.TcRelease(), and the interface pointer is set to zero at the same time.

Return value

```
FW_ObjMgr_DeleteInstance : HRESULT;
```

Name	Type	Description
FW_ObjMgr_DeleteInstance	HRESULT	Returns S_OK if the function call was successful.

Inputs

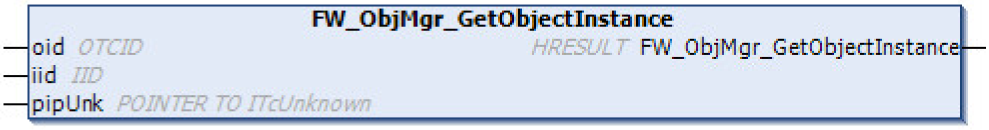
```
VAR_INPUT
    pipUnk : POINTER TO ITcUnknown;
END_VAR
```

Name	Type	Description
pipUnk	POINTER TO ITcUnknown	Specifies the address of the interface pointer to the object. The interface pointer is checked internally for null pointers.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.4 FW_ObjMgr_GetObjectInstance


FW_ObjMgr_GetObjectInstance
oid OTCID
iid IID
pipUnk POINTER TO ITcUnknown
HRESULT FW_ObjMgr_GetObjectInstance

This function returns an interface pointer to an object instance specified by means of object ID.

Return value

```
FW_ObjMgr_GetObjectInstance : HRESULT;
```

Name	Type	Description
FW_ObjMgr_GetObjectInstance	HRESULT	Returns S_OK if the function call was successful.

Inputs

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

Name	Type	Description
oid	OTCID	Object ID
iid	IID	Specifies the interface ID to which an interface pointer should be referenced.
pipUnk	POINTER TO ITcUnknown	Returns the interface pointer to the created object.

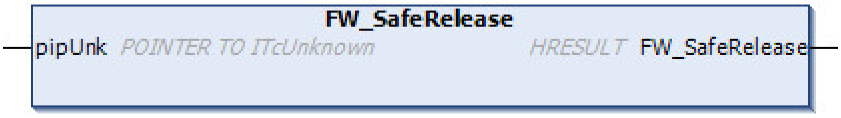
Necessary release of the interface pointers

All references must be explicitly released again. It is recommended to use FW_SafeRelease [▶ 26] in order to perform a release of the interface pointer after use. Frequently the release of the references is implemented in the destructor of the object.

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.5 FW_SafeRelease



This function decrements the reference counter of the object, analogous to ITcUnknown.TcRelease(), and at the same time sets the interface pointer to zero.

Return value

```
FW_SafeRelease : HRESULT;
```

Name	Type	Description
FW_SafeRelease	HRESULT	Returns S_OK if the function call was successful.

Inputs

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

Name	Type	Description
pipUnk	POINTER TO ITcUknown	Specifies the address of the interface pointer to the object. The interface pointer is checked internally for null pointers.

Example

This function can for example be called in the destructor of the object family, which holds an interface pointer to another object.

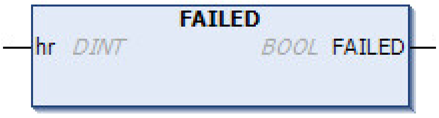
```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied
 afterwards (online change).
END_VAR
-----------------------------------------------------------------------
```

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.6 FAILED



Error codes or status codes of the type HRESULT are checked with this function for invalidity.

Return value

```
FAILED : BOOL;
```

Name	Type	Description
FAILED	BOOL	Returns TRUE if an error is present.

Inputs

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

Name	Type	Description
hr	DINT	Specification of the error code or status code of type HRESULT to be checked.

HRESULT

The type HRESULT has the special feature that errors are represented by negative values. Warnings or information can optionally be output by means of positive values.

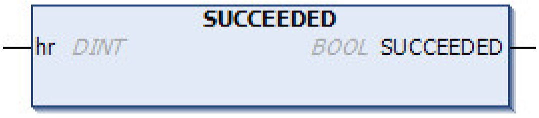
Declaration	Error range	No error	Message/info	Check functions

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.7 SUCCEEDED



Error codes or status codes of the type HRESULT are checked with this function for validity.

Return value

```
SUCCEEDED : BOOL;
```

Name	Type	Description
SUCCEEDED	BOOL	Returns TRUE if no error.

Inputs

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

Name	Type	Description
hr	DINT	Specification of the error code or status code of type HRESULT to be checked.

HRESULT

The type HRESULT has the special feature that errors are represented by negative values. Warnings or information can optionally be output by means of positive values.

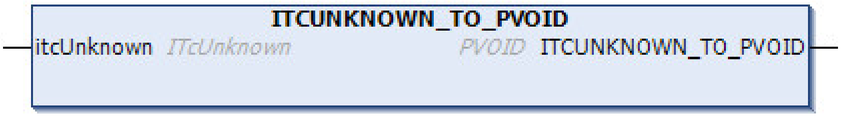
Declaration	Error range	No error	Message/info	Check functions

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.8 ITCUNKNOWN_TO_PVOID



This conversion function converts an interface pointer of the type ITcUnknown to a pointer to VOID.

Return value

```
ITCUNKNOWN_TO_PVOID : PVOID
```

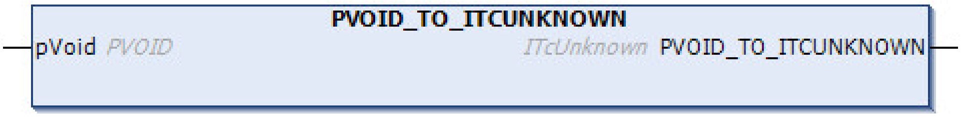
Inputs

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.9 PVOID_TO_ITCUNKNOWN



This conversion function converts a pointer to VOID to an interface pointer of the type ITcUnknown.

Return value

```
PVOID_TO_ITCUNKNOWN : ITcUnknown;
```

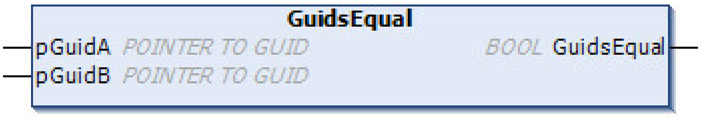
Inputs

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

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

4.10 GuidsEqual



The function GuidsEqual checks two GUID objects for their equality to one another.

Return value:

```
GuidsEqual : BOOL;
```

Name	Type	Description
GuidsEqual	BOOL	The method returns TRUE when both arguments are equal.

Inputs

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

Name	Type	Description
pGuidA	POINTER TO GUID	Pointer to GUID object
pGuidB	POINTER TO GUID	Pointer to GUID object

Requirements

TwinCAT Version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

5 Global Constants

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	Type	Value	Use	Meaning
S_OK	HRESULT	0		This constant can be used, to designate error-free processing in an HRESULT status code.
S_FALSE	HRESULT	1		This constant indicates successful processing, although the result was negative or incomplete.
S_PENDING	HRESULT	16#203		This constant indicates successful processing, although no result is available yet.
S_WATCHDOG_TIMEOUT	HRESULT	16#256		This constant indicates successful processing, although a timeout occurred. Depending on the function, the desired processing was aborted.
OTCID_CreateNewId	OTCID	16#FFFFFFFF	FW_ObjMgr_CreateAndInitInstance [▶ 23]	This constant is used to generate a new object ID.
OTCID_FirstFreeId	OTCID	16#71010000		
OTCID_LastFreeId	OTCID	16#710FFFFF		
NULL	PVOID	0		NULL pointer

5.2 Global_Version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

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

Name	Type	Description
stLibVersion_Tc3_Module	ST_LibVersion	Version information of the Tc3_Module library

To check whether the version you have is the version you need, use the function F_CmpLibVersion (defined in the Tc2_System PLC library).

6 Error Codes

The return values of the functions and methods are output by the type HRESULT.

HighWord of HRESULT	Group of error codes
16#9811	Ads Error codes

6.1 ADS Return Codes

Grouping of error codes: 0x000 [▶ 32]..., 0x500 [▶ 32]..., 0x700 [▶ 33]..., 0x1000 [▶ 35]...

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x9811 0000	ERR_NOERROR	No error.
0x1	1	0x9811 0001	ERR_INTERNAL	Internal error.
0x2	2	0x9811 0002	ERR_NORTIME	No real-time.
0x3	3	0x9811 0003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x9811 0004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x9811 0005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x9811 0006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x9811 0007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x9811 0008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x9811 0009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811 000A	ERR_NOIO	No IO.
0xB	11	0x9811 000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811 000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811 000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811 000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811 000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x9811 0010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x9811 0011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x9811 0012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x9811 0013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x9811 0014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x9811 0015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x9811 0016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x9811 0017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x9811 0018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x9811 0019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811 001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811 001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811 001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811 001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811 001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x9811 0500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x9811 0501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x9811 0502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x9811 0503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x9811 0504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x9811 0505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x9811 0506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x9811 0507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x9811 0508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x9811 0509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811 050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811 050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811 050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811 050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x9811 0700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x9811 0701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x9811 0702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x9811 0703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x9811 0704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x9811 0705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x9811 0706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x9811 0707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x9811 0708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x9811 0709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS function blocks in different tasks. It may be possible to resolve this through Multi-task data access synchronization in the PLC.
0x70A	1802	0x9811 070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811 070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811 070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811 070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811 070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811 070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x9811 0710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x9811 0711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x9811 0712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x9811 0713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x9811 0714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x9811 0715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x9811 0716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x9811 0717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x9811 0718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x9811 0719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811 071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811 071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811 071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811 071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811 071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811 071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x9811 0720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x9811 0721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x9811 0722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x9811 0723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x9811 0724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x9811 0725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x9811 0726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x9811 0727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x9811 0728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x9811 0729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811 072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	License problem: Time in the future.
0x72B	1835	0x9811 072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811 072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811 072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811 072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811 072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x9811 0730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x9811 0731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x9811 0732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x9811 0733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x9811 0734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x9811 0735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.

Hex	Dec	HRESULT	Name	Description
0x736	1846	0x9811 0736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.
0x737	1847	0x9811 0737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x9811 0738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x9811 0739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real-time.
0x740	1856	0x9811 0740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x9811 0741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x9811 0742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x9811 0743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x9811 0744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x9811 0745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x9811 0746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x9811 0747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x9811 0748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x9811 0749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x9811 0750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x9811 0751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x9811 0752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x9811 0753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x9811 0754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x9811 0755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x9811 1000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x9811 1001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x9811 1002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x9811 1003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x9811 1004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x9811 1005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x9811 1006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x9811 1007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811 100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811 100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811 100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x9811 1010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x9811 1017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x9811 1018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x9811 1019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811 101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

7 Samples

The TcCOM_Sample01 sample [▶ 36] shows how TcCOM communication can take place between two PLCs. In the process functionalities from one PLC are directly called up from the other PLC.

The TcCOM_Sample02 sample [▶ 46] shows how a PLC application can use functionalities of an existing instance of a TwinCAT C++ class. In this way separate algorithms written C++ (or Matlab) can be used easily in the PLC.
Although in the event of the use of an existing TwinCAT C++ module the TwinCAT C++ license is required on the target system, a C++ development environment is not necessary on the target system or on the development computer.

The TcCOM_Sample03 sample [▶ 50] shows how a PLC application uses functionalities of a TwinCAT C++ class by generating an instance of C++ class at the same time. In comparison to the previous sample this can offer increased flexibility.

You will find additional programming examples in the documentation of TwinCAT 3 C++. For example, it describes an additional option for calling an algorithm written in C++ from a PLC program (Sample11). In contrast to TcCOM_Sample02, here a wrapper module is programmed that each interface method implements itself. Therefore this variant is more complex. However, if you have to forego interface pointers calling the functionalities in the PLC application due to users, this variant offers an option for doing this.

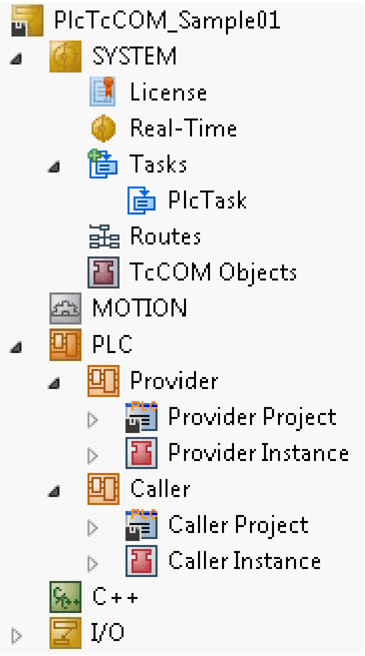
Another example in the documentation of TwinCAT 3 C++ shows how a TwinCAT C++ Module calls up a method of a function block of the PLC by TcCOM interface (Sample13).

7.1 TcCOM_Sample01_PlcToPlc

This sample describes a TcCOM communication between two PLCs.

Functionalities provided by a function block in the first PLC (also called "provider" in the sample), are called from the second PLC (also called "caller" in the sample). To this end it is not necessary for the function block or its program code to be copied. Instead the program works directly with the object instance in the first PLC.

Both PLCs must be in a TwinCAT runtime. In this connection a function block offers its methods system-wide via a globally defined interface and represents itself a TcCOM object. As is the case with every TcCOM object, such a function block is also listed at runtime in the **TcCOM Objects** node.



The procedure is explained in the following sub-chapters:

- Creating an FB in the first PLC that provides its functionality globally [▶ 37]
- Creating an FB in the second PLC that, as a simple proxy, also offers this functionality there [▶ 41]

- Execution of the sample project [▶ 44]

Downloading the sample: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/2343046667/.zip

Race Conditions in the case of Multi-Tasking (Multi-Threading) use

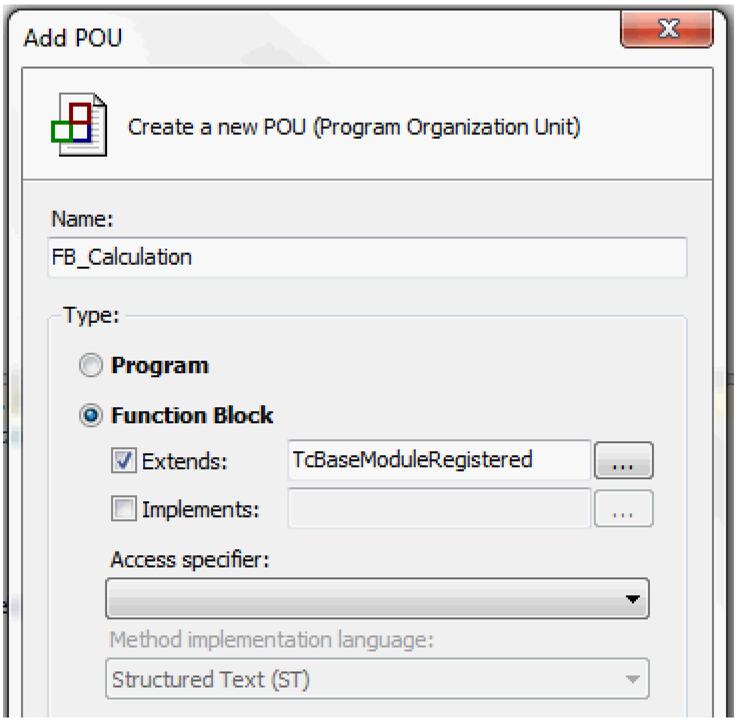
The function block that provides its functionality globally is instantiated in the first PLC. It can be used there like any function block. In addition, if it is used from a different PLC (or, for example, from a C++ module), make sure that the methods offered are thread-safe, as the various calls could take place simultaneously from different task contexts or mutually interrupt one another, depending on the system configuration. In this case the methods must not access member variables of the function block or global variables of the first PLC. If this should be absolutely necessary, prevent simultaneous access. Observe the function TestAndSet() from the Tc2_System library.

System requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

7.1.1 Creating an FB which provides its functionality globally in the first PLC

1. Create a PLC and prepare a new function block (FB) (here: FB_Calculation). Derive the function block from the TcBaseModuleRegistered [▶ 9] class, so that an instance of this function block is not only available in the same PLC, but can also be reached from a second.
   Note: as an alternative you can also modify an FB in an existing PLC.



Add POU

Create a new POU (Program Organization Unit)

Name: FB_Calculation

Type:

○ Program

● Function Block

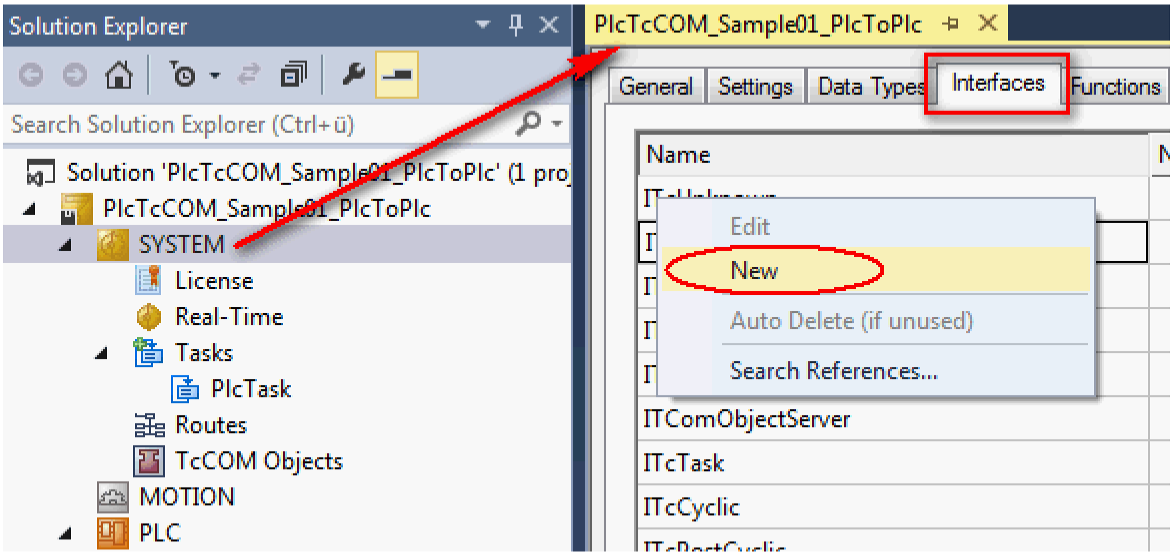
☑ Extends: TcBaseModuleRegistered

□ Implements:

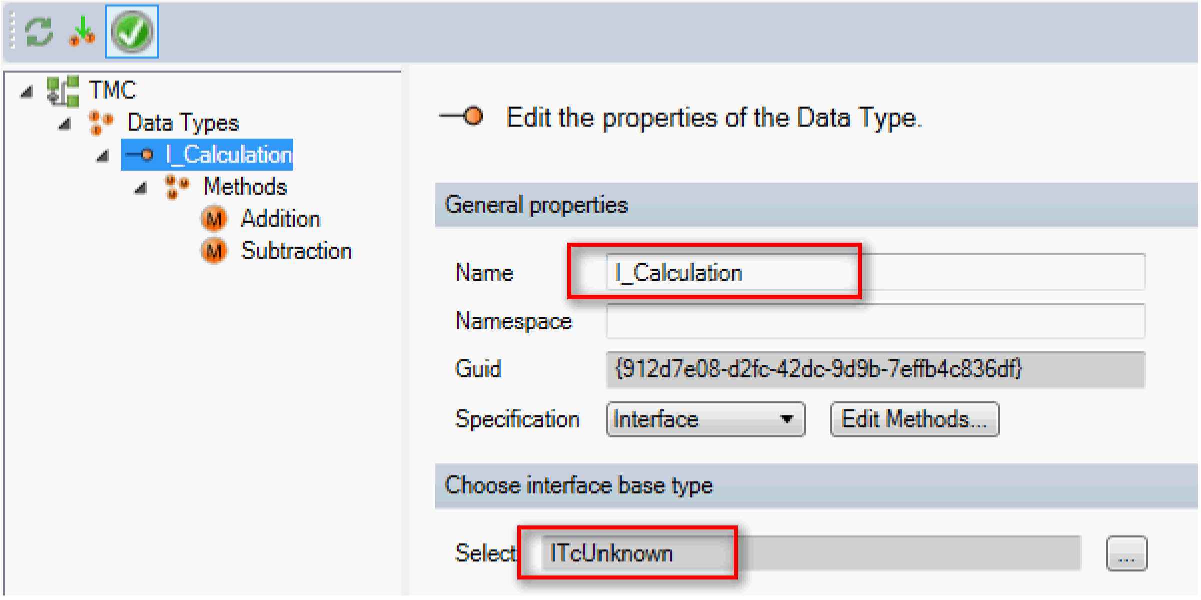
Access specifier:

Method implementation language: Structured Text (ST)

2. The function block must offer its functionality by means of methods. These are defined in a global interface, whose type is system-wide and known regardless of programming language. To create a global interface, open the Context menu in the “Interface” tab of System Properties and choose the option “New”.
   ⇨ The TMC Editor opens, which provides you with support in creating a global interface.



3. Specify the name (here: I_Calculation) and append the desired methods. The interface is automatically derived from ITcUnknown, in order to fulfill the TwinCAT TcCOM module concept.



Name: I_Calculation

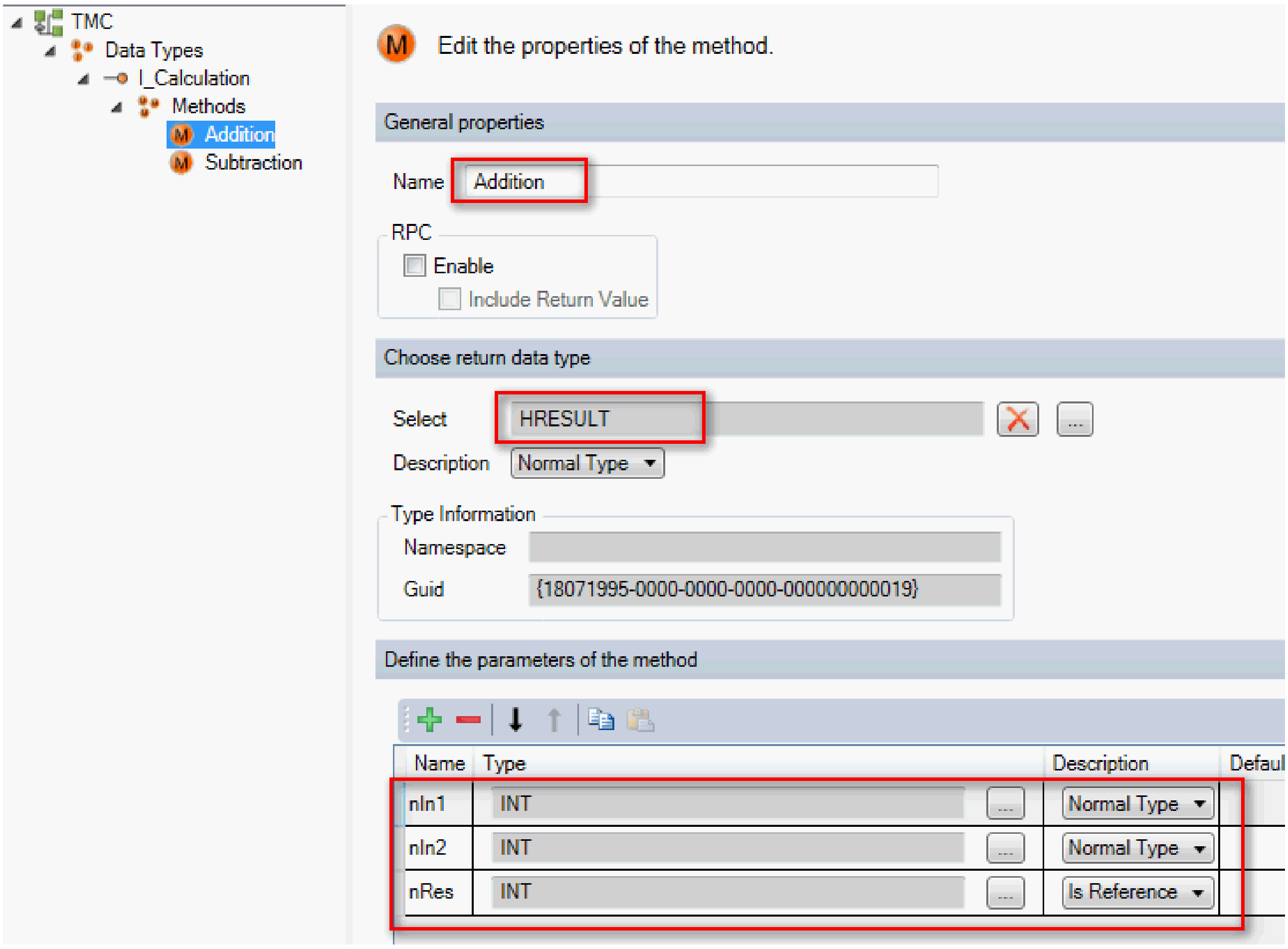
Namespace:

Guid: {912d7e08-d2fc-42dc-9d9b-7effb4c836df}

Specification: Interface

Select: ITcUnknown

4. Specify the name of the methods analogously (here: Addition() and Subtraction()) and select HRESULT as return data type. This return type is mandatory if this type of TcCOM communication should be implemented.
5. Specify the method parameters last and then close the TMC Editor.



Edit the properties of the method.

General properties

Name: Addition

RPC
□ Enable
□ Include Return Value

Choose return data type

Select: HRESULT

Description: Normal Type

Type Information

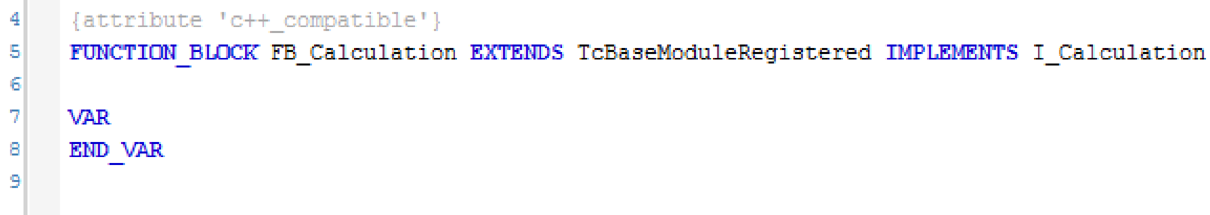
Namespace:

Guid: {18071995-0000-0000-0000-000000000019}

Define the parameters of the method

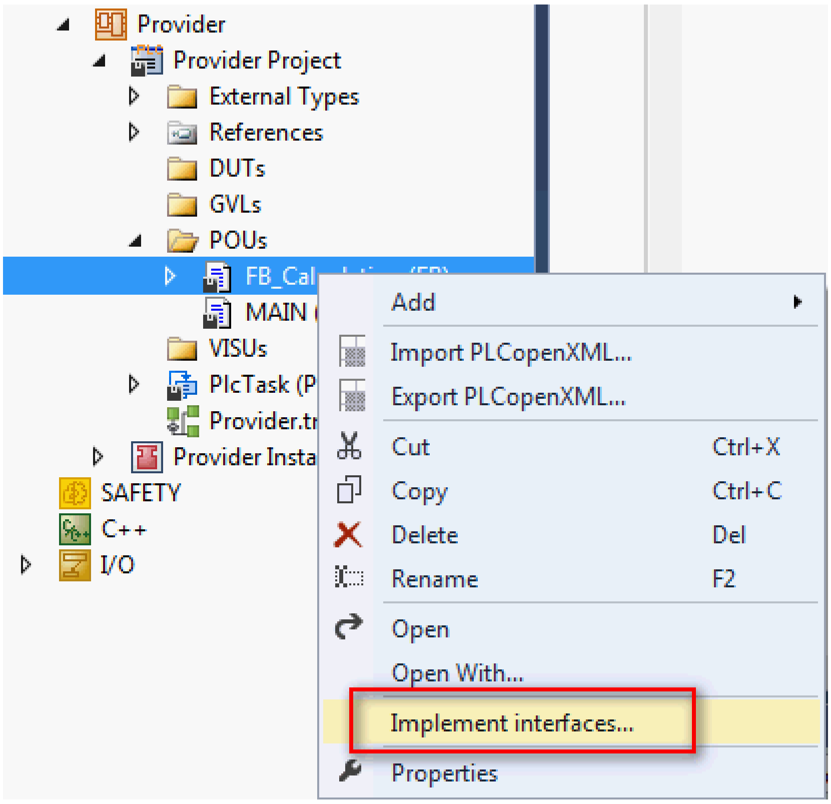
Name	Type	Description	Defaul
nIn1	INT	Normal Type	
nIn2	INT	Normal Type	
nRes	INT	Is Reference	

6. Now implement the I_Calculation interface in the FB_Calculation function block and append the c++_compatible attribute.

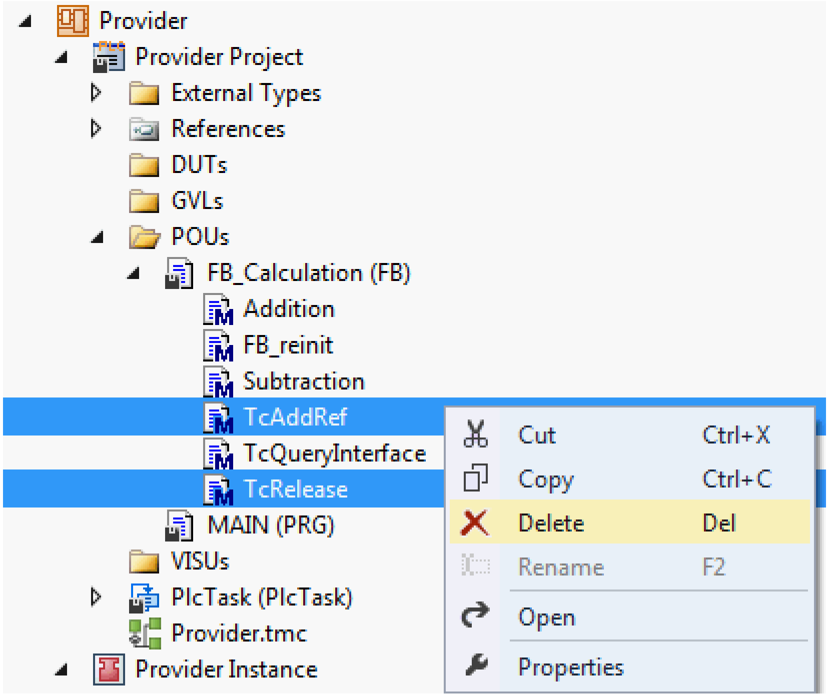


```
{attribute 'c++_compatible'}
FUNCTION_BLOCK FB_Calculation EXTENDS TcBaseModuleRegistered IMPLEMENTS I_Calculation

VAR
END_VAR

```

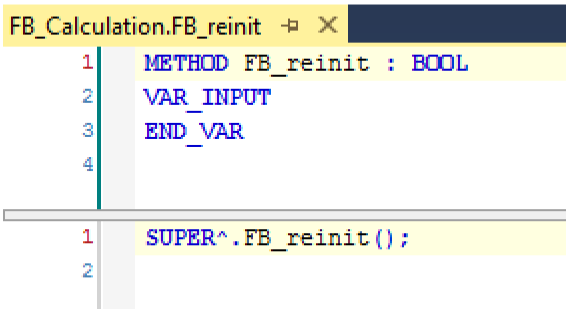
7. Choose the “Implement interfaces...” option in the Context menu of the function block in order to obtain the methods belonging to this interface.



8. Delete the two methods TcAddRef() and TcRelease() because the existing implementation of the base class should be used.



9. Create the FB_reinit() method for the FB_Calculation function block and call the basic implementation. This ensures that the FB_reinit() method of the base class will run during the online change. This is imperative.

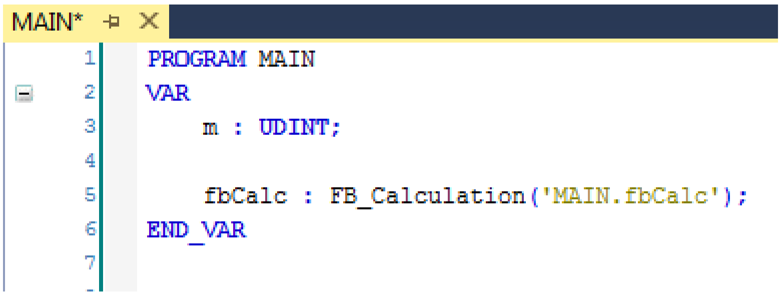


```
METHOD FB_reinit : BOOL
VAR_INPUT
END_VAR
```

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

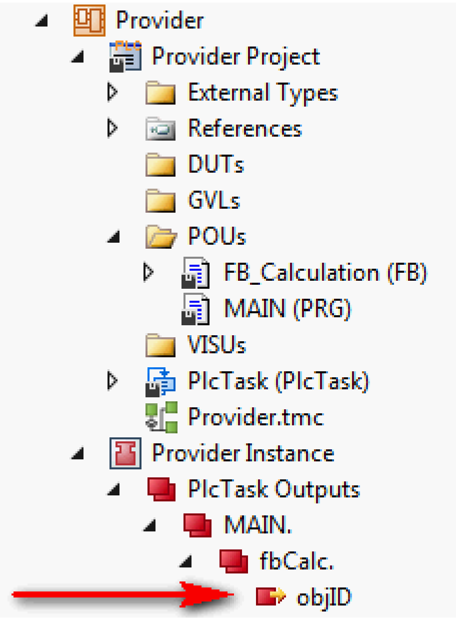
10. Implement the TcQueryInterface() method of the Interface ITcUnknown [▸ 73]. Via this method it is possible for other TwinCAT components to obtain an interface pointer to an instance of this function block and thus actuate method calls. The call for TcQueryInterface is successful if the function block or its base class provides the interface queried by means of iid (Interface ID). For this case the handed over interface pointer is allocated the address to the function block type-changed and the reference counter is incremented by means of TcAddRef().
11. Fill the two methods Addition() and Subtraction() with the corresponding code to produce the functionality: `nRes := nIn1 + nIn2` and `nRes := nIn1 - nIn2`
12. Add one or more instances of this function block in the MAIN program block or in a global variable list.

   ⇨ The implementation in the first PLC is complete.



```
PROGRAM MAIN
VAR
    m : UDINT;

    fbCalc : FB_Calculation('MAIN.fbCalc');
END_VAR
```

⇨ After compiling the PLC, the object ID of the TcCOM object which represents the instance of FB_Calculation is available as an outlet in the in the process image.

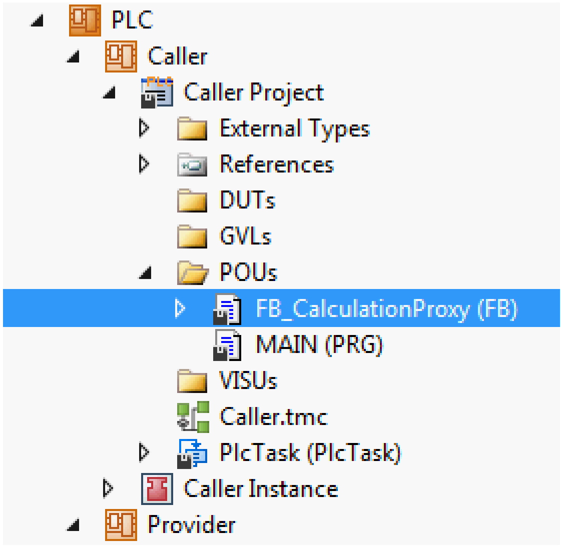


```
Provider
    Provider Project
        External Types
        References
        DUTs
        GVLs
        POUs
            FB_Calculation (FB)
            MAIN (PRG)
        VISUs
        PlcTask (PlcTask)
        Provider.tmc
    Provider Instance
        PlcTask Outputs
            MAIN.
                fbCalc.
                    objID
```

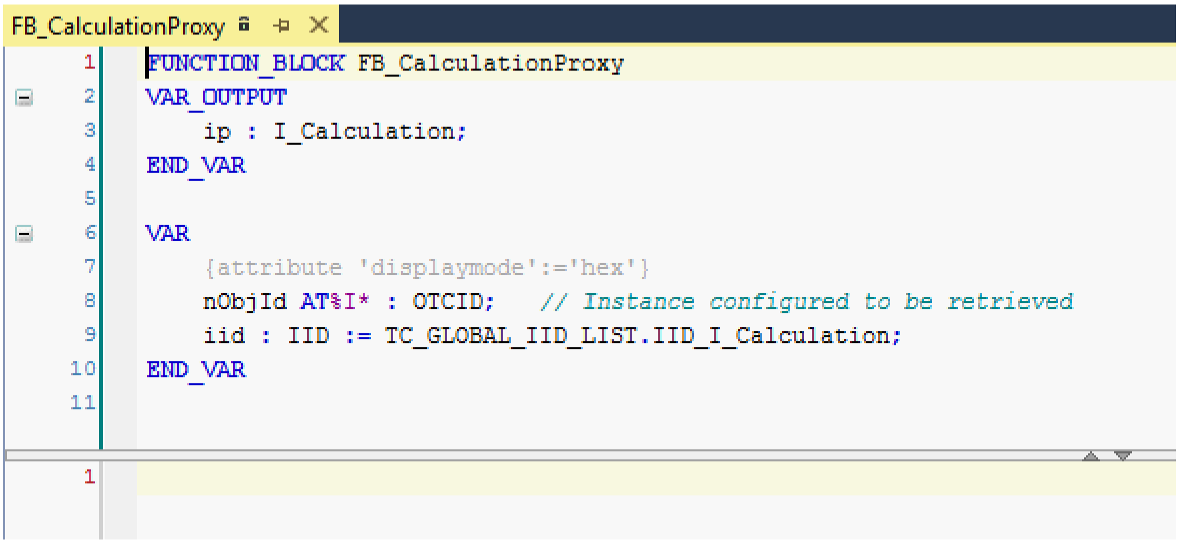
7.1.2 Creating an FB which likewise offers this functionality there as a simple proxy in the second PLC,

1. Create a PLC and append a new function block there.

   ⇨ This proxy function block should provide the functionality which was programmed in the first PLC. It does this via an interface pointer of the type of the global interface I_Calculation.

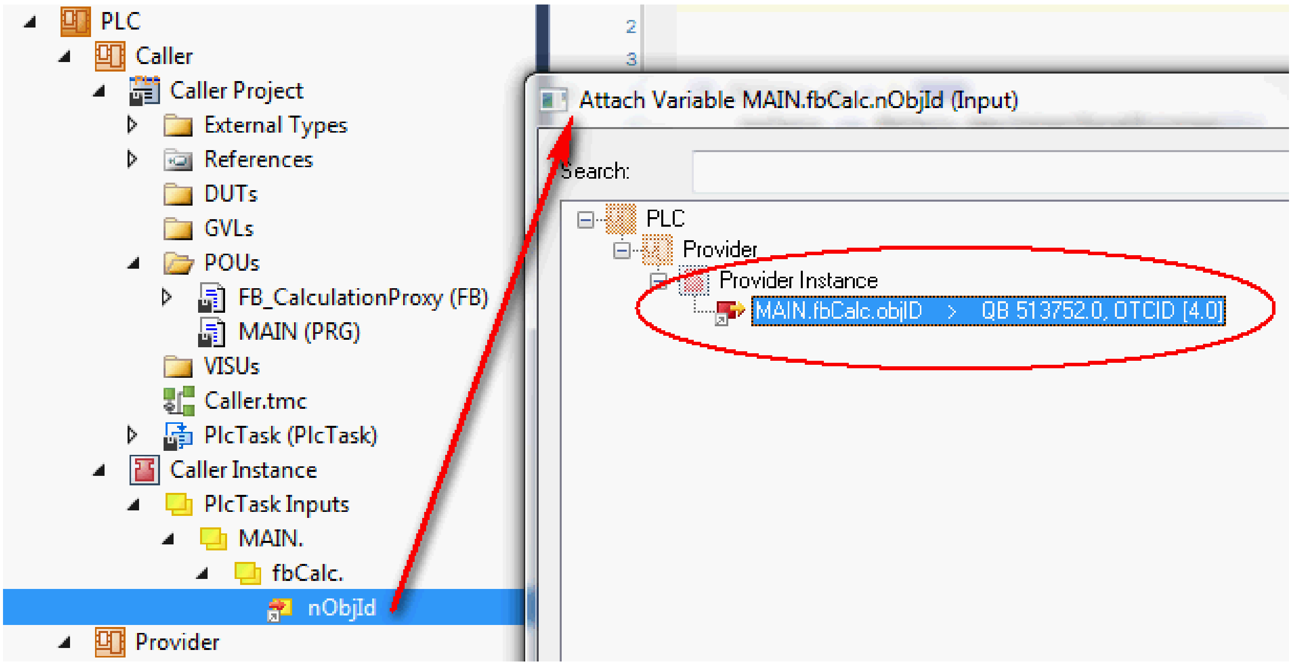


2. In the declaration part of the function block declare as an output an interface pointer to the global interface which later provides the functionality outward.

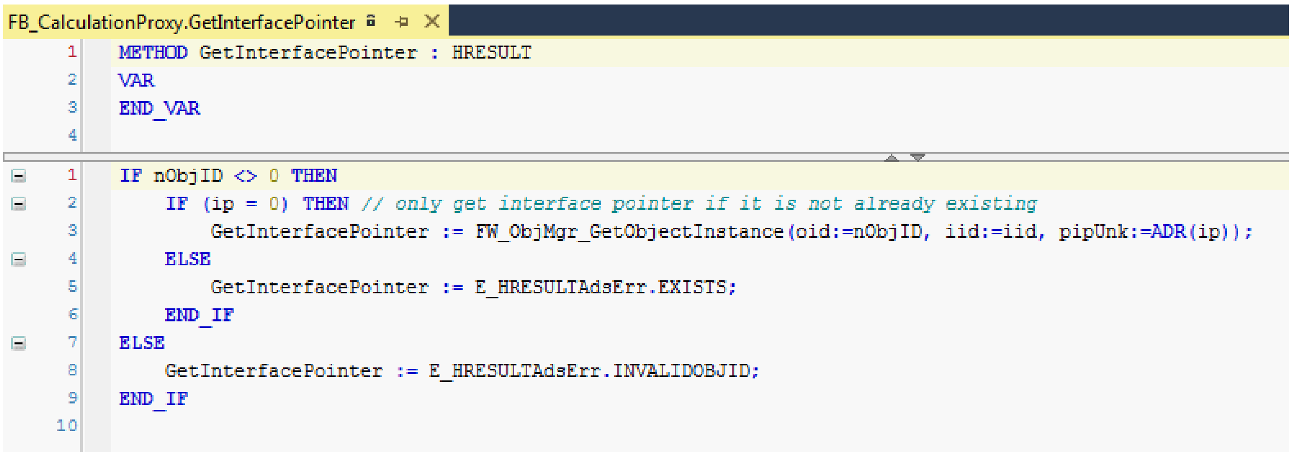


```
FUNCTION_BLOCK FB_CalculationProxy
VAR_OUTPUT
    ip : I_Calculation;
END_VAR

VAR
    {attribute 'displaymode':='hex'}
    nObjId AT%I* : OTCID;   // Instance configured to be retrieved
    iid : IID := TC_GLOBAL_IID_LIST.IID_I_Calculation;
END_VAR
```

3. In addition create the object ID and the interface ID as local member variables.
While the interface ID is already available via a global list, the object ID is assigned via a link in the process image.



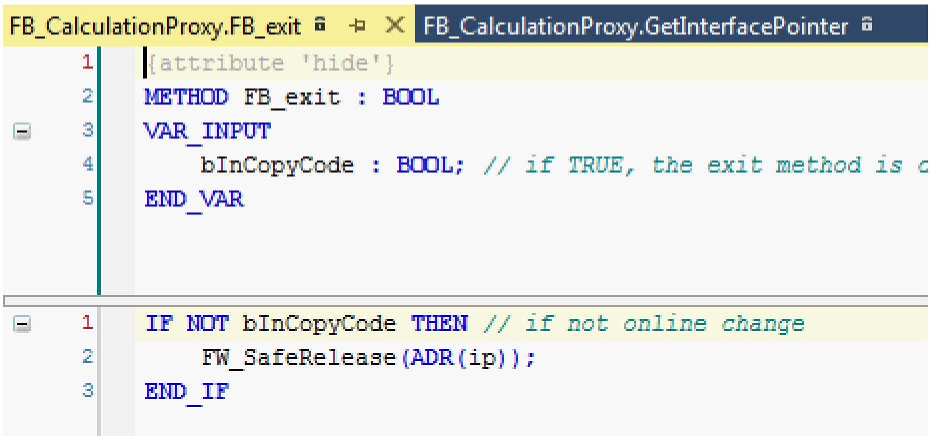
4. Implement the PLC proxy function block. First add the GetInterfacePointer() method to the function block. The interface pointer is fetched to the specified interface of the specified TcCOM object with the help of the FW_ObjMgr_GetObjectInstance() [▸ 25] function. This will only be executed if the object ID is valid and the interface pointer has not already been allocated. The object itself increments a reference counter.



```
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. It is imperative to release the used reference again. To this end call the FW_SafeRelease() function in the FB_exit destructor of the function block.

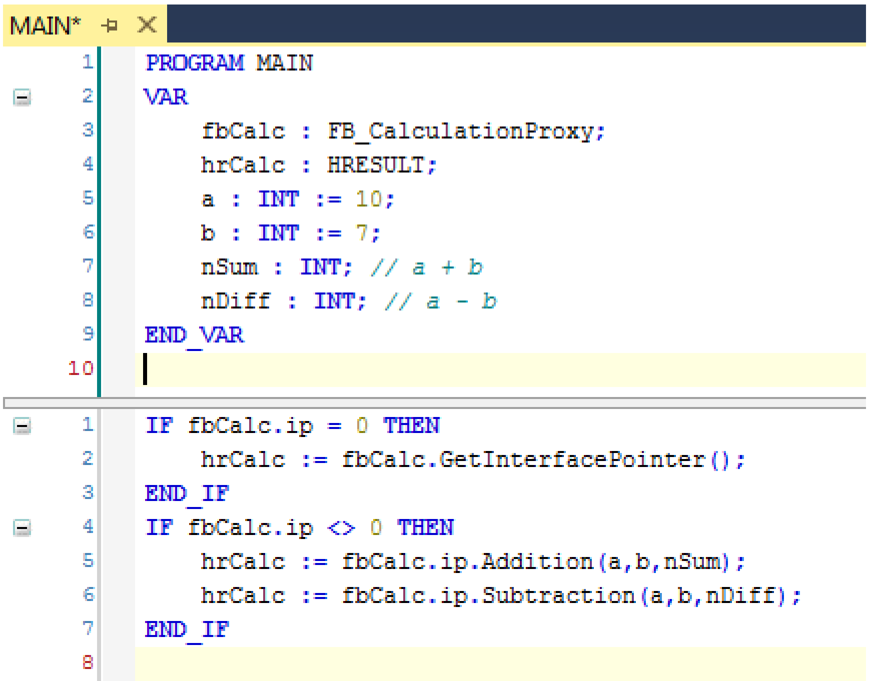

FB_CalculationProxy.FB_exit
FB_CalculationProxy.GetInterfacePointer

```
{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
```

⇨ This completes the implementation of the Proxy function block.

6. Instantiate the Proxy function block FB_CalculationProxy in the application and call its method GetInterfacePointer() to get a valid interface pointer.
An instance of the proxy block is declared in the application to call the methods provided via the interface. The calls themselves take all place over the interface pointer defined as output of the function block. As is typical for pointers a prior null check must be made. Then the methods can be called directly, also via Intellisense.



```
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

```

⇨ The sample is ready for testing.

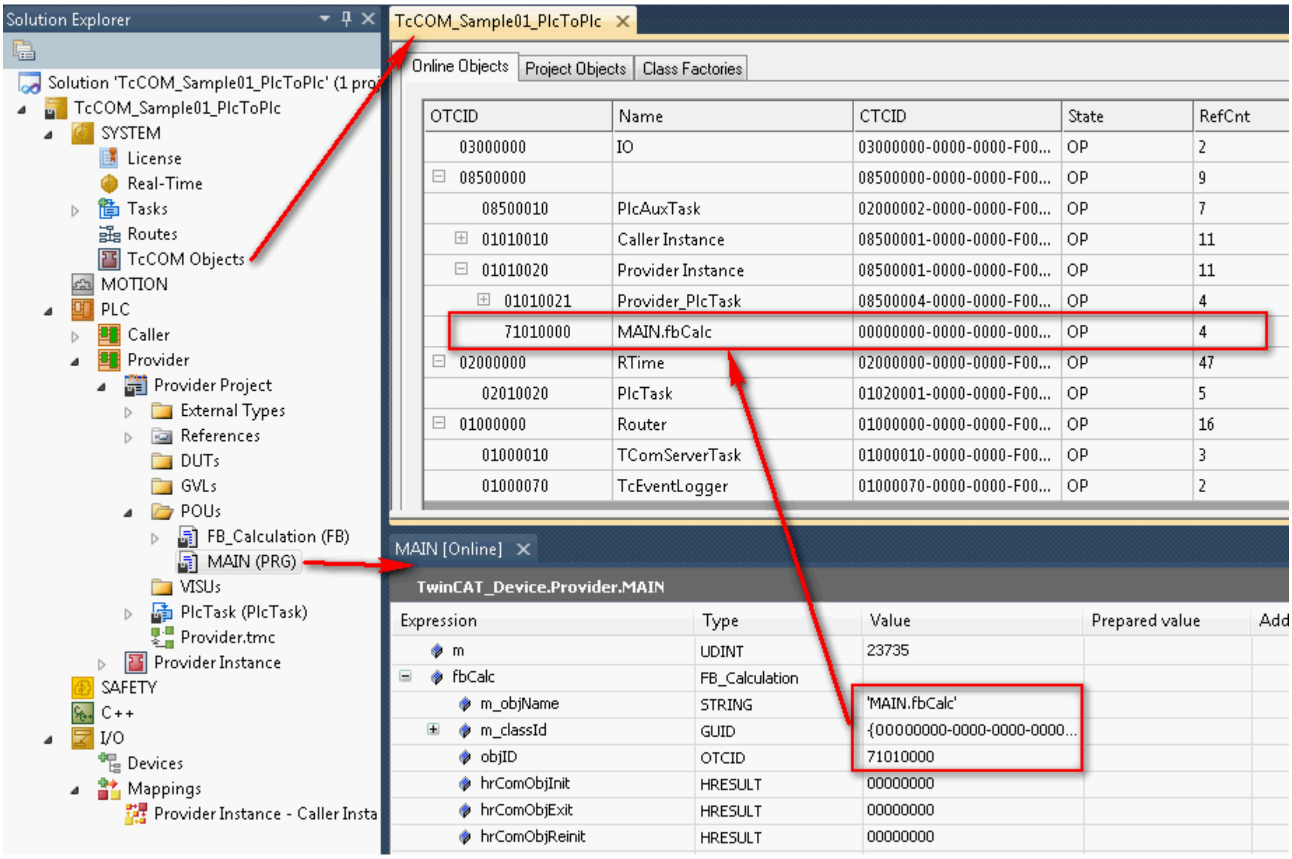
Order irrelevant

The sequence in which the two PLCs start later is irrelevant in this implementation.

7.1.3 Execution of the sample project

1. Select the destination system and compile the project.
2. Enable the TwinCAT configuration and execute a log-in and start both PLCs.

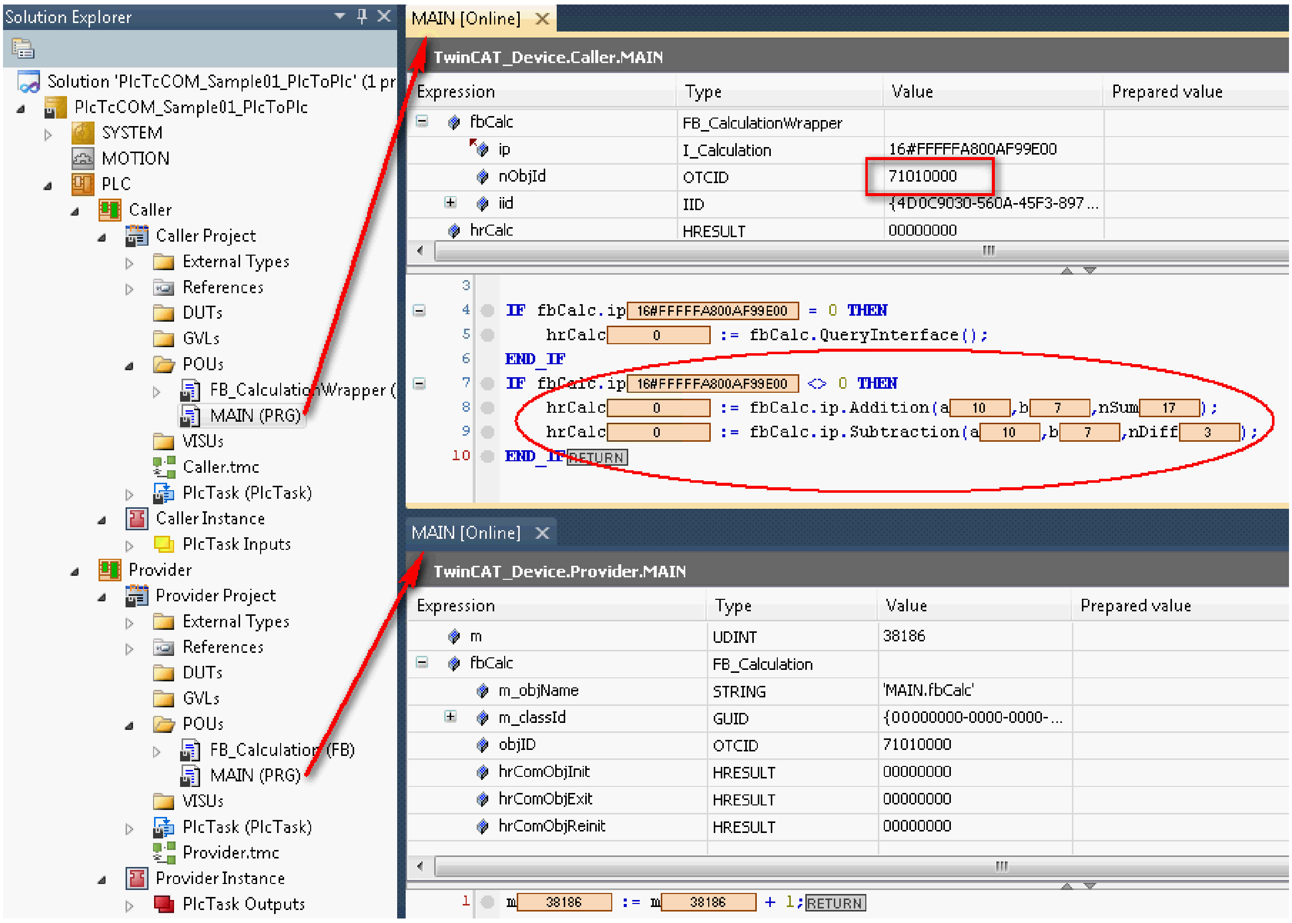
   ⇨ In the online view of the PLC application “Provider” the generated object ID of the C++ object can be seen in the PLC function block FB_Calculation. The project node “TcCOM Objects” keeps the generated object with its object ID and the selected name in its list.



OTCID	Name	CTCID	State	RefCnt
03000000	IO	03000000-0000-0000-F00...	OP	2
08500000		08500000-0000-0000-F00...	OP	9
08500010	PlcAuxTask	02000002-0000-0000-F00...	OP	7
01010010	Caller Instance	08500001-0000-0000-F00...	OP	11
01010020	Provider Instance	08500001-0000-0000-F00...	OP	11
01010021	Provider_PlcTask	08500004-0000-0000-F00...	OP	4
71010000	MAIN.fbCalc	00000000-0000-0000-000...	OP	4
02000000	RTime	02000000-0000-0000-F00...	OP	47
02010020	PlcTask	01020001-0000-0000-F00...	OP	5
01000000	Router	01000000-0000-0000-F00...	OP	16
01000010	TComServerTask	01000010-0000-0000-F00...	OP	3
01000070	TcEventLogger	01000070-0000-0000-F00...	OP	2

Expression	Type	Value	Prepared value	Add
m	UDINT	23735		
fbCalc	FB_Calculation			
m_objName	STRING	'MAIN.fbCalc'		
m_classId	GUID	{00000000-0000-0000-0000...		
objID	OTCID	71010000		
hrComObjInit	HRESULT	00000000		
hrComObjExit	HRESULT	00000000		
hrComObjReinit	HRESULT	00000000		

⇨ In the online view of the PLC application “Caller” the Proxy function block has been allocated the same object ID via the process image. The interface pointer has a valid value and the methods are executed.



Expression	Type	Value	Prepared value
fbCalc	FB_CalculationWrapper		
ip	I_Calculation	16#FFFFFA800AF99E00	
nObjId	OTCID	71010000	
iid	IID	{4D0C9030-560A-45F3-897...	
hrCalc	HRESULT	00000000	

```
IF fbCalc.ip 16#FFFFFA800AF99E00 = 0 THEN
    hrCalc 0 := fbCalc.QueryInterface();
END_IF
IF fbCalc.ip 16#FFFFFA800AF99E00 <> 0 THEN
    hrCalc 0 := fbCalc.ip.Addition(a 10 ,b 7 ,nSum 17 );
    hrCalc 0 := fbCalc.ip.Subtraction(a 10 ,b 7 ,nDiff 3 );
END_IF RETURN
```

Expression	Type	Value	Prepared value
m	UDINT	38186	
fbCalc	FB_Calculation		
m_objName	STRING	'MAIN.fbCalc'	
m_classId	GUID	{00000000-0000-0000-...	
objID	OTCID	71010000	
hrComObjInit	HRESULT	00000000	
hrComObjExit	HRESULT	00000000	
hrComObjReinit	HRESULT	00000000	

```
m 38186 := m 38186 + 1;RETURN
```

7.2 TcCOM_Sample02_PlcToCpp

This example describes a TcCOM communication between PLC and C++. In this connection a PLC application uses functionalities of an existing instance of a TwinCAT C++ class. In this way own algorithms written in C++ can be used easily in the PLC.
Although in the event of the use of an existing TwinCAT C++ driver the TwinCAT C++ license is required on the destination system, a C++ development environment is not necessary on the destination system or on the development computer.

An already built C++ driver provides one or more classes whose interfaces are deposited in the TMC description file and thus are known in the PLC.

The procedure is explained in the following sub-chapters:

1. Instantiating a TwinCAT++ class as a TwinCAT TcCOM Object [▶ 46]

2. Creating an FB in the PLC, which as a simple wrapper offers the functionality of the C++ object [▶ 47]

3. Execution of the sample project [▶ 49]

Downloading the sample: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/2343048971/.zip

System requirements

TwinCAT version	Hardware	Libraries to be Integrated
TwinCAT 3.1, Build 4020	x86, x64	Tc3_Module

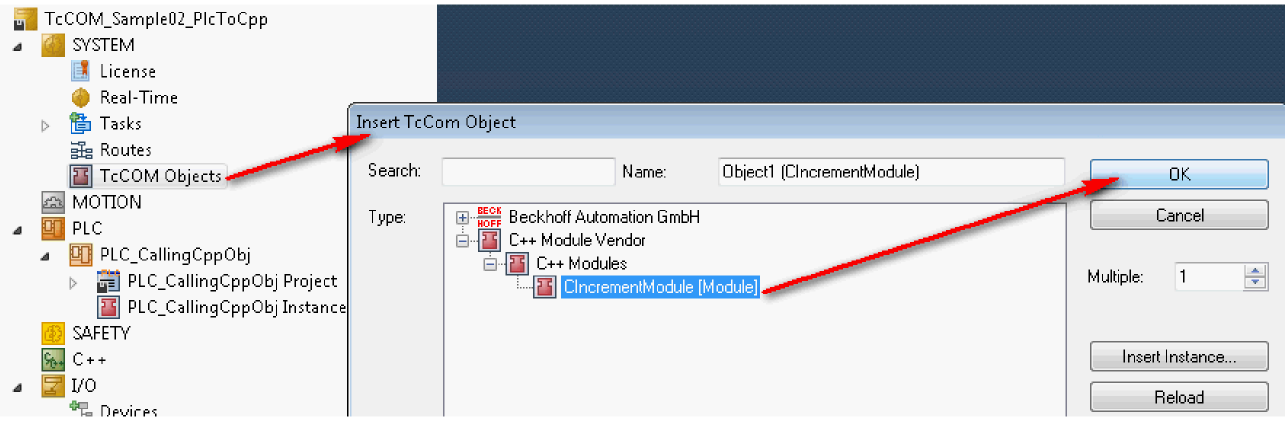
7.2.1 Instantiating a TwinCAT++ class as a TwinCAT TcCOM Object

The TwinCAT C++ driver must be available on the target system. TwinCAT offers a deployment for this purpose, so that the components only have to be stored properly on the development computer.

The existing TwinCAT C++ driver as well as its TMC description file(s) are available as a driver archive. This archive (IncrementerCpp.zip) is unpacked in the following folder:
C:\TwinCAT\3.1\CustomConfig\Modules\IncrementerCpp\

The TwinCAT Deployment copies the file(s) later in the following folder upon the activation of a configuration in the target system:
C:\TwinCAT\3.1\Driver\AutoInstall\

1. Open a TwinCAT project or create a new project.
2. Add an instance of Class CIncrementModule in the solution under the node **TcCOM Objects**.

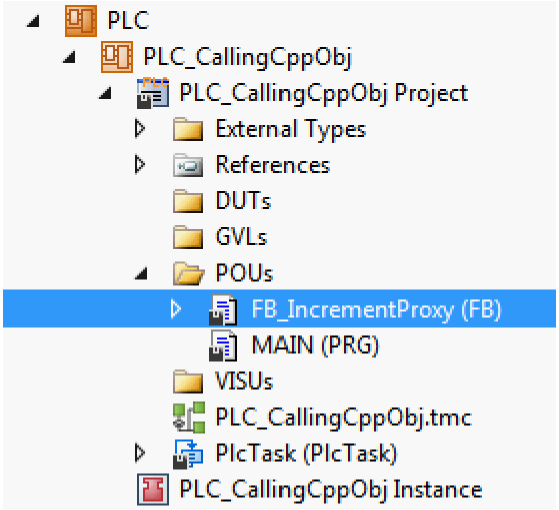


Creation of the C++ driver

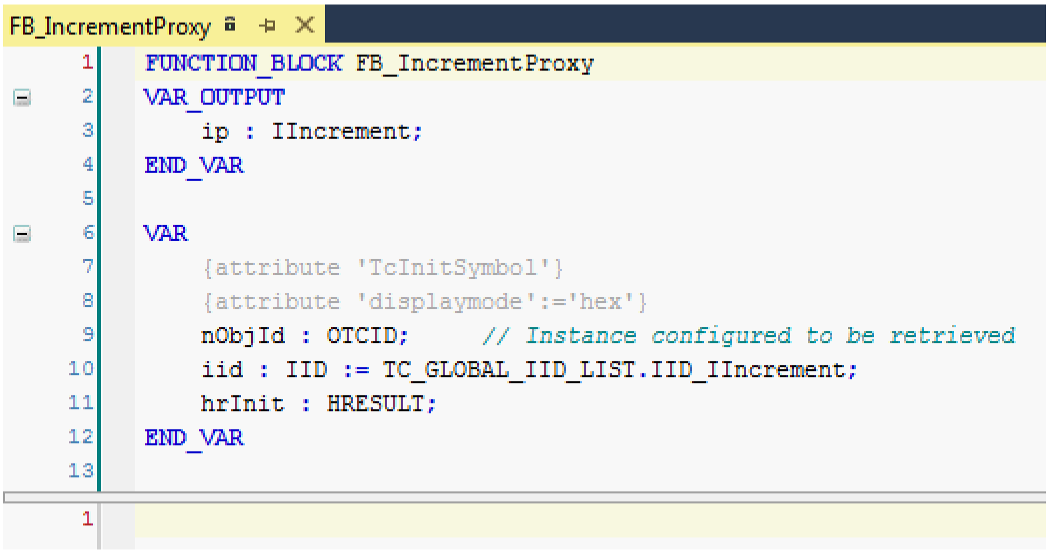
In the documentation for TwinCAT C++ there is a detailed explanation on how C++ drivers for TwinCAT are created.
To create the above-mentioned driver archive, **Publish TwinCAT Modules** is selected from the C++ project context as the last step in the creation of a driver.

7.2.2 Creating an FB in the PLC that, as a simple proxy, offers the functionality of the C++ object

1. Create a PLC and append a new function block there.
   This Proxy function block should provide the functionality that was programmed in C++. It is able to do this via an interface pointer that was defined from the C++ class and is known in the PLC due to the TMC description file.

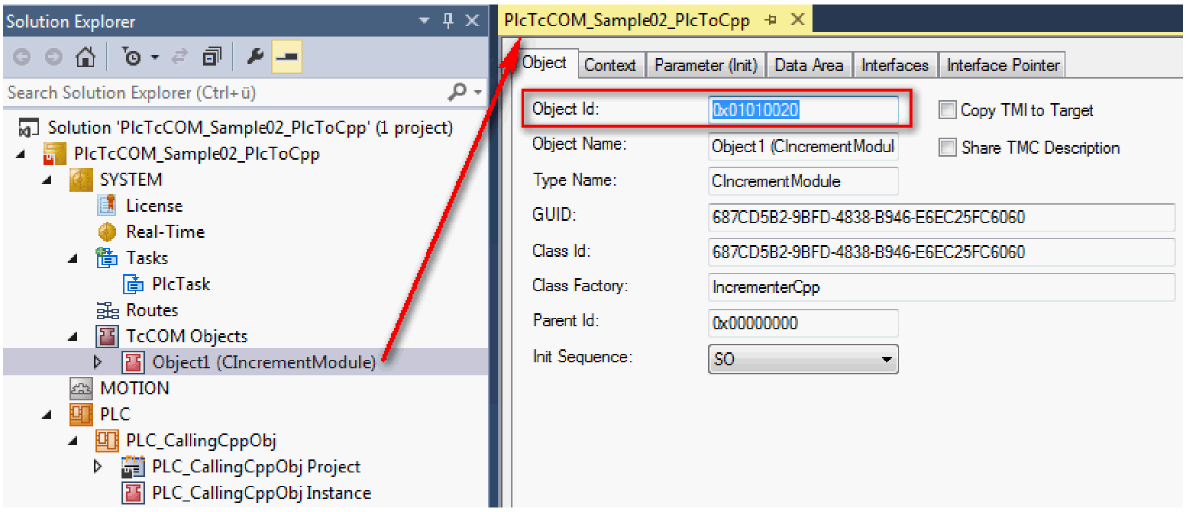


2. In the declaration part of the function block declare as an output an interface pointer to the interface which later provides the functionality outward.
3. Create the object ID and the interface ID as local member variables.
   While the interface ID is already available via a global list, the object ID is allocated via the TwinCAT symbol initialization. The TcInitSymbol attribute ensures that the variable appears in a list for external symbol initialization. The object ID of the created C++ object should be allocated.



```
FUNCTION_BLOCK FB_IncrementProxy
VAR_OUTPUT
    ip : IIncrement;
END_VAR

VAR
    {attribute 'TcInitSymbol'}
    {attribute 'displaymode':='hex'}
    nObjId : OTCID;      // Instance configured to be retrieved
    iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
    hrInit : HRESULT;
END_VAR

```

⇨ The object ID is displayed upon selection of the object under the **TcCOM Objects** node. Provided the TcInitSymbol attribute was used, the list of symbol initializations is located in the node of the PLC instance in the **Symbol Initialization** tab.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'PlcTcCOM_Sample02_PlcToCpp' (1 project)
PlcTcCOM_Sample02_PlcToCpp
SYSTEM
License
Real-Time
Tasks
PlcTask
Routes
TcCOM Objects
Object1 (CIncrementModule)
MOTION
PLC
PLC_CallingCppObj
PLC_CallingCppObj Project
PLC_CallingCppObj Instance
PlcTcCOM_Sample02_PlcToCpp
Object
Context
Parameter (Init)
Data Area
Interfaces
Interface Pointer

Object Id: 0x01010020

Object Name: Object1 (CIncrementModul

Type Name: CIncrementModule

GUID: 687CD5B2-9BFD-4838-B946-E6EC25FC6060

Class Id: 687CD5B2-9BFD-4838-B946-E6EC25FC6060

Class Factory: IncrementerCpp

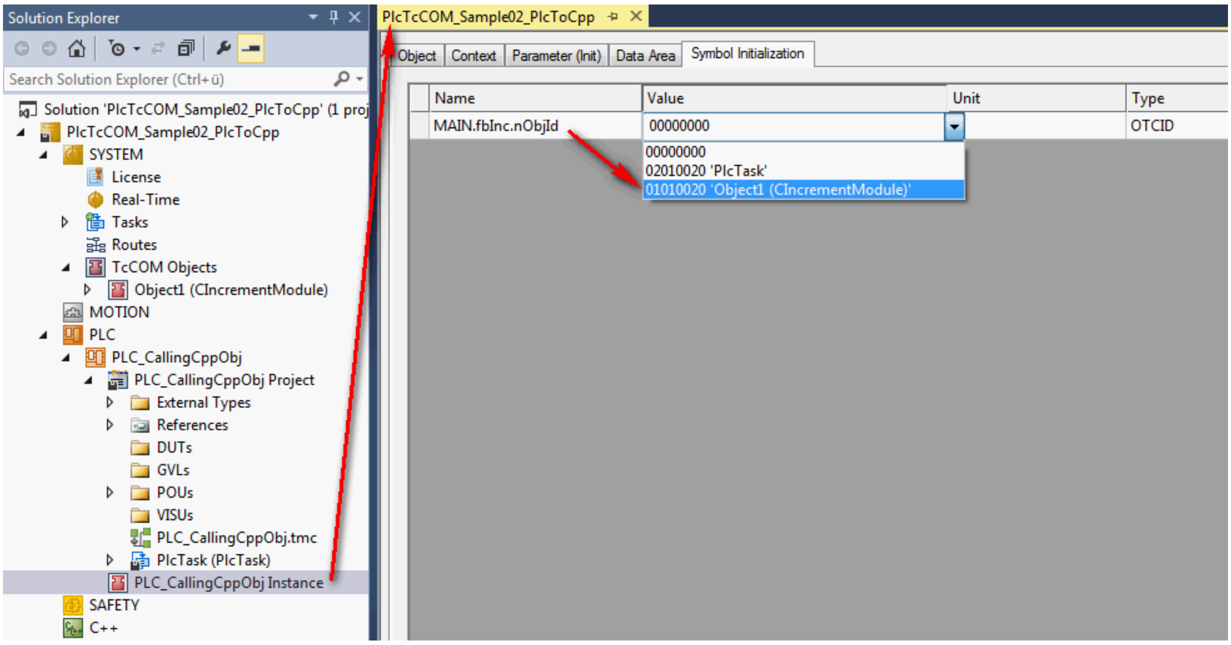
Parent Id: 0x00000000

Init Sequence: SO

☐ Copy TMI to Target

☐ Share TMC Description

4. Here, assign an existing object ID to the symbol name of the variable by drop-down. This value is assigned when the PLC is downloaded so it can be defined prior to the PLC run-time. New symbol initializations or changes are accordingly entered with a new download of the PLC.



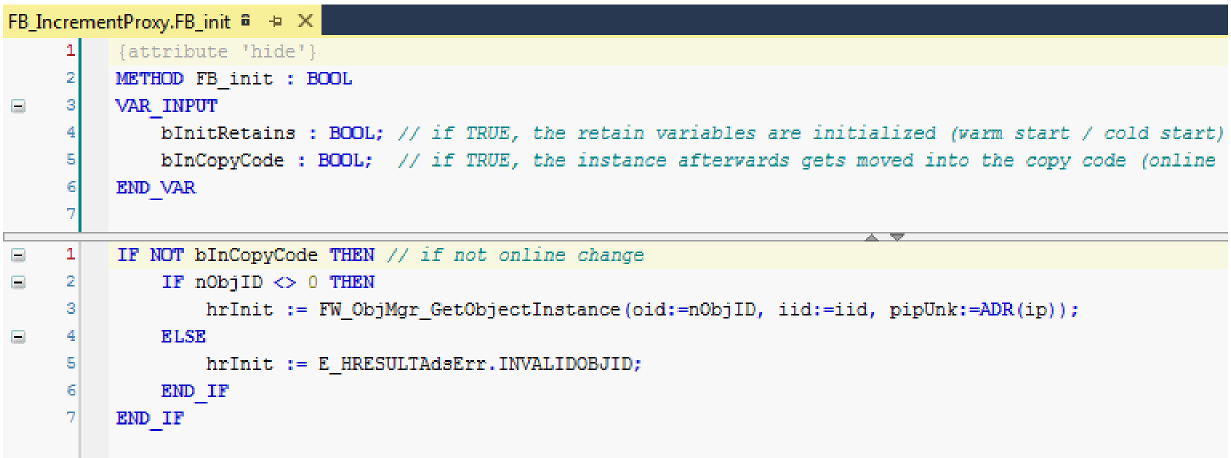
Name	Value	Unit	Type
MAIN.fbInc.nObjId	00000000		OTCID

00000000
02010020 'PlcTask'
01010020 'Object1 (CIncrementModule)'

As an alternative, the passing of the object ID could also be implemented by means of process image linking as implemented in the first sample (TcCOM_Sample01_PlcToPlc [▸ 36]).

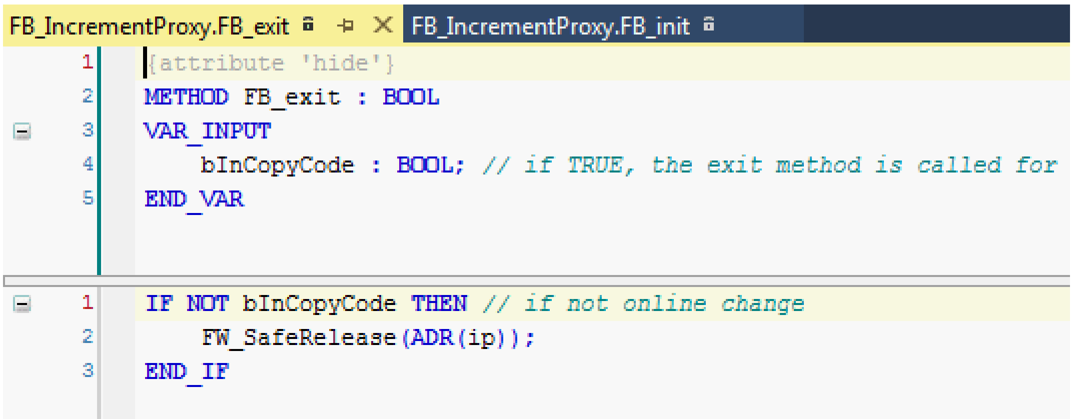
5. Implement the PLC Proxy function block.
First the FB_init constructor method is added to the function block. For the case that it is no longer an OnlineChange but rather the initialization of the function block, the interface pointer to the specified interface of the specified TcCOM object is obtained with the help of the function

FW_ObjMgr_GetObjectInstance() [▶ 25]. In this connection the object itself increments a reference counter.



```
{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
```

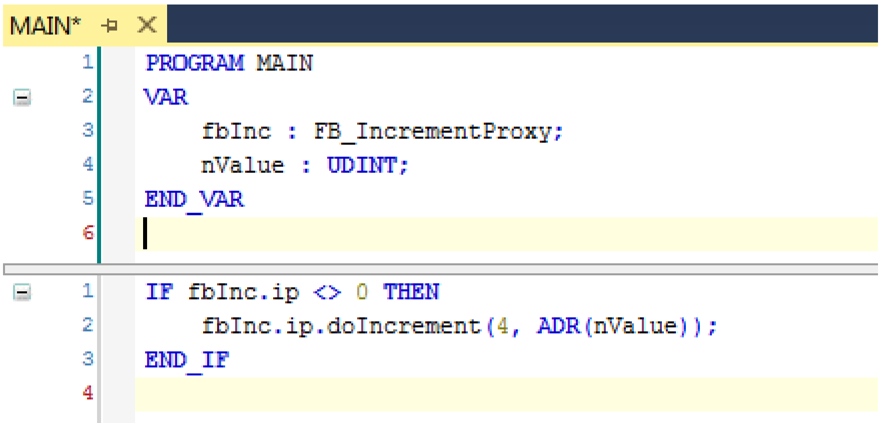
6. It is imperative to release the used reference again. To this end call the FW_SafeRelease() function [▶ 26] in the FB_exit destructor of the function block.



```
{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
```

⇨ This completes the implementation of the Proxy function block.

7. Declare an instance of the Proxy function block to call the methods provided via the interface in the application.
The calls themselves take all place over the interface pointer defined as output of the function block. As is typical for pointers a prior null check must be made. Then the methods can be called directly, also via Intellisense.



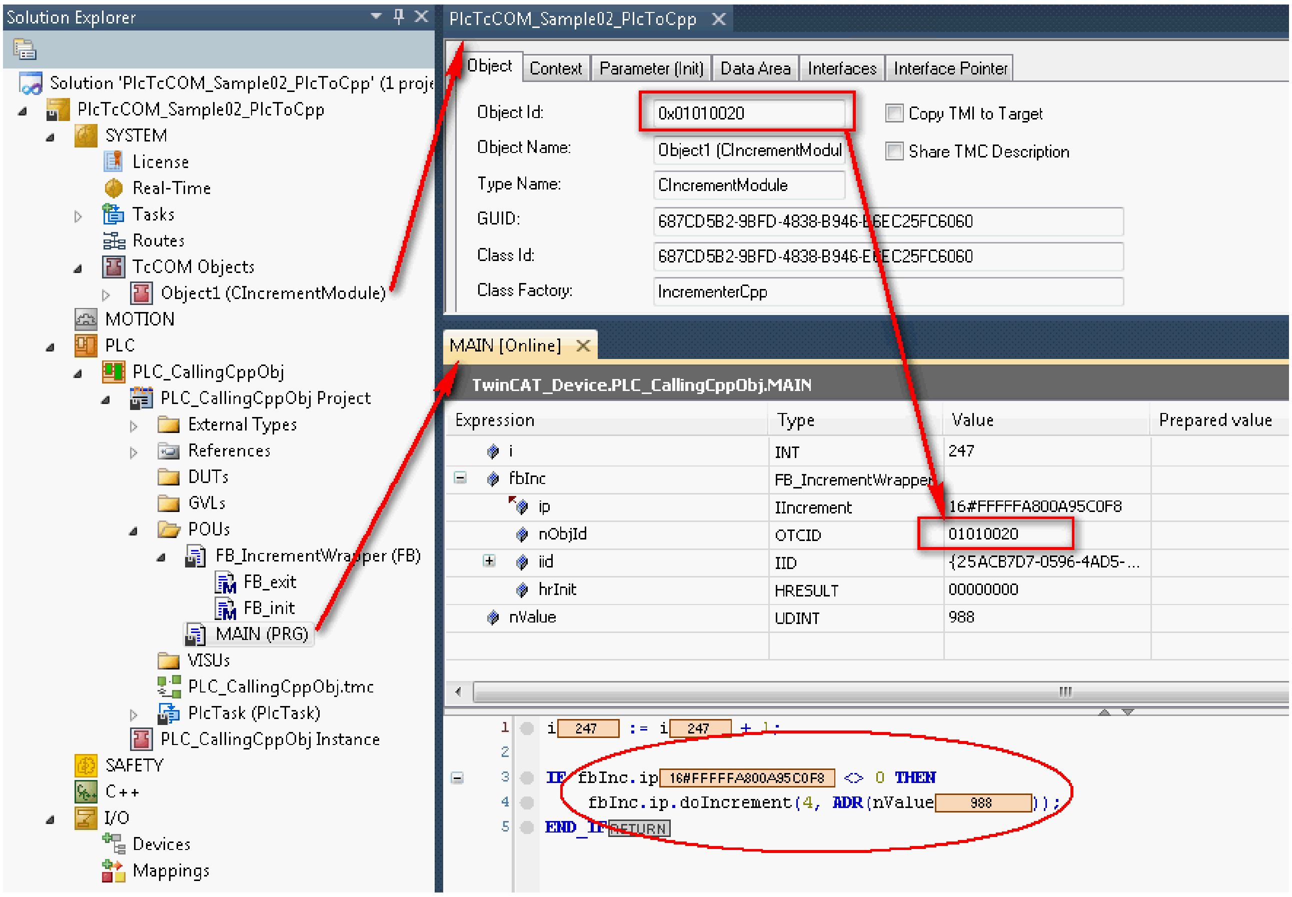
```
PROGRAM MAIN
VAR
    fbInc : FB_IncrementProxy;
    nValue : UDINT;
END_VAR


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

```

⇨ The sample is ready for testing.

7.2.3 Execution of the sample project

1. Select the destination system and compile the project.
2. Enable the TwinCAT configuration and execute a log-in as well as starting the PLC.

⇨ In the online view of the PLC application the assigned object ID of the C++ object in the PLC Proxy function block can be seen. The interface pointer has a valid value and the method will be executed.



```
Solution Explorer
Solution 'PlcTcCOM_Sample02_PlcToCpp' (1 proje
  PlcTcCOM_Sample02_PlcToCpp
    SYSTEM
      License
      Real-Time
      Tasks
      Routes
      TcCOM Objects
        Object1 (CIncrementModule)
    MOTION
    PLC
      PLC_CallingCppObj
        PLC_CallingCppObj Project
          External Types
          References
          DUTs
          GVLs
          POUs
            FB_IncrementWrapper (FB)
              FB_exit
              FB_init
            MAIN (PRG)
          VISUs
          PLC_CallingCppObj.tmc
          PlcTask (PlcTask)
        PLC_CallingCppObj Instance
    SAFETY
    C++
    I/O
      Devices
      Mappings
```

PlcTcCOM_Sample02_PlcToCpp

Object | Context | Parameter (Init) | Data Area | Interfaces | Interface Pointer

Object Id:	0x01010020
Object Name:	Object1 (CIncrementModul
Type Name:	CIncrementModule
GUID:	687CD5B2-9BFD-4838-B946-EEC25FC6060
Class Id:	687CD5B2-9BFD-4838-B946-EEC25FC6060
Class Factory:	IncrementerCpp

□ Copy TMI to Target
□ Share TMC Description

MAIN [Online]

TwinCAT_Device.PLC_CallingCppObj.MAIN

Expression	Type	Value	Prepared value
i	INT	247	
fbInc	FB_IncrementWrapper		
ip	IIncrement	16#FFFFFA800A95C0F8	
nObjId	OTCID	01010020	
iid	IID	{25ACB7D7-0596-4AD5-...	
hrInit	HRESULT	00000000	
nValue	UDINT	988	

```
i 247 := i 247 + 1;

IF fbInc.ip 16#FFFFFA800A95C0F8 <> 0 THEN
    fbInc.ip.doIncrement(4, ADR(nValue 988 ));
END_IF RETURN
```

7.3 TcCOM_Sample03_PlcCreatesCpp

Just like Sample02, this sample describes a TcCOM communication between PLC and C++. To this end a PLC application uses functionalities of a TwinCAT C++ class. The required instances of this C++ class will be created by the PLC itself in this sample. In this way own algorithms written in C++ can be used easily in the PLC.
Although in the event of the use of an existing TwinCAT C++ driver the TwinCAT C++ license is required on the destination system, a C++ development environment is not necessary on the destination system or on the development computer.

An already built C++ driver provides one or more classes whose interfaces are deposited in the TMC description file and thus are known in the PLC.

The procedure is explained in the following sub-chapters:

1. Provision of a TwinCAT C++ driver and its classes [▶ 51]

2. Creating an FB in the PLC that creates the C++ object and offers its functionality [▶ 52]

3. Execution of the sample project [▶ 54]

Downloading the sample: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/2343051531/.zip

System requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4020	x86, x64	Tc3_Module

7.3.1 Provision of a TwinCAT C++ driver and its classes

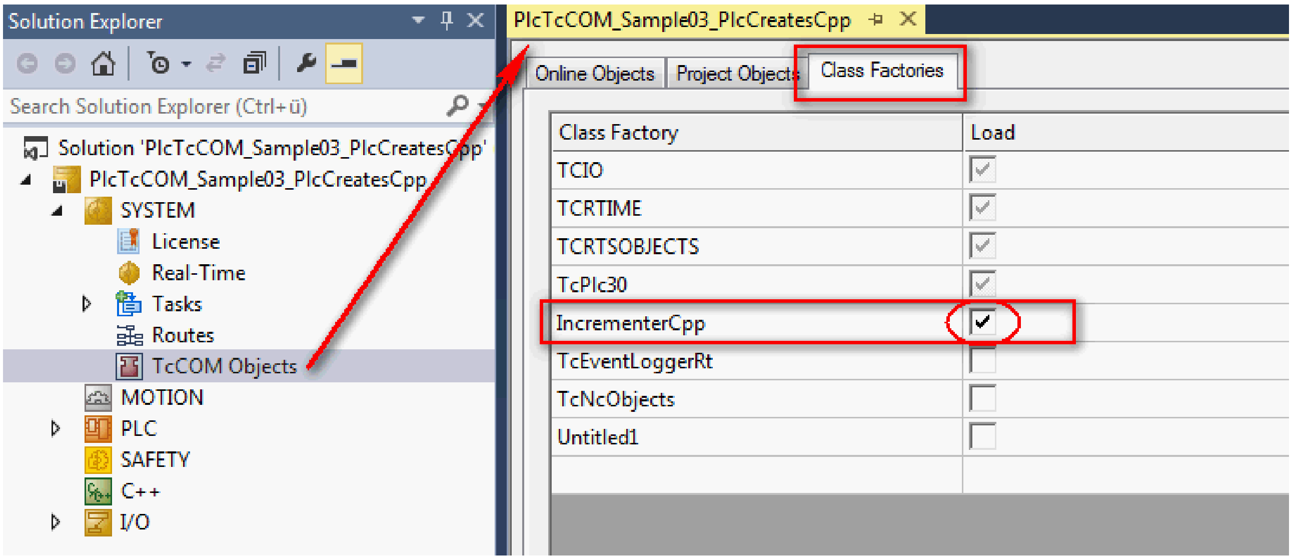
The TwinCAT C++ driver must be available on the target system. TwinCAT offers a deployment for this purpose, so that the components only have to be stored properly on the development computer.

The existing TwinCAT C++ driver as well as its TMC description file(s) are available as a driver archive. This archive (IncrementerCpp.zip) is unpacked in the following folder:
C:\TwinCAT\3.1\CustomConfig\Modules\IncrementerCpp\

The TwinCAT Deployment copies the file(s) later in the following folder upon the activation of a configuration in the target system:
C:\TwinCAT\3.1\Driver\AutoInstall\

1. Open a TwinCAT project or create a new project.
2. Select the required C++ driver in the solution under the **TcCOM Objects** node in the **Class Factories** tab.

⇨ This ensures that the driver is loaded on the target system when TwinCAT starts up. In addition this selection provides for the described deployment.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'PlcTcCOM_Sample03_PlcCreatesCpp'
PlcTcCOM_Sample03_PlcCreatesCpp
SYSTEM
License
Real-Time
Tasks
Routes
TcCOM Objects
MOTION
PLC
SAFETY
C++
I/O
PlcTcCOM_Sample03_PlcCreatesCpp
Online Objects
Project Objects
Class Factories

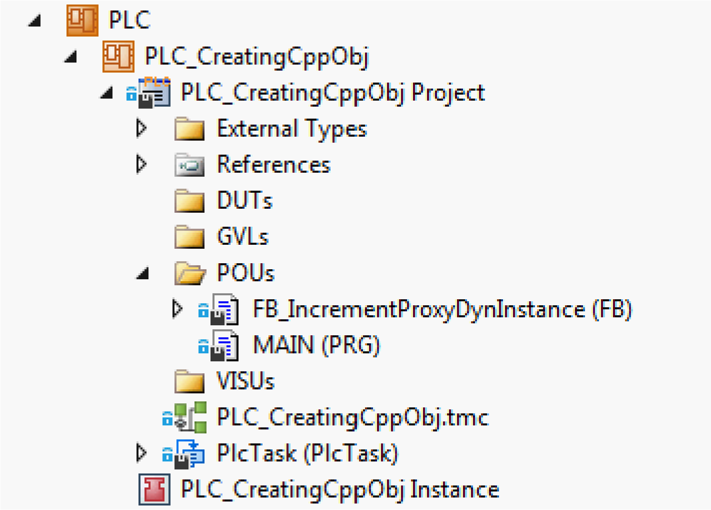
Class Factory	Load
TCIO	☑
TCRTIME	☑
TCRTSOBJECTS	☑
TcPlc30	☑
IncrementerCpp	☑
TcEventLoggerRt	☐
TcNcObjects	☐
Untitled1	☐

Creation of the C++ driver

In the documentation for TwinCAT C++ there is a detailed explanation on how C++ drivers for TwinCAT are created.
For Sample03 it is important to note that TwinCAT C++ drivers whose classes are supposed to be dynamically instantiated must be defined as “TwinCAT Module Class for RT Context”. The C++ Wizard offers a special template for this purpose.
In addition this sample uses a TwinCAT C++ class which manages without TcCOM initialization data and without TcCOM parameters.

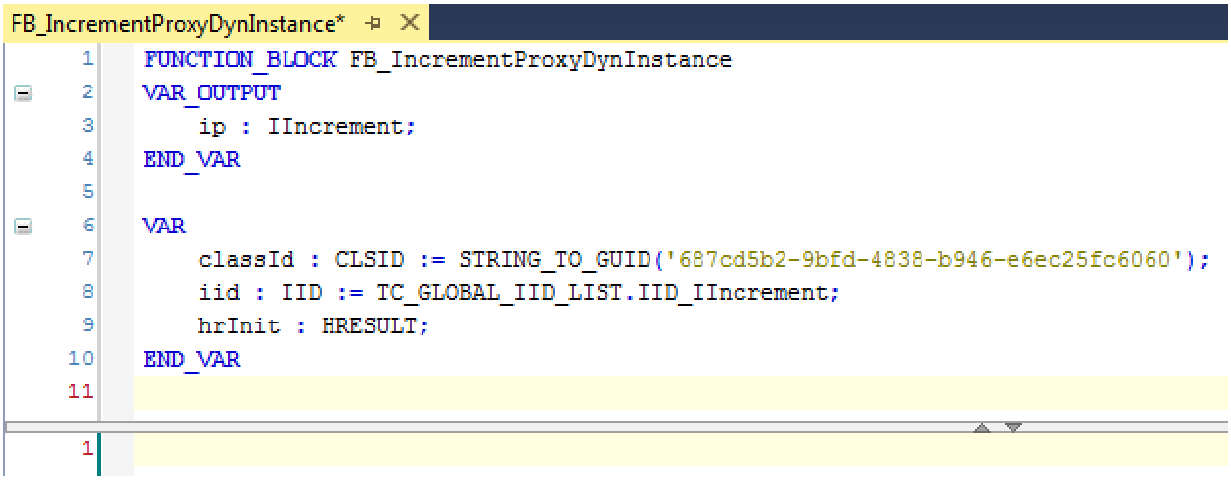
7.3.2 Creating an FB in the PLC that creates the C++ object and offers its functionality

1. Create a PLC and append a new function block there.
   This Proxy function block should provide the functionality that was programmed in C++. It manages this via an interface pointer that was defined by C++ and is known in the PLC due to the TMC description file.



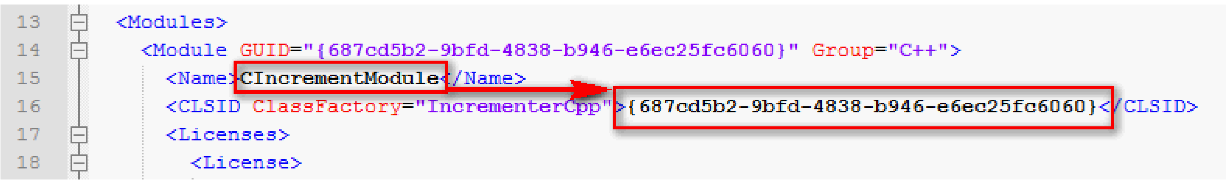
```
PLC
  PLC_CreatingCppObj
    PLC_CreatingCppObj Project
      External Types
      References
      DUTs
      GVLs
      POUs
        FB_IncrementProxyDynInstance (FB)
        MAIN (PRG)
      VISUs
      PLC_CreatingCppObj.tmc
      PlcTask (PlcTask)
    PLC_CreatingCppObj Instance
```

2. In the declaration part of the function block declare as an output an interface pointer to the interface (IIncrement) which later provides the functionality outward.



```
FUNCTION_BLOCK FB_IncrementProxyDynInstance
VAR_OUTPUT
    ip : IIncrement;
END_VAR

VAR
    classId : CLSID := STRING_TO_GUID('687cd5b2-9bfd-4838-b946-e6ec25fc6060');
    iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
    hrInit : HRESULT;
END_VAR
```

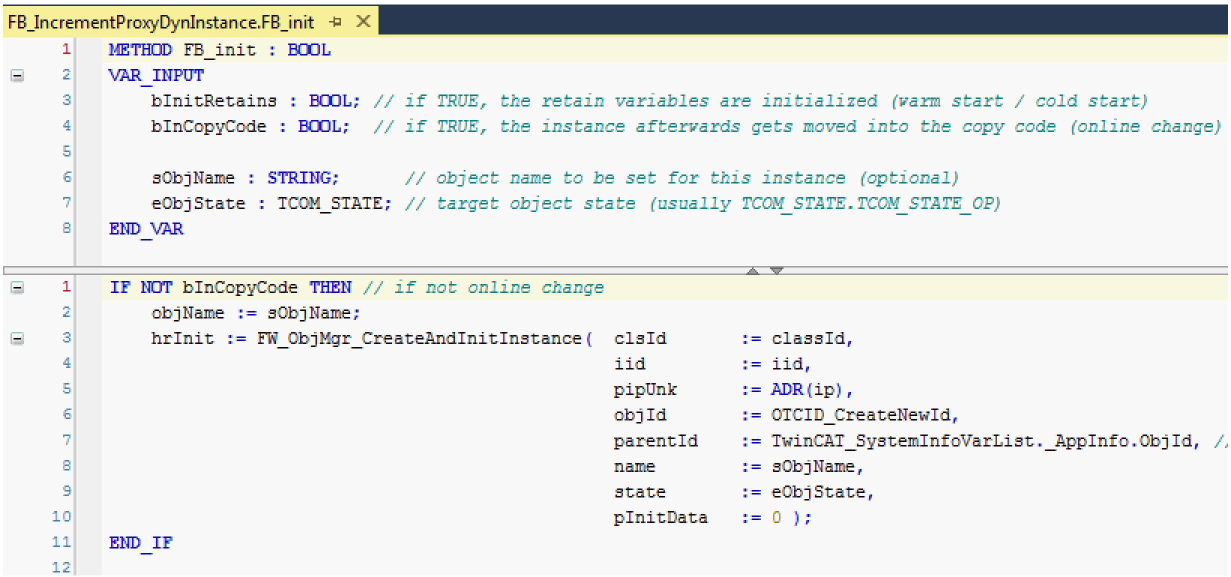
3. Create class ID and the interface ID as member variables.
   While the interface ID is already available via a global list, the class IDs, provided they are not yet supposed to be known, are determined by other means. When you open the TMC description file of the associated C++ driver you will find the corresponding GUID there.



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

4. Add the FB_init constructor method to the PLC Proxy function block.
   For the case, that it is not an online change but rather the initialization of the function block, a new TcCOM object (Class instance of the specified class) is created and the interface pointer to the specified interface is obtained. In the process the used FW_ObjMgr_CreateAndInitInstance() function [▶ 23] is also given the name and the destination state of the TcCOM object. These two parameters are declared here as input parameters of the FB_init method, whereby they are to be specified in the instantiation of the Proxy function block. The TwinCAT C++ class to be instantiated manages without TcCOM initialization

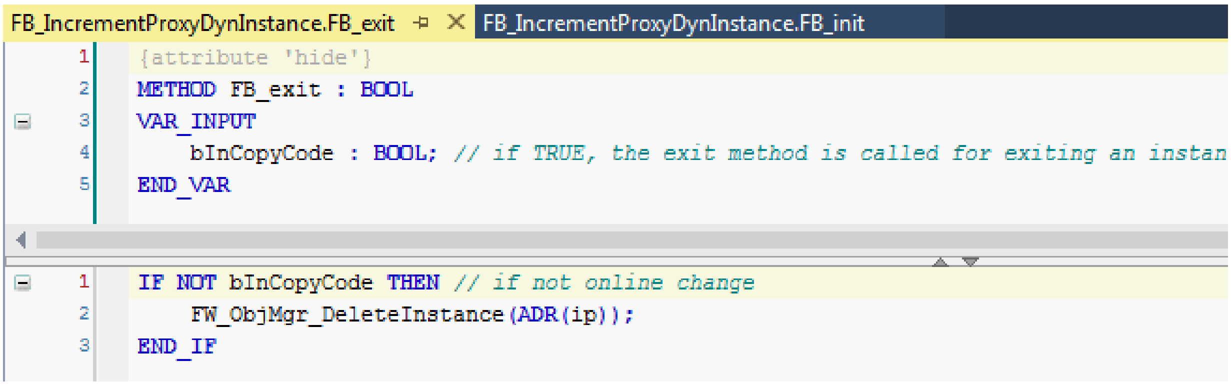
data and without TcCOM parameters.
In the case of this function call the object itself increments a reference counter.



```
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. It is imperative to release the used reference again and to delete the object, provided it is no longer being used. To this end call the FW_ObjMgr_DeleteInstance() [▶ 25] function in the FB_exit destructor of the function block.

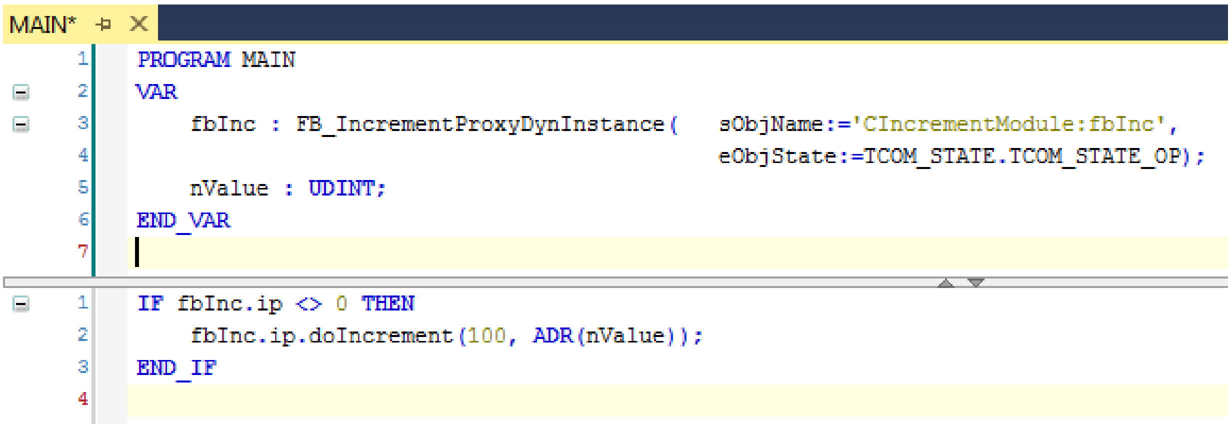

FB_IncrementProxyDynInstance.FB_exit
FB_IncrementProxyDynInstance.FB_init

```
{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
```

⇨ This completes the implementation of the Proxy function block.

6. Declare an instance of the Proxy function block to call the methods provided via the interface in the application. The calls themselves take all place over the interface pointer defined as output of the function block. As is typical for pointers a prior null check must be made. Then the methods can be called directly, also via Intellisense.



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

```

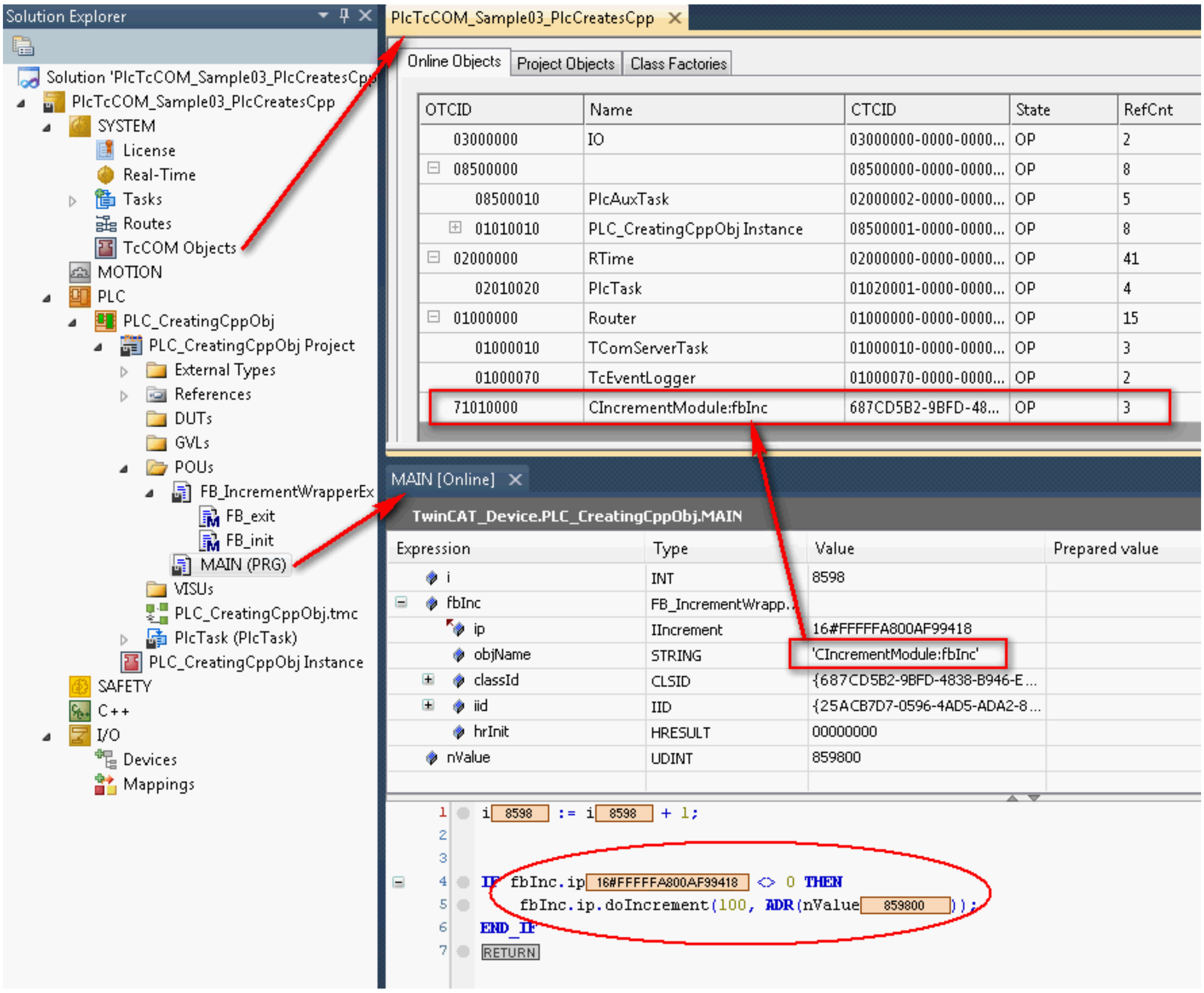
```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(100, ADR(nValue));
END_IF

```

⇨ The sample is ready for testing.

7.3.3 Execution of the sample project

1. Select the target system and compile the project.
2. Enable the TwinCAT configuration and execute a log-in as well as starting the PLC.

⇨ In the online view of the PLC application the desired TcCOM object name in the PLC Proxy function block can be seen. The project node **TcCOM Objects** keeps the generated object with the generated ID and the desired name in his list. The interface pointer has a valid value and the method will be executed.



OTCID	Name	CTCID	State	RefCnt
03000000	IO	03000000-0000-0000...	OP	2
08500000		08500000-0000-0000...	OP	8
08500010	PlcAuxTask	02000002-0000-0000...	OP	5
01010010	PLC_CreatingCppObj Instance	08500001-0000-0000...	OP	8
02000000	RTime	02000000-0000-0000...	OP	41
02010020	PlcTask	01020001-0000-0000...	OP	4
01000000	Router	01000000-0000-0000...	OP	15
01000010	TComServerTask	01000010-0000-0000...	OP	3
01000070	TcEventLogger	01000070-0000-0000...	OP	2
71010000	CIncrementModule:fbInc	687CD5B2-9BFD-48...	OP	3

Expression	Type	Value	Prepared value
i	INT	8598	
fbInc	FB_IncrementWrapp...		
ip	IIncrement	16#FFFFFA800AF99418	
objName	STRING	'CIncrementModule:fbInc'	
classId	CLSID	{687CD5B2-9BFD-4838-B946-E...	
iid	IID	{25ACB7D7-0596-4AD5-ADA2-8...	
hrInit	HRESULT	00000000	
nValue	UDINT	859800	

```
i 8598 := i 8598 + 1;


IF fbInc.ip 16#FFFFFA800AF99418 <> 0 THEN
   fbInc.ip.doIncrement(100, ADR(nValue 859800 ));
END_IF
RETURN
```

7.4 TcCOM_Sample13_CppToPlc

Description

This sample provides for communication from a C++ module to a function block of a PLC by means of method call. To this end a TcCOM interface is defined that is offered by the PLC and used by the C++ module.

The PLC page as a provider in the process corresponds to the corresponding project of the TcCOM Sample 01 [▶ 36], where an PLC is considered after PLC communication. Here a Caller is now provided in C++, which uses the same interface.

You can find the explanation of the sample in the sub-chapter “Implementation of the sample”.

Downloading the sample: TcCOM_Sample13_CppToPlc.zip

System requirements

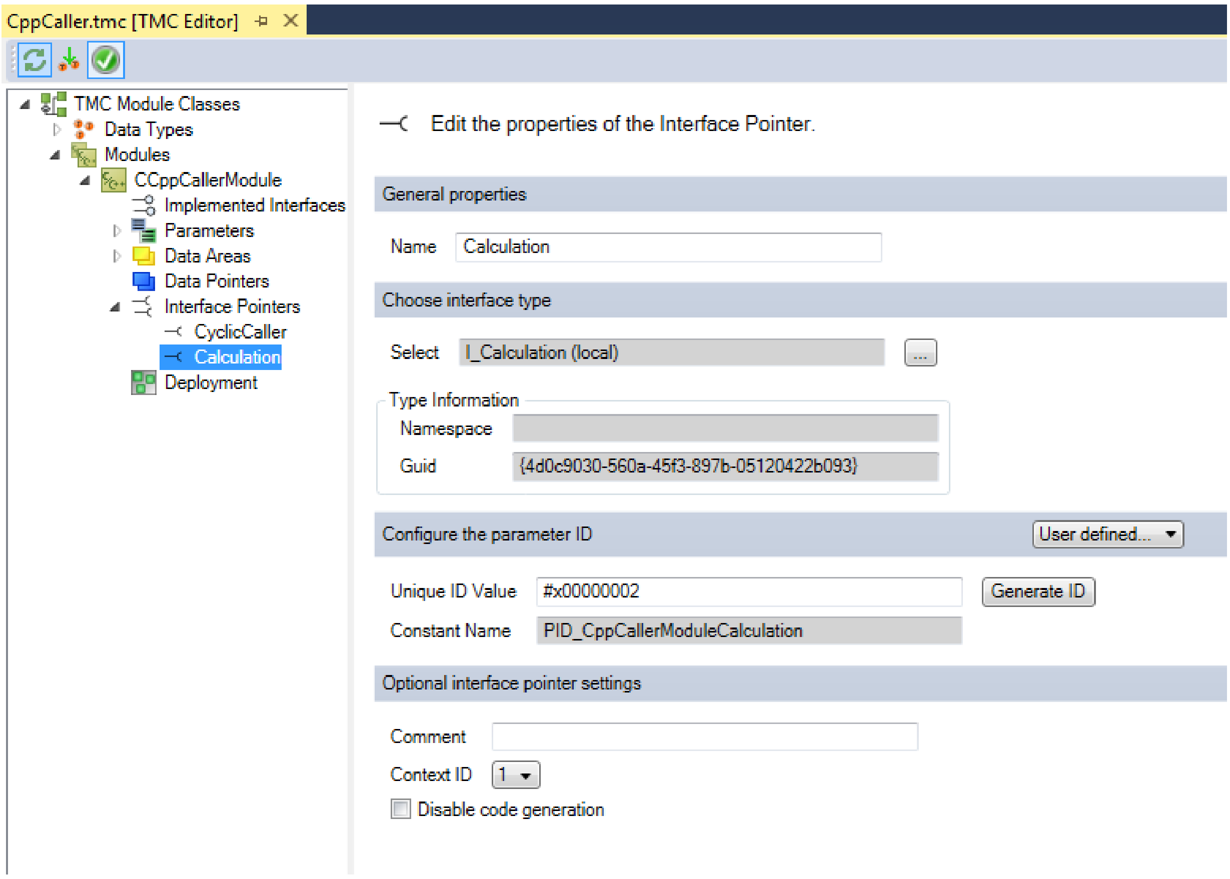
TwinCAT version	Hardware	PLC libraries to be linked
TwinCAT 3.1, Build 4020	x86, x64	Tc3_Module

7.4.1 Implementation of the sample

The PLC page adopted by TcCOM Sample 01 [▸ 36]. The function block registered there as TcCOM module offers the object ID allocated to it as an output variable.
It is the C++ module's task to make the offered interface of this function block accessible.

✓ A C++ project with a Cycle IO module is assumed.

1. In the TMC editor, create an interface pointer of the type I_Calculation with the name Calculationn). Later access occurs via this.



Edit the properties of the Interface Pointer.

General properties

Name: Calculation

Choose interface type

Select: I_Calculation (local) ...

Type Information

Namespace:

Guid: {4d0c9030-560a-45f3-897b-05120422b093}

Configure the parameter ID: User defined...

Unique ID Value: #x00000002 Generate ID

Constant Name: PID_CppCallerModuleCalculation

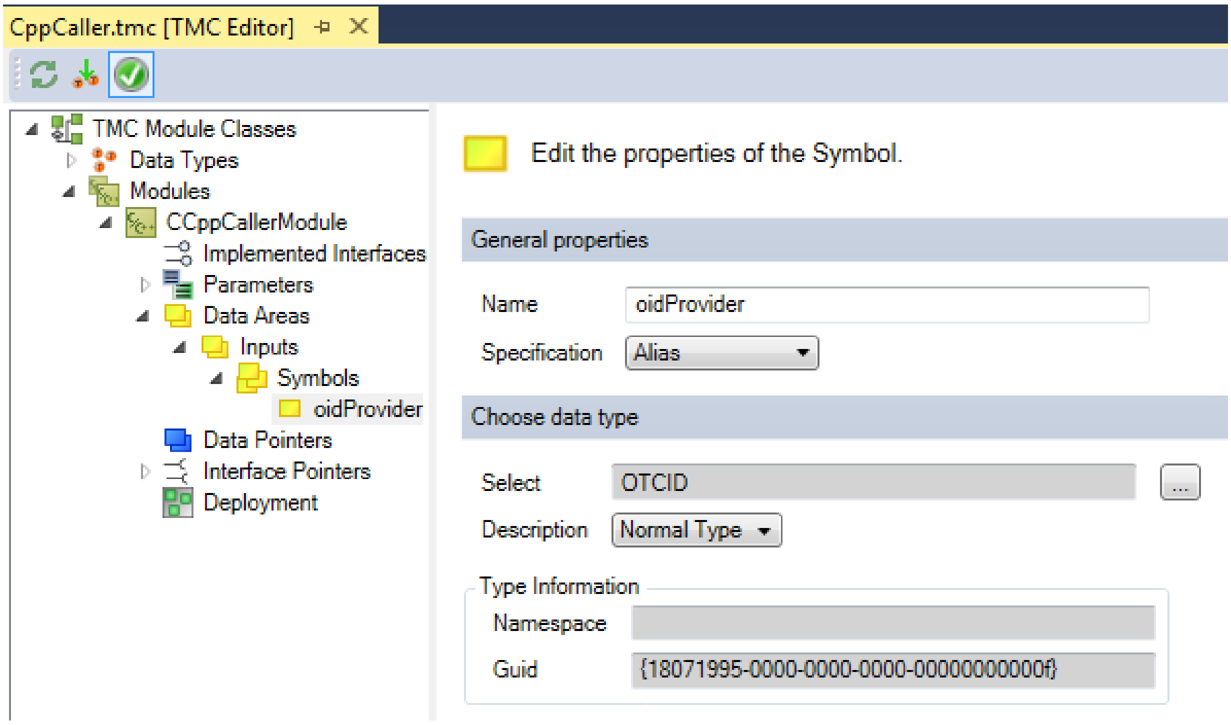
Optional interface pointer settings

Comment:

Context ID: 1

□ Disable code generation

2. The Data Area Inputs have already been created by the module wizard with the type Input-Destination. Here in the TMC editor you create an input of the type OTCID with the name oidProvider, via which the Object ID will be linked from the PLC later.



Name: oidProvider

Specification: Alias

Select: OTCID

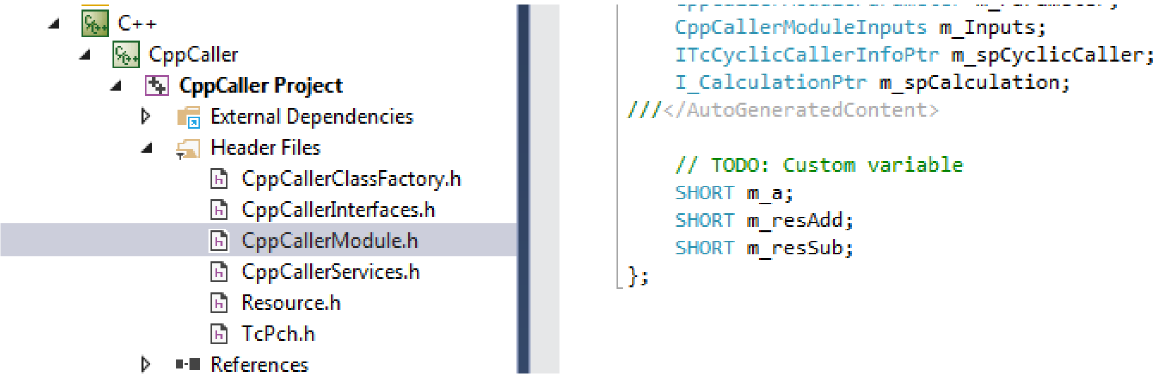
Description: Normal Type

Namespace:

Guid: {18071995-0000-0000-0000-00000000000f}

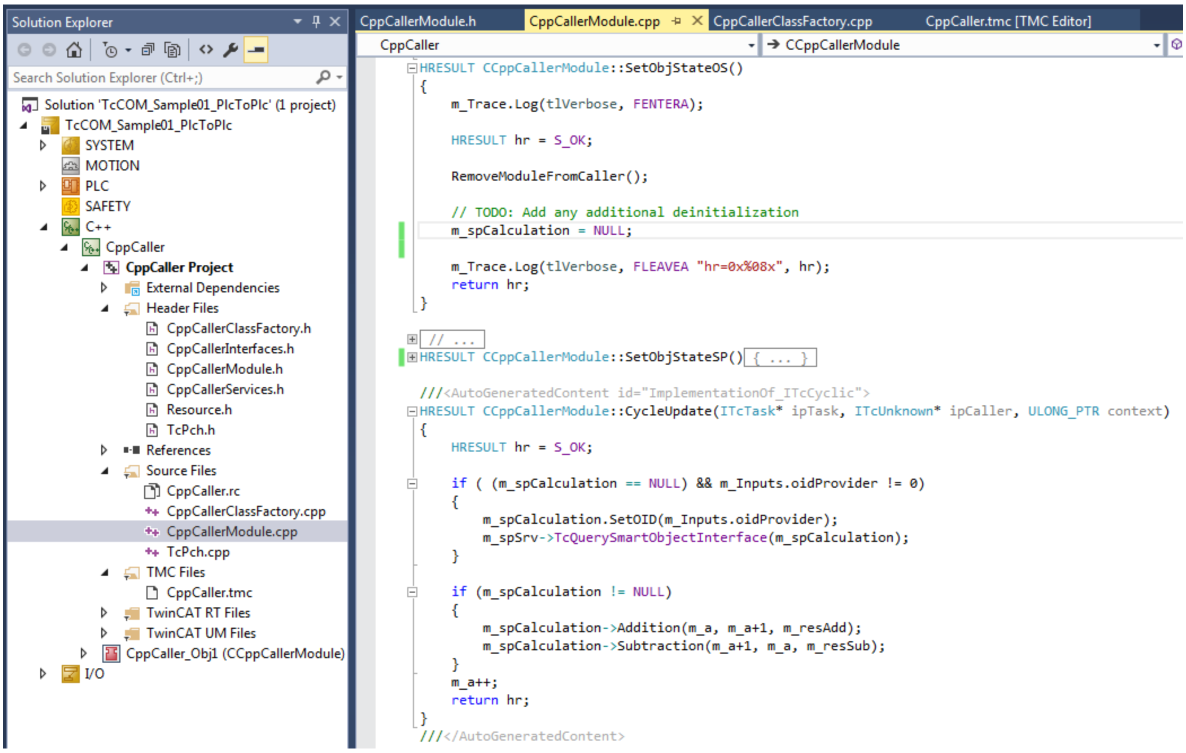
3. All other symbols are irrelevant for the sample and can be deleted.

⇨ The TMC-Code-Generator prepares the code accordingly.
In the header of the module some variables are created in order to carry out the methods calls later.



```
    CppCallerModuleInputs m_Inputs;
    ITcCyclicCallerInfoPtr m_spCyclicCaller;
    I_CalculationPtr m_spCalculation;
///</AutoGeneratedContent>

    // TODO: Custom variable
    SHORT m_a;
    SHORT m_resAdd;
    SHORT m_resSub;
};
```

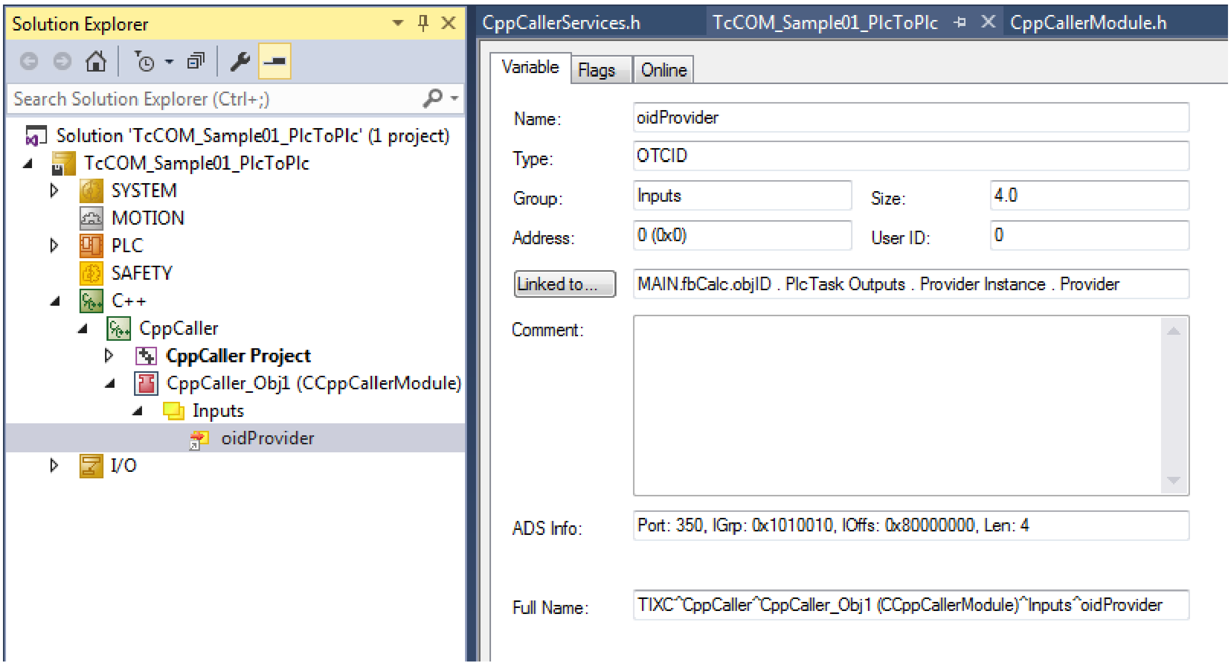
In the actual code of the module in CycleUpdate() the interface pointer is set using the object ID transmitted from the PLC. It is important that this happens in the CycleUpdate() and thus in real-time context, since the PLC must first provide the function block.
When this has taken place once, the methods can be called.



```
HRESULT CCppCallerModule::SetObjStateOS()
{
    m_Trace.Log(tlVerbose, FENTERA);

    HRESULT hr = S_OK;

    RemoveModuleFromCaller();

    // TODO: Add any additional deinitialization
    m_spCalculation = NULL;

    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}

// ...
HRESULT CCppCallerModule::SetObjStateSP() { ... }

///<AutoGeneratedContent id="ImplementationOf_ITcCyclic">
HRESULT CCppCallerModule::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    if ( (m_spCalculation == NULL) && m_Inputs.oidProvider != 0)
    {
        m_spCalculation.SetOID(m_Inputs.oidProvider);
        m_spSrv->TcQuerySmartObjectInterface(m_spCalculation);
    }

    if (m_spCalculation != NULL)
    {
        m_spCalculation->Addition(m_a, m_a+1, m_resAdd);
        m_spCalculation->Subtraction(m_a+1, m_a, m_resSub);
    }
    m_a++;
    return hr;
}
///</AutoGeneratedContent>
```

In addition, as can be seen above, the interface pointer is cleared when the program shuts down. This happens in the SetObjStateOS method.

4. Now build the C++ project.
5. Create an instance of the module.
6. Connect the input of the C++ module to the output of the PLC.



Name: oidProvider

Type: OTCID

Group: Inputs Size: 4.0

Address: 0 (0x0) User ID: 0

Linked to ... MAIN.fbCalc.objID . PlcTask Outputs . Provider Instance . Provider

Comment:

ADS Info: Port: 350, IGrp: 0x1010010, IOffs: 0x80000000, Len: 4

Full Name: TIXC^CppCaller^CppCaller_Obj1 (CCppCallerModule)^Inputs^oidProvider

⇨ The project can be started. When the PLC is running, the OID is made known through the mapping to the C++ instance. Once this has occurred, the method can be called.

8 Appendix

8.1 TcCOM Technology

The TwinCAT module concept is one of the core elements for the modularization of modern machines. This chapter describes the modular concept and working with modules.

8.1.1 The TwinCAT Component Object Model (TcCOM) concept

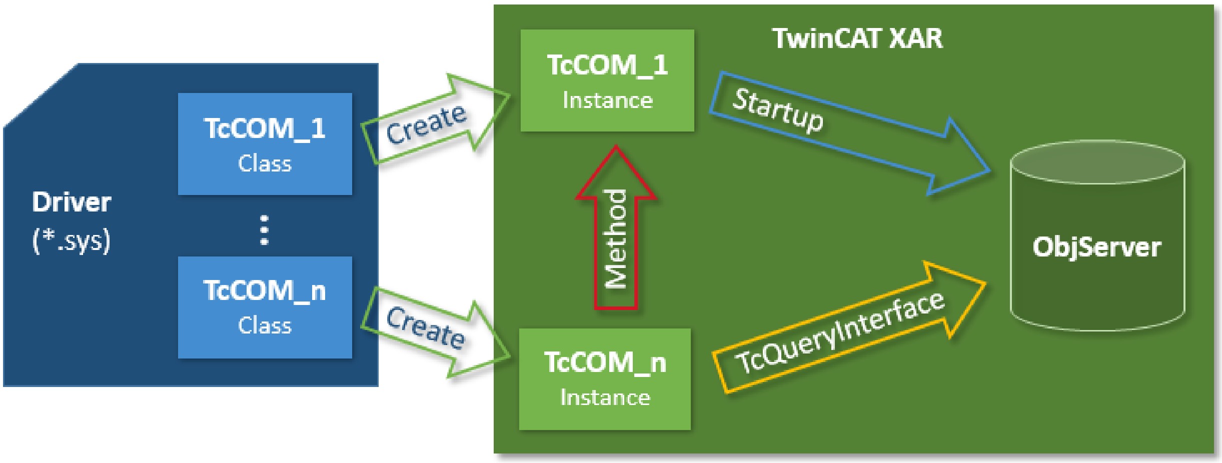
The TwinCAT Component Object Model defines the characteristics and the behavior of the modules. The model derived from the "Component Object Model" COM from Microsoft Windows describes the way in which various independently developed and compiled software components can co-operate with one another. To make that possible, a precisely defined mode of behavior and the observation of interfaces of the module must be defined, so that they can interact. Such an interface is also ideal for facilitating interaction between modules from different manufacturers, for example.

To some degree TcCOM is based on COM (Component Object Model of the Microsoft Windows world), although only a subset of COM is used. In comparison with COM, however, TcCOM contains additional definitions that go beyond COM, for example the state machine module.

Overview and application of TcCOM modules

This introductory overview is intended to make the individual topics easier to understand.

One or several TcCOM modules are consolidated in a driver. This driver is created by TwinCAT Engineering using the MSVC compiler. The modules and interfaces are described in a TMC (TwinCAT Module Class) file. The drivers and their TMC file can now be exchanged and combined between the engineering systems.



Instances of these modules are now created using the engineering facility. They are associated with a TMI file. The instances can be parameterized and linked with each other and with other modules to form the IO. A corresponding configuration is transferred to the target system, where it is executed.

Corresponding modules are started, which register with the TwinCAT ObjectServer. The TwinCAT XAR also provides the process images. Modules can query the TwinCAT ObjectServer for a reference to another object with regard to a particular interface. If such a reference is available, the interface methods can be called on the module instance.

The following sections substantiate the individual topics.

ID Management

Different types of ID are used for the interaction of the modules with each other and also within the modules. TcCOM uses GUIDs (128 bit) and 32 bit long integers.

TcCOM uses

- GUIDs for: ModulIDs, ClassIDs and InterfaceIDs.
- 32 bit long integers are used for: ParameterIDs, ObjectIDs, ContextIDs, CategoryID.

Interfaces

An important component of COM, and therefore of TcCOM too, is interfaces.

Interfaces define a set of methods that are combined in order to perform a certain task. An interface is referenced with a unique ID (InterfaceID), which must never be modified as long as the interface does not change. This ID enables modules to determine whether they can cooperate with other modules. At the same time the development process can take place independently, if the interfaces are clearly defined. Modifications of interfaces therefore lead to different IDs. The TcCOM concept is designed such that InterfaceIDs can superpose other (older) InterfaceIDs ("Hides" in the TMC description / TMC editor). In this way, both versions of the interface are available, while on the other hand it is always clear which is the latest InterfaceID. The same concept also exists for the data types.

TcCOM itself already defines a whole series of interfaces that are prescribed in some cases (e.g. ITComObject), but are optional in most. Many interfaces only make sense in certain application areas. Other interfaces are so general that they can often be re-used. Provision is made for customer-defined interfaces, so that two third-party modules can interact with each other, for example.

- All interfaces are derived from the basic interface ItcUnknown which, like the corresponding interface of COM, provides the basic services for querying other interfaces of the module (TcQueryInterface) and for controlling the lifetime of the module (TcAddRef and TcRelease).
- The ITComObject interface, which must be implemented by each module, contains methods for accessing the name, ObjectID, ObjectID of the parent, parameters and state machine of the module.

Several general interfaces are used by many modules:

- ITcCyclic is implemented by modules, which are called cyclically ("CycleUpdate"). The module can register via the ITcCyclicCaller interface of a TwinCAT task to obtain cyclic calls.
- The ITcADI interface can be used to access data areas of a module.
- ITcWatchSource is implemented by default; it facilitates ADS device notifications and other features.
- The ITcTask interface, which is implemented by the tasks of the real-time system, provides information about the cycle time, the priority and other task information.
- The ITComObjectServer interface is implemented by the ObjectServer and referenced by all modules.

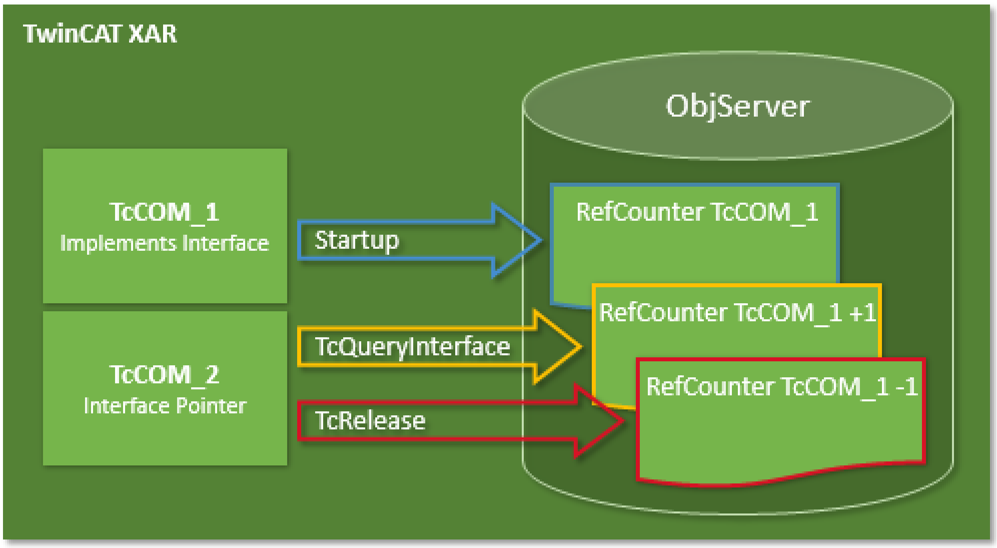
A whole series of general interfaces has already been defined. General interfaces have the advantage that their use supports the exchange and recycling of modules. User-defined interfaces should only be defined if no suitable general interfaces are available.

Class Factories

"Class Factories" are used for creating modules in C++. All modules contained in a driver have a common Class Factory. The Class Factory registers once with the ObjectServer and offers its services for the development of certain module classes. The module classes are identified by the unique ClassID of the module. When the ObjectServer requests a new module (based on the initialization data of the configurator or through other modules at runtime), the module selects the right Class Factory based on the ClassID and triggers creation of the module via its ITcClassFactory interface.

Module service life

Similar to COM, the service life of a module is determined via a reference counter (RefCounter). The reference counter is incremented whenever a module interface is queried. The counter is decremented when the interface is released. An interface is also queried when a module logs into the ObjectServer (the ITComObject interface), so that the reference counter is at least 1. The counter is decremented on logout. When the counter reaches 0, the module deletes itself automatically, usually after logout from the ObjectServer. If another module already maintains a reference (has an interface pointer), the module continues to exist, and the interface pointer remains valid, until this pointer is released.



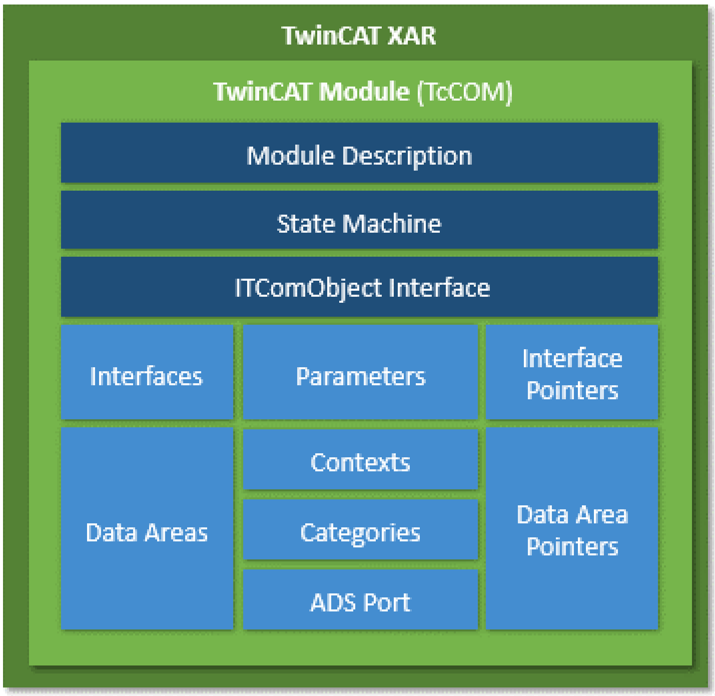
8.1.1.1 TwinCAT module properties

A TcCOM module has a number of formally defined, prescribed and optional properties. The properties are sufficiently formalized to enable interchangeable application. Each module has a module description, which describes the module properties. They are used for configuring the modules and their relationships with each other.

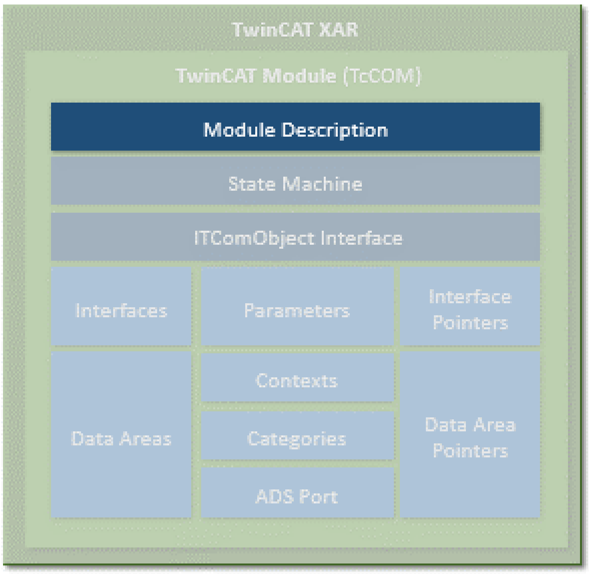
If a module is instantiated in the TwinCAT runtime, it registers itself with a central system instance, the ObjectServer. This makes it reachable and parameterizable for other modules and also for general tools. Modules can be compiled independently and can therefore also be developed, tested and updated independently. Modules can be very simple, e.g. they may only contain a basic function such as low-pass filter. Or they may be very complex internally and contain the whole control system for a machine subassembly.

There are a great many applications for modules; all tasks of an automation system can be specified in modules. Accordingly, no distinction is made between modules, which primarily represent the basic functions of an automation system, such as real-time tasks, fieldbus drivers or a PLC runtime system, and user- or application-specific algorithms for controlling a machine unit.

The diagram below shows a common TwinCAT module with his main properties. The dark blue blocks define prescribed properties, the light blue blocks optional properties.



Module description



Each TcCOM module has some general description parameters. These include a ClassID, which unambiguously references the module class. It is instantiated by the corresponding ClassFactory. Each module instance has an ObjectID, which is unique in the TwinCAT runtime. In addition there is a parent ObjectID, which refers to a possible logical parent.

The description, state machine and parameters of the module described below can be reached via the ITComObject interface (see "Interfaces").

Class description files (*.tmc)

The module classes are described in class description files (TwinCAT Module Class; *.tmc).

These files are used by developers to describe the module properties and interfaces, so that others can use and embed the module. In addition to general information (vendor data, module class ID etc.), optional module properties are described.

- Supported categories
- Implemented interfaces
- Data areas with corresponding symbols
- Parameter
- Interface pointers
- Data pointers, which can be set

The system configurator uses the class description files mainly as a basis for the integration of a module instance in the configuration, for specifying the parameters and for configuring the links with other modules.

They also include the description of all data types in the modules, which are then adopted by the configurator in its general data type system. In this way, all interfaces of the TMC descriptions present in the system can be used by all modules.

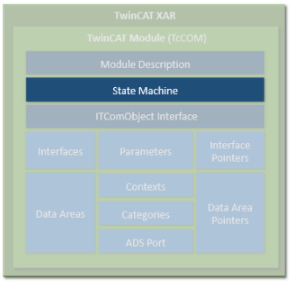
More complex configurations involving several modules can also be described in the class description files, which are preconfigured and linked for a specific application. Accordingly, a module for a complex machine unit, which internally consists of a number of submodules, can be defined and preconfigured as an entity during the development phase.

Instance description files (*.tmi)

An instance of a certain module is described in the instance description file (TwinCAT Module Instance; *.tmi). The instance descriptions are based on a similar format, although in contrast to the class description files they already contain concrete specifications for the parameters, interface pointers etc. for the special module instance within a project.

The instance description files are created by TwinCAT Engineering (XAE), when an instance of a class description is created for a specific project. They are mainly used for the exchange of data between all tools involved in the configuration. However, the instance descriptions can also be used cross-project, for example if a specially parameterized module is to be used again in a new project.

State machine

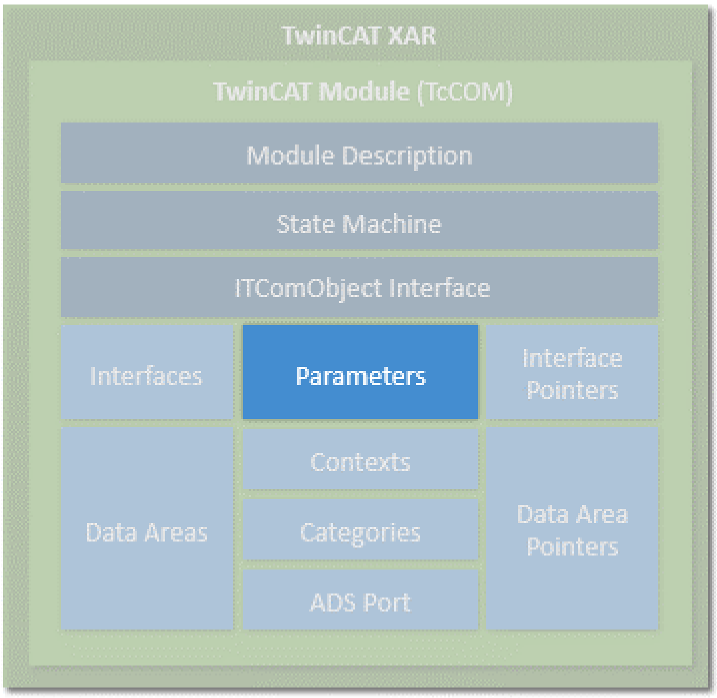


Each module contains a state machine, which describes the initialization state of the module and the means with which this state can be modified from outside. The state machine describes the states, which occur during starting and stopping of the module. This relates to module creation, parameterization and production in conjunction with the other modules.

Application-specific states (e.g. of the fieldbus or driver) can be described in their own state machines. The state machine of the TcCOM modules defines the states INIT, PREOP, SAFEOP and OP. Although the state designations are the same as under EtherCAT fieldbus, the actual states differ. When the TcCOM module implements a fieldbus driver for EtherCAT, it has two state machines (module and fieldbus state machine), which are passed through sequentially. The module state machine must have reached the operating state (OP) before the fieldbus state machine can start.

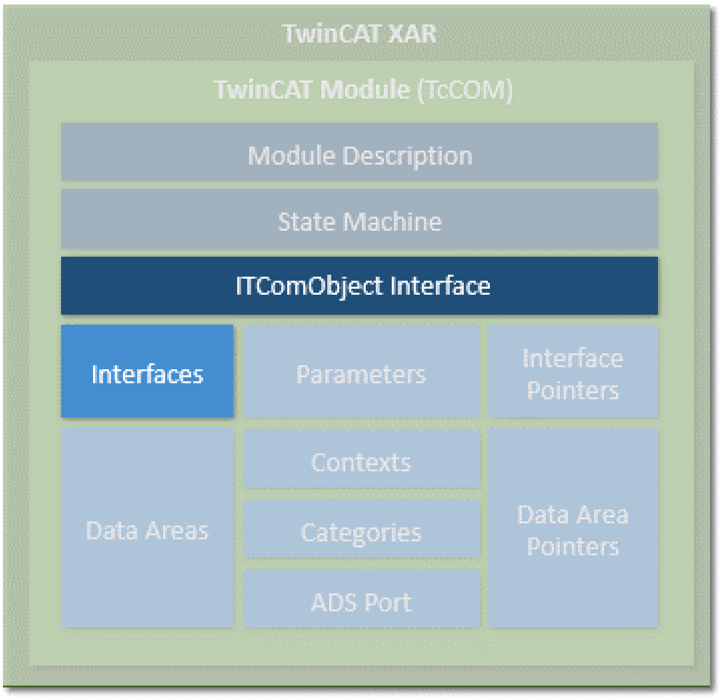
The state machine is described [▶ 67] in detail separately.

Parameter



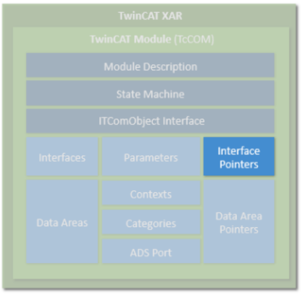
Modules can have parameters, which can be read or written during initialization or later at runtime (OP state). Each parameter is designated by a parameter ID. The uniqueness of the parameter ID can be global, limited global or module-specific. Further details can be found in the "ID Management" section. In addition to the parameter ID, the parameter contains the current data; the data type depends on the parameter and is defined unambiguously for the respective parameter ID.

Interfaces



Interfaces consist of a defined set of methods (functions), which offer modules through which they can be contacted by other modules. Interfaces are characterized by a unique ID, as described above. A module must support at least the ITComObject interface and may in addition contain as many interfaces as required. An interface reference can be queried by calling the method "TcQueryInterface" with specification of the corresponding interface ID.

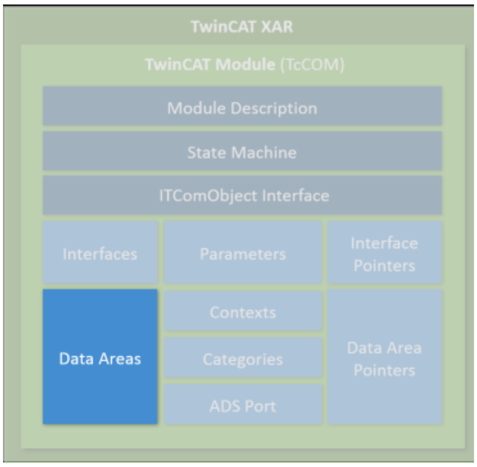
Interface pointers



Interface pointers behave like the counterpart of interfaces. If a module wants to use an interface of another module, it must have an interface pointer of the corresponding interface type and ensure that it points to the other module. The methods of the other module can then be used.

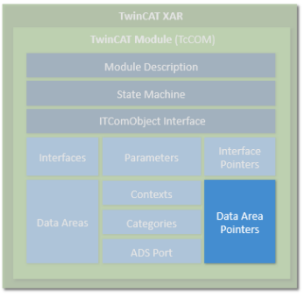
Interface pointers are usually set on startup of the state machine. During the transition from INIT to PREOP (IP), the module receives the object ID of the other modules with the corresponding interface; during the transition from PREOP to SAFEOP (PS) or SAFEOP to OP (SO), the instance of the other modules is searched with the ObjectServer, and the corresponding interface is set with the Method Query interface. During the state transition in the opposite direction, i.e. from SAFEOP to PREOP (SP) or OP to SAFEOP (OS), the interface must be enabled again.

Data areas



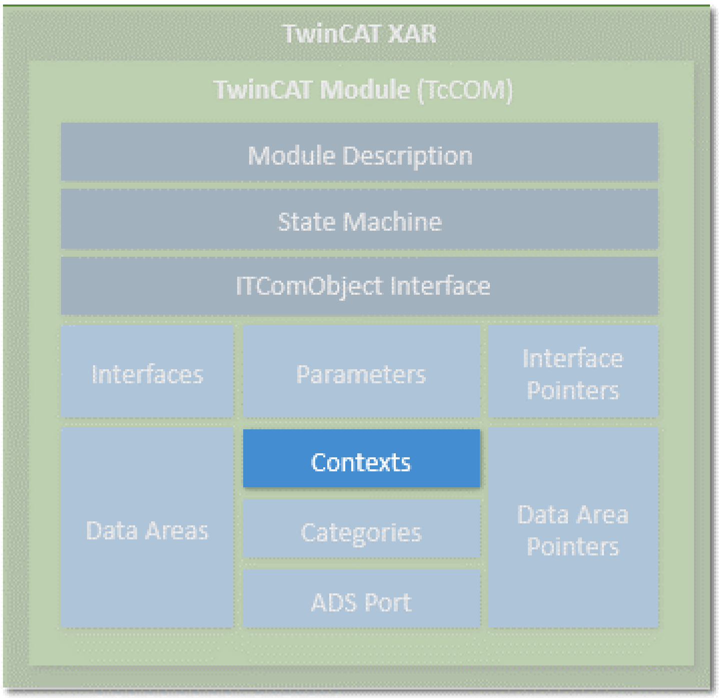
Modules can contain data areas, which can be used by the environment (e.g. by other modules or the IO area of TwinCAT). These data areas can contain any data. They are often used for process image data (inputs and outputs). The structure of the data areas is defined in the device description of the module. If a module has data areas, which it wants to make accessible for other modules, it implements the ITcADI interface to enable access to the data. Data areas can contain symbol information, which describes the structure of the respective data area in more detail.

Data area pointer



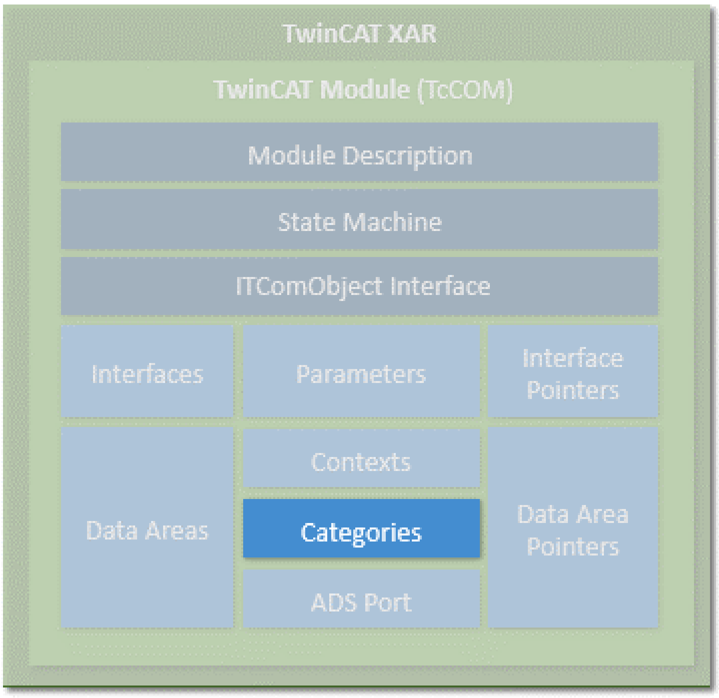
If a module wants to access the data area of other modules, it can contain data area pointers. These are normally set during initialization of the state machine to data areas or data area sections of other modules. The access is directly to the memory area, so that corresponding protection mechanisms for competing access operations have to be implemented, if necessary. In many cases it is preferable to use a corresponding interface.

Context



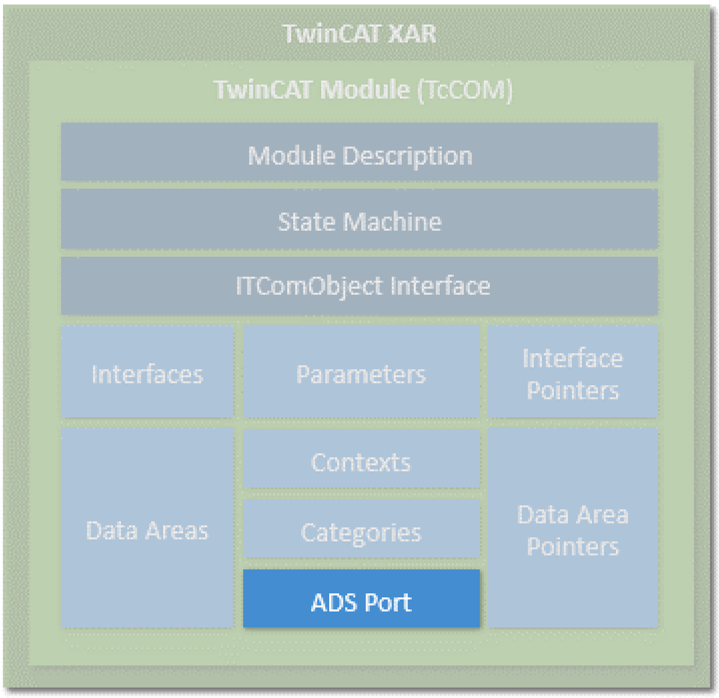
The context should be regarded as real-time task context. Context is required for the configuration of the modules, for example. Simple modules usually operate in a single time context, which therefore requires no detailed specification. Other modules may partly be active in several contexts (e.g. an EtherCAT master can support several independent real-time tasks, or a control loop can process control loops of the layer below in another cycle time). If a module has more than one time-dependent context, this must be specified the in the module description.

Categories



Modules can offer categories by implementing the interface ITComObjectCategory. Categories are enumerated by the ObjectServer, and objects, which use this to associated themselves with categories, can be queried by the ObjectServer (ITComObjectEnumPtr).

ADS



Each module that is entered in the ObjectServer can be reached via ADS. The ObjectServer uses the ITComObject interface of the modules in order to read or write parameters or to access the state machine, for example. In addition, a dedicated ADS port can be implemented, through which dedicated ADS commands can be received.

System module

In addition, the TwinCAT runtime provides a number of system modules, which make the basic runtime services available for other modules. These system modules have a fixed, constant ObjectID, through which the other modules can access it. An example for such a system module is the real-time system, which makes the basic real-time system services, i.e. generation of real-time tasks, available via the ITcRTime interface. The ADS router is also implemented as a system module, so that other modules can register their ADS port here.

Creation of modules

Modules can be created both in C++ and in IEC 61131-3. The object-oriented extensions of the TwinCAT PLC are used for this purpose. Modules from both worlds can interact via interfaces in the same way as pure C++ modules. The object-oriented extension makes the same interfaces available as in C++.

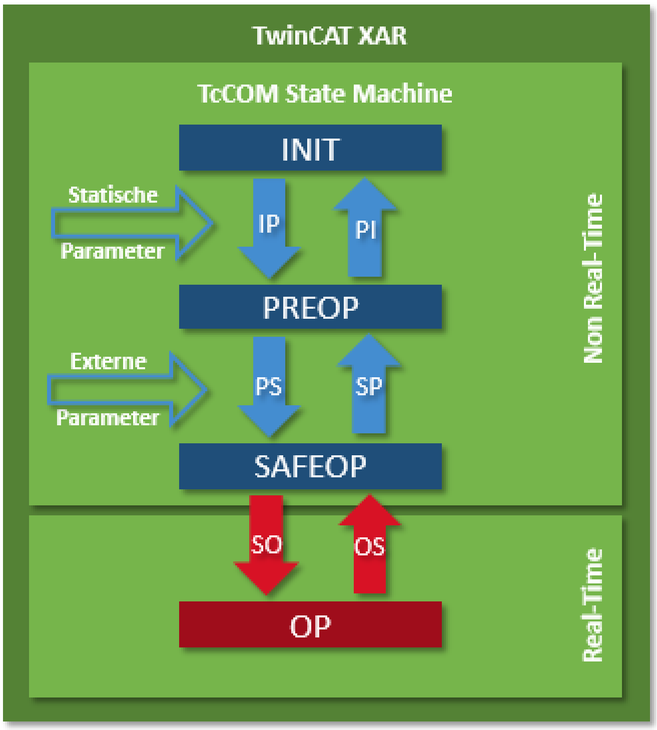
The PLC modules also register via the ObjectServer and can therefore be reached through it. PLC modules vary in terms of complexity. It makes no difference whether only a small filter module is generated or a complete PLC program is packed into a module. Due to the automation, each PLC program is a module within the meaning of TwinCAT modules. Each conventional PLC program is automatically packed into a module and registers itself with the ObjectServer and one or several task modules. Access to the process data of a PLC module (e.g. mapping with regard to a fieldbus driver) is also controlled via the defined data areas and ITcADI.

This behavior remains transparent and invisible for PLC programmers, as long as they decide to explicitly define parts of the PLC program as TwinCAT modules, so that they can be used with suitable flexibility.

8.1.1.2 TwinCAT module state machine

In addition to the states (INIT, PREOP, SAFEOP and OP), there are corresponding state transitions, within which general or module-specific actions have to be executed or can be executed. The design of the state machine is very simple. In any case, there are only transitions to the next or previous step,

resulting in the following state transitions: INIT to PREOP (IP), PREOP to SAFEOP (PS) and SAFEOP to OP (SO). In the opposite direction there are the following state transitions: OP to SAFEOP (OS), SAFEOP to PREOP (SP) and PREOP to INIT (PI). Up to and including the SAFEOP state, all states and state transitions take place within the non-real-time context. Only the transition from SAFEOP to OP, the OP state and the transition from OP to SAFEOP take place in the real-time context. This differentiation is relevant when resources are allocated or activated, or when modules register or deregister with other modules.



State: INIT

The INIT state is only a virtual state. Immediately after creation of a module, the module changes from INIT to PREOP, i.e. the IP state transition is executed. The instantiation and the IP state transition always take place together, so that the module never remains in INIT state. Only when the module is removed does it remain in INIT state for a short time.

Transition: INIT to PREOP (IP)

During the IP state transition, the module registers with the ObjectServer with its unique ObjectID. The initialization parameters, which are also allocated during object creation, are transferred to the module. During this transition the module cannot establish connections to other modules, because it is not clear whether the other modules already exist and are registered with the ObjectServer. When the module requires system resources (e.g. memory), these can be allocated during the state transition. All allocated resources have to be released again during the transition from PREOP to INIT (PI).

State: PREOP

In PREOP state, module creation is complete and the module is usually fully parameterized, even if further parameters may be added during the transition from PREOP to SAFEOP. The module is registered in the ObjectServer, although no connections with other modules have been created yet.

Transition: PREOP to SAFEOP (PS)

In this state transition the module can establish connections with other modules. To this end it has usually received, among other things, ObjectIDs of other modules with the initialization data, which are now converted to actual connections with these modules via the ObjectServer.

The transition can generally be triggered by the system according to the configurator, or by another module (e.g. the parent module). During this state transition further parameters can be transferred. For example, the parent module can transfer its own parameters to the child module.

State: SAFEOP

The module is still in the non-real-time context and is waiting to be switched to OP state by the system or by other modules.

Transition: SAFEOP to OP (SO)

The state transition from SAFEOP to OP, the state OP, and the transition from OP to SAFEOP take place in the real-time context. System resources may no longer be allocated. On the other hand, resources can now be requested by other modules, and modules can register with other modules, e.g. in order to obtain a cyclic call during tasks.

This transition should not be used for long-running tasks. For example, file operations should be executed during the PS transition.

State: OP

In OP state the module starts working and is fully active in the meaning of the TwinCAT system.

Transition: OP to SAFEOP (OS)

This state transition takes place in the real-time context. All actions from the SO transition are reversed, and all resources requested during the SO transition are released again.

Transition: SAFEOP to PREOP (SP)

All actions from the PS transition are reversed, and all resources requested during the PS transition are released again.

Transition: PREOP to INIT (PI)

All actions from the IP transition are reversed, and all resources requested during the IP transition are released again. The module signs off from the ObjectServer and usually deletes itself (see "Service life").

8.2 Interfaces

8.2.1 Interface ITComObject

The ITComObject interface is implemented by every TwinCAT module. It makes basic functionalities available.

Syntax

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

Methods

Name	Description
TcGetObjectId(OTCID& objId) [▸ 69]	Saves the object ID using the given OTCID reference.
TcSetObjectId [▸ 70]	Sets the object ID of the object to the given OTCID.
TcGetObjectName [▸ 70]	Saves the object names in the buffer with the given length.
TcSetObjectName [▸ 70]	Sets the object name of the object to given CHAR*.
TcSetObjState [▸ 71]	Initializes a transition to a predefined state.
TcGetObjState [▸ 71]	Queries the current state of the object.
TcGetObjPara [▸ 72]	Queries an object parameter identified with its PTCID.
TcSetObjPara [▸ 72]	Sets an object parameter identified with its PTCID.
TcGetParentObjId [▸ 72]	Saves the parent object ID with the help of the given OTCID reference.
TcSetParentObject [▸ 73]	Sets the parent object ID to the given OTCID.

Comments

The ITComObject interface is implemented by every TwinCAT module. It makes functionalities available regarding the state machine and Information from/to the TwinCAT system.

8.2.1.1 Method ITcComObject:TcGetObjectId(OTCID& objId)

The method saves the object ID with the help of the given OTCID reference.

Syntax

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

Parameter

objId: (type: OTCID&) reference to OTCID value.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▸ 32].

Description

The method stores Object ID using given OTCID reference.

8.2.1.2 Method ITcComObject:TcSetObjectId

The method TcSetObjectId sets object’s object ID to the given OTCID.

Syntax

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

Parameters

objId: (type: OTCID) The OTCID, which should be set.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

At present, the return value is ignored by the TwinCAT tasks.

Description

Indicates the success of the ID change.

8.2.1.3 Method ITcComObject:TcGetObjectName

The method TcGetObjectName stores the Object name into buffer with given length.

Syntax

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

Parameters

objName: (type: CHAR*) the name, which should be set.

nameLen: (type: ULONG) the maximum length to write.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The method TcGetObjectName stores the Object name into buffer with given length.

8.2.1.4 Method ITcComObject:TcSetObjectName

The method TcSetObjectName sets objects’s Object Name to the given CHAR*.

Syntax

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

Parameter

objName: (type: CHAR*) the name of the object to be set.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The method TcSetObjectName sets objects’s Object Name to the given CHAR*.

8.2.1.5 Method ITcComObject:TcSetObjState

The method TcSetObjState initializes a transition to given state.

Syntax

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

Parameter

state: (type: TCOM_STATE) displays the new state.

ipSrv: (type: ITComObjectServer*) ObjServer that handles the object.

pInitData: (type: PTComInitDataHdr) points to a list of parameters (optional), see macro IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA as an example of how the list can be iterated.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The method TcSetObjState initializes a transition to given state.

8.2.1.6 Method ITcComObject:TcGetObjState

The method TcGetObjState retrieves the current state of the object.

Syntax

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

Parameter

pState: (type: TCOM_STATE*) pointer to the state.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The TcGetObjState method queries the current state of the object.

8.2.1.7 Method ITcComObject:TcGetObjPara

The method TcGetObjPara retrieves a object parameter identified by its PTCID.

Syntax

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

Parameter

pid: (type: PTCID) parameter ID of the object parameter.

nData: (type: ULONG&) max. length of the data.

pData: (type: PVOID&) pointer to the data.

pgp: (type: PTCGP) reserved for future extension, NULL forwarded.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The method TcGetObjPara retrieves a object parameter identified by its PTCID.

8.2.1.8 Method ITcComObject:TcSetObjPara

The method TcSetObjPara sets a object parameter identified by its PTCID.

Syntax

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

Parameter

pid: (type: PTCID) parameter ID of the object parameter.

nData: (type: ULONG) max. length of the data.

pData: (type: PVOID) pointer to the data.

pgp: (type: PTCGP) reserved for future extension, NULL forwarded.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The method TcSetObjPara sets a object parameter identified by its PTCID.

8.2.1.9 Method ITcComObject:TcGetParentObjId

The method TcGetParentObjId stores Parent Object ID using given OTCID reference.

Syntax

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

Parameter

objId: (type: OTCID&) reference to OTCID value.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

Description

The method TcGetParentObjId stores Parent Object ID using given OTCID reference.

8.2.1.10 Method ITcComObject:TcSetParentObjId

The method TcSetParentObjId sets Parent Object ID using given OTCID reference.

Syntax

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

Parameter

objId: (type: OTCID) reference to OTCID value.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

At present, the return value is ignored by the TwinCAT tasks.

Description

The method TcSetParentObjId sets Parent Object ID using given OTCID reference.

8.2.2 Interface ITcUnknown

ITcUnknown defines the reference counting as well as querying a reference to a more specific interface.

Syntax

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

Declared in: TcInterfaces.h

Required include: -

Methods

Name	Description
TcAddRef [▶ 74]	Increments the reference counter.
TcQueryInterface [▶ 74]	Query of the reference to an implemented interface via the IID.
TcRelease [▶ 75]	Decrements the reference counter.

Remarks

Every TcCOM interface is directly or indirectly derived from ITcUnknown. As a consequence every TcCOM module class implements ITcUnknown, because it is derived from ITComObject.

The default implementation for ITcUnknown will delete the object if its last reference is released. Therefore an interface pointer must not be dereferenced after TcRelease() has been called.

8.2.2.1 Method ITcUnknown:TcAddRef

This method increments the reference counter.

Syntax

```
ULONG TcAddRef( )
```

Return Value

Resulting reference count value.

Description

Increments the reference counter and returns the new value..

8.2.2.2 Method ITcUnknown:TcQueryInterface

Query of an interface pointer with regard to an interface that is given by interface ID (IID).

Syntax

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

iid: (Type: RITCID) Interface IID.

pipItf: (PPVOID Type) pointer to interface pointer. Is set when the requested interface type is available from the corresponding instance.

Return value

If successful, S_OK ("0") or another positive value will be returned, cf. Return values. Extended messages refer in particular to the column HRESULT in ADS Return Codes [▶ 32].

If the demanded interface is not available, the method returns ADSERR_DEVICE_NOINTERFACE.

Description

Query reference to an implemented interface by the IID. It is recommended to use smart pointers to initialize and hold interface pointers.

Variant 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;
}
}
```

The interface id associated with the smart pointer can be used as parameter in TcQueryInterface. The operator “&” will return pointer to internal interface pointer member of the smart pointer. Variant 1 assumes that interface pointer is initialized if TcQueryInterface indicates success. If scope is left the destructor of the smart pointer spObj releases the reference.

Variant 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;
}
```

When assigning interface pointer ip to smart pointer spObj method TcQueryInterface is implicitly called with IID_ITComObject on the instance ip refers to. This results in shorter code, however it loses the original return code of TcQueryInterface.

8.2.2.3 Method ITcUnknown:TcRelease

This method decrements the reference counter.

Syntax

```
ULONG TcRelease( )
```

Return Value

Resulting reference count value.

Description

Decrements the reference counter and returns the new value.

If reference counter gets zero, object deletes itself.

More Information:
www.beckhoff.com/te1000

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