# Table of contents

1 Foreword ................................................................................................................................. 5
   1.1 Notes on the documentation .............................................................................................. 5
   1.2 Safety instructions ............................................................................................................... 6

2 Overview .................................................................................................................................... 7
   2.1 Comparison TF6310 TF6311 .............................................................................................. 7
   2.2 Restrictions ......................................................................................................................... 8

3 Installation / Licensing ............................................................................................................. 9

4 Quick Starts .............................................................................................................................. 10
   4.1 Quick Start (PLC / UDP) .................................................................................................... 10
   4.2 Quick Start (C++ / UDP) .................................................................................................. 15
   4.3 Quick Start (C++ / TCP Client) ........................................................................................ 21

5 Configuration .......................................................................................................................... 32
   5.1 Multiple network cards ....................................................................................................... 34
   5.2 Multitasking access to a network card ............................................................................... 35

6 Examples .................................................................................................................................. 37
   6.1 S01: Simple TCP Client (PLC / C++) ............................................................................... 37
   6.1.1 S01: Simple TCP Client (C++) ....................................................................................... 37
   6.1.2 S01: Simple TCP Client (PLC) ....................................................................................... 38
   6.2 S02: UDP Client Server (PLC/C++) .................................................................................. 39
   6.2.1 S02: UDP Demo (PLC) .................................................................................................. 39
   6.2.2 S02: UDP Demo (C++) ................................................................................................ 42
   6.2.3 Test client ....................................................................................................................... 43
   6.3 S03: ARP PING Demo (C++) ............................................................................................. 44
   6.4 S04: TCP Echo Server (PLC / C++) ................................................................................... 45
   6.4.1 S04: TCP Server Demo (PLC) ....................................................................................... 46
   6.4.2 S04: TCP Server Demo (C++) ....................................................................................... 48

7 Programmer's reference ........................................................................................................... 50
   7.1 TCP/UDP RT TcCom Parameter ......................................................................................... 51
   7.2 TCP/UDP RT TcCom diagnostics ..................................................................................... 53
   7.3 TCP/IP: ITcIoTcpProtocol(Recv) ...................................................................................... 54
   7.3.1 Method ITcIoTcpProtocolRecv:ReceiveData ................................................................. 56
   7.3.2 Method ITcIoTcpProtocolRecv:ReceiveEvent ............................................................... 56
   7.3.3 Method ITcIoTcpProtocol:AllocSocket ........................................................................ 57
   7.3.4 Method ITcIoTcpProtocol:FreeSocket ......................................................................... 57
   7.3.5 Method ITcIoTcpProtocol:Connect .............................................................................. 57
   7.3.6 Method ITcIoTcpProtocol:IsConnected ...................................................................... 58
   7.3.7 Method ITcIoTcpProtocol:Close ................................................................................... 58
   7.3.8 Method ITcIoTcpProtocol:Listen .................................................................................. 58
   7.3.9 Method ITcIoTcpProtocol:Accept ............................................................................... 59
   7.3.10 Method ITcIoTcpProtocol:SendData ........................................................................ 59
   7.3.11 Method ITcIoTcpProtocol:CheckReceived ................................................................. 60
   7.3.12 Method ITcIoTcpProtocol:GetRemotelnfAddr ............................................................ 60
   7.3.13 Method ITcIoTcpProtocol:GetFreeSendDataSize .................................................... 60
7.4 UDP/IP: ITcIoUdpProtocol(Recv) .............................................................. 61
  7.4.1 Method ITcIoUdpProtocolRecv:ReceiveData .................................... 61
  7.4.2 Method ITcIoUdpProtocol:SendData .................................................. 62
  7.4.3 Method ITcIoUdpProtocol:CheckReceived ........................................... 63
  7.4.4 Method ITcIoUdpProtocol:RegisterReceiver ....................................... 63
  7.4.5 Method ITcIoUdpProtocol:UnregisterReceiver .................................... 63
7.5 ARP/Ping: ITcIoArpPingProtocol(Recv) .................................................. 63
  7.5.1 Method ITcIoArpPingProtocolRecv:ArpReply .................................... 64
  7.5.2 Method ITcIoArpPingProtocolRecv:PingReply .................................... 65
  7.5.3 Method ITcIoArpPingProtocol:PingRequest ......................................... 65
  7.5.4 Method ITcIoArpPingProtocol:ArpRequest .......................................... 66
  7.5.5 Method ITcIoArpPingProtocol:RegisterReceiver .................................. 66
  7.5.6 Method ITcIoArpPingProtocol:UnregisterReceiver ............................... 67
  7.5.7 Method ITcIoArpPingProtocol:CheckReceived .................................... 67
7.6 Return values .................................................................................................. 67
8 Fault analysis ...................................................................................................... 69
  8.1 Start-up: Ip Stack ADS 1823 / 0x71f ....................................................... 69
9 Appendix ............................................................................................................ 70
  9.1 ADS Return Codes ...................................................................................... 70
1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSDE®, TC/BSDE®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

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

Copyright

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

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

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

Tip or pointer

This symbol indicates information that contributes to better understanding.
2 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. An introduction is provided by means of sample programs.

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 provide a good introduction.
- The configuration process is documented under Configuration.
- The interfaces are described in the Programmer’s reference and illustrated through samples.

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:
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 has been understood, further implementations can be derived in conjunction with the existing step-by-step instructions and samples.

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. 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 “…”:

3. The declaration part is created in this way:

```plaintext
FUNCTION_BLOCK SampleUdpEchoServer IMPLEMENTS ITcIoUdpProtocolRecv
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  {attribute 'TcInitSymbol'}
  oid: OTCID;
  ipUdp: ITcIoUdpProtocol;
  updPort: 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:

```plaintext
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:

```cpp
nReceivedPackets := nReceivedPackets + 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:
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^);
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

Note Variable names relating to TCP are used here. They have to be substituted accordingly.
12. Create the “TCP/UDP RT” module under the RT Ethernet adapter by selecting “Add Object(s)...” in the context menu.

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

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

14. Parameterize the previously created instance of the module (here: Module1) under “Interface Pointer” “TcpProt” with the OID of the created “TCP/UDP RT” object:
15. For PLC projects this configuration is also done at the instance, under the tab “Symbol Initialization”:

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. It can be observed that each call increments the counter. The client displays the returned packets at the top.

---

**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.  

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

1. A TwinCAT solution was generated
2. 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”.
3. Create a task. Under System / Tasks right-click and select “Add new Item…”
4. A normal task (without image) is sufficient.
5. In the C++ project, open the TMC editor by double-clicking on the TMC file.

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:

![Image of TMC editor with Implemented Interfaces and ITcIoUdpProtocolRecv selection]

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:

![Image of TMC editor with Interface Pointer creation and UdpProt selection]

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

```c++
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:

```c++
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, FLEAVE(A "UDP ReceiveData: IP: %d.%d.%d.%d udpSrcPort: %d DataSize: %d (hr2=%x) 
", ((PBYTE)&ipAddr)[3], ((PBYTE)&ipAddr)[2], ((PBYTE)&ipAddr)[1], ((PBYTE)&ipAddr)[0],
            udpSrcPort, nData, hr);
    return hr;
}
```

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.

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

“TCP/UDP RT” module configuration

*Note* Variable names relating to TCP are used here. They have to be substituted accordingly.
14. Create the “TCP/UDP RT” module under the RT Ethernet adapter by selecting “Add Object(s)...” in the context menu.
15. Then select the “TCP/UDP RT” module:

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

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

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 C++ module via the UDP sample Client. An output in the Visual Studio log can be generated by activating the corresponding TraceLevel (here at least TlInfo; cf. C++ tracing). The client displays the returned packets at the top.

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.

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
3. Select a Driver project

4. Use the wizard for a module class with "Cyclic IO" as the basis for the TCP client.
The result is a complete TwinCAT C++ project.

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".

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

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

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

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

```c
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
```

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

```c
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
}
```

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

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

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

```c
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
}
```

9. 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.

```c
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
HOST: 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;
}

10. 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
", 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 
", 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
}

11. 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
}

12. The module is now ready and can be compiled. (Right-click on "Build" project).
13. An instance of the module is created:

Right-click on the C++ project

and select the module

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

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).
“TCP/UDP RT” module configuration

**Note** 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.

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:

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:

   ![Configuration Interface](image)

   **Note:** 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:

   ![Log Messages](image)

   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.

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

   ![Tracelevel Configuration](image)
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 [51].

"TCP/UDP RT" module configuration

Note 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:

   ![Insert TCCom Object](image)

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

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:

   ![Parameterization](image)

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

   ![Symbol Initialization](image)

   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:

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

These objects have different object IDs:

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

Or for a C++ module:

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 to 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):

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.
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.

• 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.

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.

1. Unpack the downloaded ZIP file
2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
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 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).
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

1. Unpack the downloaded ZIP file
2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
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 “sLastReturnedMessageLength” indicates the whole length of the response.

Note: The server address is defined in the FB_init method.

The same sample is described in detail for C++ on the Quick Start [p. 21] 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).

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 [39] and C++ [42].

For testing purposes, a UDP client [43] (written in .NET) is also available.

The samples are also available in more detail as Quick Starts [10].

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

Downloading the sample.
1. Unpack the downloaded ZIP file
2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
3. Select your target system
4. Configure the network card (see below) for the target system
Examples

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 [61]` is implemented and a pointer to a `ITcIoUdpProtocol [61]` is used analogous to the **Quick Start [21]** in this sample.

To this end a PLC block is created, which implements the interface `ITcIoUdpProtocolRecv [61]` ("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:

![Diagram of UdpReceicer](image)

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
  pipItf^ := ITCUNKNOWN_TO_PVOID(ipUdpRecv);
  TcAddRef();
  TcQueryInterface := S_OK;
ELSIF GuidsEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_ITcUnknown)) THEN
  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).
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.
1. Unpack the downloaded ZIP file
2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
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` [61] is implemented and a pointer to a ` ITcIoUdpProtocol` [61] is used analogous to the Quick Start [21] 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 [15].

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).
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:

Together with the PLC / C++ examples, this results in an echo example: A UDP message is sent from the client port 11000 to the server port 10000, which returns the same data to the sender.
The client can be configured via the interface:

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

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**

- **Downloading** the sample.
  1. Unpack the downloaded ZIP file
  2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
  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) is implemented and a pointer to a **ARP/Ping**: ITcIoArpPingProtocol(Recv) is used analogous to the **Quick Start** 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.

**Concept**

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:

The output is in the messages:
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).

---

**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:
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

1. Unpack the downloaded ZIP file
2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
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 \[\text{ITcIoTcpProtocolRecv} \] is implemented and a pointer to a \[\text{ITcIoTcpProtocol} \] is used analogous to the Quick Start\[\text{10} \]s in this sample.

To this end a PLC block is created, which implements the interface \[\text{ITcIoUdpProtocolRecv} \] (“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”).
Examples

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 GuidesEqual(ADR(iid), ADR(TC_GLOBAL_IID_LIST.IID_ITcIoTcpProtocolRecv)) THEN
  ipTcpRecv := THIS^; // cast to interface pointer
  pipItf^ := ITCUNKNOWN_TO_PVOID(ipTcpRecv);
  TcAddRef();
  TcQueryInterface := S_OK;
ELSIF GuidesEqual(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).
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

Downloading the sample.

1. Unpack the downloaded ZIP file
2. Open the resulting tszip file in TwinCAT 3 via “Open Project …”
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).
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** [51]: 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)** [54]: TCP/IP protocol
- **ITcIoUdpProtocol(Recv)** [61]: UDP/IP protocol
- **ITcIoArpPingProtocol(Recv)** [63]: ARP/Ping protocol

**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):

Communication via the network interface depends on this cycle. There must be a corresponding call to the CheckReceived() methods (see API documentation [50]) 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 CheckReceived() 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 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:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IpAddress</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>SubnetMask</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>Gateway</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>DhcpEnable</td>
<td>FALSE</td>
</tr>
<tr>
<td>ManualSettings</td>
<td>FALSE</td>
</tr>
<tr>
<td>IpMaxReceivers</td>
<td>4</td>
</tr>
<tr>
<td>IpMaxPendingOnArp</td>
<td>40</td>
</tr>
<tr>
<td>IpMacCacheSize</td>
<td>64</td>
</tr>
<tr>
<td>IpMTU</td>
<td>1514</td>
</tr>
<tr>
<td>IpRecvFrameQueueSize</td>
<td>255</td>
</tr>
<tr>
<td>UdpMaxReceivers</td>
<td>4</td>
</tr>
<tr>
<td>UdpMTU</td>
<td>1514</td>
</tr>
<tr>
<td>UdpCheckCrc</td>
<td>TRUE</td>
</tr>
<tr>
<td>TTL</td>
<td>0x30</td>
</tr>
<tr>
<td>MulticastTTL</td>
<td>0x01</td>
</tr>
<tr>
<td>PassiveMode</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

### MulticastIpList

- `[]`

### TcpMTU

- `1514`

### TcpCheckCrc

- `TRUE`

### TcpMaxSocketCount

- `32`

### TcpReceiveBufferSize

- `16192`

### TcpTransmitBufferSize

- `16192`

### TcpMaxRetry

- `5`

### TcpTimeoutWait

- `60000`

### TcpTimeoutCon

- `5000`

### TcpTimeOutIdl

- `1000`

### TcpRoundTripTime

- `3000`

(Values in brackets indicate the default values.)

- **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): Not yet implemented. (Maximum transport unit size for UDP)
- 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 [35]
- 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 repetitions of TCP packets until the connection is terminated
- TcpTimeoutCon (5000): Timeout for TCP connection setup and disconnection
- TcpTimeoutWait (60000): Time period for storing handles internally after an unexpected termination of the connection
- TcpTimeoutIdle (1000): Time to callback (ReceiveEvent), 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)

### 7.2 TCP/UDP RT TcCom diagnostics

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

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:
Counters ("Cnt") are provided for the incoming and outgoing packets for the respective protocols.

In addition, the following events are counted:

- Changes in connection status of the network port (LinkStatusChanged)
- AllocFailed
- ArpTimeout
- Discarded packets (DroppedFrames)

### 7.3 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:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>ReceiveData</td>
<td>Is called by the TCP/UDP RT module as a callback to transfer data</td>
</tr>
<tr>
<td>🔄</td>
<td>ReceiveEvent</td>
<td>Is called by the TCP/UDP RT module as a callback if an event has occurred</td>
</tr>
</tbody>
</table>

### ITcIoTcpProtocol methods:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>AllocSocket</td>
<td>Allocates a socket</td>
</tr>
<tr>
<td>🔄</td>
<td>FreeSocket</td>
<td>Frees a socket</td>
</tr>
<tr>
<td>🔄</td>
<td>Connect</td>
<td>Establishes a connection to a remote terminal</td>
</tr>
<tr>
<td>🔄</td>
<td>IsConnected</td>
<td>Indicates whether a socket is connected (for inbound and outbound connections)</td>
</tr>
<tr>
<td>🔄</td>
<td>Close</td>
<td>Closes a socket</td>
</tr>
<tr>
<td>🔄</td>
<td>Listen</td>
<td>Opens a TCP port for incoming connections (see remarks)</td>
</tr>
<tr>
<td>🔄</td>
<td>Accept</td>
<td>For server functionality: Accepts incoming connections (see remarks)</td>
</tr>
<tr>
<td>🔄</td>
<td>SendData</td>
<td>Sends data (server and client functionality)</td>
</tr>
<tr>
<td>🔄</td>
<td>CheckReceived</td>
<td>Must be called cyclically; ReceiveEvent and ReceiveData are used as callback in the context of this method (server and client functionality)</td>
</tr>
<tr>
<td>🔄</td>
<td>GetRemoteIpAddr</td>
<td>Returns the remote IP address of a communication partner</td>
</tr>
<tr>
<td>🔄</td>
<td>GetFreeSendDataSize</td>
<td>Returns the number of free bytes in the TCP send buffer</td>
</tr>
</tbody>
</table>

### Remarks

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 a TCP server:
  - AllocSocket – opens a socket
  - Listen – opens a port on which connections are expected
  - Accept – is called in the ReceiveEvent() method in order to accept a connection
  - ReceiveData is called when data are received
  - SendData can be used to send data
  - FreeSocket on the Listen socket and all connection sockets for stopping

**Note** Call CheckReceived() continuously

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

Code diagram for accepting a connection:

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

- Implementation of a TCP client:
  - AllocSocket – opens a socket
  - Connect – starts connection establishment
  - IsConnected – checks whether the connection was established successfully
7.3.1 **Method ITcIoTcpProtocolRecv::ReceiveData**

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

**Syntax**

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

**Parameters**

- `socketId`: (type: ULONG) the socket on which the data were received
- `nData`: (type: ULONG) number of received data
- `pData`: (type: PVOID) pointer to the received data

**Return Value**

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

7.3.2 **Method ITcIoTcpProtocolRecv::ReceiveEvent**

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

**Syntax**

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

**Parameters**

- `socketId`: (type: ULONG) the socket on which the data were received
- `tcpEvent`: (type: TCP_EVENT) an enum element. See below

**Return Value**

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

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

```cpp
global 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.

### 7.3.3 Method ITcIoTcpProtocol:AllocSocket

Allocates a socket.

**Syntax**

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

**Parameters**

- `ipRecv`: (type: ITcIoTcpProtocolRecv*) pointer to the receiver (Recv) interface.
- `socketId`: (type: ULONG&) the generated socket

**Return Value**

- `HRESULT`: Identifies the success, cf. Return values [67]

### 7.3.4 Method ITcIoTcpProtocol:FreeSocket

Enables a socket.

**Syntax**

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

**Parameters**

- `socketId`: (type: ULONG) the socket to be enabled.

**Return Value**

- `HRESULT`: Identifies the success, cf. Return values [67]

### 7.3.5 Method ITcIoTcpProtocol:Connect

Establishes a connection to a remote station.
Syntax

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

Parameters

socketId: (type: ULONG) the socket to be used
ipRemoteAddress: (type: ULONG) IP address of the remote station to be contacted
tcpPort: (type: USHORT) port at the remote station to be contacted

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.6 Method ITcIoTcpProtocol: IsConnected

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

Syntax

HRESULT TCOMAPI IsConnected(ULONG socketId)

Parameters

socketId: (type: ULONG) the socket to be used

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.7 Method ITcIoTcpProtocol: Close

Closes a socket.

Syntax

HRESULT TCOMAPI Close(ULONG socketId)

Parameters

socketId: (type: ULONG) the socket to be closed

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.8 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)

Parameters

socketId: (type: ULONG) the socket to be used

tcpPort: (type: USHORT) the port which is scanned for incoming connections

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.9 Method ITcIoTcpProtocol: Accept

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

Syntax

HRESULT TCOMAPI Accept(ULONG socketId)

Parameters

socketId: (type: ULONG) the socket to be used

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.10 Method ITcIoTcpProtocol: SendData

Sends data (server and client functionality).

Syntax

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

Parameters

socketId: (type: ULONG) the socket to be used

nData: (type: ULONG) length of the data to be sent

pData : (type: PVOID) pointer to the data to be sent

nSendData: (type: ULONG&) returns the number of sent bytes. If this is smaller than nData, the data should be re-sent.

Return Value

HRESULT: Identifies the success, cf. Return values [67]
7.3.11  Method ITcIoTcpProtocol::CheckReceived

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

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Parameters

- 

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.12  Method ITcIoTcpProtocol::GetRemoteIpAddr

Returns the remote IP address of a communication partner.

Syntax

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

Parameters

socketId: (type: ULONG) the socket to be used

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.3.13  Method ITcIoTcpProtocol::GetFreeSendDataSize

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

Syntax

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

Parameters

socketId: (type: ULONG) the socket to be used

nData: (type: ULONG&) returns the free bytes in the buffer

Return Value

HRESULT: Identifies the success, cf. Return values [67]
7.4  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:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>ReceiveData</td>
<td>Is called by the TCP/UDP RT module as a callback to transfer data</td>
</tr>
</tbody>
</table>

ITcIoUdpProtocol methods:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>SendData</td>
<td>Sends data</td>
</tr>
<tr>
<td>🔄</td>
<td>CheckReceived</td>
<td>Must be called cyclically; ReceiveData is used as callback in the context of this method (server and client functionality)</td>
</tr>
<tr>
<td>🔄</td>
<td>RegisterReceiver</td>
<td>Registering at the TCP/UDP RT module for receiving data</td>
</tr>
<tr>
<td>🔄</td>
<td>UnregisterReceiver</td>
<td>Unregistering at the TCP/UDP RT module for receiving UDP data.</td>
</tr>
</tbody>
</table>

Remarks

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
  ◦ RegisterReceiver – opens a port for incoming data packets
  ◦ ReceiveData – is called when data packets arrive
  ◦ SendData – can be used to send data
  ◦ UnregisterReceiver – 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.

• For OnlineChange security, RegisterReceiver should be called again.

7.4.1  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)
Parameters

ipAddr: (type: ULONG) IP address of the sender
udpDestPort: (type: USHORT) port on which the data were received
udpSrcPort: (type: USHORT) port of the sender
nData: (type: ULONG) number of received bytes
pData : (type: PVOID) pointer to the received data
pVlan: (type: ETYPE_VLAN_HEADER*) structure ETYPE_VLAN_HEADER. See below

Return Value

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

The VLAN header provides information about the VLAN used for sending.

```
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;
```

7.4.2 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)
```

Parameters

ipAddr: (type: ULONG) IP address of the sender
udpDestPort: (type: USHORT) port on which the data were received
udpSrcPort: (type: USHORT) port of the sender
nData: (type: ULONG) number of bytes to be sent
pData: (type: PVOID) pointer to the data to be sent
bCalcUdpCheckSum: (type: bool) indicates whether the checksum should be calculated
pVlan: (type: ETYPE_VLAN_HEADER*) structure ETYPE_VLAN_HEADER. See below

Return Value

HRESULT: Identifies the success, cf. Return values [p 67]

The VLAN header provides information about the VLAN used for sending.

```
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;
```
7.4.3 Method ITcIoUdpProtocol:CheckReceived
Must be called cyclically; ReceiveData is used as callback in the context of this method (send and receive).

Syntax

```c
HRESULT TCOMAPI CheckReceived()
```

Parameters

- 

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.4.4 Method ITcIoUdpProtocol:RegisterReceiver
Registering at the TCP/UDP RT module for receiving data.

Syntax

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

Parameters

udpPort: (type: USHORT) port on which the data are to be received
ipRecv: (type: ITcIoUdpProtocolRecv*) pointer to the receiver (Recv) interface.

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.4.5 Method ITcIoUdpProtocol:UnregisterReceiver
Deregistering at the TCP/UDP RT module for receiving data.

Syntax

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

Parameters

udpPort: (type: USHORT) port at which the data should no longer be received

Return Value

HRESULT: Identifies the success, cf. Return values [67]

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:**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚀</td>
<td>ArpReply</td>
<td>Callback function that is invoked when an ArpReply message is received.</td>
</tr>
<tr>
<td>🚀</td>
<td>PingReply</td>
<td>Callback function that is invoked when an PingReply message is received.</td>
</tr>
</tbody>
</table>

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

**ITcIoArpPingProtocol methods:**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚀</td>
<td>ArpRequest</td>
<td>Sends an ArpRequest</td>
</tr>
<tr>
<td>🚀</td>
<td>PingRequest</td>
<td>Sends a PingRequest</td>
</tr>
<tr>
<td>🚀</td>
<td>RegisterReceiver</td>
<td>Registering at the TCP/UDP RT module for receiving data.</td>
</tr>
<tr>
<td>🚀</td>
<td>UnregisterReceiver</td>
<td>Deregistering at the TCP/UDP RT module for receiving data.</td>
</tr>
<tr>
<td>🚀</td>
<td>CheckReceived</td>
<td>Must be called cyclically; ArpReply and PingReply are used as callback in the context of this method</td>
</tr>
</tbody>
</table>

**Remarks**

- 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: ArpReply**

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

**Syntax**

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

**Parameters**

- ipAddr: (type: ULONG) IP address of the search
- macAddr: (type: ETHERNET_ADDRESS) determined MAC address
- pVlan: (type: ETYPE_VLAN_HEADER*) structure ETYPE_VLAN_HEADER. See below

**Return Value**

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.

The VLAN header provides information about the VLAN used for sending.
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:PingReply

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

Syntax

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

Parameters

ipAddr: (type: ULONG) IP address of the search
nData: (type: ULONG) number of received bytes
pData : (type: PVOID) pointer to the received data
pVlan: (type: ETYPE_VLAN_HEADER*) structure ETYPE_VLAN_HEADER. See below

Return Value

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.

The VLAN header provides information about the VLAN used for sending.

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)

Parameters

ipAddr: (type: ULONG) IP address of the target
nData: (type: ULONG) number of received bytes
pData : (type: PVOID) pointer to the received data
pVlan: (type: ETYPE_VLAN_HEADER*) structure ETYPE_VLAN_HEADER. See below
Return Value

HRESULT: Identifies the success, cf. Return values [67]

The VLAN header provides information about the VLAN used for sending.

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)

Parameters

ipAddr: (type: ULONG) IP address of the target
macAddr: (type: ETHERNET_ADDRESS*) restriction of the MAC address
pVlan: (type: ETYPE_VLAN_HEADER*) structure ETYPE_VLAN_HEADER. See below

Return Value

HRESULT: Identifies the success, cf. Return values [67]

The VLAN header provides information about the VLAN used for sending.

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(ITcIoArpPingProtocolRecv* ipRecv)

Parameters

ipRecv: (type: ITcIoArpPingProtocolRecv*) pointer to the receiver (Recv) interface.

Return Value

HRESULT: Identifies the success, cf. Return values [67]
7.5.6 Method ITcIoArpPingProtocol:UnregisterReceiver

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

Syntax

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

Parameters

Return Value

HRESULT: Identifies the success, cf. Return values [67]

7.5.7 Method ITcIoArpPingProtocol:CheckReceived

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

Syntax

```c
HRESULT TCOMAPI CheckReceived()
```

Parameters

Return Value

HRESULT: Identifies the success, cf. Return values [67]

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:
<table>
<thead>
<tr>
<th>Value (Enum)</th>
<th>Value (Numerical)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS_E_INVALID</td>
<td>0x98110 70B</td>
<td>Socket not allocated/known, transferred pointer NULL</td>
</tr>
</tbody>
</table>
| ADS_E_NOMOREHDL  | 0x98110 716       | No free sockets available. Default: 32  
see TCP/UDP RT TcCom Parameter [51]                                               |
| ADS_E_INCOMMAT   | 0x98110 70E       | 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_INVALIDDST | 0x98110 712       | TCP/UDP RT object is not in OP mode                                                                                                          |
| ADS_E_INVALIDDAT | 0x98110 706       | Problem with parameter. E.g. pData==NULL for SendData                                                                                       |
| ADS_E_EXIST      | 0x98110 70F       | Port already used otherwise                                                                                                                   |
| ADS_E_PENDING    | 0x98110 71E       | Not all data were sent (SendData)                                                                                                            |
| S_OK             | 0x0              | Call successful. IsConnected(): Connection exists                                                                                        |
| S_FAIL           | 0x1              | Call unsuccessful, 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.

Check the settings under “Adapter” for the network card in the project folder:

The configuration of the network card for the TCP/UDP RT module is documented in more detail here [32].
Appendix

9 Appendix

9.1 ADS Return Codes

Grouping of error codes: 0x000...70..., 0x500...70..., 0x700...71..., 0x1000...73...

Global error codes

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

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

**General ADS error codes**
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>1792</td>
<td>0x8911 0700</td>
<td>ADSERR_DEVICE_ERROR</td>
<td>General device error.</td>
</tr>
<tr>
<td>0x701</td>
<td>1793</td>
<td>0x8911 0701</td>
<td>ADSERR_DEVICE_SRVNOTSUPP</td>
<td>Service is not supported by the server.</td>
</tr>
<tr>
<td>0x702</td>
<td>1794</td>
<td>0x8911 0702</td>
<td>ADSERR_DEVICE_INVALIDGRP</td>
<td>Invalid index group.</td>
</tr>
<tr>
<td>0x703</td>
<td>1795</td>
<td>0x8911 0703</td>
<td>ADSERR_DEVICE_INVALIDOFFSET</td>
<td>Invalid index offset.</td>
</tr>
<tr>
<td>0x704</td>
<td>1796</td>
<td>0x8911 0704</td>
<td>ADSERR_DEVICE_INVALIDACCESS</td>
<td>Reading or writing not permitted.</td>
</tr>
<tr>
<td>0x705</td>
<td>1797</td>
<td>0x8911 0705</td>
<td>ADSERR_DEVICE_INVALIDSIZE</td>
<td>Parameter size not correct.</td>
</tr>
<tr>
<td>0x706</td>
<td>1798</td>
<td>0x8911 0706</td>
<td>ADSERR_DEVICE_INVALIDDATA</td>
<td>Invalid data values.</td>
</tr>
<tr>
<td>0x707</td>
<td>1799</td>
<td>0x8911 0707</td>
<td>ADSERR_DEVICE_NOTREADY</td>
<td>Device is not ready to operate.</td>
</tr>
<tr>
<td>0x708</td>
<td>1800</td>
<td>0x8911 0708</td>
<td>ADSERR_DEVICE_BUSY</td>
<td>Device is busy.</td>
</tr>
<tr>
<td>0x709</td>
<td>1801</td>
<td>0x8911 0709</td>
<td>ADSERR_DEVICE_INVALIDCONTEXT</td>
<td>Invalid operating context. This can result from use of ADS function blocks in different tasks. It may be possible to resolve this through multithreading synchronization in the PLC.</td>
</tr>
<tr>
<td>0x70A</td>
<td>1802</td>
<td>0x8911 070A</td>
<td>ADSERR_DEVICE_NOMEMORY</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td>0x70B</td>
<td>1803</td>
<td>0x8911 070B</td>
<td>ADSERR_DEVICE_INVALIDPARAM</td>
<td>Invalid parameter values.</td>
</tr>
<tr>
<td>0x70C</td>
<td>1804</td>
<td>0x8911 070C</td>
<td>ADSERR_DEVICE_NOTFOUND</td>
<td>Not found (files, ...).</td>
</tr>
<tr>
<td>0x70D</td>
<td>1805</td>
<td>0x8911 070D</td>
<td>ADSERR_DEVICE_SYNTAX</td>
<td>Syntax error in file or command.</td>
</tr>
<tr>
<td>0x70E</td>
<td>1806</td>
<td>0x8911 070E</td>
<td>ADSERR_DEVICE_INCOMPATIBLE</td>
<td>Objects do not match.</td>
</tr>
<tr>
<td>0x70F</td>
<td>1807</td>
<td>0x8911 070F</td>
<td>ADSERR_DEVICE_EXISTS</td>
<td>Object already exists.</td>
</tr>
<tr>
<td>0x710</td>
<td>1808</td>
<td>0x8911 0710</td>
<td>ADSERR_DEVICE_SYMBOLNOTFOUND</td>
<td>Symbol not found.</td>
</tr>
<tr>
<td>0x711</td>
<td>1809</td>
<td>0x8911 0711</td>
<td>ADSERR_DEVICE_SYMBOLVERSIONINVALID</td>
<td>Invalid symbol version. This can occur due to an online change. Create a new handle.</td>
</tr>
<tr>
<td>0x712</td>
<td>1810</td>
<td>0x8911 0712</td>
<td>ADSERR_DEVICE_INVALIDSTATE</td>
<td>Device (server) is in invalid state.</td>
</tr>
<tr>
<td>0x713</td>
<td>1811</td>
<td>0x8911 0713</td>
<td>ADSERR_DEVICE_TRANSMODENOTSUPP</td>
<td>AdsTransMode not supported.</td>
</tr>
<tr>
<td>0x714</td>
<td>1812</td>
<td>0x8911 0714</td>
<td>ADSERR_DEVICE_NOTIFYHINDINVALID</td>
<td>Notification handle is invalid.</td>
</tr>
<tr>
<td>0x715</td>
<td>1813</td>
<td>0x8911 0715</td>
<td>ADSERR_DEVICE_CLIENTUNKNOWN</td>
<td>Notification client not registered.</td>
</tr>
<tr>
<td>0x716</td>
<td>1814</td>
<td>0x8911 0716</td>
<td>ADSERR_DEVICE_NOMOREHDLS</td>
<td>No further notification handle available.</td>
</tr>
<tr>
<td>0x717</td>
<td>1815</td>
<td>0x8911 0717</td>
<td>ADSERR_DEVICE_INVALIDWATCHSIZE</td>
<td>Notification size too large.</td>
</tr>
<tr>
<td>0x718</td>
<td>1816</td>
<td>0x8911 0718</td>
<td>ADSERR_DEVICE_NOTINIT</td>
<td>Device not initialized.</td>
</tr>
<tr>
<td>0x719</td>
<td>1817</td>
<td>0x8911 0719</td>
<td>ADSERR_DEVICE_TIMEOUT</td>
<td>Device has a timeout.</td>
</tr>
<tr>
<td>0x71A</td>
<td>1818</td>
<td>0x8911 071A</td>
<td>ADSERR_DEVICE_NOINTERFACE</td>
<td>Interface query failed.</td>
</tr>
<tr>
<td>0x71B</td>
<td>1819</td>
<td>0x8911 071B</td>
<td>ADSERR_DEVICE_INVALIDINTERFACE</td>
<td>Wrong interface requested.</td>
</tr>
<tr>
<td>0x71C</td>
<td>1820</td>
<td>0x8911 071C</td>
<td>ADSERR_DEVICE_INVALIDCLSID</td>
<td>Class ID is invalid.</td>
</tr>
<tr>
<td>0x71D</td>
<td>1821</td>
<td>0x8911 071D</td>
<td>ADSERR_DEVICE_INVALIDOBJID</td>
<td>Object ID is invalid.</td>
</tr>
<tr>
<td>0x71E</td>
<td>1822</td>
<td>0x8911 071E</td>
<td>ADSERR_DEVICE_PENDING</td>
<td>Request pending.</td>
</tr>
<tr>
<td>0x71F</td>
<td>1823</td>
<td>0x8911 071F</td>
<td>ADSERR_DEVICE_ABORTED</td>
<td>Request is aborted.</td>
</tr>
<tr>
<td>0x720</td>
<td>1824</td>
<td>0x8911 0720</td>
<td>ADSERR_DEVICE_WARNING</td>
<td>Signal warning.</td>
</tr>
<tr>
<td>0x721</td>
<td>1825</td>
<td>0x8911 0721</td>
<td>ADSERR_DEVICE_INVALIDARRAYIDX</td>
<td>Invalid array index.</td>
</tr>
<tr>
<td>0x722</td>
<td>1826</td>
<td>0x8911 0722</td>
<td>ADSERR_DEVICE_SYMBOLNOTACTIVE</td>
<td>Symbol not active.</td>
</tr>
<tr>
<td>0x723</td>
<td>1827</td>
<td>0x8911 0723</td>
<td>ADSERR_DEVICE_ACCESSDENIED</td>
<td>Access denied.</td>
</tr>
<tr>
<td>0x724</td>
<td>1828</td>
<td>0x8911 0724</td>
<td>ADSERR_DEVICE_LICENSENOTFOUND</td>
<td>Missing license.</td>
</tr>
<tr>
<td>0x725</td>
<td>1829</td>
<td>0x8911 0725</td>
<td>ADSERR_DEVICE_LICENSEEXPIRED</td>
<td>License expired.</td>
</tr>
<tr>
<td>0x726</td>
<td>1830</td>
<td>0x8911 0726</td>
<td>ADSERR_DEVICE_LICENSEEXCEEDED</td>
<td>License exceeded.</td>
</tr>
<tr>
<td>0x727</td>
<td>1831</td>
<td>0x8911 0727</td>
<td>ADSERR_DEVICE_LICENSEINVALID</td>
<td>Invalid license.</td>
</tr>
<tr>
<td>0x728</td>
<td>1832</td>
<td>0x8911 0728</td>
<td>ADSERR_DEVICE_LICENSESYSTEMID</td>
<td>License problem: System ID is invalid.</td>
</tr>
<tr>
<td>0x729</td>
<td>1833</td>
<td>0x8911 0729</td>
<td>ADSERR_DEVICE_LICENSENOTTIMELIMIT</td>
<td>License not limited in time.</td>
</tr>
<tr>
<td>0x72A</td>
<td>1834</td>
<td>0x8911 072A</td>
<td>ADSERR_DEVICE_LICENSEFUTUREISSUE</td>
<td>License problem: Time in the future.</td>
</tr>
<tr>
<td>0x72B</td>
<td>1835</td>
<td>0x8911 072B</td>
<td>ADSERR_DEVICE_LICENSETIMELONG</td>
<td>License period too long.</td>
</tr>
<tr>
<td>0x72C</td>
<td>1836</td>
<td>0x8911 072C</td>
<td>ADSERR_DEVICE_EXCEPTION</td>
<td>Exception at system startup.</td>
</tr>
<tr>
<td>0x72D</td>
<td>1837</td>
<td>0x8911 072D</td>
<td>ADSERR_DEVICE_LICENSEDEPRECATED</td>
<td>License file read twice.</td>
</tr>
<tr>
<td>0x72E</td>
<td>1838</td>
<td>0x8911 072E</td>
<td>ADSERR_DEVICE_SIGNATUREINVALID</td>
<td>Invalid signature.</td>
</tr>
<tr>
<td>0x72F</td>
<td>1839</td>
<td>0x8911 072F</td>
<td>ADSERR_DEVICE_CERTIFICATEINVALID</td>
<td>Invalid certificate.</td>
</tr>
<tr>
<td>0x730</td>
<td>1840</td>
<td>0x8911 0730</td>
<td>ADSERR_DEVICE_LICENSEOEMNOTFOUND</td>
<td>Public key not known from OEM.</td>
</tr>
<tr>
<td>0x731</td>
<td>1841</td>
<td>0x8911 0731</td>
<td>ADSERR_DEVICE_LICENSERESTRICTED</td>
<td>License not valid for this system ID.</td>
</tr>
<tr>
<td>0x732</td>
<td>1842</td>
<td>0x8911 0732</td>
<td>ADSERR_DEVICE_LICENSEDEMODENIED</td>
<td>Demo license prohibited.</td>
</tr>
<tr>
<td>0x733</td>
<td>1843</td>
<td>0x8911 0733</td>
<td>ADSERR_DEVICE_INVALIDFCID</td>
<td>Invalid function ID.</td>
</tr>
<tr>
<td>0x734</td>
<td>1844</td>
<td>0x8911 0734</td>
<td>ADSERR_DEVICE_OUTOFRANGE</td>
<td>Outside the valid range.</td>
</tr>
<tr>
<td>0x735</td>
<td>1845</td>
<td>0x8911 0735</td>
<td>ADSERR_DEVICE_INVALIDALIGNMENT</td>
<td>Invalid alignment.</td>
</tr>
</tbody>
</table>
### TCP Winsock error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x73C</td>
<td>1846</td>
<td>WSAETIMEOUT</td>
<td>A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.</td>
</tr>
<tr>
<td>0x73D</td>
<td>1847</td>
<td>WSAECONNREFUSED</td>
<td>Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.</td>
</tr>
<tr>
<td>0x751</td>
<td>10065</td>
<td>WSAEHOSTUNREACH</td>
<td>No route to host - a socket operation referred to an unavailable host.</td>
</tr>
</tbody>
</table>

More Winsock error codes: Win32 error codes