# Table of contents

1 Foreword ........................................................................................................................... 5  
  1.1 Notes on the documentation .................................................................................. 5  
  1.2 Safety instructions ................................................................................................. 6  
2 Introduction ....................................................................................................................... 7  
3 Function blocks .................................................................................................................. 8  
  3.1 TcBaseModuleRegistered ......................................................................................... 8  
      3.1.1 TcAddRef ............................................................................................................. 8  
      3.1.2 TcGetObjectId ...................................................................................................... 9  
      3.1.3 TcGetObjectName ............................................................................................... 9  
      3.1.4 TcGetObjPara .................................................................................................... 10  
      3.1.5 TcGetObjState ................................................................................................. 10  
      3.1.6 TcQueryInterface .............................................................................................. 11  
      3.1.7 TcRelease .......................................................................................................... 11  
      3.1.8 TcSetObjId ......................................................................................................... 12  
      3.1.9 TcSetObjectName ............................................................................................. 12  
      3.1.10 TcSetObjPara .................................................................................................. 13  
      3.1.11 TcSetObjState ................................................................................................. 13  
4 Functions ........................................................................................................................... 15  
  4.1 FW_ObjMgr_CreateAndInitInstance ........................................................................... 15  
  4.2 FW_ObjMgr_CreateInstance ..................................................................................... 16  
  4.3 FW_ObjMgr_DeleteInstance ..................................................................................... 16  
  4.4 FW_ObjMgr_GetObjectInstance ............................................................................... 17  
  4.5 FW_SafeRelease ....................................................................................................... 17  
  4.6 FAILED ..................................................................................................................... 18  
  4.7 SUCCEEDED ............................................................................................................. 19  
  4.8 ITCUNKNOWN_TO_PVOID ...................................................................................... 19  
  4.9 PVOID_TO_ITCUNKNOWN ...................................................................................... 19  
  4.10 GuidsEqual ............................................................................................................ 20  
5 Global Constants ............................................................................................................... 21  
  5.1 GVL ......................................................................................................................... 21  
  5.2 Global_Version ........................................................................................................ 21  
6 Error Codes ....................................................................................................................... 22  
  6.1 ADS Return Codes .................................................................................................. 22  
7 Samples ............................................................................................................................... 26  
  7.1 TcCOM_Sample01_PlcToPlc ...................................................................................... 26  
      7.1.1 Creating an FB which provides its functionality globally in the first PLC .............. 27  
      7.1.2 Creating an FB which likewise offers this functionality there as a simple proxy in the second PLC, ................................................................. 31  
      7.1.3 Execution of the sample project.......................................................................... 34  
  7.2 TcCOM_Sample02_PlcToCpp ..................................................................................... 36  
      7.2.1 Instantiating a TwinCAT++ class as a TwinCAT TcCOM Object ........................ 36  
      7.2.2 Creating an FB in the PLC that, as a simple proxy, offers the functionality of the C++ object ................................................................. 37
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.3</td>
<td></td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>7.3</td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>7.3.1</td>
<td></td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>7.3.2</td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>7.3.3</td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>7.4</td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>7.4.1</td>
<td></td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>8.1</td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>8.2</td>
<td></td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>8.2.1</td>
<td></td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>8.2.2</td>
<td></td>
<td></td>
<td>64</td>
</tr>
</tbody>
</table>
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:
with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DANGER</strong></td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td><strong>WARNING</strong></td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td><strong>CAUTION</strong></td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><strong>NOTE</strong></td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

Tip or pointer

This symbol indicates information that contributes to better understanding.
2 Introduction

The PLC library Tc3_Module is used for TcCOM communication.

System requirements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target System</td>
<td>WinXP, WES, Win7, WES7, WEC7</td>
</tr>
<tr>
<td></td>
<td>IPC or CX, (x86, x64, ARM)</td>
</tr>
<tr>
<td>Min. TwinCAT version</td>
<td>3.1.4020.0</td>
</tr>
<tr>
<td>Min. TwinCAT level</td>
<td>TC1200 TC3 PLC</td>
</tr>
</tbody>
</table>
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.

How this TcCOM object is created and can be globally used in the TwinCAT system, will be explained in detail in an example [26].

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.

3.1.1 TcAddRef

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

Interface

Parameter

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

Return value

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

The resulting reference count value is returned.

Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
3.1.2 TcGetObjectId

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

**Interface**

**Parameter**

VAR_INPUT

```
objId : REFERENCE TO OTCID;
```

END_VAR

**objId**: Reference to OTCID value

**Return value**

VAR_OUTPUT

```
TcGetObjectId : HRESULT;
```

END_VAR

Gives information about success of the OTCID query.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

3.1.3 TcGetObjectName

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

**Interface**

**Parameter**

VAR_INPUT

```
objName : POINTER TO SINT;
nameLen : UDWORD;
```

END_VAR

**objName**: the name to be set

**nameLen**: the maximum length of the name to be written

**Return value**

VAR_OUTPUT

```
TcGetObjectName : DINT;
```

END_VAR

Gives information about success of the name query.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
3.1.4 TcGetObjPara

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

**Interface**

**Parameter**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>Parameter-ID of the object parameter</td>
</tr>
<tr>
<td>nData</td>
<td>Maximum length of the data</td>
</tr>
<tr>
<td>pData</td>
<td>Pointer to the data</td>
</tr>
<tr>
<td>pgp</td>
<td>reserved for future expansion, Pass NULL</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcGetObjPara</td>
<td>HRESULT</td>
</tr>
</tbody>
</table>

Gives information about success of the object parameter query.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

3.1.5 TcGetObjState

The TcGetObjState method queries the current state of the object.

**Interface**

**Parameter**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pState</td>
<td>Pointer to the state</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcGetObjState</td>
<td>HRESULT</td>
</tr>
</tbody>
</table>

Gives information about success of the state query.
**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

### 3.1.6 TcQueryInterface

Query of the reference to an implemented interface via the IID.

#### Interface

**Parameter**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>iid  : REFERENCE TO IID;</td>
</tr>
<tr>
<td>pipItf : POINTER TO PVOID;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

**iid:** Interface ID

**pipItf:** Pointer to interface pointer. Is set when the requested interface type is available from the corresponding instance.

#### Return value

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcQueryInterface : HRESULT;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

Gives information about success of the interface query.
If the requested interface is not available, returns the ADS_E_NOINTERFACE method.

#### Necessary release of the interface pointers

All references must be explicitly released again. It is recommended to use FW_SafeRelease [17] 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**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

### 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.

#### Interface

**Parameter**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th>(<em>none</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
</tbody>
</table>
Return value

VAR_OUTPUT
  TcRelease : UDINT;
END_VAR

The resulting reference count value is returned.

Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

3.1.8 TcSetObjId

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

Interface

Parameter

VAR_INPUT
  objId : OTCID;
END_VAR

objId: the OTCID to be set

Return value

VAR_OUTPUT
  TcSetObjId : HRESULT;
END_VAR

Gives information about success of the ID change.

Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

3.1.9 TcSetObjectName

The TcSetObjectName method sets the object name of the object.

Interface

Parameter

VAR_INPUT
  objName : POINTER TO SINT;
END_VAR

objName: the name to be set of the object

Return value
Gives information about success of the name change.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

### 3.1.10 TcSetObjPara

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

**Interface**

**Parameter**

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

- **pid**: Parameter-ID of the object parameter
- **nData**: maximum length of the data
- **pData**: Pointer to the data
- **pgp**: reserved for future expansion, Pass NULL

**Return value**

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

Gives information about success of the parameter change.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

### 3.1.11 TcSetObjState

The TcSetObjState method initializes a transition to the given state.

**Interface**

**Parameter**

```plaintext
VAR_INPUT
    state : TCOM_STATE;
    pSrv  : IComObjectServer;
    pInitData : POINTER TO TComInitDataAndId;
END_VAR
```

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

Gives information about success of the state change.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
VAR_INPUT
    state : TCOM_STATE;
    ipSrv : ITComObjServer;
    pInitData : POINTER TO TComInitDataHdr;
END_VAR

state: displays the new state

ipSrv: Object description

pInitData: points to a list of parameters (optional)

Return value:

VAR_OUTPUT
    TcSetObjState : HRESULT;
END_VAR

Gives information about success of the state change.

Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
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

This function generates an instance of the class specified by means of Class-ID and at the same time return 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.

**Interface**

**Parameter**

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

- **clsId**: specifies the class from which an object should be created.
- **iid**: specifies the interface ID to which an interface pointer should be referenced.
- **pipUnk**: returns the interface pointer to the created object.
- **objId**: 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**: 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**: specifies the object name which should be assigned for the newly created object.
- **state**: specifies the state into which the newly created object should be put. Typically Operational (TCOM_STATE.TCOM_STATE_OP) is specified.
- **pInitData**: Pointer to initialization parameter (optional)

**Return value**

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

**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 [16], in order to delete the generated instance at the latest in the destructor of the object which created the instance.
4.2 **FW_ObjMgr_CreateInstance**

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.

### Interface

**Parameter**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsId : CLSID;</td>
</tr>
<tr>
<td>iid : IID;</td>
</tr>
<tr>
<td>pipUnk : POINTER TO ITcUnknown;</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW_ObjMgr_CreateInstance : HRESULT;</td>
</tr>
</tbody>
</table>

### 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** in order to delete the generated instance at the latest in the destructor of the object which created the instance.

### Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

4.3 **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.

### Interface

**Parameter**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipUnk : POINTER TO ITcUnknown;</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
pipUnk: specifies the address of the interface pointer to the object. The interface pointer is checked internally for null pointers.

Return value

VAR_OUTPUT
  FW_ObjMgr_DeleteInstance : HRESULT;
END_VAR

Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

4.4 FW_ObjMgr_GetObjectInstance

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

Interface

Parameter

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

oid: Object ID

iid: specifies the interface ID to which an interface pointer should be referenced.

pipUnk: returns the interface pointer to the created object.

Return value

VAR_OUTPUT
  FW_ObjMgr_GetObjectInstance : HRESULT;
END_VAR

Necessary release of the interface pointers

All references must be explicitly released again. It is recommended to use FW_SafeRelease 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

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

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.
Interface

Parameter

VAR_INPUT

   pipUnk : POINTER TO ITcUnknown;
END_VAR

pipUnk: specifies the address of the interface pointer to the object. The interface pointer is checked internally for null pointers.

Return value

VAR_OUTPUT

   FW_SafeRelease : HRESULT;
END_VAR

Sample

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

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

4.6 FAILED

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

Interface

Parameter

VAR_INPUT

   hr : DINT;
END_VAR

Return value

VAR_OUTPUT

   FAILED : BOOL;
END_VAR

Requirements

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
4.7 SUCCEEDED

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

**Interface**

**Parameter**

VAR_INPUT

hr : DINT;
END_VAR

**Return value**

VAR_OUTPUT

SUCCEEDED : BOOL;
END_VAR

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

4.8 ITCUNKNOWN_TO_PVOID

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

**Interface**

**Parameter**

VAR_INPUT

itcUnknown : ITcUnknown;
END_VAR

**Return value**

VAR_OUTPUT

ITCUNKNOWN_TO_PVOID : PVOID
END_VAR

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

4.9 PVOID_TO_ITCUNKNOWN

This conversion function converts a pointer to VOID to an interface pointer of the type ITcUnknown.
The function GuidsEqual checks two GUID objects for their equality to one another.

**Interface**

**Parameter:**

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

**pGuidA**: Pointer to GUID object

**pGuidB**: Pointer to GUID object

**Return value:**

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

The method returns TRUE when both arguments are equal.

**Requirements**

<table>
<thead>
<tr>
<th>TwinCAT Version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
# 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
```

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

## 5.2 Global_Version

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

`stLibVersion_Tc3_Module`: Version information of the Tc3_Module-Library (type: ST_LibVersion).
6 Error Codes

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

<table>
<thead>
<tr>
<th>HighWord of HRESULT</th>
<th>Group of error codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>16#9811</td>
<td>Ads Error codes</td>
</tr>
</tbody>
</table>

### 6.1 ADS Return Codes

Grouping of error codes: 0x000 [22], 0x500 [22], 0x700 [23], 0x1000 [25]

#### Global error codes

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

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

**General ADS error codes**
## Error Codes

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

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x274C</td>
<td>10060</td>
<td>WSAETIMEOUT</td>
<td>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.</td>
</tr>
<tr>
<td>0x274D</td>
<td>10061</td>
<td>WSAECONNREFUSED</td>
<td>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.</td>
</tr>
<tr>
<td>0x2751</td>
<td>10065</td>
<td>WSAEHOSTUNREACH</td>
<td>No route to host - a socket operation referred to an unavailable host.</td>
</tr>
</tbody>
</table>

More Winsock error codes: Win32 error codes
7 Samples

The TcCOM_Sample01 sample [26] 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 [36] 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++ driver 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 [40] 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 [27]
• Creating an FB in the second PLC that, as a simple proxy, also offers this functionality there [➔ 31]
• Execution of the sample project [➔ 34]

Downloading the sample: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc3_Module/Resources/zip/9007201597787659.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

<table>
<thead>
<tr>
<th>TwinCAT version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64, ARM</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

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 [➔ 8] 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.

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.

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.
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 ToBaseModuleRegistered 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.
10. Implement the TcQueryInterface() method of the Interface ITcUnknown [64]. 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:

\[
\text{nRes} := \text{nIn1} + \text{nIn2} \quad \text{and} \quad \text{nRes} := \text{nIn1} - \text{nIn2}
\]

12. Add one or more instances of this function block in the MAIN program block or in a global variable list.

\[\Rightarrow\] The implementation in the first PLC is complete.

13. 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 process image.

### 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.

\[\Rightarrow\] 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.

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() 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.

```
FB_CalculationProxy.GetInterfacePointer ▷
1  METHOD GetInterfacePointer : HRESULT
2  VAR
3  END_VAR
4
5 IP mObjID <> 0 THEN
6     IF (ip = 0) THEN // only get interface pointer if it is not already existing
7         GetInterfacePointer := FW_ObjMgr_GetObjectInstance (oid:=mObjID, iid:=iid, pIpUnk:=ADR(ip));
8     ELSE
9         GetInterfacePointer := E_HRESULTAdErr.EXISTS;
10     END_IF
11 ELSE
12     GetInterfacePointer := E_HRESULTAdErr.INVALIDOBJID;
13 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 ▷
1  METHOD FB_exit : BOOL
2  VAR_INPUT
3      hInCopyCode : BOOL; // if TRUE, the exit method is called
4  END_VAR
5
6 IF NOT hInCopyCode THEN // if not online change
7     FW_SafeRelease (ADR(ip));
8 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.
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.
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.
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** [Top]

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

3. **Execution of the sample project** [Top]

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

**System requirements**

<table>
<thead>
<tr>
<th>TwinCAT version</th>
<th>Hardware</th>
<th>Libraries to be Integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

### 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\n
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.

```c++
FUNCTION_BLOCK FB_IncrementProxy

VAR_OUTPUT
    Ip : IIncrement;
END_VAR

VAR
    [attribute 'TcInitSymbol']
    [attribute 'displaymode':-'hex']
    nObjId : OCID; // Instance configured to be retrieved
    iid : IID := TC_GLOBAL_IW_ID_LIST.IID_IIncrement;
    hrInit : HRESULT;
END_VAR
```
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.

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 [26]).

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() [17]. In this connection the object itself increments a reference counter.

6. It is imperative to release the used reference again. To this end call the FW_SafeRelease() function [17] in the FB_exit destructor of the 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.

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.

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 [41]

2. Creating an FB in the PLC that creates the C++ object and offers its functionality [42]

3. Execution of the sample project [44]

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

System requirements

<table>
<thead>
<tr>
<th>TwinCAT version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>
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.

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.

   ![Diagram of PLC structure](image)

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

   ```
   FB_IncrementProxyDynInstance
   
   FUNCTION_BLOCK FB_IncrementProxyDynInstance
   VAR_OUTPUT
   ip : IIncrement;
   END_VAR

   VAR
   classId : CLSID := STRING_TO_GUID('687cd5b2-9bf8-4838-b946-e6ec25fc6060');
   iid : IID := ICLASSES_IDLIST.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.

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 [15] 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.
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()` function in the FB_exit destructor of the 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.

\[\text{This completes the implementation of the Proxy function block.}\]

\[\text{The sample is ready for testing.}\]
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.

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, 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

<table>
<thead>
<tr>
<th>TwinCAT version</th>
<th>Hardware</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4020</td>
<td>x86, x64</td>
<td>Tc3_Module</td>
</tr>
</tbody>
</table>

7.4.1 Implementation of the sample

The PLC page adopted by TcCOM Sample 01 [26]. 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.
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.

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.

   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.
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.

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 in detail separately.
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
Appendix

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 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
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcGetObjectoid(OTCID&amp; objId)</td>
<td>Saves the object ID using the given OTCID reference.</td>
</tr>
<tr>
<td>[60] TcSetObjectId</td>
<td>Sets the object ID of the object to the given OTCID.</td>
</tr>
<tr>
<td>[60] TcGetObjectonName</td>
<td>Saves the object names in the buffer with the given length.</td>
</tr>
<tr>
<td>[60] TcSetObjectName</td>
<td>Sets the object name of the object to given CHAR*.</td>
</tr>
<tr>
<td>[61] TcSetObjState</td>
<td>Initializes a transition to a predefined state.</td>
</tr>
<tr>
<td>[61] TcGetObjectState</td>
<td>Queries the current state of the object.</td>
</tr>
<tr>
<td>[62] TcGetObjPara</td>
<td>Queries an object parameter identified with its PTCID.</td>
</tr>
<tr>
<td>[62] TcSetObjPara</td>
<td>Sets an object parameter identified with its PTCID.</td>
</tr>
<tr>
<td>[63] TcGetParentObjId</td>
<td>Saves the parent object ID with the help of the given OTCID reference.</td>
</tr>
<tr>
<td>[63] TcSetParentObject</td>
<td>Sets the parent object ID to the given OTCID.</td>
</tr>
</tbody>
</table>
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 [22].

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 [22].

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::TcGetObjectObjectName
The method TcGetObjectObjectName stores the Object name into buffer with given length.
Syntax

```c
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 [22].

**Description**

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

**8.2.1.4 Method ITcComObject::TcSetObjectName**

The method TcSetObjectName sets object's Object Name to the given CHAR*.

Syntax

```c
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 [22].

**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

```c
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 [22].

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 [22].

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 [22].

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

```c
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 [\ref{22}].

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

```c
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 [\ref{22}].

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

```c
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** [22].

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**

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

Declared in: TcInterfaces.h

Required include: -

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcAddRef [64]</td>
<td>Increments the reference counter.</td>
</tr>
<tr>
<td>TcQueryInterface [65]</td>
<td>Query of the reference to an implemented interface via the IID.</td>
</tr>
<tr>
<td>TcRelease [66]</td>
<td>Decrements the reference counter.</td>
</tr>
</tbody>
</table>

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**

```c
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

```cpp
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](#22).

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:

```cpp
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:

```cpp
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
    HRESULT hr = S_OK;
    ITComObjectPtr spObj = ip;
    if (spObj != NULL)
    {
        spObj->TcGetObjPara(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

\[
\text{ULONG TcRelease()}
\]

Return Value

Resulting reference count value.

Description

Decrementes the reference counter and returns the new value.

If reference counter gets zero, object deletes itself.
More Information: