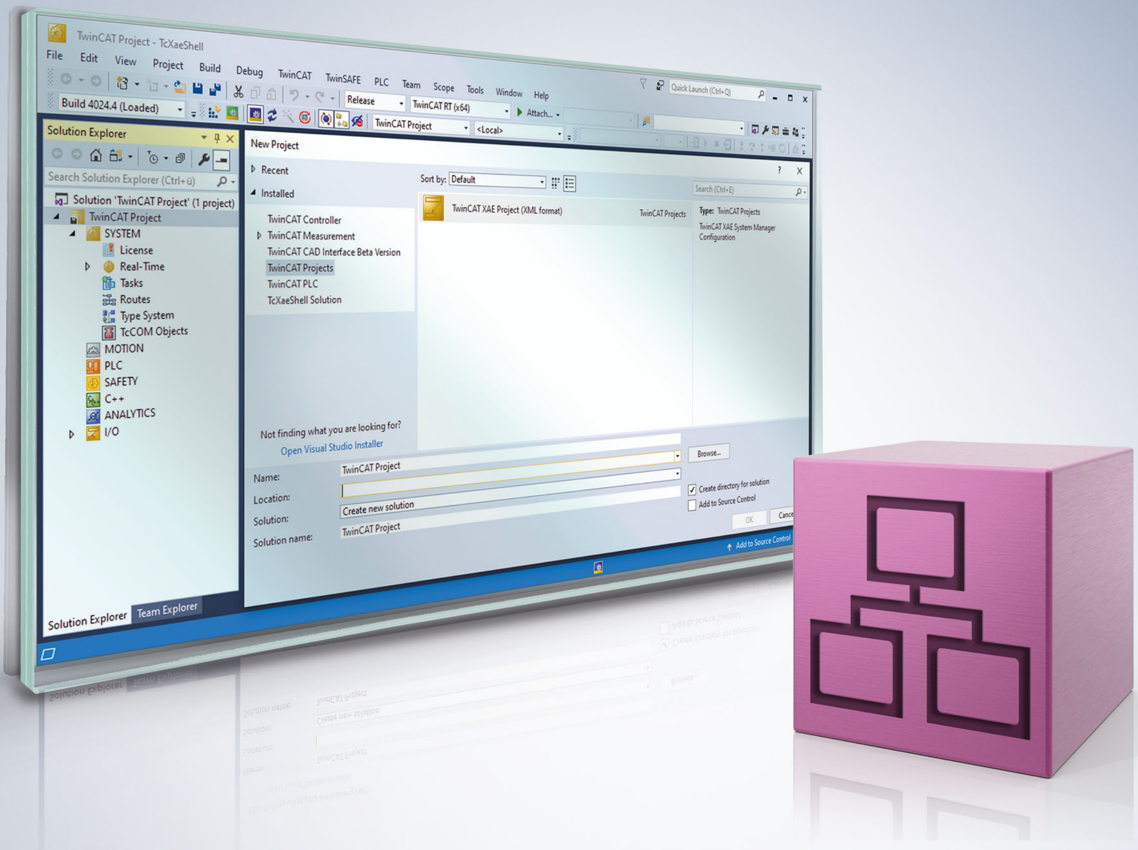
BECKHOFF New Automation Technology

Manual | EN

TF6311

TwinCAT 3 | TCP/UDP Realtime



2023-11-28 | Version: 1.8.0

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all 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 notice.
No claims to modify 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 and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

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

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

Copyright

1.2 For your safety

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Signal words

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

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

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

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

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

1.3 Notes on information security

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

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

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

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

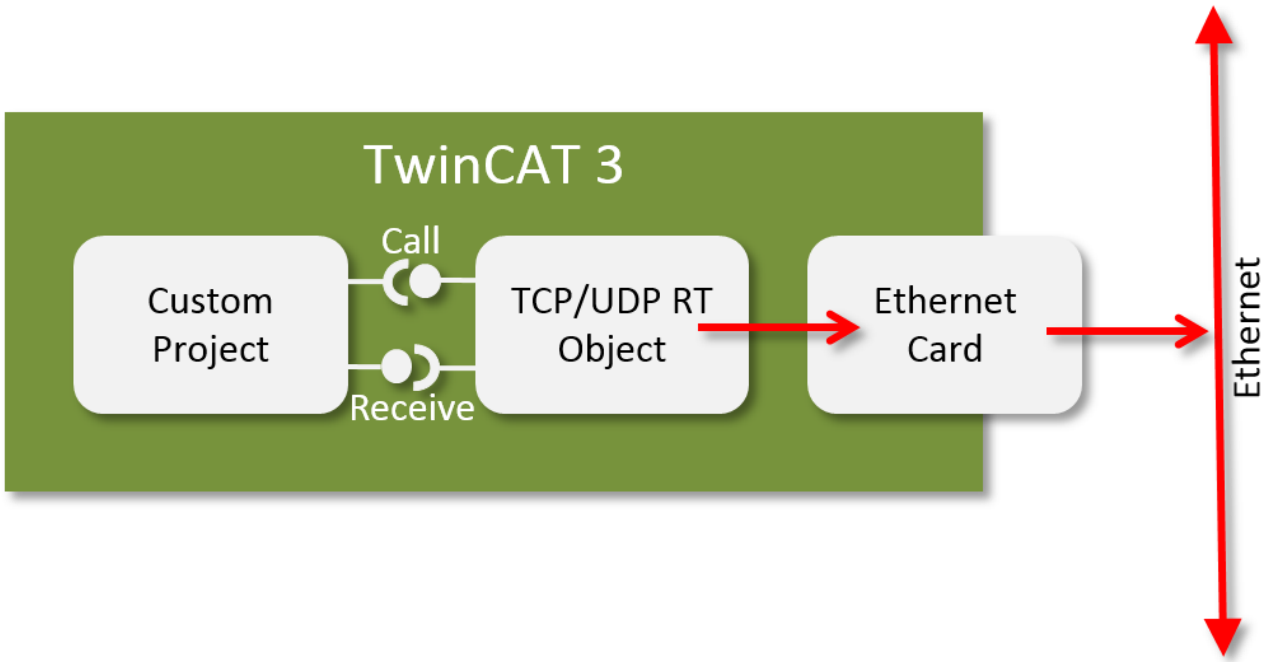
2 Overview

The “TCP/UDP Realtime” Function (TF6311) offers direct access to network cards from the real-time environment. Access can be either from the PLC (61131-3) or C++.

The following protocols are supported:

- TCP/IP
- UDP/IP
- ARP / Ping

This section describes the concept of interfaces as API [▶ 51]. An introduction is provided by means of sample programs [▶ 38].



Regardless of the protocol, the communication between the project using the protocol and TwinCAT is realized with a pair of interfaces:

- An Interface Pointer provides support for sending data and establishing connections etc.
- The implementation of a receiver interface provides feedback for the project in the form of events or data, based on callbacks.

The communication partner of these interface pairs is a “TCP/UDP RT” TcCom object, which is instantiated and configured with the network card.

- Depending on the protocol, the Quickstarts [▶ 11] provide a good introduction.
- The configuration process is documented under Configuration [▶ 33].
- The interfaces are described in the Programmer’s reference [▶ 51] and illustrated through samples [▶ 38].

2.1 Comparison TF6310 TF6311

The products TF6310 "TCP/IP" and TF6311 "TCP/UDP Realtime" offer similar functionality.

This page provides an overview of similarities and differences of the products:

	TF 6310	**TF 6311**
TwinCAT	TwinCAT 2 / 3	TwinCAT 3
Client/Server	Both	Both
Large / unknown networks	++	+
Determinism	+	++
High-volume data transfer	++	+
Programming languages	PLC	PLC and C++
Operating system	Win32/64, CE5/6/7	Win32/64, CE7
UDP-Mutlicast	Yes	No
Trial license	Yes	Yes
Protocols	TCP, UDP	TCP, UDP, Arp/Ping
Hardware requirements	Variable	TwinCAT-compatible network card
Socket configuration	See operating system (WinSock)	TCP/UDP RT TcCom Parameter [▶ 55]

The Windows firewall cannot be used, since the TF6311 is directly integrated in the TwinCAT system. In larger / unknown networks we recommend using the TF6310.

2.2 Restrictions

The following limitations exist for the product:

- No local communication in real-time or between real-time and Windows operating system. (Alternative: communication via a second network interface.)
- Multicast is not supported.
- The EL6601 and EL6614 cannot be used for TF6311 TCP/UDP real-time.
- If breakpoints are used, we strongly advise to use different network interfaces, since a breakpoint stops parts of the TwinCAT systems, which may be relevant for the communication with Engineering.

3 Installation / Licensing

The Function TF6311 requires no separate installation; all software components are available once TwinCAT 3 has been installed.

- A “TC3 TCP UDP RT” license is required.
  The dependence is entered by adding the “TCP/UDP RT” object to the project as a license. It can also be specified manually.
- A trial license can be created and used.

4 Quick Starts

This section contains detailed step-by-step instructions for some protocols.
They illustrate the use of the product in a simple manner. The samples are intended to facilitate understanding; they do not provide comprehensive implementation instructions. At the application level, the handling must be programmed in detail (e.g. the behavior on arrival of corresponding TCP events).

The function TF6311 “TCP/UDP real-time” has extensive capabilities:

- different protocols (TCP, UDP, ARP/Ping)
- different programming languages (PLC / C++) and
- communication directions (client / server)

Step-by-step instructions are not available for all combinations. Once the basic concept [▸ 8] has been understood, further implementations can be derived in conjunction with the existing step-by-step instructions and samples [▸ 38].

4.1 Quick Start (PLC / UDP)

The sample implements an “echo service”: A UDP server is started on a port (default: 10000). If this server receives a UDP packet, it returns the content to the sender (with same IP and same port).

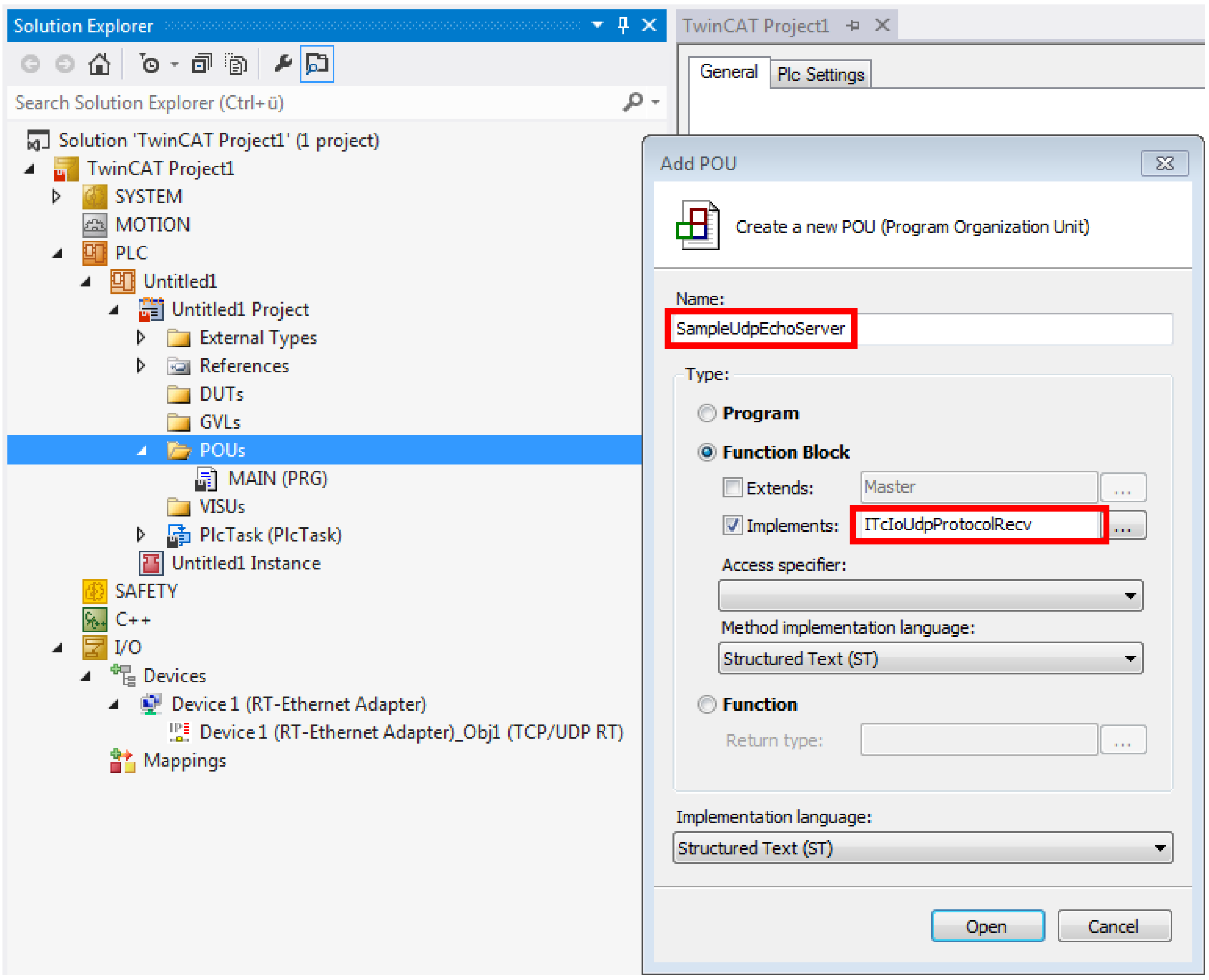
The sample is also available for download under Sample 02 [▸ 40]. In addition to the Quick Start, the download contains extended code, which does not affect the basic functionality.

Implementation of the UDP echo server in a PLC project

✓ A TwinCAT solution was generated

1. If no PLC project exists in the TwinCAT solution, you have to create one.
2. A function block is generated, which implements the interface "ItcIoUdpProtocolRecv". This creates a method, which is called when UDP packets arrive.
   By right-clicking on the node "POU" in the PLC project you can allocate names in the Popup window,

activate "SampleUdpEchoServer" and "Implements" by ticking, and select the interface mentioned with the button "…":



The declaration part of the function block contains several variables in the declaration:

- Oid: Configurable reference to the TCP/UDP RT module
- ipUdp: Interface pointer to the UdpProtocol, which is implemented by the TCP/UDP RT module
- udpPort: Port used for receiving

3. The declaration part is created in this way:

```
{attribute 'c++_compatible'}
FUNCTION_BLOCK SampleUdpEchoServer IMPLEMENTS ITcIoUdpProtocolRecv
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    {attribute 'TcInitSymbol'}
    oid:                OTCID;
    ipUdp:              ITcIoUdpProtocol;
    nUdpPort:           UINT := 10000;
    nReceivedPakets:    UINT;
    hrInit :            HRESULT;
    hrSend :            HRESULT;
END_VAR
```

The CheckReceive() method of the TCP/UDP RT module must be called in the body of the function block.

4. The body is created in this way:

```
IF ipUdp <> 0 THEN
    ipUdp.CheckReceived();
END_IF
```

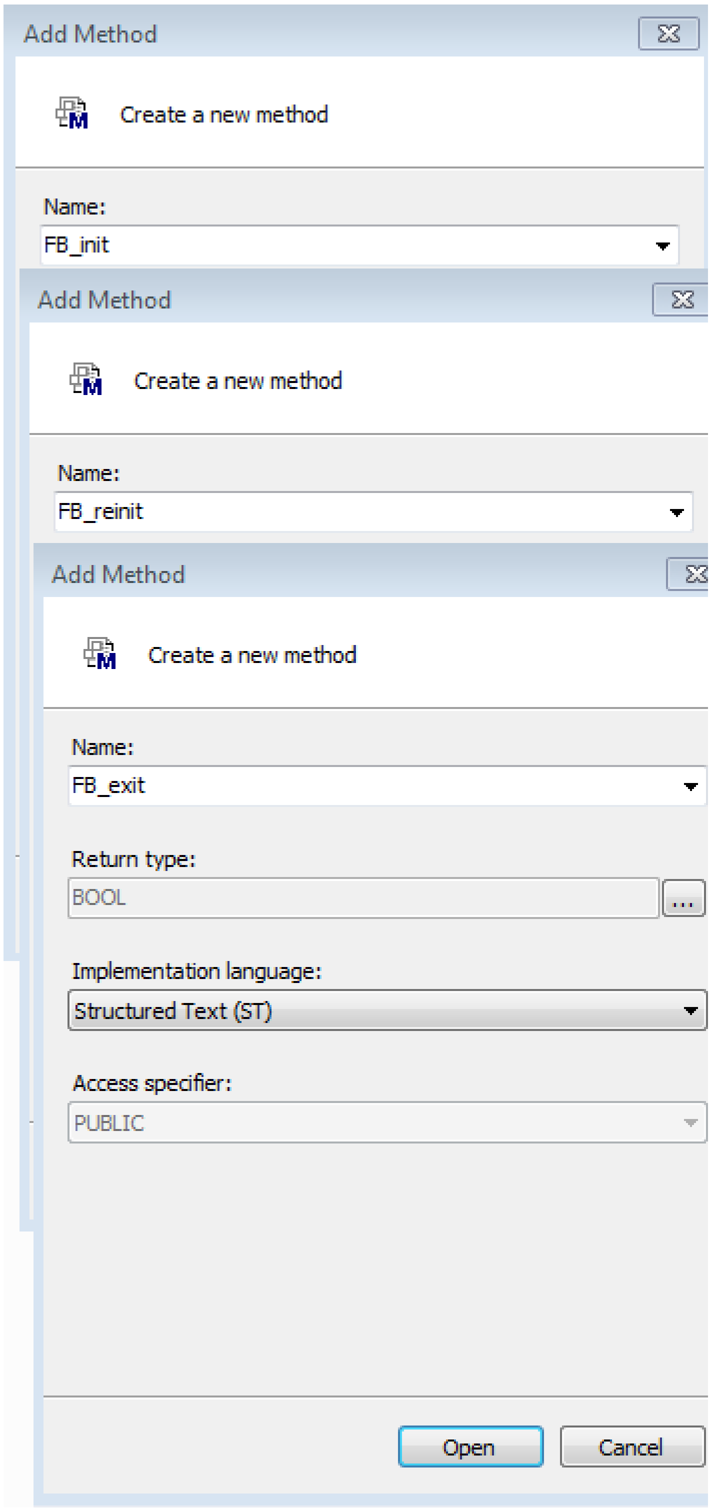
The method "ReceiveData", which was created through implementation of the interface, will be called repeatedly via "CheckReceived": one call for each packet received in the meantime.

5. The method has both sender information and data as input parameters. In this sample, the "SendData" method returns an incoming packet as response (with sender/receiver reversed). The implementation is done as follows:

```
nReceivedPakets := nReceivedPakets+1;
IF ipUdp <> 0 THEN
    hrSend := ipUdp.SendData(ipAddr, udpSrcPort, udpDestPort, nData, pData, TRUE, 0); // send
data back
END_IF
```

During start and finish, a reference to the "UdpProtocol" interface must be set from the configured OID; corresponding approvals should be taken care of during shutdown.

6. The function block requires the methods "FB_init", "FB_reinit" and "FB_exit", which can be created by right-clicking on the function block "Add…" method:


Add Method
Create a new method
Name:
FB_init

Add Method
Create a new method
Name:
FB_reinit

Add Method
Create a new method
Name:
FB_exit
Return type:
BOOL
Implementation language:
Structured Text (ST)
Access specifier:
PUBLIC

Appropriate signatures are generated automatically, so that only the actual body has to be realized. Of particular significance is the "RegisterReceiver" call, which opens a UDP port for reception.

7. The "FB_init" method requires two local variables:

```
VAR
    ipSrv: ITComObjectServer;
END_VAR
```

8. The "FB_init" method is implemented as follows:

```
IF NOT bInCopyCode THEN // no online change
IF ipUdp = 0 AND oid <> 0 THEN
hrInit := FW_ObjMgr_GetObjectInstance(oid:=oid, iid:=TC_GLOBAL_IID_LIST.IID_ITcIoUdpProtocol,
pipUnk:=ADR(ipUdp) );
IF SUCCEEDED(hrInit) THEN
IF SUCCEEDED(ipUdp.RegisterReceiver(nUdpPort, THIS^)) THEN //open port
FB_init := TRUE;
ELSE
FB_init := FALSE;
FW_SafeRelease(ADR(ipUdp));
END_IF
END_IF
ELSIF oid = 0 THEN
FB_init := FALSE;
hrInit := ERR_INVALID_PARAM;
END_IF
END_IF
```

In the "FB_reinit" method, which is executed during an OnlineChange, the TCP/UDP RT object must be supplied with the new address for the callbacks.

9. The "FB_reinit" method is implemented as follows:

```
IF (ipUdp <> 0) THEN
ipUdp.RegisterReceiver(updPort, THIS^);
FB_reinit := TRUE:
END_IF
```

The port must be closed again during shutdown (but not during OnlineChange, cf. bInCopyCode).

10. The "FB_exit" method is implemented as follows:

```
IF (NOT bInCopyCode AND ipUdp <> 0) THEN //Shutdown
     ipUdp.UnregisterReceiver(updPort);
    FW_SafeRelease(ADR(ipUdp));
     FB_exit := TRUE;
ELSE
     FB_exit := FALSE;
END_IF
```

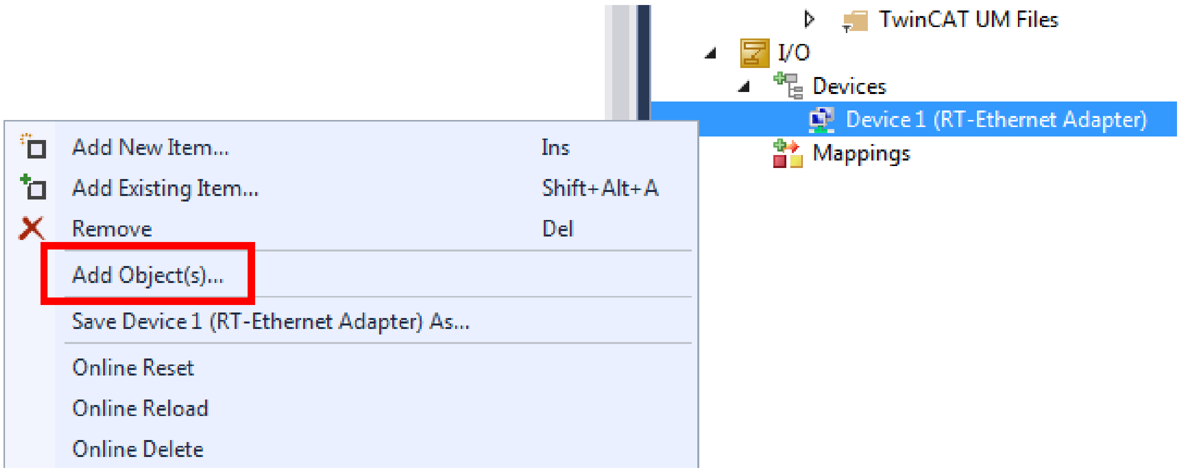
11. Finally, the function block must be called:

```
PROGRAM MAIN
VAR
        udp1 : SampleUdpEchoServer;
END_VAR

udp1();
```

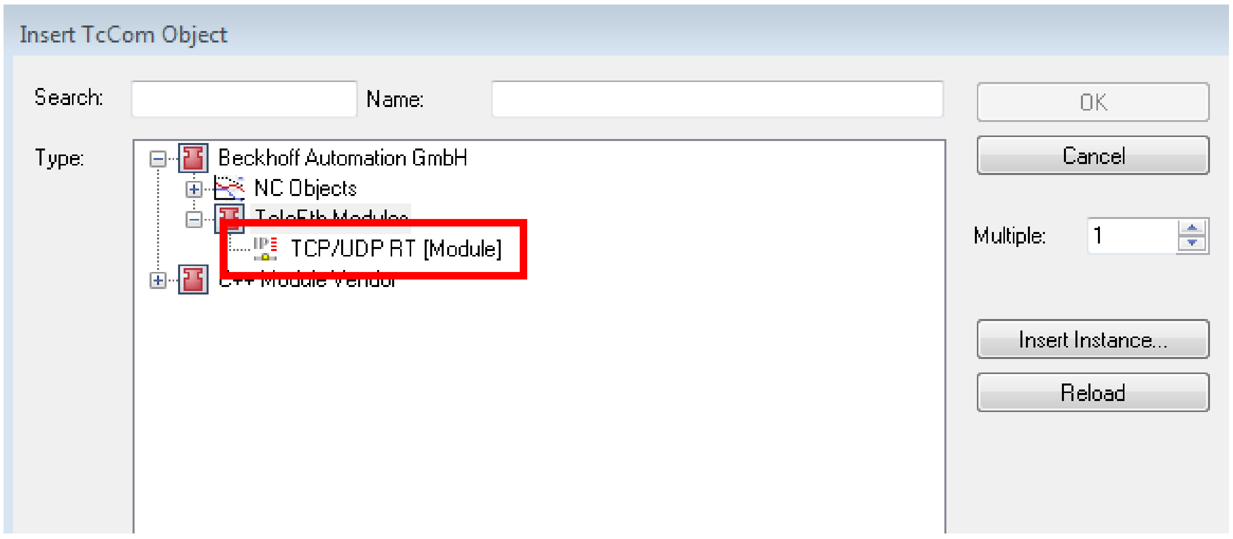
"TCP/UDP RT" module configuration

Notice **Variable names relating to TCP are used here. They have to be substituted accordingly.**

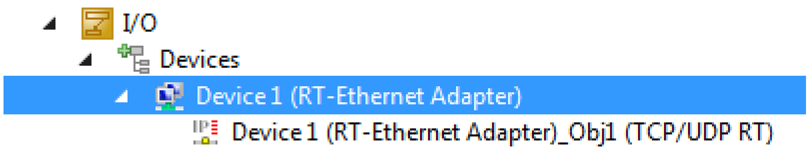
1. Create the “TCP/UDP RT” module under the RT Ethernet adapter by selecting “Add Object(s)…” in the context menu.



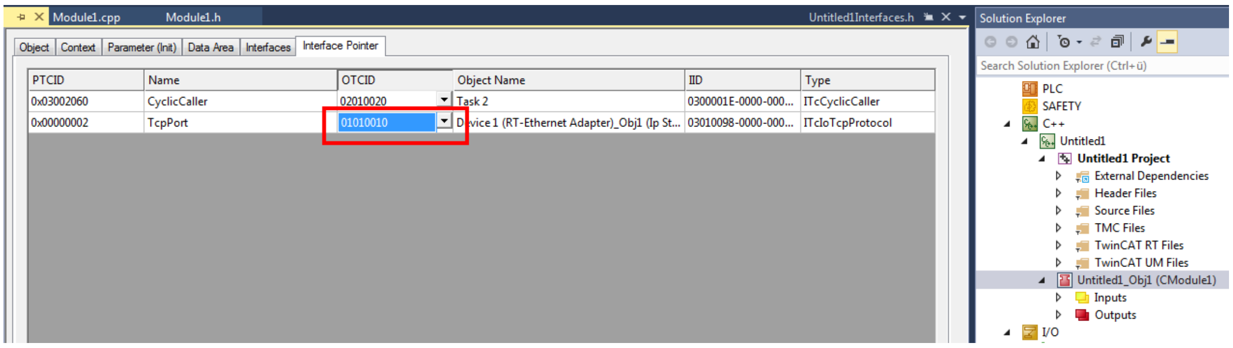
2. Then select the “TCP/UDP RT” module:



⇨ The TCP/UDP RT object is created under the adapter.

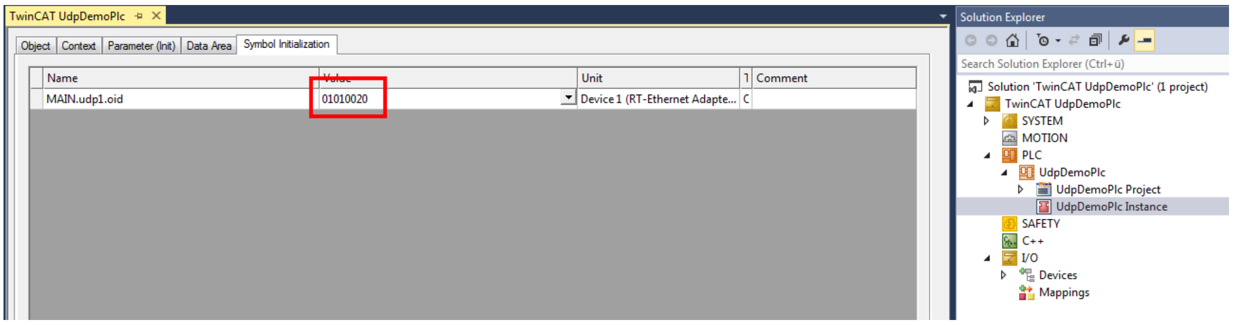

I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

3. Parameterize the previously created instance of the module (here: Module1) under “Interface Pointer” “TcpProt” with the OID of the created “TCP/UDP RT” object:



PTCID	Name	OTCID	Object Name	IID	Type
0x03002060	CyclicCaller	02010020	Task 2	0300001E-0000-000...	ITcCyclicCaller
0x00000002	TcpPort	01010010	Device 1 (RT-Ethernet Adapter)_Obj1 (Ip St...	03010098-0000-000...	ITcIoTcpProtocol

4. For PLC projects this configuration is also done at the instance, under the tab “Symbol Initialization”:



Name	Value	Unit		Comment
MAIN.udp1.oid	01010020	Device 1 (RT-Ethernet Adapte...	C	

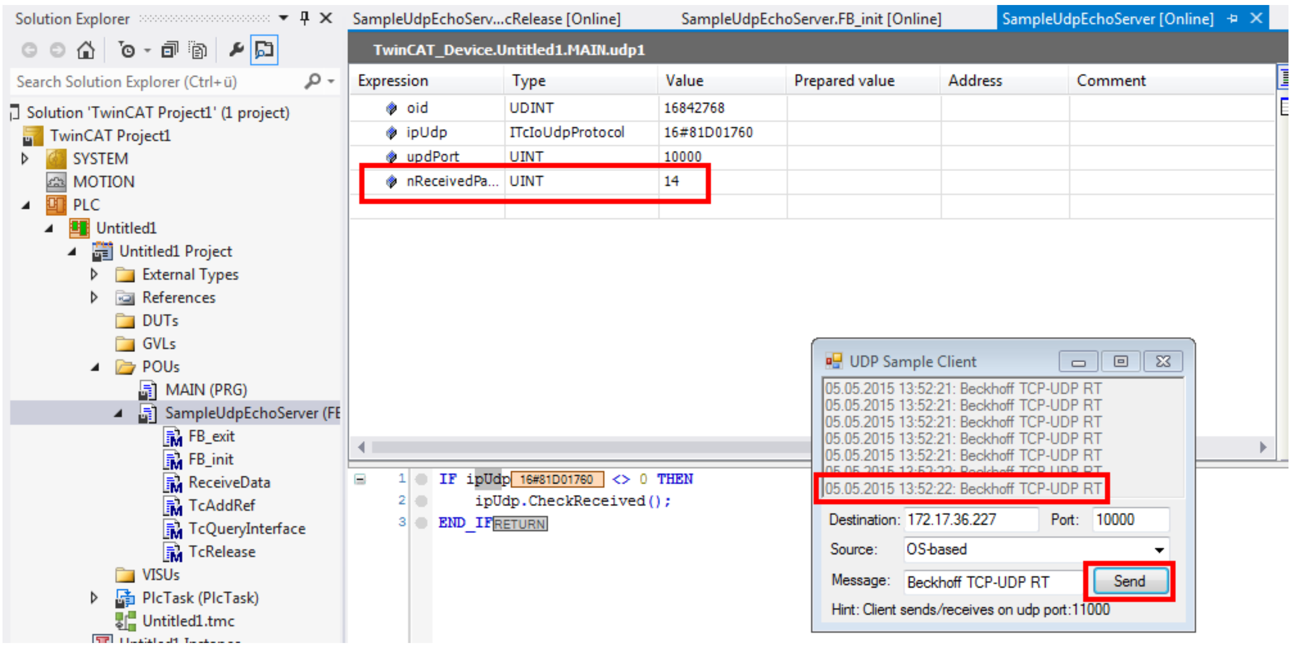
⇨ The configuration is thus completed

Disconnection by the operating system in Promiscuous mode

If Promiscuous mode is active at the RT Ethernet adapter in the “Adapter” tab, any TCP connection attempts are blocked by the operating system, since it does not recognize a port opened in the TCP/UDP RT object.

Testing

Once the configuration has been activated, a UDP packet can be sent to the PLC via the UDP Sample Client [▸ 44]. It can be observed that each call increments the counter. The client displays the returned packets at the top.



TwinCAT_Device.Untitled1.MAIN.udp1

Expression	Type	Value	Prepared value	Address	Comment
oid	UDINT	16842768			
ipUdp	ITcIoUdpProtocol	16#81D01760			
updPort	UINT	10000			
nReceivedPa...	UINT	14			

```
IF ipUdp 16#81D01760 <> 0 THEN
    ipUdp.CheckReceived();
END_IF RETURN
```

```
05.05.2015 13:52:21: Beckhoff TCP-UDP RT
05.05.2015 13:52:21: Beckhoff TCP-UDP RT
05.05.2015 13:52:21: Beckhoff TCP-UDP RT
05.05.2015 13:52:21: Beckhoff TCP-UDP RT
05.05.2015 13:52:21: Beckhoff TCP-UDP RT
05.05.2015 13:52:22: Beckhoff TCP-UDP RT
```

Destination: 172.17.36.227 Port: 10000
Source: OS-based
Message: Beckhoff TCP-UDP RT
Hint: Client sends/receives on udp port:11000

No local communication

The UDP sample client must run on a different computer than the PLC with the TCP/UDP RT object, because no local communication between the Windows operating system and the real-time is available.
Alternatively, a so-called "loop cable" can be used to connect two network ports. The UDP sample client can be forced to use a specific port by selecting the source (dropdown menu "Source").

4.2 Quick Start (C++ / UDP)

The example implements an "echo service": A UDP server is started on a port (default: 10000). If this server receives a UDP packet, it returns the content to the sender (with same IP and same port).

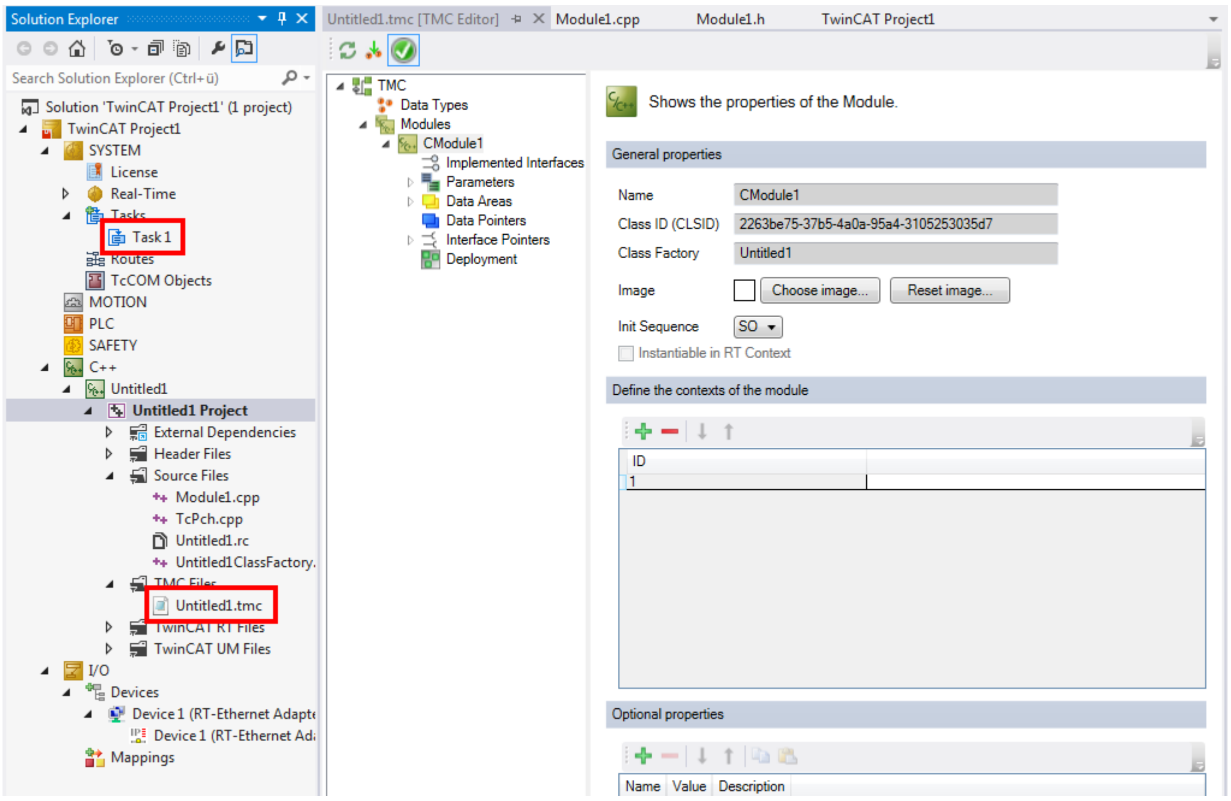
The engineering system must meet the requirements for TwinCAT 3 C++.

The example is also available for download under Sample 02 [▸ 40].

Implementation of the UDP echo server in a C++ project

✓ A TwinCAT solution was generated

1. If no C++ project exists in the TwinCAT solution, you have to create one. Please use the template for “TwinCAT Module Class with Cyclic IO”.
2. Create a task. Under System / Tasks right-click and select “Add new Item…”
   A normal task (without image) is sufficient.
3. In the C++ project, open the TMC editor by double-clicking on the TMC file.

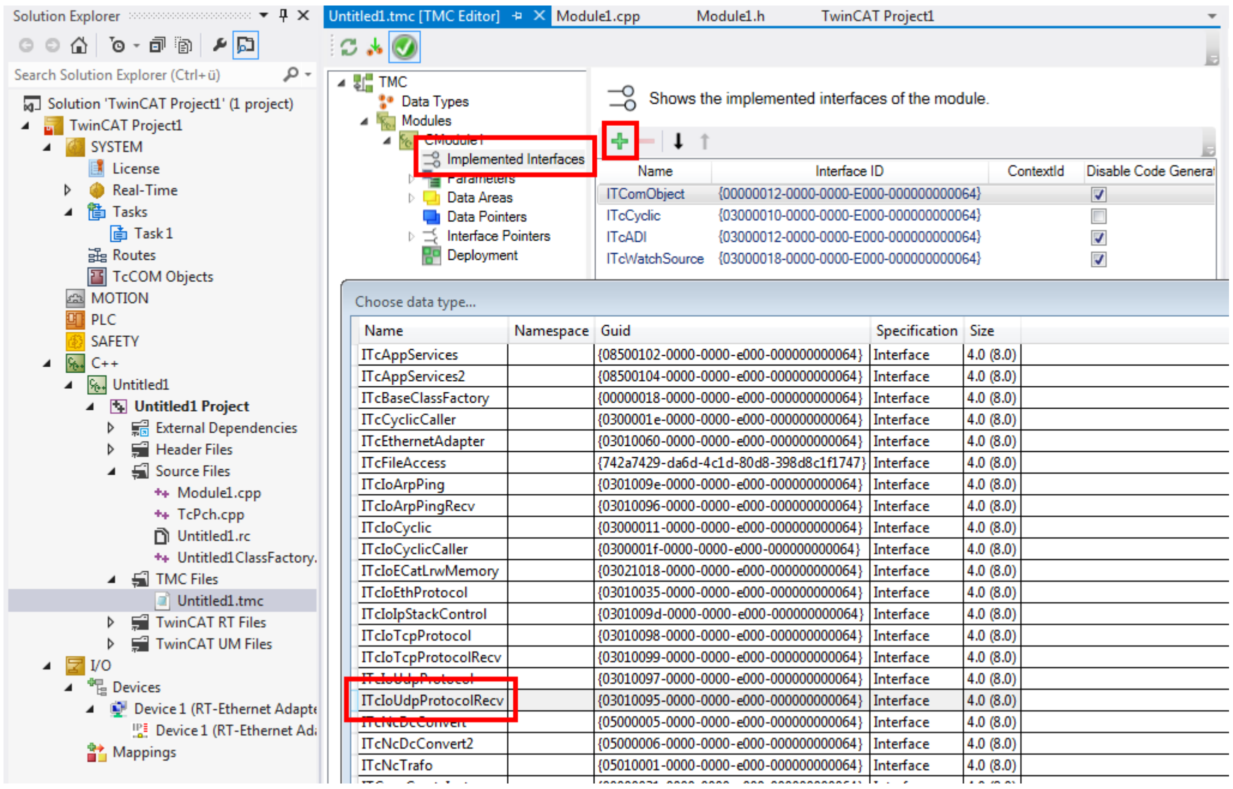


Name: CModule1
Class ID (CLSID): 2263be75-37b5-4a0a-95a4-3105253035d7
Class Factory: Untitled1
Image: ☐
Init Sequence: SO
☐ Instantiable in RT Context

ID
1

The module must implement the *ITcIoUdpProtocolRecv*. This creates a method, which is called when UDP packets arrive.

4. In the TMC editor select “Implemented interfaces” and create them with “+”. A dialog appears, in which the type *ITcIoUdpProtocolRecv* is selected:

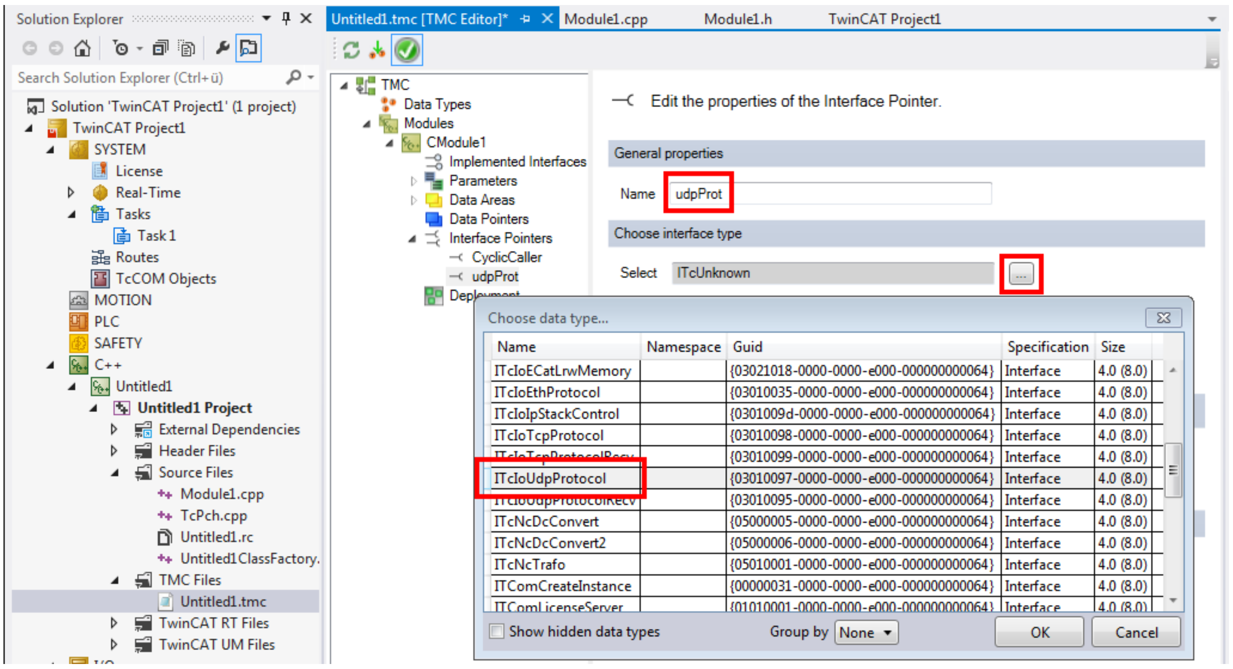


Name	Interface ID	ContextId	Disable Code Generat
ITComObject	{00000012-0000-0000-E000-000000000064}		✓
ITcCyclic	{03000010-0000-0000-E000-000000000064}		
ITcADI	{03000012-0000-0000-E000-000000000064}		✓
ITcWatchSource	{03000018-0000-0000-E000-000000000064}		✓

Name	Namespace	Guid	Specification	Size
ITcAppServices		{08500102-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcAppServices2		{08500104-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcBaseClassFactory		{00000018-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcCyclicCaller		{0300001e-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcEthernetAdapter		{03010060-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcFileAccess		{742a7429-da6d-4c1d-80d8-398d8c1f1747}	Interface	4.0 (8.0)
ITcIoArpPing		{0301009e-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoArpPingRecv		{03010096-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoCyclic		{03000011-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoCyclicCaller		{0300001f-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoECatLrwMemory		{03021018-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoEthProtocol		{03010035-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoIpStackControl		{0301009d-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocol		{03010098-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocolRecv		{03010099-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocol		{03010097-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocolRecv		{03010095-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcDcConvert		{05000005-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcDcConvert2		{05000006-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcTrafo		{05010001-0000-0000-e000-000000000064}	Interface	4.0 (8.0)

The module requires an interface pointer to *ITcIoUdpProtocol*, which contains the reference to the TCP/UDP RT object.

5. In the TMC editor select “Interface Pointer” and press “+”. An interface is created, which can be opened by double-clicking. Assign a name “UdpProt” and set the pointer type with “...:” and the selection in the dialog:



Name	Namespace	Guid	Specification	Size
ITcIoECatLrwMemory		{03021018-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoEthProtocol		{03010035-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoIpStackControl		{0301009d-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocol		{03010098-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocolRecv		{03010099-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocol		{03010097-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocolRecv		{03010095-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcDcConvert		{05000005-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcDcConvert2		{05000006-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcTrafo		{05010001-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITComCreateInstance		{00000031-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITComLicenseServer		{01010001-0000-0000-e000-000000000064}	Interface	4.0 (8.0)

6. The TMC code generator is started once. Right-click on the C++ project and select “TMC Code Generator” in the context menu.

The CheckReceived() method of the TCP/UDP RT module must be called in the CycleUpdate() method in the CPP file of the module (Module1.cpp). As a result, arriving UDP packets are transferred to the implemented method ReceiveData() via callback.

7. The CycleUpdate() method is implemented as follows

```
///<AutoGeneratedContent id="ImplementationOf_ITcCyclic">
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
HRESULT hr = S_OK;
    m_counter+=m_Inputs.Value;
    m_Outputs.Value=m_counter;
    m_spUdpProt->CheckReceived(); // ADDED
    return hr;
}
```

The method “ReceiveData”, which was created through implementation of the interface, will be called repeatedly via CheckReceived(): one call for each packet received in the meantime.

8. The method ReceiveData has both sender information and data as input parameters. In this sample, the SendData method returns an incoming packet as response (with sender/receiver reversed). The implementation is done as follows:

```
///<AutoGeneratedContent id="ImplementationOf_ITcIoUdpProtocolRecv">
HRESULT CModule1::ReceiveData(ULONG ipAddr, USHORT udpDestPort, USHORT udpSrcPort, ULONG nData,
PVOID pData, ETYPE_VLAN_HEADER* pVlan)
{
    HRESULT hr = S_OK;
    // mirror incomming data
    hr = m_spUdpProt->SendData(ipAddr, udpSrcPort, udpDestPort, nData, pData, true);
    m_Trace.Log(tlInfo, FLEAVEA "UDP ReceiveData: IP: %d.%d.%d.%d udpSrcPort: %d DataSize: %d
(hr2=%x) \n",
         ((PBYTE)&ipAddr)[3], ((PBYTE)&ipAddr)[2], ((PBYTE)&ipAddr)[1], ((PBYTE)&ipAddr)[0],
        udpSrcPort, nData, hr);
    return hr;
}
///</AutoGeneratedContent>
```

During start and finish, a reference to the “UdpProtocol” interface must be set from the configured OID; corresponding approvals should be taken care of during shutdown.

9. The start is triggered in the transition from SafeOp to Op. During this process, RegisterReceiver is of particular interest: It opens a UDP port for reception.

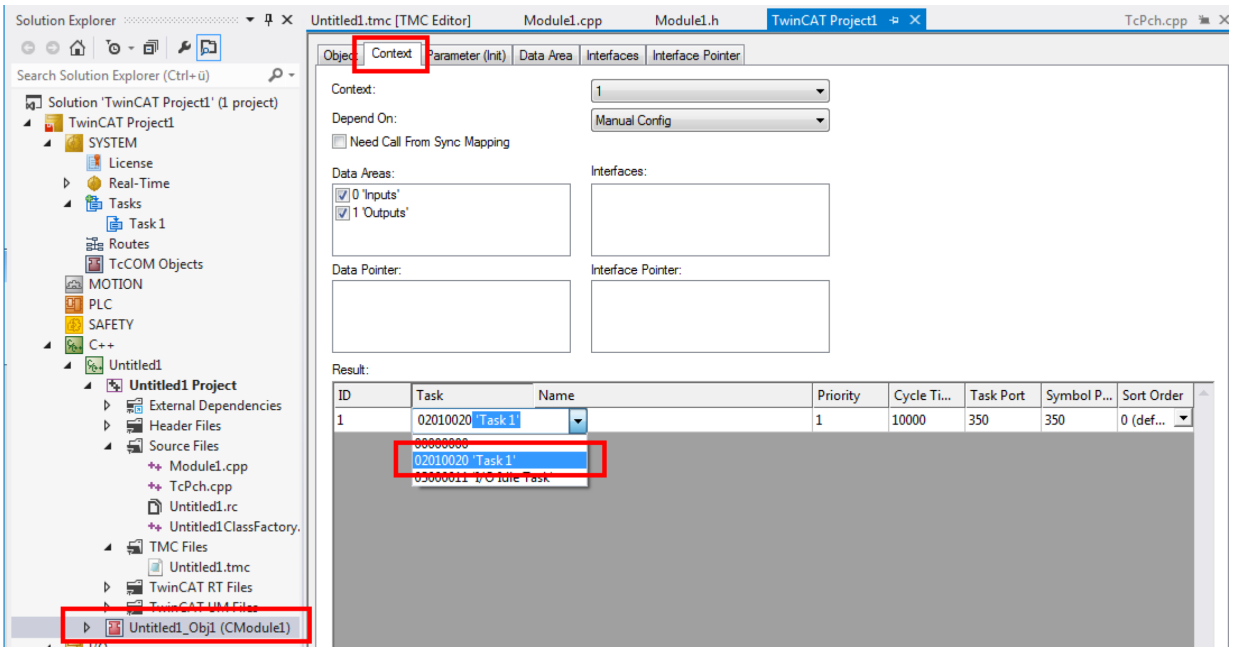
```
HRESULT CModule1::SetObjStateSO()
{
    HRESULT hr = S_OK;
    //START EDITING
    if (SUCCEEDED(hr) && m_spUdpProt.HasOID())
    {
        m_Trace.Log(tlInfo, FLEAVEA "Register UdpProt");
        if (SUCCEEDED_DBG(hr = m_spSrv->TcQuerySmartObjectInterface(m_spUdpProt)))
        {
            m_Trace.Log(tlInfo, FLEAVEA "Server: UdpProt listen to Port: %d", 10000);
            if (FAILED(hr = m_spUdpProt->RegisterReceiver(10000,
THIS_CAST(ITcIoUdpProtocolRecv))))
            {
                m_Trace.Log(tlError, FLEAVEA "Server: UdpProtRegisterReceiver failed on Port:
%d", 10000);
                m_spUdpProt = NULL;
            }
    }
    }

    // If following call is successful the CycleUpdate method will be
called,
    // eventually even before method has been left.
    hr = FAILED(hr) ? hr : AddModuleToCaller();
    // Cleanup if transition failed at some stage
    if ( FAILED(hr) )
    {
        if (m_spUdpProt != NULL)
            m_spUdpProt->UnregisterReceiver(10000);
        m_spUdpProt = NULL;
        RemoveModuleFromCaller();
    }
    //END EDITING
    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}
```

10. The stop takes place in the Op to SafeOp transition. The UDP port is closed again:

```
HRESULT CModule1::SetObjStateOS()
{
    m_Trace.Log(tlVerbose, FENTERA);
    HRESULT hr = S_OK;

    if (m_spUdpProt != NULL)
         m_spUdpProt->UnregisterReceiver(10000);
    m_spUdpProt = NULL;
    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}
```

Finally, the module has to be instantiated and configured

11. Build the project once. Right-click on the module select “Build”
12. Creating an instance of the module. Right-click on the project to open “Add new item…”. Select the appropriate module here.
13. Double-click on the module instance to enable parameterization. First select the task in the “Context” tab.



Context: 1

Depend On: Manual Config

☐ Need Call From Sync Mapping

Data Areas:
☑ 0 'Inputs'
☑ 1 'Outputs'

Interfaces:

Data Pointer:

Interface Pointer:

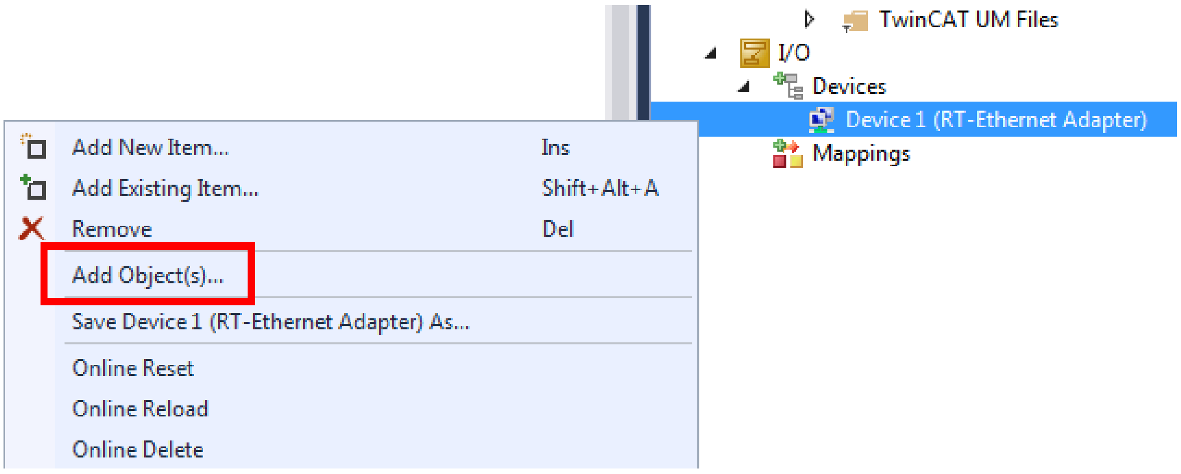
Result:

ID	Task	Name	Priority	Cycle Ti...	Task Port	Symbol P...	Sort Order
1	02010020 'Task 1'		1	10000	350	350	0 (def...

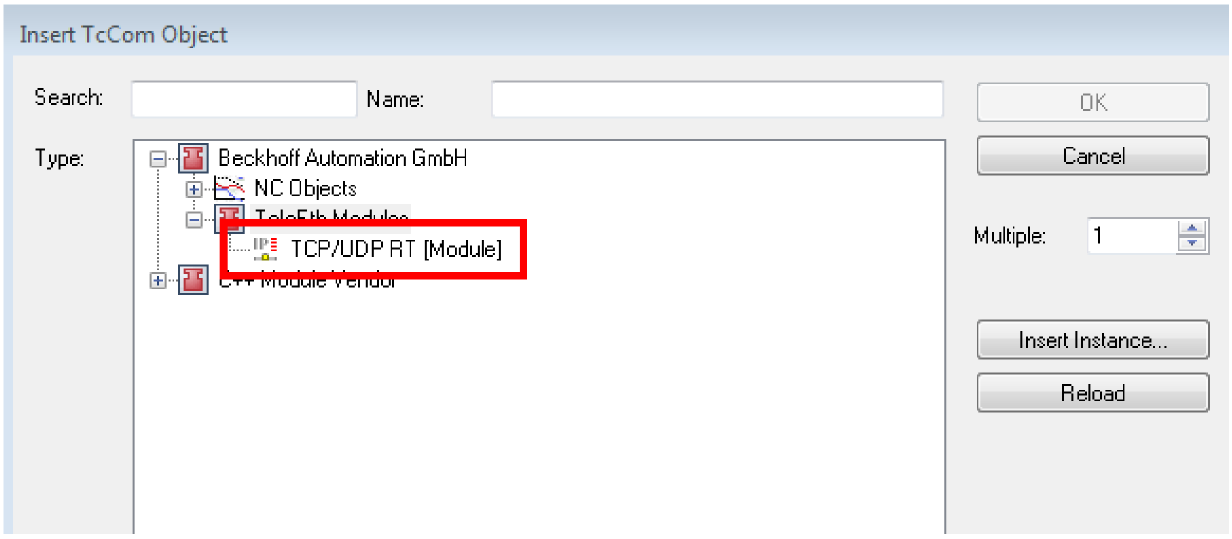
“TCP/UDP RT” module configuration

Notice **Variable names relating to TCP are used here. They have to be substituted accordingly.**

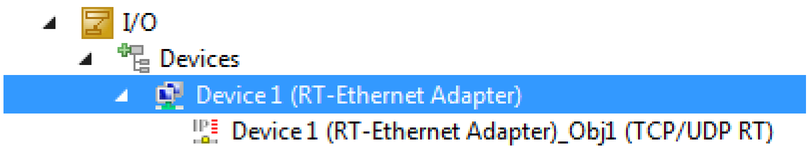
1. Create the “TCP/UDP RT” module under the RT Ethernet adapter by selecting “Add Object(s)…” in the context menu.



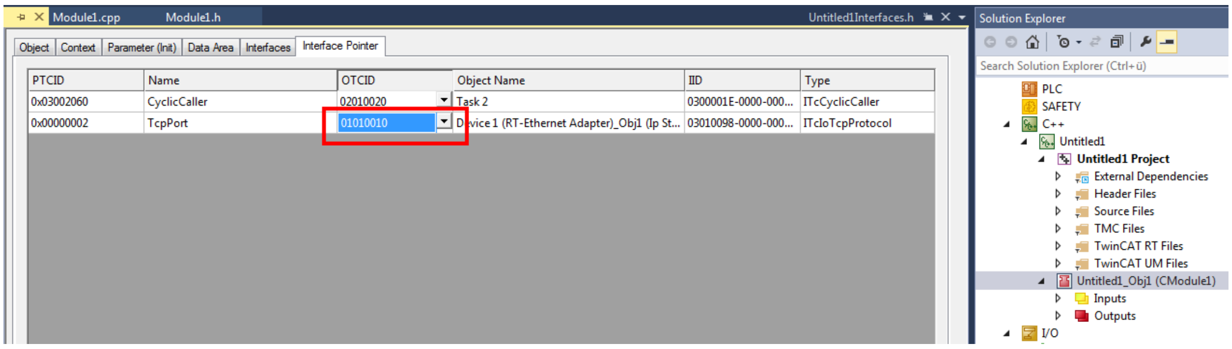
2. Then select the “TCP/UDP RT” module:



⇨ The TCP/UDP RT object is created under the adapter.

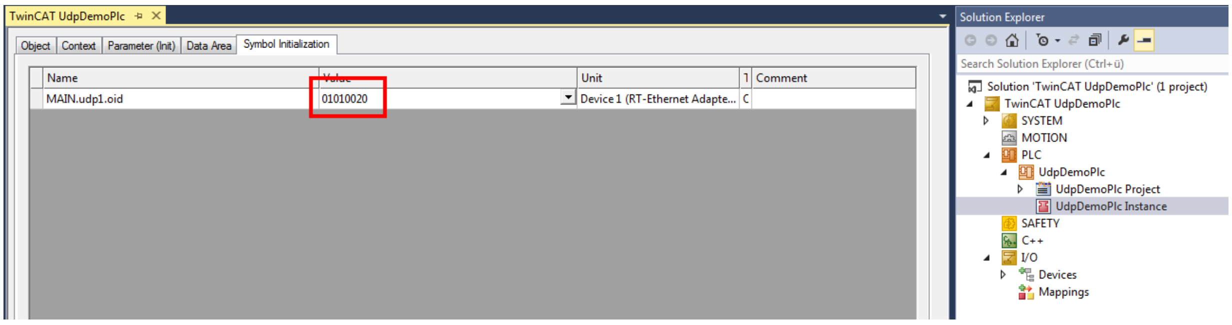

I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

3. Parameterize the previously created instance of the module (here: Module1) under “Interface Pointer” “TcpProt” with the OID of the created “TCP/UDP RT” object:



PTCID	Name	OTCID	Object Name	IID	Type
0x03002060	CyclicCaller	02010020	Task 2	0300001E-0000-000...	ITcCyclicCaller
0x00000002	TcpPort	01010010	Device 1 (RT-Ethernet Adapter)_Obj1 (Ip St...	03010098-0000-000...	ITcIoTcpProtocol

4. For PLC projects this configuration is also done at the instance, under the tab “Symbol Initialization”:



Name	Value	Unit	T	Comment
MAIN.udp1.oid	01010020	Device 1 (RT-Ethernet Adapte...	C	

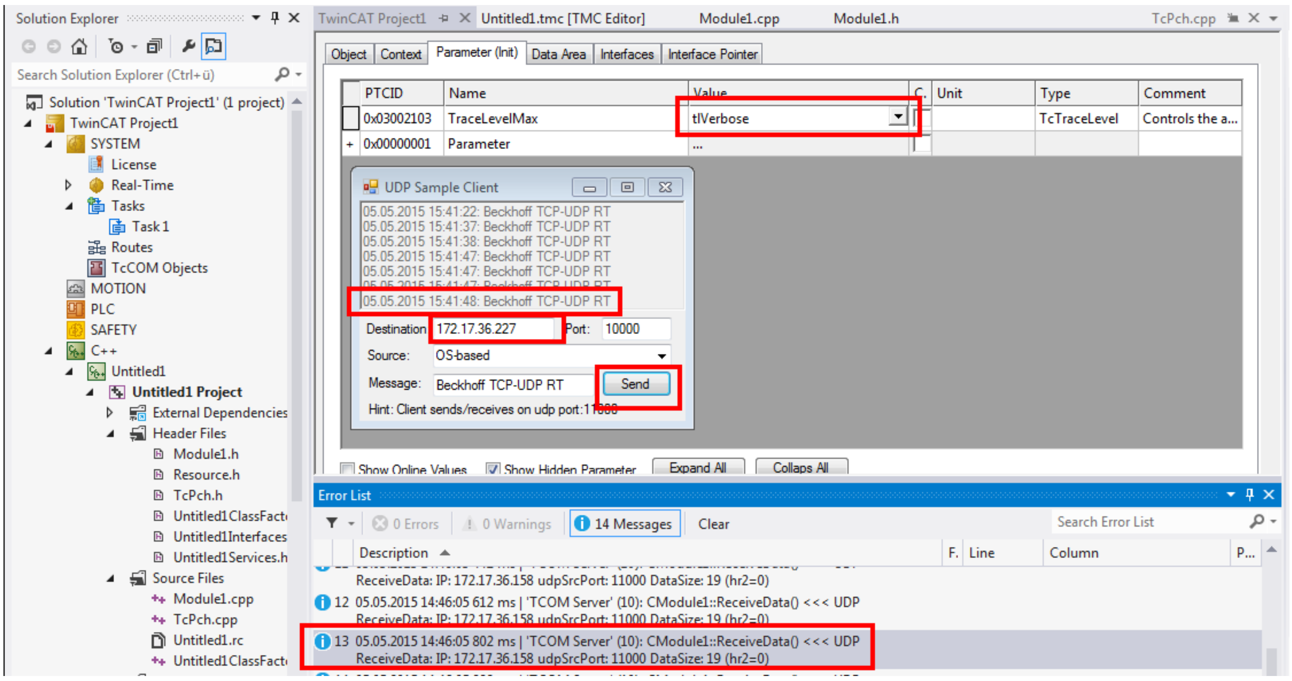
⇨ The configuration is thus completed

Disconnection by the operating system in Promiscuous mode

If Promiscuous mode is active at the RT Ethernet adapter in the “Adapter” tab, any TCP connection attempts are blocked by the operating system, since it does not recognize a port opened in the TCP/UDP RT object.

Testing

Once the configuration has been enabled, a UDP packet can be sent to the C++ module via the UDP Sample Client [▶ 44]. By activating the corresponding TraceLevel (here at least tlInfo; see C++ Tracing), an output can be generated in the Visual Studio log. The client displays the returned packets at the top.



	PTCID	Name	Value	C.	Unit	Type	Comment
	0x03002103	TraceLevelMax	tlVerbose			TcTraceLevel	Controls the a...
+	0x00000001	Parameter	...				

```
05.05.2015 15:41:22: Beckhoff TCP-UDP RT
05.05.2015 15:41:37: Beckhoff TCP-UDP RT
05.05.2015 15:41:38: Beckhoff TCP-UDP RT
05.05.2015 15:41:47: Beckhoff TCP-UDP RT
05.05.2015 15:41:47: Beckhoff TCP-UDP RT
05.05.2015 15:41:48: Beckhoff TCP-UDP RT
```

Destination: 172.17.36.227 Port: 10000
Source: OS-based
Message: Beckhoff TCP-UDP RT
Hint: Client sends/receives on udp port:11000

```
ReceiveData: IP: 172.17.36.158 udpSrcPort: 11000 DataSize: 19 (hr2=0)
12 05.05.2015 14:46:05 612 ms | 'TCOM Server' (10): CModule1::ReceiveData() <<< UDP
ReceiveData: IP: 172.17.36.158 udpSrcPort: 11000 DataSize: 19 (hr2=0)
13 05.05.2015 14:46:05 802 ms | 'TCOM Server' (10): CModule1::ReceiveData() <<< UDP
ReceiveData: IP: 172.17.36.158 udpSrcPort: 11000 DataSize: 19 (hr2=0)
```

No local communication

The UDP sample client must run on a different computer than the PLC with the TCP/UDP RT object, because no local communication between the Windows operating system and the real-time is available.
Alternatively, a so-called "loop cable" can be used to connect two network ports. The UDP sample client can be forced to use a specific port by selecting the source (dropdown menu "Source").

4.3 Quick Start (C++ / TCP Client)

This Quick Start shows the implementation of a TCP client as a TwinCAT 3 C++ project.

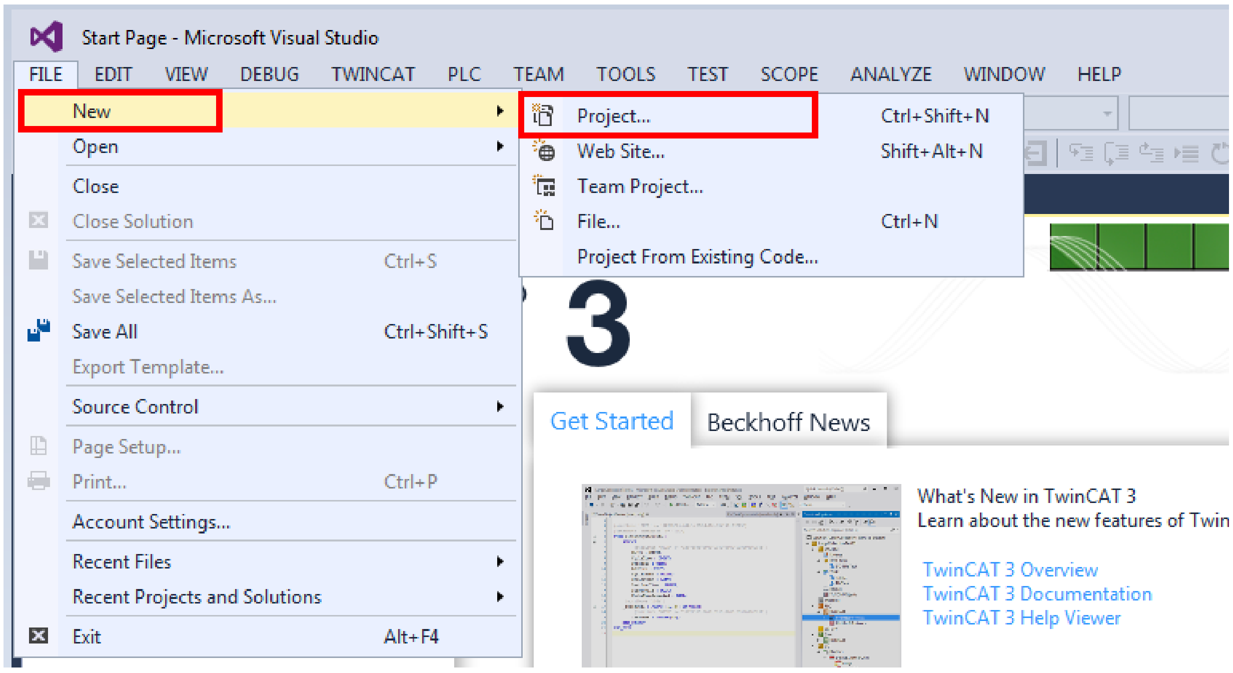
The engineering system must meet the requirements for TwinCAT 3 C++.

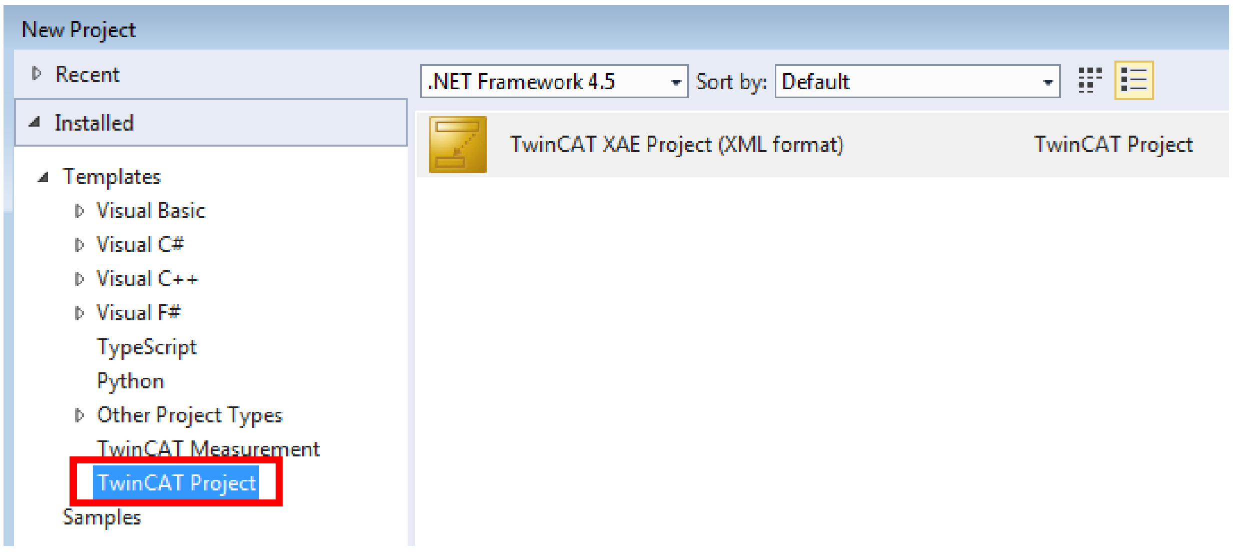
The example is also available for download under Sample 01 [▶ 38].

Creating a TwinCAT C++ project

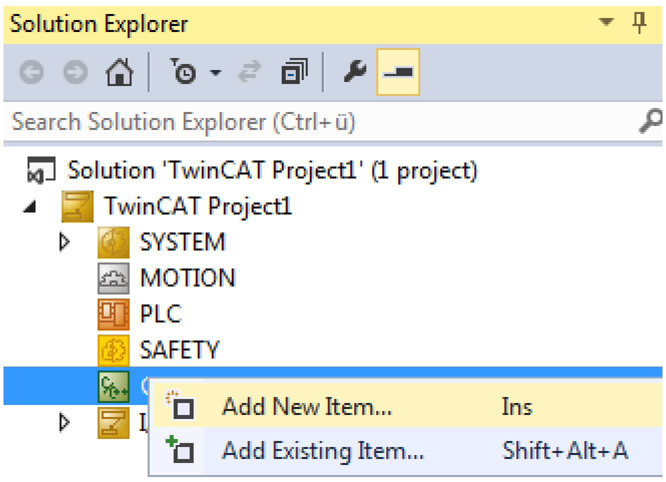
In this step, a new TwinCAT 3 C++ project is created.

1. Create a new TwinCAT project

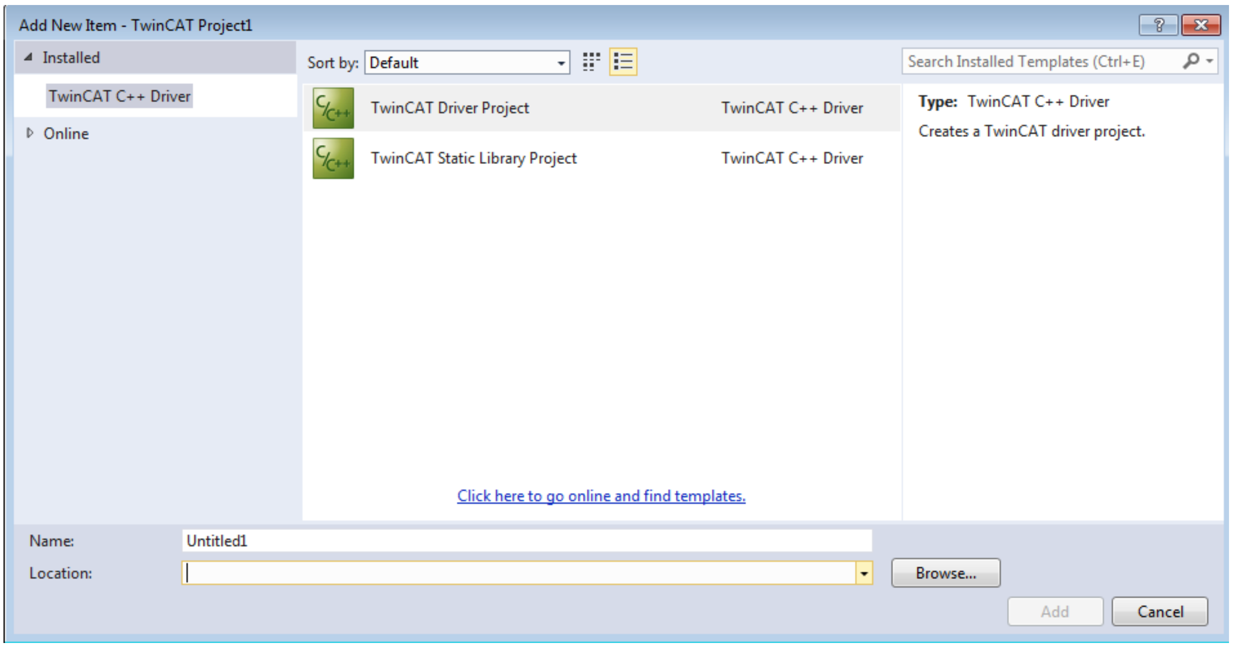




2. Add a TwinCAT C++ project


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Project1' (1 project)
TwinCAT Project1
SYSTEM
MOTION
PLC
SAFETY
Add New Item... Ins
Add Existing Item... Shift+Alt+A

3. Select a Driver project


Add New Item - TwinCAT Project1

Installed: TwinCAT C++ Driver; Online

Sort by: Default

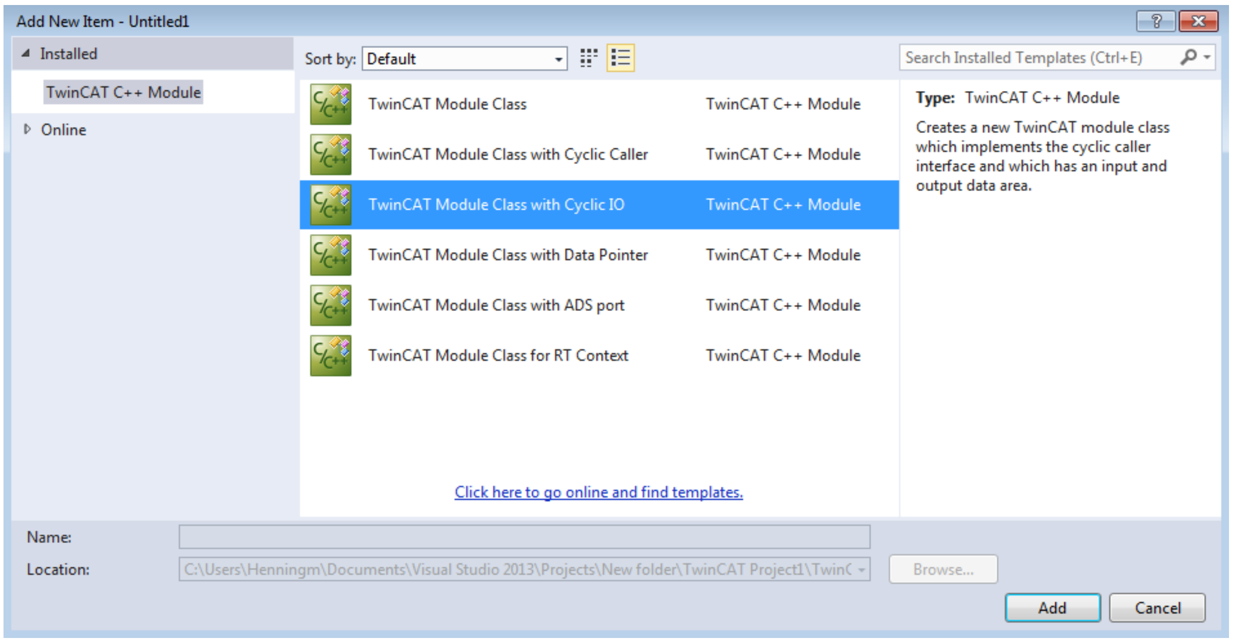
TwinCAT Driver Project	TwinCAT C++ Driver
TwinCAT Static Library Project	TwinCAT C++ Driver

Type: TwinCAT C++ Driver
Creates a TwinCAT driver project.

Click here to go online and find templates.

Name: Untitled1
Location:

4. Use the wizard for a module class with "Cyclic IO" as the basis for the TCP client.


Add New Item - Untitled1

Installed: TwinCAT C++ Module; Online

Sort by: Default

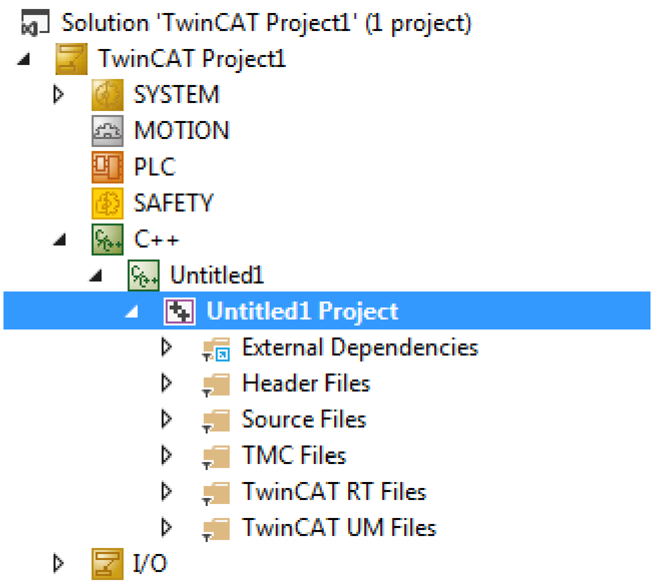
TwinCAT Module Class	TwinCAT C++ Module
TwinCAT Module Class with Cyclic Caller	TwinCAT C++ Module
TwinCAT Module Class with Cyclic IO	TwinCAT C++ Module
TwinCAT Module Class with Data Pointer	TwinCAT C++ Module
TwinCAT Module Class with ADS port	TwinCAT C++ Module
TwinCAT Module Class for RT Context	TwinCAT C++ Module

Type: TwinCAT C++ Module
Creates a new TwinCAT module class which implements the cyclic caller interface and which has an input and output data area.

Click here to go online and find templates.

Name:
Location: C:\Users\Henningm\Documents\Visual Studio 2013\Projects\New folder\TwinCAT Project1\TwinC

⇨ The result is a complete TwinCAT C++ project.

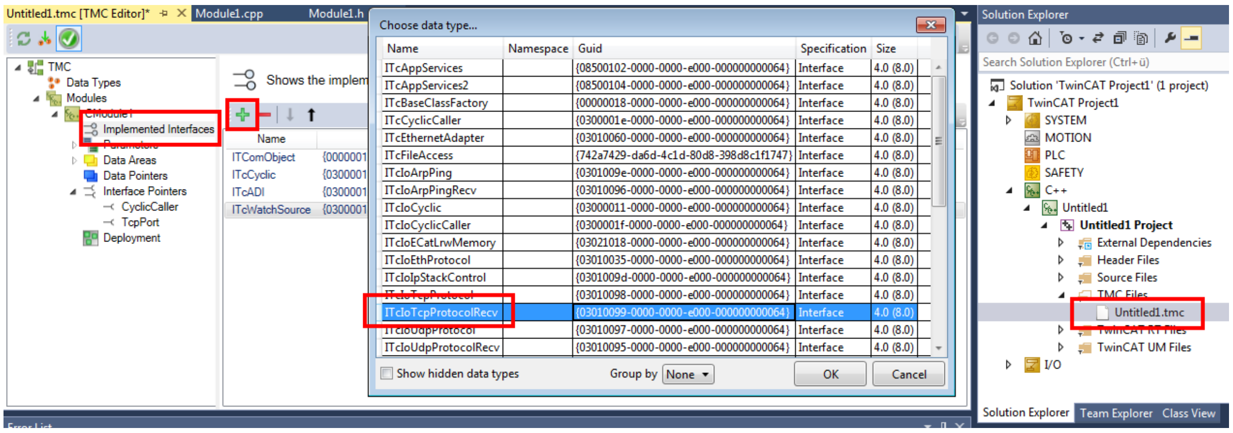


Solution 'TwinCAT Project1' (1 project)
- TwinCAT Project1
  - SYSTEM
  - MOTION
  - PLC
  - SAFETY
  - C++
    - Untitled1
      - Untitled1 Project
        - External Dependencies
        - Header Files
        - Source Files
        - TMC Files
        - TwinCAT RT Files
        - TwinCAT UM Files
  - I/O

TMC editor for creating interfaces, pointers and parameters

After creating the project, the next step involves implementation of the C++ TCP client.

1. The module created by the wizard must implement the interface "ITcIoTcpProtocolRecv". Open the TMC editor by double-clicking on the TMC file for the project. Add the interface to the module under "Implemented Interfaces".



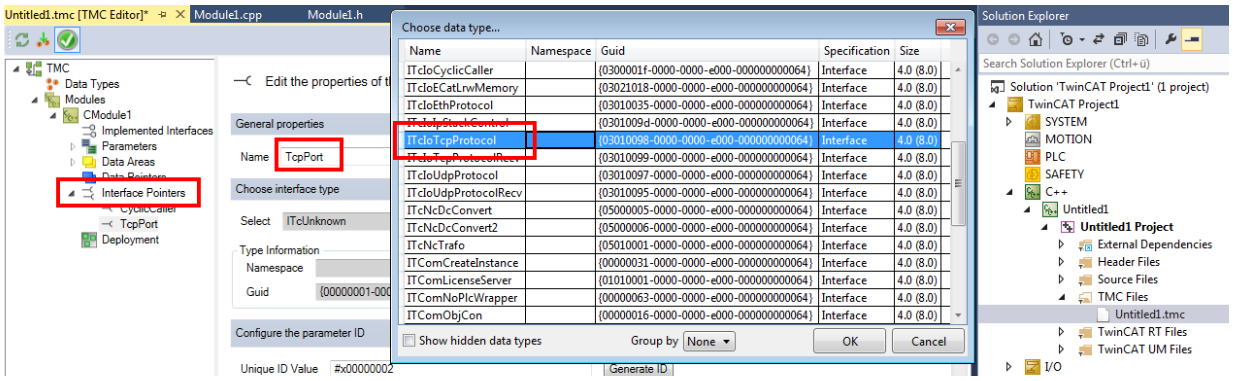
Choose data type...

Name	Namespace	Guid	Specification	Size
ITcAppServices		{08500102-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcAppServices2		{08500104-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcBaseClassFactory		{00000018-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcCyclicCaller		{0300001e-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcEthernetAdapter		{03010060-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcFileAccess		{742a7429-da6d-4c1d-80d8-398d8c1f1747}	Interface	4.0 (8.0)
ITcIoArpPing		{0301009e-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoArpPingRecv		{03010096-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoCyclic		{03000011-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoCyclicCaller		{0300001f-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoECatLrwMemory		{03021018-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoEthProtocol		{03010035-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoIpStackControl		{0301009d-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocol		{03010098-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocolRecv		{03010099-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocol		{03010097-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocolRecv		{03010095-0000-0000-e000-000000000064}	Interface	4.0 (8.0)

☐ Show hidden data types Group by None OK Cancel

Under "Implemented Interfaces" open a selection of the available interfaces by clicking on the "+" button. Select "ITcIoTcpProtocolRecv".

2. In addition, an "ITcIOTcpProtocol" interface pointer is required.

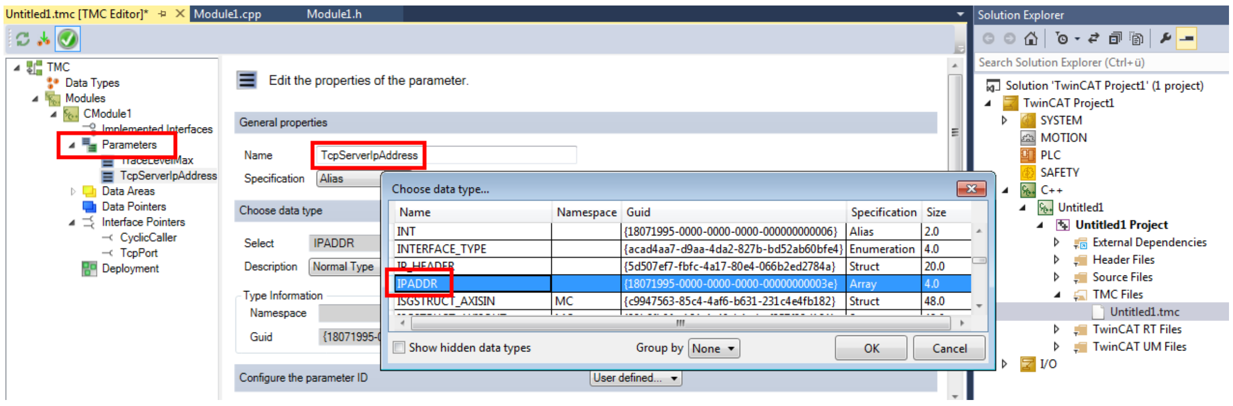


Choose data type...

Name	Namespace	Guid	Specification	Size
ITcIoCyclicCaller		{0300001f-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoECatLrwMemory		{03021018-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoEthProtocol		{03010035-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoIpStackControl		{0301009d-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocol		{03010098-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoTcpProtocolRecv		{03010099-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocol		{03010097-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcIoUdpProtocolRecv		{03010095-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcDcConvert		{05000005-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcDcConvert2		{05000006-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITcNcTrafo		{05010001-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITComCreateInstance		{00000031-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITComLicenseServer		{01010001-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITComNoPlcWrapper		{00000063-0000-0000-e000-000000000064}	Interface	4.0 (8.0)
ITComObjCon		{00000016-0000-0000-e000-000000000064}	Interface	4.0 (8.0)

☐ Show hidden data types Group by None OK Cancel

3. By creating a parameter the server IP address to be contacted and the port become configurable.



Edit the properties of the parameter.

General properties

Name: TcpServerIpAddress

Specification: Alias

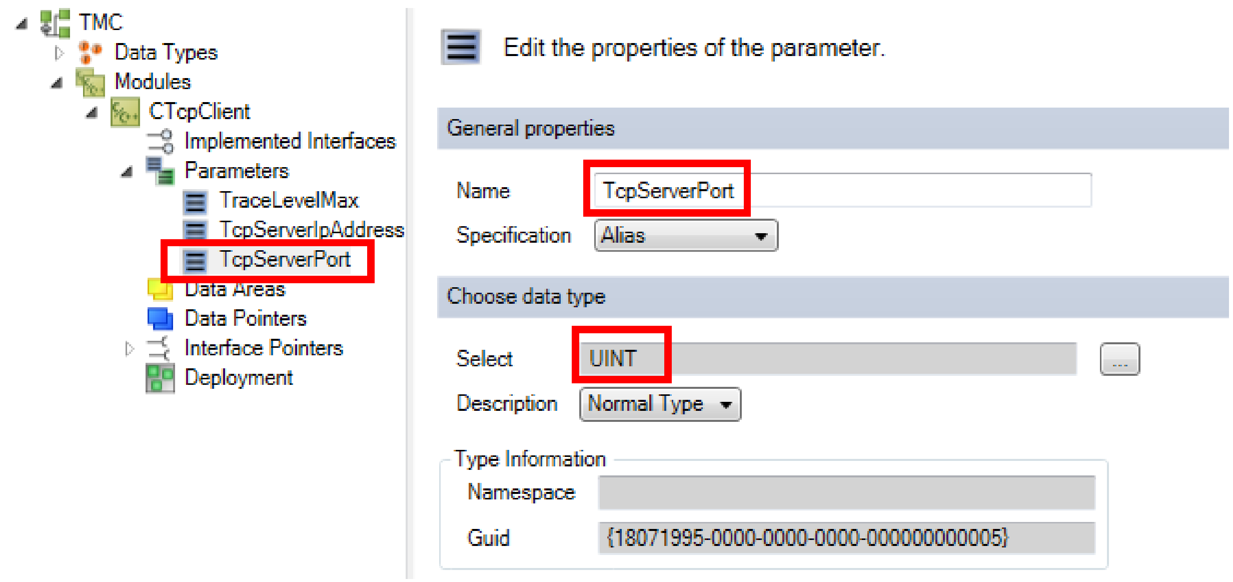
Choose data type

Select: IPADDR

Description: Normal Type

Choose data type...

Name	Namespace	Guid	Specification	Size
INT		{18071995-0000-0000-0000-000000000006}	Alias	2.0
INTERFACE_TYPE		{acad4aa7-d9aa-4da2-827b-bd52ab60bfe4}	Enumeration	4.0
IP_HEADER		{5d507ef7-fbfc-4a17-80e4-066b2ed2784a}	Struct	20.0
IPADDR		{18071995-0000-0000-0000-00000000003e}	Array	4.0
ISGSTRUCT_AXISIN	MC	{c9947563-85c4-4af6-b631-231c4e4fb182}	Struct	48.0



Edit the properties of the parameter.

General properties

Name: TcpServerPort

Specification: Alias

Choose data type

Select: UINT

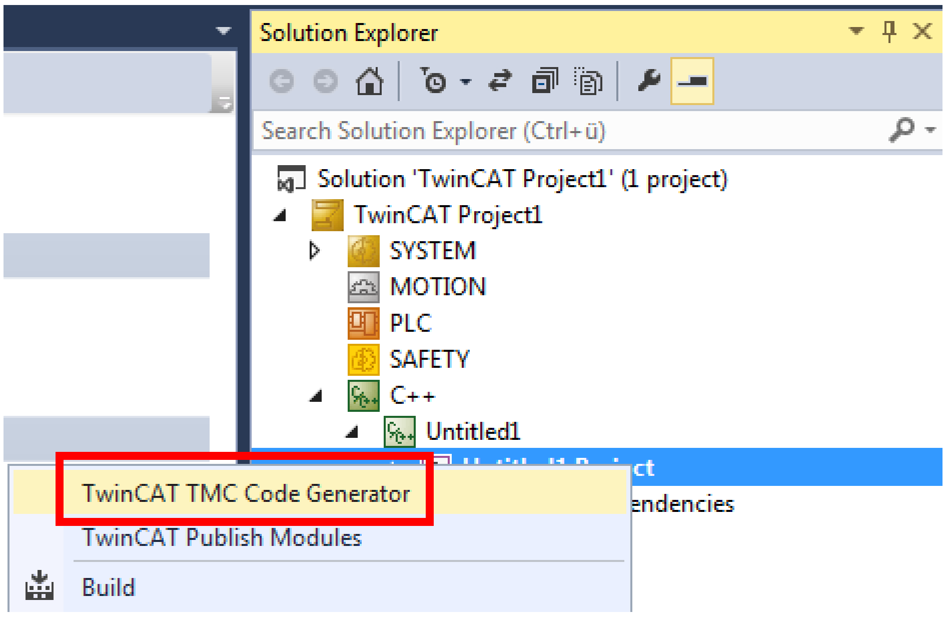
Description: Normal Type

Type Information

Namespace:

Guid: {18071995-0000-0000-0000-000000000005}

4. Now use the TMC code generator to prepare the code of the C++ module.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Project1' (1 project)
TwinCAT Project1
SYSTEM
MOTION
PLC
SAFETY
C++
Untitled1
TwinCAT TMC Code Generator
TwinCAT Publish Modules
Build

Start the TMC code generator by selecting the appropriate menu item in the context menu (right-click) of the C++ project.

⇨ All steps in the TMC editor are now completed.

Implement TCP client

1. Create two member variables in the module header file (here: Modul1.h).

```
ULONG     m_SockId;
BOOL m_bSendRequest;  //set by debugger for sending a http command
ULONG m_connections;  //count number of connection attempts
HRESULT m_hrSend;     //Last hr of SendData
```

2. These are initialized in the Constructor (Module1.cpp).

```
CModule1::CModule1()
    : m_Trace(m_TraceLevelMax, m_spSrv)
    , m_TraceLevelMax(tlAlways)
    , m_hrSend(0)
{
    m_SockId = 0; //added
    m_bSendRequest = true; //added
    m_connections = 0; //added
}
```

3. The interface pointer m_spTcpProt is now initialized in the Transition SO (i.e. in method SetObjStateSO).

```
HRESULT CTcpClient::SetObjStateSO()
{
    m_Trace.Log(tlVerbose, FENTERA);
    RESULT hr = S_OK;
    if (SUCCEEDED(hr) && m_spTcpProt.HasOID())     //added
    {                                             //added
        hr = m_spSrv->TcQuerySmartObjectInterface(m_spTcpProt); //added
    }                                             //added
    hr = FAILED(hr) ? hr : AddModuleToCaller();
```

4. In the Transition OS (i.e. method SetObjStateOS) a connection that may exist is closed, and the socket is released.

```
///////////////////////////////////////////////////////////////////////////////
// State transition from OP to SAFEOP
HRESULT CTcpClient::SetObjStateOS()
{
    //start added code
    m_Trace.Log(tlVerbose, FENTERA);
    HRESULT hr = S_OK;

    if ( m_SockId != 0 )
    {
        if (m_spTcpProt->IsConnected(m_SockId) == S_OK)
        {
            m_spTcpProt->Close(m_SockId);
            m_spTcpProt->CheckReceived();
        }
     m_spTcpProt->FreeSocket(m_SockId);
     m_SockId = 0;
    }

    RemoveModuleFromCaller();

    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
    //end added code
}
```

5. The actual process is implemented in the "CycleUpdate" method, which is called cyclically. Establishes a TCP connection to a server (address is provided in parameters "m_TcpServerIpAddress" and "m_TcpServerPort"). The connection handle is stored in the member variable "m_SockId". The connection is used to issue a simple http GET request.

```
HRESULT CTcpClient::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;
    //start added code
    if ( m_SockId == 0 )
    {
        if (SUCCEEDED_DBG(hr = m_spTcpProt->AllocSocket(THIS_CAST(ITcIoTcpProtocolRecv),
m_SockId)))
        {
            if (FAILED(hr = m_spTcpProt->Connect(m_SockId, ((PULONG)&m_TcpServerIpAddress)[0],
m_TcpServerPort)))
            {
                m_spTcpProt->FreeSocket(m_SockId);
                m_SockId = 0;
            }
else {
                m_connections++; //count number of connections
            }
        }
    }
    else
    {
    if ( m_bSendRequest && m_spTcpProt->IsConnected(m_SockId) == S_OK )
    {
        PCHAR pRequest = "GET / HTTP/1.1\r\nHOST: beckhoff.com\r\n\r\n ";
        ULONG nSendData = 0;
        m_hrSend = m_spTcpProt->SendData(m_SockId, strlen(pRequest), pRequest, nSendData);
        m_bSendRequest = false;
    }
    }

    m_spTcpProt->CheckReceived();

    //end added code
    return hr;
}
```

6. The module implements the interface "ITcIoTcpProtocolRecv", as a result of which the TMC code generator created a "ReceiveEvent" method. This is called when an event is received and must therefore be able to deal with a wide range of event types.

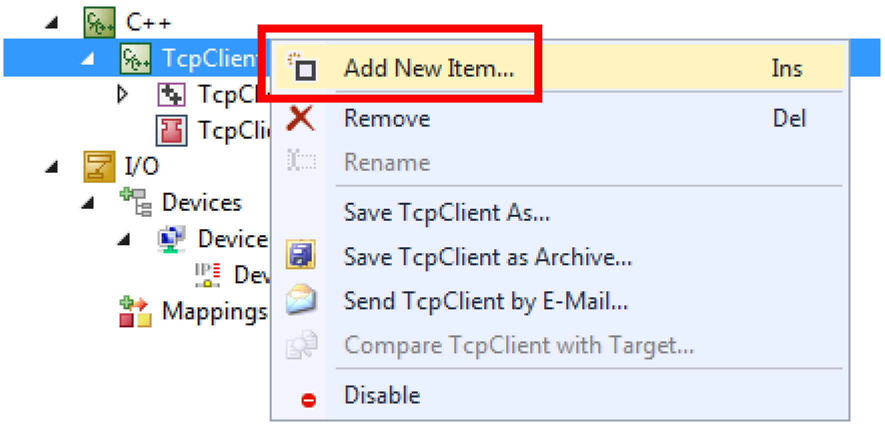
```
HRESULT CTcpClient::ReceiveEvent(ULONG socketId, TCPIP_EVENT tcpEvent)
{
//start added code
m_Trace.Log(tlInfo, FLEAVEA "Receive TCP Event: SocketId: %d Event: %d \n", socketId, tcpEvent);

    switch (tcpEvent)
    {
    case TCPIP_EVENT_ERROR:
    case TCPIP_EVENT_RESET:
    case TCPIP_EVENT_TIMEOUT:
    m_Trace.Log(tlInfo, FLEAVEA "Connection to remote server failed!\n");
           m_SockId = 0;
        break;
    case TCPIP_EVENT_CONN_CLOSED:
        m_Trace.Log(tlInfo, FLEAVEA "Close connection: SocketId: %d \n", socketId);
          m_SockId = 0;
        break;
    case TCPIP_EVENT_CONN_INCOMING:
    case TCPIP_EVENT_KEEP_ALIVE:
    case TCPIP_EVENT_CONN_IDLE:
    case TCPIP_EVENT_DATA_SENT:
    case TCPIP_EVENT_DATA_RECEIVED:
        break;
    default:
        break;
    }
    return S_OK;
    //end added code
}
```

7. Analogous to the "ReceiveEvent" method, a "ReceiveData" method was created from the "ITcIoTcpProtocolRecv" interface. It is responsible for receiving the data and is implemented as follows:

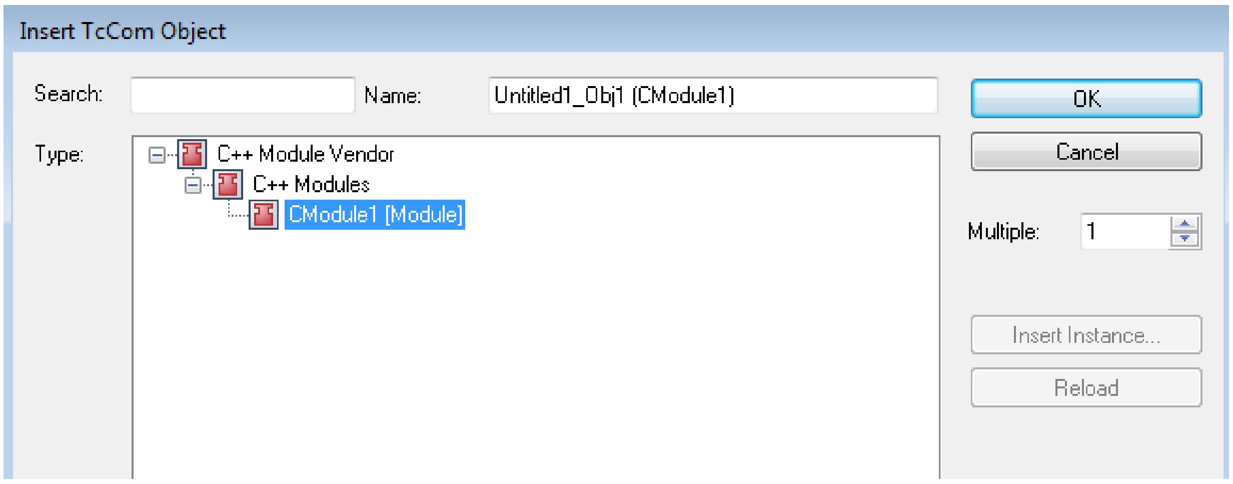
```
HRESULT CTcpClient::ReceiveData(ULONG socketId, ULONG nData, PVOID pData)
{
//start added code
    HRESULT hr = S_OK;
    PCHAR pResponse = new CHAR[100];
    memset(pResponse, 0, 100);
    memcpy(pResponse, pData, min(100, nData));
    m_Trace.Log(tlInfo, FLEAVEA "Receive answer w/ length %d : first 100 chars:'%s'", nData,
pResponse);
    return hr;
//end added code
}
```

8. The module is now ready and can be compiled. (Right-click on "Build" project).

9. An instance of the module is created:
   Right-click on the C++ project



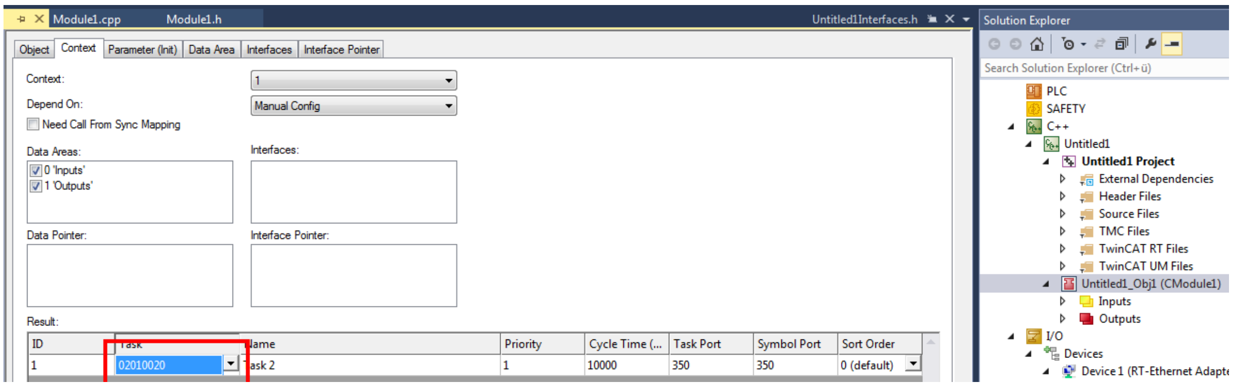
   and select the module



Search:
Name: Untitled1_Obj1 (CModule1)
Type:
- C++ Module Vendor
  - C++ Modules
    - CModule1 [Module]

Multiple: 1

⇨ The instance is associated with a task, so that the "CycleUpdate" method is called.



Context: 1
Depend On: Manual Config
☐ Need Call From Sync Mapping

Data Areas:
☑ 0 'Inputs'
☑ 1 'Outputs'

Interfaces:

Data Pointer:

Interface Pointer:

Result:

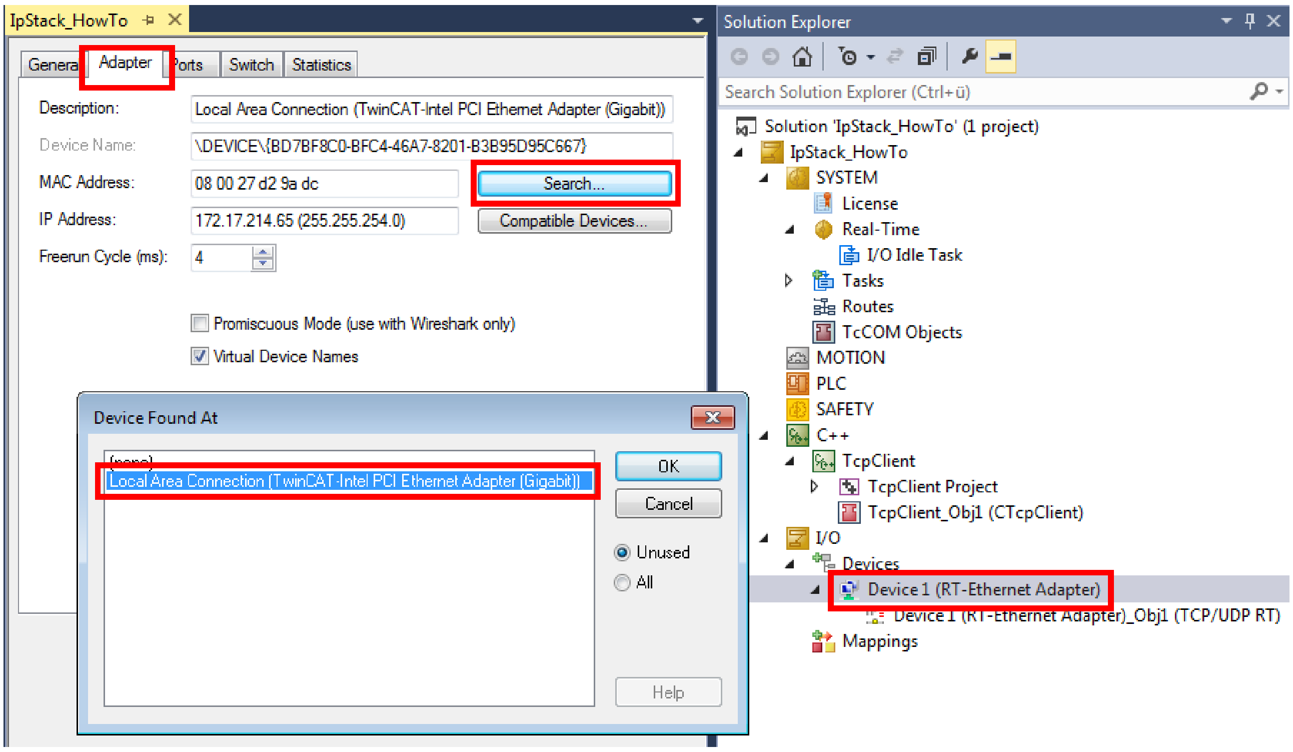
ID	Task	Name	Priority	Cycle Time (...	Task Port	Symbol Port	Sort Order
1	02010020	Task 2	1	10000	350	350	0 (default)

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

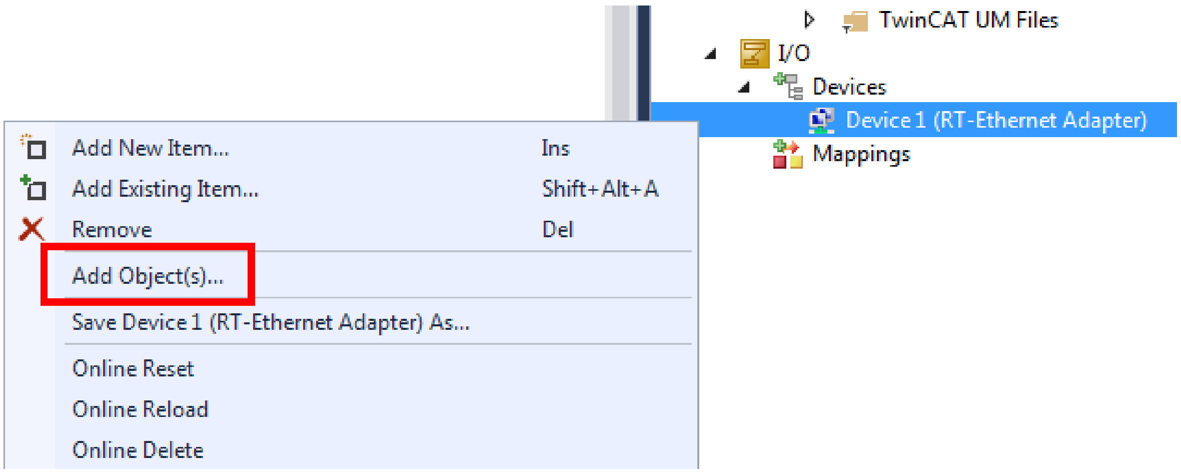
Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).


IpStack_HowTo
General
Adapter
Ports
Switch
Statistics
Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
Search...
IP Address: 172.17.214.65 (255.255.254.0)
Compatible Devices...
Freerun Cycle (ms): 4
Promiscuous Mode (use with Wireshark only)
Virtual Device Names
Device Found At
(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
OK
Cancel
Unused
All
Help
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'IpStack_HowTo' (1 project)
IpStack_HowTo
SYSTEM
License
Real-Time
I/O Idle Task
Tasks
Routes
TcCOM Objects
MOTION
PLC
SAFETY
C++
TcpClient
TcpClient Project
TcpClient_Obj1 (CTcpClient)
I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
Mappings

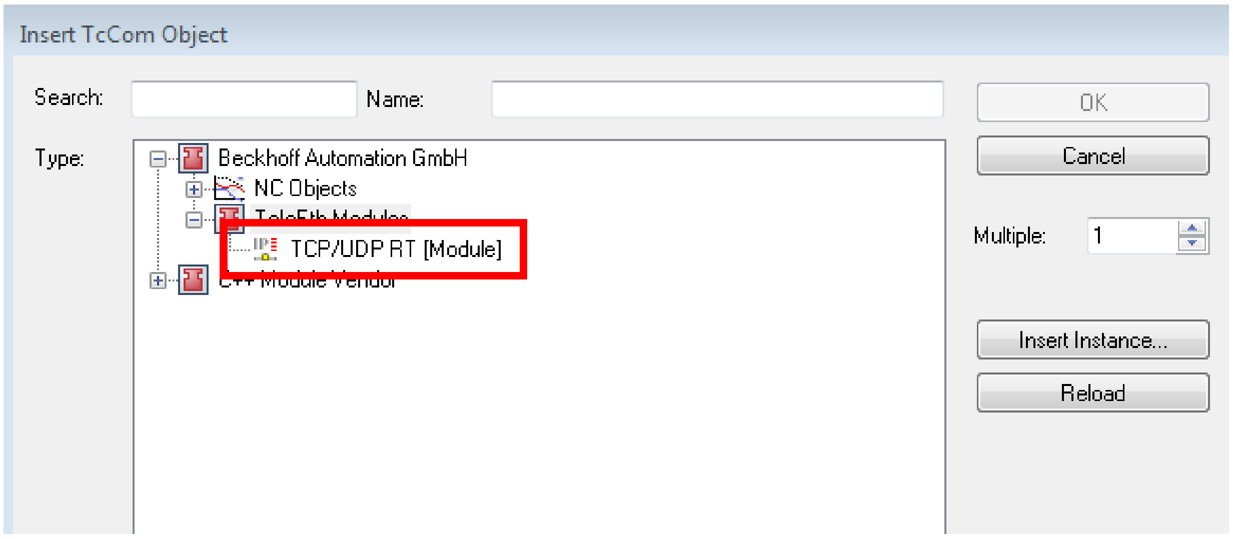
“TCP/UDP RT” module configuration

Notice **Variable names relating to TCP are used here. They have to be substituted accordingly.**

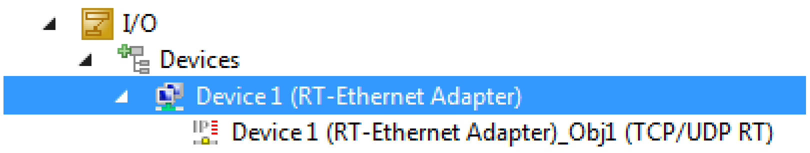
1. Create the “TCP/UDP RT” module under the RT Ethernet adapter by selecting “Add Object(s)…” in the context menu.



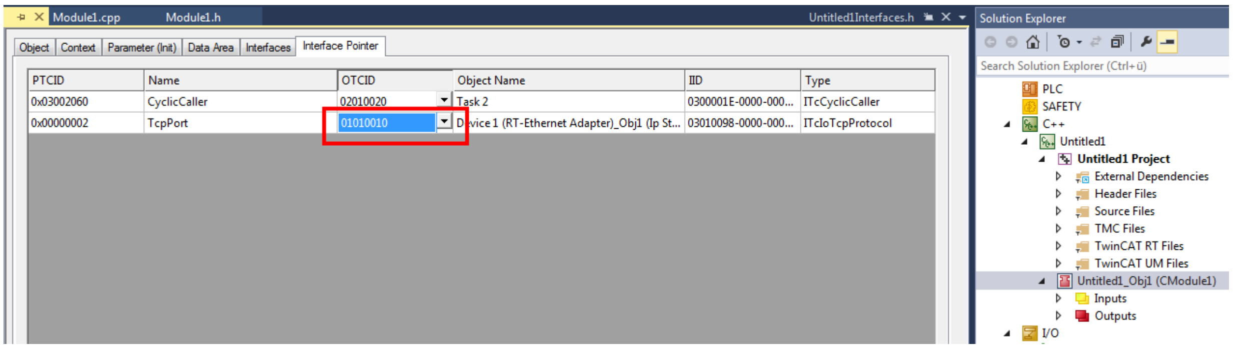
2. Then select the “TCP/UDP RT” module:



⇨ The TCP/UDP RT object is created under the adapter.

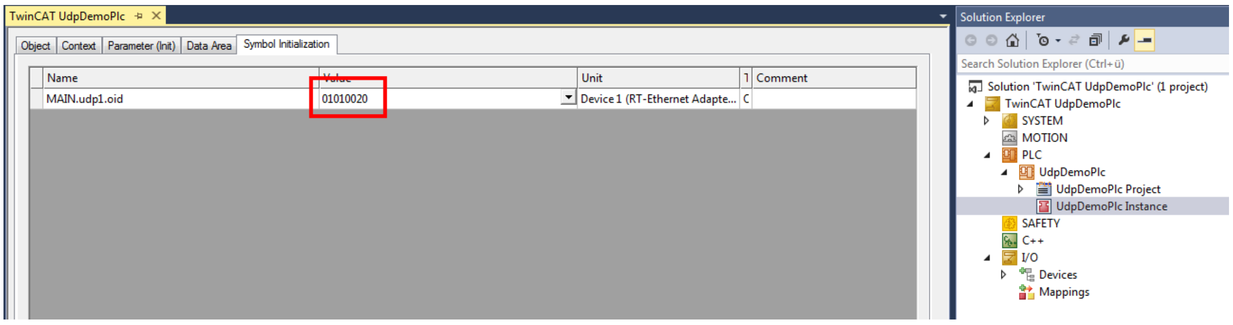

I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

3. Parameterize the previously created instance of the module (here: Module1) under “Interface Pointer” “TcpProt” with the OID of the created “TCP/UDP RT” object:



PTCID	Name	OTCID	Object Name	IID	Type
0x03002060	CyclicCaller	02010020	Task 2	0300001E-0000-000...	ITcCyclicCaller
0x00000002	TcpPort	01010010	Device 1 (RT-Ethernet Adapter)_Obj1 (Ip St...	03010098-0000-000...	ITcIoTcpProtocol

4. For PLC projects this configuration is also done at the instance, under the tab “Symbol Initialization”:



Name	Value	Unit	T	Comment
MAIN.udp1.oid	01010020	Device 1 (RT-Ethernet Adapte...	C	

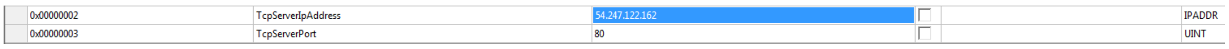
⇨ The configuration is thus completed

Disconnection by the operating system in Promiscuous mode

If Promiscuous mode is active at the RT Ethernet adapter in the “Adapter” tab, any TCP connection attempts are blocked by the operating system, since it does not recognize a port opened in the TCP/UDP RT object.

Handling

1. The sample is ready to use once you have configured both the TcpServerIpAddress and the TcpServerPort at the module instance:



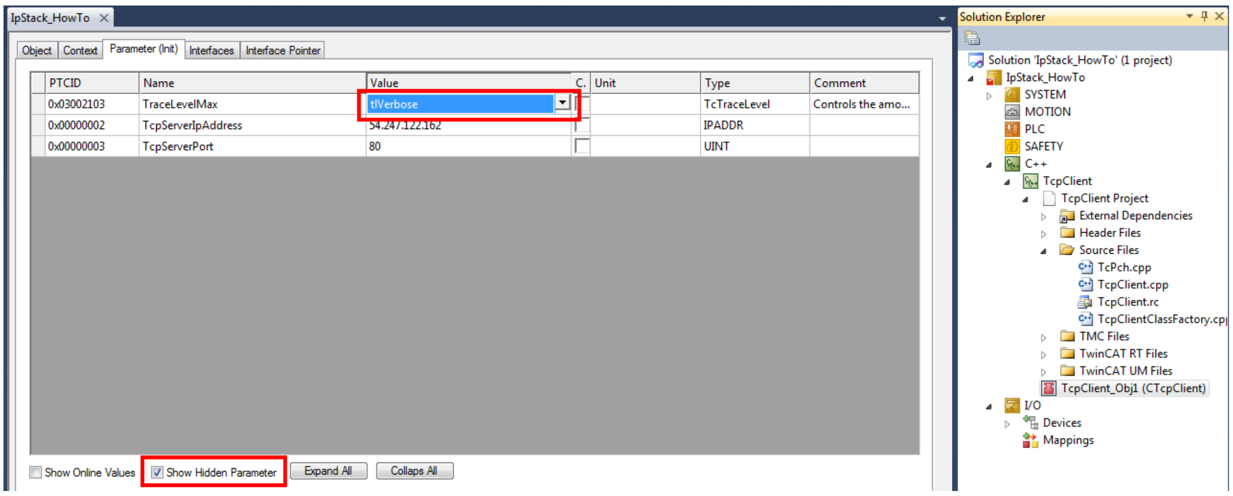
0x00000002	TcpServerIpAddress	54.247.122.162	☐		IPADDR
0x00000003	TcpServerPort	80	☐		UINT

Notice Possible source of error: A test web server 62.159.14.51 is queried in the sample. A corresponding HTTP command is stored in the source code. IP address, port, and this HTTP command may have to be adjusted.

2. After activating the configuration you can see log messages (see source code) and the first 100 bytes of the response from the server in the output:

```
MSG | 1/22/2015 3:14:14 PM 671 ms | 'TCOM Server' (10): CTcpClient::ReceiveEvent() <<< Receive TCP Event: SocketId: 1 Event: 10

MSG | 1/22/2015 3:14:14 PM 671 ms | 'TCOM Server' (10): CTcpClient::ReceiveEvent() <<< Receive TCP Event: SocketId: 1 Event: 8

MSG | 1/22/2015 3:14:14 PM 671 ms | 'TCOM Server' (10): CTcpClient::ReceiveEvent() <<< Receive TCP Event: SocketId: 1 Event: 10

MSG | 1/22/2015 3:14:14 PM 671 ms | 'TCOM Server' (10): CTcpClient::ReceiveData() <<< Receive answer w/ length 5642 : 'HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 5366
Content-Type: text/html
Server: Micr: So' |

MSG | 1/22/2015 3:14:15 PM 681 ms | 'TCOM Server' (10): CTcpClient::ReceiveEvent() <<< Receive TCP Event: SocketId: 1 Event: 7
```

3. To output these messages the “Tracelevel” can be configured (via Info):



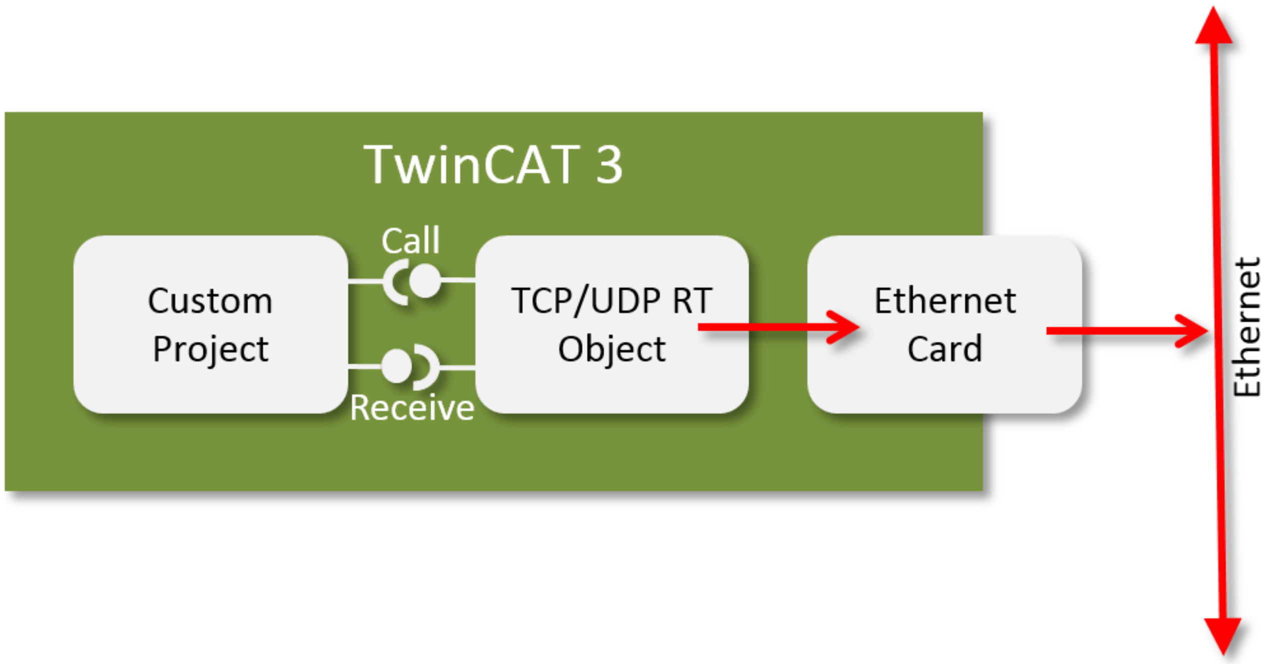
PTCID	Name	Value	C.	Unit	Type	Comment
0x03002103	TraceLevelMax	tlVerbose			TcTraceLevel	Controls the amo...
0x00000002	TcpServerIpAddress	54.247.122.162	☐		IPADDR	
0x00000003	TcpServerPort	80	☐		UINT	

The procedure is carried out once when the program starts.

A new request is sent if “m_bSendRequest” is set to TRUE (e.g. through TwinCAT Live Watch). The return of the SendData method is stored in hrSend. For the sample it can be monitored via the debugger.

5 Configuration

The integration and configuration of the "TCP/UDP RT" object is described here, starting from an existing TwinCAT project.



The "TCP/UDP RT" object is instantiated and configured. The configuration essentially consists of assigning the network card to be used.

Windows Firewall

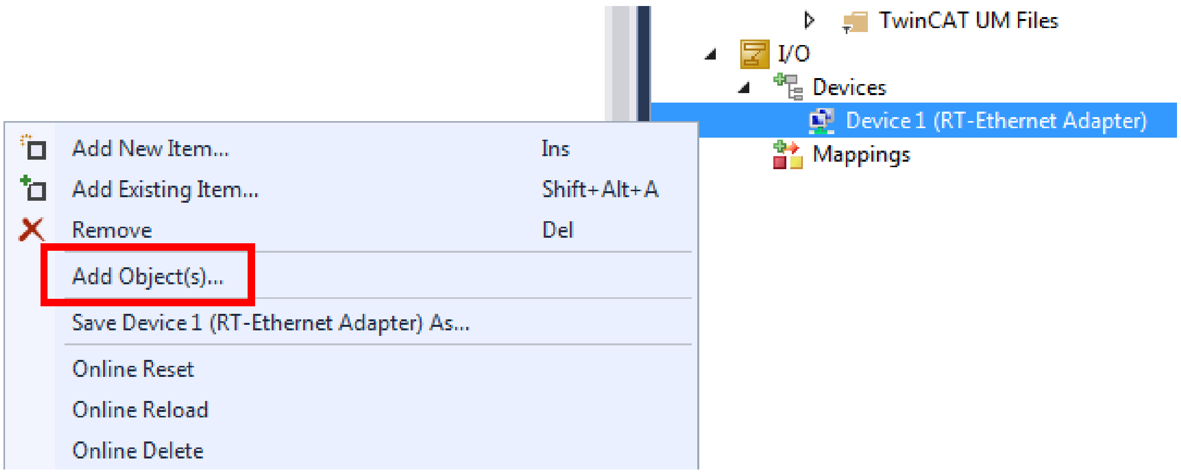
The Windows firewall cannot be used, since the TF6311 is directly integrated in the TwinCAT system.

The "TCP/UDP RT" object also contributes some parameters, which are documented here [▶ 55].

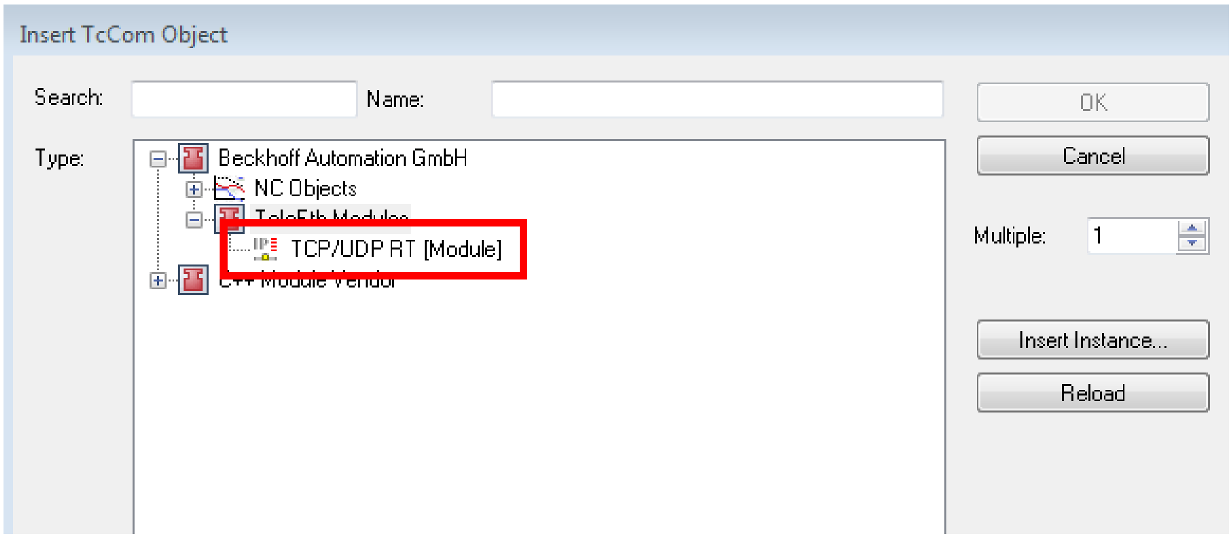
“TCP/UDP RT” module configuration

Notice **Variable names relating to TCP are used here. They have to be substituted accordingly.**

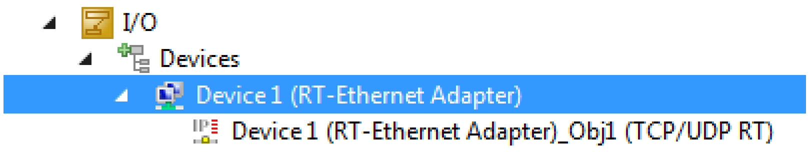
1. Create the “TCP/UDP RT” module under the RT Ethernet adapter by selecting “Add Object(s)…” in the context menu.



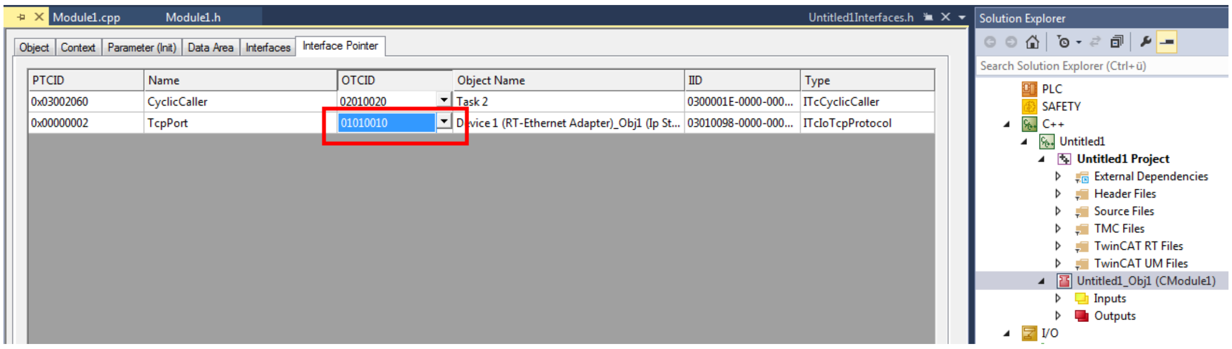
2. Then select the “TCP/UDP RT” module:



⇨ The TCP/UDP RT object is created under the adapter.

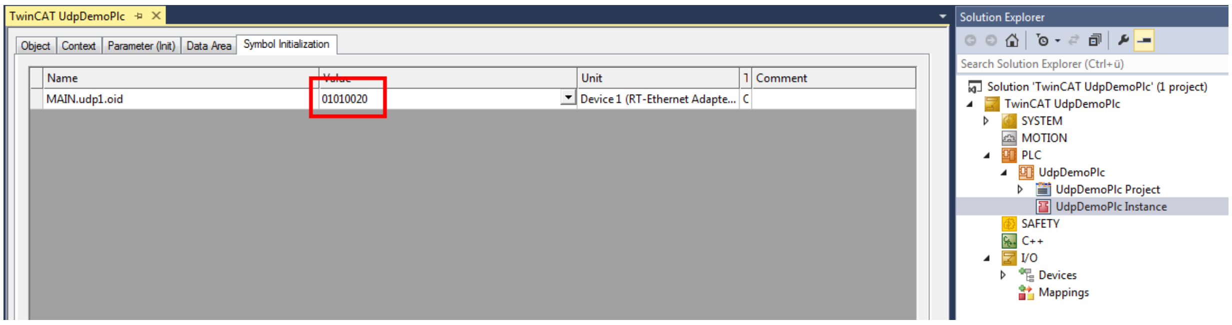

I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

3. Parameterize the previously created instance of the module (here: Module1) under “Interface Pointer” “TcpProt” with the OID of the created “TCP/UDP RT” object:



PTCID	Name	OTCID	Object Name	IID	Type
0x03002060	CyclicCaller	02010020	Task 2	0300001E-0000-000...	ITcCyclicCaller
0x00000002	TcpPort	01010010	Device 1 (RT-Ethernet Adapter)_Obj1 (Ip St...	03010098-0000-000...	ITcIoTcpProtocol

4. For PLC projects this configuration is also done at the instance, under the tab “Symbol Initialization”:



Name	Value	Unit	T	Comment
MAIN.udp1.oid	01010020	Device 1 (RT-Ethernet Adapte...	C	

⇨ The configuration is thus completed

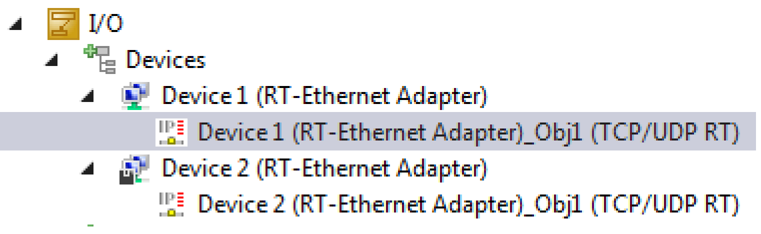
Disconnection by the operating system in Promiscuous mode

If Promiscuous mode is active at the RT Ethernet adapter in the “Adapter” tab, any TCP connection attempts are blocked by the operating system, since it does not recognize a port opened in the TCP/UDP RT object.

5.1 Multiple network cards

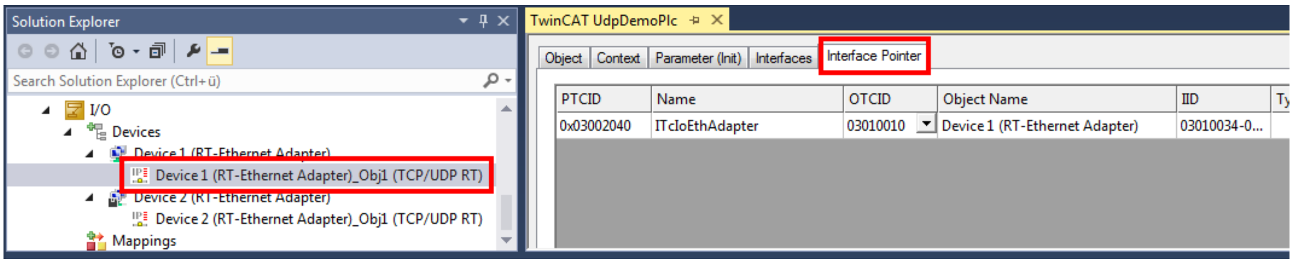
A TCP/UDP RT object is assigned to an RT Ethernet adapter by instantiating it under the objects, for example. A TCP/UDP RT object therefore always addresses precisely one network port of the controller via the RT Ethernet adapter.

If several network ports are to be used, a TCP/UDP RT object is created for each RT Ethernet adapter:



- I/O
  - Devices
    - Device 1 (RT-Ethernet Adapter)
      - Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
    - Device 2 (RT-Ethernet Adapter)
      - Device 2 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

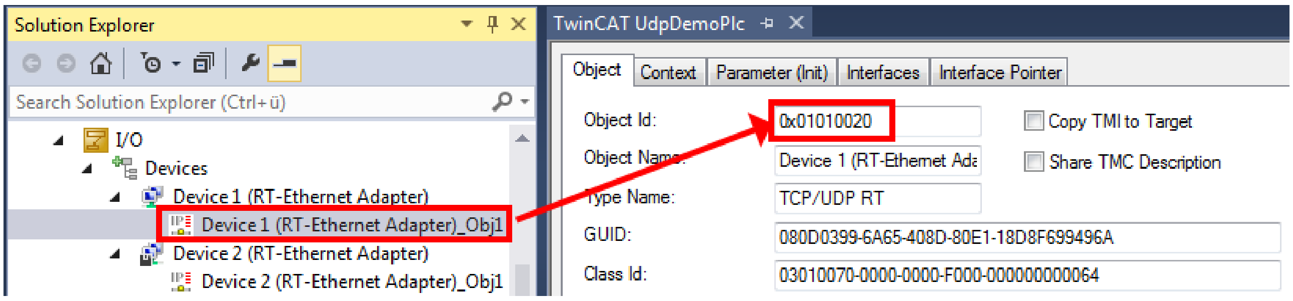
The TCP/UDP RT objects relate to the higher-level RT Ethernet adapter, if no other configuration was specified manually:



Solution Explorer
Search Solution Explorer (Ctrl+ü)
I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
Device 2 (RT-Ethernet Adapter)
Device 2 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
Mappings
TwinCAT UdpDemoPlc
Object
Context
Parameter (Init)
Interfaces
Interface Pointer

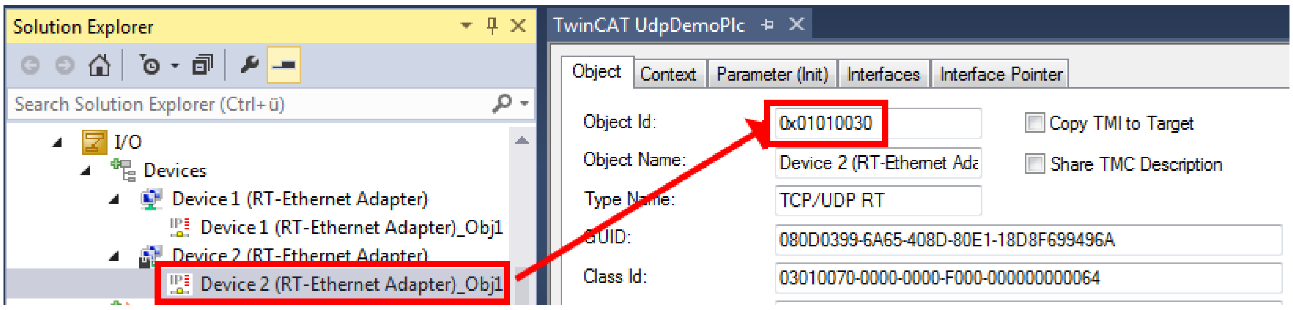
PTCID	Name	OTCID	Object Name	IID	Ty
0x03002040	ITcIoEthAdapter	03010010	Device 1 (RT-Ethernet Adapter)	03010034-0...	

These objects have different object IDs:



Solution Explorer
Search Solution Explorer (Ctrl+ü)
I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1
Device 2 (RT-Ethernet Adapter)
Device 2 (RT-Ethernet Adapter)_Obj1
TwinCAT UdpDemoPlc
Object
Context
Parameter (Init)
Interfaces
Interface Pointer
Copy TMI to Target
Share TMC Description

Object Id: 0x01010020
Object Name: Device 1 (RT-Ethernet Ada
Type Name: TCP/UDP RT
GUID: 080D0399-6A65-408D-80E1-18D8F699496A
Class Id: 03010070-0000-0000-F000-000000000064

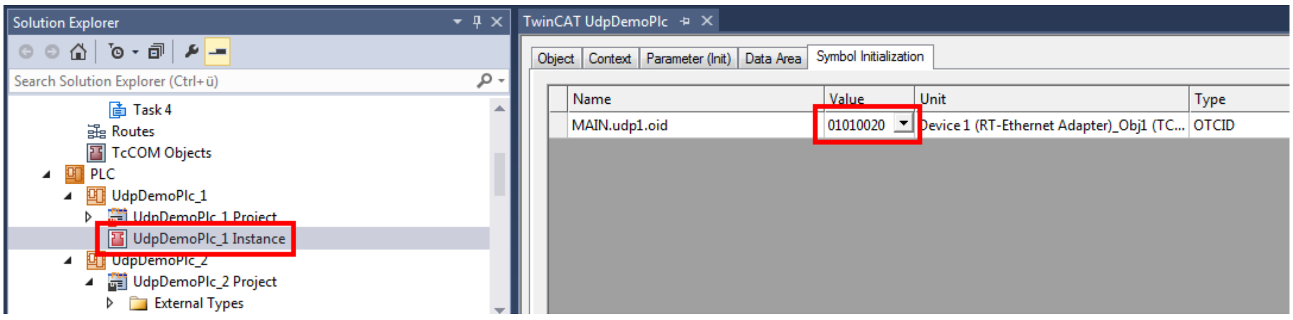


Solution Explorer
Search Solution Explorer (Ctrl+ü)
I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1
Device 2 (RT-Ethernet Adapter)
Device 2 (RT-Ethernet Adapter)_Obj1
TwinCAT UdpDemoPlc
Object
Context
Parameter (Init)
Interfaces
Interface Pointer
Copy TMI to Target
Share TMC Description

Object Id: 0x01010030
Object Name: Device 2 (RT-Ethernet Ada
Type Name: TCP/UDP RT
GUID: 080D0399-6A65-408D-80E1-18D8F699496A
Class Id: 03010070-0000-0000-F000-000000000064

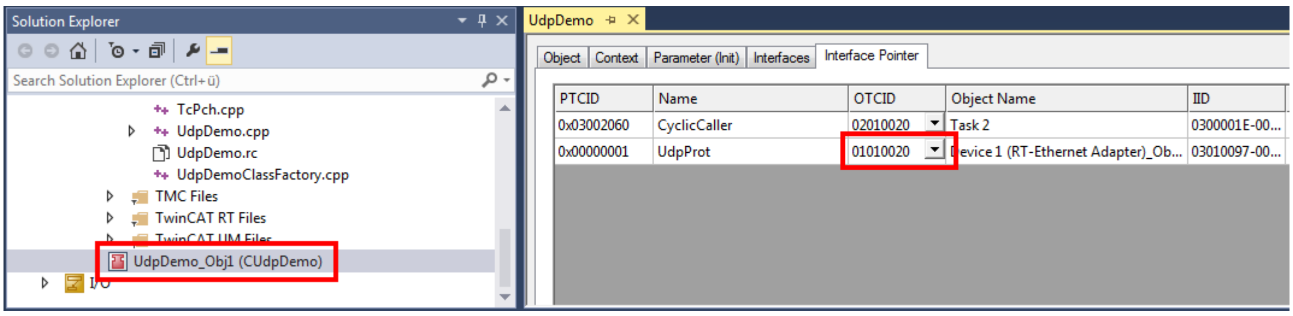
This object ID is used for referencing, as described above:

PLC:



Name	Value	Unit	Type
MAIN.udp1.oid	01010020	Device 1 (RT-Ethernet Adapter)_Obj1 (TC...	OTCID

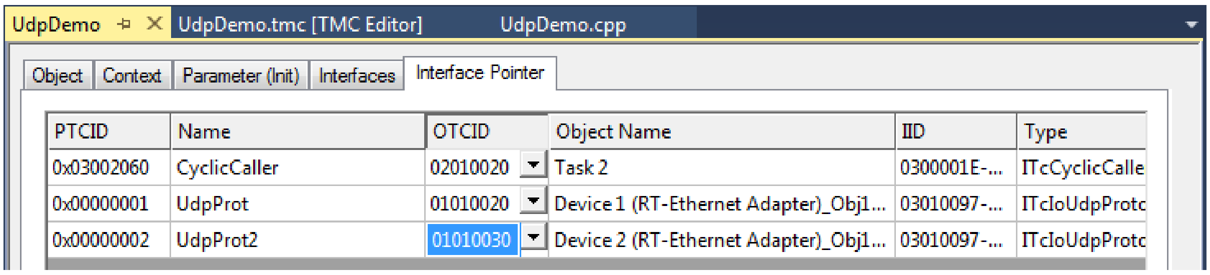
Or for a C++ module:



PTCID	Name	OTCID	Object Name	IID
0x03002060	CyclicCaller	02010020	Task 2	0300001E-00...
0x00000001	UdpProt	01010020	Device 1 (RT-Ethernet Adapter)_Ob...	03010097-00...

The use is highly dependent on the application. Some sample scenarios are provided below:

- A C++ module can be instantiated more than once. Each module can then communicate via a particular network card, based on the configuration with the corresponding object ID.
- Different PLC programs can be assigned separate TCP/UDP RT objects and thus act independently.
- A PLC or C++ program can address several TCP/UDP RT objects (and therefore several network cards), based on corresponding symbols (C++ is used as an example here):



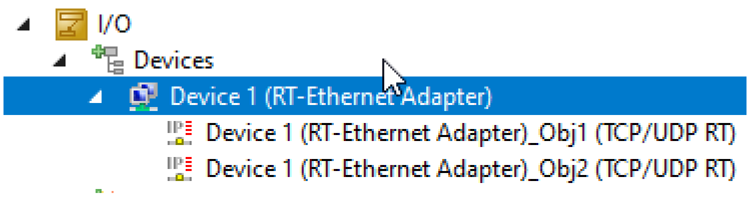
PTCID	Name	OTCID	Object Name	IID	Type
0x03002060	CyclicCaller	02010020	Task 2	0300001E-...	ITcCyclicCalle
0x00000001	UdpProt	01010020	Device 1 (RT-Ethernet Adapter)_Obj1...	03010097-...	ITcIoUdpProtc
0x00000002	UdpProt2	01010030	Device 2 (RT-Ethernet Adapter)_Obj1...	03010097-...	ITcIoUdpProtc

Object management must be implemented to suit the application. For example, the CheckReceived() calls must be applied to all objects. This also applies to calls for SendData() / RegisterReceiver() etc.

5.2 Multitask access to a network card

If a network card is to be used from several real-time contexts (tasks), it must be implemented as described here.

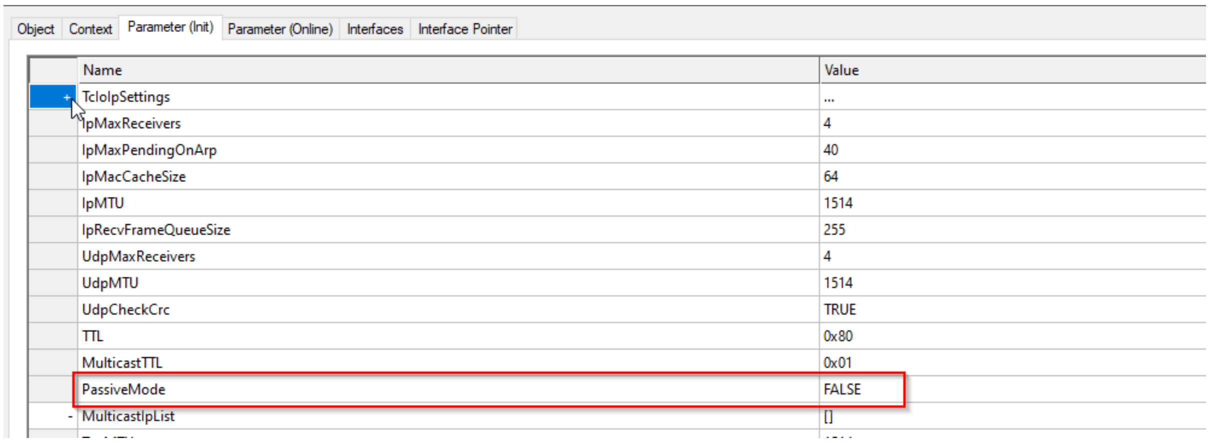
- A TCP/UDP RT object must be created for each real-time context (e.g. task) from which data is to be received or sent.



I/O
Devices
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
Device 1 (RT-Ethernet Adapter)_Obj2 (TCP/UDP RT)

- The PassiveMode parameter on all TCP/UDP RT objects specifies whether or not these objects should fetch frames received from the RT Ethernet adapter. By default, PassiveMode is set to FALSE so that packets are fetched.
  For multitask access, only one TCP/UDP RT object should fetch the data and all other objects should be configured with PassiveMode to TRUE.
  Typically, this can be the object that receives packets in the fastest cycle. Where appropriate, a lower

priority can be used for this in order to make the real-time processes of other tasks more independent of the incoming frames.


Object | Context | Parameter (Init) | Parameter (Online) | Interfaces | Interface Pointer

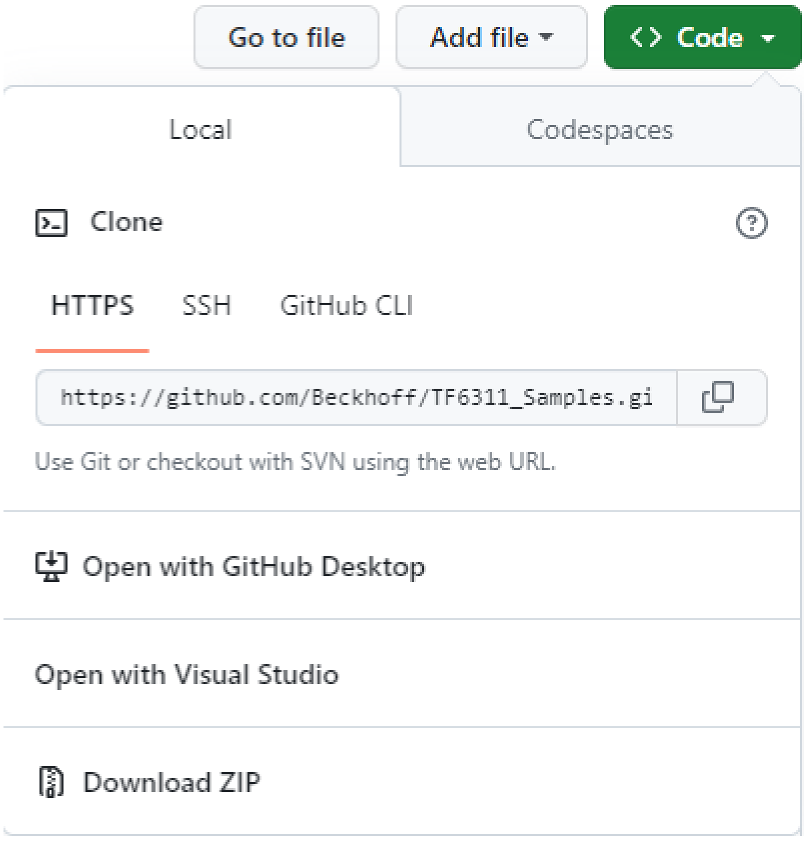
Name	Value
TcIoIpSettings	...
IpMaxReceivers	4
IpMaxPendingOnArp	40
IpMacCacheSize	64
IpMTU	1514
IpRecvFrameQueueSize	255
UdpMaxReceivers	4
UdpMTU	1514
UdpCheckCrc	TRUE
TTL	0x80
MulticastTTL	0x01
PassiveMode	FALSE
MulticastIpList	[]

- The function block must call the RegisterReceiver() / Open() method in the same context as it calls the CheckReceived() method in the cyclic process.
- The callbacks via ReceiveData()/...Event() are called in the same context as the CheckReceived() from the function block of the application previously.

6 Examples

These examples provide easy-to-follow demonstrations for dealing with the TCP/UDP RT module.

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://github.com/Beckhoff/TF6311_Samples. There you have the option to clone the repository or download a ZIP file containing the sample.



```
https://github.com/Beckhoff/TF6311_Samples.gi
```

Use Git or checkout with SVN using the web URL.

6.1 S01: Simple TCP Client (PLC / C++)

This sample shows the application of a TCP connection as client.
In this sample illustrates opening of a TCP connection with an IP address via port 80. The Beckhoff web server is used. The sample uses the connection to send an HTTP request to access a test website 62.159.14.51:80.

If the website does not fit into the receive buffer, the ReceiveData() method is called several times.

The client re-establishes a connection, if it was closed by the server, for example.

The sample is available for C++ and for the PLC.

6.1.1 S01: Simple TCP Client (C++)

This example implements a TCP client that issues a simple HTTP request and receives the response.

The download available here is preconfigured to call a test website 62.159.14.51:80.

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S01-IpStackTcpClient

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system

4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

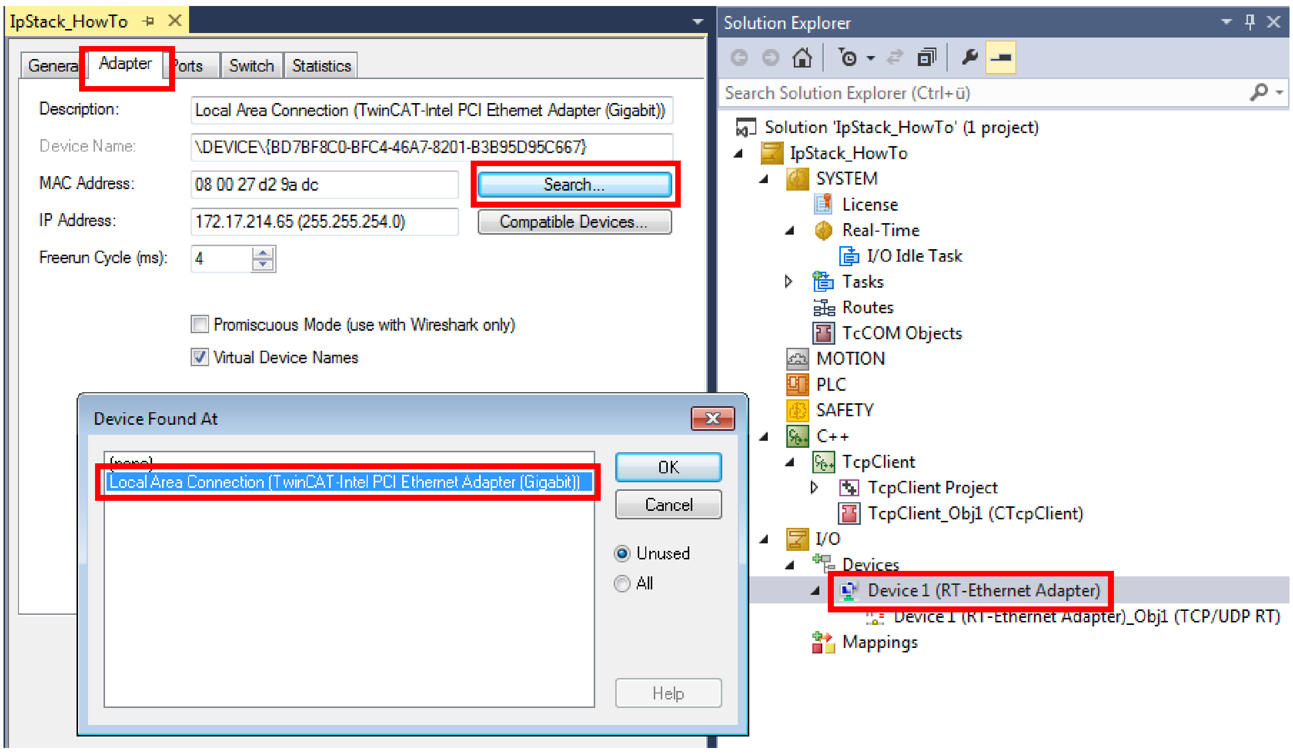
The example is described in detail on the Quick Start [▸ 22] page.

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4
☐ Promiscuous Mode (use with Wireshark only)
☑ Virtual Device Names

(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
[✓ Unused] All

```
Solution 'IpStack_HowTo' (1 project)
  IpStack_HowTo
    SYSTEM
      License
      Real-Time
        I/O Idle Task
      Tasks
      Routes
      TcCOM Objects
    MOTION
    PLC
    SAFETY
    C++
      TcpClient
        TcpClient Project
        TcpClient_Obj1 (CTcpClient)
    I/O
      Devices
        Device 1 (RT-Ethernet Adapter)
          Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
      Mappings
```

6.1.2 S01: Simple TCP Client (PLC)

This sample implements a TCP client that issues a simple HTTP request and receives the response.

The download available here is preconfigured to call a test website 62.159.14.51:80.

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S01-IpStackTcpClientPlc

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system
4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

After the startup, the PLC program can be used by setting the variable "bSend" to TRUE. The HTTP request (stored in "sMessage") is sent to the server, once the connection has been established. The first bytes of the incoming response are provided in "sLastReturnedMessage". The "sLastReturnedMessafeLength" indicates the whole length of the response.

The server address is defined in the FB_init method.

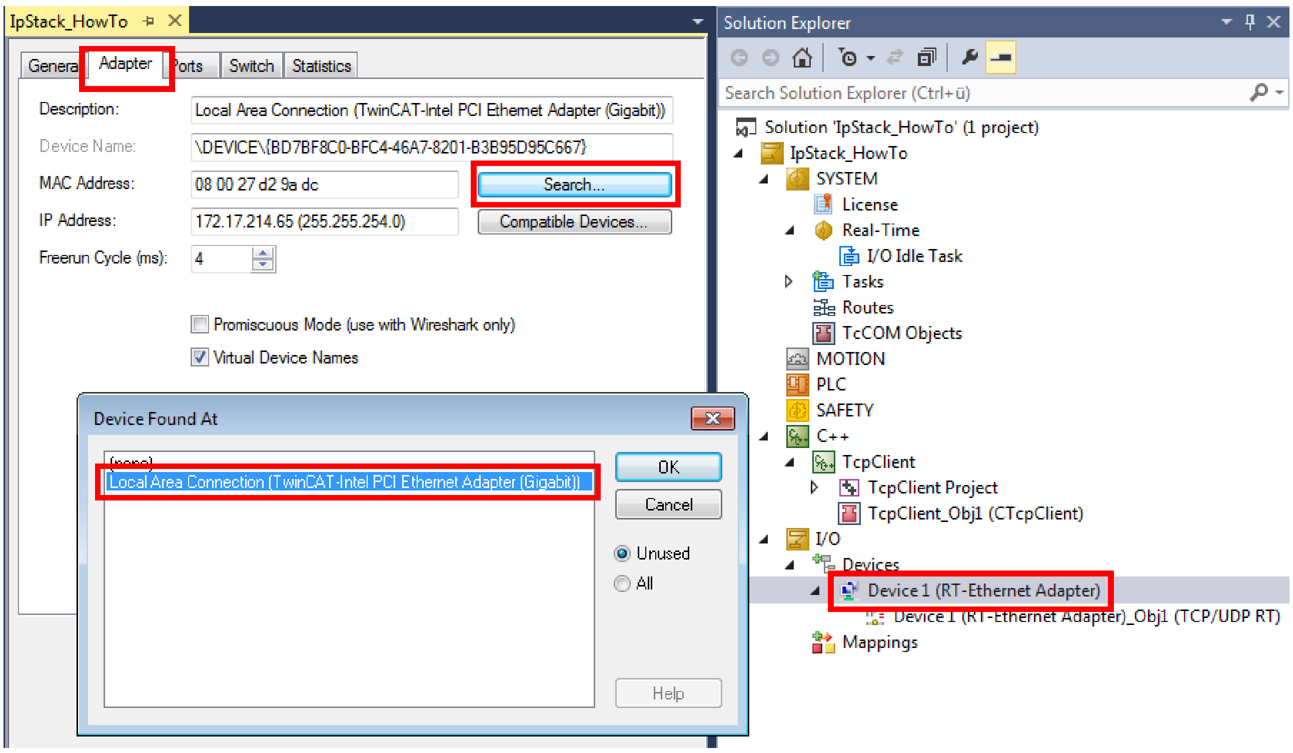
The same sample is described in detail for C++ on the Quick Start [▸ 22] page.

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4
☐ Promiscuous Mode (use with Wireshark only)
☑ Virtual Device Names

(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
[✓ Unused] All

6.2 S02: UDP Client Server (PLC/C++)

This example describes how a TwinCAT project can act as a UDP server. Thus, values can be delivered to the real-time or from the real-time on request.

The example implements an "echo service": A UDP server is started on a port (default: 10000). If this server receives a UDP packet, it returns the content to the sender (with same IP and same port). The example is available in PLC [▸ 40] and C++ [▸ 43].

For testing purposes, a UDP client [▸ 44] (written in .NET) is also available.

The samples are also available in more detail as Quick Starts [▸ 11].

6.2.1 S02: UDP Demo (PLC)

This example describes a UDP server that is implemented in a PLC project.

It receives UDP packets and returns them to the sender ("echo server").

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S02-UdpDemoPlc

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system
4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

The sample is also available in more detail as Quick Start.

The interface ITcIoUdpProtocolRecv [▶ 52] is implemented and a pointer to a ITcIoUdpProtocol [▶ 52] is used analogous to the Quick Start [▶ 22] in this sample.

To this end a PLC block is created, which implements the interface ITcIoUdpProtocolRecv [▶ 52] ("Add POU" with "Implements"). It is important to realize the connection to the TCP/UDP RT object in the "FB_init" and "FB_exit" methods. This procedure is described in more detail in Sample 11 of the C++ documentation.

The implementing function block (in sample UdpReceiver) calls the method "CheckReceived". In this way the IP stack is enabled to process incoming packets and transmit callbacks on the "ReceiveData" method of the function block.

The "ReceiveData" method uses the "SendData" method to return the data to the sender ("echo server").

Understanding

Two methods are used to establish the communication between the function block and the TcCOM object "TCP/UDP RT":

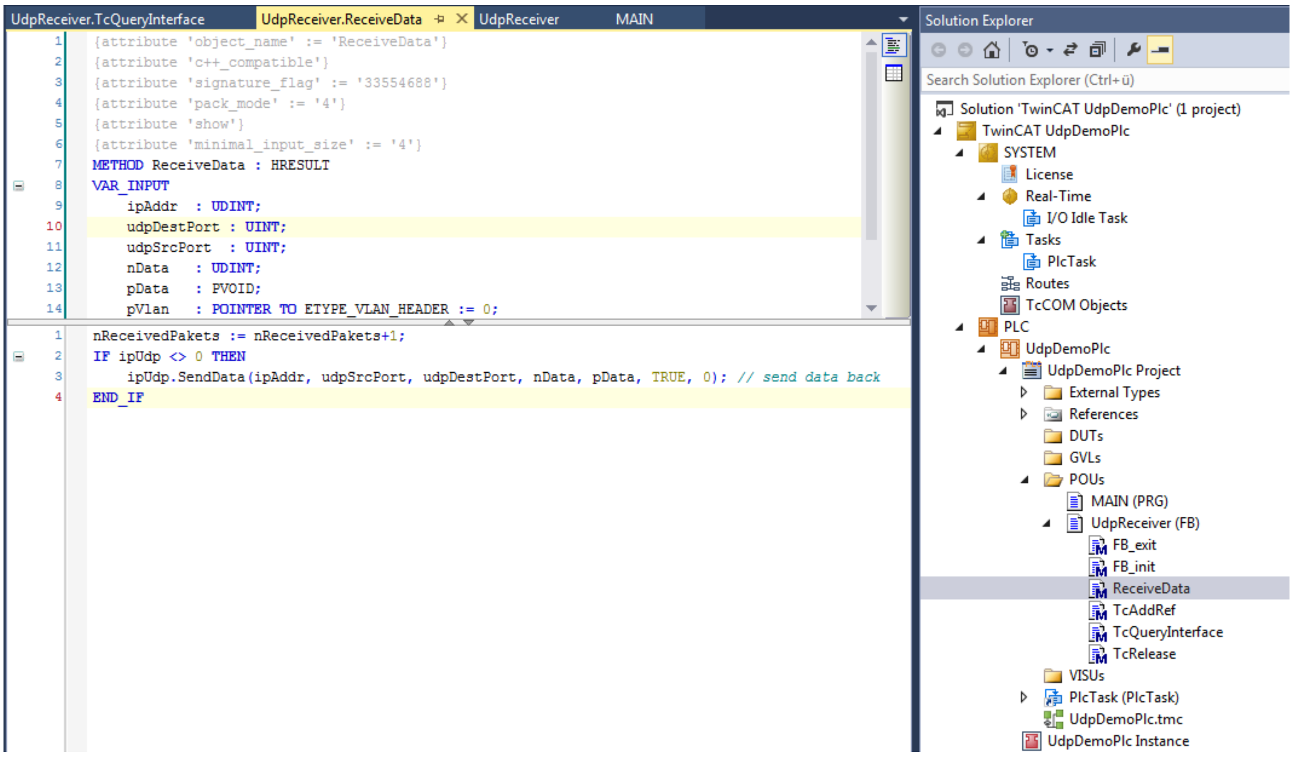
- "FB_init": This is executed automatically when the PLC is started
- "FB_exit": This is executed automatically when the PLC is stopped

This initialization phase can largely be taken from the sample code.

Two methods are responsible for the actual UDP functionality in the PLC code:

- The "ReceiveData" method in the implemented function block receives the data.
- The "SendData" method in the ITcIoUdpProtocol interface sends data.

In the sample, the "SendData" method is used in the "ReceiveData" method to return the received data:



```
{attribute 'object_name' := 'ReceiveData'}
{attribute 'c++_compatible'}
{attribute 'signature_flag' := '33554688'}
{attribute 'pack_mode' := '4'}
{attribute 'show'}
{attribute 'minimal_input_size' := '4'}
METHOD ReceiveData : HRESULT
VAR_INPUT
    ipAddr   : UDINT;
    udpDestPort : UINT;
    udpSrcPort  : UINT;
    nData    : UDINT;
    pData    : PVOID;
    pVlan    : POINTER TO ETYPE_VLAN_HEADER := 0;
```

```
nReceivedPakets := nReceivedPakets+1;
IF ipUdp <> 0 THEN
    ipUdp.SendData(ipAddr, udpSrcPort, udpDestPort, nData, pData, TRUE, 0); // send data back
END_IF
```

The TcQueryInterface method must be implemented as follows to ensure that TwinCAT detects that the corresponding interface was implemented:

```
VAR
ipUdpRecv : ITcIoUdpProtocolRecv;
ipUnknown : ITcUnknown;
END_VAR


IF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_ITcIoUdpProtocolRecv)) THEN
ipUdpRecv := THIS^; // cast to interface pointer
pipItf^ := ITCUNKNOWN_TO_PVOID(ipUdpRecv);
TcAddRef();
TcQueryInterface := S_OK;
ELSIF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_ITcUnknown)) THEN
ipUnknown := THIS^; // cast to interface pointer
pipItf^ := ITCUNKNOWN_TO_PVOID(ipUnknown);
TcAddRef();
TcQueryInterface := S_OK;
ELSE
TcQueryInterface := E_HRESULTAdsErr.NOINTERFACE ; //Call super if this fb extends some other
END_IF
```

The additionally created methods

- TcAddRef / TcRelease

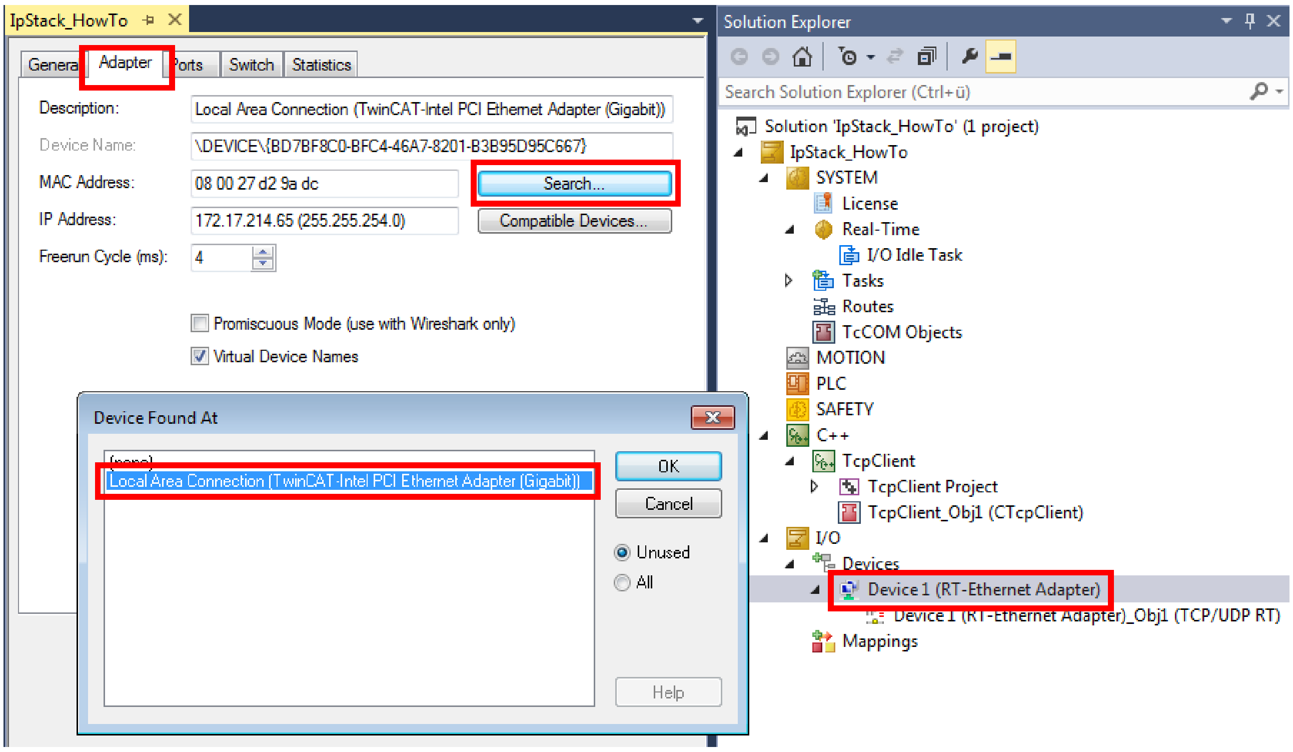
are inherited by the ITcUnknown interface and are not relevant in this context. For background information we suggest reading the chapter on the TcCOM module concept in the C++ domain.

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4

[ ] Promiscuous Mode (use with Wireshark only)
[✓] Virtual Device Names

(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))

[✓ Unused] All

```
Solution 'IpStack_HowTo' (1 project)
  IpStack_HowTo
    SYSTEM
      License
      Real-Time
        I/O Idle Task
      Tasks
      Routes
      TcCOM Objects
    MOTION
    PLC
    SAFETY
    C++
      TcpClient
        TcpClient Project
        TcpClient_Obj1 (CTcpClient)
    I/O
      Devices
        Device 1 (RT-Ethernet Adapter)
          Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
      Mappings
```

6.2.2 S02: UDP Demo (C++)

This example describes a UDP server that is implemented in C++.

It receives UDP packets and returns them to the sender ("echo server").

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S02-UdpDemo

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system
4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

The interface ITcIoUdpProtocolRecv [▶ 52] is implemented and a pointer to a ITcIoUdpProtocol [▶ 52] is used analogous to the Quick Start [▶ 22] in this example.

Using "RegisterReceiver" in the Transition SO ensures that the module is registered for the transmitted port (default: 10000). A corresponding unregistration takes place in the Transition OS.

The "CheckReceived" method is called in the "CycleUpdate" method. In this way the TCP/UDP RT module is enabled to process incoming packets and transmit callbacks on the "ReceiveData" method to the module.

The "ReceiveData" method uses the "SendData" method to return the data to the sender ("echo server").

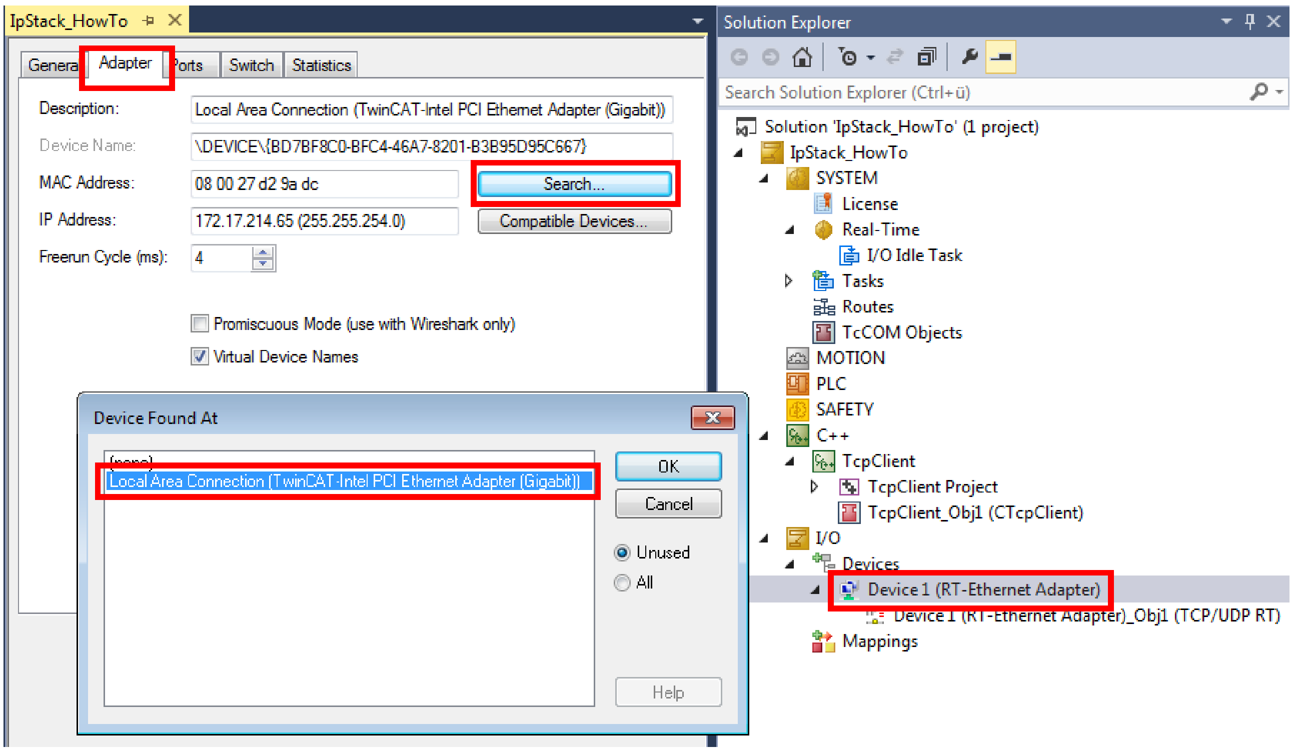
The sample is also available in more detail as Quick Start [▶ 16].

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4
[✓ Virtual Device Names]

(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))

6.2.3 Test client

The test client is used to send and receive single UDP data packets to and from a UDP server.

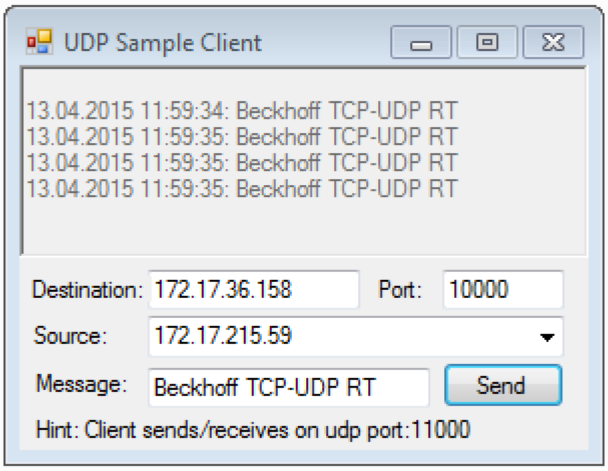
Download

Download the test client.

Unpack the ZIP file; the .exe file runs on a Windows system.

Description

The client itself uses port 11000 for sending. At the same time it opens this port and displays received messages in the upper part of the interface as a log:



```
13.04.2015 11:59:34: Beckhoff TCP-UDP RT
13.04.2015 11:59:35: Beckhoff TCP-UDP RT
13.04.2015 11:59:35: Beckhoff TCP-UDP RT
13.04.2015 11:59:35: Beckhoff TCP-UDP RT
```

Destination: 172.17.36.158 Port: 10000
Source: 172.17.215.59
Message: Beckhoff TCP-UDP RT
Hint: Client sends/receives on udp port:11000

Together with the PLC / C++ samples, this results in an echo sample:
A UDP message is sent from the client port 11000 to the server port 10000, which sends the same data back to the sender.

The client can be configured via the interface:

- Destination: Destination IP address
- Port: The port that is addressed in the target
- Source: Sender network card (IP address).
  "OS-based" operating system deals with selection of the appropriate network card.
- Message

The TF6311 "TCP/UDP Realtime" does not allow local communication. However, for testing purposes a different network interface can be selected via "Source", so that the UDP packet leaves the computer through one network card and arrives on the other network card ("loop cable").

6.3 S03: ARP PING Demo (C++)

This example describes an ARP and PING client.

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S03-PingClient

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system
4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

The interface ARP/Ping: ITcIoArpPingProtocol(Recv) [▸ 64] is implemented and a pointer to a ARP/Ping: ITcIoArpPingProtocol(Recv) [▸ 65] is used analogous to the Quick Start [▸ 22] in this example.

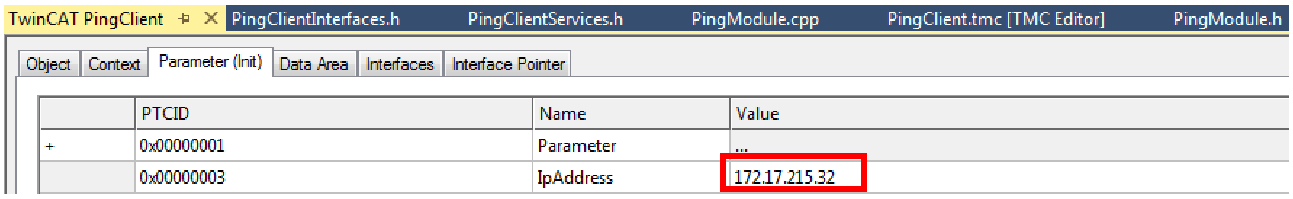
Using "RegisterReceiver" in the Transition SO ensures that the module is registered for receiving Arp and Ping messages. A corresponding unregistration takes place in the Transition OS.

The "CheckReceived" method is called in the "CycleUpdate" method. In this way the TCP/UDP RT module is enabled to process incoming packets and transmit callbacks on the "ArpReply" und "PingReply" methods to the module.

Understanding

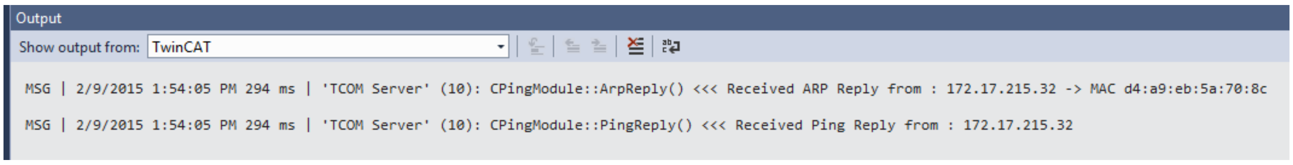
The procedure is carried out once when the program starts.

If "m_bSendRequest" is set to TRUE (e.g. through TwinCAT Live Watch), a new request (ARP and Ping) is sent to the IP address defined here:



	PTCID	Name	Value
+	0x00000001	Parameter	...
	0x00000003	IpAddress	172.17.215.32

The output is in the messages:



```
MSG | 2/9/2015 1:54:05 PM 294 ms | 'TCOM Server' (10): CPingModule::ArpReply() <<< Received ARP Reply from : 172.17.215.32 -> MAC d4:a9:eb:5a:70:8c
MSG | 2/9/2015 1:54:05 PM 294 ms | 'TCOM Server' (10): CPingModule::PingReply() <<< Received Ping Reply from : 172.17.215.32
```

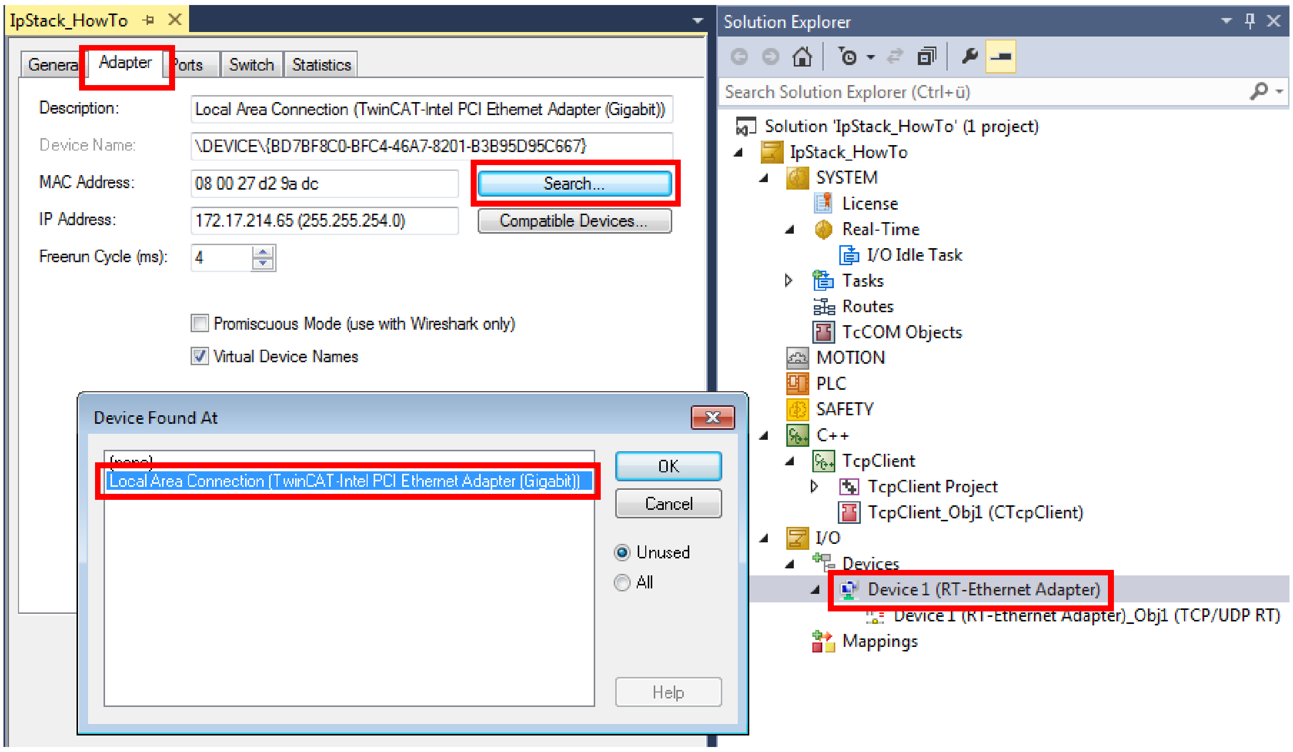
To output these messages the "Tracelevel" can be configured (via Info).

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4
Promiscuous Mode (use with Wireshark only)
[✓] Virtual Device Names
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))

6.4 S04: TCP Echo Server (PLC / C++)

This sample describes a TCP server accepting an income connection.
Data sent to this server are simply returned as “echo”.

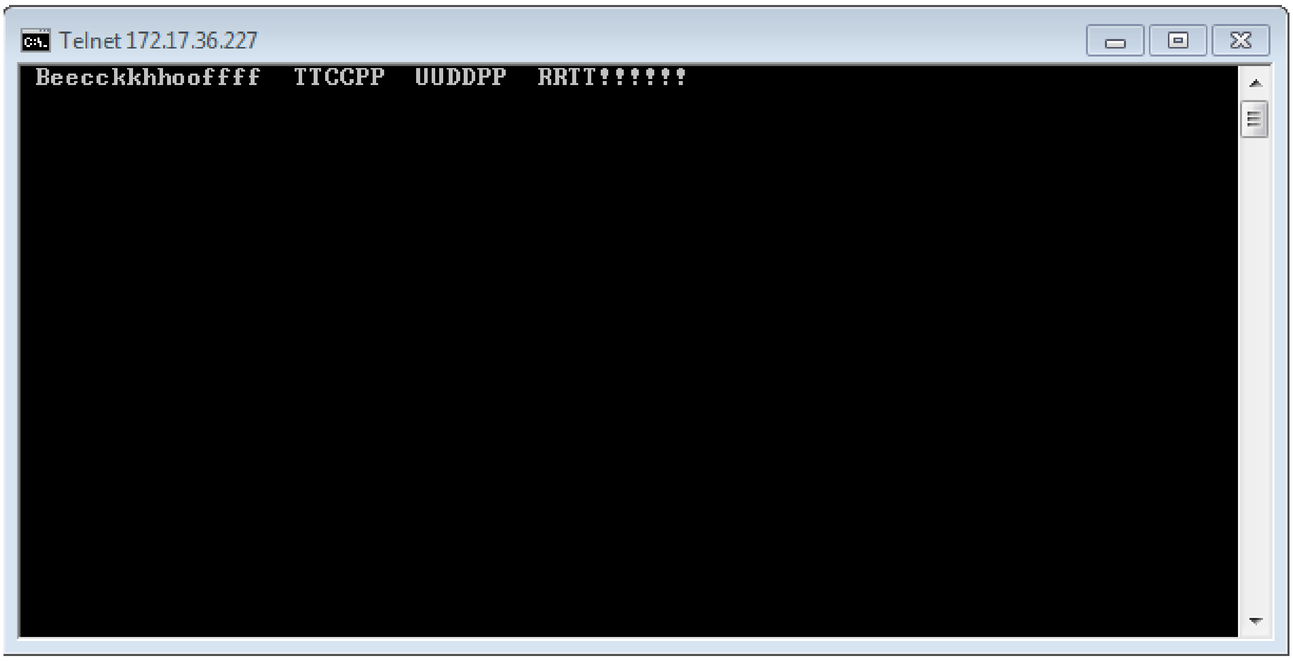
The same sample is available for C++ and PLC. By default, the server runs on port 11000.

Testing the sample

The sample can be tested via "telnet".

```
%>telnet 192.168.1.1 11000
```

If a character is sent via telnet, it is returned immediately. A picture similar to the following emerges:



```
Beecckkhhooffff  TTCCPP  UUDDPP  RRTT!!!!!!
```

6.4.1 S04: TCP Server Demo (PLC)

This sample describes a TCP server that is implemented in a PLC project.

It accepts a TCP connection, receives TCP packets and returns them to the sender (“echo server”).

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S04-TCPServerPlc

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system
4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

The interface ITcIoTcpProtocolRecv [▶ 58] is implemented and a pointer to a ITcIoTcpProtocol [▶ 58] is used analogous to the Quick Start [▶ 11]s in this sample.

To this end a PLC block is created, which implements the interface ITcIoUdpProtocolRecv [▶ 58] (“Add POU” with “Implements”). It is important to realize the connection to the TCP/UDP RT object in the “FB_init” and “FB_exit” methods. In particular, the Quick Starts illustrate how this OnlineChange can be implemented securely. The procedure is described in more detail in Sample 11 of the C++ documentation.

The implementing function block (in sample TCPServer) calls the method “CheckReceived”. In this way the IP stack is enabled to process incoming packets and transmit callbacks relating to the “ReceiveData” and “ReceiveEvent” methods of the function block.

In order to take into account incoming connections, a port is opened in FB_init via “AllocSocket” and “Listen”. “Accept” is called in the “ReceiveEvent”, if an event to establish a connection has occurred.

In this sample the “ReceiveData” method uses the “SendData” method to return the data to the sender (“echo server”).

Understanding

Two methods are used to establish the communication between the function block and the TcCOM object “TCP/UDP RT”:

- „FB_init“: This is executed automatically when the PLC is started.
- „FB_exit“: This is executed automatically when the PLC is stopped.

This initialization phase can largely be taken from the sample code.

Two methods are responsible for the actual TCP functionality in the PLC code:

- The “ReceiveData” method in the implemented function block receives the data.
- The “ReceiveEvent” method indicates events occurring at the implemented function block.
- The “SendData” method in the ITcIoTcpProtocol interface sends data.

In the sample, the “SendData” method is used in the “ReceiveData” method to return the received data:
The TcQueryInterface method must be implemented as follows to ensure that TwinCAT detects that the corresponding interface was implemented:

```
VAR
ipTcpRecv : ITcIoTcpProtocolRecv;
ipUnknown : ITcUnknown;
END_VAR


IF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_ITcIoTcpProtocolRecv)) THEN
ipTcpRecv := THIS^; // cast to interface pointer
pipItf^ := ITCUNKNOWN_TO_PVOID(ipUdpRecv);
TcAddRef();
TcQueryInterface := S_OK;
ELSIF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_ITcUnknown)) THEN
ipUnknown := THIS^; // cast to interface pointer
pipItf^ := ITCUNKNOWN_TO_PVOID(ipUnknown);
TcAddRef();
TcQueryInterface := S_OK;
ELSE
TcQueryInterface := E_HRESULTAdsErr.NOINTERFACE ; //Call super if this fb extends some other
END_IF
```

The additionally created methods

- TcAddRef / TcRelease

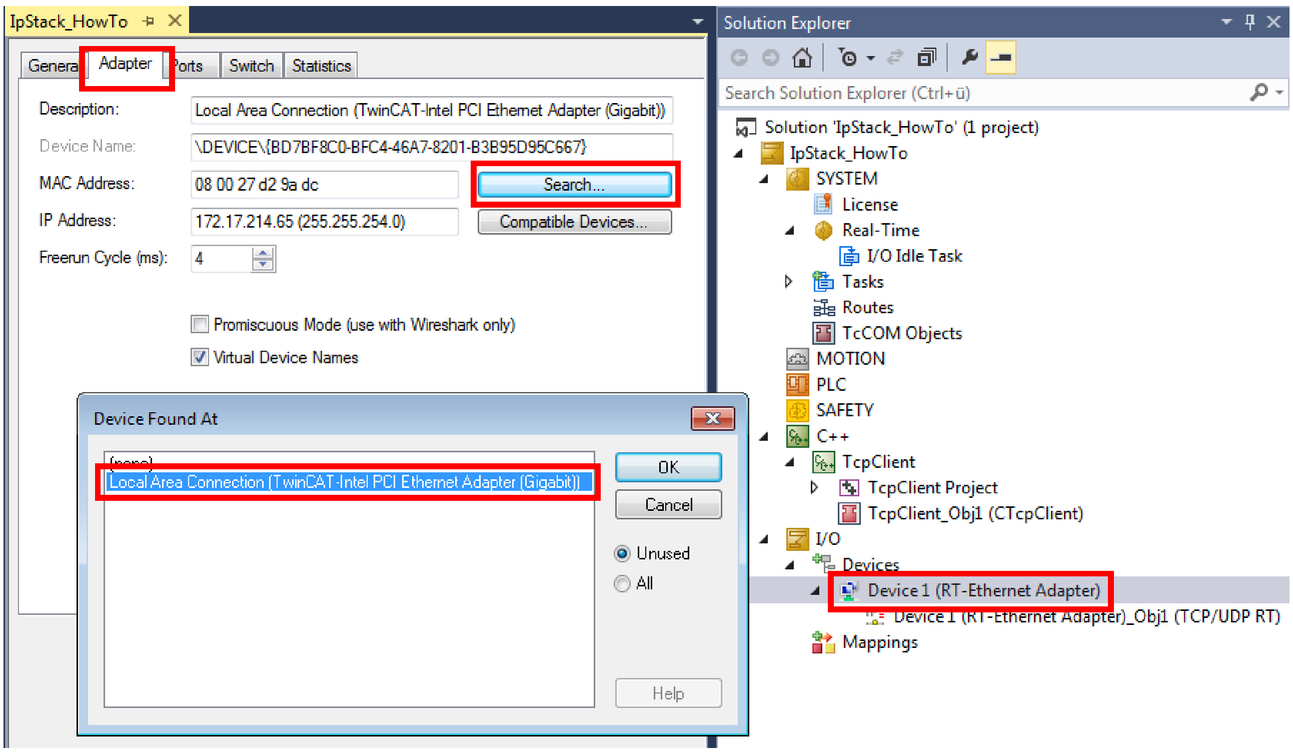
are inherited by the ITcUnknown interface and are not relevant in this context. For background information we suggest reading the chapter on the TcCOM module concept in the C++ domain.

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4

☐ Promiscuous Mode (use with Wireshark only)
☑ Virtual Device Names

(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))

[✓ Unused] All

6.4.2 S04: TCP Server Demo (C++)

This sample describes a TCP server that is implemented in C++.

It accepts a TCP connection, receives TCP packets and returns them to the sender (“echo server”).

Download

Download the sample: https://github.com/Beckhoff/TF6311_Samples/tree/main/S04-TCPServer

1. Get the sample from GitHub, unzip the downloaded ZIP file if necessary
2. Open the project with TwinCAT XAE
3. Select your target system
4. Configure the network card (see below) for the target system
5. Build the **sample on your local machine** (e.g. Build->Build Solution)
6. Activate the configuration

Description

The interface ITcIoTcpProtocolRecv is implemented and a pointer to a ITcIoTcpProtocol is used analogous to the Quick Starts in this sample.

The “CheckReceived” method is called in the “CycleUpdate” method. In this way the TCP/UDP RT module is enabled to process incoming packets and transmit callbacks on the “ReceiveEvent” und “ReceiveData” methods to the module.

In order to take into account incoming connections, a port is opened in “CycleUpdate” via “AllocSocket” and “Listen”. “Accept” is called in the “ReceiveEvent”, if an event to establish a connection has occurred.

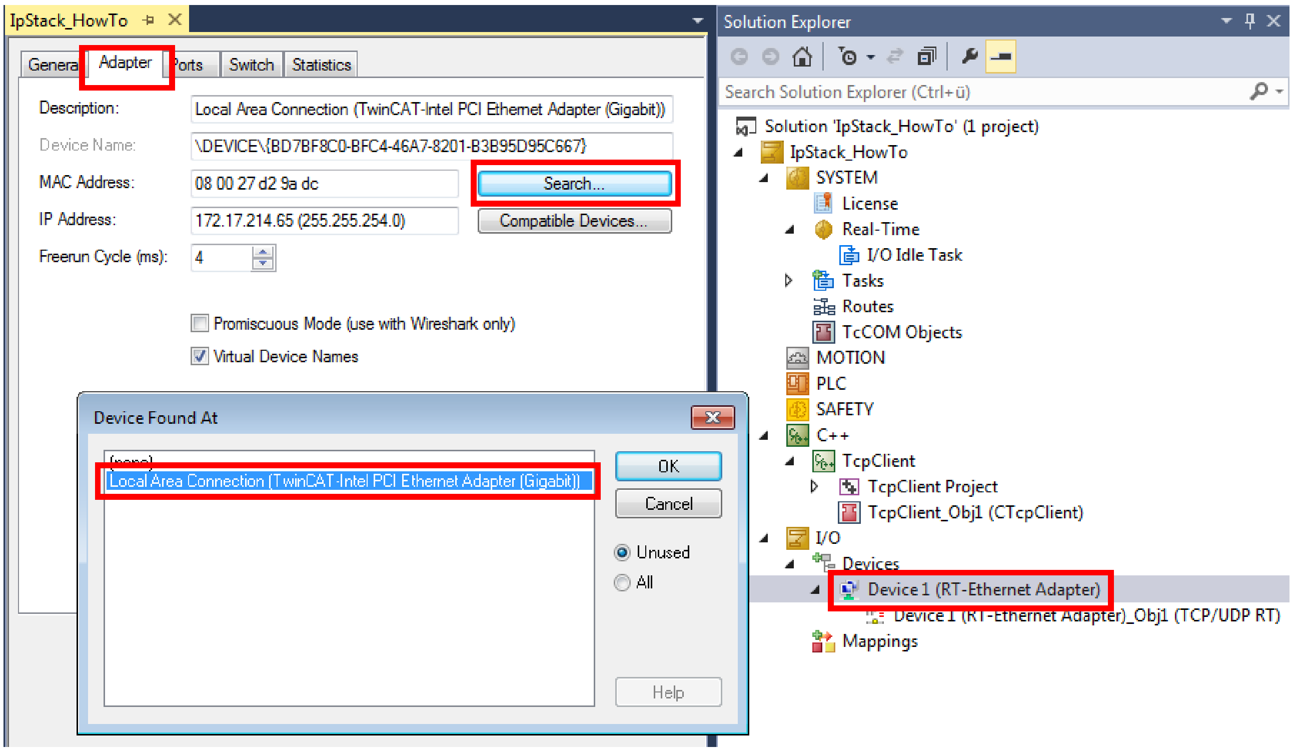
In this sample the “ReceiveData” method uses the “SendData” method to return the data to the sender (“echo server”).

Preparing the network card

For the TCP/UDP RT module, make sure that the RT Ethernet adapter in the TwinCAT solution is connected with the correct network card (with TwinCAT driver).

Local configuration only

Installation of the driver on compatible network cards via the button "Compatible Devices" always takes place locally. On a controller with TwinCAT XAR, the program TcRteInstall.exe can be used. It is included in the installation (usually under C:TwinCAT\3.1\System).



Description: Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Device Name: \DEVICE\{BD7BF8C0-BFC4-46A7-8201-B3B95D95C667}
MAC Address: 08 00 27 d2 9a dc
IP Address: 172.17.214.65 (255.255.254.0)
Freerun Cycle (ms): 4
[ ] Promiscuous Mode (use with Wireshark only)
[✓] Virtual Device Names

Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
[✓ Unused] All

Solution 'IpStack_HowTo' (1 project)
- IpStack_HowTo
  - SYSTEM
    - License
    - Real-Time
      - I/O Idle Task
    - Tasks
    - Routes
    - TcCOM Objects
  - MOTION
  - PLC
  - SAFETY
  - C++
    - TcpClient
      - TcpClient Project
      - TcpClient_Obj1 (CTcpClient)
  - I/O
    - Devices
      - Device 1 (RT-Ethernet Adapter)
        - Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)
    - Mappings

7 Programmer's reference

The programmer's reference provides an overview of the different parameters, interfaces and their methods.

These include:

- TCP/UDP RT TcCOM Parameters [▶ 55]: The parameters of the actual TCP/UDP RT module enable the configuration.

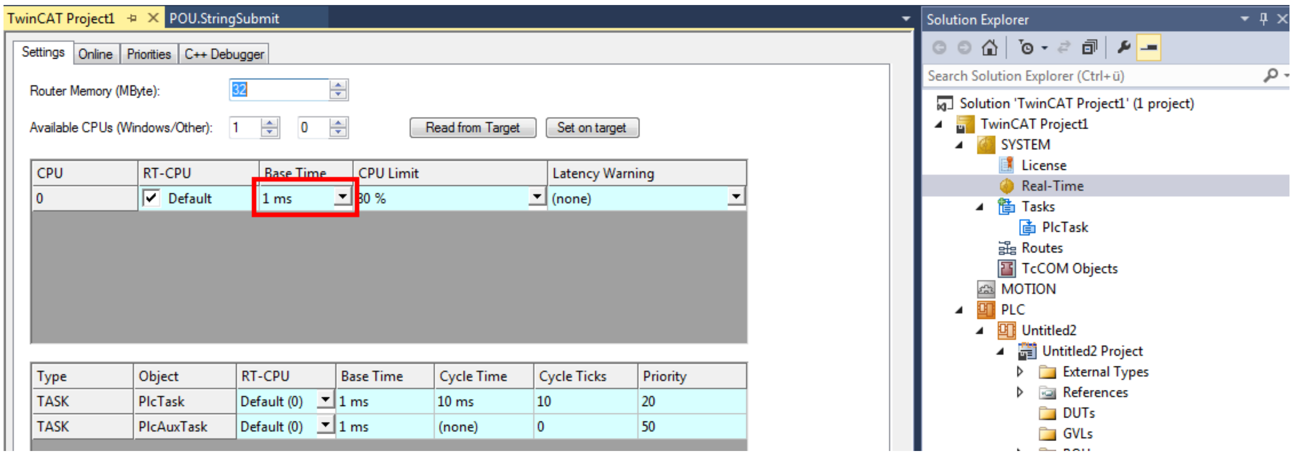
The TCP/UDP RT module can be used by different protocols. An InterfacePointer and an interface to be implemented always go hand in hand:

- ITcIoTcpProtocol(Recv): [▶ 58] TCP/IP protocol
- ITcIoUdpProtocol(Recv) [▶ 52]: UDP/IP protocol
- ITcIoArpPingProtocol(Recv) [▶ 64]: ARP/Ping protocol

For all uses of IP addresses (e.g. "IpAddr"), the most significant elements are displayed in the last position. (Example: 192.168.2.1 -> 01 02 A8 C0)

Performance

The TCP/UDP RT TcCOM object runs in real-time. Thus, the module is also directly dependent on the cycling of the real-time. The frequency with which data can be communicated can therefore be influenced by the cycling of the task used (and therefore also the real-time settings):

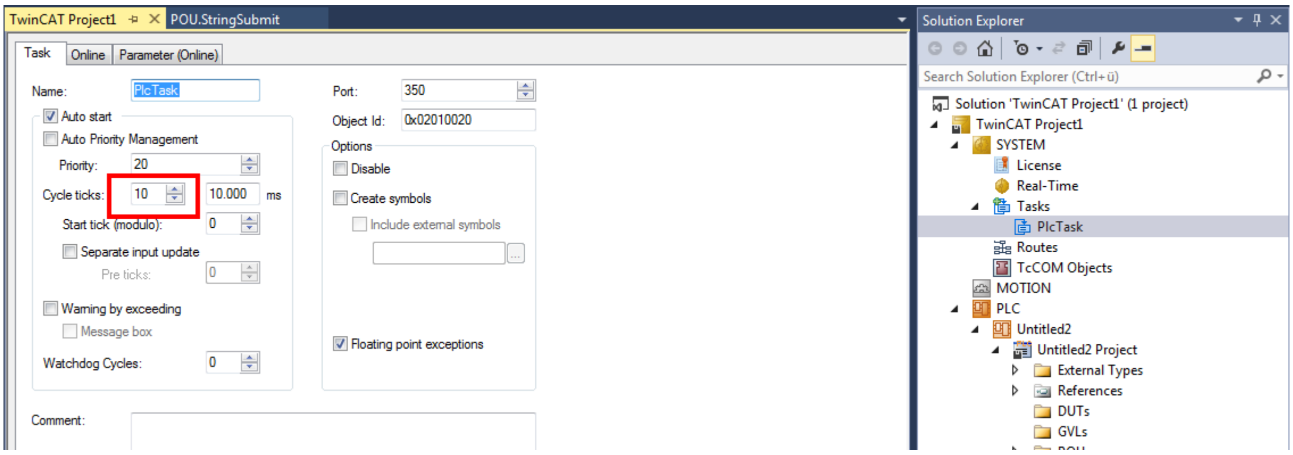


Router Memory (MByte): 32

Available CPUs (Windows/Other): 1 0

CPU	RT-CPU	Base Time	CPU Limit	Latency Warning
0	✓ Default	1 ms	80 %	(none)

Type	Object	RT-CPU	Base Time	Cycle Time	Cycle Ticks	Priority
TASK	PlcTask	Default (0)	1 ms	10 ms	10	20
TASK	PlcAuxTask	Default (0)	1 ms	(none)	0	50



Name: PlcTask
Port: 350
Object Id: 0x02010020
[✓] Auto start
[ ] Auto Priority Management
Priority: 20
Cycle ticks: 10 10.000 ms
Start tick (modulo): 0
[ ] Separate input update
Pre ticks: 0
[ ] Warning by exceeding
[ ] Message box
Watchdog Cycles: 0
[ ] Disable
[ ] Create symbols
[ ] Include external symbols
[✓] Floating point exceptions
Comment:

Communication via the network interface depends on this cycle. A corresponding call to the CheckReceived() methods (see API documentation [▶ 51]) must be made in each cycle.

Incoming data: CheckReceived()

Context of the incoming data

The customer must ensure that the method CheckReceived is called cyclically. Samples illustrate the procedure in PLC and C++

The CheckReveived() method is called cyclically in order to ensure that the data can be provided in the same context as the client project. The protocol-dependent Receive() methods of the customer project are called within this method call, if data have been received.

Disconnection of Engineering connection on breakpoints

If breakpoints are used, we strongly advise to use different network interfaces, since a breakpoint stops parts of the TwinCAT systems, which may be relevant for the communication with Engineering.

7.1 UDP/IP: ITcIoUdpProtocol(Recv)

The ITcIoUdpProtocol and ITcIoUdpProtocolRecv interfaces enable UDP/IP communication from the real-time environment.

A project that uses this interface contains a pointer to an ITcIoUdpProtocol object and implements ITcIoUdpProtocolRecv itself. ITcIoUdpProtocolRecv serves as callback interface for receiving data from the TCP/UDP RT module within the application.

Multiple calls of Receive()

During the implementation it should be noted that CheckReceived() will result in the callback to Receive() occurring several times within a cycle, if multiple packets have arrived between the cycles.
A buffer in the form a queue may therefore have to be provided.

ITcIoUdpProtocolRecv methods:

Name	Description
ReceiveData [▸ 53]	Is called by the TCP/UDP RT module as a callback to transfer data

ITcIoUdpProtocol methods:

Name	Description
SendData [▸ 53]	Sends data
CheckReceived [▸ 54]	Must be called cyclically. ReceiveData is used as callback in the context of this method (server and client functionality).
RegisterReceiver [▸ 54]	Registering at the TCP/UDP RT module for receiving data.
UnregisterReceiver [▸ 55]	Unregistering at the TCP/UDP RT module for receiving UDP data.

The client and server implementation process is briefly described here. Only an overview is provided; the samples illustrate the application.

Implementation of a UDP sender / receiver

Name	Description
RegisterReceiver [▸ 54]	Opens a port for incoming data packets.
ReceiveData [▸ 53]	Is called when data packets arrive.
SendData [▸ 53]	Can be used to send data.
UnregisterReceiver [▸ 55]	For logout from (closing of) the port, e.g. during shutdown.

To receive UDP data, registration is required by calling RegisterReceiver. This can be done in SetObjStateSO or FB_init.

Data is provided by a callback of method ReceiveData from ITcIoUdpProtocolRecv.

While TwinCAT switches from RUN mode to Config mode, all modules should unregister via UnregisterReceiver. This can be done in SetObjStateOS() or FB_exit.

NOTICE

OnlineChange security

For OnlineChange security, RegisterReceiver should be called again.

7.1.1 Method ITcIoUdpProtocolRecv:ReceiveData

Is called by the TCP/UDP RT module as a callback to transfer data.

Syntax

```
HRESULT TCOMAPI ReceiveData(ULONG ipAddr, USHORT udpDestPort, USHORT udpSrcPort, ULONG nData, PVOID
pData, ETYPE_VLAN_HEADER* pVlan=0)
```

Return value

Type	Description
HRESULT	Indicates success and must be provided accordingly by the implemented module.

Parameter

Name	Type	Description
ipAddr	ULONG	The IP address of the sender. IP addresses are displayed with the most significant element in the last position. (Example: 192.168.2.1 -> 01 02 A8 C0)
udpDestPort	USHORT	Port on which the data was received.
udpSrcPort	USHORT	Port of the sender.
nData	ULONG	Number of bytes received.
pData	PVOID	Pointer to the received data.
pVlan	ETYPE_VLAN_HEADER	ETYPE_VLAN_HEADER structure - see below.

The VLAN header represents information about the VLAN.

```
typedef struct _ETYPE_VLAN_HEADER
{
USHORT VLanType;
unsigned short VLanIdH   : 4;
unsigned short reserved1 : 1;
unsigned short Priority  : 3;
unsigned short VLanIdL   : 8;
} ETYPE_VLAN_HEADER, *PETYPE_VLAN_HEADER;
```

7.1.2 Method ITcIoUdpProtocol:SendData

Sends data.

Syntax

```
HRESULT TCOMAPI SendData(ULONG ipAddr, USHORT udpDestPort, USHORT udpSrcPort, ULONG nData, PVOID
pData, bool bCalcUdpCheckSum=0, ETYPE_VLAN_HEADER* pVlan=0)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
ipAddr	ULONG	The IP address of the receiver. IP addresses are displayed with the most significant element in the last position. (Example: 192.168.2.1 -> 01 02 A8 C0)
udpDestPort	USHORT	The port of the receiver.
udpSrcPort	USHORT	The port of the sender.
nData	ULONG	Number of date to be sent in bytes.
pData	PVOID	Pointer to the data to be sent.
bCalcUdpCheckSum	BOOL	Indicates whether the checksum should be calculated.
pVlan	ETYPE_VLAN_HEADER	ETYPE_VLAN_HEADER structure, see below.

The VLAN header represents information about the VLAN.

```
typedef struct _ETYPE_VLAN_HEADER
{
USHORT VLanType;
unsigned short VLanIdH   : 4;
unsigned short reserved1 : 1;
unsigned short Priority  : 3;
unsigned short VLanIdL   : 8;
} ETYPE_VLAN_HEADER, *PETYPE_VLAN_HEADER;
```

7.1.3 Method ITcIoUdpProtocol:CheckReceived

Must be called cyclically; ReceiveData is used as callback in the context of this method (send and receive).

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

7.1.4 Method ITcIoUdpProtocol:RegisterReceiver

Registering at the TCP/UDP RT module for receiving data.

Syntax

```
HRESULT TCOMAPI RegisterReceiver(USHORT udpPort, ITcIoUdpProtocolRecv* ipRecv)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
udpPort	USHORT	Port on which the data is to be received.
ipRecv	ITcIoUdpProtocolRecv*	Pointer to the receiver (Recv) interface.

7.1.5 Method ITcIoUdpProtocol:UnregisterReceiver

Unregistering at the TCP/UDP RT module for receiving data.

Syntax

```
HRESULT TCOMAPI UnregisterReceiver(USHORT udpPort)
```

Return value

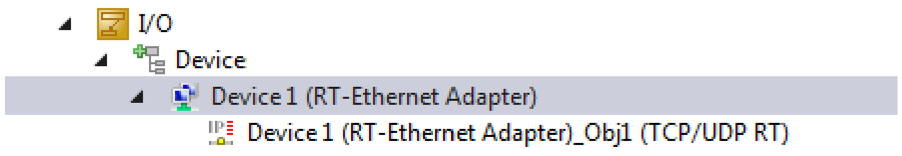
Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

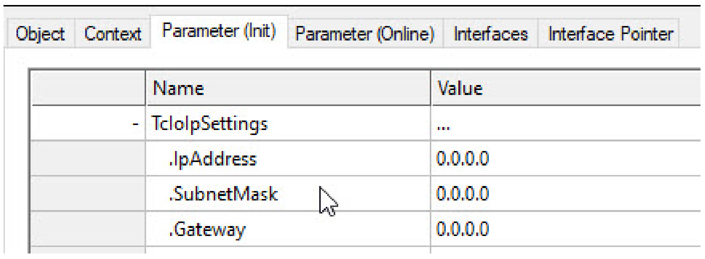
Name	Type	Description
udpPort	USHORT	Port at which the data should no longer be received.

7.2 TCP/UDP RT TcCom Parameter

In addition to the interfaces, the TcCOM object "TCP/UDP RT" is the main component of the function. An instantiation usually takes place under the device:


I/O
Device
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

Double-click to open the instance, so that the parameters documented below can be used:



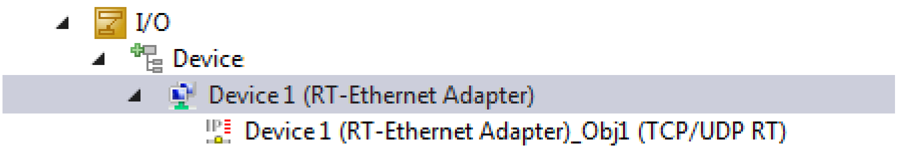
Name	Value
- TcIoIpSettings	...
.IpAddress	0.0.0.0
.SubnetMask	0.0.0.0
.Gateway	0.0.0.0

Name	Default value	Description
TcIoIpSettings.IpAddress	0.0.0.0	Own (local) IP address used for communication.
TcIoIpSettings.SubnetMask	0.0.0.0	Own subnet mask
TcIoIpSettings.Gateway	0.0.0.0	Gateway used to reach communication partners outside your own network.
TcIoIpSettings.DhcpEnable	FALSE	Not yet implemented.
TcIoIpSettings.ManualSettings	FALSE	Set to FALSE: The operating system uses the current IP configuration of the referenced adapter. Set to TRUE: Parameters of TcIoIpSettings* are used.
IpMaxReceivers	4	Maximum number of permitted IP-based protocols.
IpMaxPendingOnArp	40	Maximum number of entries in the ARP Request Table.
IpMacCacheSize	64	Number of entries in MAC cache, i.e. IP address to MAC address allocations. Caching is implemented as LRU.
IpMTU	1514	Not yet implemented. (Maximum transport unit size for IP packets)
IpRecvFrameQueueSize	255	Number of entries in the queue for receiving Udp packets.
UdpMaxReceivers	4	Maximum number of UDP receivers
UdpMTU	1514	From TwinCAT 3.1 Build 4026: Maximum Transport Unit size for UDP. Fragmentation is ready. In earlier versions (<= Build 4024) this parameter has no function
UdpCheckCrc	TRUE	Set to TRUE means that UDP packets with incorrect checksum are discarded.
TTL	0x80	TTL in the IP header of the frames to be sent.
MultiCastTTL	0x01	TTL of the MultiCast frames to be sent.
PassiveMode	FALSE	If TRUE, no frames are fetched from the RT network adapter frames by this instance. See Multitask access to a network card [▶ 36]
MulticastIpList	[]	Multicast addresses for receiving MultiCast packets.
TcpMTU	1514	Not yet implemented. (Maximum transport unit size for TCP)
TcpCheckCrc	TRUE	Incoming TCP frames are checked for valid checksum and discarded, if the checksum is incorrect.
TcpMaxSocketCount	32	Maximum number of sockets that are managed by the IP stack.
TcpReceiveBufferSize	16192	Number of received bytes that can be cached with a TCP connection.
TcpTransmitBufferSize	16192	Number of bytes to be sent that can be cached in the TCP stack with a connection.
TcpMaxRetry	5	Number of retries of TCP packets until the connection is terminated.
TcpTimeoutCon	5000	Timeout for TCP connection establishment and disconnection.
TcpTimeoutWait	60000	Timespan for storing handles internally after an unexpected termination of the connection.
TcpTimeoutIdle	1000	Time to callback (ReveiveEvent), if no response.
TcpRoundTripTime	3000	Start value for the timeout of data packets. Is adjusted dynamically depending on the connection quality (depending on the packet round-trip time).

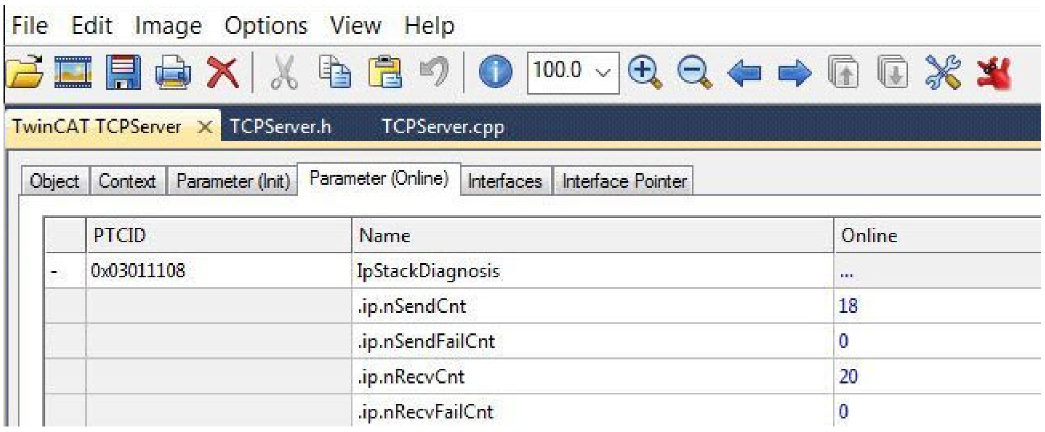
Times are given in milliseconds.

7.3 TCP/UDP RT TcCom diagnostics

The TcCOM object TCP/UDP RT represents the coupling of customer project with the hardware.


I/O
Device
Device 1 (RT-Ethernet Adapter)
Device 1 (RT-Ethernet Adapter)_Obj1 (TCP/UDP RT)

In addition to parameters, it therefore also contains diagnostic information, which is described here. Once Engineering can communicate with the target system and the program runs smoothly, various information is provided via the received and sent packets:


File Edit Image Options View Help
100.0
TwinCAT TCPServer
TCPServer.h
TCPServer.cpp
Object Context Parameter (Init) Parameter (Online) Interfaces Interface Pointer

	PTCID	Name	Online
-	0x03011108	IpStackDiagnosis	...
		.ip.nSendCnt	18
		.ip.nSendFailCnt	0
		.ip.nRecvCnt	20
		.ip.nRecvFailCnt	0

Name	Value	Description
IpStackDiagnosis	…	Diagnostic information of the IP stack
.ip.nSendCnt	18	Number of IP packets sent
.ip.nSendFailCnt	0	Number of IP packets not sent
.ip.nRecvCnt	20	Number of packets received
.ip.nRecvFailCnt	0	Number of packets not received
.arpRequest.nSendCnt	0	Arp-Requests: Number of packets sent
.arpRequest.nSendFailCnt	0	Arp-Requests: Number of packets not sent
.arpRequest.nRecvCnt	12	Arp-Requests: Number of packets received
.arpRequest.nRecvFailCnt	0	Arp-Requests: Number of packets not received
.arpReply.nSendCnt	12	Arp-Reply: Number of packets sent
.arpReply.nSendFailCnt	0	Arp-Reply: Number of packets not sent
.arpReply.nRecvCnt	0	Arp-Reply: Number of packets received
.arpReply.nRecvFailCnt	0	Arp-Reply: Number of packets not received
.pingRequest.nSendCnt	0	Ping-Request: Number of packets sent
.pingRequest.nSendFailCnt	0	Ping-Request: Number of packets not sent
.pingRequest.nRecvCnt	0	Ping-Request: Number of packets received
.pingRequest.nRecvFailCnt	0	Ping-Request: Number of packets not received
.pingReply.nSendCnt	0	Ping-Reply: Number of packets sent
.pingReply.nSendFailCnt	0	Ping-Reply: Number of packets not sent
.pingReply.nRecvCnt	0	Ping-Reply: Number of packets received
.pingReply.nRecvFailCnt	0	Ping-Reply: Number of packets not received
.nLinkStatusChangedCnt	1	Number of link changes
.nAllocFailCnt	0	Number of failed allocations
.nArpTimeoutFrames	0	Number of arp frames in the timeout
.nDroppedFrames	0	Number of discarded packages

7.4 TCP/IP: ITcIoTcpProtocol(Recv)

The ITcIoTcpProtocol and ITcIoTcpProtocolRecv interfaces enable TCP/IP communication from the real-time environment.

A project that uses this interface contains a pointer to an ITcIoTcpProtocol object and implements ITcIoTcpProtocolRecv itself. ITcIoTcpProtocolRecv serves as a callback interface for receiving data and events from the TCP/IP module within the application. The interfaces are based on a socket API. Before a socket can be used, it must be allocated with AllocSocket().

ITcIoTcpProtocolRecv methods:

Name	Description
ReceiveData [▸ 59]	Is called by the TCP/UDP RT module as a callback to transfer data.
ReceiveEvent [▸ 60]	Is called by the TCP/UDP RT module as a callback if an event has occurred.

ITcIoTcpProtocol methods:

Name	Description
AllocSocket [▸ 60]	Allocates a socket.
FreeSocket [▸ 61]	Enables a socket.
Connect [▸ 61]	Establishes a connection to a remote terminal.
IsConnected [▸ 61]	Indicates whether a socket is connected (for inbound and outbound connections).
Close [▸ 62]	Closes a socket.
Listen [▸ 62]	Opens a TCP port for incoming connections (see remarks).
Accept [▸ 62]	For server functionality: Accepts incoming connections (see remarks).
SendData [▸ 63]	Sends data (server and client functionality).
CheckReceived [▸ 63]	Must be called cyclical; ReceiveEvent and ReceiveData are used as callback in the context of this method (server and client functionality).
GetRemoteIpAddr [▸ 63]	Returns the remote IP address of a communication partner.
GetFreeSendDataSize [▸ 64]	Returns the number of free bytes in the TCP send buffer.

Call CheckReceived() continuously.

Perhaps call AllocSocket() again in the event of an OnlineChange, in order to refresh the callback target.

The client and server implementation process is described here, independent of programming languages. Only an overview is provided; the samples illustrate the application.

Implementation of an TCP server:

Name	Description
AllocSocket [▶ 60]	Opens a socket.
Listen [▶ 62]	Opens a port on which connections are expected.
Accept [▶ 62]	Is called in the ReceiveEvent() method in order to accept a connection.
ReceiveData [▶ 59]	Is called when data are received.
SendData [▶ 63]	Can be used to send data.
FreeSocket [▶ 61]	On the Listen socket and all connection sockets for stopping.

Code diagram for accepting a connection:

```
HRESULT CIpStackDemo::ReceiveEvent(ULONG socketId, TCPIP_EVENT tcpEvent)...
case TCPIP_EVENT_CONN_INCOMING:
m_spTcpProt->Accept(socketId);
break;
```

Implementation of a TCP client:

Name	Description
AllocSocket [▶ 60]	Opens a socket.
Connect [▶ 61]	Starts connection establishment.
IsConnected [▶ 61]	Checks whether the connection was established successfully.
ReceiveData [▶ 59]	Is called when data are received.
SendData [▶ 63]	Can be used to send data.
FreeSocket [▶ 61]	On the Listen socket and all connection sockets for stopping.

Disconnection by the operating system in Promiscuous mode

If Promiscuous mode is active at the RT Ethernet adapter in the “Adapter” tab, any TCP connection attempts are blocked by the operating system, since it does not recognize a port opened in the TCP/UDP RT object.

7.4.1 Method ITcIoTcpProtocolRecv:ReceiveData

Is called by the TCP/UDP RT module as a callback to transfer data.

Syntax

```
HRESULT TCOMAPI ReceiveData(ULONG socketId, ULONG nData, PVOID pData)
```

Return value

Name	Type	Description
ReceiveData	HRESULT	Indicates success and must be provided accordingly by the implemented module.

Parameter

Name	Type	Description
socketId	ULONG	The socket on which data was received.
nData	ULONG	Number of data received.
pData	PVOID	Pointer to the received data.

7.4.2 Method ITcIoTcpProtocolRecv:ReceiveEvent

Is called by the TCP/UDP RT module as a callback if an event has occurred.

Syntax

```
HRESULT TCOMAPI ReceiveEvent(ULONG socketId, TCPIP_EVENT tcpEvent)
```

Return value

Name	Type	Description
ReceiveEvent	HRESULT	Indicates success and must be provided accordingly by the implemented module.

Parameter

Name	Type	Description
socketId	ULONG	The socket on which data was received.
tcpEvent	TCP_EVENT	An element of the Enum.

The enumeration TCP_EVENT refers to different events, which can occur with a TCP connection:

```
enum TCPIP_EVENT : ULONG {
TCPIP_EVENT_NONE = 0,
TCPIP_EVENT_ERROR = 1,
TCPIP_EVENT_RESET = 2,
TCPIP_EVENT_TIMEOUT = 3,
TCPIP_EVENT_CONN_ESTABLISHED = 4,
TCPIP_EVENT_CONN_INCOMING = 5,
TCPIP_EVENT_CONN_CLOSED = 6,
TCPIP_EVENT_CONN_IDLE = 7,
TCPIP_EVENT_DATA_RECEIVED = 8,
TCPIP_EVENT_DATA_SENT = 9,
TCPIP_EVENT_KEEP_ALIVE = 10,
TCPIP_EVENT_LINKCONNECT = 11,
TCPIP_EVENT_LINKDISCONNECT = 12
};
```

An implementation of the method should provide a switch case over all elements, so that the system can respond according to the event.

The application of events for a TCP server is described in the interface overview.

7.4.3 Method ITcIoTcpProtocol:AllocSocket

Allocates a socket.

Syntax

```
HRESULT TCOMAPI AllocSocket(ITcIoTcpProtocolRecv* ipRecv, ULONG& socketId)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▸ 68].

Parameter

Name	Type	Description
ipRecv	ITcIoTcpProtocolRecv	Pointer to the receiver (Recv) interface.
socketId	ULONG&	The generated socket.

7.4.4 Method ITcIoTcpProtocol:FreeSocket

Enables a socket.

Syntax

```
HRESULT TCOMAPI AllocSocket(ULONG socketId)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be enabled.

7.4.5 Method ITcIoTcpProtocol:Connect

Establishes a connection to a remote terminal.

Syntax

```
HRESULT TCOMAPI Connect(ULONG socketId, ULONG ipRemoteAddress, USHORT tcpPort)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.
ipRemoteAddress	ULONG	IP address of the remote terminal to be contacted. IP addresses are displayed with the most significant element in the last position. (Example: 192.168.2.1 -> 01 02 A8 C0)
tcpPort	USHORT	Port of the remote terminal to be contacted.

7.4.6 Method ITcIoTcpProtocol:IsConnected

Indicates whether a socket is connected (for inbound and outbound connections).

Syntax

```
HRESULT TCOMAPI IsConnected(ULONG socketId)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.

7.4.7 Method ITcIoTcpProtocol:Close

Closes a socket.

Syntax

```
HRESULT TCOMAPI Close(ULONG socketId)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be closed.

7.4.8 Method ITcIoTcpProtocol:Listen

Opens a TCP port for incoming connections. The application is described in the interface overview.

Syntax

```
HRESULT TCOMAPI Listen(ULONG socketId, USHORT tcpPort)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.
tcpPort	USHORT	The port which is scanned for incoming connections.

7.4.9 Method ITcIoTcpProtocol:Accept

Accepts income connections. The application is described in the interface overview.

Syntax

```
HRESULT TCOMAPI Accept(ULONG socketId)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.

7.4.10 Method ITcIoTcpProtocol:SendData

Sends data (server and client functionality).

Syntax

```
HRESULT TCOMAPI SendData(ULONG socketId, ULONG nData, PVOID pData, ULONG& nSendData)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.
nData	ULONG	Length of the data to be sent.
pData	PVOID	Pointer to the data to be sent.
nSendData	ULONG&	Returns the number of sent bytes. If this is smaller than nData, the data should be re-sent.

7.4.11 Method ITcIoTcpProtocol:CheckReceived

Must be called cyclical; ReceiveEvent and ReceiveData are used as callback in the context of this method (server and client functionality).

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

-

7.4.12 Method ITcIoTcpProtocol:GetRemoteIpAddr

Returns the remote IP address of a communication partner.

Syntax

```
HRESULT TCOMAPI GetRemoteIpAddr(ULONG socketId, ULONG& remoteIpAddr)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.

7.4.13 Method ITcIoTcpProtocol:GetFreeSendDataSize

Returns the number of free bytes in the TCP send buffer.

Syntax

```
HRESULT TCOMAPI GetRemoteIpAddr(ULONG socketId, ULONG& nData)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
socketId	ULONG	The socket to be used.
nData	ULONG&	Returns the free bytes in the buffer.

7.5 ARP/Ping: ITcIoArpPingProtocol(Recv)

The interfaces ITcIoArpPingProtocol and ITcIoArpPingProtocolRecv enable sending of ARP and Ping messages from the real-time environment.

A project that uses this interface contains a pointer to an ITcIoArpPingProtocol object and implements ITcIoArpPingProtocolRecv itself. ITcIoArpPingProtocolRecv serves as callback interface for receiving data from the TCP/UDP RT module within the application.

ITcIoArpPingProtocolRecv methods:

Name	Description
ArpReply [▶ 65]	Callback function that is invoked when an ArpReply message is received.
PingReply [▶ 65]	Callback function that is invoked when an PingReply message is received.

If these methods return S_OK, the packet is regarded as processed and is not forwarded to the operating system. If necessary, S_FALSE should be returned.

ITcIoArpPingProtocol methods:

Name	Description
ArpRequest [▸ 66]	Sends an ArpRequest
PingRequest [▸ 66]	Sends a PingRequest
RegisterReceiver [▸ 67]	Registering at the TCP/UDP RT module for receiving data.
UnregisterReceiver [▸ 67]	Unregistering at the TCP/UDP RT module for receiving data.
CheckReceived [▸ 68]	Must be called cyclically; ArpReply and PingReply are used as callback in the context of this method

To receive ARP or Ping data, registration is required by calling RegisterReceiver. This can be done in SetObjStateSO().

Data is provided by a callback of method ArpReceive or PingReceive from ITcIoArpPingProtocolRecv.

During the shutdown, all modules should unregister via UnregisterReceiver. This can be done in SetObjStateOS().

7.5.1 Method ITcIoArpPingProtocolRecv:PingReply

Callback function that is invoked when an PingReply message is received.

Syntax

```
HRESULT TCOMAPI PingReply(ULONG ipAddr, ULONG nData, PVOID pData, ETYPE_VLAN_HEADER* pVlan=0)
```

Return value

Type	Description
HRESULT	Indicates success and must be provided accordingly by the implemented module. If this is not S_OK, the response continues to be transferred to the operating system.

Parameter

Name	Type	Description
ipAddr	ULONG	The IP address of the search.
nData	ULONG	Number of bytes received.
pData	PVOID	Pointer to the received data.
pVlan	ETYPE_VLAN_HEADER*	ETYPE_VLAN_HEADER structure, see below.

The VLAN header represents information about the VLAN.

```
typedef struct _ETYPE_VLAN_HEADER
{
USHORT VLanType;
unsigned short VLanIdH   : 4;
unsigned short reserved1 : 1;
unsigned short Priority  : 3;
unsigned short VLanIdL   : 8;
} ETYPE_VLAN_HEADER, *PETYPE_VLAN_HEADER;
```

7.5.2 Method ITcIoArpPingProtocolRecv:ArpReply

Callback function that is invoked when an ArpReply message is received.

Syntax

```
HRESULT TCOMAPI ArpReply(ULONG ipAddr, ETHERNET_ADDRESS macAddr, ETYPE_VLAN_HEADER* pVlan=0)
```

Return value

Type	Description
HRESULT	Indicates success and must be provided accordingly by the implemented module. If this is not S_OK, the response continues to be transferred to the operating system.

Parameter

Name	Type	Description
ipAddr	ULONG	The IP address of the search.
macAddr	ETHERNET_ADDRESS	Determined MAC address.
pVlan	ETYPE_VLAN_HEADER*	ETYPE_VLAN_HEADER structure, see below.

The VLAN header represents information about the VLAN.

```
typedef struct _ETYPE_VLAN_HEADER
{
USHORT VLanType;
unsigned short VLanIdH   : 4;
unsigned short reserved1 : 1;
unsigned short Priority  : 3;
unsigned short VLanIdL   : 8;
} ETYPE_VLAN_HEADER, *PETYPE_VLAN_HEADER;
```

7.5.3 Method ITcIoArpPingProtocol:PingRequest

Sends a ping request.

Syntax

```
HRESULT TCOMAPI PingRequest(ULONG ipAddr, ULONG nData=0, PVOID pData=0, ETYPE_VLAN_HEADER* pVlan=0)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
ipAddr	ULONG	The IP address of the target.
nData	ULONG	Number of bytes received.
pData	PVOID	Pointer to the received data.
pVlan	ETYPE_VLAN_HEADER*	ETYPE_VLAN_HEADER structure, see below.

The VLAN header represents information about the VLAN.

```
typedef struct _ETYPE_VLAN_HEADER
{
USHORT VLanType;
unsigned short VLanIdH   : 4;
unsigned short reserved1 : 1;
unsigned short Priority  : 3;
unsigned short VLanIdL   : 8;
} ETYPE_VLAN_HEADER, *PETYPE_VLAN_HEADER;
```

7.5.4 Method ITcIoArpPingProtocol:ArpRequest

Sends an ARP request.

Syntax

```
HRESULT TCOMAPI ArpRequest(ULONG ipAddr, ETHERNET_ADDRESS* macAddr=0, ETYPE_VLAN_HEADER* pVlan=0)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
ipAddr	ULONG	The IP address of the target.
macAddr	ETHERNET_ADDRESS*	Restriction of the MAC address.
pVlan	ETYPE_VLAN_HEADER*	ETYPE_VLAN_HEADER structure, see below.

The VLAN header represents information about the VLAN.

```
typedef struct _ETYPE_VLAN_HEADER
{
USHORT VLanType;
unsigned short VLanIdH   : 4;
unsigned short reserved1 : 1;
unsigned short Priority  : 3;
unsigned short VLanIdL   : 8;
} ETYPE_VLAN_HEADER, *PETYPE_VLAN_HEADER;
```

7.5.5 Method ITcIoArpPingProtocol:RegisterReceiver

Registering at the TCP/UDP RT module for receiving responses (ARP / Ping).

Syntax

```
HRESULT TCOMAPI RegisterReceiver(ITcIoArpPingRecv* ipRecv)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
ipRecv	ITcIoArpPingRecv*	Pointer to the receiver (Recv) interface.

7.5.6 Method ITcIoArpPingProtocol:UnregisterReceiver

Unregistering at the TCP/UDP RT module for receiving responses (ARP / Ping).

Syntax

```
HRESULT TCOMAPI UnregisterReceiver(ITcIoArpPingRecv* ipRecv)
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▶ 68].

Parameter

Name	Type	Description
ipRecv	ITcIoArpPingRecv	Reference to the receiver to be unregistered

7.5.7 Method ITcIoArpPingProtocol:CheckReceived

Must be called cyclically; ArpReply and PingReply are used as callback in the context of this method.

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Return value

Type	Description
HRESULT	Indicates success, see Return values [▸ 68].

7.6 Return values

The interface functions have HRESULT as return values. The returned values are derived from the ADS Return Codes [▸ 70]. Their meaning for TF6311:

Value (Enum)	Value (Numeric)	Description
ADS_E_INVALIDPARM	0x9811070B	Socket not allocated/known, transferred pointer NULL
ADS_E_NOMOREHDLS	0x98110716	No free sockets available. Default: 32 see TCP/UDP RT TcCom Parameter [▸ 55]
ADS_E_INCOMPATIBLE	0x9811070E	Socket in wrong state. E.g. Connect() attempt, if a socket was previous used with Listen(); Close() without previous connection; Send() without connection; Socket Listen(), if a Listen() call was already issued.
ADS_E_INVALIDSTATE	0x98110712	TCP/UDP RT object is not in OP mode
ADS_E_INVALIDDATA	0x98110706	Problem with parameter. E.g. pData==NULL for SendData
ADS_E_EXISTS	0x9811070F	Port already used otherwise
ADS_E_PENDING	0x9811071E	Not all data were sent (SendData)
S_OK	0x0	Call successful. IsConnected(): Connection exists
S_FAIL	0x1	Call not successful, general error IsConnected(): Connection does not exist

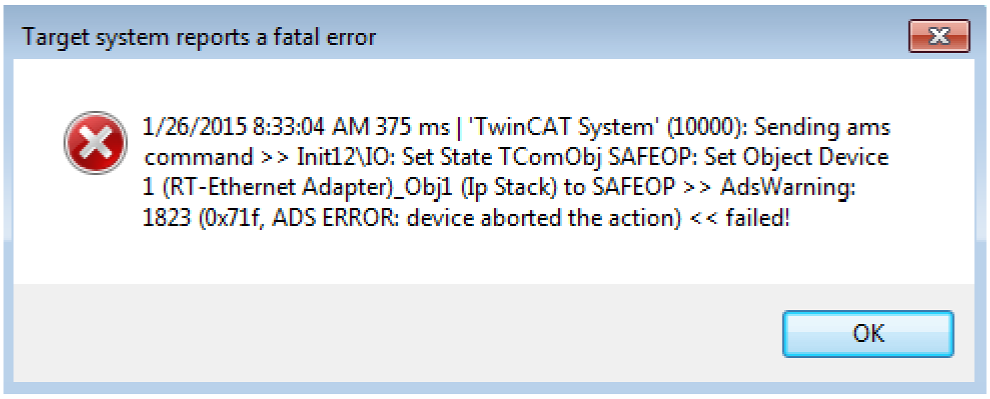
The values from the range 0x9811 are defined in the enumeration "E_HRESULTAdsErr" (PLC) and corresponding ADS_E_* (C++) "defines".

8 Fault analysis

At this point, it is usual practice to list problems or situations in connection with handling the product, together with an error description.

8.1 Start-up: Ip Stack ADS 1823 / 0x71f

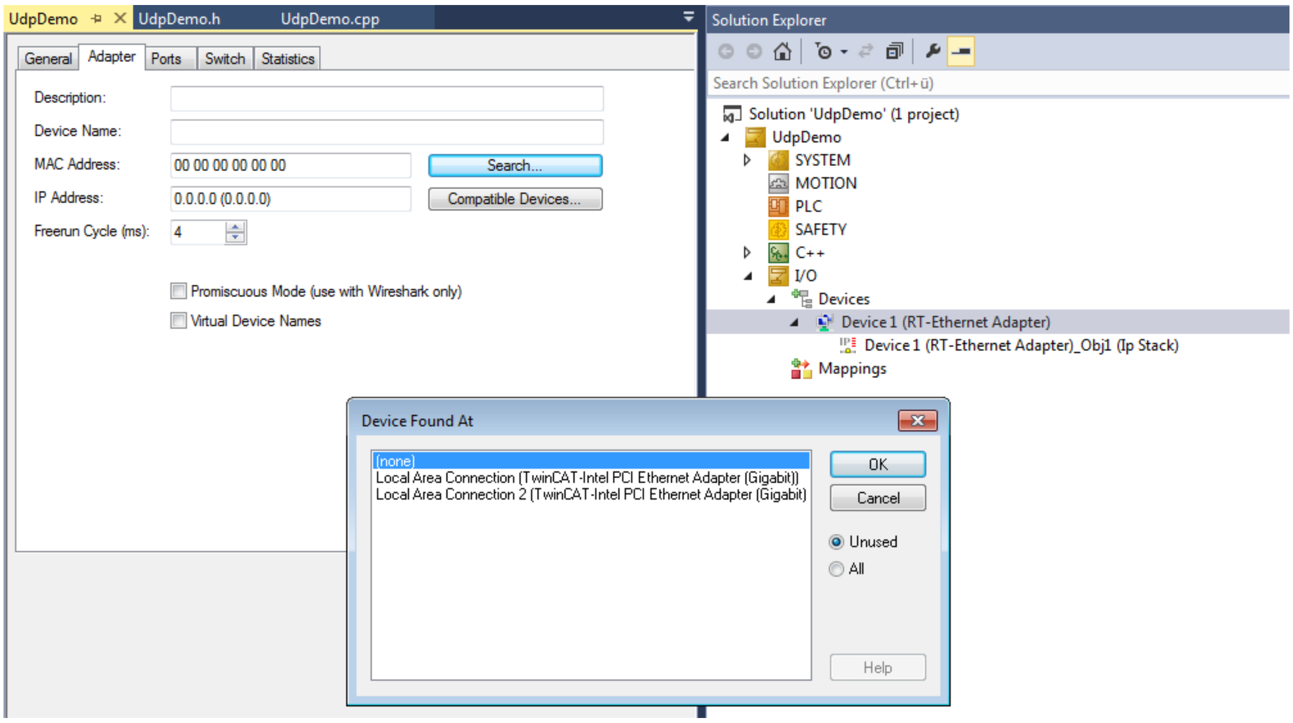
If ADS error 1823 (0x71f) occurs when an IP stack TcCOM object is started, the configuration of the network card is probably incorrect.



Target system reports a fatal error

```
1/26/2015 8:33:04 AM 375 ms | 'TwinCAT System' (10000): Sending ams
command >> Init12\IO: Set State TComObj SAFEOP: Set Object Device
1 (RT-Ethernet Adapter)_Obj1 (Ip Stack) to SAFEOP >> AdsWarning:
1823 (0x71f, ADS ERROR: device aborted the action) << failed!
```

Check the settings under "Adapter" for the network card in the Solution:



Description:	
Device Name:	
MAC Address:	00 00 00 00 00 00
IP Address:	0.0.0.0 (0.0.0.0)
Freerun Cycle (ms):	4

☐ Promiscuous Mode (use with Wireshark only)
☐ Virtual Device Names

(none)
Local Area Connection (TwinCAT-Intel PCI Ethernet Adapter (Gigabit))
Local Area Connection 2 (TwinCAT-Intel PCI Ethernet Adapter (Gigabit)

● Unused
○ All

The configuration of the network card for the TCP/UDP RT module is documented in more detail here [▶ 33].

9 Appendix

9.1 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 70]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 70]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 71]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 73]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was cancelled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

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

More Information:
www.beckhoff.com/tf6311