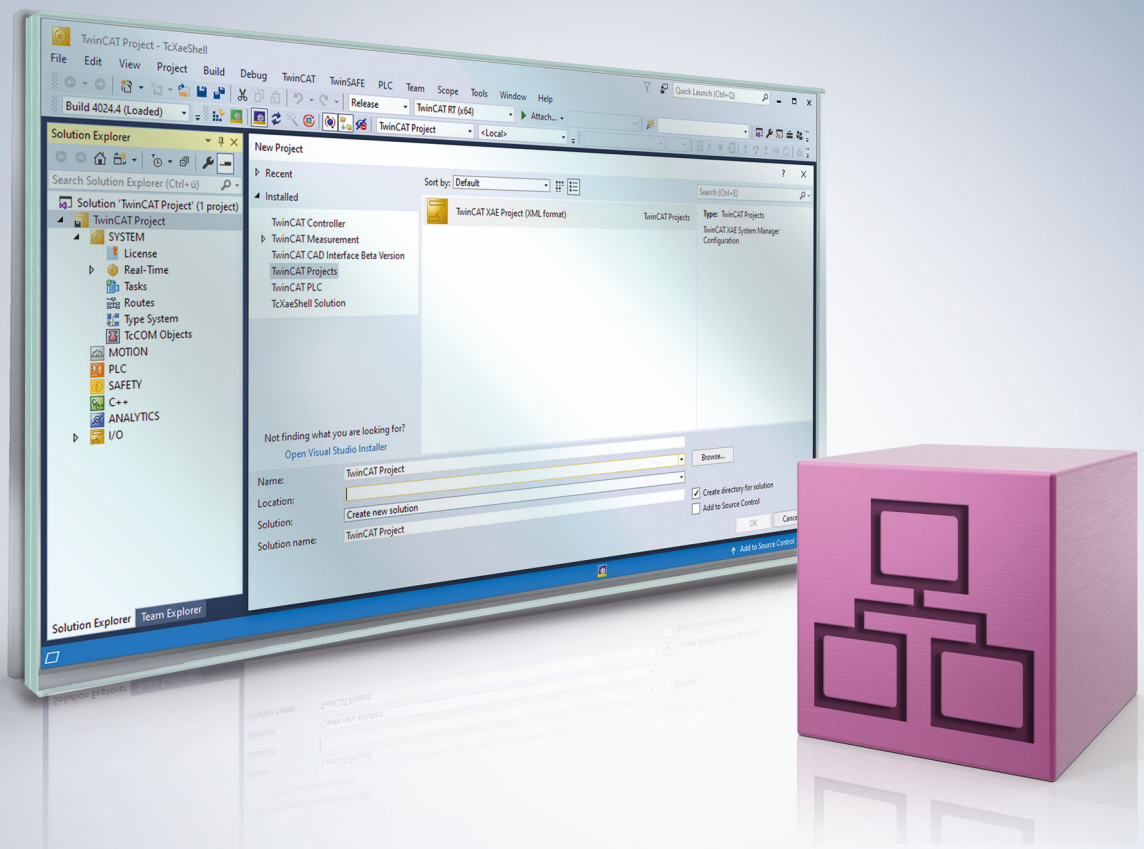
BECKHOFF New Automation Technology

Handbuch | DE

TF6311

TwinCAT 3 | TCP/UDP Realtime



28.11.2023 | Version: 1.8.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

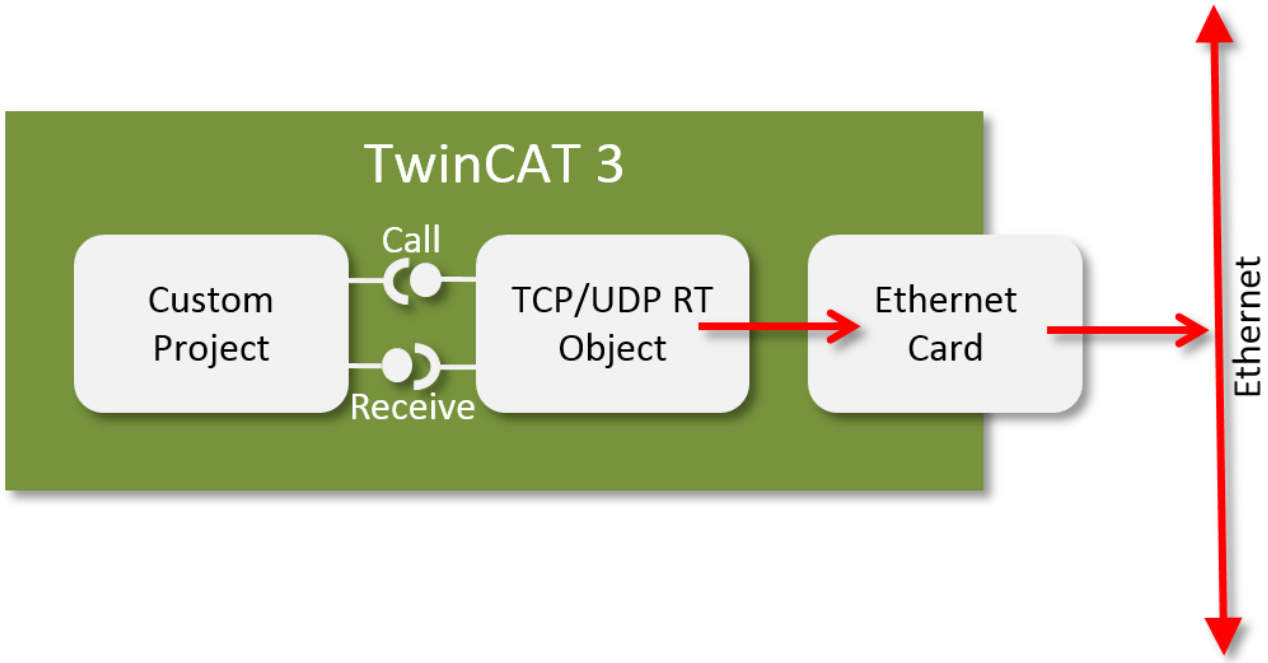
2 Übersicht

Die „TCP/UDP Realtime“ Function (TF6311) bietet den direkten Zugriff auf Netzwerkkarten aus der Echtzeitumgebung. Der Zugriff kann dabei entweder aus der PLC (61131-3) oder C++ erfolgen.

Die folgenden Protokolle werden unterstützt:

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

Hier werden die Schnittstellen als API [▶ 53] beschrieben und ein schneller Einstieg durch Beispielprogramme [▶ 40] ermöglicht.



Unabhängig vom Protokoll wird die Kommunikation zwischen nutzendem Projekt und TwinCAT durch ein Paar von Interfaces realisiert:

- Durch einen Interface Pointer wird die Möglichkeit gegeben Daten zu versenden, Verbindungen aufzubauen usw.
- Durch die Implementierung eines Receiver-Interfaces werden dem nutzenden Projekt durch Callbacks entsprechende Rückmeldungen in Form von Events oder Daten bereitgestellt.

Kommunikationspartner dieser Interface-Paare ist ein TcCom Objekt „TCP/UDP RT“, welches instanziiert und mit der Netzwerkkarte parametriert wird.

- Ein guter Einstiegspunkt je nach Protokoll sind die Quickstarts [▶ 11]
- Der Ablauf der Konfiguration ist unter Konfiguration [▶ 35] dokumentiert.
- Die Schnittstellen sind in der Programmierreferenz [▶ 53] beschrieben und durch Beispiele [▶ 40] erklärt.

2.1 Vergleich TF6310 TF6311

Die Produkte TF6310 „TCP/IP“ und TF6311 „TCP/UDP Realtime“ bieten eine ähnliche Funktionalität.

Auf dieser Seite werden Gemeinsamkeiten und Unterschiede der Produkte gegenübergestellt:

	TF 6310	TF 6311
TwinCAT	TwinCAT 2 / 3	TwinCAT 3
Client/Server	Beides	Beides
Große / Unbekannte Netzwerke	++	+
Determinismus	+	++
Großer Datentransfer	++	+
Programmiersprachen	PLC	PLC und C++
Betriebssystem	Win32/64, CE5/6/7	Win32/64, CE7
UDP-Mutlicast	Ja	Nein
Test-Lizenz	Ja	Ja
Protokolle	TCP, UDP	TCP, UDP, Arp/Ping
HW-Anforderungen	Beliebig	TwinCAT-kompatible Netzwerkkarte
Socket Konfiguration	Siehe Betriebssystem (WinSock)	TCP/UDP RT TcCom Parameter [▶ 57]

Da das TF6311 direkt im TwinCAT System integriert ist, kann die Windows Firewall nicht genutzt werden. In größeren / unbekannten Netzwerken empfiehlt es sich das TF6310 zu nutzen.

2.2 Einschränkungen

Es existieren folgende Einschränkungen des Produktes:

- keine lokale Kommunikation innerhalb der Echtzeit oder zwischen Echtzeit und Windows Betriebssystem. (Alternative: Kommunikation über eine zweite Netzwerkschnittstelle.)
- Multicasting wird nicht unterstützt.
- Die EL6601 und EL6614 können nicht für TF6311 TCP/UDP Realtime verwendet werden.
- Beim Arbeiten mit Breakpoints sollten unbedingt unterschiedliche Netzwerkschnittstellen genutzt werden, da ein Breakpoint Teile des TwinCAT Systems anhält, was auch die Kommunikation zum Engineering betreffen kann.

3 Installation / Lizenzierung

Die Function TF6311 benötigt keine separate Installation; nach Installation von TwinCAT 3 sind sämtliche Softwarekomponenten verfügbar.

- Es ist eine Lizenz „TC3 TCP UDP RT“ nötig.
  Die Abhängigkeit wird durch Hinzufügen des „TCP/UDP RT“ Objektes zum Projekt als Lizenz eingetragen. Sie kann auch manuell erfolgen.
- Eine Trial-Lizenz kann erstellt und genutzt werden.

4 Quick Starts

An dieser Stelle sind ausführliche Schritt-für-Schritt-Anleitungen für einige Protokolle dokumentiert. Sie stellen auf einfache Weise die Verwendung des Produktes dar. Die Beispiele sollen das Verständnis erleichtern; sie stellen keine umfassenden Implementierungen bereit. Auf Anwendungsebene muss der Umgang im Detail (z.B. das Verhalten bei entsprechend eintreffenden TCP Events) ausprogrammiert werden.

Die Function TF6311 „TCP/UDP Realtime“ hat weitreichende Fähigkeiten:

- unterschiedliche Protokolle (TCP, UDP, ARP/Ping)
- unterschiedliche Programmiersprachen (PLC / C++) und
- Kommunikationsrichtungen (Client / Server)

Nicht für alle Kombinationen existieren Schritt-für-Schritt-Anleitungen. Wenn das grundlegende Konzept [▶ 8] verstanden wurde, können weitergehende Implementierungen zusammen mit den vorhandenen Schritt-für-Schritt-Anleitungen und Beispielen [▶ 40] abgeleitet werden.

4.1 Quick Start (PLC / UDP)

Das Beispiel implementiert einen „Echo-Dienst“: Hierbei wird auf einem Port (Standard: 10000) ein UDP Server gestartet. Wenn dieser ein UDP Paket empfängt, sendet er den Inhalt zurück an den Absender (mit gleicher IP und gleichem Port).

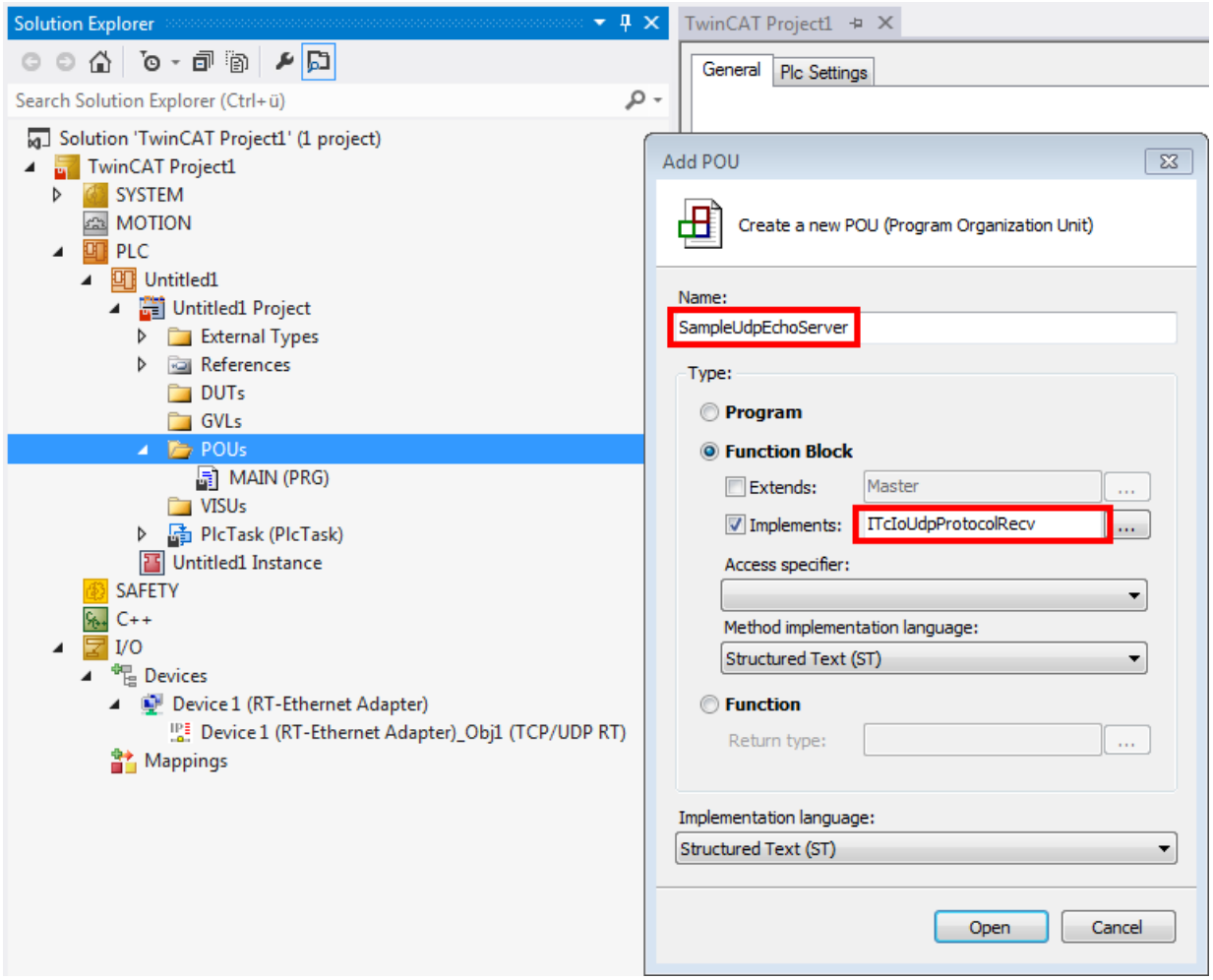
Das Beispiel ist auch als Download Sample 02 [▶ 42] verfügbar. Der Download enthält zusätzlich zum Quick Start hier erweiterten Code, der jedoch an der Funktionalität selber nichts ändert.

Implementierung des UDP Echo Servers in einem PLC Projekt

✓ Eine TwinCAT Solution wurde erzeugt

1. Wenn noch kein PLC Projekt in der TwinCAT Solution vorhanden ist, müssen Sie dieses anlegen.
2. Es wird ein Funktionsblock erzeugt, der das Interface „ITcIoUdpProtocolRecv“ implementiert. Hierdurch wird eine Methode erzeugt, die aufgerufen wird, falls UDP Pakete eintreffen.
   Per Rechts-Klick auf den Knoten „POU“ im PLC Projekt können Sie im Popup Namen vergeben

„SampleUdpEchoServer“, und „Implements“ per Haken aktivieren sowie das genannte Interface durch den Button „…“ auswählen:



Der Deklarationsteil des Funktionsbausteins erhält in der Deklaration einige Variablen:

- Oid: Konfigurierbare Referenz auf das TCP/UDP RT Modul
- ipUdp: Interface Pointer auf das UdpProtocol, welches vom TCP/UDP RT Modul implementiert wird
- udpPort: Port, der zum Empfang genutzt wird

3. Der Deklarationsteil wird somit angelegt:

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

Der Rumpf des Funktionsbausteins muss die CheckReceive() Methode des TCP/UDP RT Moduls aufgerufen werden.

4. Der Rumpf wird somit angelegt:

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

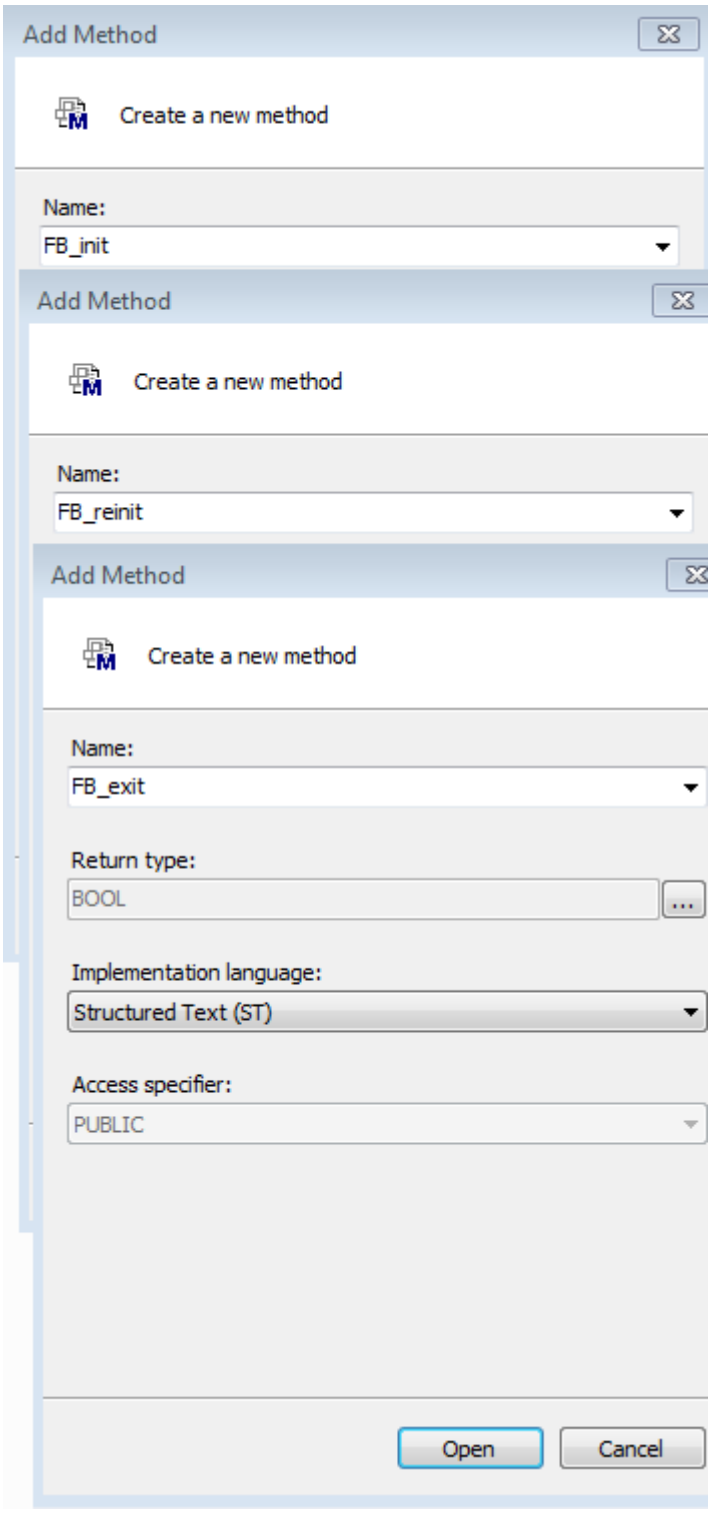
Die durch die Implementierung des Interfaces angelegte Methode “ReceiveData” wird durch das „CheckReceived“ mehrfach aufgerufen werden: Für jedes zwischenzeitlich empfangene Paket ein Aufruf.

5. Die Methode besitzt sowohl Absender-Informationen wie auch Daten als Eingangsparameter. In diesem Beispiel wird als Reaktion auf ein eintreffendes Paket das Paket (mit vertauschtem Absender/ Empfänger) durch die „SendData“ Methode wieder rückgesendet. Die Implementierung erfolgt folgendermaßen:

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

Beim Starten und Beenden muss aus der konfigurierten OID eine Referenz auf das „UdpProtocol“ Interface gesetzt werden; entsprechende Freigaben sind beim Runterfahren zu erledigen.

6. Der Baustein benötigt hierfür die Methoden „FB_init“ „FB_reinit“ und „FB_exit“, welche Sie durch einen Rechts-Click auf den Baustein „Add…“ Methode anlegen:



Die passenden Signaturen werden dabei automatisch generiert, sodass nur der eigentliche Rumpf realisiert werden muss. Wichtig ist dabei insbesondere der „RegisterReceiver“ Aufruf, der einen UDP Port zum Empfang öffnet.

7. In der „FB_init“ Methode werden zwei lokale Variablen benötigt:

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

8. Die Implementierung der „FB_init“ Methode erfolgt folgenderweise:

```
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 der „FB_reinit“ Methode, die beim OnlineChange ausgeführt wird, muss dem TCP/UDP RT Objekt die neue Adresse für die Callbacks mitgeteilt werden.

9. Die Implementierung der „FB_reinit“ Methode erfolgt folgenderweise:

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

Beim Herunterfahren (aber nicht beim OnlineChange vgl. bInCopyCode) muss der Port entsprechend wieder geschlossen werden.

10. Die Implementierung der „FB_exit“ Methode erfolgt folgenderweise:

```
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. Abschließend muss der Funktionsbaustein noch aufgerufen werden:

```
PROGRAM MAIN
VAR
        udp1 : SampleUdpEchoServer;
END_VAR

udp1();
```

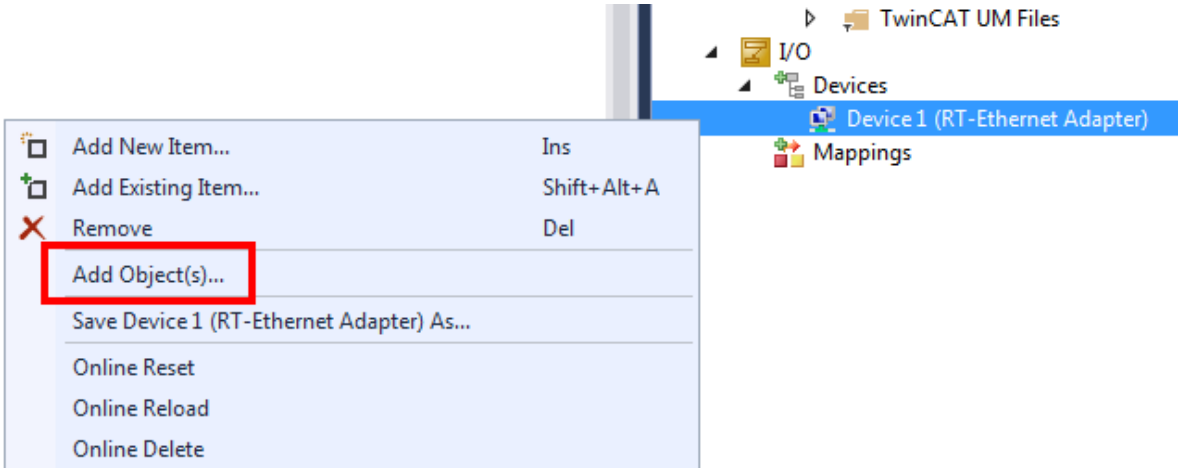
“TCP/UDP RT” Modul Konfiguration

HINWEIS

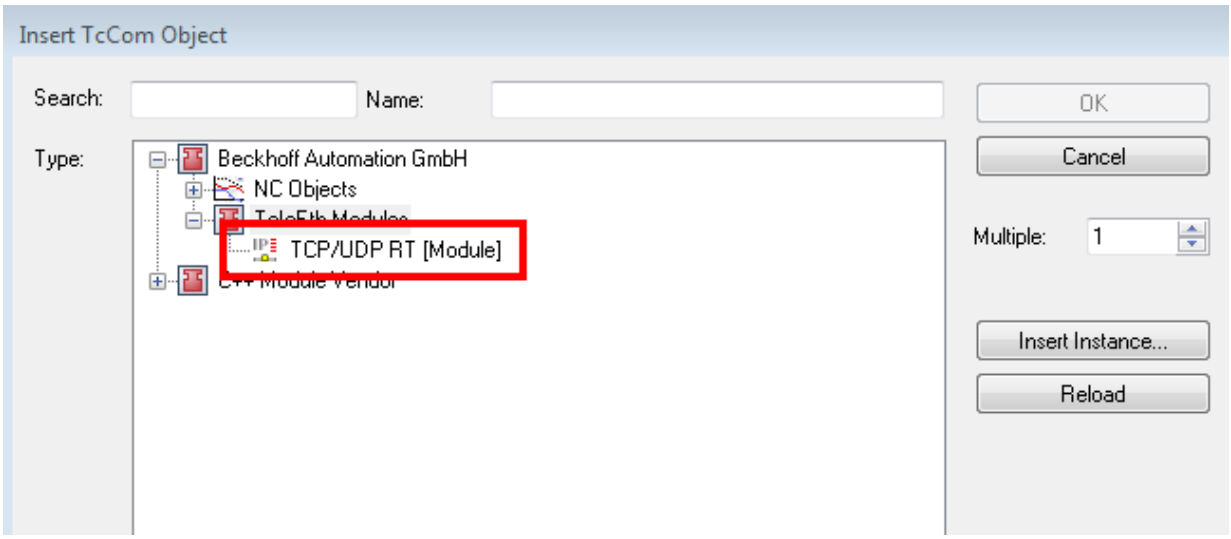
Variablennamen

Hier werden Variablennamen in Bezug auf TCP verwendet. Diese sind entsprechend zu ersetzen.

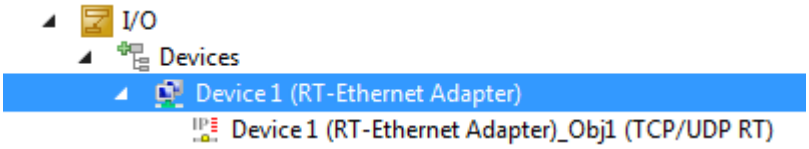
1. Legen Sie das „TCP/UDP RT“ Modul unterhalb des RT-Ethernet-Adapters an, indem Sie „Add Object(s)…“ im Context-Menü anwählen.



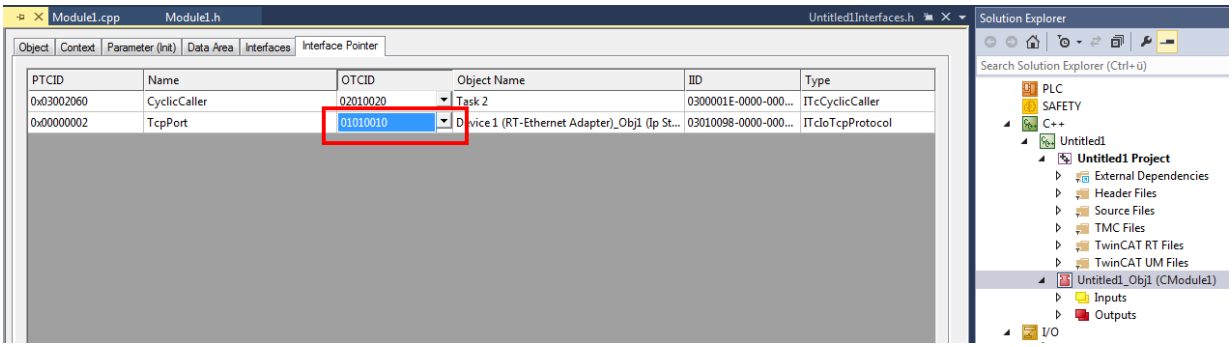
2. Dann wählen Sie das „TCP/UDP RT“ Modul aus:



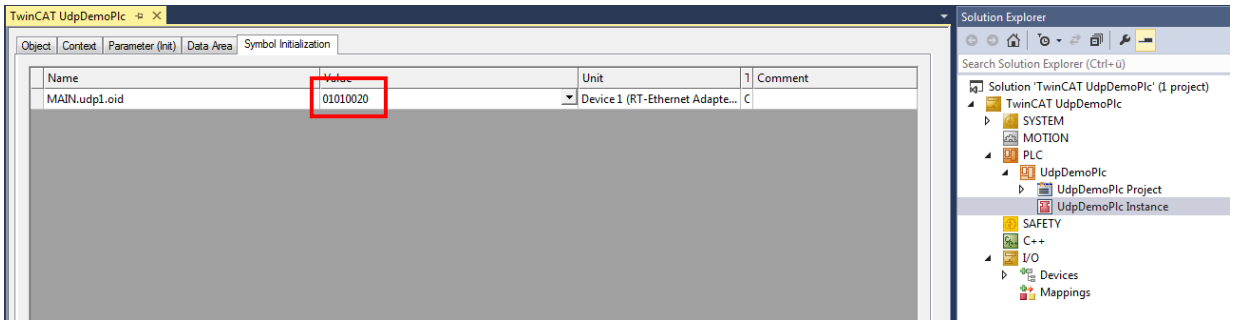
⇨ Das TCP/UDP RT Objekt wird unterhalb des Adapters angelegt.



3. Parametrieren Sie die zuvor angelegte Instanz des Moduls (hier: Modul1) unter „Interface Pointer“ „TcpProt“ mit der OID des angelegten „TCP/UDP RT“ Objekts:



4. Bei PLC Projekten ist diese Konfiguration ebenso an der Instanz vorzunehmen, hier jedoch unter dem Reiter „Symbol Initialization“:



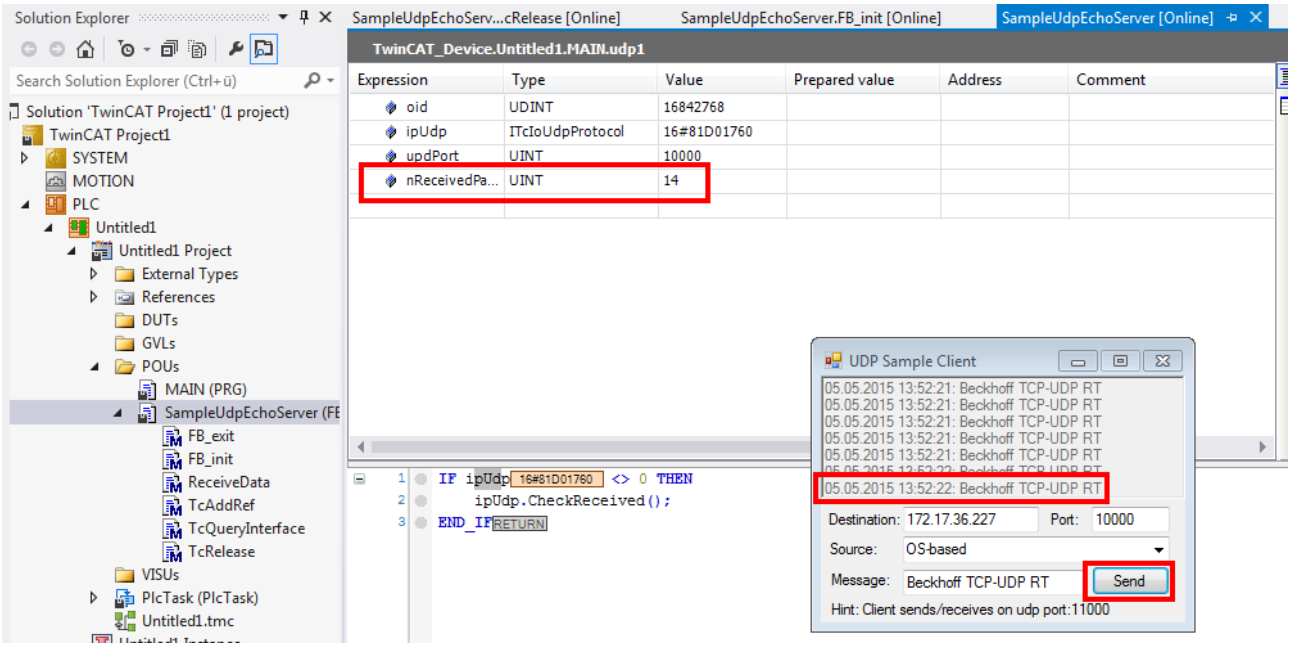
⇨ Damit ist die Konfiguration abgeschlossen

Verbindungsabbruch durch Betriebssystem bei Promiscuous Mode

Wenn an dem RT-Ethernet Adapter im Tab „Adapter“ der Promiscuous Mode eingeschaltet ist, werden eintreffende TCP Verbindungsaufbauten durch das Betriebssystem abgebrochen, da dieses einen im TCP/UDP RT Objekt geöffneten Port nicht kennt.

Testing

Nachdem die Konfiguration aktiviert wurde, kann mittels des UDP Sample Clients [▶ 46] ein UDP Paket zu der PLC gesendet werden. Es kann beobachtet werden, dass jeder Aufruf den Zähler erhöht. Der Client zeigt die zurückgesendeten Pakete im oberen Bereich an.



Keine lokale Kommunikation

Der UDP Sample Client muss auf einem anderen Rechner laufen als die PLC mit dem TCP/UDP RT Objekt, denn es ist keine lokale Kommunikation von dem Windows Betriebssystem in die Echtzeit möglich.
Möglich ist alternativ die Verwendung eines „Loop-Kabels“, welches 2 Netzwerk-Ports verbindet.
Der UDP Sample Client kann durch die Auswahl der Quelle (Dropdown „Source“) veranlasst werden einen spezifischen Port zu verwenden.

4.2 Quick Start (C++ / UDP)

Das Beispiel implementiert einen „Echo-Dienst“: Hierbei wird auf einem Port (Standard: 10000) ein UDP Server gestartet. Wenn dieser ein UDP Paket empfängt, sendet er den Inhalt zurück an den Absender (mit gleicher IP und gleichem Port).

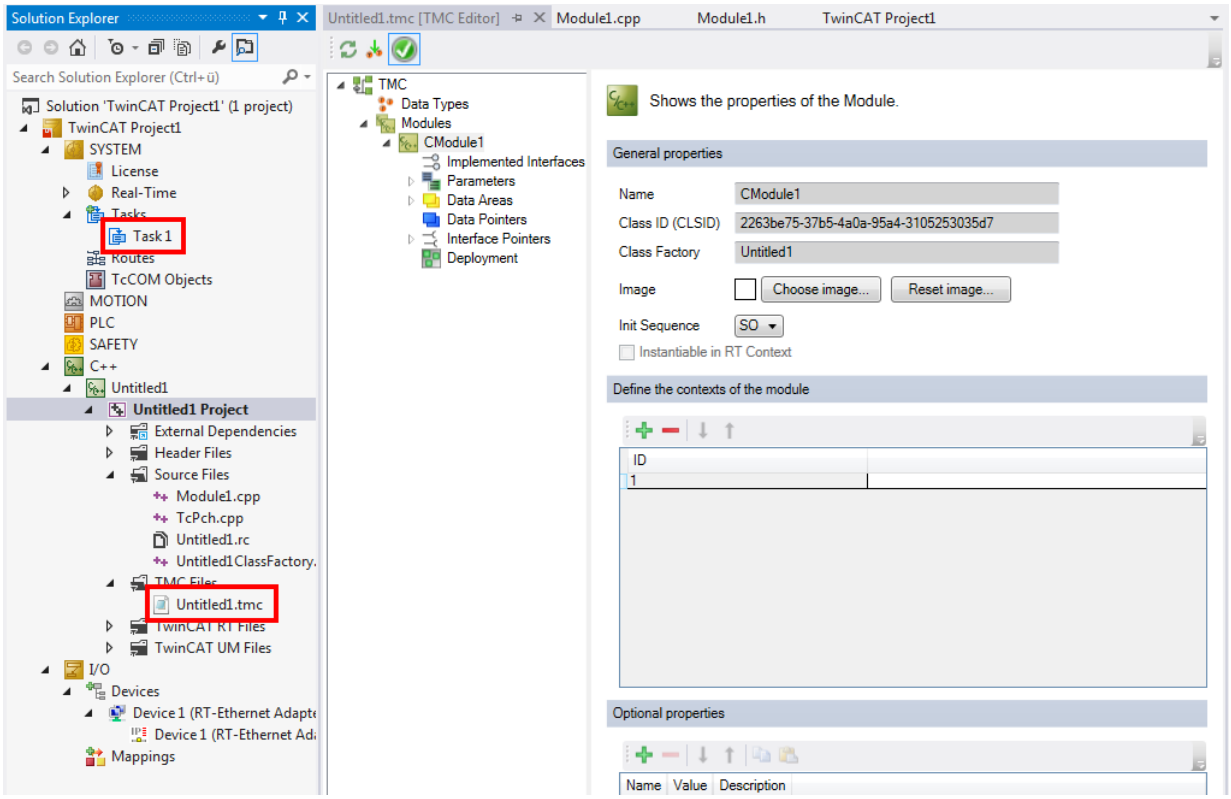
Das Engineering System muss dabei die Voraussetzungen für TwinCAT 3 C++ erfüllen.

Das Beispiel ist auch als Download Sample 02 [▶ 42] verfügbar.

Implementierung des UDP Echo Servers in einem C++ Projekt

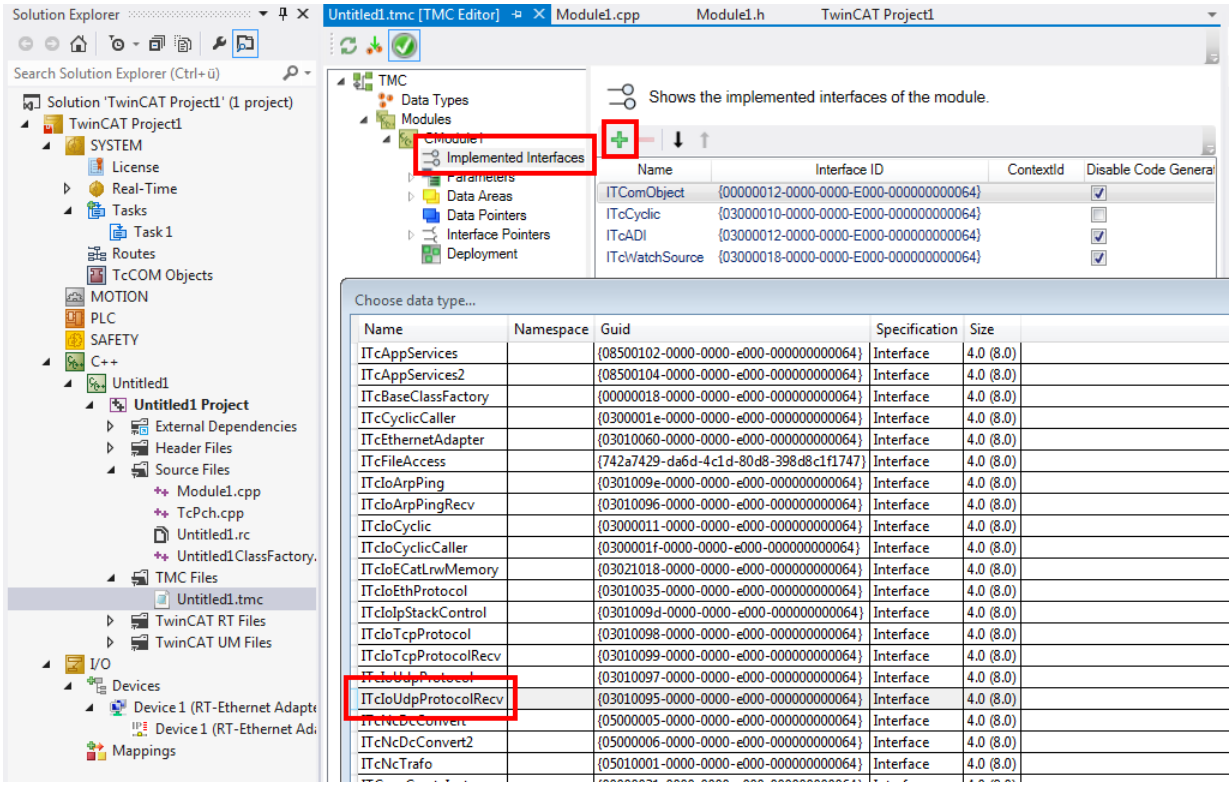
✓ Eine TwinCAT Solution wurde erzeugt

1. Wenn noch kein C++ Projekt in der TwinCAT Solution vorhanden ist, müssen Sie dieses angelegen. Verwenden Sie bitte die Vorlage für „TwiNCAT Module Class with Cyclic IO“.
2. Legen Sie ein Task an. Unter System / Tasks Rechts-Klick und „Add new Item…“ Ein normaler Task (ohne Image) ist ausreichend.
3. In dem C++ Projekt öffnen Sie den TMC Editor durch einen Doppel-Klick auf die TMC Datei.



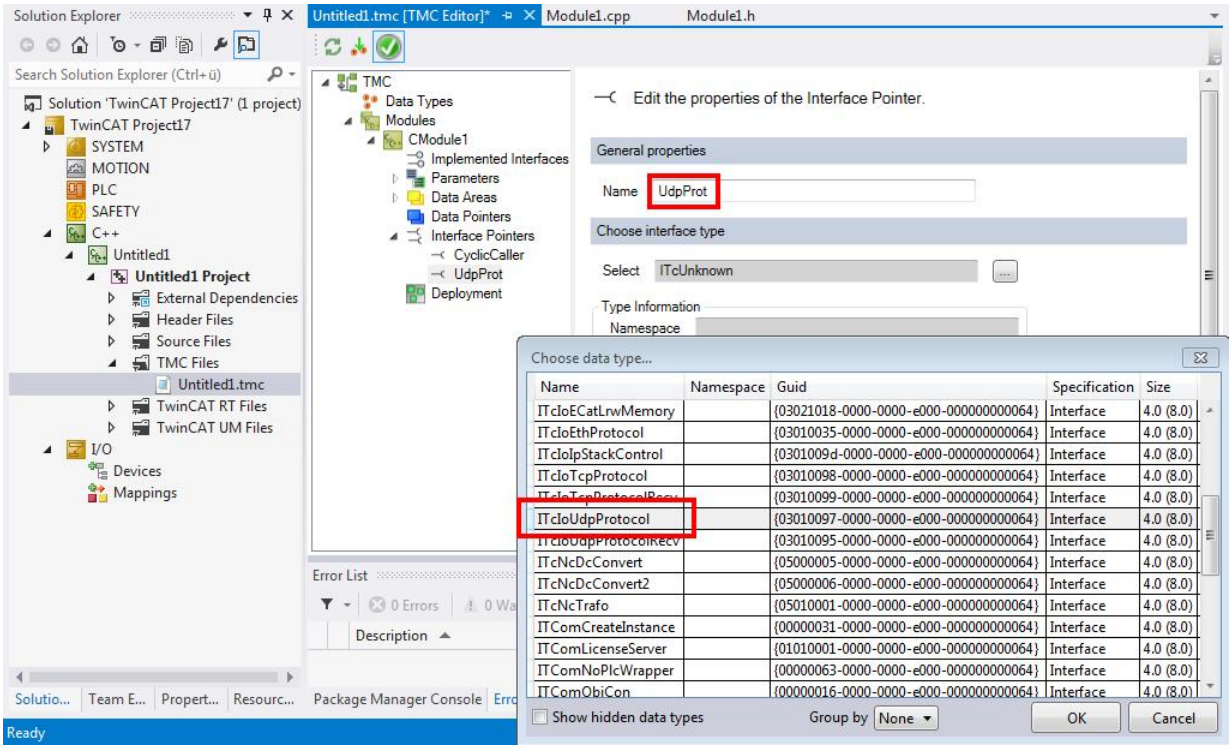
Das Modul muss das *ITcIoUdpProtocolRecv* implementieren. Hierdurch wird eine Methode erzeugt, die aufgerufen wird, falls UDP Pakete eintreffen.

4. Wählen Sie im TMC Editor „Implemented Interfaces“ legen diese per „+“ an. Es erscheint ein Dialog, in dem der Typ *ITcIoUdpProtocolRecv* ausgewählt wird:



Das Modul benötigt einen Interface Pointer zum *ITcIoUdpProtocol*, welches die Referenz auf das TCP/UDP RT Objekt beinhalten wird.

5. Wählen Sie im TMC Editor „Interface Pointer“ aus und drücken Sie “+”. Es wird ein Interface angelegt, welches per Doppel-Click geöffnet werden kann. Vergeben Sie einen Namen „UdpProt“ und setzen Sie den Typ des Pointers durch „..:“ und die Auswahl im Dialog:



6. Der TMC Codegenerator wird einmal gestartet. Rechts-Klick auf dem C++ Projekt und im Kontext Menü „TMC Code Generator“ auswählen.

In der CPP Datei des Moduls (Module1.cpp) muss in der CycleUpdate() Methode die Methode CheckReceived() des TCP/UDP RT Moduls aufgerufen werden. Hierdurch werden eintreffende UDP Pakete per Callback an die implementierte Methode ReceiveData() übergeben.

7. Die CycleUpdate() Methode wird folgendermaßen implementiert

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

Die durch die Implementierung des Interfaces angelegte Methode “ReceiveData” wird durch das CheckReceived() mehrfach aufgerufen werden: Für jedes zwischenzeitlich empfangene Paket ein Aufruf.

8. Die Methode ReceiveData besitzt sowohl Absender-Informationen wie auch Daten als Eingangsparameter. In diesem Beispiel wird als Reaktion auf ein eintreffendes Paket das Paket (mit vertauschtem Absender/Empfänger) durch die SendData Methode wieder rückgesendet. Die Implementierung erfolgt folgendermaßen:

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

Beim Starten und Beenden muss aus der konfigurierten OID eine Referenz auf das „UdpProtocol“ Interface gesetzt werden; entsprechende Freigaben sind beim Runterfahren zu erledigen.

9. Das Starten wird in der Transition SafeOp to Op durchgeführt. Insbesondere ist hierbei RegisterReceiver von Interesse: Er öffnet einen UDP Port zum Empfang.

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

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

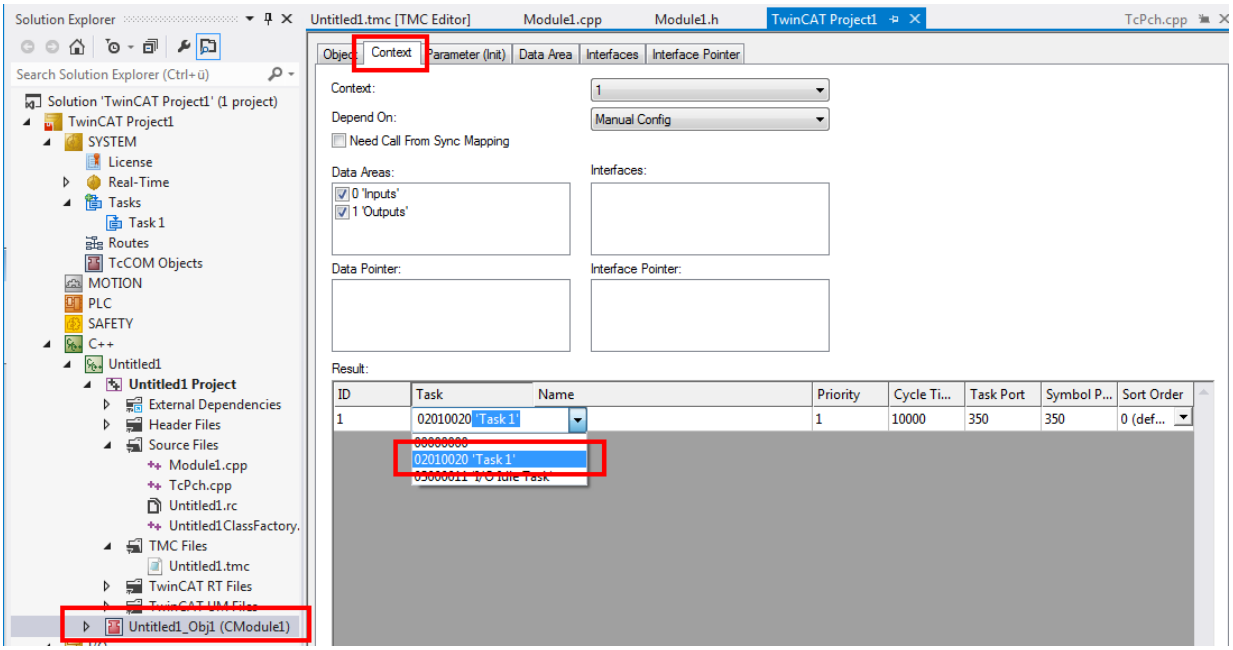
```
    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}
```

10. Das Beenden erfolgt in der Op to SafeOp Transition. Hier wird der UDP Port wieder geschlossen:

```
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;
}
```

Abschließend muss das Modul instanziiert und konfiguriert werden

11. Bauen Sie das Projekt einmal. Mit einem Rechts-Klick auf das Modul wählen Sie „Build“
12. Eine Instanz des Moduls anlegen. Mit einem Rechts-Klick auf das Projekt öffnet sich „Add new item…“. Wählen Sie hier das passende Modul aus.
13. Durch Doppel-Klick auf die Modul-Instanz wird die Parametrierung möglich. Wählen Sie zuerst im Tab „Context“ den Task aus.



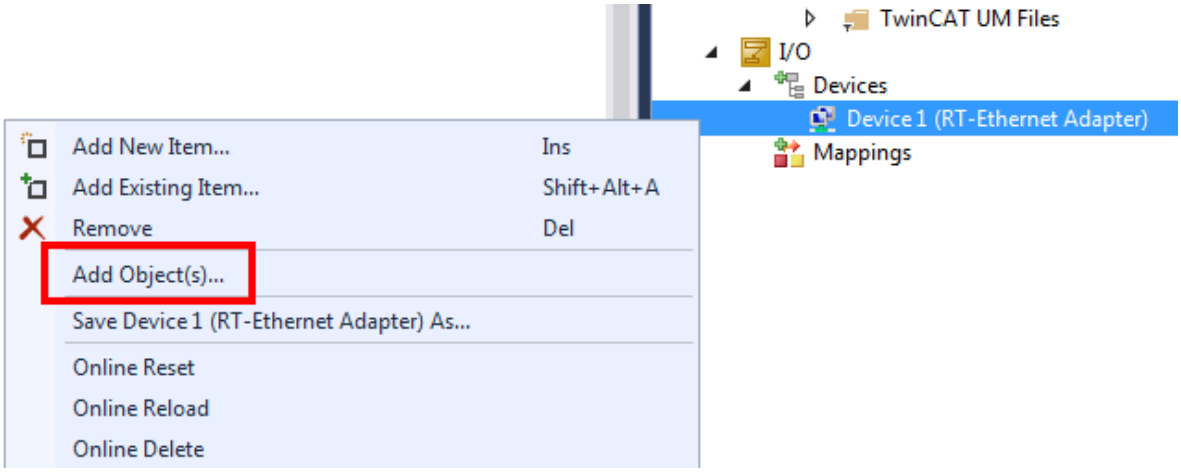
“TCP/UDP RT” Modul Konfiguration

HINWEIS

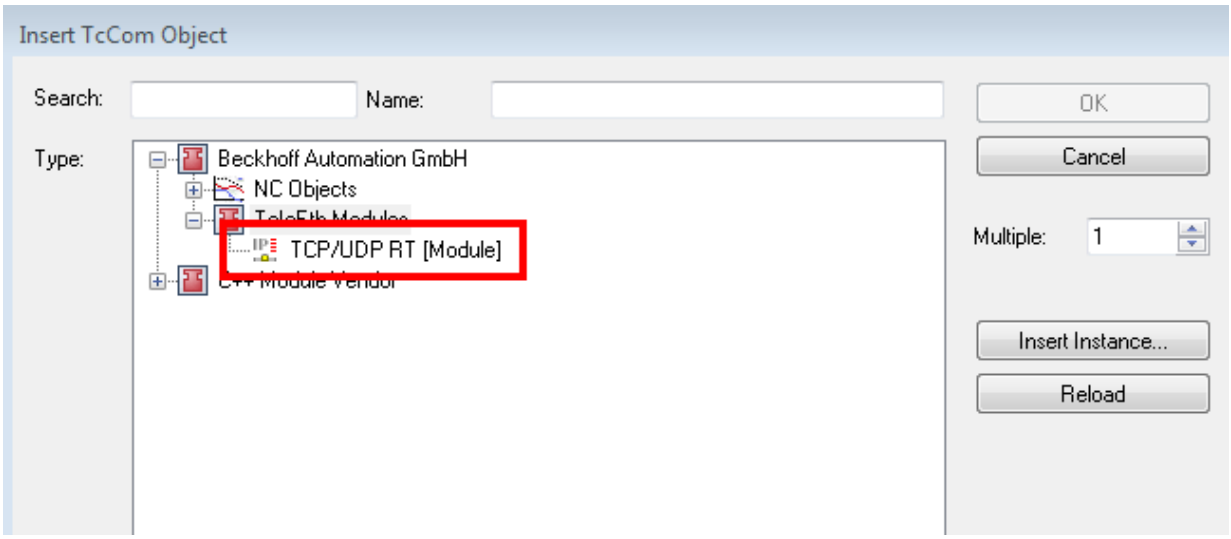
Variablennamen

Hier werden Variablennamen in Bezug auf TCP verwendet. Diese sind entsprechend zu ersetzen.

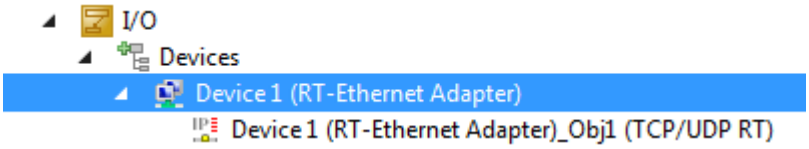
1. Legen Sie das „TCP/UDP RT“ Modul unterhalb des RT-Ethernet-Adapters an, indem Sie „Add Object(s)…“ im Context-Menü anwählen.



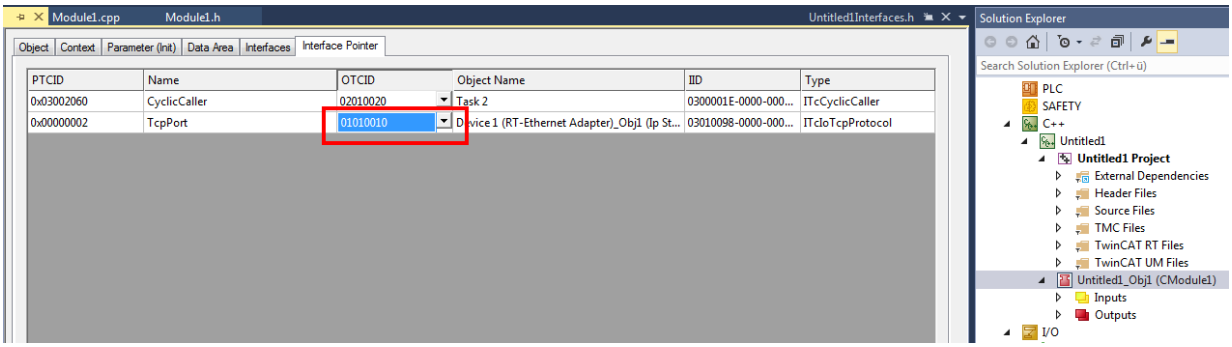
2. Dann wählen Sie das „TCP/UDP RT“ Modul aus:



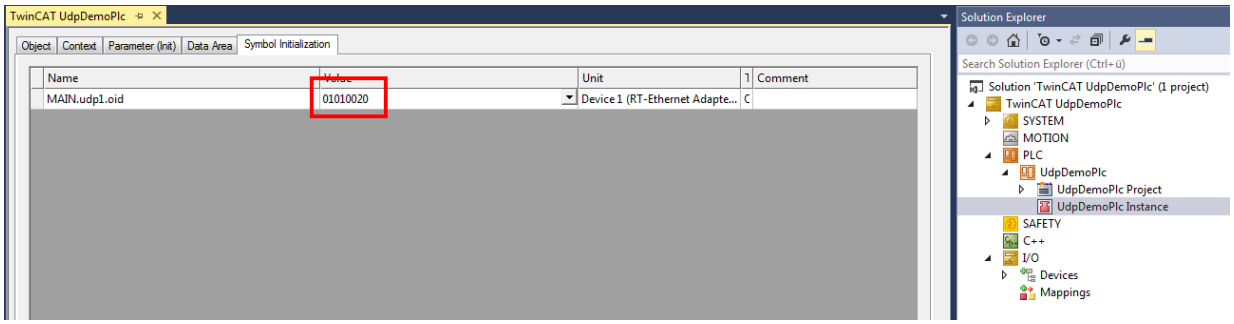
⇨ Das TCP/UDP RT Objekt wird unterhalb des Adapters angelegt.



3. Parametrieren Sie die zuvor angelegte Instanz des Moduls (hier: Modul1) unter „Interface Pointer“ „TcpProt“ mit der OID des angelegten „TCP/UDP RT“ Objekts:



4. Bei PLC Projekten ist diese Konfiguration ebenso an der Instanz vorzunehmen, hier jedoch unter dem Reiter „Symbol Initialization“:



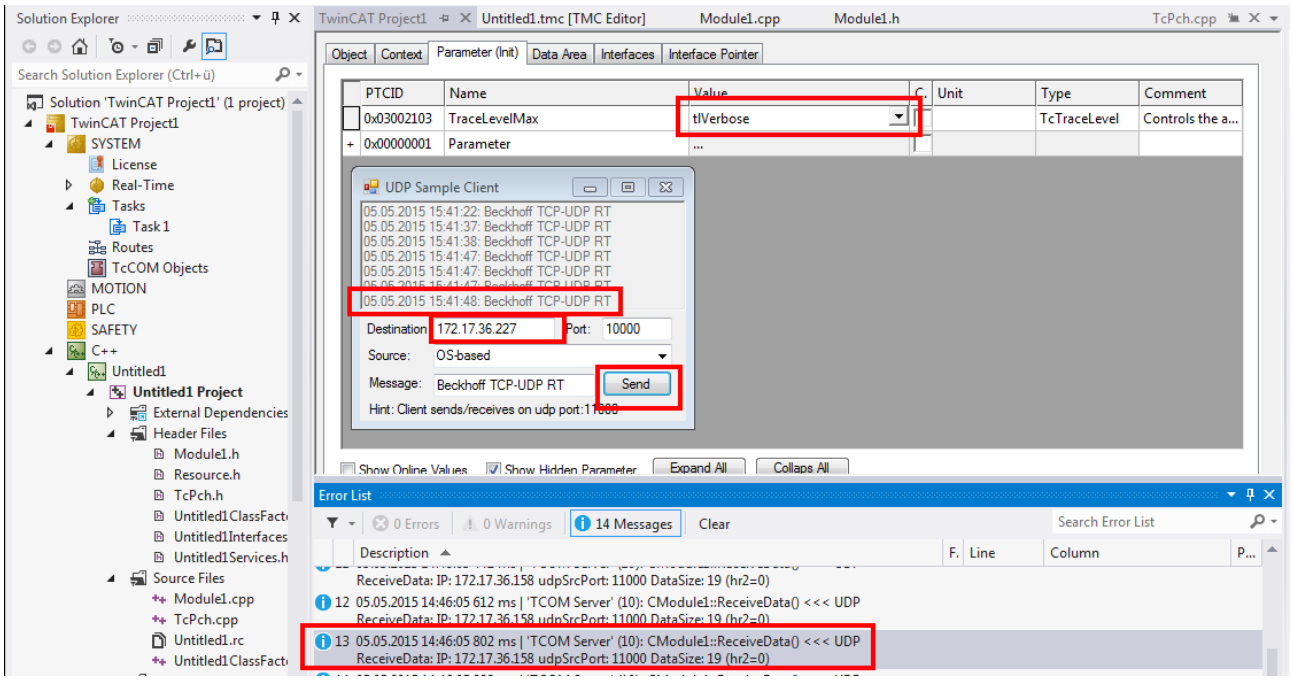
⇨ Damit ist die Konfiguration abgeschlossen

Verbindungsabbruch durch Betriebssystem bei Promiscuous Mode

Wenn an dem RT-Ethernet Adapter im Tab „Adapter“ der Promiscuous Mode eingeschaltet ist, werden eintreffende TCP Verbindungsaufbauten durch das Betriebssystem abgebrochen, da dieses einen im TCP/UDP RT Objekt geöffneten Port nicht kennt.

Testing

Nachdem die Konfiguration aktiviert wurde, kann mittels des UDP Sample Clients [▶ 46] ein UDP Paket zu dem C++ Modul gesendet werden. Durch aktivieren des entsprechenden TraceLevels (hier mindestens tlInfo; vgl. C++ Tracing) kann eine Ausgabe im Log des Visual Studios erzeugt werden. Der Client zeigt die zurückgesendeten Pakete im oberen Bereich an.



Keine lokale Kommunikation

Der UDP Sample Client muss auf einem anderen Rechner laufen als die PLC mit dem TCP/UDP RT Objekt, denn es ist keine lokale Kommunikation von dem Windows Betriebssystem in die Echtzeit möglich.
Möglich ist alternativ die Verwendung eines „Loop-Kabels“, welches 2 Netzwerk-Ports verbindet.
Der UDP Sample Client kann durch die Auswahl der Quelle (Dropdown „Source“) veranlasst werden einen spezifischen Port zu verwenden.

4.3 Quick Start (C++ / TCP Client)

Dieses Quick Start zeigt die Implementierung eines TCP Clients als TwinCAT 3 C++ Projekt.

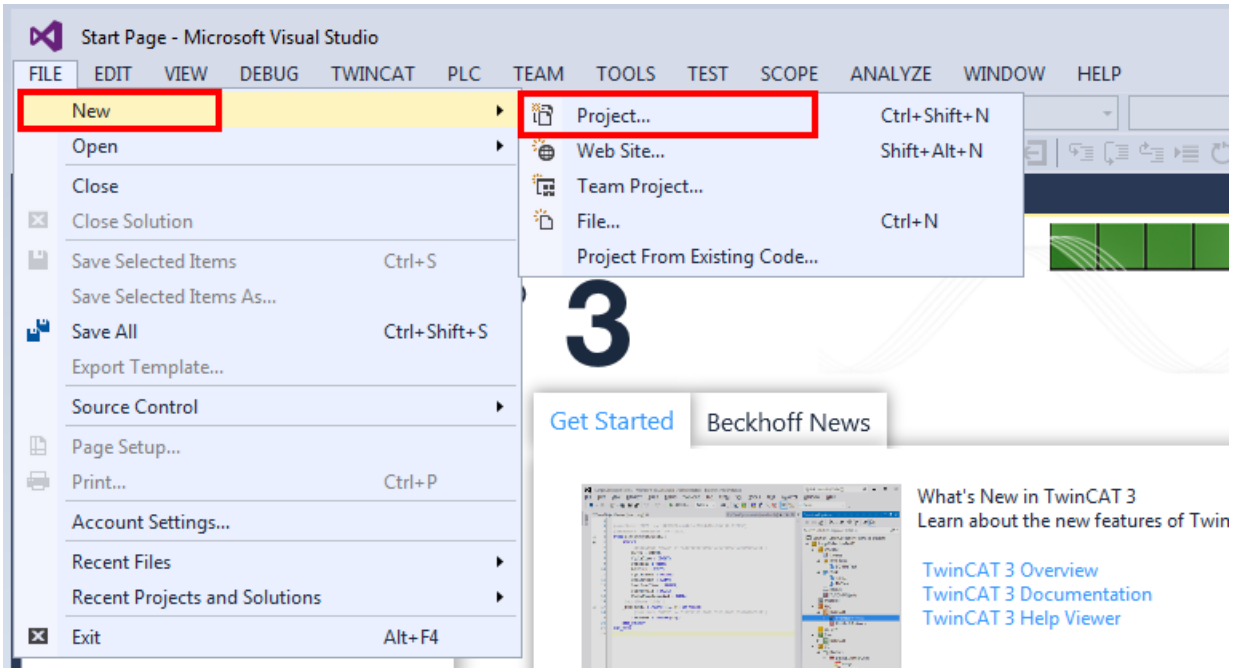
Das Engineering System muss dabei die Voraussetzungen für TwinCAT 3 C++ erfüllen.

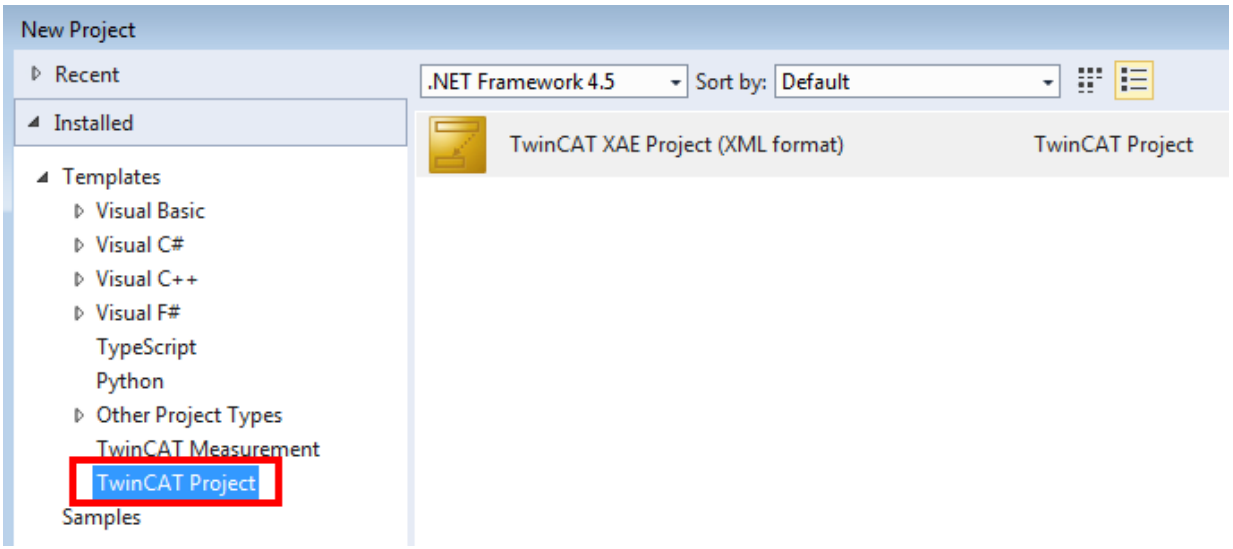
Das Beispiel ist auch als Download Sample 01 [▶ 40] verfügbar.

TwinCAT C++ Projekt anlegen

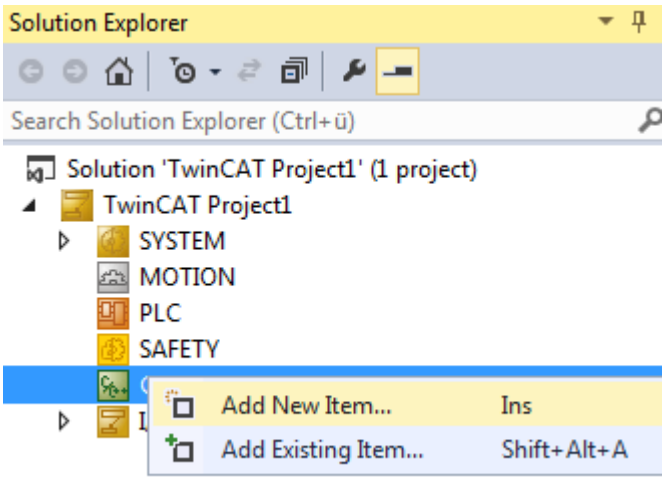
In diesem Arbeitsschritt wird ein neues TwinCAT 3 C++ Projekt angelegt.

1. Legen Sie ein neues TwinCAT Projekt an

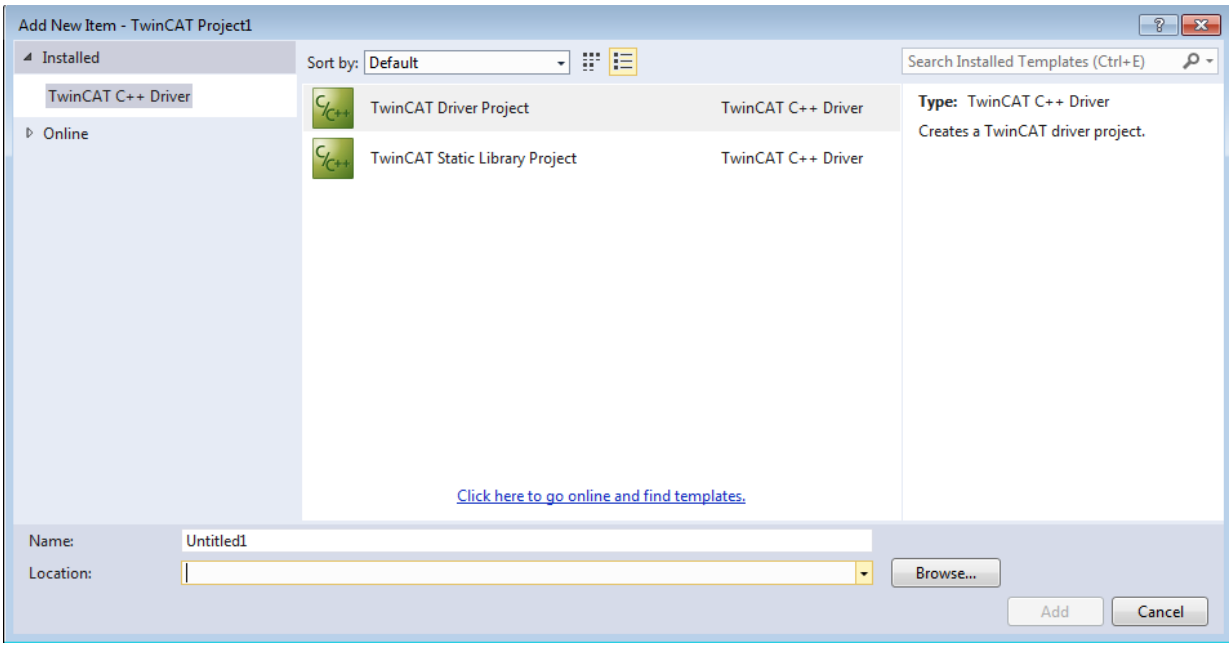




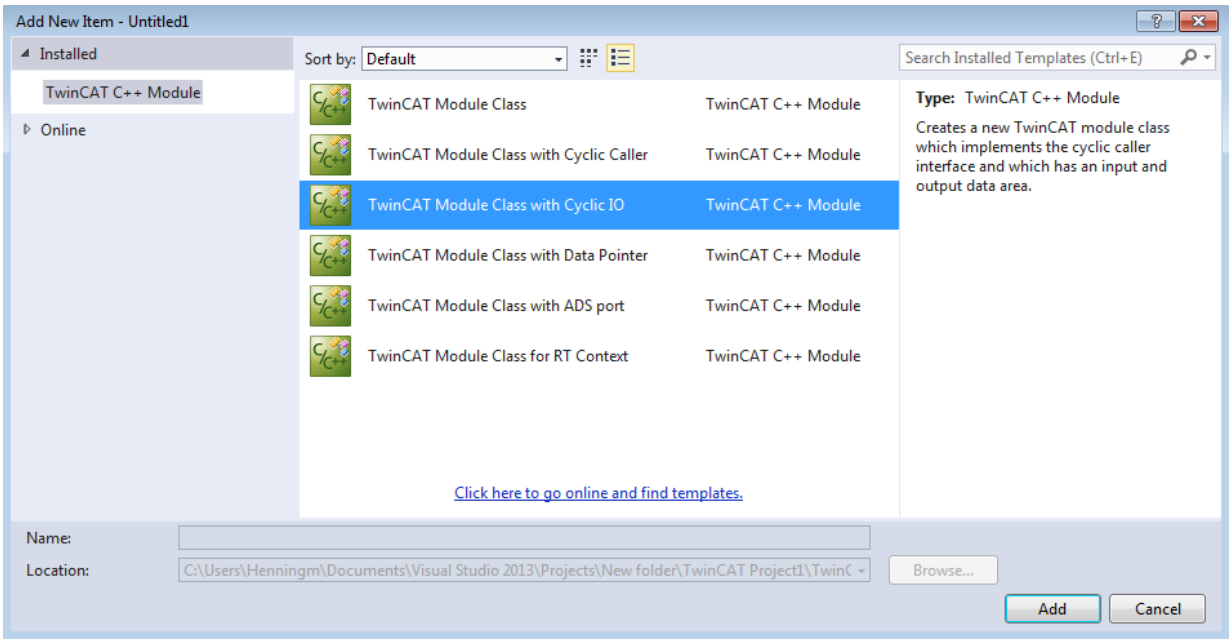
2. Fügen Sie ein TwinCAT C++ Projekt hinzu



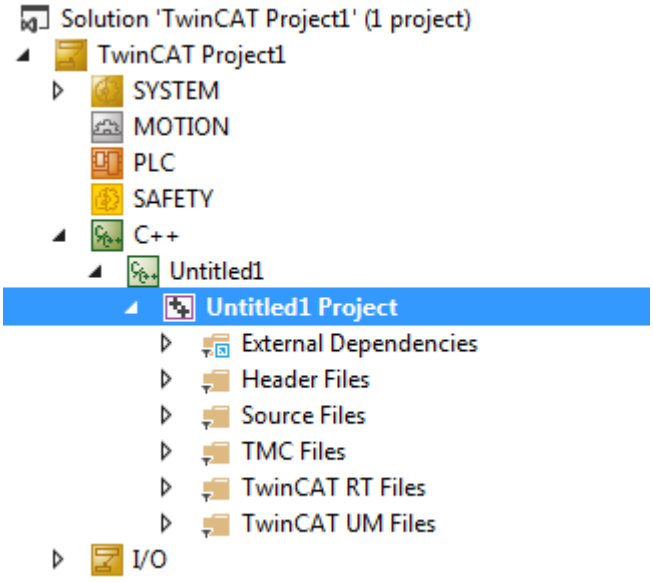
3. Wählen Sie ein Driver Projekt aus



4. Verwenden Sie als Grundlage für den TCP Client den Wizard für eine Modul-Klasse mit „Cyclic IO“.



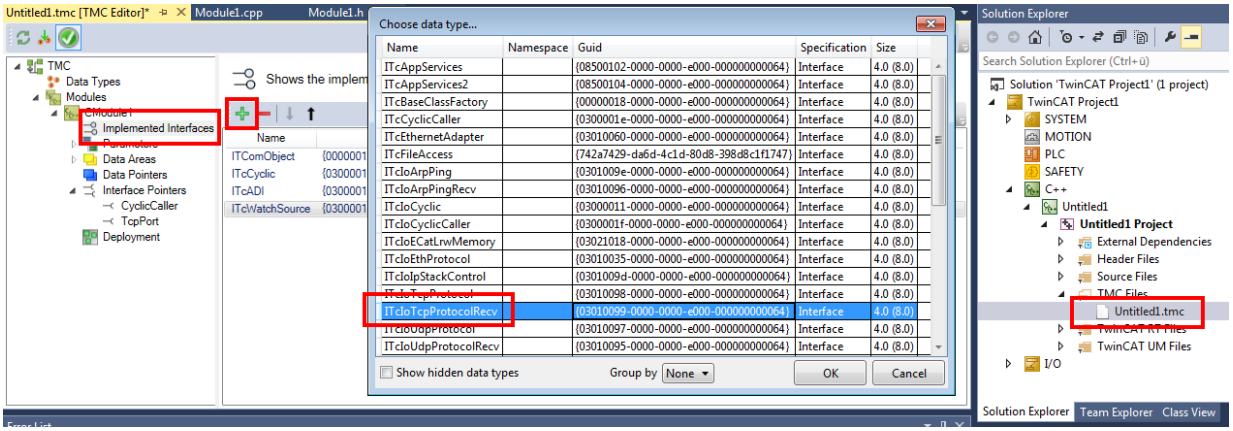
⇨ Als Ergebnis liegt ein fertiges TwinCAT C++ Projekt vor.



TMC Editor zum Anlegen von Interfaces, Pointern und Parametern

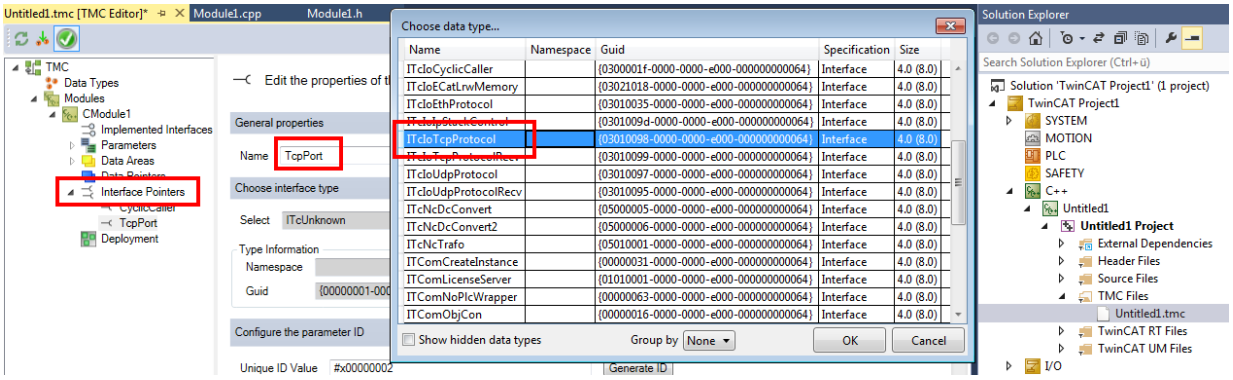
Nach dem Anlegen des Projektes wird in diesem Arbeitsschritt die Implementierung des C++ TCP Clients vorgenommen.

1. Das durch den Wizard erstellte Modul muss das Interface „ITcIoTcpProtocolRecv“ implementieren. Öffnen Sie den TMC Editor, indem Sie auf die TMC Datei des Projektes doppelklicken. Fügen Sie das Interface dem Modul unter „Implemented Interfaces“ hinzu.

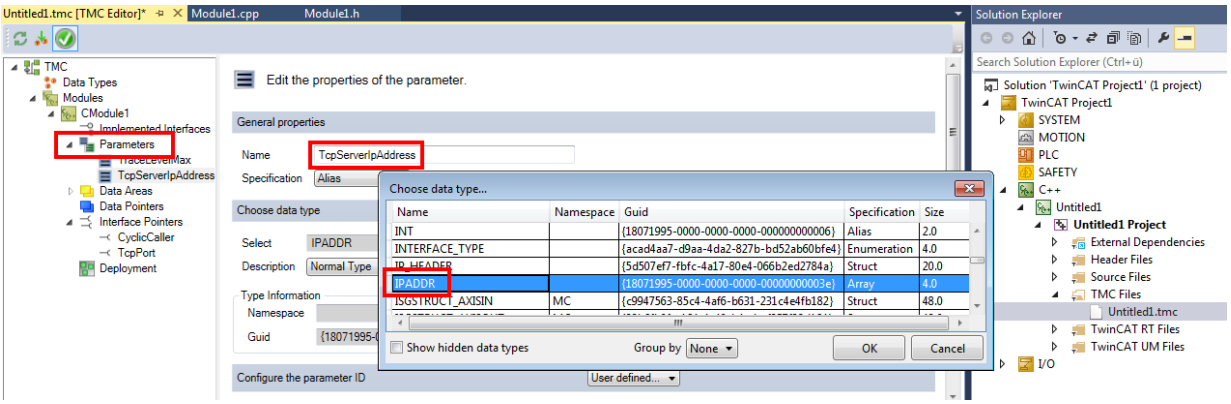


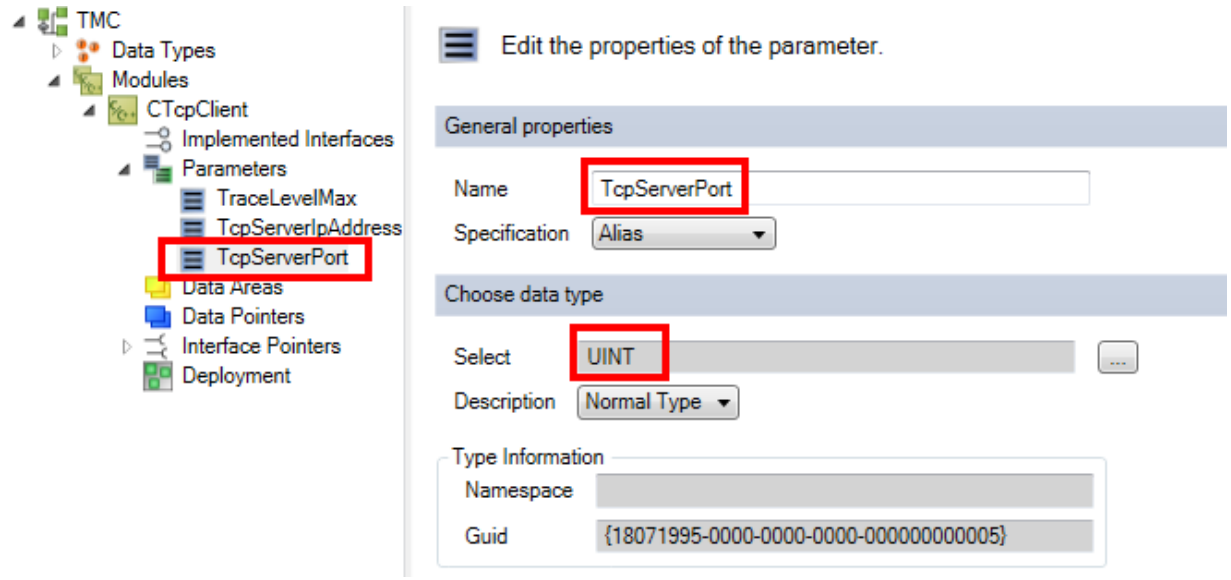
Unter „Implemented Interfaces“ öffnen Sie eine Auswahl der bereitstehenden Interfaces durch einen Klick auf den „+“-Button. Wählen Sie hier „ITcIoTcpProtocolRecv“ aus.

2. Zusätzlich benötigen Sie einen Interface Pointer „ITcIOTcpProtocol“.

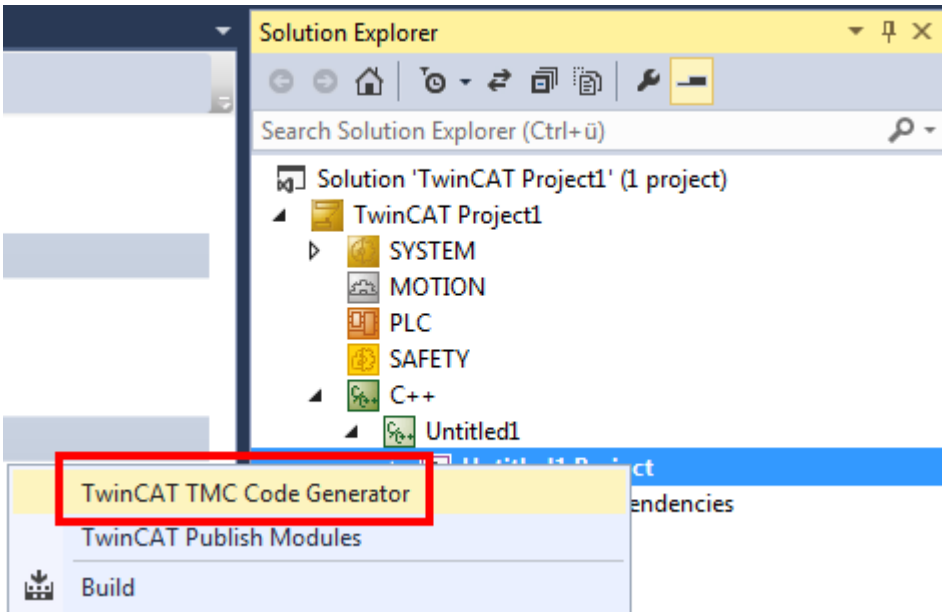


3. Durch Anlegen eines Parameters werden die zu kontaktierende Server IP Adresse und der Port konfigurierbar.





4. Benutzen Sie nun den TMC Code Generator, um den Code des C++ Moduls vorzubereiten.



Starten Sie den TMC Code Generator indem Sie auf dem C++ Projekt im Kontexmenü (Rechts-Klick) den entsprechenden Menüpunkt auswählen.

⇨ Alle Schritte im TMC Editor sind nun abgeschlossen.

TCP Client implementieren

1. Legen Sie in der Modul Header Datei (hier: Modul1.h) zwei Member-Variablen an.

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

2. Diese werden in dem Constructor (Module1.cpp) initialisiert.

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

3. Der Interface Pointer m_spTcpProt wird nun in der Transition SO (also in Methode SetObjStateSO) initialisiert.

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

4. In der Transition OS (also Methode SetObjStateOS) wird eine evtl. vorhandene Verbindung abgebaut und der Socket freigegeben.

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

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

    RemoveModuleFromCaller();

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

5. In der „CycleUpdate“ Methode, die zyklisch aufgerufen wird, wird der eigentliche Ablauf implementiert. Hier wird eine TCP Verbindung zu einem Server aufgebaut (Adresse wird in Parametern „m_TcpServerIpAddress“ und „m_TcpServerPort“ bereitgestellt). Das Handle zur Verbindung wird in der Member-Variable „m_SockId“ abgelegt. Die Verbindung wird genutzt, um einen einfachen http-GET-Request abzusetzen.

```
HRESULT CTcpClient::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;
    //start added code
    if ( m_SockId == 0 )
    {
        if (SUCCEEDED_DBG(hr = m_spTcpProt->AllocSocket(THIS_CAST(ITcIoTcpProtocolRecv),
m_SockId)))
        {
```

```
            if (FAILED(hr = m_spTcpProt->Connect(m_SockId, ((PULONG)&m_TcpServerIpAddress)[0],
m_TcpServerPort)))
            {
                m_spTcpProt->FreeSocket(m_SockId);
                m_SockId = 0;
            }
else {
                m_connections++; //count number of connections
            }
        }
    }
    else
    {
    if ( m_bSendRequest && m_spTcpProt->IsConnected(m_SockId) == S_OK )
    {
        PCHAR pRequest = "GET / HTTP/1.1\r\nHOST: beckhoff.com\r\n\r\n ";
        ULONG nSendData = 0;
        m_hrSend = m_spTcpProt->SendData(m_SockId, strlen(pRequest), pRequest, nSendData);
        m_bSendRequest = false;
    }
    }

    m_spTcpProt->CheckReceived();

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

6. Das Modul implementiert das Interface „ITcIoTcpProtocolRecv“, wodurch der TMC Code Generator eine Methode „ReceiveEvent“ angelegt hat. Diese wird aufgerufen, wenn ein Event empfangen wurde und muss somit mit den unterschiedlichen Event-Typen umgehen können.

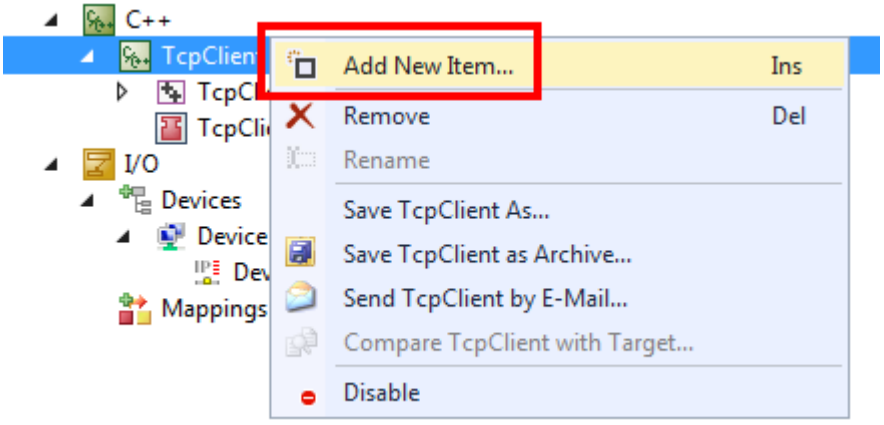
```
HRESULT CTcpClient::ReceiveEvent(ULONG socketId, TCPIP_EVENT tcpEvent)
{
//start added code
m_Trace.Log(tlInfo, FLEAVEA "Receive TCP Event: SocketId: %d Event: %d \n", socketId, tcpEvent);

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

7. Äquivalent zu der „ReceiveEvent“ Methode wurde eine „ReceiveData“ Methode ebenfalls aus dem Interface „ITcIoTcpProtocolRecv“ angelegt. Diese ist für das Empfangen der Daten zuständig und wird wie folgt implementiert:

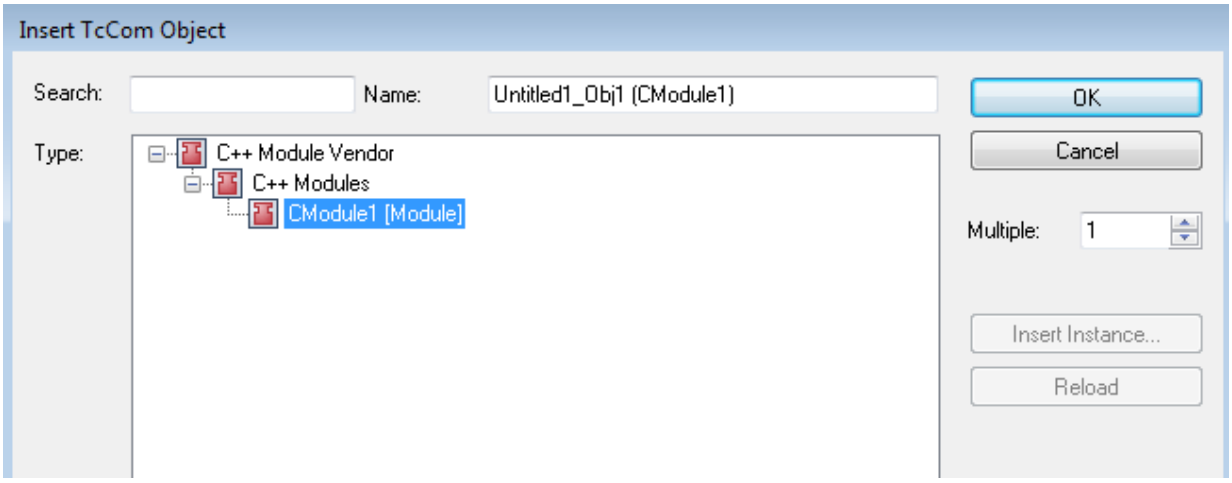
```
HRESULT CTcpClient::ReceiveData(ULONG socketId, ULONG nData, PVOID pData)
{
//start added code
    HRESULT hr = S_OK;
    PCHAR pResponse = new CHAR[100];
    memset(pResponse, 0, 100);
    memcpy(pResponse, pData, min(100, nData));
    m_Trace.Log(tlInfo, FLEAVEA "Receive answer w/ length %d : first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

8. Das Modul ist nun fertig und kann kompiliert werden. (Rechts-Klick auf das Projekt “Build”).

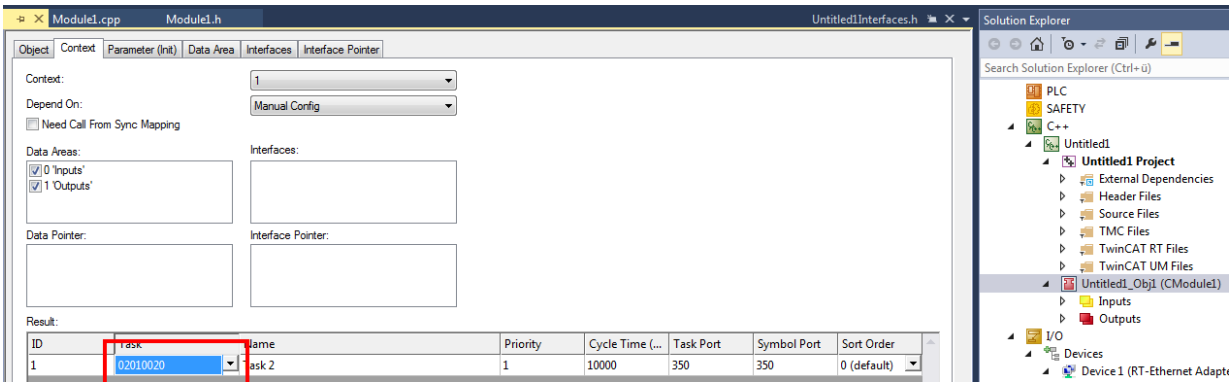
9. Eine Instanz des Moduls wird angelegt:
   Dafür Rechts-Klick auf das C++ Projekt



und Auswahl des Moduls



⇨ Die Instanz wird mit einem Task verbunden, sodass die „CycleUpdate“ Methode aufgerufen wird.

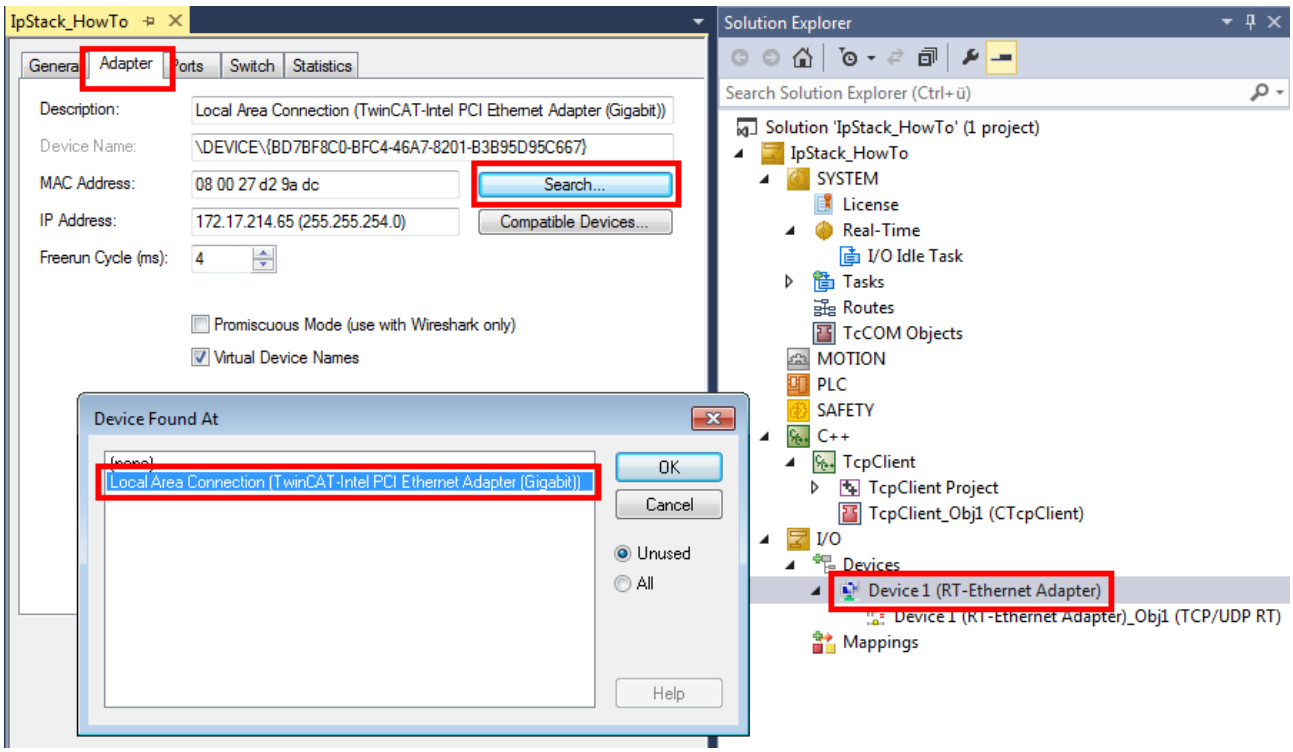


Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



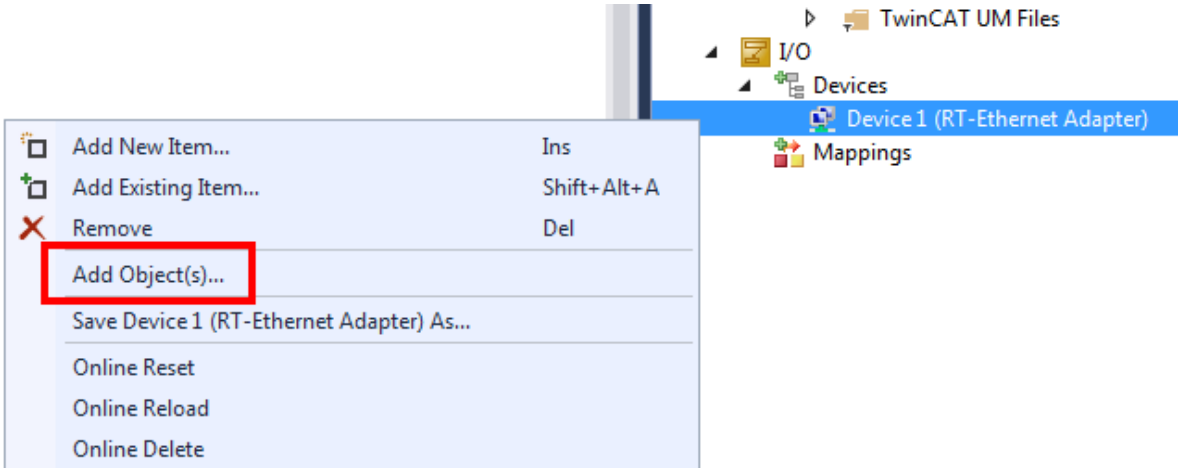
“TCP/UDP RT” Modul Konfiguration

HINWEIS

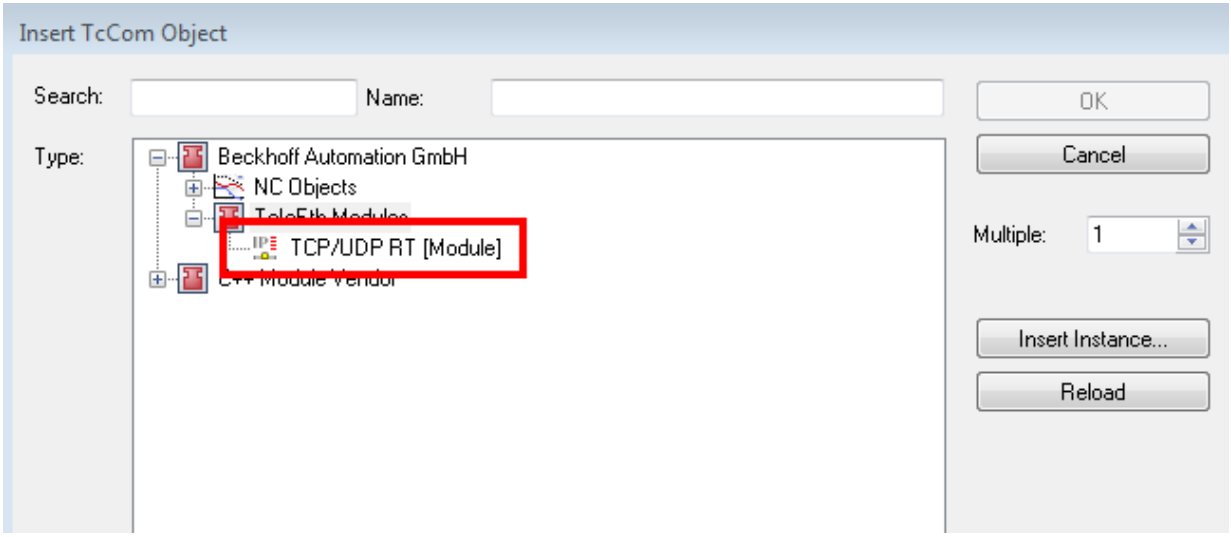
Variablennamen

Hier werden Variablennamen in Bezug auf TCP verwendet. Diese sind entsprechend zu ersetzen.

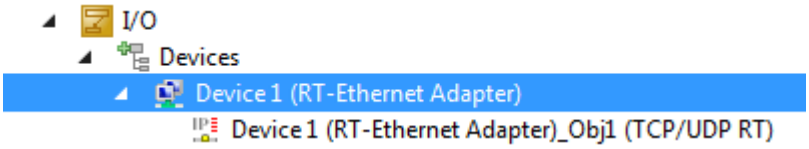
1. Legen Sie das „TCP/UDP RT“ Modul unterhalb des RT-Ethernet-Adapters an, indem Sie „Add Object(s)…“ im Context-Menü anwählen.



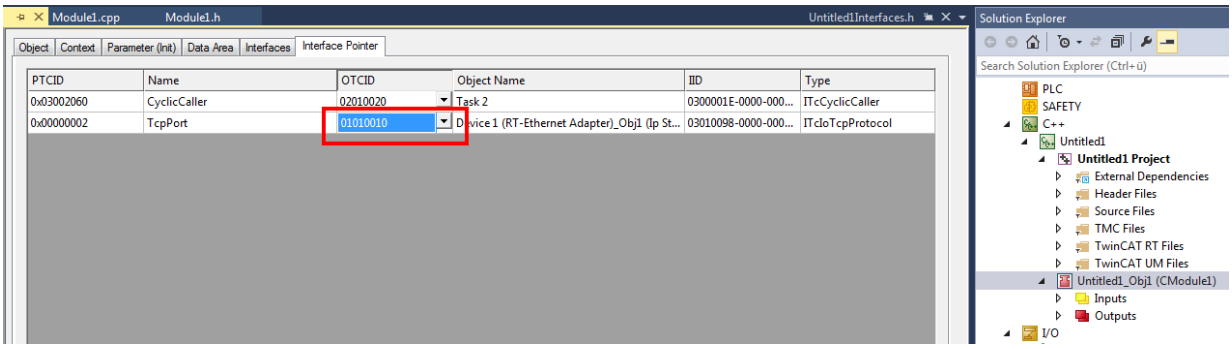
2. Dann wählen Sie das „TCP/UDP RT“ Modul aus:



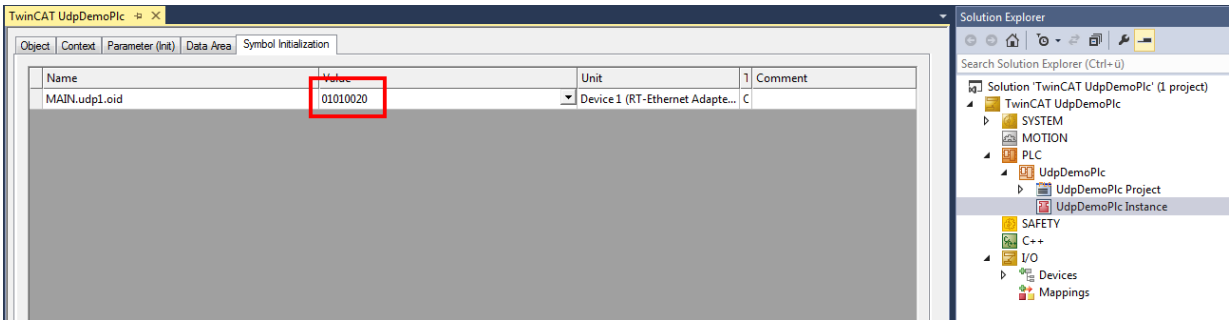
⇨ Das TCP/UDP RT Objekt wird unterhalb des Adapters angelegt.



3. Parametrieren Sie die zuvor angelegte Instanz des Moduls (hier: Modul1) unter „Interface Pointer“ „TcpProt“ mit der OID des angelegten „TCP/UDP RT“ Objekts:



4. Bei PLC Projekten ist diese Konfiguration ebenso an der Instanz vorzunehmen, hier jedoch unter dem Reiter „Symbol Initialization“:



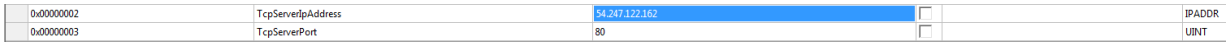
⇨ Damit ist die Konfiguration abgeschlossen

Verbindungsabbruch durch Betriebssystem bei Promiscuous Mode

Wenn an dem RT-Ethernet Adapter im Tab „Adapter“ der Promiscuous Mode eingeschaltet ist, werden eintreffende TCP Verbindungsaufbauten durch das Betriebssystem abgebrochen, da dieses einen im TCP/UDP RT Objekt geöffneten Port nicht kennt.

Handhabung

1. Das Beispiel ist betriebsbereit, nachdem Sie an der Modulinstanz sowohl die TcpServerIpAddress als auch den TcpServerPort konfiguriert haben:

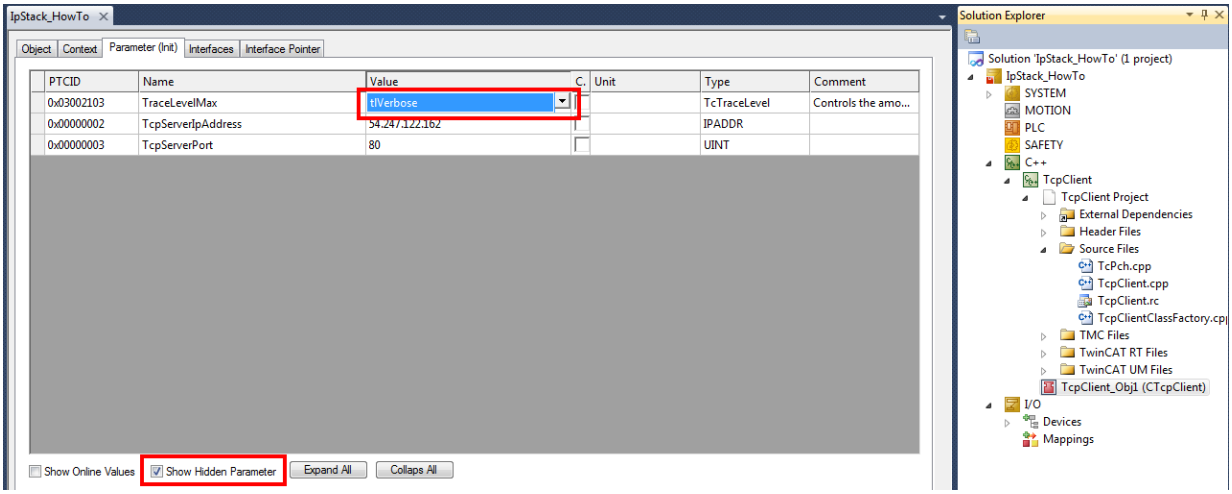


Hinweis Mögliche Fehlerquelle: Hier im Beispiel wird ein Test-Webserver 62.159.14.51 abgefragt. Dafür ist im Sourcecode oben ein entsprechender HTTP Befehl hinterlegt. IP-Adresse, Port und dieser HTTP Befehl müssen ggf. angepasst werden.

2. Nach dem Aktivieren der Konfiguration können Sie im Output sowohl Log Meldungen (vgl. Source Code) als auch die ersten 100 Bytes der Antwort des Servers sehen:

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

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

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

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

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

3. Für die Ausgabe dieser Meldungen kann der „Tracelevel“ (auf Info) konfiguriert werden:

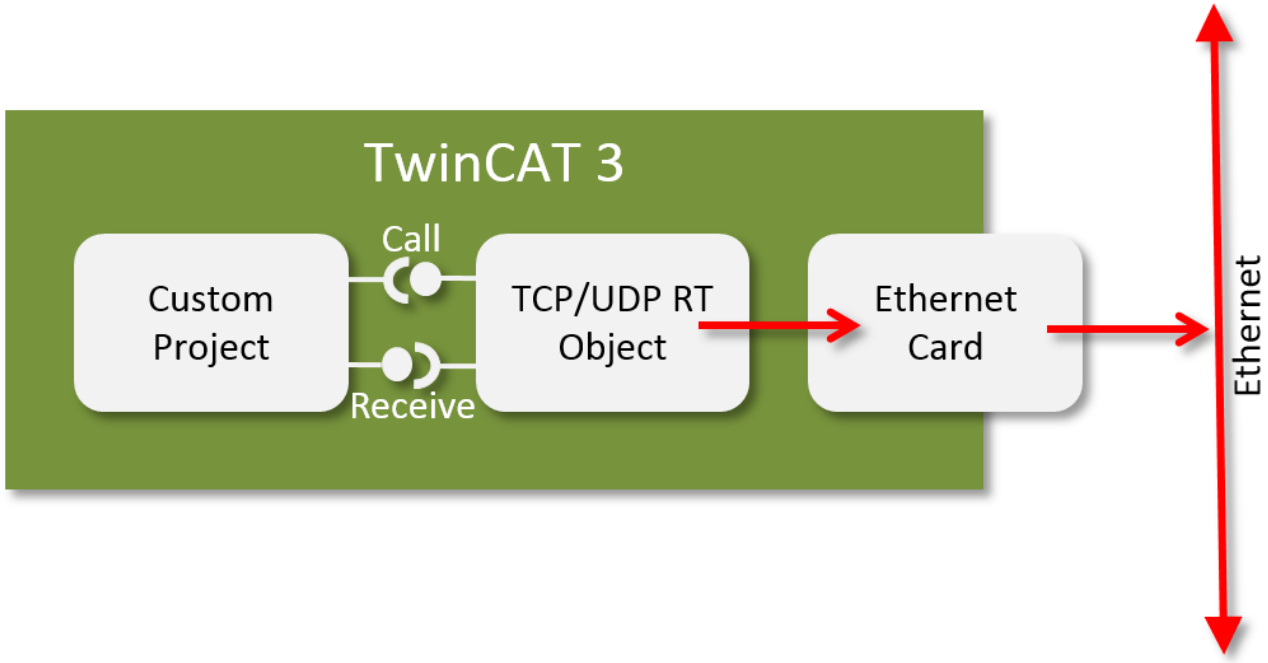


Der Ablauf erfolgt einmalig beim Start des Programms.

Wenn „m_bSendRequest“ auf TRUE gesetzt wird (z.B. durch das TwinCAT Live Watch), wird ein neuer Request gesendet. Die Rückgabe der SendData Methode wird in hrSend abgelegt – für das Beispiel kann sie per Debugger beobachtet werden.

5 Konfiguration

Ausgehend von einem existierenden TwinCAT Projekt wird an dieser Stelle die Einbindung und Konfiguration des „TCP/UDP RT“ Objektes beschrieben.



Das „TCP/UDP RT“ Objekt wird zum einen instanziiert und zum anderen konfiguriert. Die Konfiguration ist dabei im Wesentlichen die Zuordnung der Netzwerkkarte, die verwendet werden soll.

Windows Firewall

Da das TF6311 direkt im TwinCAT System integriert ist, kann die Windows Firewall nicht genutzt werden.

Das „TCP/UDP RT“ Objekt bringt auch einige Parameter mit, die hier [▶ 57] dokumentiert sind.

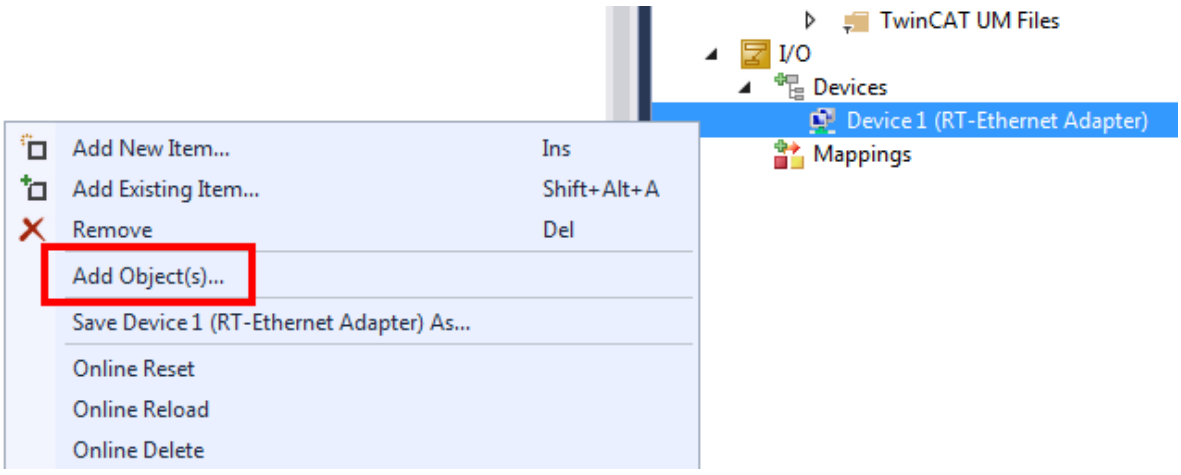
“TCP/UDP RT” Modul Konfiguration

HINWEIS

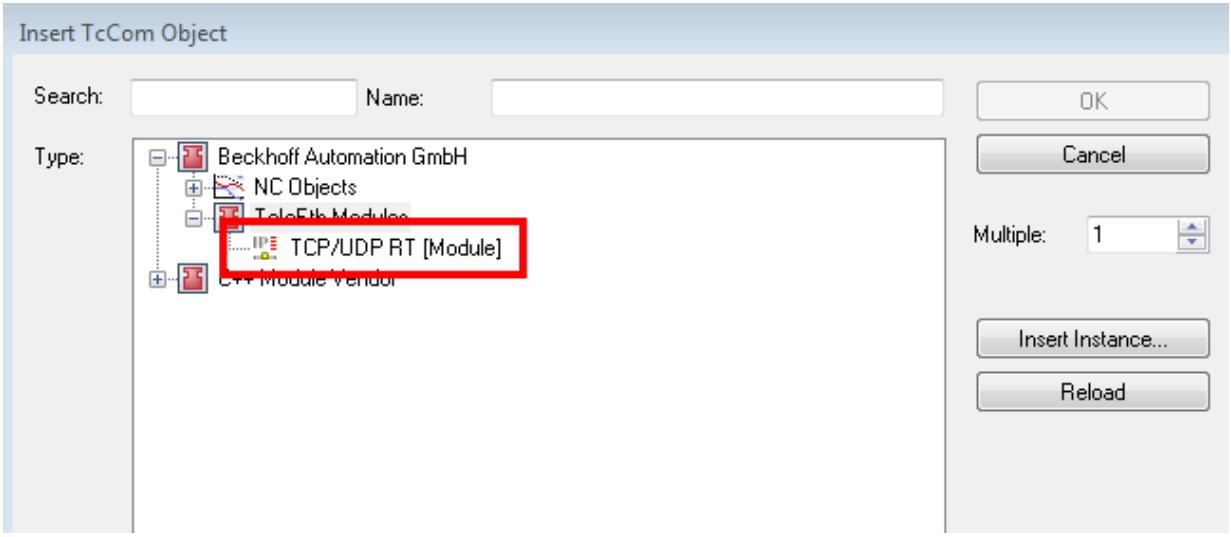
Variablennamen

Hier werden Variablennamen in Bezug auf TCP verwendet. Diese sind entsprechend zu ersetzen.

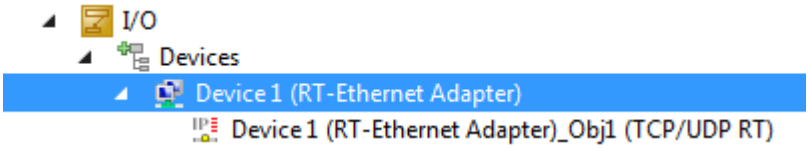
1. Legen Sie das „TCP/UDP RT“ Modul unterhalb des RT-Ethernet-Adapters an, indem Sie „Add Object(s)…“ im Context-Menü anwählen.



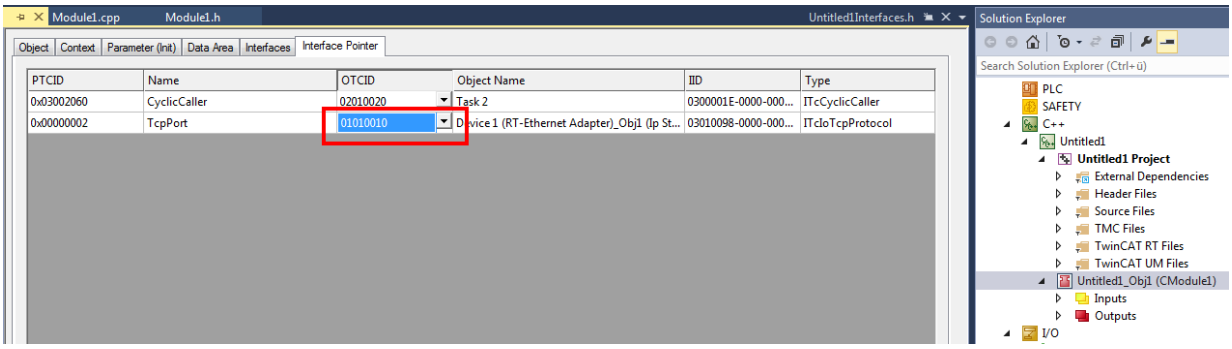
2. Dann wählen Sie das „TCP/UDP RT“ Modul aus:



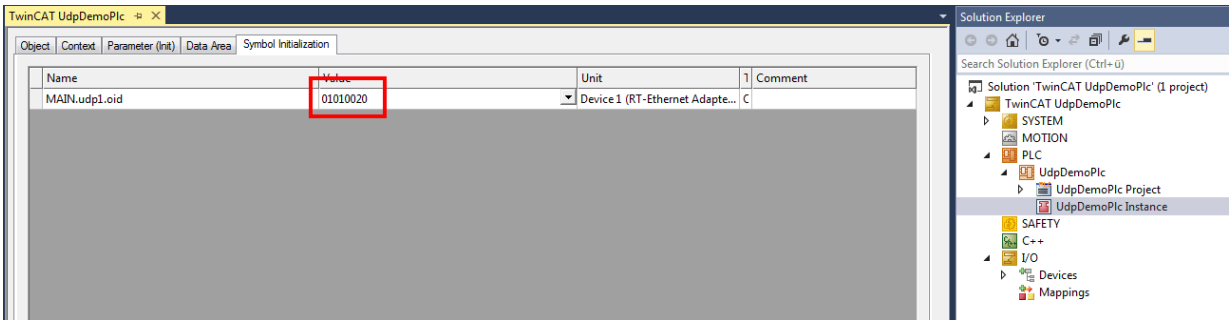
⇨ Das TCP/UDP RT Objekt wird unterhalb des Adapters angelegt.



3. Parametrieren Sie die zuvor angelegte Instanz des Moduls (hier: Modul1) unter „Interface Pointer“ „TcpProt“ mit der OID des angelegten „TCP/UDP RT“ Objekts:



4. Bei PLC Projekten ist diese Konfiguration ebenso an der Instanz vorzunehmen, hier jedoch unter dem Reiter „Symbol Initialization“:



⇨ Damit ist die Konfiguration abgeschlossen

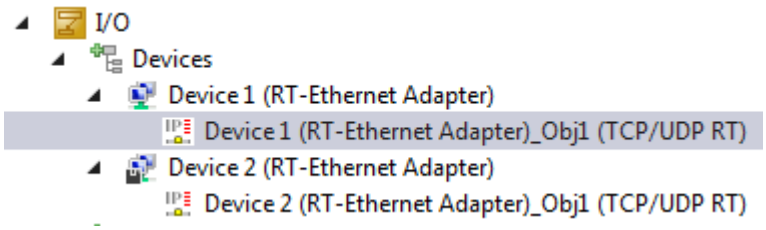
Verbindungsabbruch durch Betriebssystem bei Promiscuous Mode

Wenn an dem RT-Ethernet Adapter im Tab „Adapter“ der Promiscuous Mode eingeschaltet ist, werden eintreffende TCP Verbindungsaufbauten durch das Betriebssystem abgebrochen, da dieses einen im TCP/UDP RT Objekt geöffneten Port nicht kennt.

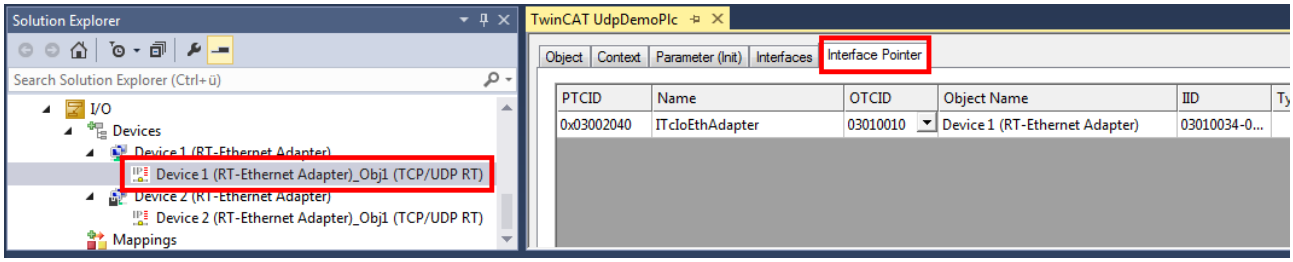
5.1 Mehrere Netzwerkkarten

Ein TCP/UDP RT Objekt ist einem RT-Ethernet-Adapter zugeordnet indem es beispielsweise unterhalb von den Objekten instanziiert wurde. Ein TCP/UDP RT Objekt spricht also über den RT-Ethernet-Adapter immer genau einen Netzwerk-Port der Steuerung an.

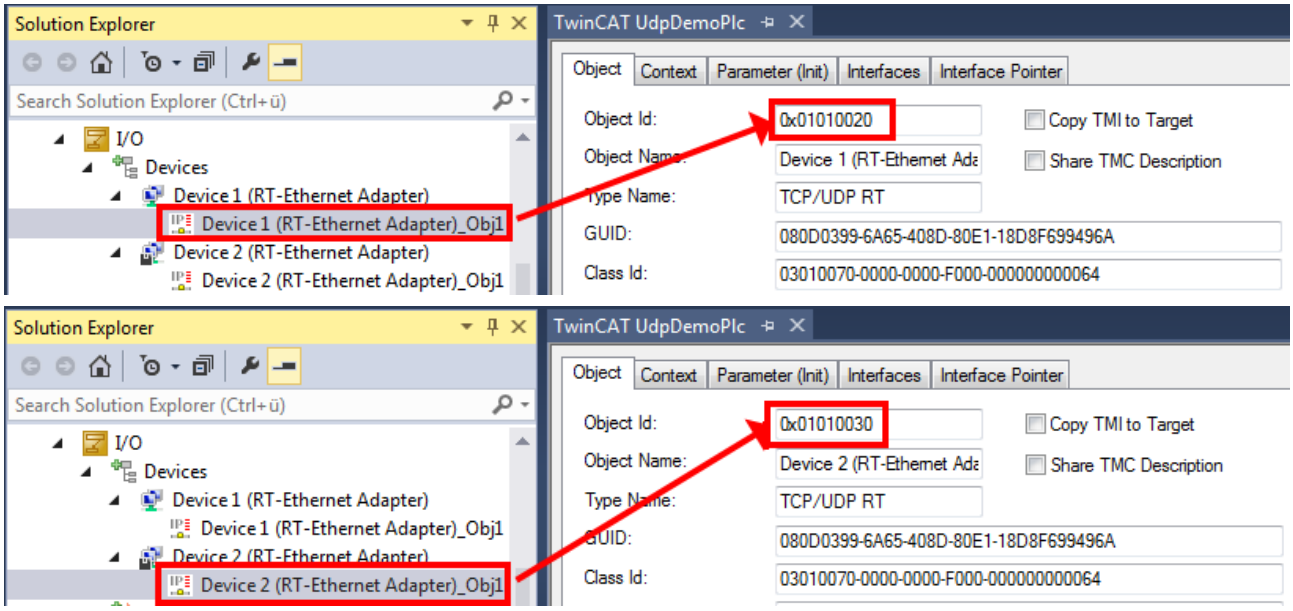
Wenn nun mehrere Netzwerk-Ports genutzt werden sollen, wird pro RT-Ethernet-Adapter ein TCP/UDP RT Objekt erzeugt:



Die TCP/UDP RT Objekte beziehen sich dabei auf den übergelagerten RT-Ethernet Adapter, wenn nicht manuell etwas anderes konfiguriert wurde:

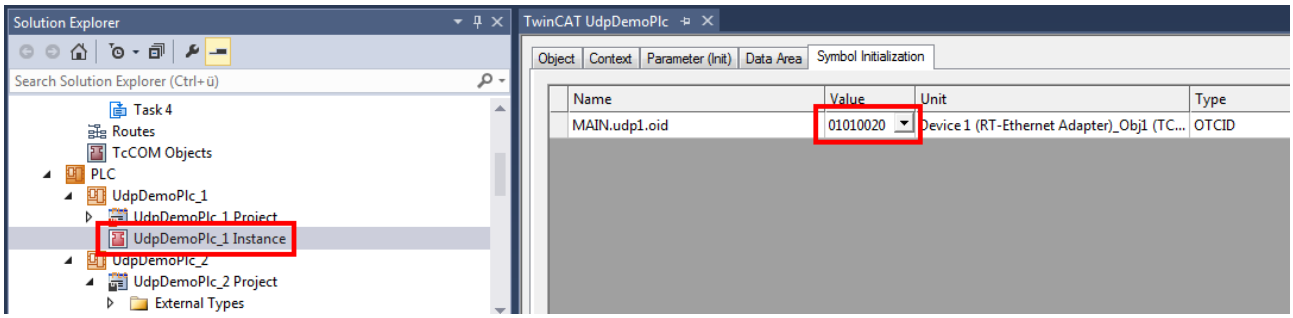


Diese Objekte haben unterschiedliche Objekt IDs:

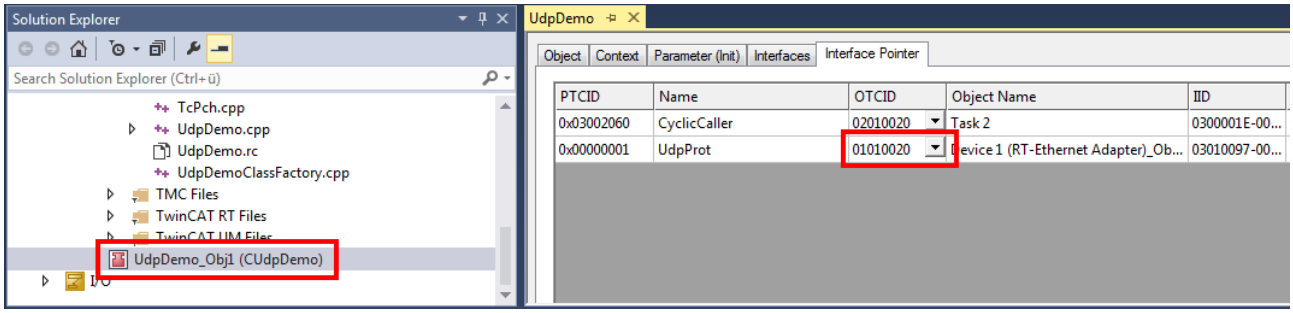


Diese Objekt ID wird zur Referenzierung verwendet, wie es zuvor auch beschrieben ist:

PLC:

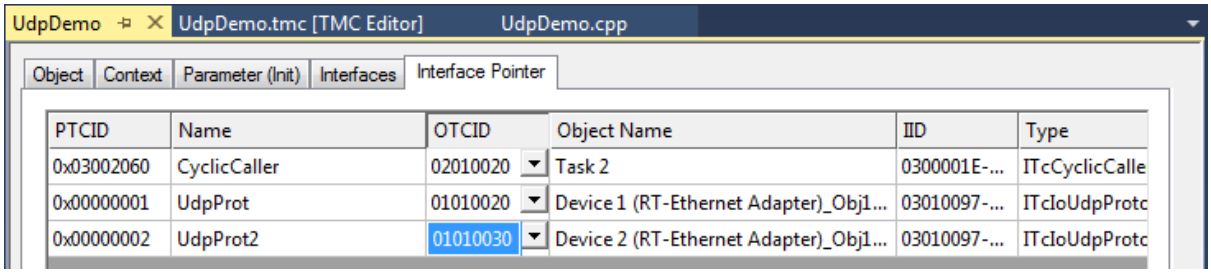


Oder für ein C++ Modul:



Die Verwendung ist stark applikationsabhängig. Hier einige Beispiel-Szenarien:

- Ein C++ Modul kann mehr als einmal instanziiert werden. Jedes Modul kann dann durch Konfiguration mit der entsprechenden Objekt ID über eine bestimmte Netzwerkkarte kommunizieren.
- Unterschiedliche PLC Programme können jeweils ein TCP/UDP RT Objekt zugewiesen bekommen und somit unabhängig agieren.
- Ein PLC oder C++ Programm kann auch mehrere TCP/UDP RT Objekte (und damit mehrere Netzwerkkarten) ansprechen indem entsprechende Symbole vorgesehen werden (hier am Beispiel C++):

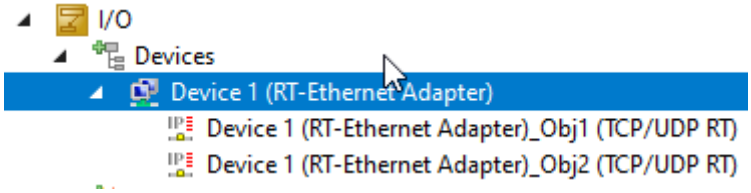


Hierbei ist dann applikativ eine entsprechende Verwaltung der Objekte zu realisieren. Beispielsweise müssen die CheckReceived() Aufrufe auf allen Objekten vorgenommen werden und auch Aufrufe für z.B. SendData() / RegisterReceiver() etc. müssen auf den entsprechenden Objekten vorgenommen werden.

5.2 Multitask Zugriff auf eine Netzwerkkarte

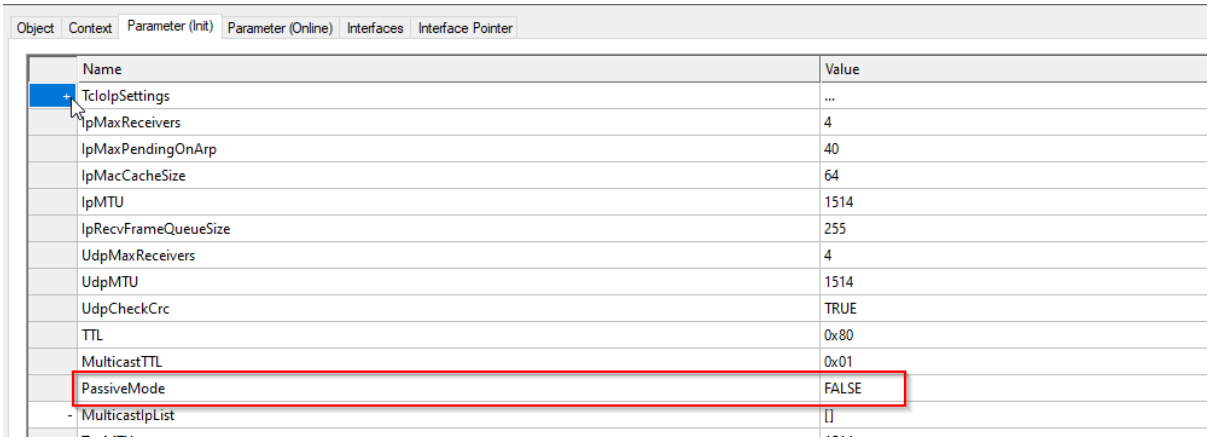
Wenn eine Netzwerkkarte aus mehreren Echtzeit-Kontexten (Tasks) genutzt werden soll, so muss dieses wie hier beschrieben realisiert werden.

- Es muss pro Echtzeit-Kontext (z. B. Task), aus welchem Daten empfangen oder gesendet werden sollen, ein TCP/UDP RT-Objekt angelegt werden.



- Der Parameter PassiveMode an allen TCP/UDP RT-Objekten legt fest, ob diese Objekte von dem RT-Ethernet Adapter eingegangene Frames abholen sollen oder nicht. Per Default ist PassiveMode auf FALSE, sodass die Pakete abgeholt werden.
  Bei Multitask-Zugriffen sollten nur ein TCP/UDP RT Objekt die Daten abholen und alle anderen Objekte mit PassiveMode auf TRUE konfiguriert werden.
  Üblicherweise kann dieses das Objekt sein, welches im schnellsten Zyklus Pakete empfängt. Ggf.

kann hierfür eine niedrigere Priorität verwendet werden, um die Echtzeitabläufe anderer Tasks unabhängiger von den eintreffenden Frames zu gestalten.

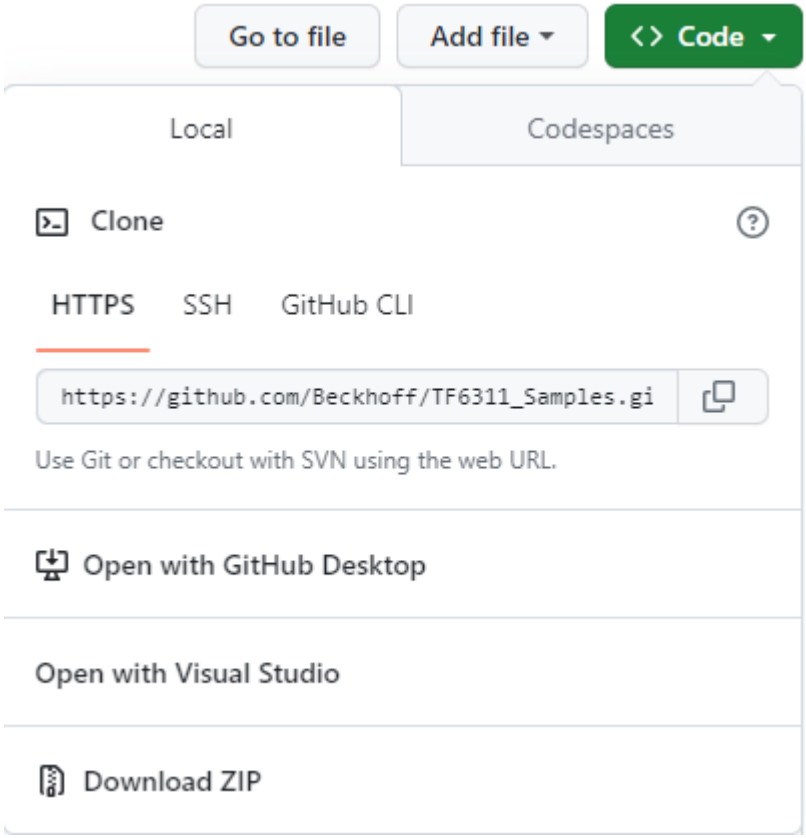


- Der Funktionsbaustein muss in dem gleichen Kontext die RegisterReceiver() / Open() wie auch im zyklischen Ablauf die CheckReceived() Methode aufrufen.
- Die Callbacks über ReceiveData()/...Event() werden im gleichen Kontext aufgerufen, wie zuvor das CheckReceived() vom Funktionsbaustein der Anwendung.

6 Beispiele

Diese Beispiele stellen einfach nachzuvollziehende Demonstrationen für den Umgang mit dem TCP/UDP RT Modul dar.

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://github.com/Beckhoff/TF6311_Samples . Sie haben dort die Möglichkeit das Repository zu clonen oder ein ZIP File mit dem Sample herunterzuladen.



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

Dieses Beispiel zeigt die Verwendung einer TCP Verbindung als Client.
In diesem Beispiel wird eine TCP Verbindung zu einer IP Adresse mit Port 80 aufgemacht. Dabei handelt es sich um den Webserver von Beckhoff. Das Beispiel nutzt die Verbindung dann einen HTTP-Request abzuschicken und somit eine Test-Webseite von 62.159.14.51:80 abzurufen.

Passt die Webseite nicht in den Empfangspuffer, werden mehrere Aufrufe der ReceiveData() Methode erfolgen.

Der Client führt jeweils einen erneuten Verbindungsaufbau durch, falls die Verbindung z.B. durch den Server abgebaut wird.

Das Beispiel ist für C++ und für die PLC verfügbar.

6.1.1 S01: Simple TCP Client (C++)

Dieses Beispiel realisiert einen TCP Client, der einen einfachen HTTP-Request absetzt und die Antwort empfängt.

Der hier verfügbare Download ist vorkonfiguriert, um eine Test-Webseite von 62.159.14.51:80 abzurufen.

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S01-IpStackTcpClient

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen
4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

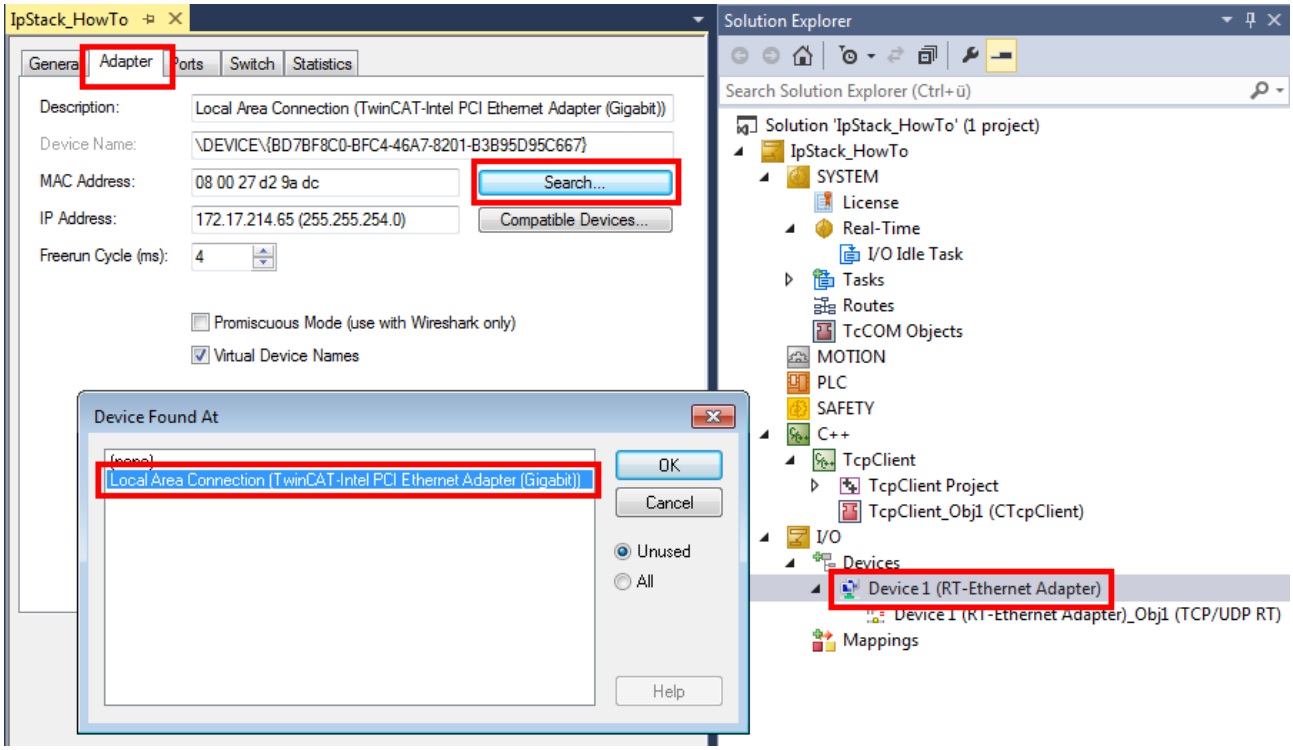
Das Beispiel ist ausführlich auf der Seite Quick Start [▶ 23] erläutert.

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



6.1.2 S01: Simple TCP Client (PLC)

Dieses Beispiel realisiert einen TCP Client, der einen einfachen HTTP-Request absetzt und die Antwort empfängt.

Der hier verfügbare Download ist vorkonfiguriert, um eine Test-Webseite von 62.159.14.51:80 abzurufen.

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S01-IpStackTcpClientPlc

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen

4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

Nach dem Starten kann das PLC-Programm genutzt werden indem die Variable „bSend“ auf TRUE gesetzt wird. Der HTTP-Request (abgelegt in „sMessage“) wird nach Verbindungsaufbau dem Server gesendet. Die ersten Bytes der ankommenden Antwort werden in „sLastReturnedMessage“ bereitgestellt. Die „sLastReturnedMessafeLength“ gibt die gesamte Länge der Antwort an.

Die Server Adresse wird in der FB_init Methode definiert.

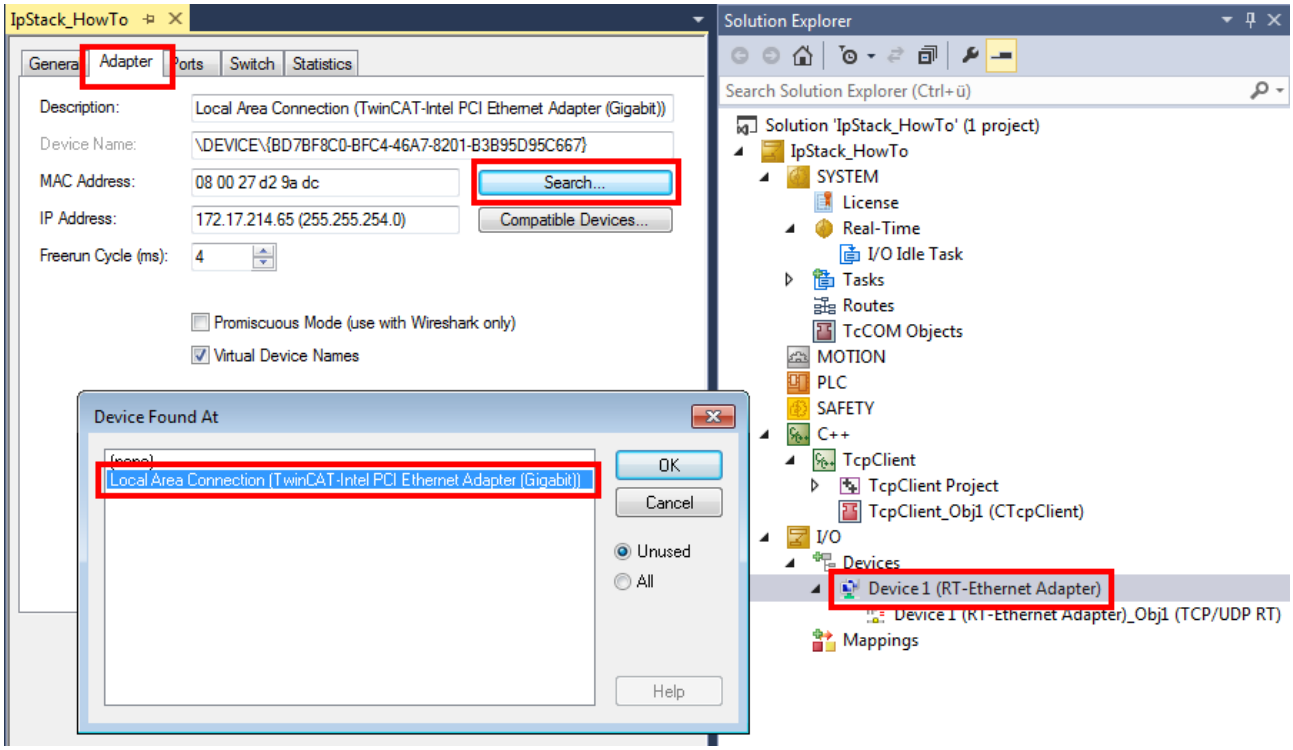
Das gleiche Beispiel ist in C++ ausführlich auf der Seite Quick Start [▶ 23] erläutert.

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



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

Dieses Beispiel beschreibt, wie ein TwinCAT-Projekt als UDP Server agieren kann. Somit können Werte in die Echtzeit bzw. auf Anfrage aus der Echtzeit geliefert werden.

Das Beispiel implementiert dabei einen „Echo-Dienst“: Hierbei wird auf einem Port (Standard: 10000) ein UDP Server gestartet. Wenn dieser ein UDP Paket empfängt, sendet er den Inhalt zurück an den Absender (mit gleicher IP und gleichem Port). Das Beispiel ist sowohl in PLC [▶ 43] als auch in C++ [▶ 45] verfügbar.

Zu Testzwecken ist zusätzlich ein UDP Client [▶ 46] (geschrieben in .NET) verfügbar.

Die Samples sind als auch ausführlich als Quick Starts [▶ 11] verfügbar.

6.2.1 S02: UDP Demo (PLC)

Dieses Beispiel beschreibt einen UDP Server, der in einem PLC Projekt implementiert ist.

Es empfängt UDP Pakete und sendet sie zurück zum Absender („Echo-Server“).

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S02-UdpDemoPlc

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen
4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

Das Sample ist auch ausführlich als Quick Start verfügbar.

Analog zum Quick Start [▶ 23] wird in diesem Beispiel das Interface ITcIoUdpProtocolRecv [▶ 54] implementiert und ein Pointer auf ein ITcIoUdpProtocol [▶ 54] verwendet.

Dafür wird ein PLC Baustein angelegt, der das Interface ITcIoUdpProtocolRecv [▶ 54] implementiert („Add POU“ mit „Implements“). Dabei ist es wichtig in den Methoden „FB_init“ und „FB_exit“ die Verbindung zum TCP/UDP RT Objekt zu realisieren. Dieses Vorgehen ist ausführlicher im Sample 11 der C++ Dokumentation beschrieben.

Der implementierende Funktionsbaustein (im Sample UdpReceiver) ruft die Methode „CheckReceived“ auf. Dem IP Stack wird so die Möglichkeit gegeben, ankommende Pakete zu bearbeiten und Callbacks auf die Methode „ReceiveData“ des Funktionsbausteins zu übermitteln.

Die „ReceiveData“ Methode nutzt die „SendData“ Methode um die Daten zum Absender zurückzusenden („Echo-Server“).

Verständnis

Damit die Kommunikation zwischen Funktionsbaustein und TcCOM Objekt „TCP/UDP RT“ aufgebaut wird, werden zwei Methoden verwendet:

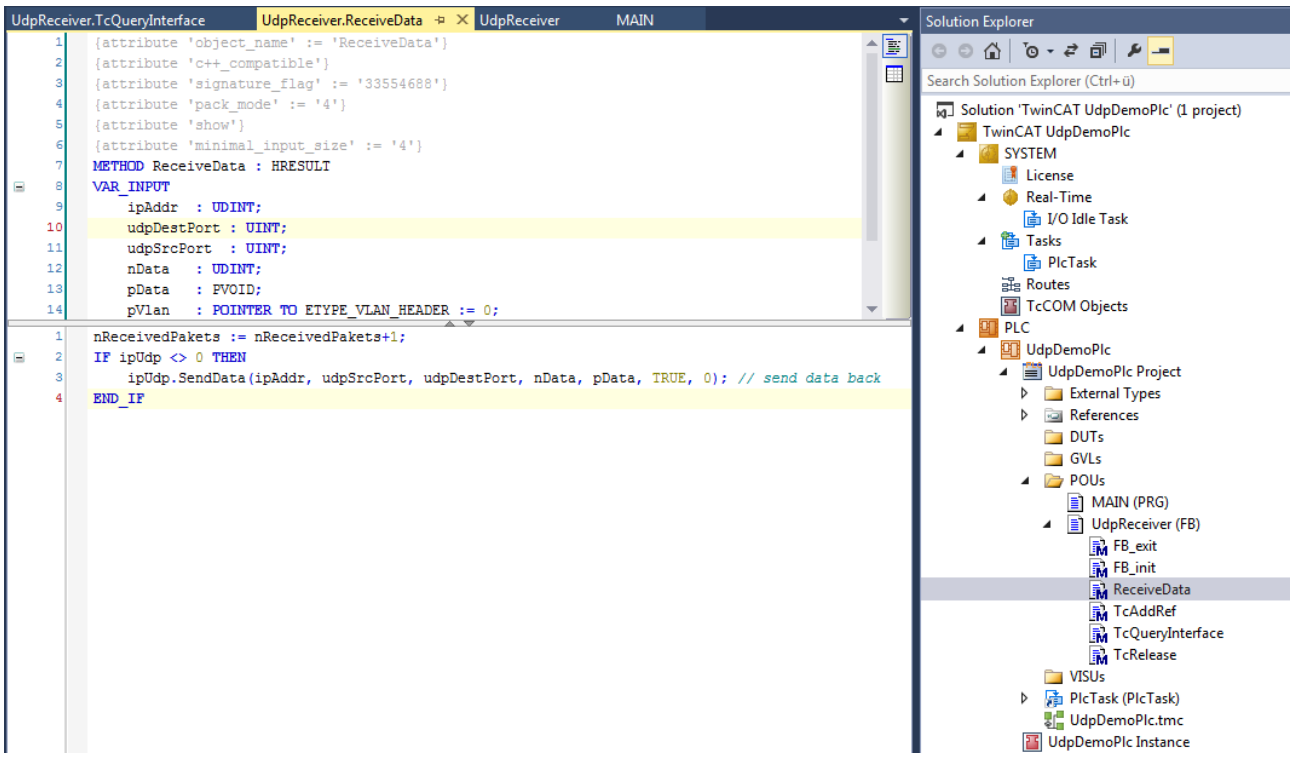
- „FB_init“: Diese wird automatisch ausgeführt, wenn die PLC gestartet wird
- „FB_exit“: Diese wird automatisch ausgeführt, wenn die PLC angehalten wird

Diese Initialisierungsphase kann größtenteils aus dem Beispiel-Code übernommen werden.

Für die eigentliche UDP Funktionalität sind im PLC Code zwei Methoden zuständig:

- Die „ReceiveData“ Methode am implementierten Funktionsbaustein empfängt die Daten.
- Die “SendData” Methode am ITcIoUdpProtocol Interface sendet Daten.

In dem Beispiel wird in der „ReceiveData“ Methode die „SendData“ Methode verwendet um die empfangenen Daten zurückzusenden:



Die Methode TcQueryInterface muss folgendermaßen implementiert werden, damit TwinCAT erkennt, dass das entsprechende Interface implementiert wurde:

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


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

Die zusätzlich angelegten Methoden

- TcAddRef / TcRelease

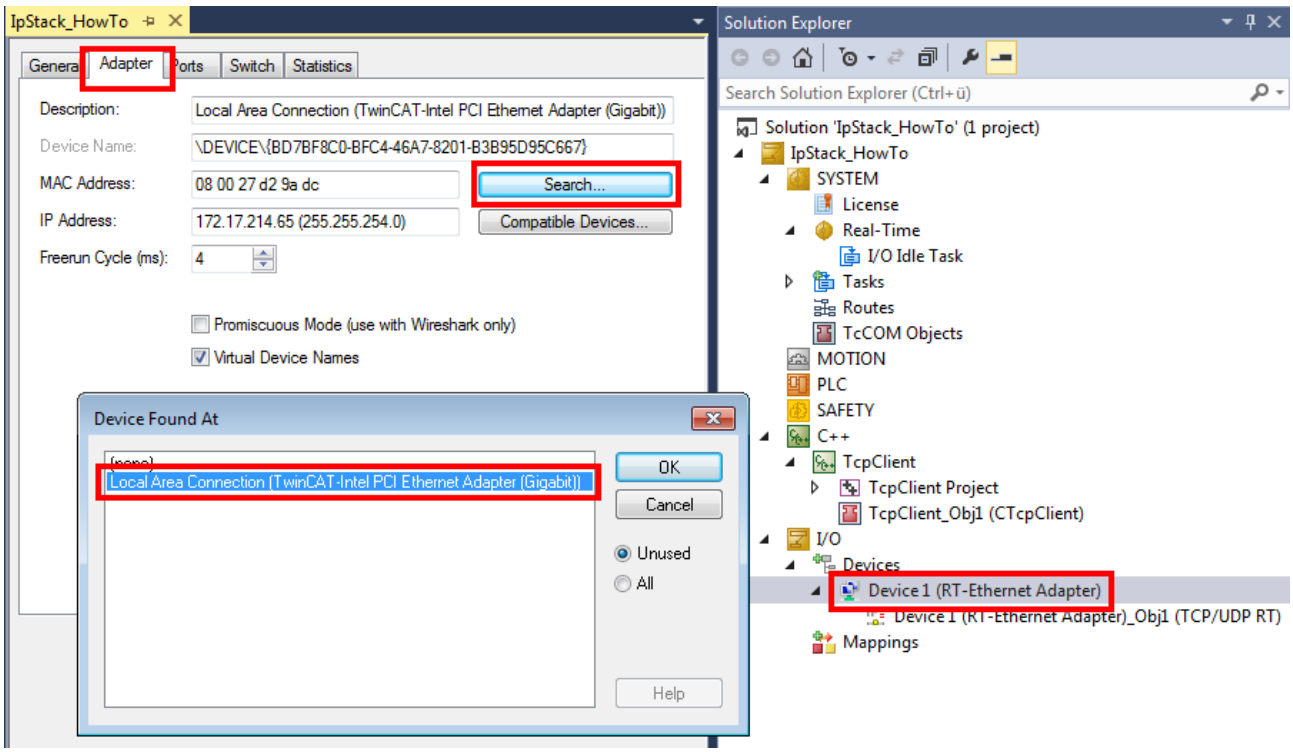
werden von dem Interface ITcUnknown geerbt und sind hier nicht relevant. Als Hintergrund kann das Kapitel über das TcCOM-Modul-Konzept im C++ Bereich betrachtet werden.

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



6.2.2 S02: UDP Demo (C++)

Dieses Beispiel beschreibt einen UDP Server, der in C++ implementiert ist.

Er empfängt UDP Pakete und sendet sie zurück zum Absender („Echo-Server“).

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S02-UdpDemo

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen
4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

Analog zum Quick Start [▶ 23] wird in diesem Beispiel das Interface ITcIoUdpProtocolRecv [▶ 54] implementiert und ein Pointer auf ein ITcIoUdpProtocol [▶ 54] verwendet.

In der Transition SO wird durch die Nutzung von „RegisterReceiver“ erreicht, dass das Modul für den übermittelten Port (Standard: 10000) angemeldet wird. In der Transition OS wird eine entsprechende Abmeldung vorgenommen.

In der „CycleUpdate“ Methode wird die „CheckReceived“ Methode aufgerufen. Dem TCP/UDP RT Modul wird so die Möglichkeit gegeben, ankommende Pakete zu bearbeiten und Callbacks auf die Methode „ReceiveData“ dem Modul zu übermitteln.

Die „ReceiveData“ Methode nutzt die „SendData“ Methode, um die Daten zum Absender zurückzusenden („Echo-Server“).

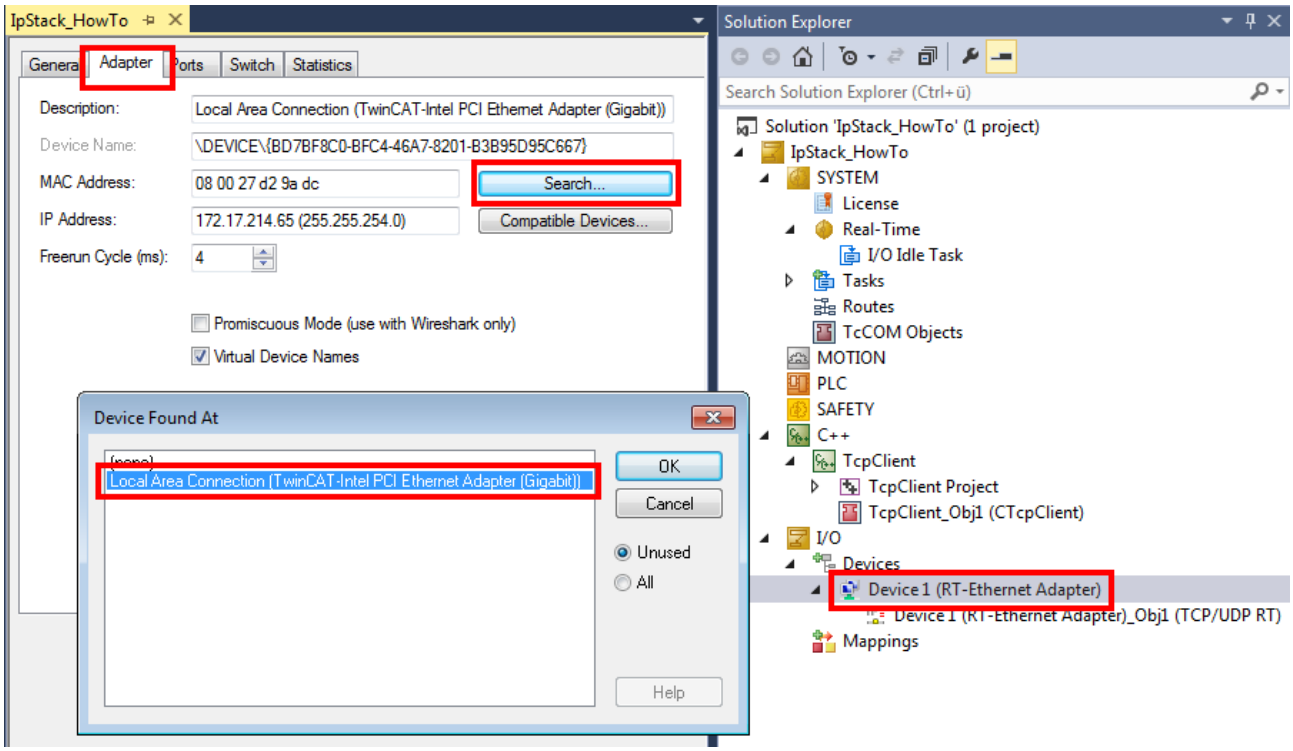
Das Sample ist auch ausführlich als Quick Start [▶ 17] verfügbar.

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



6.2.3 Test Client

Der Testclient dient dazu, einzelne UDP Datenpakete an einen UDP Server zu senden und zu empfangen.

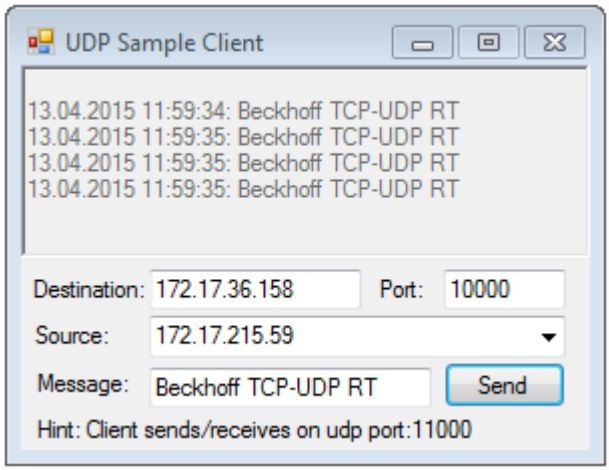
Download

Download des Test Clients.

Entpacken Sie die ZIP Datei; die .exe ist auf einem Windows-System ausführbar.

Beschreibung

Der Client selber nutzt den Port 11000 zum Senden. Gleichzeitig öffnet er diesen Port und zeigt empfangene Nachrichten im oberen Teil der Oberfläche als Protokoll an:



Zusammen mit den PLC / C++ Beispielen, ergibt sich somit ein Echo-Beispiel:
Eine UDP Nachricht wird von dem Client Port 11000 an den Server Port 10000 gesendet, der die gleichen Daten an den Absender zurück sendet.

Der Client ist über die Oberfläche konfigurierbar:

- Destination: Ziel IP Adresse
- Port: Port, der im Ziel angesprochen wird
- Source: Absender-Netzwerkkarte (IP-Adresse).
  „OS-based“: Betriebssystem übernimmt Auswahl der passenden Netzwerkkarte.
- Nachricht (Message)

Das TF6311 „TCP/UDP Realtime“ erlaubt keine lokale Kommunikation. Zu Testzwecken kann jedoch durch „Source“ eine andere Netzwerkschnittstelle ausgewählt werden, sodass das UDP Paket den Rechner über die eine Netzwerkkarte verlässt und auf der anderen Netzwerkarte eintrifft („Loop-Kabel“).

6.3 S03: ARP PING Demo (C++)

Dieses Beispiel beschreibt einen ARP und PING Client.

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S03-PingClient

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen
4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

Analog zum Quick Start [▶ 23] wird in diesem Beispiel das Interface ITcIoArpPingProtocolRecv: Methoden [▶ 67] implementiert und ein Pointer auf ein ITcIoArpPingProtocol: Methoden [▶ 67] verwendet.

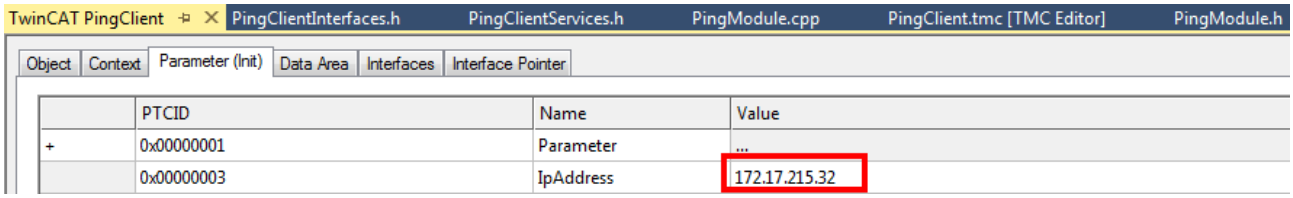
In der Transition SO wird durch die Nutzung von „RegisterReceiver“ erreicht, dass das Modul für den Empfang von Arp und Ping Nachrichten angemeldet wird. In der Transition OS wird eine entsprechende Abmeldung vorgenommen.

In der „CycleUpdate“ Methode wird die „CheckReceived“ Methode aufgerufen. Dem TCP/UDP RT Modul wird so die Möglichkeit gegeben, ankommende Pakete zu bearbeiten und durch Callbacks auf die Methoden „ArpReply“ und „PingReply“ dem Modul zu übermitteln.

Verständnis

Der Ablauf erfolgt einmalig beim Start des Programms.

Wenn „m_bSendRequest“ auf TRUE gesetzt wird (z.B. durch das TwinCAT Live Watch), wird ein neuer Request (ARP und Ping) an die IP Adresse gesendet, die hier definiert wurde:



Die Ausgabe erfolgt in den Meldungen:

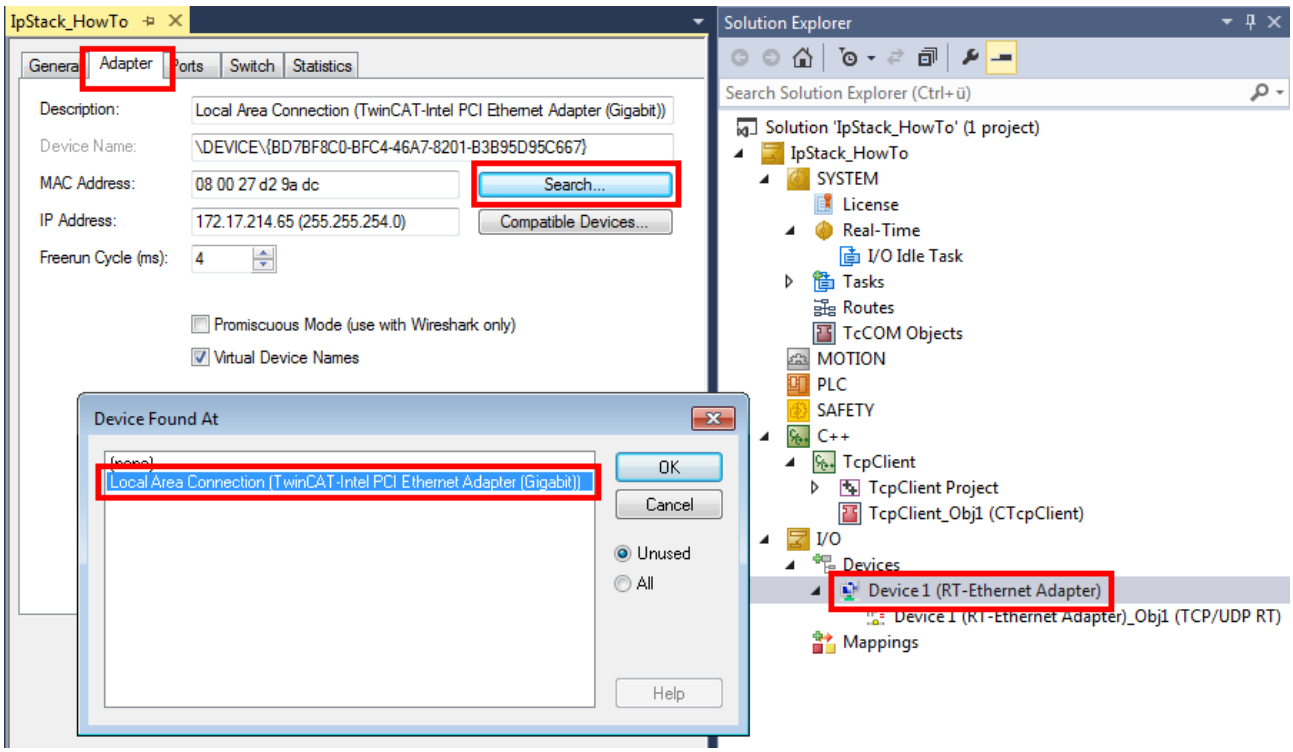
Für die Ausgabe dieser Meldungen kann der „Tracelevel“ (auf Info) konfiguriert werden.

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



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

Dieses Beispiel beschreibt einen TCP Server, welcher eine einkommende Verbindung annimmt. Daten, die an diesen Server gesendet werden, werden als „Echo“ einfach zurück gesendet.

Das Sample ist für C++ und für PLC äquivalent vorhanden. Der Server läuft standardmäßig auf Port 11000.

Testen des Samples

Das Sample kann durch „telnet“ getestet werden.

```
%>telnet 192.168.1.1 11000
```

Wenn ein Zeichen per telnet gesendet wird, wird dieses sofort rückgesendet. Somit ergibt sich ein Bild ähnlich zu dem Folgenden:

6.4.1 S04: TCP Server Demo (PLC)

Dieses Beispiel beschreibt einen TCP Server, der in einem PLC Projekt implementiert ist.

Es nimmt eine TCP Verbindung entgegen, empfängt TCP Pakete und sendet sie zurück zum Absender („Echo-Server“).

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S04-TCPServerPlc

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen
4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

Analog zu den Quick Start [▶ 11]s wird in diesem Beispiel das Interface ITcIoTcpProtocolRecv [▶ 60] implementiert und ein Pointer auf ein ITcIoTcpProtocol [▶ 61] verwendet.

Dafür wird ein PLC Baustein angelegt, der das Interface ITcIoUdpProtocolRecv [▶ 60] implementiert („Add POU“ mit „Implements“). Dabei ist es wichtig in den Methoden „FB_init“ und „FB_exit“ die Verbindung zum TCP/UDP RT Objekt zu realisieren. Die Quick Starts zeigen insbesondere, wie dieses OnlineChange sicher realisiert werden kann. Das allgemeine Vorgehen ist ausführlicher im Sample 11 der C++ Dokumentation beschrieben.

Der implementierende Funktionsbaustein (im Sample TCPServer) ruft die Methode „CheckReceived“ auf. Dem IP Stack wird so die Möglichkeit gegeben, ankommende Pakete zu bearbeiten und Callbacks auf die Methoden „ReceiveData“ und „ReceiveEvent“ des Funktionsbausteins zu übermitteln.

Um auf eingehende Verbindungen zu achten, wird per „AllocSocket“ und „Listen“ in der FB_init ein Port geöffnet. In der „ReceiveEvent“ wird „Accept“ aufgerufen, falls ein Event zum Verbindungsaufbau aufgetreten ist.

Die „ReceiveData“ Methode nutzt in diesem Beispiel die „SendData“ Methode um die Daten zum Absender zurückzusenden („Echo-Server“).

Verständnis

Damit die Kommunikation zwischen Funktionsbaustein und TcCOM Objekt „TCP/UDP RT“ aufgebaut wird, werden zwei Methoden verwendet:

- „FB_init“: Diese wird automatisch ausgeführt, wenn die PLC gestartet wird.
- „FB_exit“: Diese wird automatisch ausgeführt, wenn die PLC angehalten wird.

Diese Initialisierungsphase kann größtenteils aus dem Beispiel-Code übernommen werden.

Für die eigentliche TCP Funktionalität sind im PLC Code zwei Methoden zuständig:

- Die „ReceiveData“ Methode am implementierten Funktionsbaustein empfängt die Daten.
- Die „ReceiveEvent“ Methode am implementierten Funktionsbaustein signalisiert auftretende Events.
- Die “SendData” Methode am ITcIoTcpProtocol Interface sendet Daten.

In dem Beispiel wird in der „ReceiveData“ Methode die „SendData“ Methode verwendet um die empfangenen Daten zurückzusenden.
Die Methode TcQueryInterface muss folgendermaßen implementiert werden, damit TwinCAT erkennt, dass das entsprechende Interface implementiert wurde:

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


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

Die zusätzlich angelegten Methoden

- TcAddRef / TcRelease

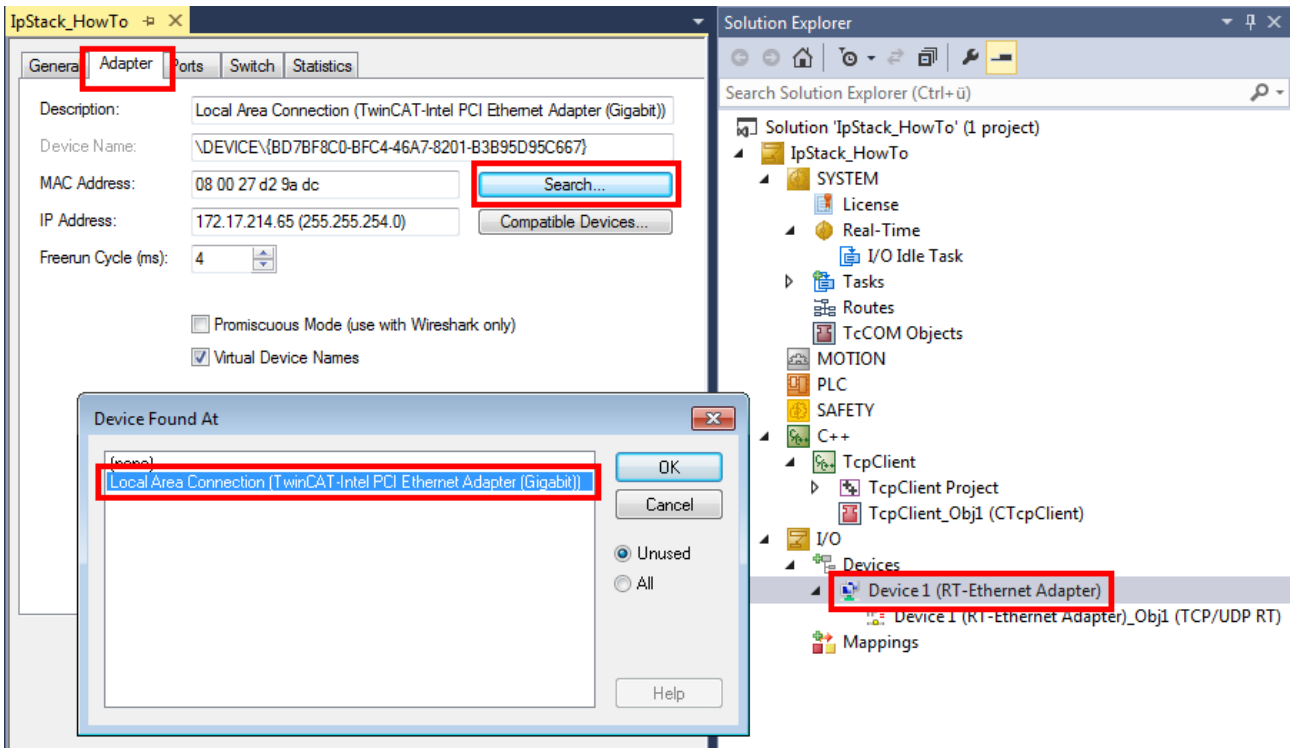
werden von dem Interface ITcUnknown geerbt und sind hier nicht relevant. Als Hintergrund kann das Kapitel über das TcCOM-Modul-Konzept im C++ Bereich betrachtet werden.

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



6.4.2 S04: TCP Server Demo (C++)

Dieses Beispiel beschreibt einen TCP Server, der in C++ implementiert ist.

Es nimmt eine TCP Verbindung entgegen, empfängt TCP Pakete und sendet sie zurück zum Absender („Echo-Server“).

Download

Download des Beispiels: https://github.com/Beckhoff/TF6311_Samples/tree/main/S04-TCPServer

1. Beispiel von GitHub beziehen, die heruntergeladene ZIP-Datei ggf. entpacken
2. Projekt mit TwinCAT XAE öffnen
3. Ihr Zielsystem auswählen
4. Konfiguration der Netzwerkkarte (siehe unten) für das Zielsystem vornehmen
5. Das **Beispiel auf Ihrer lokalen Maschine bauen** (z.B. Build->Build Solution)
6. Die Konfiguration aktivieren

Beschreibung

Analog zu den Quick Starts wird in diesem Beispiel das Interface ITcIoTcpProtocolRecv implementiert und ein Pointer auf ein ITcIoTcpProtocol verwendet.

In der „CycleUpdate“ Methode wird die „CheckReceived“ Methode aufgerufen. Dem TCP/UDP RT Modul wird so die Möglichkeit gegeben, ankommende Pakete zu bearbeiten und Callbacks auf die Methoden „ReceiveEvent“ und „ReceiveData“ dem Modul zu übermitteln.

Um auf eingehende Verbindungen zu achten, wird per „AllocSocket“ und „Listen“ in der „CycleUpdate“ auch ein Port geöffnet. In der „ReceiveEvent“ wird „Accept“ aufgerufen, falls ein Event zum Verbindungsaufbau aufgetreten ist.

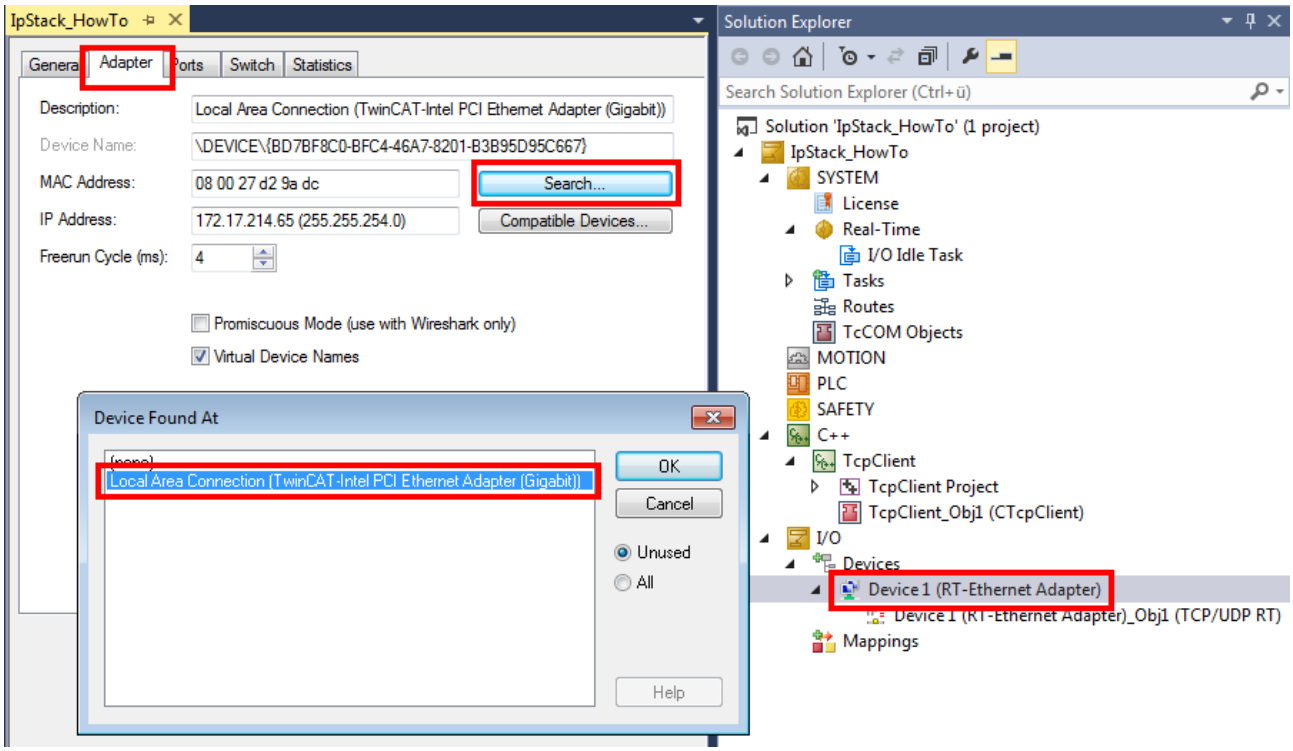
Die „ReceiveData“ Methode nutzt in diesem Beispiel die „SendData“ Methode, um die Daten zum Absender zurückzusenden („Echo-Server“).

Vorbereitung Netzwerkkarte

Stellen Sie für das TCP/UDP RT Modul sicher, dass der RT-Ethernet-Adapter in der TwinCAT Solution auf die richtige Netzwerkkarte (mit TwinCAT Treiber) verbunden ist.

Nur Lokale Konfiguration

Die Installation des Treibers auf kompatiblen Netzwerkkarten über den Button „Compatible Devices“ erfolgt immer lokal. Auf einer Steuerung mit TwinCAT XAR kann das mitinstallierte Programm TcRteInstall.exe (normalerweise unter C:\TwinCAT\3.1\System) genutzt werden.



7 Programmierreferenz

Die Programmierreferenz stellt eine Übersicht der unterschiedlichen Parameter, Interfaces und ihrer Methoden dar.

Hierzu gehören:

- TCP/UDP RT TcCOM Parameters [▶ 57]: Die Parameter des eigentlichen TCP/UDP RT Moduls ermöglichen die Konfiguration.

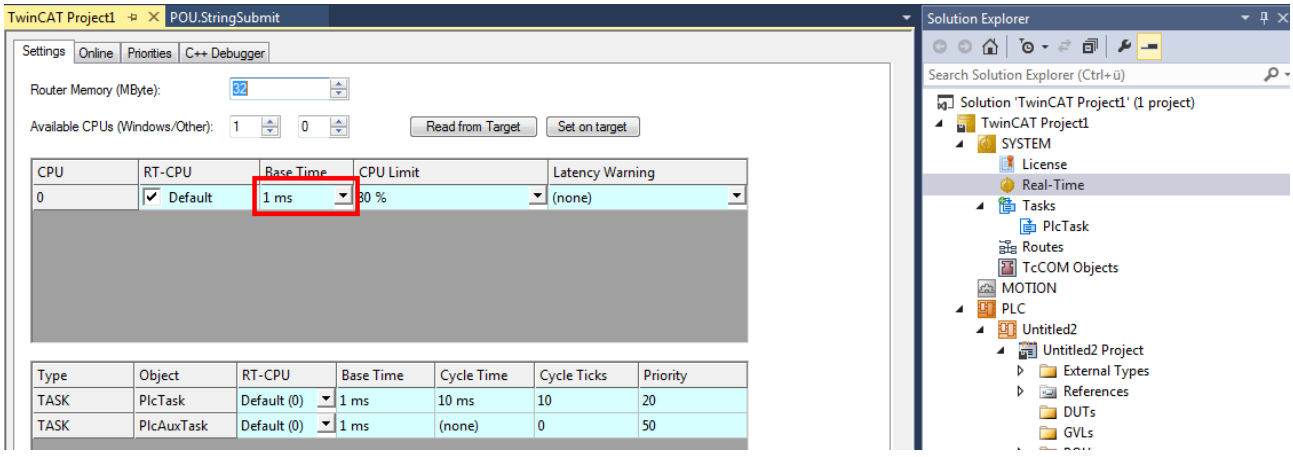
Der TCP/UDP RT Modul kann durch unterschiedliche Protokolle verwendet werden. Es gehören immer ein InterfacePointer und ein zu implementierendes Interface zusammen:

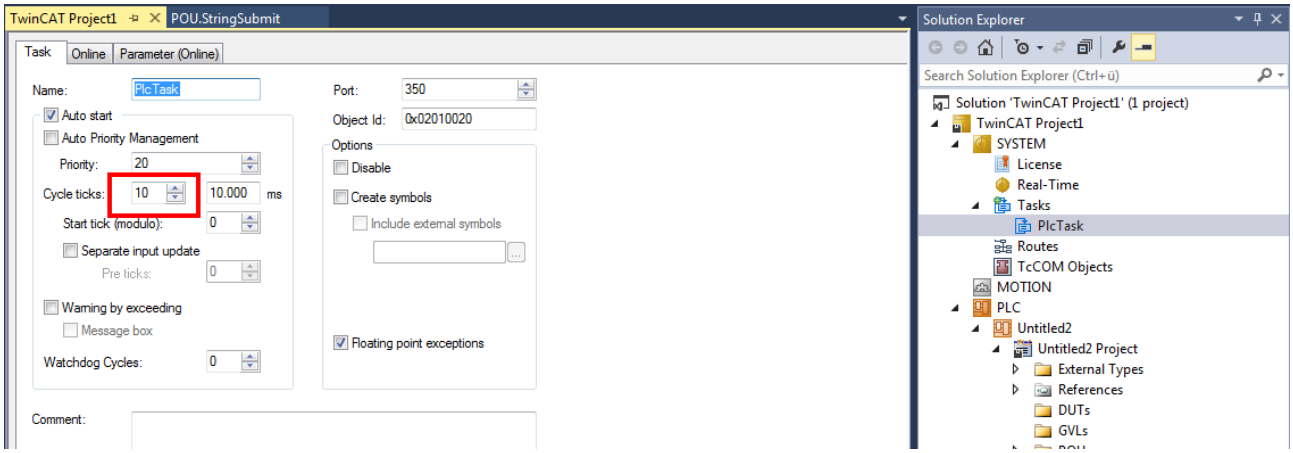
- ITcIoTcpProtocol(Recv): [▶ 60] TCP/IP Protokoll
- ITcIoUdpProtocol(Recv) [▶ 54]: UDP/IP Protokoll
- ITcIoArpPingProtocol(Recv) [▶ 67]: ARP/Ping Protokoll

Für alle Verwendungen von IP Adressen (z.B. „IpAddr“) werden die höchstwertige Element an letzter Stelle dargestellt. (Beispiel: 192.168.2.1 -> 01 02 A8 C0)

Performance

Das TCP/UDP RT TcCOM Objekt läuft in der Echtzeit. Damit ist das Modul auch direkt von der Taktung der Echtzeit abhängig. Die Frequenz mit der Daten kommuniziert werden können, ist also durch die Taktung der verwendeten Task (und damit auch der Echtzeiteinstellungen) beeinflussbar:





Die Kommunikation über die Netzwerkschnittstelle hängt von diesem Zyklus ab. Es muss in jedem Zyklus eine entsprechender Aufruf der CheckReceived() Methoden (siehe API Dokumentation [▶ 53]) erfolgen.

Einkommende Daten: CheckReceived()

Context der Eintreffenden Daten

Es muss durch den Kunden sichergestellt werden, dass die Methode CheckReceived zyklisch aufgerufen wird. Beispiele zeigen das Vorgehen in PLC und C++

Damit die Daten im gleichen Context bereitgestellt werden können wie das Kundenprojekt läuft, wird zyklisch die CheckReveived()-Methode aufgerufen. Innerhalb dieses Methodenaufrufs werden, falls Daten empfangen wurden, die Protokoll-abhängigen Receive() Methoden des Kundenprojektes aufgerufen.

Abbruch der Engineering Verbindung bei Breakpoints

Beim Arbeiten mit Breakpoints sollten unbedingt unterschiedliche Netzwerkschnittstellen genutzt werden, da ein Breakpoint Teile des TwinCAT Systems anhält, was auch die Kommunikation zum Engineering betreffen kann.

7.1 UDP/IP: ITcIoUdpProtocol(Recv)

Die Interfaces ITcIoUdpProtocol und ITcIoUdpProtocolRecv ermöglichen eine UDP/IP-Kommunikation aus der Echtzeitumgebung heraus.

Ein Projekt, das dieses Interface verwendet, enthält einen Pointer auf ein ITcIoUdpProtocol-Objekt und implementiert ITcIoUdpProtocolRecv selbst. ITcIoUdpProtocolRecv dient als Callback-Interface, um Daten innerhalb der Applikation vom TCP/UDP RT Modul empfangen zu können.

Mehrfach-Aufrufe von Receive()

Es muss bei der Implementierung beachtet werden, dass durch das CheckReceived() der Callback auf das Receive() mehrfach in einem Zyklus auftreten wird, falls zwischen den Zyklen mehrere Pakete eingetroffen sind.
Bei der Implementierung ist somit ggf. ein Zwischenspeicher in Form einer Queue vorzusehen.

ITcIoUdpProtocolRecv Methoden:

Name	Description
ReceiveData [▶ 55]	Wird vom TCP/UDP RT-Modul als Callback aufgerufen, um Daten zu übergeben

ITcIoUdpProtocol Methoden:

Name	Description
SendData [▶ 55]	Sendet Daten
CheckReceived [▶ 56]	Muss zyklisch aufgerufen werden. ReceiveData wird im Context dieser Methode als Callback genutzt (Server und Client-Funktionalität).
RegisterReceiver [▶ 56]	Registrieren am TCP/UDP RT-Modul für den Empfang von Daten.
UnregisterReceiver [▶ 57]	De-Registrieren am TCP/UDP RT-Modul für den Empfang von UDP-Daten.

Hier in Stichpunkten der Ablauf einer Client-/ und Serverimplementierung. Es soll nur ein Überblick gegeben werden, die Beispiele zeigen die konkrete Verwendung.

Implementierung eines UDP-Senders/-Empfängers

Name	Beschreibung
RegisterReceiver [▶ 56]	Öffnet einen Port für einkommende Datenpakete.
ReceiveData [▶ 55]	Wird aufgerufen, wenn Datenpakete ankommen.
SendData [▶ 55]	Kann genutzt werden, um Daten zu versenden.
UnregisterReceiver [▶ 57]	Zum Abmelden (Schließen) des Ports z.B. beim Runterfahren.

Zum Empfang von UDP-Daten ist eine Registrierung durch den Aufruf von RegisterReceiver erforderlich. Dieses kann in SetObjStateSO bzw. FB_init erfolgen.

Daten werden durch einen Callback der Methode ReceiveData von ITcIoUdpProtocolRecv bereitgestellt.

Während TwinCAT vom RUN Mode in den Config Mode geht, sollten sich alle Module durch UnregisterReceiver abmelden. Dieses kann in SetObjStateOS() bzw. FB_exit erfolgen.

HINWEIS

OnlineChange-Sicherheit

Zur OnlineChange Sicherheit sollte RegisterReceiver erneut aufgerufen werden.

7.1.1 Methode ITcIoUdpProtocolRecv:ReceiveData

Wird vom TCP/UDP RT Modul als Callback aufgerufen, um Daten zu übergeben.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg und muss vom implementierenden Modul entsprechend geliefert werden.

Parameter

Name	Typ	Beschreibung
ipAddr	ULONG	Die IP-Adresse des Absenders. IP Adressen werden mit höchstwertige Element an letzter Stelle dargestellt. (Beispiel: 192.168.2.1 -> 01 02 A8 C0)
udpDestPort	USHORT	Port, auf dem die Daten empfangen wurden.
udpSrcPort	USHORT	Port des Absenders.
nData	ULONG	Anzahl der empfangenen Bytes.
pData	PVOID	Pointer auf die empfangenen Daten.
pVlan	ETYPE_VLAN_HEADER	Struktur ETYPE_VLAN_HEADER - siehe unten.

Der VLAN Header repräsentiert Informationen über das VLAN.

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

7.1.2 Methode ITcIoUdpProtocol:SendData

Sendet Daten.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
ipAddr	ULONG	Die IP Adresse des Empfängers. IP Adressen werden mit höchstwertige Element an letzter Stelle dargestellt. (Beispiel: 192.168.2.1 -> 01 02 A8 C0)
udpDestPort	USHORT	Der Port des Empfängers.
udpSrcPort	USHORT	Der Port des Absenders.
nData	ULONG	Anzahl der zu versendenden Daten in Bytes.
pData	PVOID	Pointer auf die zu versendenden Daten.
bCalcUdpCheckSum	BOOL	Gibt an, ob die Checksumme berechnet werden soll.
pVlan	ETYPE_VLAN_HEADER	Struktur ETYPE_VLAN_HEADER, siehe unten.

Der VLAN Header repräsentiert Informationen über das VLAN.

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

7.1.3 Methode ITcIoUdpProtocol:CheckReceived

Muss zyklisch aufgerufen werden, ReceiveData wird im Context dieser Methode als Callback genutzt (Senden und Empfangen).

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

7.1.4 Methode ITcIoUdpProtocol:RegisterReceiver

Registrieren am TCP/UDP RT Modul für den Empfang von Daten.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
udpPort	USHORT	Port auf dem die Daten empfangen werden sollen.
ipRecv	ITcIoUdpProtocolRecv*	Pointer zu dem Empfänger (Recv) Interface.

7.1.5 Methode ITcIoUdpProtocol:UnregisterReceiver

De-Registrieren am TCP/UDP RT Modul für den Empfang von Daten.

Syntax

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

Rückgabewert

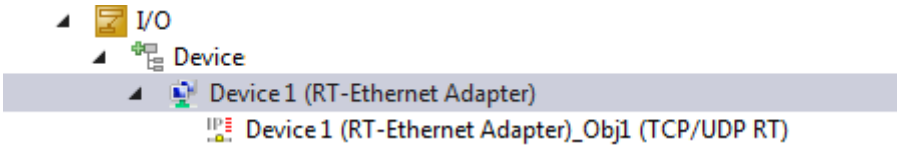
Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

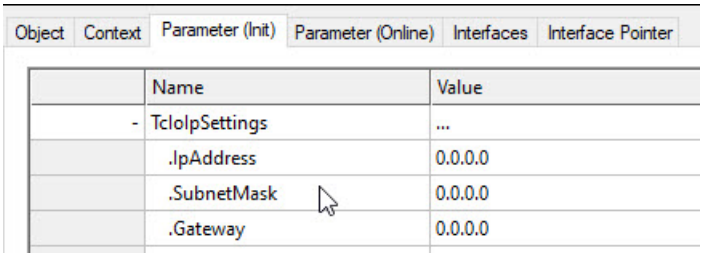
Name	Typ	Beschreibung
udpPort	USHORT	Port auf dem die Daten nicht weiter empfangen werden sollen.

7.2 TCP/UDP RT TcCom Parameter

Neben den Interfaces ist das TcCOM Objekt „TCP/UDP RT“ die wesentliche Komponente der Function. Eine Instanziierung wird normalerweise unterhalb des Devices vorgenommen:



Durch einen Doppelklick wird die Instanz geöffnet und die Parameter, die im Folgenden dokumentiert sind, können genutzt werden:



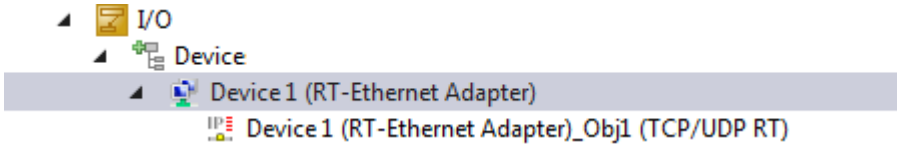
Name	Default-Wert	Beschreibung
TcIoIpSettings.IpAddress	0.0.0.0	Eigene (lokale) IP-Adresse, die zur Kommunikation genutzt wird.
TcIoIpSettings.SubnetMask	0.0.0.0	Eigene Subnetzmaske
TcIoIpSettings.Gateway	0.0.0.0	Gateway, welches genutzt wird, um Kommunikationspartner außerhalb des eigenen Netzes zu erreichen.
TcIoIpSettings.DhcpEnable	FALSE	Noch nicht implementiert.
TcIoIpSettings.ManualSettings	FALSE	Auf FALSE gesetzt: Es wird die aktuelle IP-Konfiguration des referenzierten Adapters vom Betriebssystem verwendet. Auf TRUE gesetzt: Parameter von TcIoIpSettings* werden genutzt.
IpMaxReceivers	4	Max. Anzahl der maximal erlaubten IP-basierenden Protokolle.
IpMaxPendingOnArp	40	Max. Anzahl der Einträge in der ARP Request Table.
IpMacCacheSize	64	Anzahl der Einträge in MAC-Cache, also der Zuordnungen IP Adresse zu MAC Adresse. Caching ist implementiert als LRU.
IpMTU	1514	Noch nicht implementiert. (Maximum Transport Unit Größe für IP Pakete)
IpRecvFrameQueueSize	255	Anzahl der Einträge in der Queue zum Empfang von Udp Paketen.
UdpMaxReceivers	4	Max. Anzahl der UDP-Empfänger
UdpMTU	1514	Ab TwinCAT 3.1 Build 4026: Maximum Transport Unit Größe für UDP. Fragmentierung steht bereit. In früheren Versionen (<= Build 4024) ist dieser Parameter ohne Funktion
UdpCheckCrc	TRUE	Auf TRUE gesetzt bedeutet, dass UDP Pakete mit falscher Checksum verworfen werden.
TTL	0x80	TTL im IP-Header der zu versendenden Frames.
MultiCastTTL	0x01	TTL der zu sendenden MultiCast Frames.
PassiveMode	FALSE	Bei TRUE werden durch diese Instanz keine Frames vom RT-Netzwerkadapter Frames abgeholt. Siehe Multitask Zugriff auf eine Netzwerkkarte [▶ 38]
MulticastIpList	[]	Multicast-Adressen zum Empfang von MultiCast Paketen.
TcpMTU	1514	Noch nicht implementiert. (Maximum Transport Unit Größe für TCP)
TcpCheckCrc	TRUE	Eingehende TCP-Frames werden auf gültige Checksumme überprüft und ggf. verworfen.
TcpMaxSocketCount	32	Max. Anzahl von Sockets, die von dem IP Stack verwaltet werden.
TcpReceiveBufferSize	16192	Anzahl empfangener Bytes die bei einer TCP-Verbindung zwischengespeichert werden können.
TcpTransmitBufferSize	16192	Anzahl zu sendender Bytes die bei einer Verbindung im TCP-Stack zwischengespeichert werden können.
TcpMaxRetry	5	Anzahl der Wiederholungen von TCP-Paketen bis die Verbindung beendet wird.
TcpTimeoutCon	5000	Timeout für TCP Verbindungsaufbau und -abbau.
TcpTimeoutWait	60000	Zeitspanne, wie lange Handles intern gehalten werden nach einem unerwarteten Abbruch der Verbindung.
TcpTimeoutIdle	1000	Zeitspanne bis zum Callback (ReveiveEvent), wenn keine Antwort erfolgt.

Name	Default-Wert	Beschreibung
TcpRoundTripTime	3000	Startwert für den Timeout von Datenpaketen. Wird dynamisch nach Verbindungsqualität angepasst (je nach Paketumlaufzeit).

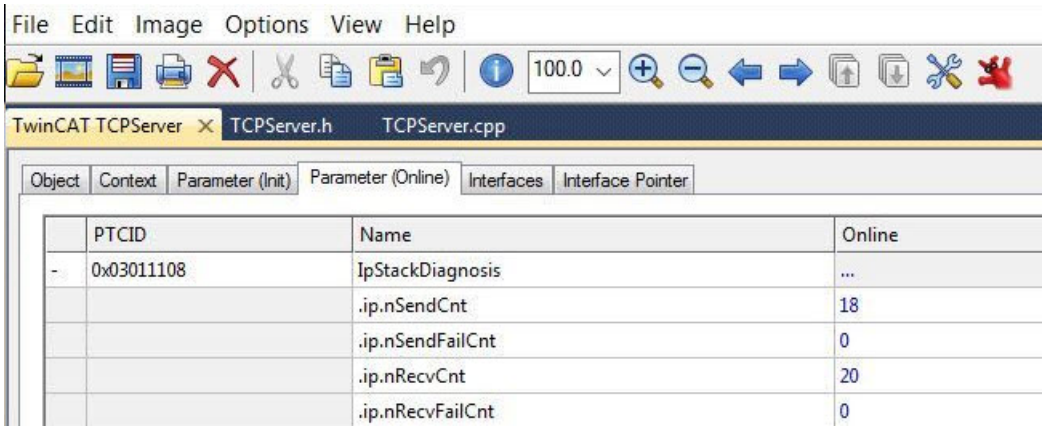
Zeiten sind in Millisekunden angegeben.

7.3 TCP/UDP RT TcCom Diagnose

Das TcCOM Objekt TCP/UDP RT stellt die Kopplung des Kundenprojektes an die Hardware dar.



Deswegen besitzt es neben den Parametern auch Diagnose-Informationen, welche hier beschrieben sind. Wenn das Engineering mit dem Zielsystem kommunizieren kann und das Programm entsprechend läuft, werden unterschiedliche Informationen über die empfangenen und versendeten Pakete bereitgestellt:



Name	Wert	Beschreibung
IpStackDiagnosis	…	Diagnose Informationen des IP Stacks
.ip.nSendCnt	18	Anzahl der gesendeten IP Pakete
.ip.nSendFailCnt	0	Anzahl der nicht gesendeten IP Pakete
.ip.nRecvCnt	20	Anzahl der empfangenen Pakete
.ip.nRecvFailCnt	0	Anzahl der nicht empfangenen Pakete
.arpRequest.nSendCnt	0	Arp-Requests: Anzahl der gesendeten Pakete
.arpRequest.nSendFailCnt	0	Arp-Requests: Anzahl der nicht gesendeten Pakete
.arpRequest.nRecvCnt	12	Arp-Requests: Anzahl der empfangenen Pakete
.arpRequest.nRecvFailCnt	0	Arp-Requests: Anzahl der nicht empfangenen Pakete
.arpReply.nSendCnt	12	Arp-Reply: Anzahl der gesendeten Pakete
.arpReply.nSendFailCnt	0	Arp-Reply: Anzahl der nicht gesendeten Pakete
.arpReply.nRecvCnt	0	Arp-Reply: Anzahl der empfangenen Pakete
.arpReply.nRecvFailCnt	0	Arp-Reply: Anzahl der nicht empfangenen Pakete
.pingRequest.nSendCnt	0	Ping-Request: Anzahl der gesendeten Pakete
.pingRequest.nSendFailCnt	0	Ping-Request: Anzahl der nicht gesendeten Pakete
.pingRequest.nRecvCnt	0	Ping-Request: Anzahl der empfangenen Pakete
.pingRequest.nRecvFailCnt	0	Ping-Request: Anzahl der nicht empfangenen Pakete
.pingReply.nSendCnt	0	Ping-Reply: Anzahl der gesendeten Pakete
.pingReply.nSendFailCnt	0	Ping-Reply: Anzahl der nicht gesendeten Pakete
.pingReply.nRecvCnt	0	Ping-Reply: Anzahl der empfangenen Pakete
.pingReply.nRecvFailCnt	0	Ping-Reply: Anzahl der nicht empfangenen Pakete
.nLinkStatusChangedCnt	1	Anzahl der Link-Änderungen
.nAllocFailCnt	0	Anzahl der fehlgeschlagenen Allokationen
.nArpTimeoutFrames	0	Anzahl der Arp-Frames im Timeout
.nDroppedFrames	0	Anzahl der verworfenen Pakete

7.4 TCP/IP: ITcIoTcpProtocol(Recv)

Die Interfaces ITcIoTcpProtocol und ITcIoTcpProtocolRecv ermöglichen eine TCP/IP Kommunikation aus der Echtzeitumgebung heraus.

Ein Projekt, welches dieses Interface verwendet, enthält einen Pointer auf ein ITcIoTcpProtocol Objekt und implementiert ITcIoTcpProtocolRecv selbst. ITcIoTcpProtocolRecv dient als Callbackinterface um Daten und Events innerhalb der Applikation vom TCP/IP Modul empfangen zu können. Die Interfaces sind an eine Socket API angelehnt.
Bevor ein Socket genutzt werden kann, muss dieser mit AllocSocket() allokiert werden.

ITcIoTcpProtocolRecv Methoden:

Name	Description
ReceiveData [▶ 62]	Wird vom TCP/UDP RT Modul als Callback aufgerufen, um Daten zu übergeben.
ReceiveEvent [▶ 62]	Wird vom TCP/UDP RT Modul als Callback aufgerufen, falls ein Event aufgetreten ist.

ITcIoTcpProtocol Methoden:

Name	Description
AllocSocket [▶ 63]	Allokiert einen Socket.
FreeSocket [▶ 63]	Gibt einen Socket frei.
Connect [▶ 64]	Baut eine Verbindung zu einer Gegenstelle auf.
IsConnected [▶ 64]	Gibt Auskunft, ob ein Socket verbunden ist (für eingehende und ausgehenden Verbindungen).
Close [▶ 64]	Schließt einen Socket.
Listen [▶ 65]	Öffnet einen TCP Port für eingehende Verbindungen (siehe Remarks).
Accept [▶ 65]	Für Serverfunktionalität: Akzeptiert einkommende Verbindungen (siehe Remarks).
SendData [▶ 65]	Sendet Daten (Server und Client-Funktionalität).
CheckReceived [▶ 66]	Muss zyklisch aufgerufen werden; ReceiveEvent und ReceiveData werden im Context dieser Methode als Callback genutzt (Server und Client-Funktionalität).
GetRemoteIpAddr [▶ 66]	Liefert die entfernte IP Adresse eines Kommunikationspartners.
GetFreeSendDataSize [▶ 66]	Liefert die Anzahl an freien Bytes im TCP Sendebuffer.

CheckReceived() kontinuierlich aufrufen.

AllocSocket() ggf. erneut bei OnlineChange aufrufen um das Ziel der Callbacks zu erneuern.

Hier wird Programmiersprachen-unabhängig der Ablauf einer Client-/ und Serverimplementierung beschrieben.
Dabei soll nur ein Überblick gegeben werden, die Beispiele zeigen die konkrete Verwendung.

Implementierung eines TCP Servers:

Name	Beschreibung
AllocSocket [▶ 63]	Öffnet einen Socket.
Listen [▶ 65]	Öffnet einen Port auf dem Verbindungen erwartet werden.
Accept [▶ 65]	In der ReceiveEvent()-Methode aufgerufen um eine Verbindung anzunehmen.
ReceiveData [▶ 62]	Wird bei empfangenen Daten aufgerufen.
SendData [▶ 65]	Kann genutzt werden, um Daten zu versenden.
FreeSocket [▶ 63]	Auf den Listen-Socket sowie alle Verbindungs-Sockets zum Beenden.

Code-Skizze für das Akzeptieren einer Verbindung:

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

Implementierung eines TCP Clients:

Name	Beschreibung
AllocSocket [▶ 63]	Öffnet einen Socket.
Connect [▶ 64]	Startet den Verbindungsaufbau.
IsConnected [▶ 64]	Überprüft, ob die Verbindung erfolgreich aufgebaut wurde.
ReceiveData [▶ 62]	Wird bei empfangenen Daten aufgerufen.
SendData [▶ 65]	Kann genutzt werden, um Daten zu versenden.
FreeSocket [▶ 63]	Auf den Listen-Socket sowie alle Verbindungs-Sockets zum Beenden.

Verbindungsabbruch durch Betriebssystem bei Promiscuous Mode

Wenn an dem RT-Ethernet Adapter im Tab „Adapter“ der Promiscuous Mode eingeschaltet ist, werden eintreffende TCP Verbindungsaufbauten durch das Betriebssystem abgebrochen, da dieses einen im TCP/UDP RT Objekt geöffneten Port nicht kennt.

7.4.1 Methode ITcIoTcpProtocolRecv:ReceiveData

Wird vom TCP/UDP RT Modul als Callback aufgerufen, um Daten zu übergeben.

Syntax

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

Rückgabewert

Name	Typ	Beschreibung
ReceiveData	HRESULT	Bezeichnet den Erfolg und muss vom implementierenden Modul entsprechend geliefert werden.

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der Socket auf dem Daten empfangen wurden.
nData	ULONG	Anzahl der empfangenen Daten.
pData	PVOID	Pointer auf die empfangenen Daten.

7.4.2 Methode ITcIoTcpProtocolRecv:ReceiveEvent

Wird vom TCP/UDP RT Modul als Callback aufgerufen, falls ein Event aufgetreten ist.

Syntax

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

Rückgabewert

Name	Typ	Beschreibung
ReceiveEvent	HRESULT	Bezeichnet den Erfolg und muss vom implementierenden Modul entsprechend geliefert werden.

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der Socket auf dem Daten empfangen wurden.
tcpEvent	TCP_EVENT	Ein Element der Enum.

Das Enumeration TCP_EVENT bezeichnet unterschiedliche Ereignisse, die bei einer TCP-Verbindung auftreten können:

```
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
};
```

Eine Implementierung der Methode soll damit ein Switch-Case über alle Elemente bereitstellen, sodass entsprechend des Events reagiert werden kann.

Zur Verwendung der Events für ein TCP-Server wird die Verwendung in der Interface-Übersicht beschrieben.

7.4.3 Methode ITcIoTcpProtocol:AllocSocket

Allokiert einen Socket.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
ipRecv	ITcIoTcpProtocolRecv	Pointer zu dem Empfänger (Recv) Interface.
socketId	ULONG&	Der erzeugte Socket.

7.4.4 Methode ITcIoTcpProtocol:FreeSocket

Gibt einen Socket frei.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der freizugebende Socket.

7.4.5 Methode ITcIoTcpProtocol:Connect

Baut eine Verbindung zu einer Gegenstelle auf.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.
ipRemoteAddress	ULONG	IP-Adresse der zu kontaktierenden Gegenstelle. IP Adressen werden mit höchstwertige Element an letzter Stelle dargestellt. (Beispiel: 192.168.2.1 -> 01 02 A8 C0)
tcpPort	USHORT	Port an der zu kontaktierenden Gegenstelle.

7.4.6 Methode ITcIoTcpProtocol:IsConnected

Gibt Auskunft, ob ein Socket verbunden ist (für eingehende und ausgehende Verbindungen).

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.

7.4.7 Methode ITcIoTcpProtocol:Close

Schließt einen Socket.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu schließende Socket.

7.4.8 Methode ITcIoTcpProtocol:Listen

Öffnet einen TCP-Port für eingehende Verbindungen. Die Verwendung wird in der Interface-Übersicht beschrieben.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.
tcpPort	USHORT	Der Port auf dem nach eingehenden Verbindungen geschaut wird.

7.4.9 Methode ITcIoTcpProtocol:Accept

Akzeptiert einkommende Verbindungen. Die Verwendung wird in der Interface-Übersicht beschrieben.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.

7.4.10 Methode ITcIoTcpProtocol:SendData

Sendet Daten (Server- und Client-Funktionalität).

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.
nData	ULONG	Länge der zu versendenden Daten.
pData	PVOID	Pointer auf die zu versendenden Daten.
nSendData	ULONG&	Gibt die Anzahl der versendeten Bytes zurück. Sollte dieser kleiner als nData sein, ist ein erneuter Versand der Daten vorzunehmen.

7.4.11 Methode ITcIoTcpProtocol:CheckReceived

Muss zyklisch aufgerufen werden; ReceiveEvent und ReceiveData werden im Context dieser Methode als Callback genutzt (Server und Client-Funktionalität).

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

-

7.4.12 Methode ITcIoTcpProtocol:GetRemoteIpAddr

Liefert die entfernte IP-Adresse eines Kommunikationspartners.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.

7.4.13 Methode ITcIoTcpProtocol:GetFreeSendDataSize

Liefert die Anzahl an freien Bytes im TCP Sendebuffer.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
socketId	ULONG	Der zu verwendende Socket.
nData	ULONG&	Liefert die freien Bytes im Buffer.

7.5 ARP/Ping: ITcIoArpPingProtocol(Recv)

Die Interfaces ITcIoArpPingProtocol und ITcIoArpPingProtocolRecv ermöglichen ARP- und Ping-Nachrichten aus der Echtzeitumgebung heraus abzusetzen.

Ein Projekt, welches dieses Interface verwendet, enthält einen Pointer auf ein ITcIoArpPingProtocol Objekt und implementiert ITcIoArpPingProtocolRecv selbst. ITcIoArpPingProtocolRecv dient als Callbackinterface, um Daten innerhalb der Applikation vom TCP/UDP RT Modul empfangen zu können.

ITcIoArpPingProtocolRecv Methoden:

Name	Beschreibung
ArpReply [▶ 68]	Callbackfunktion die bei Empfang einer ArpReply Nachricht aufgerufen wird.
PingReply [▶ 67]	Callbackfunktion die bei Empfang einer PingReply Nachricht aufgerufen wird.

Wenn diese Methoden S_OK als Rückgabewert liefern, wird das Paket als verarbeitet betrachtet und nicht weiter an das Betriebssystem geleitet. Ggf. sollte S_FALSE rückgegeben werden.

ITcIoArpPingProtocol Methoden:

Name	Beschreibung
ArpRequest [▶ 69]	Sendet einen ArpRequest
PingRequest [▶ 68]	Sendet einen PingRequest
RegisterReceiver [▶ 69]	Registrieren am TCP/UDP RT Modul für den Empfang von Daten.
UnregisterReceiver [▶ 70]	De-Registrieren am TCP/UDP RT Modul für den Empfang von Daten.
CheckReceived [▶ 70]	Muss zyklisch aufgerufen werden; ArpReply und PingReply werden im Context dieser Methode als Callback genutzt

Zum Empfang von ARP- oder Ping-Daten ist eine Registrierung durch den Aufruf von RegisterReceiver erforderlich. Dieses kann in SetObjStateSO() erfolgen.

Daten werden durch einen Callback der Methode ArpReceive bzw. PingReceive von ITcIoArpPingProtocolRecv bereitgestellt.

Während des Shutdowns sollten sich alle Module durch UnregisterReceiver abmelden. Dieses kann in SetObjStateOS() erfolgen.

7.5.1 Methode ITcIoArpPingProtocolRecv:PingReply

Callbackfunktion die bei Empfang einer PingReply Nachricht aufgerufen wird.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg und muss vom implementierenden Modul entsprechend geliefert werden. Sollte dies nicht S_OK sein, wird die Antwort weiter an das Betriebssystem gereicht.

Parameter

Name	Typ	Beschreibung
ipAddr	ULONG	Die IP Adresse der Suche.
nData	ULONG	Anzahl der empfangenen Bytes.
pData	PVOID	Pointer auf die empfangenen Daten.
pVlan	ETYPE_VLAN_HEADER*	Struktur ETYPE_VLAN_HEADER, siehe unten.

Der VLAN Header repräsentiert Informationen über das VLAN.

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

7.5.2 Methode ITcIoArpPingProtocolRecv:ArpReply

Callbackfunktion die bei Empfang einer ArpReply Nachricht aufgerufen wird.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg und muss vom implementierenden Modul entsprechend geliefert werden. Sollte dies nicht S_OK sein, wird die Antwort weiter an das Betriebssystem gereicht.

Parameter

Name	Typ	Beschreibung
ipAddr	ULONG	Die IP Adresse der Suche.
macAddr	ETHERNET_ADDRESS	Ermittelte MAC-Adresse.
pVlan	ETYPE_VLAN_HEADER*	Struktur ETYPE_VLAN_HEADER, siehe unten.

Der VLAN Header repräsentiert Informationen über das VLAN.

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

7.5.3 Methode ITcIoArpPingProtocol:PingRequest

Sendet einen Ping Request.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
ipAddr	ULONG	Die IP-Adresse des Ziels.
nData	ULONG	Anzahl der empfangenen Bytes.
pData	PVOID	Pointer auf die empfangenen Daten.
pVlan	ETYPE_VLAN_HEADER*	Struktur ETYPE_VLAN_HEADER, siehe unten.

Der VLAN Header repräsentiert Informationen über das VLAN.

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

7.5.4 Methode ITcIoArpPingProtocol:ArpRequest

Sendet einen ARP Request.

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
ipAddr	ULONG	Die IP Adresse des Ziels.
macAddr	ETHERNET_ADDRESS*	Einschränkung der MAC Adresse.
pVlan	ETYPE_VLAN_HEADER*	Struktur ETYPE_VLAN_HEADER, siehe unten.

Der VLAN Header repräsentiert Informationen über das VLAN.

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

7.5.5 Methode ITcIoArpPingProtocol:RegisterReceiver

Registrieren am TCP/UDP RT Modul für den Empfang von Antworten (ARP / Ping).

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
ipRecv	ITcIoArpPingRecv*	Pointer zu dem Empfänger (Recv) Interface.

7.5.6 Methode ITcIoArpPingProtocol:UnregisterReceiver

De-Registrieren am TCP/UDP RT Modul für den Empfang von Antworten (ARP / Ping).

Syntax

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

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

Parameter

Name	Typ	Beschreibung
ipRecv	ITcIoArpPingRecv	Referenz auf den zu de-registrierenden Empfänger

7.5.7 Methode ITcIoArpPingProtocol:CheckReceived

Muss zyklisch aufgerufen werden; ArpReply und PingReply werden im Context dieser Methode als Callback genutzt.

Syntax

```
HRESULT TCOMAPI CheckReceived()
```

Rückgabewert

Typ	Beschreibung
HRESULT	Bezeichnet den Erfolg, vgl. Rückgabewerte [▶ 70].

7.6 Rückgabewerte

Die Funktionen der Interfaces haben als Rückgabewerte HRESULT. Die rückgegebenen Werte leiten sich aus den ADS Return Codes [▶ 73] ab. Ihre Bedeutung für das TF6311:

Wert (Enum)	Wert (Numerisch)	Beschreibung
ADS_E_INVALIDPARM	0x9811070B	Socket nicht allokiert/bekannt, übergebene Pointer NULL
ADS_E_NOMOREHDLS	0x98110716	Keine freien Sockets verfügbar. Default: 32 siehe TCP/UDP RT TcCom Parameter [▶ 57]
ADS_E_INCOMPATIBLE	0x9811070E	Socket im falschen Zustand. Z.B. Versuch eines Connect(), wenn vorher Socket mit Listen() genutzt; Close() ohne vorherige Verbindung; Send() ohne Verbindung; Socket Listen(), wenn bereits ein Listen() aufgerufen wurde.
ADS_E_INVALIDSTATE	0x98110712	TCP/UDP RT Objekt ist nicht in OP Mode
ADS_E_INVALIDDATA	0x98110706	Problem mit Parameter. Z.B. pData==NULL bei SendData
ADS_E_EXISTS	0x9811070F	Port schon anderweitig verwendet
ADS_E_PENDING	0x9811071E	Es wurden nicht alle Daten versendet (SendData)
S_OK	0x0	Aufruf erfolgreich. IsConnected(): Verbindung besteht
S_FAIL	0x1	Aufruf nicht erfolgreich, Allgemeiner Fehler IsConnected(): Verbindung besteht nicht

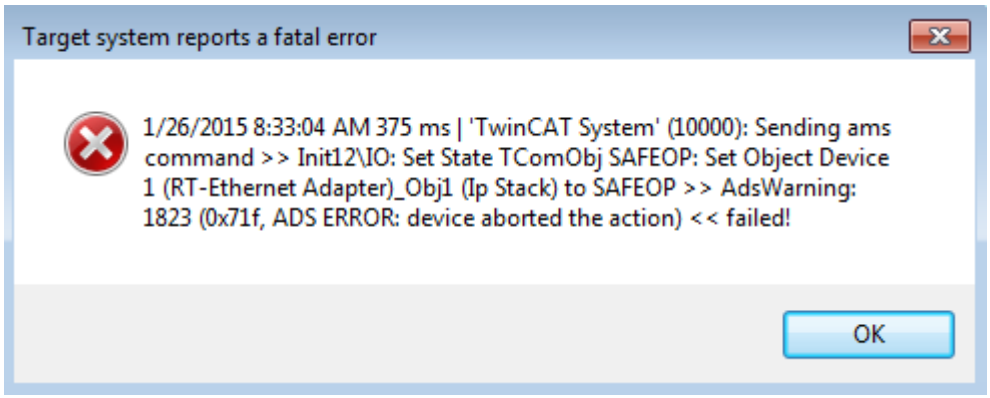
Die Werte aus dem Bereich 0x9811 sind in der Enumeration „E_HRESULTAdsErr“ (PLC) sowie entsprechende defines ADS_E_* (C++) definiert.

8 Fehleranalyse

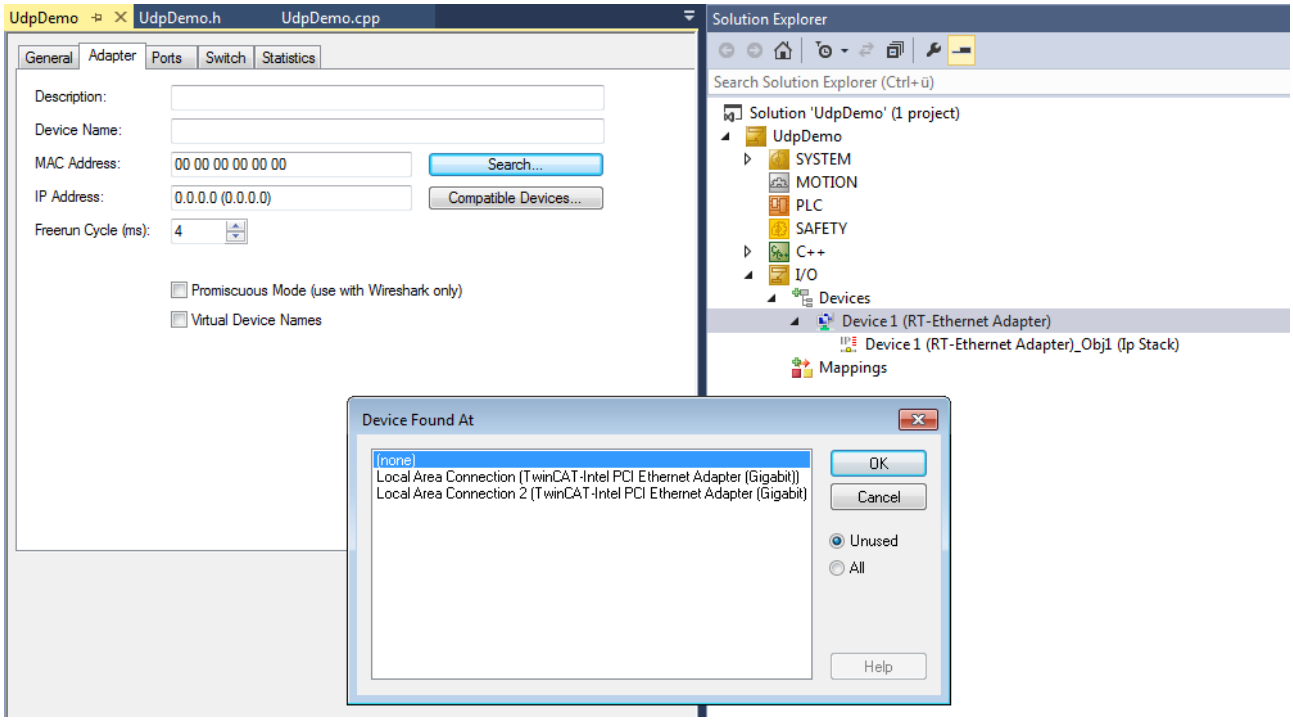
An dieser Stelle werden häufige Probleme oder Situationen beim Umgang mit dem Produkt zusammen mit einer Fehlerbeschreibung aufgelistet.

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

Wenn beim Starten eines IP Stack TcCOM Objektes der ADS Fehler 1823 (0x71f) auftritt, ist vermutlich die Konfiguration der Netzwerkkarte nicht korrekt.



Kontrollieren Sie die Einstellungen unter „Adapter“ der Netzwerkkarte in der Projektmappe:



Ausführlicher ist die Konfiguration der Netzwerkkarte für das TCP/UDP RT Modul hier [▶ 35] dokumentiert.

9 Anhang

9.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 73]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 73]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 74]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 76]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.

Hex	Dec	HRESULT	Name	Beschreibung
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

Mehr Informationen:
www.beckhoff.de/tf6311

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