BECKHOFF New Automation Technology

Handbuch | DE

TS6310

TwinCAT 2 | TCP/IP Connection Server

Supplement | Communication

10.08.2023 | Version: 1.3

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, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche 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 weiter entwickelt.
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

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

Der TwinCAT TCP/IP Connection Server ermöglicht die Implementierung/Realisierung eines oder mehrerer TCP/IP-Server/-Clients in der TwinCAT SPS.

Produktkomponenten

- TcpIp.Lib (implementiert TCP/IP- und UDP-Basisfunktionen);
- TcSocketHelper.Lib (implementiert TCP/IP-Hilfsfunktionen);
- TcSNMP.Lib (implementiert SNMP V1-Hilfsfunktionen ab v1.0.59);
- TwinCAT TCP/IP Connection Server (TwinCAT Server);

3 Einführung

TCP

Bei dem TCP-Protokoll handelt es sich um ein verbindungsorientiertes Protokoll, analog wie bei einer Telefonverbindung, wo die Gesprächsteilnehmer die Verbindung zuerst aufbauen müssen. Das TCP-Protokoll wird dort eingesetzt, wo eine Bestätigung für die vom Client oder Server gesendeten Daten benötigt wird. Die Integrität der Daten wird vom Protokoll verwaltet, was mehr Ressourcen erfordert. Das TCP-Protokoll ist gut geeignet um größere Datenmengen zu verschicken.

TCP ist ein streamorientiertes Transportprotokoll, d.h. es transportiert einen Datenstrom ohne definierten Anfang und Ende. Es werden dabei weder Informationen zur Länge, noch Informationen über Anfang und Ende einer Nachricht übertragen. Für den Sender ist dies unproblematisch da er weiß, wie viele Datenbytes er verschickt. Der Empfänger kann jedoch nicht erkennen, wo eine Nachricht im Datenstrom aufhört und wo die nächste im Datenstrom beginnt. Ein Leseaufruf auf der Empfängerseite liefert immer nur die gerade im Empfangspuffer vorhandenen Daten (u.U. können es weniger oder mehr sein als der Datenblock, der vom anderen Teilnehmer gesendet wurde).

Der Sender muss eine Nachrichtenstruktur festlegen, die beim Empfänger bekannt ist und interpretiert werden kann. Die Nachrichtenstruktur kann sich im einfachen Fall aus den Daten und einem abschließenden Steuerzeichen (z.B. carriage return) zusammensetzen. Das abschließende Steuerzeichen signalisiert das Ende einer Nachricht.

Die Nachrichtenstruktur für die Übertragung von Binärdaten mit einer variablen Länge wird oft auf folgende Weise festgelegt: In den ersten Datenbytes wird ein spezielles Steuerzeichen (start delimiter) und die Datenlänge der darauffolgenden Daten eingetragen. Der Empfänger kann dadurch den Nachrichtenanfang und das Ende erkennen.

Für eine minimale TCP/IP-Clientimplementierung in der SPS werden folgende Funktionsbausteine benötigt:

- Fürs Aufbauen und Abbauen der Verbindung zum Remote-Server eine Instanz des FB_SocketConnect [▶ 18] und FB_SocketClose [▶ 19] Funktionsbausteins (Tipp: FB_ClientServerConnection [▶ 40] kapselt die Funktionalität beider Funktionsbausteine in einem Baustein);
- Für den Datenaustausch mit dem Remote-Server eine Instanz des FB_SocketSend [▶ 24] und/oder FB_SocketReceive [▶ 25] Funktionsbausteins;

Für eine minimale TCP/IP-Serverimplementierung in der SPS werden folgende Funktionsbausteine benötigt:

- Fürs Öffnen des Listener-Sockets eine Instanz des FB_SocketListen [▶ 22] Funktionsbausteins. Fürs Aufbauen und Abbauen der Verbindung/-en zu den Remote-Clients eine Instanz des FB_SocketAccept [▶ 23] und FB_SocketClose [▶ 19] Funktionsbausteins (Tipp: FB_ServerClientConnection [▶ 37] kapselt die Funktionalität aller drei Funktionsbausteine in einem Baustein).
- Für den Datenaustausch mit den Remote-Clients eine Instanz des FB_SocketSend [▶ 24] und/oder FB_SocketReceive [▶ 25] Funktionsbausteins;

In jedem SPS-Laufzeitsystem, in dem Sie ein Socket öffnen, benötigen Sie eine Instanz des FB_SocketCloseAll [▶ 20] Funktionsbausteins;

Die Instanzen der FB_SocketAccept [▶ 23] und FB_SocketReceive [▶ 25] Funktionsbausteine werden zyklisch (pollend) aufgerufen, alle anderen nach Bedarf.

UDP

Bei dem UDP-Protokoll handelt es sich um ein verbindungsloses Protokoll. Die Daten werden von einem Teilnehmer zum anderen gesendet, ohne das eine explizite Verbindung existiert. Das UDP-Protokoll ist gut geeignet um kleine Datenmengen zu verschicken. Eine UDP-Applikation kann beides sein: ein Client oder

ein Server. Das UDP-Protokoll garantiert nicht, dass die gesendeten Daten tatsächlich am Ziel ankommen (es wird keine Bestätigung für die empfangenen Pakete gesendet). Die einzelnen Datenpakete können auch am Ziel in einer anderen Reihenfolge ankommen oder verlorengehen.

UDP ist ein paketorientiertes/nachrichtenorientiertes Transportprotokoll, d.h. der gesendete Datenblock wird auf der Empfängerseite auch als kompletter Datenblock empfangen.

Für eine minimale UDP-Server/Client-Implementierung werden folgende Funktionsbausteine benötigt:

- Fürs Öffnen und Schließen eines UDP-Sockets eine Instanz des FB_SocketUdpCreate [▶ 26] und eine Instanz des FB_SocketClose [▶ 19] Funktionsbausteins;
- Für den Datenaustausch mit anderen Teilnehmern eine Instanz des FB_SocketUdpSendTo [▶ 28] und/ oder FB_SocketUdpReceiveFrom [▶ 29] Funktionsbausteins;
- In jedem SPS-Laufzeitsystem, in dem Sie ein UDP-Socket öffnen, eine Instanz des FB_SocketCloseAll [▶ 20] Funktionsbausteins;

Die Instanzen des FB_SocketUdpReceiveFrom [▶ 29] Funktionsbausteins werden zyklisch (pollend) aufgerufen, alle anderen nach Bedarf.

Weitere Informationen finden Sie auf den folgenden Dokumentationsseiten.

Glossar

Begriff	Beschreibung
TwinCAT TCP/IP Connection Server	Ein TwinCAT Server. Der Server ermöglicht das Öffnen, Schließen, Versenden und Empfangen der Daten über die Windows Sockets.
Remote-Client	Ein Client auf einem entfernten Rechner (aus der Serversicht: mit dem der Server kommunizieren will).
Remote-Server	Ein Server auf einem entfernten Rechner (aus der Clientsicht: mit dem der Client kommunizieren will).
Local-Client	Ein Client auf dem lokalen Rechner.
Local-Server	Ein Server auf dem lokalen Rechner.
Verbindungshandle (Socket-Handle)	Eine SPS-Variable vom Typ T_HSOCKET [▶ 33]

4 Systemvoraussetzungen

Die folgenden Systemvoraussetzungen müssen für eine ordnungsgemäße Funktion des Supplements TwinCAT TCP/IP Server erfüllt sein.

Windows XP-Plattform

Derzeit werden alle Beckhoff Embedded-PC/IPC mit den folgenden Betriebssystemen unterstützt: Windows XP, Windows XP Embedded, Windows Embedded Standard 2009, Windows Vista, Windows 7. Eventuell auftretende Unterschiede zwischen diesen Betriebssystemvarianten werden in der Installationsanleitung [▶ 12] beschrieben. Als weitere Voraussetzung muss auf dem TCP/IP Server ein TwinCAT2 (mindestens PLC) installiert sein, welches während des Betriebs entweder im Config- oder Run-Mode laufen kann.

Windows CE-Plattform

Derzeit werden sowohl alle Beckhoff Embedded-PC/IPC mit den folgenden Betriebssystemen unterstützt: Windows CE5, Windows CE6, Windows CE7.

5 Installation

Dieser Teil der Dokumentation führt den Benutzer Schritt-für-Schritt durch den Installationsvorgang des TwinCAT TCP/IP Server Supplements für Windows XP basierte Betriebssysteme. Es wird hierbei auf die folgenden Themen eingegangen:

- Herunterladen der Setup-Datei
- Starten der Installation

Herunterladen der Setup-Datei

Wie viele andere TwinCAT Supplement-Produkte auch, steht TwinCAT TCP/IP Server als Download auf dem Beckhoff FTP-Server zur Verfügung. Es handelt sich hierbei um die jeweils aktuelle Version des Produkts, welche sowohl in einer 30-Tage Demoversion als auch als Vollversion lizensierbar ist. Bitte führen Sie die folgenden Schritte durch, um die Setup-Datei zu downloaden:

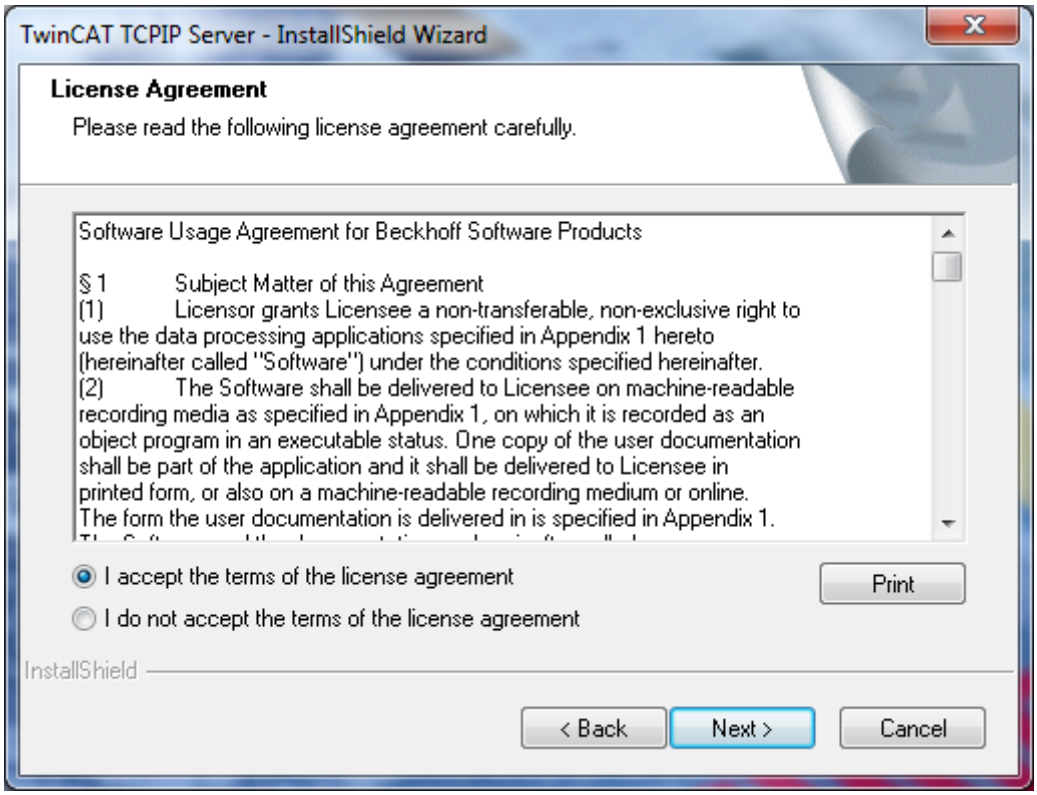
- Wählen Sie von der BECKHOFF Webseite den TS6310 | TwinCAT TCP/IP Server aus.
- (Optional) Übertragen Sie die Datei auf das TwinCAT-Laufzeitsystem, auf welchem Sie das Supplement installieren möchten

Starten der Installation

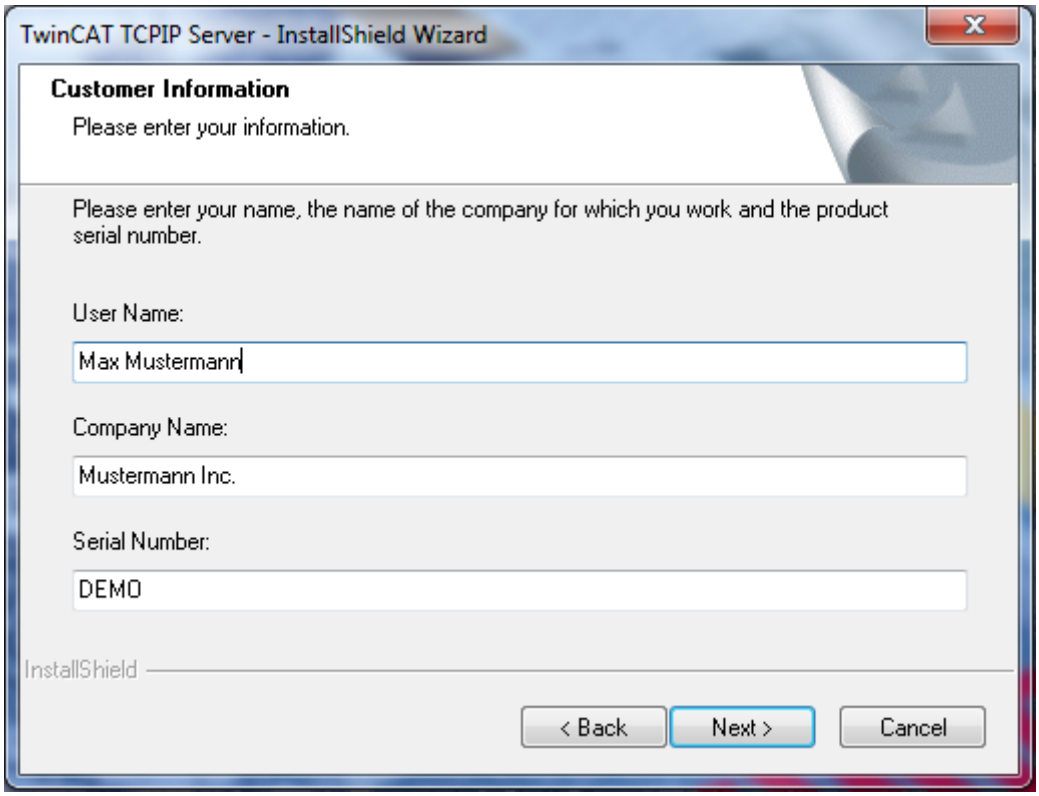
Um das Supplement zu installieren, führen Sie bitte die folgenden Schritte durch:

> Bitte starten Sie die Installation unter Windows 7 32-bit/64-bit per "Als Administrator ausführen", indem Sie die Setup-Datei mit der rechten Maustaste anklicken und die entsprechende Option im Kontextmenü auswählen.

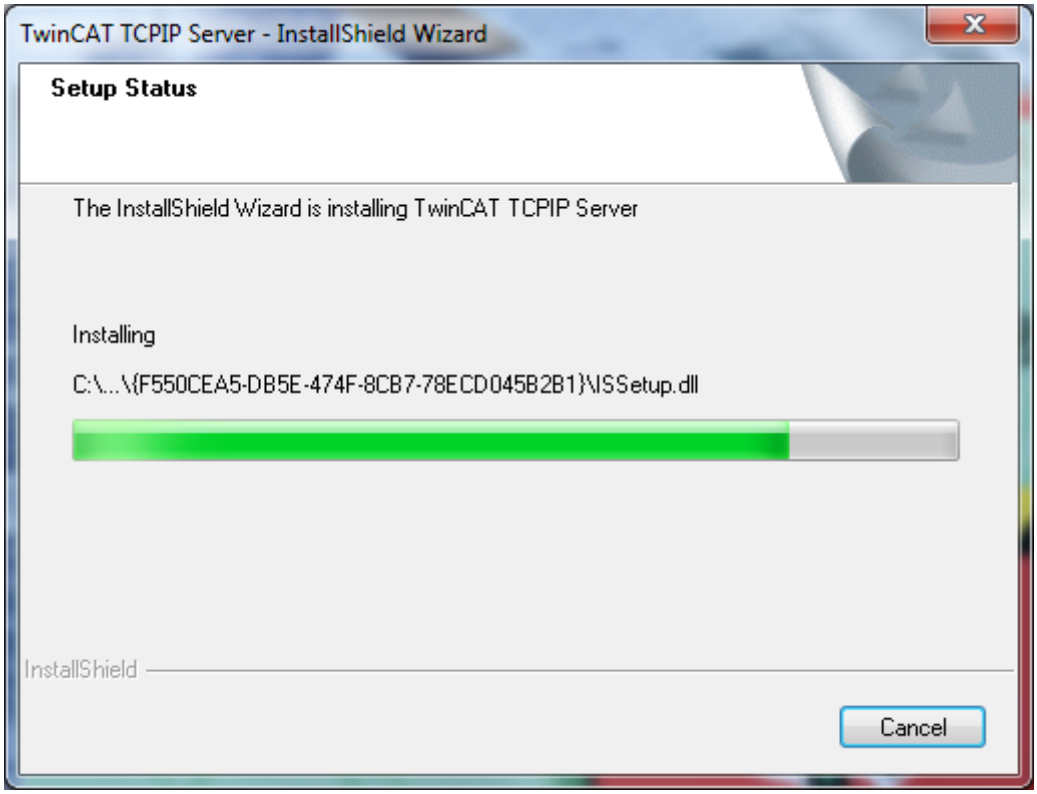
- Führen Sie einen Doppelklick auf die heruntergeladene Datei **"TcpIpServer.exe"** aus.
- Wählen Sie eine **Sprache** aus, in der Sie die Software installieren möchten.
- Klicken Sie auf "Next" und akzeptieren Sie dann die **Endbenutzervereinbarung.**



- Geben Sie Ihre **Benutzerdaten** ein. Alle sichtbaren Felder sind hierbei Pflichtfelder. Möchten Sie eine 30-Tage Demoversion installieren, so geben Sie als Lizenzschlüssel bitte "DEMO" ein.



- Klicken Sie auf **"Install"** um die Installation zu starten.



- Zum Abschluss der Installation **starten Sie bitte Ihren Computer neu.**

6 Installation Windows CE

Dieser Teil der Dokumentation beschreibt, wie das Supplement-Produkt TwinCAT TCP/IP Server auf einem BECKHOFF Embedded PC Controller basierend auf Windows CE installiert wird, z.B. CX1000, CX1020, CX9000, CX9001, CX9010, CP62xx, C69xx, ...

Der Installationsvorgang unter Windows CE besteht aus insgesamt vier Schritten:

- Herunterladen der Setup-Datei
- Installation auf einem Host-PC
- Übertragen der Setup-Datei auf das Windows CE Gerät
- Ausführen der Installation auf dem Windows CE Gerät

Herunterladen der Setup-Datei

Wie viele andere TwinCAT Supplement-Produkte auch, steht TwinCAT TCP/IP Server CE als Download auf dem Beckhoff FTP-Server zur Verfügung. Es handelt sich hierbei um die jeweils aktuelle Version des Produkts. Bitte führen Sie den folgenden Schritt durch, um die Setup-Datei zu downloaden:

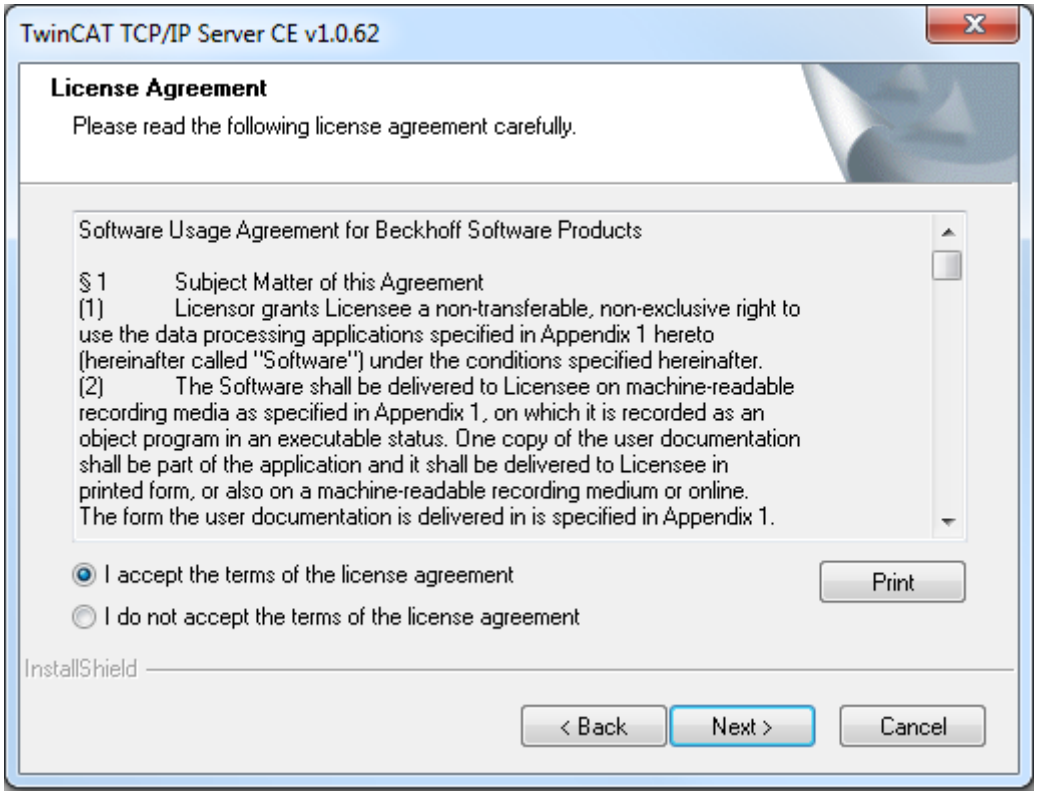
- Wählen Sie von der BECKHOFF Webseite den TS6310-0030 | TwinCAT TCP/IP Server CE aus.

Installation auf dem Host-PC

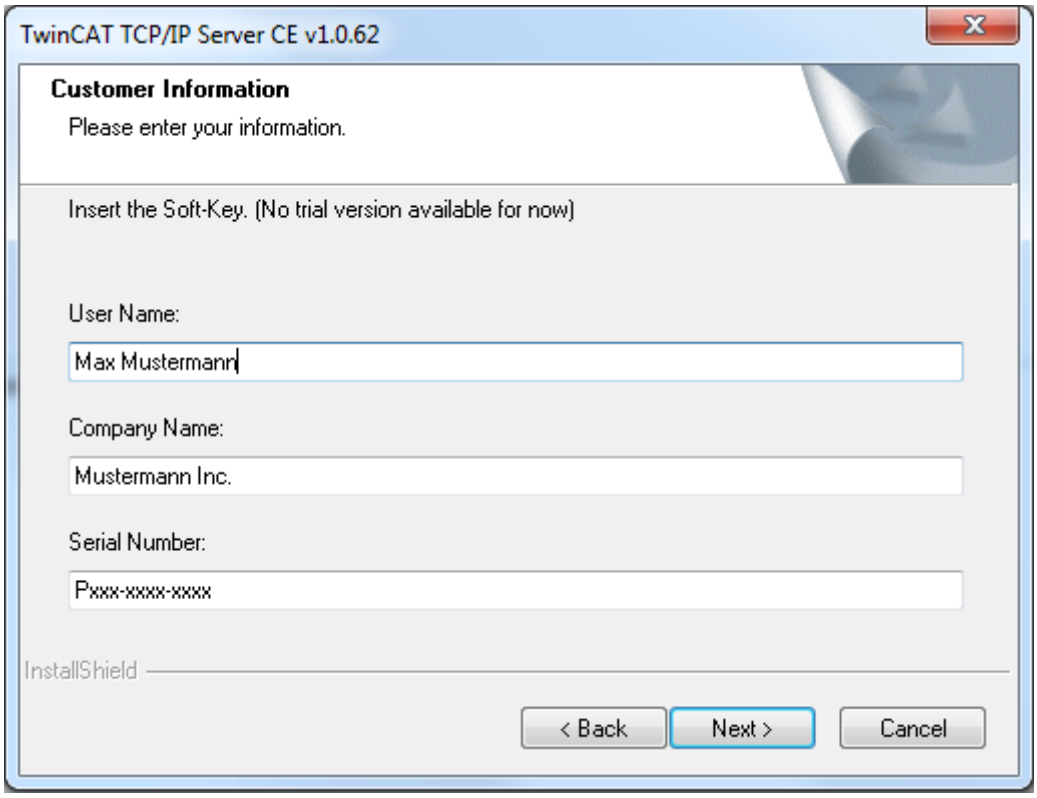
Um an die Installationsdateien für Windows CE zu gelangen, muss die heruntergeladene Setup-Datei zunächst auf einem Host-PC installiert werden. Dies kann ein beliebiges Windows XP basiertes System sein. Führen Sie die folgenden Schritte durch:

Bitte beachten Sie: Eine 30-Tage Demoversion des TCP/IP Server für Windows CE ist momentan nicht verfügbar. Sie benötigen also einen gültigen Produktschlüssel für die Installation.

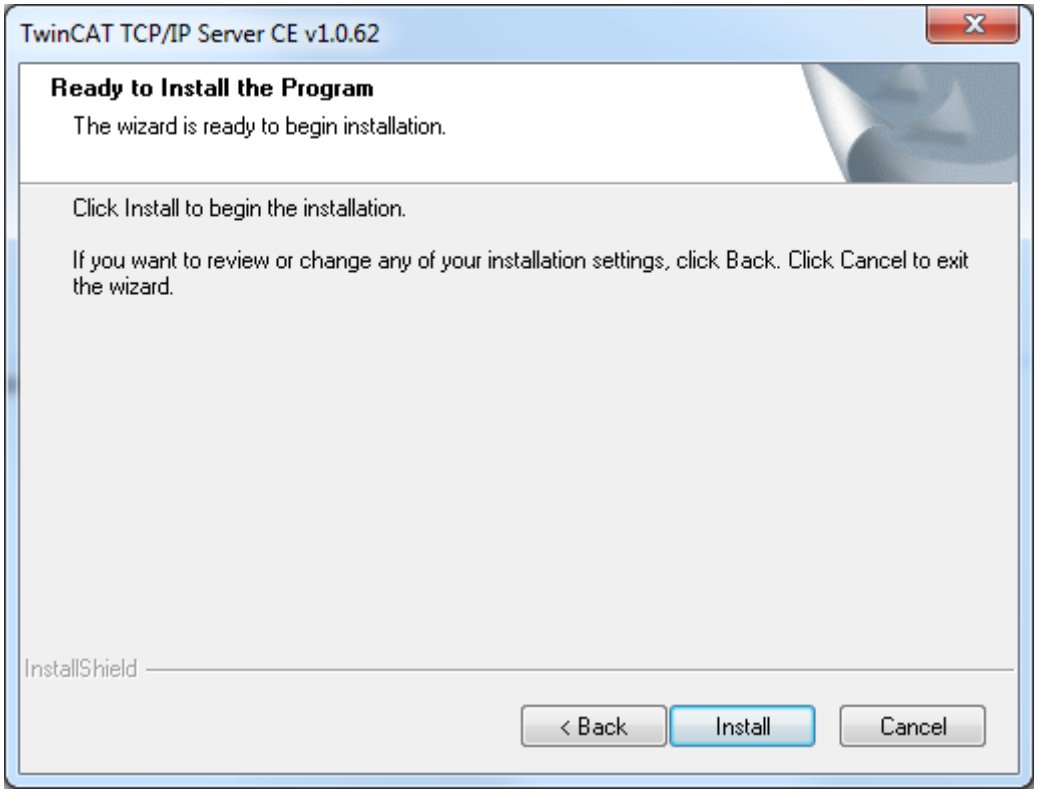
- Führen Sie einen Doppelklick auf die heruntergeladene Datei **"TcTCPIPSvrCE.exe"** aus.
- Wählen Sie eine **Sprache** aus, in der Sie die Software installieren möchten.
- Klicken Sie auf "Next" und akzeptieren Sie dann die **Endbenutzervereinbarung.**



- Geben Sie Ihre **Benutzerdaten** ein. Alle sichtbaren Felder sind hierbei Pflichtfelder.



- Klicken Sie anschließend auf **"Install"** um die Installation zu starten.



Nach der Installation liegen die Windows CE Installationsdateien nun im Ordner **".\TwinCAT\CE"**. Dieser Ordner enthält eine Vielzahl an verschiedenen CE Installationsdateien in Form von CAB-Dateien:

- **TCPIP\Install\TcTCPIPSvrCe.I586.cab**: TCP/IP Server für x86 basierte CPUs (wie z.B. CX10xx, CP62xx, C69xx, ...)
- **TCPIP\Install\TcTCPIPSvrCe.ARMV4I.cab**: TCP/IP Server für ARM basierte CPUs (wie z.B. CX9001, CX9010, CP6608, ...)

Übertragen der Setup-Datei auf das Windows CE Gerät

Übertragen Sie nun die entsprechende Installationsdatei auf Ihren Controller. Dies kann über die folgenden Mechanismen geschehen:

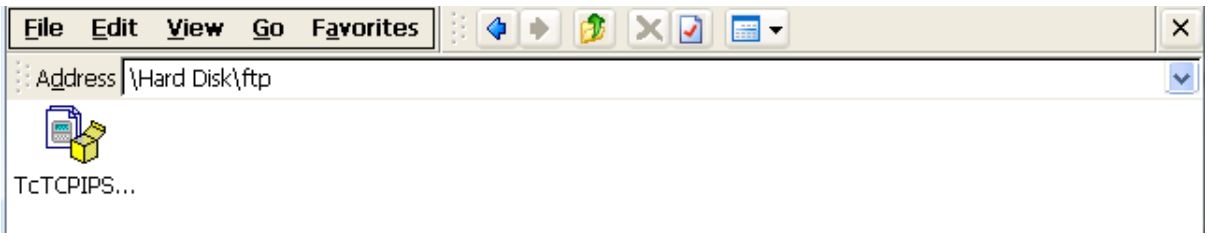
- über ein Shared Folder
- über den in Windows CE integrierten FTP-Server
- über ActiveSync
- über einen CF Adapter

Sie finden entsprechende Anleitungen in der "Windows CE" Rubrik hier im Infosys.

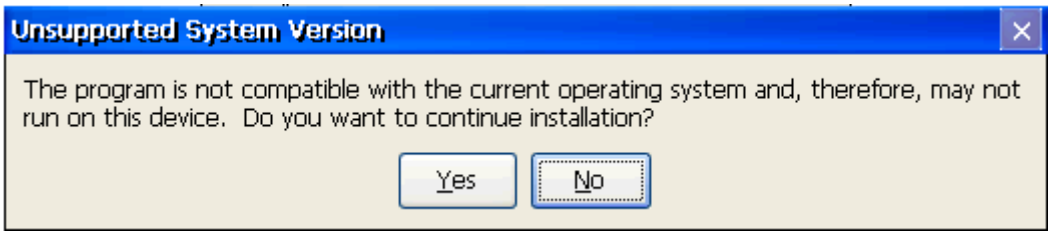
Ausführen der Installation auf dem Windows CE Gerät

Die auf den Controller übertragene Installationsdatei "**TcTCPIPSvrCe.xxxx.CAB**" muss nun installiert werden. Hierzu führen Sie bitte die folgenden Schritte auf dem CE-Gerät durch:

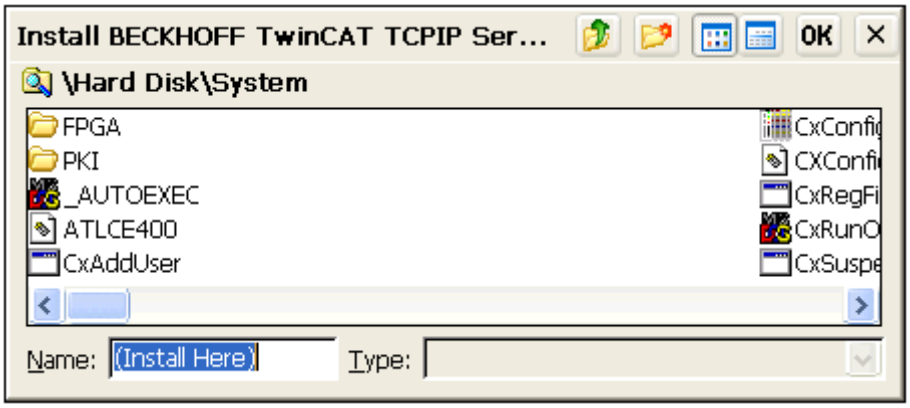
- Navigieren Sie zu dem Ordner, in den Sie die Installationsdatei übertragen haben,



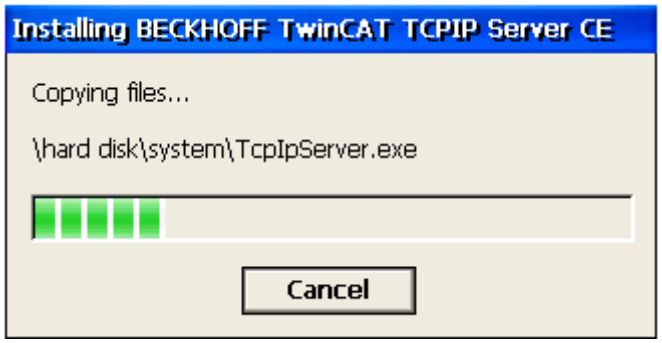
- Führen Sie einen Doppelklick auf die CAB-Datei aus. Sollten Sie eine MessageBox mit dem Hinweis "Program is not compatible with current operating system" erhalten, so überprüfen Sie, ob Sie die korrekte CAB-Datei (ARM, I586) für Ihre Plattform verwendet haben.
- Sind Sie sich sicher, dass die CAB-Datei korrekt ist, bestätigen Sie die Meldung mit **"Yes"**.



- Bestätigen Sie das Zielverzeichnis**"\Hard Disk\System"** mit **"Ok".**



- Zum Starten der Installation klicken Sie in der rechten oberen Ecke auf **"Ok"**



Nach der Installation löscht sich die Installationsdatei automatisch.

Der TCP/IP Server erst nach dem nächsten Neustart des Systems verfügbar.

7 SPS-Bibliotheken

7.1 Tcplp.lib

Mit den Funktionsbausteinen der **Tcplp.Lib** können in der TwinCAT SPS Client- oder Server-Applikationen realisiert werden. Diese können entweder über **User Datagram Protocol (UDP)** oder über **Transmission Control Protocol (TCP)** Daten mit anderen Kommunikationsteilnehmern austauschen**.**

Systemvoraussetzungen:

Programmierumgebung:

- NT4, W2K, XP, XPe;
- TwinCAT System Version 2.8 oder höher;
- TwinCAT Installation Level: TwinCAT PLC oder höher;

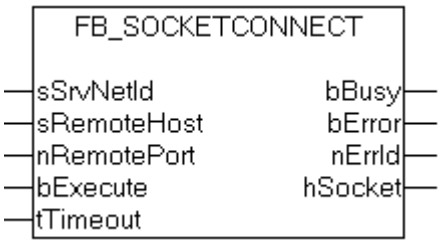
Zielplattform:

- TwinCAT SPS-Laufzeitsystem Version 2.8 oder höher.
- PC or CX (x86)
  - TwinCAT TCP/IP Connection Server **v1.0.0.0** oder höher;
  - NT4, W2K, XP, XPe, CE (image v1.75 oder höher);
- CX (ARM)
  - TwinCAT TCP/IP Connection Server **v1.0.0.44** oder höher;
  - CE (image v2.13 oder höher);

Installation:

Die SPS-Bibliothek wird mit dem TwinCAT TCP/IP Connection Server mitgeliefert und während der Installation in den ...\TwinCAT\PLC\Lib-Ordner kopiert.

7.1.1 FB_SocketConnect



Mit dem Funktionsbaustein FB_SocketConnect kann ein Local-Client über den TwinCAT TCP/IP Connection Server eine neue TCP/IP Verbindung zu einem Remote-Server aufbauen. Bei erfolgreicher Verbindung wird ein neuer Socket geöffnet und am *hSocket*-Ausgang das dazugehörige Verbindungshandle zurückgeliefert. Das Verbindungshandle wird dann z.B. von den Funktionsbausteinen FB_SocketSend [▶ 24] und FB_SocketReceive [▶ 25] benötigt, um mit einem Remote-Server Daten austauschen zu können. Eine nicht mehr benötigte Verbindung wird mit dem Funktionsbaustein FB_SocketClose [▶ 19] geschlossen. Es können mehrere Clients gleichzeitig eine Verbindung zum Remote-Server aufbauen. Für jeden neuen Client wird ein neuer Socket geöffnet und ein neues Verbindungshandle zurückgeliefert. Jedem Client wird von dem TwinCAT TCP/IP Connection Server automatisch eine neue IP-Portnummer zugewiesen.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    sRemoteHost     : STRING(15);
    nRemotePort     : UDINT;
```

```
    bExecute        : BOOL;
    tTimeout        : TIME := T#45s;(*!!!*)
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

sRemoteHost: Die IP-Adresse (Ipv4) des Remote-Servers als String (z.B.: '172.33.5.1'). Für einen Server auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

nRemotePort: Die IP-Portnummer des Remote-Servers (z.B.: 200).

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Setzen Sie den Wert *tTimeout* nicht zu niedrig, da bei einer Netzwerkunterbrechung Timeoutzeiten von > 30s auftreten können. Bei einem zu niedrigen Wert wird die Kommandoausführung vorzeitig unterbrochen und der ADS-Fehlercode: 1861 (timeout elapsed) statt des Winsocket-Fehlers: WSAETIMEDOUT zurückgeliefert.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hSocket     : T_HSOCKET;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

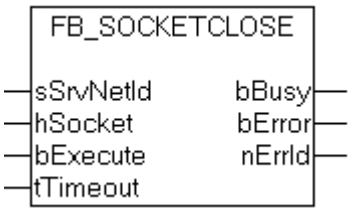
nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

hSocket: Das TCP/IP Verbindungshandle [▶ 33] zu dem neu geöffneten Local-Client Socket.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.2 FB_SocketClose



Mit dem Funktionsbaustein FB_SocketClose kann ein geöffneter TCP/IP- oder UDP-Socket geschlossen werden.

TCP/IP: Der Listener-Socket wird mit dem Funktionsbaustein FB_SocketListen [▶ 22], ein Local-Client-Socket mit FB_SocketConnect [▶ 18] und ein Remote-Client-Socket mit FB_SocketAccept [▶ 23] geöffnet.

UDP: Der UDP-Socket wird mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 26] geöffnet.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    hSocket         : T_HSOCKET;
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: TCP/IP: Das Verbindungshandle [▶ 33] des zu schließenden Listener-, Remote- oder Local-Client-Sockets. UDP: Das Verbindungshadle des zu schließenden UDP-Sockets.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

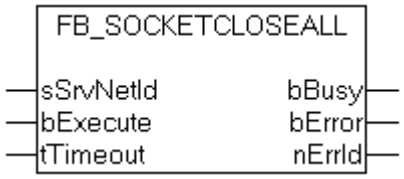
bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.3 FB_SocketCloseAll



Beim TwinCAT Restart oder TwinCAT Stopp wird auch der TwinCAT TCP/IP Connection Server gestoppt. Alle bereits geöffneten Sockets (TCP/IP und UDP Verbindungshandles) werden dabei automatisch geschlossen. Nach einem "PLC reset" oder "Rebuild all..." oder einem neuen "Download" wird das SPS-Programm zurückgesetzt und die Informationen über die bereits geöffneten Sockets (Verbindungshandles) sind in der SPS nicht mehr vorhanden. Die geöffneten Verbindungen können dann nicht mehr ordnungsgemäß geschlossen werden.

Mit dem Funktionsbaustein FB_SocketCloseAll können alle Verbindungshandles (TCP/IP und UDP Sockets) geschlossen werden, die von einem SPS-Laufzeitsystem geöffnet wurden. D.h. wenn Sie FB_SocketCloseAll in einer der Tasks des ersten Laufzeitsystems (Port 801) aufrufen, werden alle Sockets geschlossen, die in dem ersten Laufzeitsystem geöffnet wurden. In jedem SPS-Laufzeitsystem, in dem die Socket-Funktionsbausteine benutzt werden, sollte eine Instanz von FB_SocketCloseAll beim SPS-Start aufgerufen werden (siehe weiter unten im Text).

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    nErrId      :UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

Beispiel für eine Implementierung in ST:

Durch den folgenden Programmcode werden die vor einem "SPS Reset" oder "Download" geöffneten Verbindungshandles (Sockets) beim erneuten SPS Start ordnungsgemäß geschlossen.

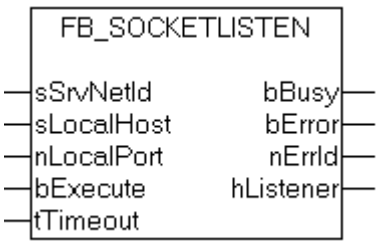
```
PROGRAM MAIN
VAR
    fbSocketCloseAll  : FB_SocketCloseAll;
    bCloseAll         : BOOL := TRUE;
END_VAR
```

```
IF bCloseAll THEN(*On PLC reset or program download close all old connections *)
    bCloseAll := FALSE;
fbSocketCloseAll( sSrvNetId:= '', bExecute:= TRUE, tTimeout:= T#10s );
ELSE
    fbSocketCloseAll( bExecute:= FALSE );
END_IFIFNOT fbSocketCloseAll.bBusy THEN(*...
... continue program execution...
...*)
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.4 FB_SocketListen



Mit dem Funktionsbaustein FB_SocketListen kann über den TwinCAT TCP/IP Connection Server ein neuer Listener-Socket geöffnet werden. Über einen Listener-Socket kann der TwinCAT TCP/IP Connection Server auf ankommende Verbindungsanforderungen von Remote-Clients achten. Bei Erfolg wird am *hListner*-Ausgang das dazugehörige Verbindungshandle zurückgeliefert. Dieses Handle wird von dem Funktionsbaustein FB_SocketAccept [▶ 23] benötigt. Ein nicht mehr benötigter Listener-Socket wird mit dem Funktionsbaustein FB_SocketClose [▶ 19] geschlossen. Auf einem Rechner kann nur ein Listener-Socket mit der gleichen IP-Portnummer geöffnet werden.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    sLocalHost      : STRING(15);
    nLocalPort      : UDINT;
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

sLocalHost: Die Local-Server IP-Adresse (Ipv4) als String (z.B.: '172.13.15.2'). Für einen Server auf dem lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

nLocalPort: Der Local-Server IP-Port (z.B.: 200).

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hListener   : T_HSOCKET;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

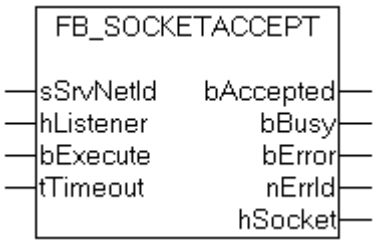
hListener: Das Verbindungshandle [▶ 33] zum neuen Listener-Socket.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
		(Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)

7.1.5 FB_SocketAccept



Die beim TwinCAT TCP/IP Connection Server ankommenden Remote-Client Verbindungsanforderungen müssen angenommen (akzeptiert) werden. Der Funktionsbaustein FB_SocketAccept nimmt die ankommenden Remote-Client Verbindungsanforderungen an, öffnet einen neuen Remote-Client-Socket und liefert das dazugehörige Verbindungshandle zurück. Das Verbindungshandle wird dann z.B. von den Funktionsbausteinen FB_SocketSend [▶ 24] und FB_SocketReceive [▶ 25] benötigt, um mit dem Remote-Client Daten austauschen zu können. Alle ankommenden Verbindungsanforderungen müssen zuerst angenommen werden. Eine nicht mehr benötigte oder unerwünschte Verbindung kann mit dem Funktionsbaustein FB_SocketClose [▶ 19] geschlossen werden.

Eine Serverimplementierung benötigt mindestens eine Instanz dieses Funktionsbausteins. Diese Instanz muss zyklisch (pollend) in einer SPS-Task aufgerufen werden. Durch eine positive Flanke am *bExecute*-Eingang (z.B. alle 5 Sekunden) kann der Baustein zyklisch aktiviert werden.

Beim Erfolg wird der *bAccepted*-Ausgang gesetzt und das Verbindungshandle zum neuen Remote-Client am *hSocket*-Ausgang zurückgeliefert. Es wird kein Fehler zurückgeliefert, wenn keine neuen Remote-Client Verbindungsanforderungen vorliegen. Es können mehrere Remote-Clients gleichzeitig eine Verbindung zum Server aufbauen. Die Verbindungshandles mehrerer Remote-Clients können nacheinander, durch mehrere Aufrufe des Funktionsbausteins, abgeholt werden. Jedes Verbindungshandle zu einem Remote-Client kann nur ein Mal abgeholt werden. Es empfiehlt sich die Verbindungshandles in einer Liste (Array) zu halten. Neue Verbindungen werden der Liste hinzugefügt und die geschlossenen müssen aus der Liste entfernt werden.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    hListener       : T_HSOCKET;
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse desTwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hListener: Das Verbindungshandle [▶ 33] des Listener-Sockets. Dieses Handle muss vorher mit dem Funktionsbaustein FB_SocketListen [▶ 22] angefordert werden.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bAccepted   : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hSocket     : T_HSOCKET;
END_VAR
```

bAccepted: Dieser Ausgang wird gesetzt, wenn eine neue Verbindung zu einem Remote-Client hergestellt wurde.

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt so lange gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

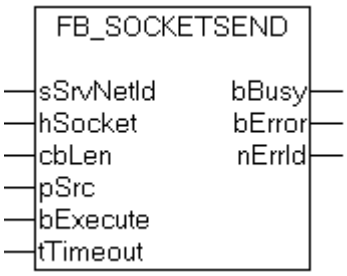
nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

hSocket: Das Verbindungshandle [▶ 33] eines neuen Remote-Clients.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.6 FB_SocketSend



Mit dem Funktionsbaustein FB_SocketSend können über den TwinCAT TCP/IP Connection Server Daten zu Remote-Clients oder Remote-Server gesendet werden. Eine Remote-Clientverbindung muss vorher mit dem Funktionsbaustein FB_SocketAccept [▶ 23] oder eine Remote-Serververbindung mit dem Funktionsbaustein FB_SocketConnect [▶ 18] aufgebaut werden.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    hSocket     : T_HSOCKET;
    cbLen       : UDINT;
    pSrc        : DWORD;
    bExecute    : BOOL;
    tTimeout    : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: Das Verbindungshandle [▶ 33] des Kommunikationspartners zu dem Daten gesendet werden sollen.

cbLen: Die Anzahl der zu sendenden Daten in Bytes.

pSrc: Die Adresse (Pointer) auf den Sendepuffer.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Wenn der Sendepuffer des Sockets voll ist, weil z.B. der Remote-Kommunikationspartner nicht schnell genug die gesendeten Daten empfängt oder sehr viele Daten gesendet werden, liefert der FB_SocketSend-Funktionsbaustein nach der *tTimeout*-Zeit einen ADS-Timeoutfehler: 1861 zurück. In diesem Fall muss der Wert der tTimeout-Eingangsvariablen entsprechend erhöht werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

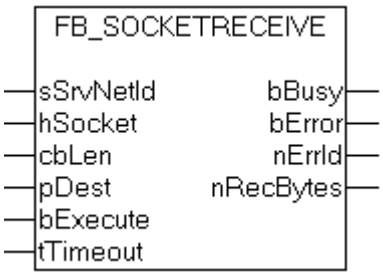
bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.7 FB_SocketReceive



Mit dem Funktionsbaustein FB_SocketReceive können über den TwinCAT TCP/IP Connection Server Daten eines Remote-Clients oder Remote-Servers empfangen werden. Eine Remote-Clientverbindung muss vorher mit dem Funktionsbaustein FB_SocketAccept [▶ 23] und eine Remote-Serververbindung mit dem Funktionsbaustein FB_SocketConnect [▶ 18] aufgebaut werden. Die Daten können in einem TCP/IP-Netzwerk fragmentiert (in mehreren Paketen) empfangen oder verschickt werden. Es ist also möglich, dass nicht alle Daten auf einmal mit einem Aufruf der Instanz von FB_SocketReceive empfangen werden können. Aus diesem Grund muss die Instanz zyklisch (pollend) in der SPS-Task aufgerufen werden, so lange bis alle benötigten Daten empfangen wurden. Dabei wird eine steigende Flanke, z.B. alle 100ms, an dem bExecute-Eingang erzeugt. Beim Erfolg werden die zuletzt empfangenen Daten in den Empfangspuffer hineinkopiert. Der *nRecBytes*-Ausgang liefert die Anzahl der zuletzt erfolgreich empfangenen Datenbytes zurück. Wenn beim letzten Aufruf keine neuen Daten gelesen werden konnten, liefert der Funktionsbaustein keinen Fehler und *nRecBytes* == Null.

Bei einem einfachen Protokoll in dem z.B. ein Nullterminierter String von einem Remote-Server empfangen werden soll, muss der Funktionsbaustein FB_SocketReceive z.B. so oft aufgerufen werden, bis in den empfangenen Daten die Nullterminierung erkannt wurde.

Wenn der Remote-Teilnehmer vom TCP/IP-Netzwerk getrennt wurde (nur auf der Remote-Seite) und der lokale Teilnehmer noch im TCP/IP-Netzwerk hängt, liefert der FB_SocketReceive-Funktionsbaustein keinen Fehler und keine Daten. Der geöffnete Socket existiert immer noch, es werden nur keine Daten empfangen. Die Applikation wird dann möglicherweise endlos auf die restlichen Frame-Datenbytes warten. Es wird empfohlen in der Applikation eine Timeout-Überwachung zu implementieren. Wenn nach einer bestimmten Zeit, z.B. 10 Sekunden, immer noch nicht alle Frame-Datenbytes empfangen wurden, muss die Verbindung geschlossen und neu initialisiert werden.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    hSocket     : T_HSOCKET;
    cbLen       : UDINT;
    pDest       : DWORD;
    bExecute    : BOOL;
    tTimeout    : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: Das Verbindungshandle [▶ 33] des Kommunikationspartners dessen Daten empfangen werden sollen.

cbLen: Die maximal verfügbare Puffergröße für die zu lesenden Daten in Bytes.

pDest: Die Adresse (Pointer) auf den Empfangspuffer.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    nRecBytes   : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

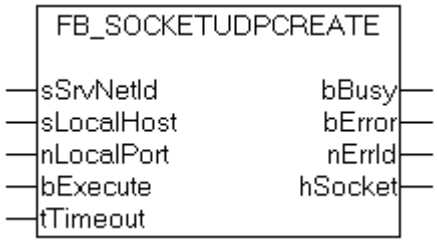
nRecBytes: Die Anzahl der zuletzt erfolgreich empfangen Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.8 FB_SocketUdpCreate

Ab der Produktversion: TwinCAT TCP/IP Connection Server v1,0,0,31 und höher



Mit dem Funktionsbaustein FB_SocketUdpCreate kann ein Client/Server-Socket für den User Datagram Protocol (UDP) geöffnet werden. Beim Erfolg wird ein neuer Socket geöffnet und am *hSocket*-Ausgang das dazugehörige Socket-Handle zurückgeliefert. Das Handle wird dann z.B. von den Funktionsbausteinen FB_SocketUdpSendTo [▶ 28] und FB_SocketUdpReceiveFrom [▶ 29] benötigt, um mit einem Remote-Teilnehmer Daten austauschen zu können. Ein nicht mehr benötigter UDP-Socket kann mit dem Funktionsbaustein FB_SocketClose [▶ 19] geschlossen werden. Die Portadresse *nLocalHost* wird intern von dem TCP/IP Connection Server für den UDP-Protokoll reserviert (es wird ein "Bind" durchgeführt). Es können mehrere Netzwerkadapter in einem PC existieren. Der Eingangsparameter *sLocalHost* bestimmt den Netzwerkadapter, der benutzt werden soll. Wenn Sie die *sLocalHost*-Eingangsvariable ignorieren (Leerstring), dann wird von dem TCP/IP Connection Server der Default-Netzwerkadapter benutzt. Es ist meistens der erste Netzwerkadapter aus der Liste der Netzwerkadapter in der Systemsteuerung.

- Wenn Sie beim Aufruf von FB_SocketUdpCreate als *sLocalHost* einen Leerstring angegeben haben und der PC vom Netzwerk getrennt wurde, dann öffnet das System einen neuen Socket unter der Software-Loopback-IP-Adresse: '127.0.0.1'.
- Wenn im PC zwei oder mehr Netzwerkadapter vorhanden sind und Sie als *sLocalHost* einen Leerstring angegeben haben, der Default-Netzwerkadapter aber vom Netzwerk getrennt wurde, dann wird der neue Socket unter der der IP-Adresse des zweiten Netzwerkadapters geöffnet.
- Um das Öffnen der Sockets unter einer anderen IP-Adresse zu verhindern können Sie die *sLocalHost*-Adresse explizit angeben oder die zurück gelieferte Adresse in der Handle-Variable (hSocket) überprüfen, den Socket schließen und erneut öffnen.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    sLocalHost      : STRING(15);
    nLocalPort      : UDINT;
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

sLocalHost: Die lokale IP-Adresse (Ipv4) des UDP Client/Server-Sockets als String (z.B.: '172.33.5.1'). Für den Default-Netzwerkadapter kann auch ein Leerstring angegeben werden

nLocalPort: Die lokale IP-Portnummer des UDP Client/Server-Sockets (z.B.: 200).

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hSocket     : T_HSOCKET;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

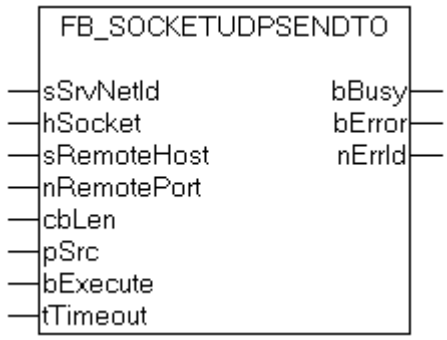
hSocket: Das Handle [▶ 33] des neu geöffneten UDP Client/Server-Sockets.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC or CX (x86)	TcpIp.Lib (v1.0.4 und höher) (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.9 FB_SocketUdpSendTo

Ab der Produktversion: TwinCAT TCP/IP Connection Server v1,0,0,31 und höher



Mit dem Funktionsbaustein FB_SocketUdpSendTo können UDP-Daten über den TwinCAT TCP/IP Connection Server zu einem Remote-Teilnehmer gesendet werden. Der UDP-Socket muss vorher mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 26] geöffnet werden.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    hSocket     : T_HSOCKET;
    sRemoteHost : STRING(15);
    nRemotePort : UDINT;
    cbLen       : UDINT;
    pSrc        : DWORD;
    bExecute    : BOOL;
    tTimeout    : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: Das Handle [▶ 33] eines geöffneten UDP-Sockets.

sRemoteHost: Die IP-Adresse (Ipv4) des Remote-Teilnehmers an den Daten gesendet werden sollen als String (z.B.: '172.33.5.1'). Für einen Teilnehmer auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

nRemotePort: Die IP-Portnummer des Remote-Teilnehmers an den Daten gesendet werden sollen (z.B.: 200).

cbLen: Die Anzahl der zu sendenden Daten in Bytes. Die maximale Anzahl der zu versendenden Datenbytes ist standardmäßig auf 8192 Bytes begrenzt (durch die Deklaration der TCPADS_MAXUDP_BUFFSIZE-Konstante in der Bibliothek, um Speicherresourcen zu schonen).

pSrc: Die Adresse (Pointer) auf den Sendepuffer.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ab der Produktversion: TwinCAT TCP/IP Connection Server **v1.0.50** und höher kann die maximale Anzahl der zu versendenden Datenbytes (wenn unbedingt notwendig) erhöht werden.

1) Definieren Sie in dem SPS-Projekt die globale Konstante um (in unserem Beispiel wollen wir die maximale Anzahl der Datenbytes auf 32000 Bytes erhöhen):

```
VAR_GLOBAL CONSTANT
    TCPADS_MAXUDP_BUFFSIZE : UDINT :=32000;
END_VAR
```

2) Aktivieren dann die Option *"Konstanten ersetzen"* im TwinCAT PLC Control->"*Projekt->Optionen...->Übersetzungsoptionen"* dialogfenster.

3) Übersetzen Sie das Projekt.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

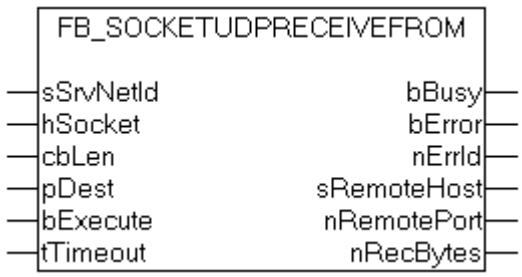
nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (v1.0.4 und höher) (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.10 FB_SocketUdpReceiveFrom

Ab der Produktversion: TwinCAT TCP/IP Connection Server v1,0,0,31 und höher



Mit dem Funktionsbaustein FB_SocketUdpReceiveFrom können über den TwinCAT TCP/IP Connection Server Daten eines geöffneten UDP-Sockets empfangen werden. Der UDP-Socket muss vorher mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 26] geöffnet werden. Die Instanz des FB_SocketUdpReceive-Funktionsbausteins muss zyklisch (pollend) in der SPS-Task aufgerufen werden. Dabei wird eine steigende Flanke z.B. alle 100ms an dem bExecute-Eingang erzeugt. Bei Erfolg werden die zuletzt empfangenen

Daten in den Empfangspuffer hineinkopiert. Der *nRecBytes*-Ausgang liefert die Anzahl der zuletzt erfolgreich empfangenen Datenbytes zurück. Wenn beim letzten Aufruf keine neuen Daten gelesen werden konnten, liefert der Funktionsbaustein keinen Fehler und *nRecBytes* == Null.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    hSocket     : T_HSOCKET;
    cbLen       : UDINT;
    pDest       : DWORD;
    bExecute    : BOOL;
    tTimeout    : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: Das Handle [▶ 33] eines geöffneten UDP-Sockets dessen Daten empfangen werden sollen.

cbLen: Die maximal verfügbare Puffergröße für die zu lesenden Daten in Bytes. Die maximale Anzahl der zu empfangenden Datenbytes ist standardmässig auf 8192 Bytes begrenzt (durch die Deklaration der TCPADS_MAXUDP_BUFFSIZE-Konstante in der Bibliothek, um Speicherresourcen zu schonen).

pDest: Die Adresse (Pointer) auf den Empfangspuffer.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ab der Produktversion: TwinCAT TCP/IP Connection Server **v1.0.50** und höher kann die maximale Anzahl der zu versendenden Datenbytes (wenn unbedingt notwendig) erhöht werden.

1) Definieren Sie in dem SPS-Projekt die globale Konstante um (in unserem Beispiel wollen wir die maximale Anzahl der Datenbytes auf 32000 Bytes erhöhen):

```
VAR_GLOBAL CONSTANT
    TCPADS_MAXUDP_BUFFSIZE : UDINT :=32000;
END_VAR
```

2) Aktivieren dann die Option *"Konstanten ersetzen"* im TwinCAT PLC Control->"*Projekt->Optionen...->Übersetzungsoptionen"* dialogfenster.

3) Übersetzen Sie das Projekt.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    sRemoteHost : STRING(15);
    nRemotePort : UDINT;
    nRecBytes   : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

sRemoteHost: Beim Erfolg die IP-Adresse (Ipv4) des Remote-Teilnehmers dessen Daten empfangen wurden.

nRemotePort: Beim Erfolg die IP-Portnummer des Remote-Teilnehmers dessen Daten empfangen wurden (z.B.: 200).

nRecBytes: Die Anzahl der zuletzt erfolgreich empfangen Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (v1.0.4 und höher) (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.11 FB_SocketUdpAddMulticastAddress

Ab der Produktversion: TwinCAT TCP/IP Connection Server 1.0.64 oder höher

Bindet den Server an eine Multicast IP Adresse, so dass Multicast Pakete empfangen werden können. Dieser Funktionsbaustein erwartet eine bereits hergestellte UDP Socketverbindung, welche über den Funktionsbaustein FB_SocketUdpCreate [▶ 26] hergestellt werden kann.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    hSocket         : T_HSOCKET;
    sMulticastAddr  : STRING(15);
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: Das Verbindungshandle [▶ 33] des Listener-Sockets. Dieses Handle muss vorher mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 26] angefordert werden.

sMulticastAddr: Multicast IP Adresse, an welche das Binding erfolgen soll.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt es bis zur Quittierung.

bError: Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId : Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC or CX (x86)	TcpIp.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.12 FB_SocketUdpDropMulticastAddress

Ab der Produktversion: TwinCAT TCP/IP Connection Server 1.0.64 oder höher

Entfernt das Binding an eine Multicast IP-Adresse, welches vorher über den Funktionsbaustein FB_SocketUdpAddMulticastAddress [▶ 31] eingerichtet wurde.

VAR_INPUT

```
VAR_INPUT
    sSrvNetId       : T_AmsNetId := '';
    hSocket         : T_HSOCKET;
    sMulticastAddr  : STRING(15);
    bExecute        : BOOL;
    tTimeout        : TIME := T#5s;
END_VAR
```

sSrvNetId: String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

hSocket: Das Verbindungshandle [▶ 33] des Listener-Sockets. Dieses Handle muss vorher mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 26] angefordert werden.

sMulticastAddr: Multicast IP-Adresse, an welche das Binding erfolgen soll.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt es bis zur Quittierung.

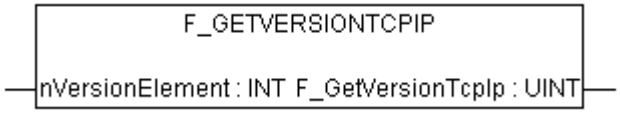
bError: Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId : Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC or CX (x86)	TcpIp.lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.13 F_GetVersionTcpIp



Mit dieser Funktion können Versionsinformationen der SPS-Bibliothek ausgelesen werden.

FUNCTION F_GetVersionTcpIp : UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

nVersionElement : Versionselement, das gelesen werden soll. Mögliche Parameter:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.14 ST_SockAddr

Struktur mit Adressinformationen eines geöffneten Sockets.

```
TYPE ST_SockAddr : (* Local or remote endpoint address *)
STRUCT
    nPort   : UDINT;        (* Internet Protocol (IP) port. *)
    sAddr   : STRING(15);   (* String containing an (Ipv4) Internet Protocol dotted address. *)
END_STRUCT
END_TYPE
```

nPort: Internet-Protokoll (IP) port;

sAddr: Durch Punkte getrennte Internetprotokolladresse (Ipv4) als String z.B.:"172.34.12.3";

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.1.15 T_HSOCKET

Variablen von diesem Typ repräsentieren ein Verbindungshandle oder ein Handle eines geöffneten Sockets. Über dieses Handle können Daten an einen Socket gesendet oder empfangen werden. Mit dem Handle kann ein geöffneter Socket wieder geschlossen werden.

```
TYPE T_HSOCKET :
STRUCT
    handle      : UDINT;
    localAddr   : ST_SockAddr; (* Local address *)
    remoteAddr  : ST_SockAddr; (* Remote endpoint address *)
END_STRUCT
END_TYPE
```

handle: Internes Socket-Handle des TwinCAT TCP/IP Connection Servers;

localAddr: Lokale Socketadresse;

remoteAddr: Remote Socketadresse;

Über den TwinCAT TCP/IP Connection Server können folgende Sockets geöffnet und geschlossen werden: Listener-Socket, Remote-Client-Socket oder Local-Client-Socket. Je nachdem, welcher von diesen Sockets von dem TwinCAT TCP/IP Connection Server geöffnet wurde, werden in die *localAddr*- und *remoteAddr*-Variablen die passenden Addressinformationen eingetragen.

Das Verbindungshandle auf der Serverseite

- Der Funktionsbaustein FB_SocketListen [▶ 22] öffnet einen Listener-Socket und liefert das Verbindungshandle des Listener-Sockets zurück;
- Das Verbindungshandle des Listener-Sockets wird an den Funktionsbaustein FB_SocketAccept [▶ 23] übergeben. FB_SocketAccept liefert dann die Verbindungshandles der Remote-Clients zurück;
- Für jeden verbundenen Remote-Client liefert der Funktionsbaustein FB_SocketAccept ein neues Verbindungshandle;
- Das Verbindungshandle wird dann an die Funktionsbausteine FB_SocketSend [▶ 24] und/oder FB_SocketReceive [▶ 25] übergeben, um Daten mit den Remote-Clients austauschen zu können;
- Ein Verbindungshandle eines nicht erwünschten oder nicht mehr benötigten Remote-Clients wird an den Funktionsbaustein FB_SocketClose [▶ 19] übergeben und so der Remote-Client-Socket geschlossen;
- Ein nicht mehr benötigtes Verbindungshandle des Listener-Sockets wird auch an den Funktionsbaustein FB_SocketClose übergeben und so der Listener-Socket geschlossen;

Das Verbindungshandle auf der Clientseite

- Der Funktionsbaustein FB_SocketConnect [▶ 18] liefert das Verbindungshandle eines Local-Client-Sockets zurück;
- Dieses Verbindungshandle wird dann an die Funktionsbausteine FB_SocketSend [▶ 24] und FB_SocketReceive [▶ 25] übergeben, um Daten mit einem Remote-Server austauschen zu können;
- Das gleiche Verbindungshandle wird dann an den Funktionsbaustein FB_SocketClose [▶ 19] übergeben, um eine nicht mehr benötigte Verbindung zu schließen.

Mit dem Funktionsbaustein FB_SocketCloseAll [▶ 20] werden alle Sockets geschlossen, die von einem SPS-Laufzeitsystem geöffnet wurden. D.h. wenn Sie FB_SocketCloseAll in einer der Tasks des ersten Laufzeitsystems (Port 801) aufrufen, werden alle Sockets geschlossen, die in dem ersten Laufzeitsystem geöffnet wurden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

ST_SockAddr [▶ 33]

7.1.16 E_WinsockError

```
TYPE E_WinsockError :
  (
    WSOK,
    WSAEINTR                :=      10004   ,(* A blocking operation was interrupted by a call to
WSACancelBlockingCall. *)
    WSAEBADF                :=      10009   ,(* The file handle supplied is not valid. *)
```

```
    WSAEACCES               :=      10013   ,(* An attempt was made to access a socket in a way
forbidden by its access permissions. *)
    WSAEFAULT               :=      10014   ,
(* The system detected an invalid pointer address in attempting to use a pointer argument in a call.
 *)
    WSAEINVAL               :=      10022   ,(* An invalid argument was supplied. *)
    WSAEMFILE               :=      10024   ,(* Too many open sockets. *)
    WSAEWOULDBLOCK          :=      10035   ,(* A non-
blocking socket operation could not be completed immediately. *)
    WSAEINPROGRESS          :=      10036   ,(* A blocking operation is currently executing. *)
    WSAEALREADY             :=      10037   ,(* An operation was attempted on a non-
blocking socket that already had an operation in progress.  *)
    WSAENOTSOCK             :=      10038   ,
(* An operation was attempted on something that is not a socket.  *)
    WSAEDESTADDRREQ         :=      10039   ,
(* A required address was omitted from an operation on a socket.  *)
    WSAEMSGSIZE             :=      10040   ,(* A message sent on a datagram socket was larger than
the internal message buffer or some other network limit, or the buffer used to receive a datagram
into was smaller than the datagram itself. *)
    WSAEPROTOTYPE           :=      10041   ,
(* A protocol was specified in the socket function call that does not support the semantics of the s
ocket type requested.  *)
    WSAENOPROTOOPT          :=      10042   ,
(* An unknown, invalid, or unsupported option or level was specified in a getsockopt or setsockopt c
all.  *)
    WSAEPROTONOSUPPORT      :=      10043   ,
(* The requested protocol has not been configured into the system, or no implementation for it exist
s.  *)
    WSAESOCKTNOSUPPORT      :=      10044   ,(* The support for the specified socket type does not
exist in this address family. *)
    WSAEOPNOTSUPP           :=      10045   ,
(* The attempted operation is not supported for the type of object referenced.  *)
    WSAEPFNOSUPPORT         :=      10046   ,
(* The protocol family has not been configured into the system or no implementation for it exists.
*)
    WSAEAFNOSUPPORT         :=      10047   ,
(* An address incompatible with the requested protocol was used.  *)
    WSAEADDRINUSE           :=      10048   ,(* Only one usage of each socket address (protocol/
network address/port) is normally permitted. *)
    WSAEADDRNOTAVAIL    :=      10049   ,(* The requested address is not valid in its context.  *)
    WSAENETDOWN             :=      10050   ,(* A socket operation encountered a dead network.  *)
    WSAENETUNREACH          :=      10051   ,
(* A socket operation was attempted to an unreachable network.  *)
    WSAENETRESET            :=      10052   ,(* The connection has been broken due to keep-alive
activity detecting a failure while the operation was in progress. *)
    WSAECONNABORTED         :=      10053   ,
(* An established connection was aborted by the software in your host machine.  *)
    WSAECONNRESET           :=      10054   ,
(* An existing connection was forcibly closed by the remote host.  *)
    WSAENOBUFS              :=      10055   ,
(* An operation on a socket could not be performed because the system lacked sufficient buffer space
 or because a queue was full.  *)
    WSAEISCONN              :=      10056   ,
(* A connect request was made on an already connected socket.  *)
    WSAENOTCONN             :=      10057   ,
(* A request to send or receive data was disallowed because the socket is not connected and (when se
nding on a datagram socket using a sendto call) no address was supplied.  *)
    WSAESHUTDOWN            :=      10058   ,
(* A request to send or receive data was disallowed because the socket had already been shut down in
 that direction with a previous shutdown call.  *)
    WSAETOOMANYREFS         :=      10059   ,(* Too many references to some kernel object.  *)
    WSAETIMEDOUT            :=      10060   ,
(* A connection attempt failed because the connected party did not properly respond after a period o
f time, or established connection failed because connected host has failed to respond.  *)
    WSAECONNREFUSED         :=      10061   ,
(* No connection could be made because the target machine actively refused it.  *)
    WSAELOOP                :=      10062   ,(* Cannot translate name.  *)
    WSAENAMETOOLONG         :=      10063   ,(* Name component or name was too long.  *)
    WSAEHOSTDOWN            :=      10064   ,
(* A socket operation failed because the destination host was down.  *)
    WSAEHOSTUNREACH         :=      10065   ,
(* A socket operation was attempted to an unreachable host.  *)
    WSAENOTEMPTY            :=      10066   ,(* Cannot remove a directory that is not empty.  *)
    WSAEPROCLIM             :=      10067   ,
(* A Windows Sockets implementation may have a limit on the number of applications that may use it s
imultaneously.  *)
    WSAEUSERS               :=      10068   ,(* Ran out of quota.  *)
    WSAEDQUOT               :=      10069   ,(* Ran out of disk quota.  *)
    WSAESTALE               :=      10070   ,(* File handle reference is no longer available.  *)
    WSAEREMOTE              :=      10071   ,(* Item is not available locally.  *)
```

```
    WSASYSNOTREADY          :=      10091   ,(* WSAStartup cannot function at this time because the
underlying system it uses to provide network services is currently unavailable. *)
    WSAVERNOTSUPPORTED      :=      10092   ,
(* The Windows Sockets version requested is not supported.  *)
    WSANOTINITIALISED       :=      10093   ,
(* Either the application has not called WSAStartup, or WSAStartup failed.  *)
    WSAEDISCON              :=      10101   ,
(* Returned by WSARecv or WSARecvFrom to indicate the remote party has initiated a graceful shutdown
 sequence.  *)
    WSAENOMORE              :=      10102   ,
(* No more results can be returned by WSALookupServiceNext.  *)
    WSAECANCELLED           :=      10103   ,
(* A call to WSALookupServiceEnd was made while this call was still processing. The call has been ca
nceled.  *)
    WSAEINVALIDPROCTABLE    :=      10104   ,(* The procedure call table is invalid.  *)
    WSAEINVALIDPROVIDER     :=      10105   ,(* The requested service provider is invalid.  *)
    WSAEPROVIDERFAILEDINIT  :=      10106   ,
(* The requested service provider could not be loaded or initialized.  *)
    WSASYSCALLFAILURE       :=      10107   ,(* A system call that should never fail has failed.  *)
    WSASERVICE_NOT_FOU      ND      :=      10108   ,
(* No such service is known. The service cannot be found in the specified name space.  *)
    WSATYPE_NOT_FOUND       :=      10109   ,(* The specified class was not found.  *)
    WSA_E_NO_MORE           :=      10110   ,
(* No more results can be returned by WSALookupServiceNext.  *)
    WSA_E_CANCELLED         :=      10111   ,
(* A call to WSALookupServiceEnd was made while this call was still processing. The call has been ca
nceled.  *)
    WSAEREFUSED             :=      10112   ,
(* A database query failed because it was actively refused.  *)
    WSAHOST_NOT_FOUND       :=      11001   ,(* No such host is known.  *)
    WSATRY_AGAIN            :=      11002   ,
(* This is usually a temporary error during hostname resolution and means that the local server did
not receive a response from an authoritative server.  *)
    WSANO_RECOVERY          :=      11003   ,(* A non-
recoverable error occurred during a database lookup.  *)
    WSANO_DATA              :=      11004   (* The requested name is valid and was found in the data
base, but it does not have the correct associated data being resolved for.  *)
  );
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcpIp.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.1.17 Globale Variablen

```
VAR_GLOBAL CONSTANT
    AMSPORT_TCPIPSRV            :UINT   := 10201;

    TCPADS_IGR_CONLIST          :UDINT  :=16#80000001;
    TCPADS_IGR_CLOSEBYHDL       :UDINT  :=16#80000002;
    TCPADS_IGR_SENDBYHDL        :UDINT  :=16#80000003;
    TCPADS_IGR_PEERBYHDL        :UDINT  :=16#80000004;
    TCPADS_IGR_RECVBYHDL        :UDINT  :=16#80000005;
    TCPADS_IGR_RECVFROMBYHDL    :UDINT  :=16#80000006;(* TCP/
IP Connection Server v1,0,0,31 and higher *)
    TCPADS_IGR_SENDTOBYHDL  :UDINT  :=16#80000007;(* TCP/
IP Connection Server v1,0,0,31 and higher *)

    TCPADSCONLST_IOF_CONNECT    :UDINT  :=1;
    TCPADSCONLST_IOF_LISTEN     :UDINT  :=2;
    TCPADSCONLST_IOF_CLOSEALL   :UDINT  :=3;
    TCPADSCONLST_IOF_ACCEPT     :UDINT  :=4;
    TCPADSCONLST_IOF_UDPBIND    :UDINT  :=5;(* TCP/IP Connection Server v1,0,0,31 and higher *)

    TCPADS_MAXUDP_BUFFSIZE      : UDINT :=1472;(* TCP/IP Connection Server v1,0,0,31 and higher *)
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Alle TwinCAT v2.8.0 Versionen und höher	PC oder CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.2 TcSocketHelper.lib

Die Bibliothek **TcSocketHelper.Lib** beinhaltet TCP/IP-Hilfsfunktionen.

Systemvoraussetzungen:

Programmierumgebung:

- TwinCAT System version 2.9 oder höher (NT4, W2K, XP, XPe);
- TwinCAT Installation level: TwinCAT PLC oder höher;

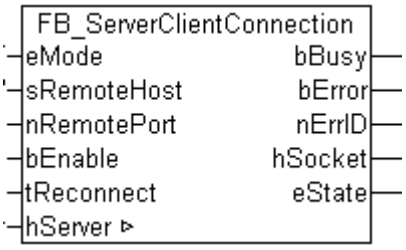
Zielplattform:

- TwinCAT SPS-Laufzeitsystem version 2.9 oder höher;
- PC or CX (x86)
  - TwinCAT TCP/IP Connection Server **v1.0.0.41** oder höher;
  - NT4, W2K, XP, XPe, CE (image v1.75 oder höher);
- CX (ARM)
  - TwinCAT TCP/IP Connection Server **v1.0.0.44** oder höher;
  - CE (image v2.13 oder höher);

Installation:

Die SPS-Bibliothek wird mit dem TwinCAT TCP/IP Connection Server mitgeliefert und während der Installation in den ...\TwinCAT\PLC\Lib-Ordner kopiert.

7.2.1 FB_ServerClientConnection



Mit dem FB_ServerClientConnection-Funktionsbaustein kann eine Server-Verbindung verwaltet werden (auf- und abgebaut werden). FB_ServerClientConnection vereinfacht die Implementierung einer Server-Applikation indem er die Funktionalität von den drei Funktionsbausteinen FB_SocketListen [▶ 22], FB_SocketAccept [▶ 23] und FB_SocketClose [▶ 19] bereits intern kapselt. Die integrierte Debug-Ausgabe des Verbindungsstatus erleichtert die Fehlersuche bei Konfigurations- oder Kommunikationsfehlern. Eine minimale Server-Applikation benötigt zusätzlich nur noch jeweils eine Instanz vom FB_SocketSend [▶ 24] und/ oder eine Instanz vom FB_SocketReceive [▶ 25] Funktionsbaustein.

Eine typische Server-Applikation stellt im ersten Schritt mit dem FB_ServerClientConnection-Funktionsbaustein die Verbindung zum Client her (genauer gesagt wird der eingehende Verbindungswunsch von der Server-Applikation akzeptiert). Im nächsten Schritt können dann Instanzen von FB_SocketSend und/ oder FB_SocketReceive benutzt werden, um Daten mit dem Server auszutauschen. Wann eine Verbindung geschlossen wird, hängt von den Anforderungen der Applikation ab.

VAR_IN_OUT

```
VAR_IN_OUT
    hServer         : T_HSERVER;
END_VAR
```

hServer: Das Server-Handle [▶ 43]. Diese Eingangsvariable muss vorher mit der Funktion F_CreateServerHnd [▶ 42] initialisiert werden.

VAR_INPUT

```
VAR_INPUT
    eMode       : E_SocketAcceptMode := eACCEPT_ALL;
    sRemoteHost : STRING(15) := '';
    nRemotePort : UDINT := 0;
    bEnable     : BOOL;
    tReconnect  : TIME := T#1s;
END_VAR
```

eMode: Legt fest, ob alle oder nur bestimmte Verbindungen [▶ 44] akzeptiert werden sollen.

sRemoteHost: Die IP-Adresse (Ipv4) des Remote-Clients dessen Verbindung akzeptiert werden soll als String (z.B.: '172.33.5.1'). Für einen Client auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

nRemotePort: Die IP-Portnummer des Remote-Clients dessen Verbindung akzeptiert werden soll (z.B.: 200).

bEnable: Solange dieser Eingang TRUE ist, wird zyklisch versucht eine neue Verbindung herzustellen, solange bis eine Verbindung hergestellt wurde. Mit FALSE kann eine hergestellte Verbindung wieder geschlossen werden.

tReconnect: Zykluszeit mit welcher der Funktionsbaustein versucht, eine Verbindung herzustellen. Frühestens nach dieser Zeit wird ein erneuter Versuch durchgeführt, eine neue Verbindung zu akzeptieren.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    hSocket     : T_HSOCKET;
    eState      : E_SocketConnectionState := eSOCKET_DISCONNECTED;
END_VAR
```

bBusy: Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.

bError: Wird TRUE, sobald ein Fehler aufgetreten ist.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

hSocket: Das Verbindungshandle zu dem neu geöffneten Remote-Client Socket. Diese Variable wird bei Erfolg an die Instanzen der Funktionsbausteine FB_SocketSend [▶ 24] und/oder FB_SocketReceive [▶ 25] übergeben.

eState: Liefert den aktuellen Verbindungsstatus.

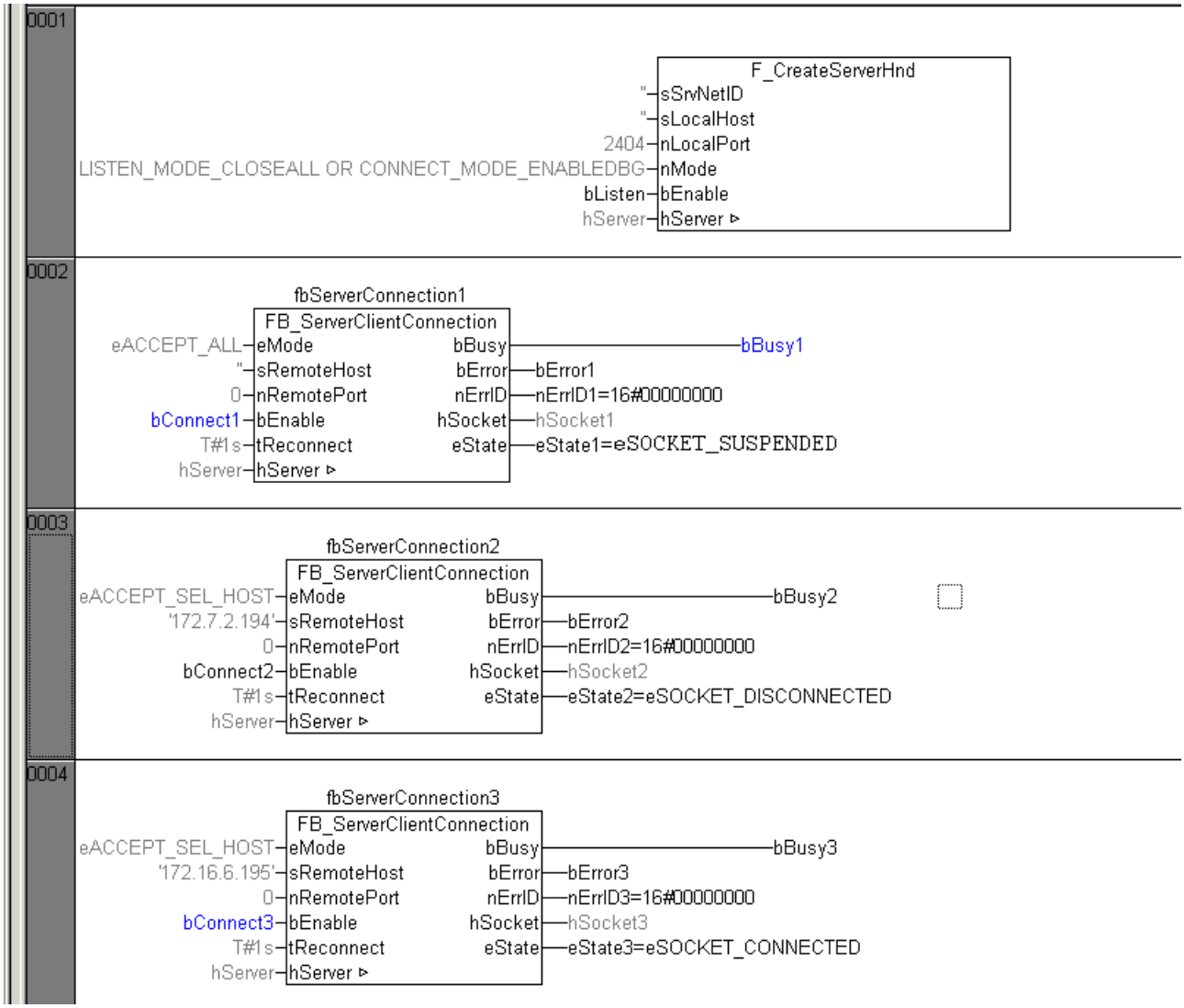
Beispiel in FUP:

Das folgende Beispiel zeigt die Initialisierung einer Server-Handle-Variablen. Das Server-Handle wird dann an drei Instanzen des FB_ServerClientConnection-Funktionsbausteins übergeben.

```
PROGRAM MAIN
VAR
    hServer         : T_HSERVER;
    bListen         : BOOL;

    fbServerConnection1 : FB_ServerClientConnection;
    bConnect1       : BOOL;
    bBusy1      : BOOL;
```

```
    bError1         : BOOL;
    nErrID1         : UDINT;
    hSocket1    : T_HSOCKET;
    eState1         : E_SocketConnectionState;

    fbServerConnection2 : FB_ServerClientConnection;
    bConnect2       : BOOL;
    bBusy2      : BOOL;
    bError2         : BOOL;
    nErrID2         : UDINT;
    hSocket2    : T_HSOCKET;
    eState2         : E_SocketConnectionState;

    fbServerConnection3 : FB_ServerClientConnection;
    bConnect3       : BOOL;
    bBusy3      : BOOL;
    bError3         : BOOL;
    nErrID3         : UDINT;
    hSocket3    : T_HSOCKET;
    eState3         : E_SocketConnectionState;
END_VAR
```

Online-Ansicht:



Die erste Verbindung ist aktiviert (*bConnect1*=TRUE), die Verbindung wurde aber noch nicht hergestellt (Passive open).

Die zweite Verbindung wurde noch nicht aktiviert (*bConnect2*=FALSE) (Closed).

Die dritte Verbindung wurde aktiviert (*bConnect3*=TRUE) und es wurde eine Verbindung zum Remote-Client hergestellt (Established).

Weitere Anwendungsbeispiele (inklusive Sourcecode) finden Sie hier: Beispiele [▶ 80]

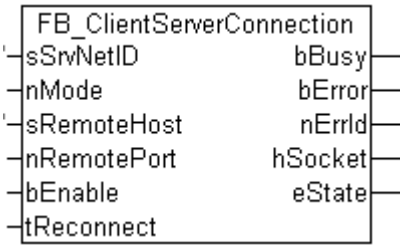
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden)

Sehen Sie dazu auch

- E_SocketConnectionState [▶ 44]

7.2.2 FB_ClientServerConnection



Mit dem Funktionsbaustein FB_ClientServerConnection kann eine Client-Verbindung verwaltet werden (auf- und abgebaut werden). FB_ClientServerConnection vereinfacht die Implementierung einer Client-Applikation in dem er die Funktionalität von den zwei Funktionsbausteinen FB_SocketConnect [▶ 18] und FB_SocketClose [▶ 19] bereits intern kapselt. Die integrierte Debug-Ausgabe des Verbindungsstatus erleichtert die Fehlersuche bei Konfigurations- oder Kommunikationsfehlern. Eine minimale Client-Applikation benötigt zusätzlich nur noch jeweils eine Instanz vom FB_SocketSend [▶ 24] und/oder eine Instanz vom FB_SocketReceive [▶ 25] Funktionsbaustein.

Eine typische Client-Applikation stellt im ersten Schritt mit dem FB_ClientServerConnection-Funktionsbaustein die Verbindung zum Server her. Im nächsten Schritt können dann Instanzen von FB_SocketSend und/oder FB_SocketReceive benutzt werden, um Daten mit dem Server auszutauschen. Wann eine Verbindung geschlossen wird, hängt von den Anforderungen der Applikation ab.

VAR_INPUT

```
VAR_INPUT
    sSrvNetID       : T_AmsNetID := '';
    nMode           : DWORD := 0;
    sRemoteHost     : STRING(15) := '';
    nRemotePort     : UDINT;
    bEnable         : BOOL;
    tReconnect      : TIME := T#45s;
END_VAR
```

sSrvNetID: String mit der Ams-Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

nMode: Parameter-Flags (Modi). Die zulässigen Parameter sind in der Tabelle aufgeführt und können mit einer ODER-Verknüpfung kombiniert werden:

Flag	Beschreibung
CONNECT_MODE_ENABLEDBG	Aktiviert das Loggen von Debug-Meldungen im Application-Log. Um die Debug-Meldungen zu sehen, öffnen Sie den TwinCAT System Manager und aktivieren Sie die Loggeransicht.

sRemoteHost: Die IP-Adresse (Ipv4) des Remote-Servers als String (z.B.: '172.33.5.1'). Für einen Server auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

nRemotePort: Die IP-Portnummer des Remote-Servers (z.B.: 200).

bEnable: Solange dieser Eingang TRUE ist wird zyklisch versucht eine Verbindung aufzubauen, solange bis eine Verbindung aufgebaut wurde. Mit FALSE kann eine aufgebaute Verbindung wieder geschlossen werden.

tReconnect: Zykluszeit mit der der Funktionsbaustein versucht die Verbindung aufzubauen. Spätestens nach dieser Zeit wird der Versuch abgebrochen und ein neuer gestartet.

Setzen Sie den Wert *tReconnect* nicht zu niedrig, da bei einer Netzwerkunterbrechung Timeoutzeiten von > 30s auftreten können. Bei einem zu niedrigen Wert wird die Kommandoausführung vorzeitig unterbrochen und der ADS-Fehlercode: 1861 (timeout elapsed) statt des Winsocket-Fehlers: WSAETIMEDOUT zurückgeliefert.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hSocket     : T_HSOCKET;
    eState      : E_SocketConnectionState := eSOCKET_DISCONNECTED;
END_VAR
```

bBusy: Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.

bError: Wird TRUE, sobald ein Fehler aufgetreten ist.

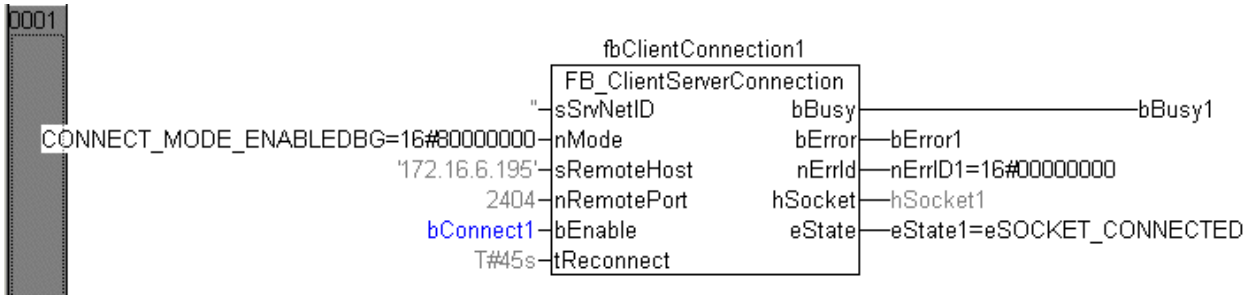
nErrID: Liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 88].

hSocket: Das Verbindungshandle zu dem neu geöffneten Local-Client Socket. Diese Variable wird bei Erfolg an die Instanzen der Funktionsbausteine FB_SocketSend [▶ 24] und/oder FB_SocketReceive übergeben.

eState: Liefert den aktuellen Verbindungsstatus [▶ 44].

Beispiel für einen Aufruf in FUP:

```
PROGRAM MAIN
VAR
    fbClientConnection1     : FB_ClientServerConnection;
    bConnect1               : BOOL;
    bBusy1                  : BOOL;
    bError1                 : BOOL;
    nErrID1                 : UDINT;
    hSocket1                : T_HSOCKET;
    eState1                 : E_SocketConnectionState;
END_VAR
```



Weitere Anwendungsbeispiele (inklusive Sourcecode) finden Sie hier: Beispiele [▶ 80]

Voraussetzungen

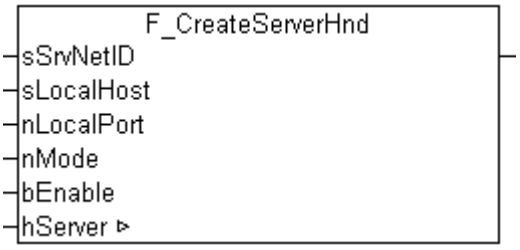
Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden

Sehen Sie dazu auch

FB_SocketReceive [▶ 25]

7.2.3 F_CreateServerHnd



Mit der Funktion F_CreateServerHnd werden die internen Parameter einer Server-Handle-Variablen *hServer* initialisiert/gesetzt. Das Server-Handle wird dann an die Instanzen des FB_ServerClientConnection [▶ 37] Funktionsbausteins per VAR_IN_OUT übergeben. Mit einer Instanz des FB_ServerClientConnection Funktionsbausteins kann eine Verbindung des Servers auf einfache Weise verwaltet werden (auf- und abgebaut werden). Soll ein Server mehrere Verbindungen gleichzeitig aufbauen können, dann wird das gleiche Server-Handle an mehrere Instanzen des FB_ServerClientConnection Funktionsbausteins übergeben.

FUNCTION F_CreateServerHnd : BOOL

```
VAR_IN_OUT
    hServer         : T_HSERVER;
END_VAR
VAR_INPUT
    sSrvNetID       : T_AmsNetID := '';
    sLocalHost      : STRING(15) := '';
    nLocalPort      : UDINT := 0;
    nMode           : DWORD := LISTEN_MODE_CLOSEALL (* OR CONNECT_MODE_ENABLEDBG*);
 bEnable            : BOOL := TRUE;
END_VAR
```

hServer: Die Server-Handle [▶ 43]-Variable deren interne Parameter initialisiert werden sollen.

sSrvNetID: String mit der Ams-Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

sLocalHost:Die Local-Server IP-Adresse (Ipv4) als String (z.B.: '172.13.15.2'). Für einen Server auf dem lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

nLocalPort:Der Local-Server IP-Port (z.B.: 200).

nMode: Parameter-Flags (Modi). Die zulässigen Parameter sind in der Tabelle aufgeführt und können mit einer ODER Verknüpfung kombiniert werden:

Flag	Beschreibung
LISTEN_MODE_CLOSEALL	Alle vorher geöffneten Socket-Verbindungen werden zuerst geschlossen (default).
CONNECT_MODE_ENABLEDBG	Aktiviert das Loggen von Debug-Meldungen im Application-Log. Um die Debug-Meldungen zu sehen, öffnen Sie den TwinCAT System Manager und aktivieren Sie die Loggeransicht.

bEnable: Dieser Eingang legt das Verhalten des Listener-Sockets fest. Ein vorher geöffneter Listener-Socket bleibt geöffnet solange dieser Eingang TRUE ist . Wenn dieser Eingang FALSE ist, dann wird der Listener-Socket automatisch geschlossen aber erst dann, nach dem die letzte (vorher) akzeptierte Verbindung auch geschlossen wurde.

Rückgabeparameter	Bedeutung
TRUE	Kein Fehler
FALSE	Falscher/unzulässiger Parameterwert

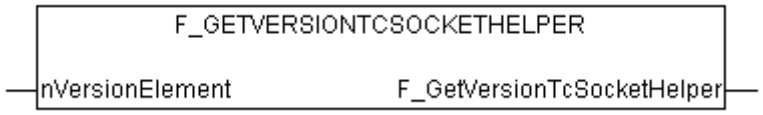
Beispiel:

Siehe in der Beschreibung von FB_ServerClientConnection [▶ 37].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden

7.2.4 F_GetVersionTcSocketHelper



Mit dieser Funktion können Versionsinformationen der SPS-Bibliothek ausgelesen werden.

FUNCTION F_GetVersionTcSocketHelper : UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

nVersionElement : Versionselement, das gelesen werden soll. Mögliche Parameter:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden

7.2.5 T_HSERVER

Eine Variable von diesem Typ repräsentiert ein TCP/IP-Server-Handle. Das Handle muss vor der Benutzung mit der Funktion F_CreateServerHnd [▶ 42] initialisiert werden. Dabei werden die internen Parameter der T_HSERVER-Variablen gesetzt.

Die Strukturelemente sollen nicht direkt beschrieben oder verändert werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC oder CX (x86)	TcSocketHelper.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden

7.2.6 E_SocketAcceptMode

```
TYPE E_SocketAcceptMode:
(* Connection accept modes *)
(
    eACCEPT_ALL, (* Accept connection to all remote clients *)
    eACCEPT_SEL_HOST, (* Accept connection to selected host address *)
    eACCEPT_SEL_PORT, (* Accept connection to selected port address *)
    eACEPT_SEL_HOST_PORT (* Accept connection to selected host and port address *)
);
END_TYPE
```

Über eine Variable vom Typ E_SocketAcceptMode wird festgelegt welche Verbindungen von einem Server akzeptiert werden sollen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden)

7.2.7 E_SocketConnectionState

```
TYPE E_SocketConnectionState:
(
    eSOCKET_DISCONNECTED,
    eSOCKET_CONNECTED,
    eSOCKET_SUSPENDED
);
END_TYPE
```

Status der TCP/IP-Socketverbindung (eSOCKET_SUSPENDED == der Status wechselt gerade z.B. von eSOCKET_CONNECTED=>eSOCKET_DISCONNECTED).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	(Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden)

7.2.8 Globale Konstanten

Konstante	Wert	Beschreibung
LISTEN_MODE_CLOSEALL	16#00000001	Erzwingt die Schließung aller vorher geöffneten Verbindungen.
LISTEN_MODE_USEOPENED	16#00000002	Ist dieses Flag gesetzt, dann wird versucht von einem bereits geöffneten Listener-Socket das Verbindungshandle zu benutzen und mit diesem Handle neue Verbindungen zu akzeptieren. (es wurde z.B. Listener-Socket geöffnet und anschließend ein SPS-Reset durchgeführt). Nur in besonderen Fällen sinnvoll, weil die bereits geöffneten Client-Verbindungen nicht mehr benutzt werden können.
CONNECT_MODE_ENABLEDBG	16#80000000	Ist dieses Flag gesetzt, dann werden Debugmeldungen beim Auf- oder Abbau der TcpIp-Verbindungen in die Application-Log geschrieben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >= 1030	PC or CX (x86)	TcSocketHelper.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib; TcpIp.Lib werden automatisch eingebunden
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

7.3 TcSnmp.lib

Die TcSnmp.Lib ermöglicht das Versenden von SNMP Traps und das Verarbeiten von SNMP Get Befehlen der SNMP Version 1.

Es muss der **TwinCAT TCP/IP Connection Server** installiert sein.

Produktkomponenten

- TcSNMP.Lib

Installation

Windows NT (NT4, W2K, XP, XPe)

Befolgen Sie folgende Schritte:

- Installieren Sie den TwinCAT TCP/P Connection Server. Nach der Installation wird der Server mit TwinCAT automatisch gestartet.
- Die SPS-Bibliotheken werden in den ...\TwinCAT\PLC\Lib-Ordner kopiert.

7.3.1 FB_SEND_TRAP

Der Funktionsblock erlaubt SNMPv1 Traps zu verschicken. Sie können eine variable Anzahl an Variablen Bindings hinzufügen, um sie an einen SNMP Manager via Traps zu verschicken. Folgende Datentypen werden vom Funktionsblock unterstützt: OCTET_STRING, INTEGER32, COUNTER32, GAUGE32, TIMETICKS, OBJECT_ID

Die maximale Paketgröße ist begrenzt auf 2000 Bytes. Wenn mehr Bytes übertragen werden, erhält der Funktionsblock einen Fehler zurück.

Der Funktionsblock FB_SEND_TRAP sollte in jedem Zyklus aufgerufen werden, um ordnungsgemäß zu funktionieren.

```
VAR_INPUT
    bEnable               :BOOL;
    bExecute              :BOOL;
    sLocalHostIp          :STRING(15);
    sLocalHostPort        :UDINT;
    sManagerIP            :STRING(15);
    sTcIpConnSvrAddr      :T_AmsNetId;
    sObjectID             :T_MAXSTRING;
    sCommunity            :STRING(60);
    iGenericTrapNumber    :INT;
    bySpecificTrapNumber  :BYTE;
    nVarBindings          :USINT;
    pArrVarBinding        :POINTER TO ARRAY[1..iMAX_TRAPBUF_SIZE] OF ST_SNMP_VariableBinding;
END_VAR
```

bEnable: Mit der steigenden Flanke am Eingang versucht das System einen Socket zu erzeugen. Wenn dieser erstellt wurde, wird bEnabled auf TRUE gesetzt. Der Socket kann mit einer fallende Flanke wieder geschlossen werden.

bExecute: Sendet einen Trap mit einer steigenden Flanke an bExecute. Eine steigende Flanke setzt die Ausgänge nErrID und bError zurück. Es wird ein offener Socket benötigt.

sLocalHostIP: String der die IP-Adresse (v4) des lokalen Host (z.B. '172.33.5.1') beinhaltet. Wenn mehr als ein Netzwerkadapter auf dem Rechner vorhanden sind, erlaubt der Parameter den spezifischen Adapter zu verwenden.

sLocalHostPort(optional): Die Port Nr.

sManagerIP: IP Adresse (IPv4) des SNMP Managers.

sTcIpConnSvrAddr: nicht unterstützt.

sObjectID: Beinhaltet den numerischen Wert der die MIB (Management Information Base) des Gerätes repräsentiert. Maximale Länge des Strings ist 255. Der gültige Wertebereich für jeden Nummer ist 0...65535 (e.g. '1.3.1.3.2555.3')

sCommunity: Beinhaltet den SNMP Community String (e.g. public)

iGenericTrapNumber: Die SNMP Generic Trap Nummer definiert in E_SNMP_GenericTrapNumber.

bySpecificTrapNumber: Die SNMP Specific Trap Nummer wird automatisch auf 0 gesetzt, wenn iGenericTrapNumber nicht 0x06(E_SNMP_EnterpriseSpecific) ist. Gültiger Wertebereich ist 1...255.

nVarBindings (optional): Anzahl der Elemente von pArrVarBinding. Maximum ist iMAX_TRAPBUF_SIZE (definiert in den globalen Variablen).

pArrVarBinding (optional): Pointer auf ein Array des SNMP_ST_VariableBinding.

```
VAR_OUTPUT
    bBusy         :BOOL;
    bEnabled      :BOOL;
    bError        :BOOL;
    nErrID        :DINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bEnabled: Der Ausgang wird gesetzt, wenn ein Socket geöffnet wird.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrID: Liefert bei einem bError-Ausgang die TwinCAT TCP/IP Connection Server error [▶ 88]Fehlernummer zurück.

Der Funktionsblock wurde mit folgender Software getestet:

SNMP Trap Watcher (BTT Software)

Wireshark 1.2.5

iReasoning MIB Browser Personal Edition 7.0

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.3.2 FB_GetSnmp

Der Funktionsblock ermöglicht es, auf SNMPv1 Get-Befehle zu reagieren. Sie können eine variable Anzahl an Variablen Bindings hinzufügen, um sie an einen SNMP Manager via Traps zu verschicken. Folgende Datentypen werden vom Funktionsblock unterstützt: OCTET_STRING, INTEGER32, COUNTER32, GAUGE32, TIMETICKS, OBJECT_ID

```
VAR_INPUT
    bEnable               :BOOL;
    bReceive              :BOOL;
    bSendTrap             :BOOL;
    bSendResponse         :BOOL;
    sLocalHostIp          :STRING(15);
    nLocalHostPort        :UDINT;
    sManagerIP            :STRING(15);
    sTcIpConnSvrAddr      :T_AmsNetId :='';
    sObjectID             :T_MAXSTRING;
    sCommunity            :STRING(60);
    iGenericTrapNumber    :INT;
    iSpecificTrapNumber   :BYTE;
    nVarBindings          :USINT := 0;
    pArrVarBinding        :POINTER TO ARRAY[1..iMAX_TRAPBUF_SIZE] OF ST_SNMP_VariableBinding;
    iError                :INT;
END_VAR
```

bEnable: Mit der steigenden Flanke am Eingang versucht das System einen Socket zu erzeugen. Wenn dieser erstellt wurde, wird bEnabled auf TRUE gesetzt. Der Socket kann mit einer fallende Flanke wieder geschlossen werden.

bReceive: Signalisiert eine eingehende Anfrage.

bSendTrap: Signalisiert einen zu versendenden Trap.

bSendResponse: Signalisiert eine zu versendende Antwort.

sLocalHostIP: String der die IP-Adresse (v4) des lokalen Host (z.B. '172.33.5.1') beinhaltet. Wenn mehr als ein Netzwerkadapter auf dem Rechner vorhanden sind, erlaubt der Parameter den spezifischen Adapter zu verwenden.

nLocalHostPort(optional): Die zu verwendende Port Nr.

sManagerIP: IP Adresse (IPv4) des SNMP Managers.

sTcIpConnSvrAddr: nicht unterstützt.

sObjectID: Beinhaltet den numerischen Wert der die MIB (Management Information Base) des Gerätes repräsentiert. Maximale Länge des Strings ist 255. Der gültige Wertebereich für jede Nummer ist 0...65535 (e.g. '1.3.1.3.2555.3')

sCommunity: Beinhaltet den SNMP Community String (e.g. public).

iGenericTrapNumber: Die SNMP Generic Trap Nummer definiert in E_SNMP_GenericTrapNumber.

iSpecificTrapNumber: Die SNMP Specific Trap Nummer wird automatisch auf 0 gesetzt, wenn iGenericTrapNumber nicht 0x06 (E_SNMP_EnterpriseSpecific) ist. Gültiger Wertebereich ist 1...255.

nVarBindings (optional): Anzahl der Elemente von pArrVarBinding. Maximum ist iMAX_TRAPBUF_SIZE (definiert in den globalen Variablen).

pArrVarBinding (optional): Pointer auf ein Array des SNMP_ST_VariableBinding.

iError: Rückgabe von spezifischen SNMP Fehlercodes.

```
VAR_OUTPUT

    bBusy               :BOOL;
    bError              :BOOL;
    bEnabled            :BOOL;
    bReceived           :BOOL;
    iRecSNMPVersion     :INT; (* SNMP Version,sample SNMPv1 = 0 *)
    sRecCommunity       :STRING(60); (*Community Name *)
    iRecPDUType         :INT;(* SNMP PDU Type, sample GET = A0 *)
    arrRecPDURequestID  :ARRAY[0..3] OF BYTE; (* SNMP PDU Request ID *)
    iRecPDUError        :INT; (*SNMP PDU Error *)
    iRecPDUErrorIndex   :INT; (* SNMP PDU Error Index*)
    sRecObjectID        :T_MAXSTRING; (* Object Identifier *)
    arrRecObjectID      :ARRAY[0..128] OF UDINT;
    nErrID              :UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

bEnabled: Der Ausgang wird gesetzt, wenn ein Socket geöffnet wird.

bReceived: Der Ausgang wird gesetzt, wenn ein Get Befehl eingegangen ist.

iRecSNMPVersion: Angabe der SNMP Version. SNMPv1 = 0.

sRecCommunity: Angabe des Community Strings.

iRecPDUType: Angabe des PDU Types.

arrRecPDURequestID: Array mit den PDU Request IDs.

iRecPDUError: Ausgabe der PDU ErrorID.

iRecPDUErrorIndex: Ausgabe des Fehler Index.

sRecObjectID: String mit der ObejctID.

arrRecObjectID: Der Ausgang wird gesetzt, wenn ein Socket geöffnet wird.

nErrID: Liefert bei einem bError-Ausgang die TwinCAT TCP/IP Connection Server error [▶ 88]Fehlernummer zurück.

Der Funktionsblock wurde mit folgender Software getestet:

SNMP Trap Watcher (BTT Software)

Wireshark 1.2.5

iReasoning MIB Browser Personal Edition 7.0

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.3.3 F_GetVersionTcSNMP

Mit dieser Funktion können Versionsinformationen der SPS-Bibliothek ausgelesen werden.

FUNCTION F_GetVersionTcSNMP : UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

nVersionElement : Versionselement, das gelesen werden soll. Mögliche Parameter:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 oder höher	PC or CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.3.4 SNMP_ST_VariableBinding

Struktur von SNMP Variable Bindings

```
TYPE ST_SNMP_VariableBinding :
    STRUCT
        iType       :INT;
        iLength     :INT;
        pArrValue   :POINTER TO ARRAY[1..10000] OF BYTE;
        sOID        :STRING(300) ;
    END_STRUCT
END_TYPE
```

iType: Der SNMP Datentyp definiert in E_SNMP_DataTypes.

iLength: Die Elementeanzahl von pArrValue.

pArrValue: Zeiger auf das aktuelle Arrayfeld.

sOID: String der den numerischen Wert des "variable bindings" beinhaltet.

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcSnmp.Lib (TcpIp.Lib; Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.3.5 E_SNMP_GenericTrapNumber

```
TYPE E_SNMP_GenericTrapNumber :
(
    E_SNMP_ColdStart:= 16#00,
    E_SNMP_WarmStart:= 16#01,
    E_SNMP_LinkDown:=16#02,
    E_SNMP_LinkUp:= 16#03,
    E_SNMP_AuthentificationFailure:= 16#04,
    E_SNMP_EgpNeighborLoss:= 16#05,
    E_SNMP_EnterpriseSpecific:= 16#06
);
END_TYPE
```

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.3.6 E_SNMP_DataTypes

```
TYPE E_SNMP_DataTypes :
(
    E_SNMP_INTEGER:= 16#02,
    E_SNMP_OCTETSTRING:= 16#04,
    E_SNMP_OBJECTID:=16#06,
    E_SNMP_SEQUENCE := 16#30,
    E_SNMP_IPADDRESS:= 16#40,
    E_SNMP_COUNTER32:= 16#41,
    E_SNMP_GAUGE32:= 16#42,
    E_SNMP_TIMETICKS:= 16#43,
    E_SNMP_TRAPTYPE:= 16#A4
);
END_TYPE
```

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

7.3.7 Globale Variablen

VAR_GLOBAL CONSTANT

```
VAR_GLOBAL CONSTANT
    iMAX_TRAPBUF_SIZE :USINT := 255;
END_VAR
```

iMAX_TRAPBUF_SIZE: Maximale Anzahl an Elementen für pArrVarBinding (POINTER TO ARRAY[1..iMAX_TRAPBUF_SIZE] OF ST_SNMP_VariableBinding;)

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcpIp.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

8 Beispiele

8.1 Tcplp.lib

8.1.1 TCP Beispiel

Das folgende Beispiel zeigt eine Implementierung eines "Echo"-Clients/-Servers. Der Local-Client soll lediglich einen Test-String zum Remote-Server in bestimmten Abständen (z.B. jede Sekunde) senden. Der Remote-Server soll darauf sofort den gleichen String unverändert an den Client zurücksenden.

Der Client soll als Funktionsbaustein implementiert werden, von dem mehrere Instanzen angelegt werden können. Zudem soll der Server mit mehreren Clients kommunizieren können.

Von dem Server können auch mehrere Instanzen angelegt werden. Jede Server-Instanz wird dann über eine andere Portnummer angesprochen. Die Serverimplementierung ist schwieriger, wenn der Server mit mehr als nur einem Client kommunizieren soll. Außerdem wird eine Implementierung eines passenden Clients in .NET vorgestellt. Das Beispiel kann als Ansatz benutzt werden um eigene, komplexere Implementierungen realisieren zu können.

Systemvoraussetzungen

- TwinCAT v2.8 oder höher. Level: Mindestens TwinCAT PLC.
- Installierter TwinCAT TCP/IP Connection Server. Wenn Sie für den Test zwei PCs benutzen, dann sollte der TwinCAT TCP/IP Connection Server auf beiden PCs installiert werden. Auf einem CX-Sytem muss das entsprechende CAB-File installiert werden.

Projektsourcen

- https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383886219.zip
- https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383887627.zip
- https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383889035.zip

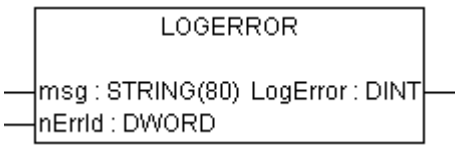
Projektbeschreibung

- FB_LocalClient-Funktionsbaustein [▶ 55]
- FB_LocalServer-Funktionsbaustein [▶ 60]
- Test der Funktionsbausteine [▶ 53]
- .NET Client-Projekt [▶ 66]

Hilfsfunktionen im Beispielprojekt

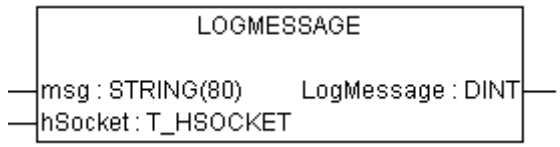
Im Beispiel werden einige Funktionen, Konstanten und Funktionsbausteine benutzt, die im Folgenden kurz beschrieben werden müssen:

```
FUNCTION LogError : DINT
```



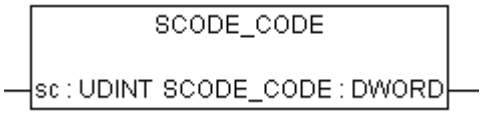
Die Funktion schreibt eine Meldung mit dem Fehlercode in das Logbuch des Betriebssystems (Event Viewer). Die globale Variable *bLogDebugMessages* muss vorher auf TRUE gesetzt werden.

```
FUNCTION LogMessage : DINT
```



Die Funktion schreibt eine Meldung in das Logbuch des Betriebssystems (Event Viewer), wenn ein neuer Socket geöffnet oder geschlossen wurde. Die globale Variable *bLogDebugMessages* muss vorher auf TRUE gesetzt werden.

```
FUNCTION SCODE_CODE : DWORD
```



Die Funktion maskiert die niederwertigsten 16Bits eines Win32 Fehlercodes aus und liefert diese zurück.

Globale Konstanten/Variablen

Name	Default value	Description
bLogDebugMessages	TRUE	Aktiviert/Deaktiviert das Schreiben der Meldungen in das Logbuch des Betriebssystems;
MAX_CLIENT_CONNECTIONS	5	Maximale Anzahl der Remote-Clients, die eine Verbindung zum Server gleichzeitig aufbauen können;
MAX_PLCPRJ_RXBUFFER_SIZE	1000	Maximale Länge des internen Empfangspuffers;
PLCPRJ_RECONNECT_TIME	T#3s	SERVER: Nach Ablauf dieser Zeit versucht der Local-Server den Listener-Socket neu zu öffnen; CLIENT: Nach Ablauf dieser Zeit versucht der Local-Client die Verbindung zum Remote-Server wiederherzustellen;
PLCPRJ_SEND_CYCLE_TIME	T#1s	In diesen Abständen wird der Test-String vom Local-Client zum Remote-Server zyklisch gesendet;
PLCPRJ_RECEIVE_POLLING_TIME	T#1s	SERVER und CLIENT: In diesem Zyklus werden die Empfangsdaten gelesen (gepollt);
PLCPRJ_RECEIVE_TIMEOUT	T#10s (CLIENT) T#50s (SERVER)	SERVER: Nach dieser Zeit bricht der Local-Server den Empfang ab, wenn in dieser Zeit keine Datenbytes empfangen werden konnten; CLIENT: Nach dieser Zeit bricht der Local-Client den Empfang ab, wenn in dieser Zeit keine Datenbytes empfangen werden konnten;
PLCPRJ_ACCEPT_POLLING_TIME	T#1s	In diesen Zeitabständen versucht der Locale-Server die Verbindungsanforderungen des Remote-Clients anzunehmen (akzeptieren);
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	Interner Beispielprojekt-Fehlercode: Es wurden zu viele Zeichen ohne Nullterminierung empfangen;

Name	Default value	Description
PLCPRJ_ERROR_RECEIVE_TIMEOUT	16#8102	Interner Beispielprojekt-Fehlercode: Es konnten keine neue Daten innerhalb der Timeoutzeit (PLCPRJ_RECEIVE_TIMEOUT) empfangen werden.

8.1.1.1 Test der Client- und Server-Funktionsbausteine

1. Öffnen Sie das https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383886219.zip auf dem lokalen PC. Die IP-Adresse des Servers muss an Ihr Remote-System angepasst werden (Initialisierungswerte der *sRemoteHost*-Variablen). Laden Sie das Projekt in das SPS-Laufzeitsystem auf dem lokalen PC. Starten Sie die SPS.

```
PROGRAM MAIN
VAR
    fbClient1         : FB_LocalClient := ( sRemoteHost:= '172.16.11.83' (* IP address of remote
server! *), nRemotePort:= 200 );
    fbClient2         : FB_LocalClient := ( sRemoteHost:= '172.16.11.83', nRemotePort:= 200 );
    fbClient3         : FB_LocalClient := ( sRemoteHost:= '172.16.11.83', nRemotePort:= 200 );
    fbClient4         : FB_LocalClient := ( sRemoteHost:= '172.16.11.83', nRemotePort:= 200 );
    fbClient5         : FB_LocalClient := ( sRemoteHost:= '172.16.11.83', nRemotePort:= 200 );
    bEnableClient1    : BOOL := TRUE;
    bEnableClient2    : BOOL := FALSE;
    bEnableClient3    : BOOL := FALSE;
    bEnableClient4    : BOOL := FALSE;
    bEnableClient5    : BOOL := FALSE;
    fbSocketCloseAll  : FB_SocketCloseAll := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    bCloseAll         : BOOL := TRUE;
    nCount            : UDINT;
END_VAR
```

```
IF bCloseAll THEN (*On PLC reset or program download close all old connections *)
    bCloseAll := FALSE;
    fbSocketCloseAll( bExecute:= TRUE );
ELSE
    fbSocketCloseAll( bExecute:= FALSE );
END_IF

IF NOT fbSocketCloseAll.bBusy THEN
    nCount := nCount + 1;

    fbClient1( bEnable := bEnableClient1, sToServer := CONCAT( 'CLIENT1-', UDINT_TO_STRING( nCount )
 ) );
    fbClient2( bEnable := bEnableClient2, sToServer := CONCAT( 'CLIENT2-', UDINT_TO_STRING( nCount )
 ) );
    fbClient3( bEnable := bEnableClient3, sToServer := CONCAT( 'CLIENT3-', UDINT_TO_STRING( nCount )
 ) );
    fbClient4( bEnable := bEnableClient4  );
    fbClient5( bEnable := bEnableClient5  );
END_IF
```

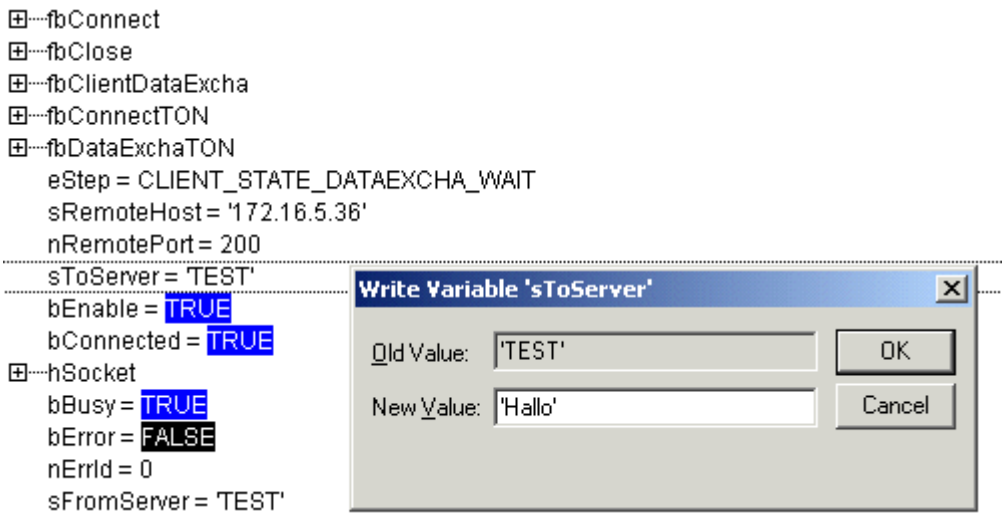
Beim Setzen einer der *bEnableCientX*-Variablen können bis zu 5 Clientinstanzen aktiviert werden. Jeder Client sendet ca. jede Sekunde einen String zum Server (default: 'TEST'). Der gleiche String wird vom Server zum Client zurückgesendet (Echo-Server). Für den Test wird bei den ersten drei Instanzen ein String mit einem Zählerwert automatisch generiert. Der erste Client wird beim Programmstart automatisch aktiviert.

2. Öffnen Sie das https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383887627.zip auch auf dem lokalen PC. Als Zielsystem wählen Sie ein SPS-Laufzeitsystem auf einem Remote PC aus. Laden Sie das SPS-Programm in das Laufzeitsystem. Starten Sie die SPS.

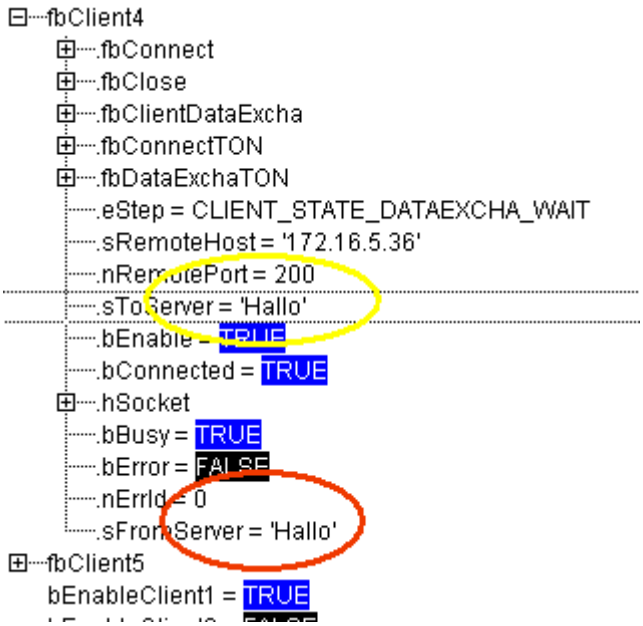
```
PROGRAM MAIN
VAR
    fbServer        : FB_LocalServer := ( sLocalHost := '172.16.11.83' (*own IP address!
*), nLocalPort := 200 );
    bEnableServer   : BOOL := TRUE;
    fbSocketCloseAll    : FB_SocketCloseAll :=  ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT )
;
    bCloseAll           : BOOL := TRUE;
END_VAR
```

```
F bCloseAll THEN (*On PLC reset or program download close all old connections *)
    bCloseAll := FALSE;
    fbSocketCloseAll( bExecute:= TRUE );
ELSE
    fbSocketCloseAll( bExecute:= FALSE );
END_IF

IF NOT fbSocketCloseAll.bBusy THEN
    fbServer( bEnable := bEnableServer );
END_IF
```

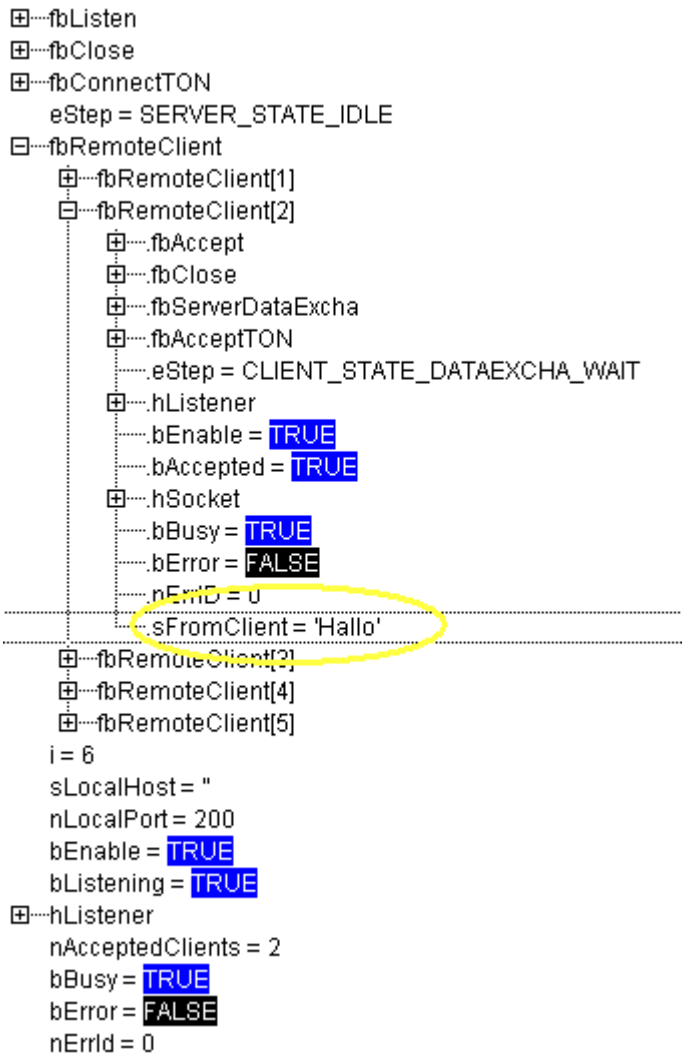
3. Setzen Sie die *bEnableCilent4*-Variable im Client-Projekt auf TRUE. Der zweite Client versucht dann eine Verbindung zum Server aufzubauen. Beim Erfolg wird der 'TEST'-String zyklisch gesendet. Öffnen Sie jetzt die *fbClient4*-Instanz des FB_LocalClient-Funktionsbausteins.



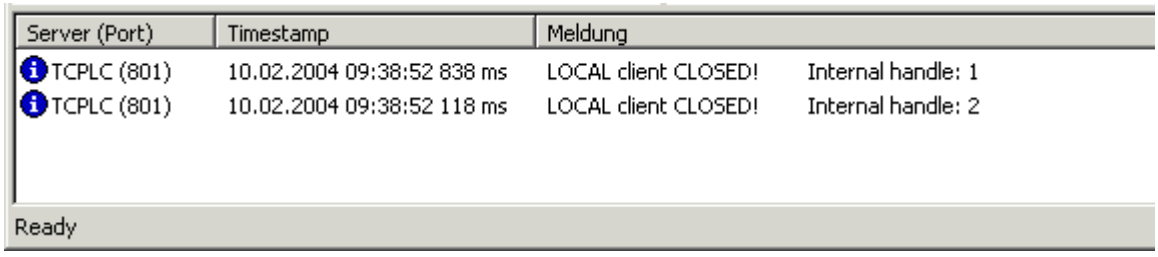
Öffnen Sie den Dialog zum Schreiben der *sToString*-Variablen mit einem Doppelklick. Ändern Sie den Wert der Stringvariablen z.B. auf 'Hallo'. Schließen Sie den Dialog mit OK. Schreiben Sie den neuen Wert in die SPS (CTRL+F7). Kurz danach kann auch der vom Server zurückgelesene Wert online gesehen werden.



4. In dem Server-Projekt öffnen Sie jetzt die *fbServer*-Instanz des FB_LocalServer-Funktionsbausteins. Unser String: 'Hallo' kann in den Online-Daten des Servers gesehen werden.



5. In den Server- und Client-Beispielen werden Meldungen beim Auf-/Aubbauen der Verbindung und bei einem Fehler in das Logbuch des Betriebssystem geschrieben. Es erleichtert die mögliche Fehlersuche. Diese Meldungen können im Logger Output des TwinCAT System Managers angezeigt werden. Starten Sie den TwinCAT System Manager auf dem lokalen System und aktivieren Sie den Logger Output. Deaktivieren Sie jetzt die beiden Clients (*bEnableClient1* und *bEnableClient4* auf FALSE).

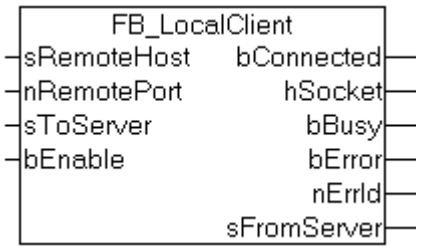


Es können auch die Meldungen des Servers auf dem lokalen PC angezeigt werden. Dazu müssen Sie eine zweite Instanz des TwinCAT System Managers auf dem lokalen PC öffnen und als Zielsystem im TwinCAT System Manager den Remote-PC auswählen.

8.1.1.2 PLC Client

8.1.1.2.1 FB_LocalClient

Hier können Sie die kompletten Sourcen zum Client-Projekt entpacken: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383886219.zip;



Beim gesetzten *bEnable*-Eingang wird immer wieder versucht, nach Ablauf der PLCPRJ_RECONNECT_TIME, die Verbindung zum Remote-Server herzustellen. Der Remote-Server wird über die *sRemoteHost*-IP Adresse und die *nRemotePort*-IP Portadresse identifiziert. Der Datenaustausch zum Server wurde in einem separaten Funktionsbaustein FB_ClientDataExcha [▶ 58] gekapselt. Der Datenaustausch wird immer zyklisch nach Ablauf der PLCPRJ_SEND_CYCLE_TIME durchgeführt. Dabei wird die *sToServer*-Stringvariable zum Server gesendet und der vom Server zurückgesendete String am Ausgang *sFormServer* zurückgeliefert. Eine andere Implementierung, in der der Remote-Server bei Bedarf angesprochen wird, ist aber ebenfalls möglich. Beim Fehler wird die vorhandene Verbindung geschlossen und eine neue aufgebaut.

Interface

```
FUNCTION_BLOCK FB_LocalClient
VAR_INPUT
    sRemoteHost         : STRING(15):= '127.0.0.1';(* IP adress of remote server *)
    nRemotePort         : UDINT := 0;
    sToServer           : T_MaxString:= 'TEST';
    bEnable             : BOOL;
END_VAR
VAR_OUTPUT
    bConnected          : BOOL;
    hSocket             : T_HSOCKET;
    bBusy               : BOOL;
    bError              : BOOL;
    nErrId              : UDINT;
    sFromServer         : T_MaxString := '';
END_VAR
VAR
    fbConnect           : FB_SocketConnect :=  ( sSrvNetId := '' );
    fbClose             : FB_SocketClose := ( sSrvNetId := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbClientDataExcha   : FB_ClientDataExcha;

    fbConnectTON        : TON;
    fbDataExchaTON      : TON;
    eStep               : E_ClientSteps;
END_VAR
```

Implementierung

```
CASE eStep OF
    CLIENT_STATE_IDLE:
        IF bEnable XOR bConnected THEN
            bBusy := TRUE;
            bError := FALSE;
            nErrid := 0;
            sFromServer := '';
            IF bEnable THEN
                fbConnectTON( IN := FALSE );
                eStep := CLIENT_STATE_CONNECT_START;
            ELSE
                eStep := CLIENT_STATE_CLOSE_START;
            END_IF
        ELSIF bConnected THEN
            fbDataExchaTON( IN := FALSE );
            eStep := CLIENT_STATE_DATAEXCHA_START;
        ELSE
            bBusy := FALSE;
        END_IF

    CLIENT_STATE_CONNECT_START:
        fbConnectTON( IN := TRUE, PT := PLCPRJ_RECONNECT_TIME );
        IF fbConnectTON.Q THEN
            fbConnectTON( IN := FALSE );
            fbConnect(  bExecute := FALSE  );
            fbConnect(sRemoteHost   := sRemoteHost,
```

```
                nRemotePort     := nRemotePort,
                bExecute        := TRUE );
            eStep := CLIENT_STATE_CONNECT_WAIT;
        END_IF

    CLIENT_STATE_CONNECT_WAIT:
        fbConnect( bExecute := FALSE );
        IF NOT fbConnect.bBusy THEN
            IF NOT fbConnect.bError THEN
                bConnected      := TRUE;
                hSocket         := fbConnect.hSocket;
                eStep       := CLIENT_STATE_IDLE;
                LogMessage( 'LOCAL client CONNECTED!', hSocket );
            ELSE
                LogError( 'FB_SocketConnect', fbConnect.nErrId );
                nErrId := fbConnect.nErrId;
                eStep := CLIENT_STATE_ERROR;
            END_IF
        END_IF

    CLIENT_STATE_DATAEXCHA_START:
        fbDataExchaTON( IN := TRUE, PT := PLCPRJ_SEND_CYCLE_TIME );
        IF fbDataExchaTON.Q THEN
            fbDataExchaTON( IN := FALSE );
            fbClientDataExcha( bExecute := FALSE );
            fbClientDataExcha(      hSocket         := hSocket,
                                sToServer       := sToServer,
                                bExecute    := TRUE );
            eStep := CLIENT_STATE_DATAEXCHA_WAIT;
        END_IF

    CLIENT_STATE_DATAEXCHA_WAIT:
        fbClientDataExcha( bExecute := FALSE );
        IF NOT fbClientDataExcha.bBusy THEN
            IF NOT fbClientDataExcha.bError THEN
                sFromServer     := fbClientDataExcha.sFromServer;
                eStep       := CLIENT_STATE_IDLE;
            ELSE
                (* possible errors are logged inside of fbClientDataExcha function block *)
                nErrId := fbClientDataExcha.nErrId;
                eStep :=CLIENT_STATE_ERROR;
            END_IF
        END_IF

    CLIENT_STATE_CLOSE_START:
        fbClose( bExecute := FALSE );
        fbClose(    hSocket:= hSocket,
            bExecute:= TRUE );
        eStep := CLIENT_STATE_CLOSE_WAIT;

    CLIENT_STATE_CLOSE_WAIT:
        fbClose( bExecute := FALSE );
        IF NOT fbClose.bBusy THEN
            LogMessage( 'LOCAL client CLOSED!', hSocket );
            bConnected := FALSE;
            MEMSET( ADR(hSocket), 0, SIZEOF(hSocket));
            IF fbClose.bError THEN
                LogError( 'FB_SocketClose (local client)', fbClose.nErrId );
                nErrId := fbClose.nErrId;
                eStep := CLIENT_STATE_ERROR;
            ELSE
                bBusy := FALSE;
                bError := FALSE;
                nErrId := 0;
                eStep := CLIENT_STATE_IDLE;
            END_IF
        END_IF

    CLIENT_STATE_ERROR: (* Error step *)
        bError := TRUE;
        IF bConnected THEN
            eStep := CLIENT_STATE_CLOSE_START;
        ELSE
            bBusy := FALSE;
            eStep := CLIENT_STATE_IDLE;
        END_IF
END_CASE
```

Sehen Sie dazu auch

- FB_SocketConnect [▶ 18]
- FB_SocketClose [▶ 19]
- FB_ClientDataExcha [▶ 58]

8.1.1.2.2 FB_ClientDataExcha

Bei einer steigenden Flanke am *bExecute*-Eingang wird ein Nullterminierter-String zum Remote-Server gesendet und ein vom Remote-Server zurück gelieferter String zurück gelesen. Der Funktionsbaustein versucht die Daten so lange zu lesen, bis eine Nullterminierung in dem empfangenen String erkannt wurde. Der Empfang wird bei einem Fehler, und wenn innerhalb der Timeoutzeit: PLCPRJ_RECEIVE_TIMEOUT keine neuen Daten empfangen wurden, abgebrochen. Der nächste Lesevorgang wird nach einer Verzögerungszeit ausgeführt, wenn beim letzten Lesevorgang keine neuen Daten gelesen werden konnten. Die Systemauslastung verringert sich dadurch.

Interface

```
FUNCTION_BLOCK FB_ClientDataExcha
VAR_INPUT
    hSocket         : T_HSOCKET;
    sToServer       : T_MaxString;
    bExecute        : BOOL;
END_VAR
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    sFromServer : T_MaxString;
END_VAR
VAR
    fbSocketSend    : FB_SocketSend;
    fbSocketReceive : FB_SocketReceive;
    fbReceiveTON    : TON;
    fbDisconnectTON : TON;
    RisingEdge      : R_TRIG;
    eStep           : E_DataExchaSteps;
    cbReceived, startPos, endPos, idx       : UDINT;
    cbFrame         : UDINT;
    rxBuffer        : ARRAY[0..MAX_PLCPRJ_RXBUFFER_SIZE] OF BYTE;
END_VAR
```

Implementierung

```
RisingEdge( CLK := bExecute );
CASE eStep OF
    DATAEXCHA_STATE_IDLE:
        IF RisingEdge.Q THEN
            bBusy := TRUE;
            bError := FALSE;
            nErrid := 0;
            cbReceived := 0;
            fbReceiveTON( IN := FALSE, PT := T#0s ); (* don't wait, read the first answer data immed
iately *)
            fbDisconnectTON( IN := FALSE, PT := T#0s );(* disable timeout check first *)
            eStep := DATAEXCHA_STATE_SEND_START;
        END_IF

    DATAEXCHA_STATE_SEND_START:
        fbSocketSend( bExecute := FALSE );
        fbSocketSend(   hSocket := hSocket,
                pSrc    := ADR( sToServer ),
                cbLen   := LEN( sToServer ) + 1,(* string length inclusive zero delimiter *)
                bExecute:= TRUE );
        eStep := DATAEXCHA_STATE_SEND_WAIT;

    DATAEXCHA_STATE_SEND_WAIT:
```

```
        fbSocketSend( bExecute := FALSE );
        IF NOT fbSocketSend.bBusy THEN
            IF NOT fbSocketSend.bError THEN
                eStep := DATAEXCHA_STATE_RECEIVE_START;
            ELSE
                LogError( 'FB_SocketSend (local client)', fbSocketSend.nErrId );
                nErrId := fbSocketSend.nErrId;
                eStep := DATAEXCHA_STATE_ERROR;
            END_IF
        END_IF

    DATAEXCHA_STATE_RECEIVE_START:
        fbDisconnectTON(  );
        fbReceiveTON( IN := TRUE );
        IF fbReceiveTON.Q THEN
            fbReceiveTON( IN := FALSE );
            fbSocketReceive( bExecute := FALSE );
            fbSocketReceive(  hSocket:= hSocket,
                    pDest:= ADR( rxBuffer ) + cbReceived,
                    cbLen:= SIZEOF( rxBuffer ) - cbReceived,
                    bExecute:= TRUE );
            eStep := DATAEXCHA_STATE_RECEIVE_WAIT;
        END_IF

    DATAEXCHA_STATE_RECEIVE_WAIT:
        fbSocketReceive( bExecute := FALSE );
        IF NOT fbSocketReceive.bBusy THEN
            IF NOT fbSocketReceive.bError THEN
                IF (fbSocketReceive.nRecBytes > 0) THEN(* bytes received *)
                    startPos        := cbReceived;(* rxBuffer array index of first data byte *)
                    endPos          := cbReceived + fbSocketReceive.nRecBytes  - 1;
(* rxBuffer array index of last data byte *)
                    cbReceived      := cbReceived + fbSocketReceive.nRecBytes;
(* calculate the number of received data bytes *)
                    cbFrame     := 0;(* reset frame length *)
                    IF cbReceived < SIZEOF( sFromServer ) THEN(* no overflow *)
                        fbReceiveTON( PT := T#0s ); (* bytes received => increase the read (polling)
 speed *)
                        fbDisconnectTON( IN := FALSE );(* bytes received => disable timeout check *)
                        (* search for string end delimiter *)
                        FOR idx := startPos TO endPos BY 1 DO
                            IF rxBuffer[idx] = 0 THEN(* string end delimiter found *)
                                cbFrame := idx + 1;
(* calculate the length of the received string (inclusive the end delimiter) *)
                                MEMCPY(  ADR( sFromServer ), ADR( rxBuffer ), cbFrame );
(* copy the received string to the output variable (inclusive the end delimiter) *)
                                MEMMOVE( ADR( rxBuffer ), ADR( rxBuffer[cbFrame] ),  cbReceived -
 cbFrame  );(* move the reamaining data bytes *)
                                cbReceived := cbReceived - cbFrame;
(* recalculate the remaining data byte length *)
                                bBusy := FALSE;
                                eStep := DATAEXCHA_STATE_IDLE;
                                EXIT;
                            END_IF
                        END_FOR
                    ELSE(* there is no more free read buffer space => the answer string should be te
rminated *)
                        LogError( 'FB_SocketReceive (local client)',  PLCPRJ_ERROR_RECEIVE_BUFFER_OV
ERFLOW );
                        nErrId := PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW;(* buffer overflow !*)
                        eStep := DATAEXCHA_STATE_ERROR;
                    END_IF
                ELSE(* no bytes received *)
                    fbReceiveTON( PT := PLCPRJ_RECEIVE_POLLING_TIME );
(* no bytes received => decrease the read (polling) speed *)
                    fbDisconnectTON( IN := TRUE, PT := PLCPRJ_RECEIVE_TIMEOUT );
(* no bytes received => enable timeout check*)
                    IF fbDisconnectTON.Q THEN (* timeout error*)
                        fbDisconnectTON( IN := FALSE );
                        LogError( 'FB_SocketReceive (local client)',  PLCPRJ_ERROR_RECEIVE_TIMEOUT )
;
                        nErrID := PLCPRJ_ERROR_RECEIVE_TIMEOUT;
                        eStep := DATAEXCHA_STATE_ERROR;
                    ELSE(* repeat reading *)
                        eStep := DATAEXCHA_STATE_RECEIVE_START; (* repeat reading *)
                    END_IF
                END_IF
            ELSE(* receive error *)
                LogError( 'FB_SocketReceive (local client)',  fbSocketReceive.nErrId );
                nErrId := fbSocketReceive.nErrId;
```

```
                eStep := DATAEXCHA_STATE_ERROR;
            END_IF
        END_IF

    DATAEXCHA_STATE_ERROR:(* error step *)
        bBusy := FALSE;
        bError := TRUE;
        cbReceived := 0;
        eStep := DATAEXCHA_STATE_IDLE;
END_CASE
```

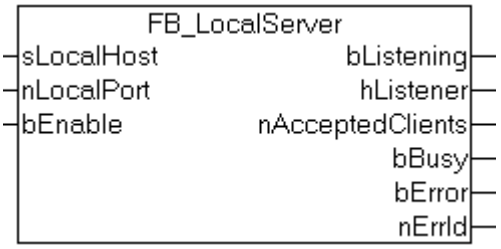
Sehen Sie dazu auch

- FB_SocketSend [▶ 24]
- FB_SocketReceive [▶ 25]

8.1.1.3 PLC Server

8.1.1.3.1 FB_LocalServer

Hier können Sie die kompletten Sourcen zum Server-Projekt entpacken: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383887627.zip;



Dem Server muss zuerst eine eindeutige *sLocalHost*-IP Adresse und eine *nLocaPort*-IP Portnummer zugewiesen werden. Beim gesetzten *bEnable*-Eingang versucht der Local-Server immer wieder nach Ablauf der PLCPRJ_RECONNECT_TIME den Listener-Socket zu öffnen. Im Regelfall kann der Listener-Socket beim ersten Versuch geöffnet werden, wenn sich der TwinCAT TCP/IP Connection Server auf dem lokalen PC befindet. Die Funktionalität eines Remote-Clients wurde in dem Funktionsbaustein FB_RemoteClient [▶ 62] gekapselt. Die Instanzen der Remote-Clients werden aktiviert, nachdem der Listener-Socket geöffnet werden konnte. Jede Instanz vom FB_RemoteClient entspricht einem Remote-Client mit dem der Local-Server gleichzeitig kommunizieren kann. Die maximale Anzahl der mit dem Server kommunizierenden Remote-Clients kann durch den Wert der MAX_CLIENT_CONNECTIONS Konstanten verändert werden. Bei einem Fehler werden zuerst alle Remote-Client Verbindungen und dann der Listener-Socket geschlossen. Der *nAcceptedClients*-Ausgang gibt Auskunft über die aktuelle Anzahl der verbundenen Clients.

Interface

```
FUNCTION_BLOCK FB_LocalServer
VAR_INPUT
    sLocalHost              : STRING(15) := '127.0.0.1';(* own IP address! *)
    nLocalPort              : UDINT := 0;
    bEnable                 : BOOL;
END_VAR
VAR_OUTPUT
    bListening              : BOOL;
    hListener               : T_HSOCKET;
    nAcceptedClients            : UDINT;
    bBusy               : BOOL;
    bError              : BOOL;
    nErrId              : UDINT;
END_VAR
VAR
    fbListen                : FB_SocketListen  := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT
 );
    fbClose                 : FB_SocketClose  := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT
);
    fbConnectTON            : TON := ( PT := PLCPRJ_RECONNECT_TIME );
    eStep               : E_ServerSteps;
```

```
    fbRemoteClient          : ARRAY[1..MAX_CLIENT_CONNECTIONS ] OF FB_RemoteClient;
    i                   : UDINT;
END_VAR
```

Implementierung

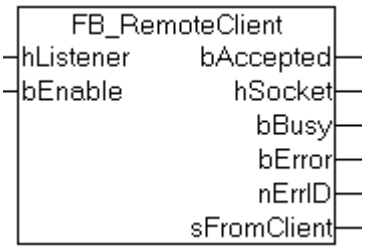
```
CASE eStep OF

    SERVER_STATE_IDLE:
        IF bEnable XOR bListening THEN
            bBusy := TRUE;
            bError := FALSE;
            nErrId := 0;
            IF bEnable THEN
                fbConnectTON( IN := FALSE );
                eStep := SERVER_STATE_LISTENER_OPEN_START;
            ELSE
                eStep := SERVER_STATE_REMOTE_CLIENTS_CLOSE;
            END_IF
        ELSIF bListening THEN
            eStep := SERVER_STATE_REMOTE_CLIENTS_COMM;
        END_IF

    SERVER_STATE_LISTENER_OPEN_START:
        fbConnectTON( IN := TRUE, PT := PLCPRJ_RECONNECT_TIME );
        IF fbConnectTON.Q THEN
            fbConnectTON( IN := FALSE );
            fbListen(  bExecute := FALSE  );
            fbListen(       sLocalHost      := sLocalHost,
                nLocalPort      := nLocalPort,
                bExecute    := TRUE );
            eStep := SERVER_STATE_LISTENER_OPEN_WAIT;
        END_IF

    SERVER_STATE_LISTENER_OPEN_WAIT:
        fbListen( bExecute := FALSE );
        IF NOT fbListen.bBusy THEN
            IF NOT fbListen.bError THEN
                bListening      := TRUE;
                hListener       := fbListen.hListener;
                eStep       := SERVER_STATE_IDLE;
                LogMessage( 'LISTENER socket OPENED!', hListener );
            ELSE
                LogError( 'FB_SocketListen',  fbListen.nErrId );
                nErrId := fbListen.nErrId;
                eStep := SERVER_STATE_ERROR;
            END_IF
        END_IF

    SERVER_STATE_REMOTE_CLIENTS_COMM:
        eStep := SERVER_STATE_IDLE;
        nAcceptedClients := 0;
        FOR i:= 1 TO MAX_CLIENT_CONNECTIONS DO
            fbRemoteClient[ i ]( hListener := hListener, bEnable := TRUE );
            IF NOT  fbRemoteClient[ i ].bBusy AND fbRemoteClient[ i ].bError THEN (*FB_SocketAccept
returned error!*)
                eStep := SERVER_STATE_REMOTE_CLIENTS_CLOSE;
                EXIT;
            END_IF
            (* count the number of connected remote clients *)
            IF fbRemoteClient[ i ].bAccepted THEN
                nAcceptedClients := nAcceptedClients + 1;
            END_IF
        END_FOR

    SERVER_STATE_REMOTE_CLIENTS_CLOSE:
        nAcceptedClients := 0;
        eStep := SERVER_STATE_LISTENER_CLOSE_START; (* close listener socket too *)
        FOR i:= 1 TO MAX_CLIENT_CONNECTIONS DO
            fbRemoteClient[ i ]( bEnable := FALSE );(* close all remote client (accepted) sockets *)
            (* check if all remote client sockets are closed *)
            IF fbRemoteClient[ i ].bAccepted THEN
                eStep := SERVER_STATE_REMOTE_CLIENTS_CLOSE; (* stay here and close all remote client
s first *)
                nAcceptedClients := nAcceptedClients + 1;
            END_IF
        END_FOR

    SERVER_STATE_LISTENER_CLOSE_START:
        fbClose( bExecute := FALSE );
```

```
        fbClose(    hSocket := hListener,
            bExecute:= TRUE );
        eStep := SERVER_STATE_LISTENER_CLOSE_WAIT;

    SERVER_STATE_LISTENER_CLOSE_WAIT:
        fbClose( bExecute := FALSE );
        IF NOT fbClose.bBusy THEN
            LogMessage( 'LISTENER socket CLOSED!', hListener );
            bListening := FALSE;
            MEMSET( ADR(hListener), 0, SIZEOF(hListener));
            IF fbClose.bError THEN
                LogError( 'FB_SocketClose (listener)',  fbClose.nErrId );
                nErrId := fbClose.nErrId;
                eStep := SERVER_STATE_ERROR;
            ELSE
                bBusy := FALSE;
                bError := FALSE;
                nErrId := 0;
                eStep := SERVER_STATE_IDLE;
            END_IF
        END_IF

    SERVER_STATE_ERROR:
        bError := TRUE;
        IF bListening THEN
            eStep := SERVER_STATE_REMOTE_CLIENTS_CLOSE;
        ELSE
            bBusy := FALSE;
            eStep := SERVER_STATE_IDLE;
        END_IF
END_CASE
```

Sehen Sie dazu auch

- FB_SocketListen [▶ 22]
- FB_SocketClose [▶ 19]
- FB_RemoteClient [▶ 62]

8.1.1.3.2 FB_RemoteClient



Beim gesetzten *bEnable*-Eingang wird nach Ablauf der PLCPRJ_ACCEPT_POOLING_TIME versucht, die Verbindungsanforderung eines Remote-Clients anzunehmen (zu akzeptieren). Der Datenaustausch zum Remote-Client wurde in einem separaten Funktionsbaustein FB_ServerDataExcha [▶ 64] gekapselt. Nach einem erfolgreichen Aufbau der Verbindung wird die Instanz vom FB_ServerDataExcha Funktionsbaustein aktiviert. Bei einem Fehler wird die angenommene Verbindung geschlossen und eine neue aufgebaut.

Interface

```
FUNCTION_BLOCK FB_RemoteClient
VAR_INPUT
    hListener               : T_HSOCKET;
    bEnable                 : BOOL;
END_VAR
VAR_OUTPUT
    bAccepted               : BOOL;
    hSocket                 : T_HSOCKET;
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    sFromClient             : T_MaxString;
END_VAR
VAR
```

```
    fbAccept                : FB_SocketAccept   := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOU
T );
    fbClose                 : FB_SocketClose  := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT
);
    fbServerDataExcha                : FB_ServerDataExcha;
    fbAcceptTON             : TON  := ( PT := PLCPRJ_ACCEPT_POLLING_TIME );
    eStep               : E_ClientSteps;
END_VAR
```

Implementierung

```
CASE eStep OF

    CLIENT_STATE_IDLE:
        IF bEnable XOR bAccepted THEN
            bBusy := TRUE;
            bError := FALSE;
            nErrId := 0;
            sFromClient := '';
            IF bEnable THEN
                fbAcceptTON( IN := FALSE );
                eStep := CLIENT_STATE_CONNECT_START;
            ELSE
                eStep := CLIENT_STATE_CLOSE_START;
            END_IF
        ELSIF bAccepted THEN
            eStep := CLIENT_STATE_DATAEXCHA_START;
        ELSE
            bBusy := FALSE;
        END_IF

    CLIENT_STATE_CONNECT_START:
        fbAcceptTON( IN := TRUE, PT := PLCPRJ_ACCEPT_POLLING_TIME );
        IF fbAcceptTON.Q THEN
            fbAcceptTON( IN := FALSE );
            fbAccept(  bExecute := FALSE  );
            fbAccept(       hListener := hListener,
                        bExecute:= TRUE );
            eStep := CLIENT_STATE_CONNECT_WAIT;
        END_IF

    CLIENT_STATE_CONNECT_WAIT:
        fbAccept( bExecute := FALSE );
        IF NOT fbAccept.bBusy THEN
            IF NOT fbAccept.bError THEN
                IF fbAccept.bAccepted THEN
                    bAccepted := TRUE;
                    hSocket := fbAccept.hSocket;
                    LogMessage( 'REMOTE client ACCEPTED!', hSocket );
                END_IF
                eStep := CLIENT_STATE_IDLE;
            ELSE
                LogError( 'FB_SocketAccept',  fbAccept.nErrId );
                nErrId := fbAccept.nErrId;
                eStep := CLIENT_STATE_ERROR;
            END_IF
        END_IF

    CLIENT_STATE_DATAEXCHA_START:
        fbServerDataExcha( bExecute := FALSE );
        fbServerDataExcha(      hSocket := hSocket,
                                bExecute := TRUE );
        eStep := CLIENT_STATE_DATAEXCHA_WAIT;

    CLIENT_STATE_DATAEXCHA_WAIT:
        fbServerDataExcha( bExecute := FALSE, sFromClient=>sFromClient );
        IF NOT fbServerDataExcha.bBusy THEN
            IF NOT fbServerDataExcha.bError THEN
                eStep := CLIENT_STATE_IDLE;
            ELSE
                (* possible errors are logged inside of fbServerDataExcha function block *)
                nErrId := fbServerDataExcha.nErrID;
                eStep := CLIENT_STATE_ERROR;
            END_IF
        END_IF
```

```
    CLIENT_STATE_CLOSE_START:
        fbClose( bExecute := FALSE );
        fbClose(    hSocket:= hSocket,
                bExecute:= TRUE );
        eStep := CLIENT_STATE_CLOSE_WAIT;

    CLIENT_STATE_CLOSE_WAIT:
        fbClose( bExecute := FALSE );
        IF NOT fbClose.bBusy THEN
            LogMessage( 'REMOTE client CLOSED!', hSocket );
            bAccepted := FALSE;
            MEMSET( ADR( hSocket ), 0, SIZEOF( hSocket ) );
            IF fbClose.bError THEN
                LogError( 'FB_SocketClose (remote client)', fbClose.nErrId );
                nErrId := fbClose.nErrId;
                eStep := CLIENT_STATE_ERROR;
            ELSE
                bBusy := FALSE;
                bError := FALSE;
                nErrId := 0;
                eStep := CLIENT_STATE_IDLE;
            END_IF
        END_IF

    CLIENT_STATE_ERROR:
        bError := TRUE;
        IF bAccepted THEN
            eStep := CLIENT_STATE_CLOSE_START;
        ELSE
            eStep := CLIENT_STATE_IDLE;
            bBusy := FALSE;
        END_IF

END_CASE
```

Sehen Sie dazu auch

- FB_SocketAccept [▶ 23]
- FB_SocketClose [▶ 19]
- FB_ServerDataExcha [▶ 64]

8.1.1.3.3 FB_ServerDataExcha

Abb. 1: FB_ServerDataExcha

Bei einer steigenden Flanke am *bExecute*-Eingang wird ein Nullterminierter-String vom Remote-Client gelesen, und wenn eine Nullterminierung erkannt wurde, an den Remote-Client zurückgesendet. Der Funktionsbaustein versucht die Daten so lange zu lesen, bis eine Nullterminierung in dem empfangenen String erkannt wurde. Der Empfang wird bei einem Fehler, und wenn innerhalb der Timeoutzeit: PLCPRJ_RECEIVE_TIMEOUT keine neuen Daten empfangen wurden, abgebrochen. Der nächste Lesevorgang wird nach einer Verzögerungszeit ausgeführt, wenn beim letzten Lesevorgang keine neuen Daten gelesen werden konnten. Die Systemauslastung verringert sich dadurch.

Interface

```
FUNCTION_BLOCK FB_ServerDataExcha
VAR_INPUT
    hSocket     : T_HSOCKET;
    bExecute        : BOOL;
END_VAR
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sFromClient     : T_MaxString;
```

```
END_VAR
VAR
    fbSocketReceive : FB_SocketReceive  := ( sSrvNetId := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbSocketSend    : FB_SocketSend := ( sSrvNetId := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    eStep       : E_DataExchaSteps;
    RisingEdge      : R_TRIG;
    fbReceiveTON    : TON;
    fbDisconnectTON     : TON;
    cbReceived, startPos, endPos, idx       : UDINT;
    cbFrame     : UDINT;
    rxBuffer        : ARRAY[0..MAX_PLCPRJ_RXBUFFER_SIZE] OF BYTE;
END_VAR
```

Implementierung

```
RisingEdge( CLK := bExecute );
CASE eStep OF

    DATAEXCHA_STATE_IDLE:
        IF RisingEdge.Q THEN
            bBusy   := TRUE;
            bError  := FALSE;
            nErrId  := 0;
            fbDisconnectTON( IN := FALSE, PT := T#0s );(* disable timeout check first *)
            fbReceiveTON( IN := FALSE, PT := T#0s ); (* receive first request immediately *)
            eStep   := DATAEXCHA_STATE_RECEIVE_START;
        END_IF

    DATAEXCHA_STATE_RECEIVE_START: (* Receive remote client data *)
        fbReceiveTON(  IN := TRUE );
        IF fbReceiveTON.Q THEN
            fbReceiveTON( IN := FALSE );
            fbSocketReceive( bExecute := FALSE );
            fbSocketReceive(    hSocket := hSocket,
                    pDest   := ADR( rxBuffer ) + cbReceived,
                    cbLen   := SIZEOF( rxBuffer ) - cbReceived,
                    bExecute    := TRUE );
            eStep := DATAEXCHA_STATE_RECEIVE_WAIT;
        END_IF

    DATAEXCHA_STATE_RECEIVE_WAIT:
        fbSocketReceive( bExecute := FALSE );
        IF NOT fbSocketReceive.bBusy THEN
            IF NOT fbSocketReceive.bError THEN
                IF (fbSocketReceive.nRecBytes > 0) THEN(* bytes received *)

                    startPos        := cbReceived;(* rxBuffer array index of first data byte *)
                    endPos          := cbReceived + fbSocketReceive.nRecBytes  - 1;
(* rxBuffer array index of last data byte *)
                    cbReceived      := cbReceived + fbSocketReceive.nRecBytes;
(* calculate the number of received data bytes *)
                    cbFrame     := 0;(* reset frame length *)

                    IF cbReceived < SIZEOF( sFromClient ) THEN(* no overflow *)
                        fbReceiveTON( IN := FALSE, PT := T#0s ); (* bytes received => increase the r
ead (polling) speed *)
                        fbDisconnectTON( IN := FALSE, PT := PLCPRJ_RECEIVE_TIMEOUT );
(* bytes received => disable timeout check *)

                        (* search for string end delimiter *)
                        FOR idx := startPos TO endPos BY 1 DO
                            IF rxBuffer[idx] = 0 THEN(* string end delimiter found *)
                                cbFrame := idx + 1;
(* calculate the length of the received string (inclusive the end delimiter) *)
                                MEMCPY(  ADR( sFromClient ), ADR( rxBuffer ), cbFrame );
(* copy the received string to the output variable (inclusive the end delimiter) *)
                                MEMMOVE( ADR( rxBuffer ), ADR( rxBuffer[cbFrame] ),  cbReceived -
 cbFrame  );(* move the reamaining data bytes *)
                                cbReceived := cbReceived - cbFrame;
(* recalculate the reamaining data byte length *)
                                eStep := DATAEXCHA_STATE_SEND_START;
                                EXIT;
                            END_IF
                        END_FOR

                    ELSE(* there is no more free read buffer space => the answer string should be te
rminated *)
                        LogError( 'FB_SocketReceive (remote client)',  PLCPRJ_ERROR_RECEIVE_BUFFER_O
VERFLOW );
                        nErrId := PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW;(* buffer overflow !*)
```

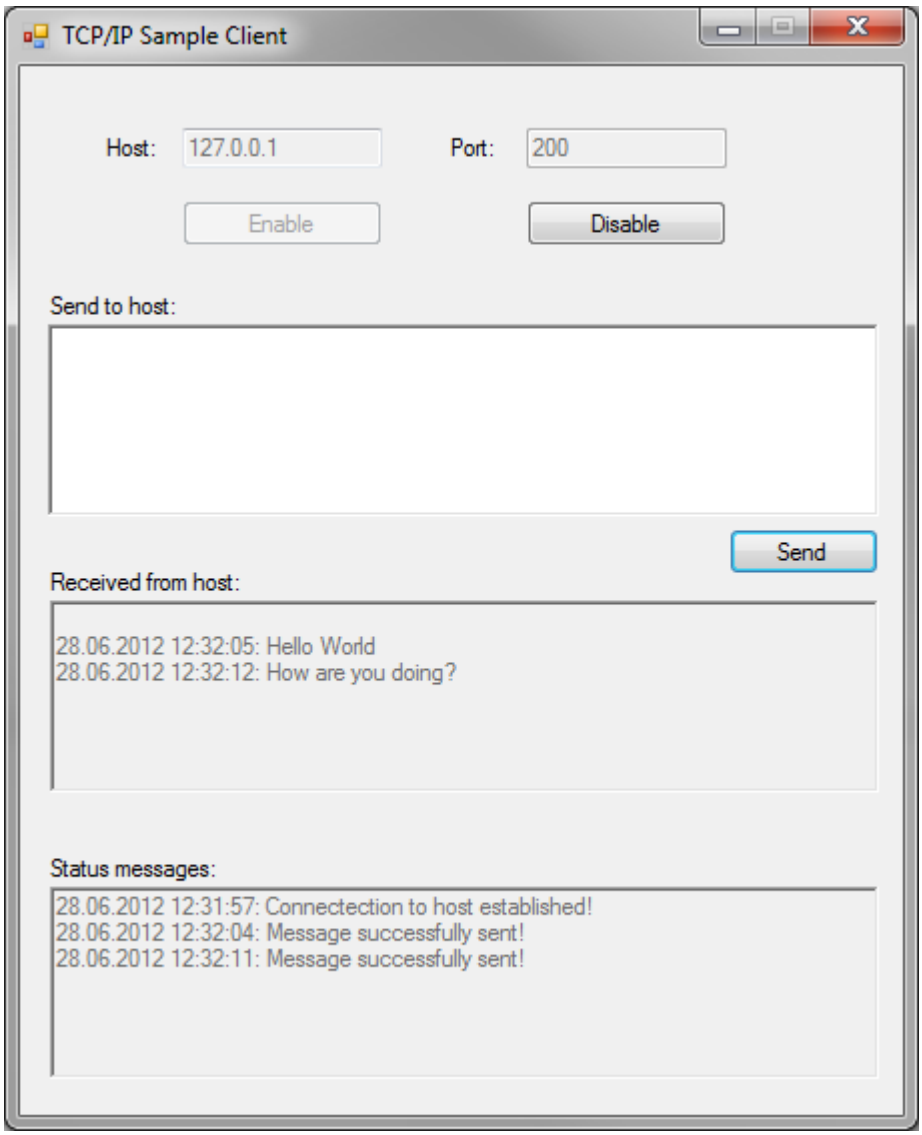
```
                        eStep := DATAEXCHA_STATE_ERROR;
                    END_IF

                ELSE(* no bytes received *)
                    fbReceiveTON( IN := FALSE, PT := PLCPRJ_RECEIVE_POLLING_TIME );
(* no bytes received => decrease the read (polling) speed *)
                    fbDisconnectTON( IN := TRUE, PT := PLCPRJ_RECEIVE_TIMEOUT );
(* no bytes received => enable timeout check*)
                    IF fbDisconnectTON.Q THEN (* timeout error*)
                        fbDisconnectTON( IN := FALSE );
                        LogError( 'FB_SocketReceive (remote client)',  PLCPRJ_ERROR_RECEIVE_TIMEOUT
);
                        nErrID := PLCPRJ_ERROR_RECEIVE_TIMEOUT;
                        eStep := DATAEXCHA_STATE_ERROR;
                    ELSE(* repeat reading *)
                        eStep := DATAEXCHA_STATE_RECEIVE_START; (* repeat reading *)
                    END_IF
                END_IF
            ELSE(* receive error *)
                LogError( 'FB_SocketReceive (remote client)',  fbSocketReceive.nErrId );
                nErrId := fbSocketReceive.nErrId;
                eStep := DATAEXCHA_STATE_ERROR;
            END_IF
        END_IF

    DATAEXCHA_STATE_SEND_START:
        fbSocketSend( bExecute := FALSE );
        fbSocketSend(   hSocket := hSocket,
                pSrc    := ADR( sFromClient ),
                cbLen   := LEN( sFromClient ) + 1,(* string length inclusive the zero delimiter *)
                bExecute:= TRUE );
        eStep := DATAEXCHA_STATE_SEND_WAIT;

    DATAEXCHA_STATE_SEND_WAIT:
        fbSocketSend( bExecute := FALSE );
        IF NOT fbSocketSend.bBusy THEN
            IF NOT fbSocketSend.bError THEN
                bBusy := FALSE;
                eStep := DATAEXCHA_STATE_IDLE;
            ELSE
                LogError( 'fbSocketSend (remote client)',  fbSocketSend.nErrId );
                nErrId := fbSocketSend.nErrId;
                eStep := DATAEXCHA_STATE_ERROR;
            END_IF
        END_IF

    DATAEXCHA_STATE_ERROR:
        bBusy := FALSE;
        bError := TRUE;
        cbReceived := 0;(* reset old received data bytes *)
        eStep := DATAEXCHA_STATE_IDLE;
END_CASE
```

Sehen Sie dazu auch

- FB_SocketReceive [▶ 25]
- FB_SocketSend [▶ 24]

8.1.1.4 .NET Sample-Client

Dieses Projektbeispiel demonstriert, wie Sie einen TCP-client für den TCP/IP Server durch eine .NET4.0 Applikation in C# realisieren können.



Der Client basiert auf den .NET Bibliotheken System.Net und System.Net.Sockets, welche einem Programmier einen einfachen Zugriff auf Socket-Funktionalitäten bieten. Durch klicken des Buttons "Enable" versucht die Anwendung zyklisch (Zykluszeit abhängig von der Konstanten TIMERTICK [ms]) eine Verbindung zum Server herzustellen. Bei erfolgreichem Verbindungsaufbau kann ein String mit einer Maximallänge von 255 Zeichen an den Server übermittelt werden. Der Server nimmt diesen String entgegen und sendet ihn an den Client zurück (sogenanntes "Echo"-Verhalten). Seitens des Servers wird die Verbindung automatisch getrennt, wenn er innerhalb einer gewissen Zeitspanne keine Daten vom Client erhalten hat. Dies lässt sich durch die Variable PLCPRJ_RECEIVE_TIMEOUT im SPS-Programm des Servers einstellen - standardmäßig steht dieser Timeout-Wert jedoch auf 50 Sekunden.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

/* ######################################################################################
* This sample TCP/IP client connects to a TCP/IP-Server, sends a message and waits for the
* response. It is being delivered together with our TCP-Sample, which implements an echo server
* in PLC.
 * ###################################################################################### */
namespace TcpIpServer_SampleClient
{
    public partial class Form1 : Form
```

```
    {
    /* ######################################################################################
     * Constants
     * ###################################################################################### */
    private const int RCVBUFFERSIZE = 256; // buffer size for receive buffer
    private const string DEFAULTIP = "127.0.0.1";
    private const string DEFAULTPORT = "200";
    private const int TIMERTICK = 100;

    /* ######################################################################################
     * Global variables
     * ###################################################################################### */
    private static bool _isConnected; // signals whether socket connection is active or not
    private static Socket _socket; // object used for socket connection to TCP/IP-Server
    private static IPEndPoint _ipAddress; // contains IP address as entered in text field
    private static byte[] _rcvBuffer; // receive buffer used for receiving response from TCP/IP-
Server
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        _rcvBuffer = newbyte[RCVBUFFERSIZE];

        /
* ######################################################################################
         * Prepare GUI
         * ######################################################################################
# */
        cmd_send.Enabled = false;
        cmd_enable.Enabled = true;
        cmd_disable.Enabled = false;
        rtb_rcvMsg.Enabled = false;
        rtb_sendMsg.Enabled = false;
        rtb_statMsg.Enabled = false;
        txt_host.Text = DEFAULTIP;
        txt_port.Text = DEFAULTPORT;

        timer1.Enabled = false;
        timer1.Interval = TIMERTICK;
        _isConnected = false;
    }

    private void cmd_enable_Click(object sender, EventArgs e)
    {
        /
* ######################################################################################
         * Parse IP address in text field, start background timer and prepare GUI
         * ######################################################################################
# */
        try
        {
            _ipAddress = newIPEndPoint(IPAddress.Parse(txt_host.Text), Convert.ToInt32(txt_port.Text
));
            timer1.Enabled = true;
            cmd_enable.Enabled = false;
            cmd_disable.Enabled = true;
            rtb_sendMsg.Enabled = true;
            cmd_send.Enabled = true;
            txt_host.Enabled = false;
            txt_port.Enabled = false;
            rtb_sendMsg.Focus();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Could not parse entered IP address. Please check spelling and retry. "
+ ex);
        }
    }

    /* ######################################################################################
     * Timer periodically checks for connection to TCP/IP-Server and reestablishes if not connected
     * ###################################################################################### */
private void timer1_Tick(object sender, EventArgs e)
    {
        if (!_isConnected)
            connect();
    }
```

```
    private void connect()
    {
        /
* ##############################################################################################
         * Connect to TCP/IP-Server using the IP address specified in the text field
         * ##############################################################################################
# */
        try
        {
            _socket = newSocket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);
            _socket.Connect(_ipAddress);
            _isConnected = true;
            if (_socket.Connected)
                rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Connectection to host establishe
d!\n");
            else
                rtb_statMsg.AppendText(DateTime.Now.ToString() + ": A connection to the host could n
ot be established!\n");
        }
        catch (Exception ex)
        {
            MessageBox.Show("An error occured while establishing a connection to the server: " + ex)
;
        }
    }

    private void cmd_send_Click(object sender, EventArgs e)
    {
        /
* ##############################################################################################
         * Read message from text field and prepare send buffer, which is a byte[] array. The last
         * character in the buffer needs to be a termination character, so that the TCP/IP-
Server knows
         * when the TCP stream ends. In this case, the termination character is '0'.
         * ##############################################################################################
# */
        ASCIIEncoding enc = new ASCIIEncoding();
        byte[] tempBuffer = enc.GetBytes(rtb_sendMsg.Text);
        byte[] sendBuffer = new byte[tempBuffer.Length + 1];
        for (int i = 0; i < tempBuffer.Length; i++)
            sendBuffer[i] = tempBuffer[i];
        sendBuffer[tempBuffer.Length] = 0;

        /
* ##############################################################################################
         * Send buffer content via TCP/IP connection
         * ##############################################################################################
# */
        try
        {
            int send = _socket.Send(sendBuffer);
            if (send == 0)
                throw new Exception();
            else
            {
            /
* ##############################################################################################
             * As the TCP/IP-
Server returns a message, receive this message and store content in receive buffer.
             * When message receive is complete, show the received message in text field.
             * ##############################################################################################
##### */
                rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Message successfully sent!\n");
                IAsyncResult asynRes = _socket.BeginReceive(_rcvBuffer, 0, 256, SocketFlags.None, nu
ll, null);
                if (asynRes.AsyncWaitHandle.WaitOne())
                {
                    int res = _socket.EndReceive(asynRes);
                    char[] resChars = newchar[res + 1];
                    Decoder d = Encoding.UTF8.GetDecoder();
                    int charLength = d.GetChars(_rcvBuffer, 0, res, resChars, 0, true);
                    String result = newString(resChars);
                    rtb_rcvMsg.AppendText("\n" + DateTime.Now.ToString() + ": " + result);
                    rtb_sendMsg.Clear();
                }
            }
        }
        catch (Exception ex)
        {
```

```
            MessageBox.Show("An error occured while sending the message: " + ex);
        }
    }

    private void cmd_disable_Click(object sender, EventArgs e)
    {
        /
* ##############################################################################################
         * Disconnect from TCP/IP-Server, stop the timer and prepare GUI
         * ##############################################################################################
# */
        timer1.Enabled = false;
        _socket.Disconnect(true);
        if (!_socket.Connected)
        {
            _isConnected = false;
            cmd_disable.Enabled = false;
            cmd_enable.Enabled = true;
            txt_host.Enabled = true;
            txt_port.Enabled = true;
            rtb_sendMsg.Enabled = false;
            cmd_send.Enabled = false;
            rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Connectection to host closed!\n");
            rtb_rcvMsg.Clear();
            rtb_statMsg.Clear();
        }
    }
  }
}
```

8.1.2 UDP Beispiel

Das folgende Beispiel zeigt die Implementierung einer einfachen Peer-To-Peer-Applikation in der SPS. Die vorgestellte SPS-Applikation kann einen Test-String zu einem Remote-PC senden und gleichzeitig Test-Strings von einem Remote-PC empfangen. Die Test-Strings werden auf dem Bildschirm des Zielrechners in einer Messagebox ausgegeben. Außerdem wird eine einfache Implementierung des passenden Kommunikationspartners in .NET vorgestellt. Das Beispiel kann als Ansatz für eigene, komplexere Implementierungen benutzt werden.

Systemvoraussetzungen

- TwinCAT v2.8 oder höher. Level: Mindestens TwinCAT PLC.
- Installierter TwinCAT TCP/IP Connection Server (v1,0,0,31 oder höher). Wenn Sie für den Test zwei PC's benutzen, dann sollte der TwinCAT TCP/IP Connection Server auf beiden PC's installiert werden.
- TwinCAT SPS-Bibliothek TcpIp.Lib (v1,0,4 oder höher).

Projektsourcen

Die Sourcen der beiden SPS-Teilnehmer unterscheiden sich nur durch unterschiedliche IP-Addressen der Remote-Kommunikationspartner.

- SPS-Projekt: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383890443.zip
- SPS-Projekt: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383891851.zip
- .NET: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383893259.zip

Projektbeschreibung

- Peer-To-Peer SPS-Applikation [▶ 73]
- .NET Kommunikationspartner für die SPS [▶ 77]
- Test der Applikationen [▶ 72]

Hilfsfunktionen im Beispielprojekt

Im Beispiel werden einige Funktionen, Konstanten und Funktionsbausteine benutzt, die im Folgenden kurz beschrieben werden müssen:

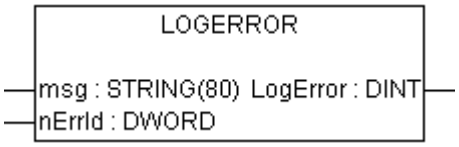
```
FUNCTION_BLOCK FB_Fifo
VAR_INPUT
    new     : ST_FifoEntry;
END_VAR
VAR_OUTPUT
    bOk     : BOOL;
    old     : ST_FifoEntry;
END_VAR
```

Ein einfacher Fifo-Funktionsbaustein. Eine Instanz von diesem Baustein wird als Sende-Fifo und eine als Empfangs-Fifo benutzt. Die zu sendenden Nachrichten werden in den Sende-Fifo, und die empfangenen Nachrichten in den Empfangs-Fifo abgelegt. Die bOk-Ausgangsvariable wird auf FALSE gesetzt, wenn bei der letzen Aktion (*AddTail* oder *RemoveHead*) Fehler aufgetreten sind (Fifo leer oder überfüllt).

Ein Fifo-Eintrag besteht aus folgenden Komponenten:

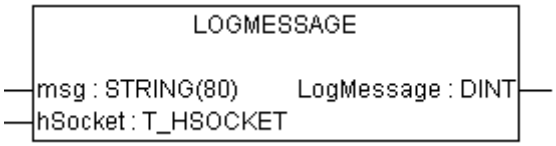
```
TYPE ST_FifoEntry :
STRUCT
    sRemoteHost : STRING(15);       (* Remote address. String containing an (Ipv4) Internet Protocol
 dotted address. *)
 nRemotePort : UDINT;     (* Remote Internet Protocol (IP) port. *)
 msg : STRING;       (* Udp packet data *)
END_STRUCT
END_TYPE
```

```
FUNCTION LogError : DINT
```



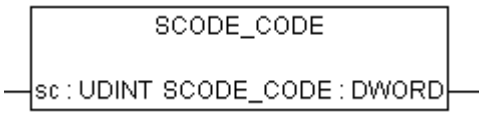
Die Funktion schreibt eine Meldung mit dem Fehlercode in das Logbuch des Betriebssystems (Event Viewer). Die globale Variable *bLogDebugMessages* muss vorher auf TRUE gesetzt werden.

```
FUNCTION LogMessage : DINT
```



Die Funktion schreibt eine Meldung in das Logbuch des Betriebssystems (Event Viewer) wenn ein neuer Socket geöffnet oder geschlossen wurde. Die globale Variable *bLogDebugMessages* muss vorher auf TRUE gesetzt werden.

```
FUNCTION SCODE_CODE : DWORD
```



Die Funktion maskiert die niederwertigsten 16Bits eines Win32 Fehlercodes aus und liefert diese zurück.

Globale Konstanten/Variablen

Name	Default value	Description
g_sTcIpConnSvrAddr	''	Die Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Default: Leerstring (der Server befindet sich auf dem lokalen PC);

Name	Default value	Description
bLogDebugMessages	TRUE	Aktiviert/Deaktiviert das Schreiben der Meldungen in das Logbuch des Betriebssystems;
PLCPRJ_ERROR_SENDFIFO_OVERFLOW	16#8103	Interner Beispielprojekt-Fehlercode: Der Sende-Fifo ist voll.
PLCPRJ_ERROR_RECFIFO_OVERFLOW	16#8104	Interner Beispielprojekt-Fehlercode: Der Empfangs-Fifo ist voll.

8.1.2.1 UDP Beispiel: Test der Peer-To-Peer-Applikationen

Für den Test benötigen Sie zwei PC's. Sie können den Test aber auch mit zwei Laufzeitsystemen auf einem PC durchführen. Die Konstanten mit den Portnummern und den IP-Adressen der Kommunikationspartner müssen entsprechend modifiziert werden.

Beispielkonfiguration für den Test mit 2 PC's :

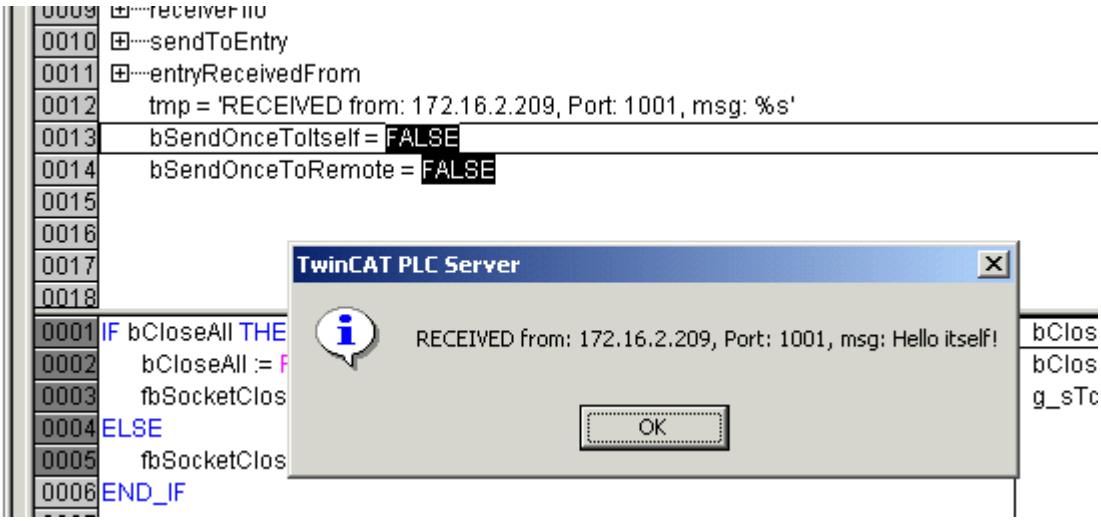
Der Teilnehmer A befindet sich auf dem lokalen PC und hat die IP-Adresse: '172.16.2.209'

Der Teilnehmer B befindet sich auf dem Remote-PC und hat die IP-Adresse: '172.16.6.195'

Test der SPS-Teilnehmer

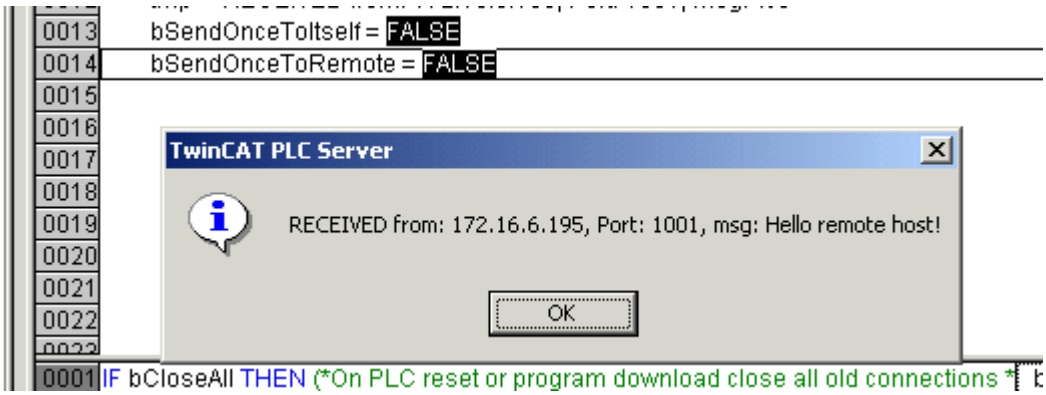
1. Lokaler PC: Öffnen Sie das https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383890443.zip auf dem lokalen PC. Die Konstante REMOTE_HOST_IP in MAIN muss an die tatsächliche IP-Adresse Ihres Remote-Systems angepasst werden (in unserem Beispiel: '172.16.6.195'). Laden Sie das Projekt in das SPS-Laufzeitsystem. Starten Sie die SPS.

2. Lokaler PC: Im Online-Betrieb beschreiben Sie im MAIN die boolsche Variable **bSendOnceToItself** einmal mit dem Wert TRUE. Kurz danach sollte eine Messagebox mit dem Test-String erscheinen. Die UDP-Daten wurden dabei an den eigenen Port und IP-Adresse gesendet.



3. Remote-PC: Öffnen Sie das https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383891851.zip auf dem Remote-PC. Die Konstante REMOTE_HOST_IP in MAIN muss an die IP-Adresse Ihres lokalen PC's angepasst werden (in unserem Beispiel: '172.16.2.209'). Laden Sie das Projekt in das SPS-Laufzeitsystem. Starten Sie die SPS.

4. Remote-PC: Im Online-Betrieb beschreiben Sie im MAIN die boolsche Variable **bSendOnceToRemote** einmal mit mit dem Wert TRUE. Kurz danach sollte eine Messagebox mit dem Test-String auf dem lokalen PC erscheinen.

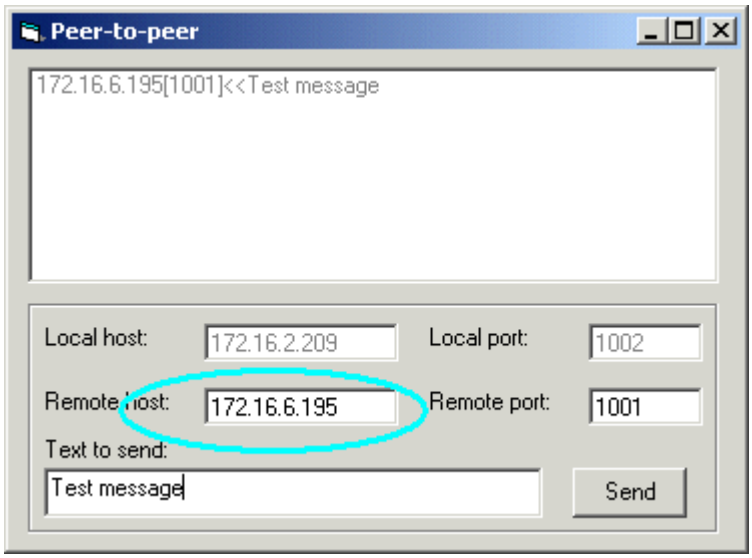


Test mit der Visual Basic Applikation

Hier können Sie die Visual Basic Sourcen entpacken: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383894667.zip.

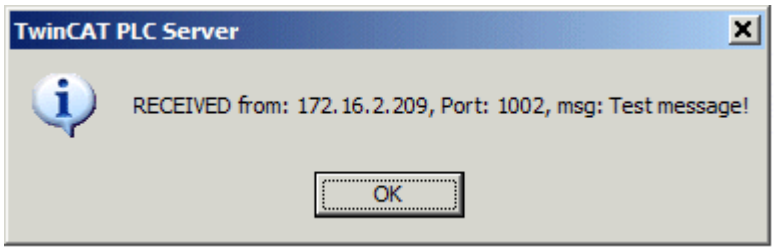
5. Lokaler PC: Starten Sie die Visual Basic Applikation (PeerToPeer.exe).

6. Lokaler PC: In dem VB-Dialog muss die IP-Adresse des Remote-Hosts an die tatsächliche IP-Adresse des Remote-PC's angepasst werden (in unserem Beispiel: '172.16.6.195').



7. Lokaler PC: Drücken Sie den Send-Button. An den Remote-Teilnehmer mit der Portnummer 1001 wird ein Test-String gesendet. In unserem Fall ist es die SPS-Applikation.

8. Remote PC: Kurz danach sollte eine Messagebox mit dem Test-String erscheinen.



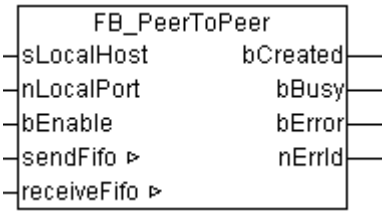
8.1.2.2 PLC Client/Server

8.1.2.2.1 UDP Beispiel: Peer-To-Peer SPS-Teilnehmer A und B

Hier können Sie die kompletten Sourcen entpacken: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383890443.zip, und https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383891851.zip.

Die benötigte Funktionalität wurde in dem Funktionsbaustein FB_PeerToPeer gekapselt. Jeder der Kommunikationsparter benutzt eine Instanz des FB_PeerToPeer-Funktionsbausteins. Durch eine steigende Flanke am *bEnable*-Eingang wird der Baustein aktiviert. Dabei wird ein neuer UDP-Socket geöffnet und der Datenaustauch gestartet. Die Socket-Adresse wird durch die Variablen *sLocalHost* und *nLocalPort* festgelegt. Eine fallende Flanke stoppt den Datenaustausch und schließt den Socket. Die zu sendenden Daten werden per Referenz (VAR_IN_OUT) über die Variable *sendFifo* an den Baustein übergeben. Die empfangenen Daten werden in die Variable *receiveFifo* abgelegt.

FUNCTION_BLOCK FB_PeerToPeer



Interface

```
VAR_IN_OUT
    sendFifo        : FB_Fifo;
    receiveFifo     : FB_Fifo;
END_VAR
VAR_INPUT
    sLocalHost      : STRING(15);
    nLocalPort      : UDINT;
    bEnable         : BOOL;
END_VAR
VAR_OUTPUT
    bCreated        : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
VAR
    fbCreate        : FB_SocketUdpCreate;
    fbClose     : FB_SocketClose;
    fbReceiveFrom   : FB_SocketUdpReceiveFrom;
    fbSendTo        : FB_SocketUdpSendTo;
    hSocket         : T_HSOCKET;
    eStep       : E_ClientServerSteps;
    sendTo      : ST_FifoEntry;
    receivedFrom    : ST_FifoEntry;
END_VAR
```

Implementierung

```
CASE eStep OF
    UDP_STATE_IDLE:
        IF bEnable XOR bCreated THEN
            bBusy   := TRUE;
            bError  := FALSE;
            nErrid  := 0;
            IF bEnable THEN
                eStep := UDP_STATE_CREATE_START;
            ELSE
                eStep := UDP_STATE_CLOSE_START;
            END_IF
        ELSIF bCreated THEN
            sendFifo.RemoveHead( old => sendTo );
            IF sendFifo.bOk THEN
                eStep := UDP_STATE_SEND_START;
            ELSE (* empty *)
                eStep := UDP_STATE_RECEIVE_START;
            END_IF
        ELSE
            bBusy := FALSE;
        END_IF

    UDP_STATE_CREATE_START:
        fbCreate(  bExecute := FALSE  );
        fbCreate(       sSrvNetId:= g_sTcIpConnSvrAddr,
```

```
            sLocalHost:= sLocalHost,
            nLocalPort:= nLocalPort,
            bExecute:= TRUE );
        eStep := UDP_STATE_CREATE_WAIT;

    UDP_STATE_CREATE_WAIT:
        fbCreate( bExecute := FALSE );
        IF NOT fbCreate.bBusy THEN
            IF NOT fbCreate.bError THEN
                bCreated := TRUE;
                hSocket := fbCreate.hSocket;
                eStep := UDP_STATE_IDLE;
                LogMessage( 'Socket opened (UDP)!', hSocket );
            ELSE
                LogError( 'FB_SocketUdpCreate', fbCreate.nErrId );
                nErrId := fbCreate.nErrId;
                eStep := UDP_STATE_ERROR;
            END_IF
        END_IF

    UDP_STATE_SEND_START:
        fbSendTo( bExecute := FALSE );
        fbSendTo(       sSrvNetId:=g_sTcIpConnSvrAddr,
            sRemoteHost := sendTo.sRemoteHost,
            nRemotePort := sendTo.nRemotePort,
            hSocket:= hSocket,
            pSrc:= ADR( sendTo.msg ),
            cbLen:= LEN( sendTo.msg ) + 1, (* include the end delimiter *)
         bExecute:= TRUE );
        eStep := UDP_STATE_SEND_WAIT;

    UDP_STATE_SEND_WAIT:
        fbSendTo( bExecute := FALSE );
        IF NOT fbSendTo.bBusy THEN
            IF NOT fbSendTo.bError THEN
                eStep := UDP_STATE_RECEIVE_START;
            ELSE
                LogError( 'FB_SocketSendTo (UDP)', fbSendTo.nErrId );
                nErrId := fbSendTo.nErrId;
                eStep := UDP_STATE_ERROR;
            END_IF
        END_IF

    UDP_STATE_RECEIVE_START:
        MEMSET( ADR( receivedFrom ), 0, SIZEOF( receivedFrom ) );
        fbReceiveFrom( bExecute := FALSE );
        fbReceiveFrom(  sSrvNetId:=g_sTcIpConnSvrAddr,
                hSocket:= hSocket,
                pDest:= ADR( receivedFrom.msg ),
                cbLen:= SIZEOF( receivedFrom.msg ) - 1, (*without string delimiter *)
             bExecute:= TRUE );
        eStep := UDP_STATE_RECEIVE_WAIT;

    UDP_STATE_RECEIVE_WAIT:
        fbReceiveFrom( bExecute := FALSE );
        IF NOT fbReceiveFrom.bBusy THEN
            IF NOT fbReceiveFrom.bError THEN
                IF fbReceiveFrom.nRecBytes > 0 THEN
                    receivedFrom.nRemotePort := fbReceiveFrom.nRemotePort;
                    receivedFrom.sRemoteHost := fbReceiveFrom.sRemoteHost;
                    receiveFifo.AddTail( new := receivedFrom );
                    IF NOT receiveFifo.bOk THEN(* Check for fifo overflow *)
                     LogError( 'Receive fifo overflow!', PLCPRJ_ERROR_RECFIFO_OVERFLOW );
                    END_IF
                END_IF
                eStep := UDP_STATE_IDLE;
            ELSIF fbReceiveFrom.nErrId = 16#80072746 THEN
                LogError( 'The connection is reset by remote side.', fbReceiveFrom.nErrId );
                eStep := UDP_STATE_IDLE;
            ELSE
                LogError( 'FB_SocketUdpReceiveFrom (UDP client/server)', fbReceiveFrom.nErrId );
                nErrId := fbReceiveFrom.nErrId;
                eStep := UDP_STATE_ERROR;
            END_IF
        END_IF

    UDP_STATE_CLOSE_START:
        fbClose( bExecute := FALSE );
        fbClose(    sSrvNetId:= g_sTcIpConnSvrAddr,
            hSocket:= hSocket,
```

```
            bExecute:= TRUE );
        eStep := UDP_STATE_CLOSE_WAIT;

    UDP_STATE_CLOSE_WAIT:
        fbClose( bExecute := FALSE );
        IF NOT fbClose.bBusy THEN
            LogMessage( 'Socket closed (UDP)!', hSocket );
            bCreated := FALSE;
            MEMSET( ADR(hSocket), 0, SIZEOF(hSocket));
            IF fbClose.bError THEN
                LogError( 'FB_SocketClose (UDP)', fbClose.nErrId );
                nErrId := fbClose.nErrId;
                eStep := UDP_STATE_ERROR;
            ELSE
                bBusy := FALSE;
                bError := FALSE;
                nErrId := 0;
                eStep := UDP_STATE_IDLE;
            END_IF
        END_IF

    UDP_STATE_ERROR: (* Error step *)
        bError := TRUE;
        IF bCreated THEN
            eStep := UDP_STATE_CLOSE_START;
        ELSE
            bBusy := FALSE;
            eStep := UDP_STATE_IDLE;
        END_IF
END_CASE
```

MAIN-Programm

Nach einem Programm-Download oder SPS-Reset müssen die vorher geöffnten Sockets geschlossen werden. Dies geschieht beim SPS-Start durch den einmaligen Aufruf einer Instanz des FB_SocketCloseAll [▶ 20]-Funktionsbausteins. Bei einer steigenden Flanke an einer der Variablen: *bSendOnceToItself* oder *bSendOnceToRemote*, wird ein neuer Fifo-Eintrag generiert und in den Sende-Fifo abgelegt. Empfangene Nachrichten werden aus dem Empfangs-Fifo entnommen und in einer Messagebox angezeigt.

```
PROGRAM MAIN
VAR CONSTANT
    LOCAL_HOST_IP       : STRING(15)        := '';
    LOCAL_HOST_PORT     : UDINT         := 1001;
    REMOTE_HOST_IP      : STRING(15)        := '172.16.2.209';
    REMOTE_HOST_PORT        : UDINT             := 1001;
END_VAR
VAR
    fbSocketCloseAll        : FB_SocketCloseAll;
    bCloseAll           : BOOL := TRUE;

    fbPeerToPeer        : FB_PeerToPeer;
    sendFifo            : FB_Fifo;
    receiveFifo         : FB_Fifo;
    sendToEntry         : ST_FifoEntry;
    entryReceivedFrom           : ST_FifoEntry;
    tmp             : STRING;

    bSendOnceToItself           : BOOL;
    bSendOnceToRemote           : BOOL;
END_VAR
```

```
IF bCloseAll THEN (*On PLC reset or program download close all old connections *)
 bCloseAll := FALSE;
    fbSocketCloseAll( sSrvNetId:= g_sTcIpConnSvrAddr, bExecute:= TRUE, tTimeout:= T#10s );
ELSE
    fbSocketCloseAll( bExecute:= FALSE );
END_IF

IF NOT fbSocketCloseAll.bBusy AND NOT fbSocketCloseAll.bError THEN

    IF bSendOnceToRemote THEN
        bSendOnceToRemote               := FALSE;                       (* clear flag *)
     sendToEntry.nRemotePort         := REMOTE_HOST_PORT;            (* remote host port number*)
     sendToEntry.sRemoteHost         := REMOTE_HOST_IP;                  (* remote host IP address *
)
     sendToEntry.msg             := 'Hello remote host!';            (* message text*);
```

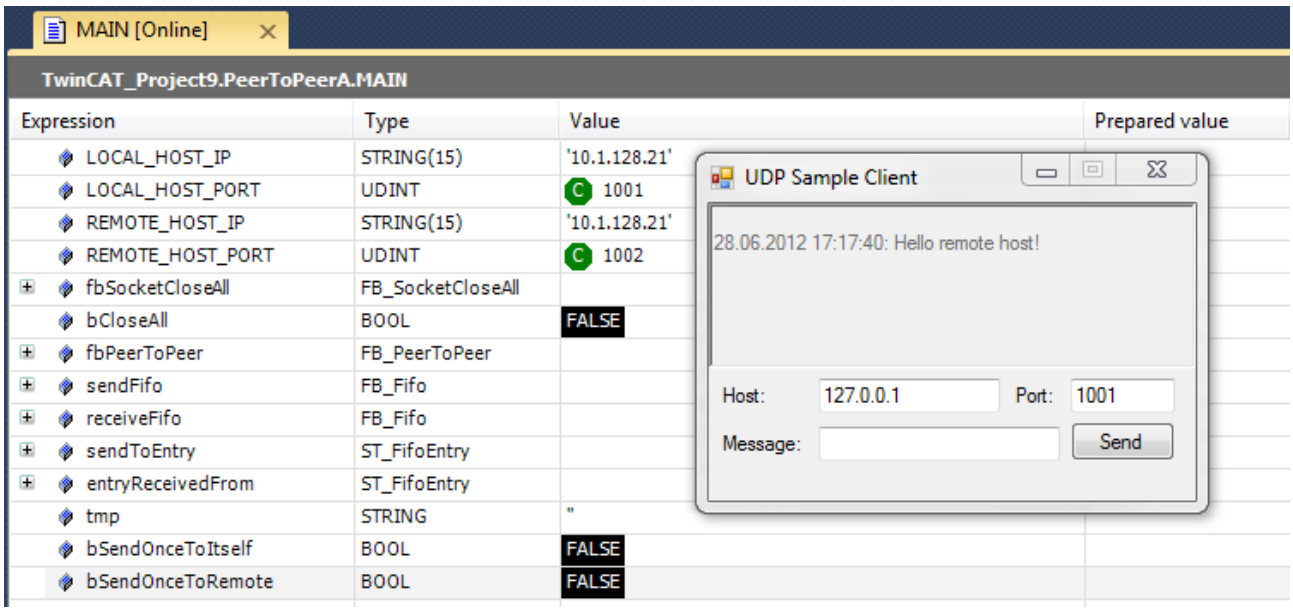
```
    sendFifo.AddTail( new := sendToEntry );                (* add new entry to the send queue*)
    IF NOT sendFifo.bOk THEN                              (* check for fifo overflow*)
        LogError( 'Send fifo overflow!', PLCPRJ_ERROR_SENDFIFO_OVERFLOW );
       END_IF
   END_IF

   IF bSendOnceToItself THEN
       bSendOnceToItself              := FALSE;                      (* clear flag *)
    sendToEntry.nRemotePort         := LOCAL_HOST_PORT;             (* nRemotePort == nLocalPort =>
send it to itself *)
    sendToEntry.sRemoteHost         := LOCAL_HOST_IP;                (* sRemoteHost == sLocalHos
t =>send it to itself *)
    sendToEntry.msg             := 'Hello itself!';             (* message text*);
    sendFifo.AddTail( new := sendToEntry );                (* add new entry to the send queue*)
    IF NOT sendFifo.bOk THEN                              (* check for fifo overflow*)
        LogError( 'Send fifo overflow!', PLCPRJ_ERROR_SENDFIFO_OVERFLOW );
       END_IF
   END_IF

  (* send and receive messages *)
 fbPeerToPeer( sendFifo := sendFifo, receiveFifo := receiveFifo, sLocalHost := LOCAL_HOST_IP, nLocal
Port := LOCAL_HOST_PORT, bEnable := TRUE );

  (* remove all received messages  from receive queue *)
 REPEAT
       receiveFifo.RemoveHead( old => entryReceivedFrom );
       IF receiveFifo.bOk THEN
           tmp := CONCAT( 'RECEIVED from: ', entryReceivedFrom.sRemoteHost  );
           tmp := CONCAT( tmp, ', Port: ' );
           tmp := CONCAT( tmp, UDINT_TO_STRING( entryReceivedFrom.nRemotePort ) );
           tmp := CONCAT( tmp, ', msg: %s' );
           ADSLOGSTR(  ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_MSGBOX, tmp, entryReceivedFrom.msg );
       END_IF
   UNTIL NOT receiveFifo.bOk
   END_REPEAT

END_IF
```

Sehen Sie dazu auch

- FB_SocketUdpCreate [▶ 26]
- FB_SocketClose [▶ 19]
- FB_SocketUdpReceiveFrom [▶ 29]
- FB_SocketUdpSendTo [▶ 28]
- T_HSOCKET [▶ 33]

8.1.2.3 UDP sample: .NET Peer-to-Peer Kommunikation

Dieses Beispiel demonstriert, wie Sie einen .NET Kommunikationspartner für die SPS-Beispiele Peer-to-Peer device A oder B realisieren können.



Wie das Beispiel funktioniert

Dieses Beispiel nutzt die .NET Bibliotheken System.Net und System.Net.Sockets, um durch die Klasse UdpClient einen UDP-Client zu realisieren. Parallel zum Empfangen eingehender UDP-Pakete durch einen Background-Thread, kann auch ein String an ein remote-Gerät übermittelt werden, indem dessen IP-Adresse und Portnummer eingegeben und der Button "Send" angeklickt wird.

DIe folgende Anleitung basiert auf dem folgenden Setup:

- Das SPS-Projekt Peer-to-Peer device A läuft auf einem Computer mit der IP-Adresse 10.1.128.21
- Die .NET Anwendung läuft auf einem Computer mit der IP-Adresse 10.1.128.30

Einrichtung des SPS-Projekts

Dieses .NET Beispiel läuft im Zusammenspiel mit den SPS-Beispielen Peer-to-Peer device A oder B. Wenn Sie die Anwendung auf einem anderen Computer als die SPS-Laufzeit ausführen, müssen Sie die IP-Adressen beider Geräte anhand Ihrer tatsächlichen Umgebung entsprechend in den Beispielprojekten anpassen. Die folgende Tabelle geht von oben erwähnten Beispiel-Setup aus:

Einstellung	Typ	Beschreibung
LOCAL_HOST_IP	Globale Konstante	10.1.128.21 (IP-Adresse des Computers mit der SPS-Laufzeit)
LOCAL_HOST_PORT	Globale Konstante	1001 (Standard-Wert)
REMOTE_HOST_IP	Globale Konstante	10.1.128.30 (IP-Adresse des Computers mit der .NET Anwendung)
REMOTE_HOST_PORT	Globale Konstante	1002 (Standard-Wert)
bSendOnceToRemote	Globale Variable	Wenn auf TRUE gesetzt, wird ein UDP-Paket an die IP-Adresse REMOTE_HOST_IP gesendet

Einrichtung der .NET Anwendung

Einstellung	Typ	Beschreibung
DEFAULTIP	Globale Konstante	10.1.128.21 (Wird in dem Textfeld "Host" als Standardwert eingetragen.)
DEFAULTDESTPORT	Globale Konstante	1001 (Wird in dem Textfeld "Port" als Standardwert eingetragen.)
DEFAULTOWNPORT	Globale Konstante	1002 (Entspricht dem Wert der globalen Konstante REMOTE_HOST_PORT im SPS-Projekt, siehe oben.)

Einstellung	Typ	Beschreibung
DEFAULTSOURCEPORT	Globale Konstante	Wird als Quell-Port benutzt, über den das UDP-Paket verschickt wird. Dieser kann auf der Standardeinstellung belassen werden.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Net;
using System.Net.Sockets;
using System.Threading;


namespace TcpIpServer_SampleClientUdp
{
    publicpartialclassForm1 : Form
    {
    /
* ############################################################################################     * C
onstants     * ############################################################################################
##### */privateconststring DEFAULTIP = "127.0.0.1";
    privateconstint DEFAULTDESTPORT = 1001; // Used as destination portprivateconstint DEFAULTOWNPOR
T = 1002; // Used for binding when listening for UDP messagesprivateconstint DEFAULTSOURCEPORT = 110
00; // Only used as source port when sending UDP messages/
* ############################################################################################     * G
lobal variables     * ############################################################################
############ */privatestaticUdpClient _udpClient;
    privatestaticIPEndPoint _ipAddress; // Contains IP address as entered in text fieldprivatestatic
Thread _rcvThread; // Background thread used to listen for incoming UDP packetspublic Form1()
    {
        InitializeComponent();
    }


    /
* ############################################################################################     * E
vent handler method called when button "Send" is pressed     * ######################################
################################################### */
privatevoid cmd_send_Click(object sender, EventArgs e)
    {
        byte[] sendBuffer = null;

        /
* ############################################################################################
 * Preparing UdpClient, connecting to UDP server and sending content of text field         * #######
################################################################################ */try
        {
        _ipAddress = newIPEndPoint(IPAddress.Parse(txt_host.Text), Convert.ToInt32(txt_port.Text));
        _udpClient = newUdpClient(DEFAULTSOURCEPORT);
        _udpClient.Connect(_ipAddress);

        sendBuffer = Encoding.ASCII.GetBytes(txt_send.Text);
        _udpClient.Send(sendBuffer, sendBuffer.Length);

        _udpClient.Close();
        }
        catch (Exception ex)
        {
        MessageBox.Show("An unknown error occured: " + ex);
        }
    }


    /
* ############################################################################################     * E
vent handler method called when application starts     * ########################################
############################################## */privatevoid Form1_Load(object sender, EventArgs e)
    {
        txt_host.Text = DEFAULTIP;
        txt_port.Text = DEFAULTDESTPORT.ToString();
        rtb_rcv.Enabled = false;
```

```
        /
* ##########################################################################################
 * Creating background thread which synchronously listens for incoming UDP packets         * #######
########################################################################################## */
        _rcvThread = newThread(rcvThreadMethod);
        _rcvThread.Start();
    }


    /
* ##########################################################################################     * D
elegate, so that background thread may write into text field on GUI     * ##########################
################################################################## */
publicdelegatevoidrcvThreadCallback(string text);


    /
* ##########################################################################################     * M
ethod called by background thread     * ##########################################################
############################## */privatevoid rcvThreadMethod()
    {
        /
* ##########################################################################################
 * Listen on any available local IP address and specified port (DEFAULTOWNPORT)          * ##########
########################################################################################## */
byte[] rcvBuffer = null;
        IPEndPoint ipEndPoint = newIPEndPoint(IPAddress.Any, DEFAULTOWNPORT);;
        UdpClient udpClient = newUdpClient(ipEndPoint);

        /
* ##########################################################################################
 * Continously start a synchronous listen for incoming UDP packets. If a packet has arrived,
 * write its content to receive buffer and then into the text field. After that, start circle
  * again.          * ##########################################################################################
########## */while (true)
        {
        rcvBuffer = udpClient.Receive(ref ipEndPoint); // synchronous call
        rtb_rcv.Invoke(newrcvThreadCallback(this.AppendText), newobject[] { "\n" + DateTime.Now.ToSt
ring() + ": " + Encoding.ASCII.GetString(rcvBuffer) });
        }
    }


    /
* ##########################################################################################     * H
elper method for delegate     * ##########################################################################
##################### */privatevoid AppendText(string text)
    {
        rtb_rcv.AppendText(text);
    }


    /
* ##########################################################################################     * S
top background thread when application closes     * ##############################################
######################################### */
privatevoid Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        _rcvThread.Abort();
    }
    }
}
```

8.2 TcSocketHelper.lib-Beispiele

Die vorgestellten Beispiele nutzen die Funktionalitäten der TcSocketHelper.Lib.

Systemvoraussetzungen:

- TwinCAT Version 2.10 Build 1331 oder höher;
- Installierter TwinCAT Connection Server v1.0.0.47 oder höher auf dem Client- und Server-PC;

Die Kommunikationseinstellungen die in den Beispielen verwendet werden:

- SPS-Client-Applikation: Die Port- und IP-Adresse des Remote-Servers: 200, "127.0.0.1";
- SPS-Server-Applikation: Die Port- und IP-Adresse des Local-Servers: 200, "127.0.0.1";

Wenn Sie die Client- und Server-Applikation auf zwei verschiedenen PC's testen wollen, dann muss die Port-Adresse und die IP-Adresse entsprechend angepasst werden (die Verbindung mit dem PING-Befehl in der Eingabeaufforderung testen).

Mit den Default-Werten können Sie den Client und Server auf einem PC testen, indem Sie die Client-Applikation in das erste SPS-Laufzeitsystem (801) und die Server-Applikation in das zweite SPS-Laufzeitsystem (811) laden.

Das Verhalten der SPS-Beispielprojekte wird durch folgende Konstanten festgelegt.

Konstante	Wert	Beschreibung
PLCPRJ_MAX_CONNECTIONS	5	Max. Anzahl der Server->Client-Verbindungen. Ein Server kann Verbindungen zu mehr als einem Client aufbauen. Ein Client kann immer nur zu einem Server Verbindung aufbauen.
PLCPRJ_SERVER_RESPONSE_TIMEOUT	T#10s	Max. Verzögerungszeit (Timeout-Zeit) nach der ein Server eine Antwort an den Client senden soll.
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	Zykluszeit in der ein Client Sendedaten (TX) an den Server sendet.
PLCPRJ_RECEIVER_POLLING_CYCLE_TIME	T#200ms	Zykluszeit in der ein Client oder Server nach Empfangsdaten (RX) pollend fragt.
PLCPRJ_BUFFER_SIZE	10000	Max. interne Puffergröße für RX/TX-Daten.

Die SPS-Beispiele definieren und benutzen folgende interne Fehlercodes:

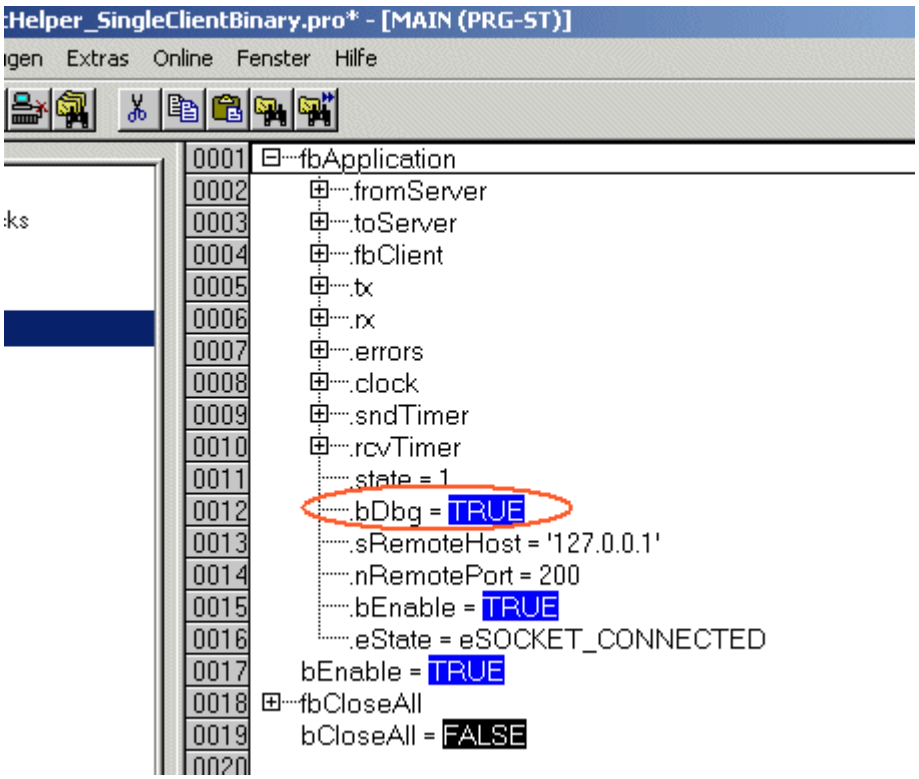
Fehlercode	Wert	Beschreibung
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	Der interne Empfangspuffer meldet einen Überlauf.
PLCPRJ_ERROR_SEND_BUFFER_OVERFLOW	16#8102	Der interne Sendepuffer meldet einen Überlauf.
PLCPRJ_ERROR_RESPONSE_TIMEOUT	16#8103	Der Server hat die Antwort in der angegebenen Timeoutzeit nicht gesendet.
PLCPRJ_ERROR_INVALID_FRAME_FORMAT	16#8104	Das Telegramm hat eine fehlerhafte Formatierung (Grösse, fehlerhaften Datenbytes usw.).

SPS Project	Beschreibung
https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383896075.zip (Client) https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383897483.zip (Server)	Implementierung eines "Echo"-Clients/-Servers. Der Client sendet zyklisch einen Test-String (**sToServer**) zum Remote-Server. Der Server sendet den gleichen String unverändert an den Client zurück (**sFromServer**).
https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383898891.zip (Client) https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383900299.zip (Server)	Wie oben, mit dem Unterschied, dass der Server bis zu 5 Verbindungen aufbauen kann. Die Client-Applikation besitzt 5-Client-Instanzen. Jede Instanz baut eine Verbindung zum Server auf.
https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383901707.zip (Client) https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383903115.zip (Server)	Eine Client-Server-Applikation für den Austausch von Binärdaten. Es wird ein einfaches Beispielprotokoll verwendet. Im Protokoll-Header wird die Länge der Binärdaten und ein Framezähler für die gesendeten und empfangenen Telegramme übertragen.

SPS Project	Beschreibung
	Die Struktur der Binärdaten wird durch die SPS-Struktur: **ST_ApplicationBinaryData** festgelegt. Die Binärdaten werden an den Header angehängt und übertragen. Die Instanzen der Binärstruktur haben auf der Client-Seite den Namen: **toServer**, **fromServer** bzw. auf der Server-Seite: **toClient**, **fromClient**. Sie können die Strukturdeklaration auf der Client und Server-Seite an Ihre Anforderungen anpassen. Die Strukturdeklaration muss aber auf beiden Seiten gleich sein! Die max. Größe der Struktur darf aber die max. Puffergröße der Sende-/Empfangs-Fifos nicht überschreiten! Die max. Puffergröße ist durch eine Konstante festgelegt. Die Server-Funktionalität ist im Funktionsbaustein **FB_ServerApplication** und die Client-Funktionalität ist im Funktionsbaustein **FB_ClientApplication** implementiert. In der Standard-Implementierung sendet der Client die Daten der Binärstruktur zyklisch zum Server und wartet auf eine Antwort vom Server. Der Server modifiziert einige Daten und sendet diese zurück an den Client. Wenn Sie eine andere Funktionalität benötigen, müssen Sie die Funktionsbausteine **FB_ServerApplication** und **FB_ClientApplication** entsprechend modifizieren.
https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383904523.zip (Client) https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383905931.zip (Server)	Wie oben, mit dem Unterschied, dass der Server bis zu 5 Verbindungen aufbauen kann. Die Client-Applikation besitzt 5-Client-Instanzen. Jede Instanz baut eine Verbindung zum Server auf.

Die Client- bzw. Server-Applikationen (FB_ServerApplication, FB_ClientApplication) wurden als Funktionsbausteine implementiert. Die Applikation und die Verbindung können dadurch mehrfach instanziiert werden.

Für die Fehlersuche können Sie die Eingangsvariablen *bDbg* auf TRUE setzen und so die Debug-Ausgabe der gesendeten Daten im TwinCAT System Manager Log View aktivieren:



8.3 TcSnmp.lib

8.3.1 Sample: Client trap

This sample describes a simple Trap send from a PLC to a SNMP management server. Traps can be used to alert tresholds. On every hundred increment of the counter a trap will be send.

Download: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383907339.zip

Variable Declaration

```
PROGRAM MAIN
VAR
    SendTrap:       FB_SendTrap;
    iState:         INT := 0;
    iCounter:       UDINT:= 0;
    stVarBind:      ST_SNMP_VariableBinding;
END_VAR
```

PLC Program

```
CASE iState OF

    0: (* send trap on every hundred cycle *)
        iCounter := iCounter + 1;
        IF ((iCounter MOD 100) = 0) THEN
            iState := 10;
        END_IF

    10: (* enable FB *)
        SendTrap.bEnable := TRUE;
        IF SendTrap.bEnabled THEN
            iState := 20;
        END_IF

    20: (* set SNMP trap parameter *)
        stVarBind.iType := E_SNMP_INTEGER;
        stVarBind.iLength := SIZEOF(iCounter);
        stVarBind.pArrValue := ADR(iCounter);

        (* TODO: assign object ID of management information base (MIB) *)
        SendTrap.sObjectID := '1.3.6.1.2.1.1.5.0';
```

```
        (* TODO: check default community string *)
        SendTrap.sCommunity:= 'public';
        SendTrap.iGenericTrapNumber:= E_SNMP_WarmStart;
        SendTrap.bExecute := TRUE;
        iState := 30;

    30: (* reset *)
        SendTrap.bExecute := FALSE;
        SendTrap.bEnable := FALSE;
        IF NOT SendTrap.bBusy THEN
            iState := 0;
        END_IF

        IF SendTrap.bError THEN
            SendTrap.bEnable := FALSE;
            iState := 99;
        END_IF

    99:(* Error case *)
        ;
END_CASE

SendTrap(
    (* TODO: add device IP of TwinCAT device *)
        sLocalHostIp := '',
    (* TODO: add SNMP Manager IP *)
        sManagerIP := ''
    );
```

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcSnmp.lib (TcpIp.Lib; Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

8.3.2 Sample: SNMP multiple client trap

This Sample describes the sending of a trap by multiple values. Traps can be used to alert tresholds. On every hundred increment of the counter a trap will be send.

Download: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383908747.zip

Variablen Declaration

```
PROGRAM MAIN
VAR
    SendTrap:       FB_SendTrap;
    iState:         INT := 0;
    iCounter:       UINT:= 0;
    sObjectId:      STRING :='1.3.6.1.4.1.2';
    sMessage:       T_MaxString:='Information string';
    iGauge:         UDINT:= 4294967295;
    iTimeTicks:     UDINT:= 1000000;
    arrVarBind:     ARRAY[1..5] OF ST_SNMP_VariableBinding;
END_VAR
```

PLC Program

```
CASE iState OF

    0: (* send trap on every hundred cycle *)
        iCounter := iCounter + 1;
        IF ((iCounter MOD 100) = 0) THEN
            iState := 10;
        END_IF

    10:(* enable FB *)
        SendTrap.bEnable := TRUE;
        IF SendTrap.bEnabled THEN
            iState := 20;
        END_IF
```

```
    20: (* set SNMP trap parameter *)
        arrVarBind[1].iType := E_SNMP_INTEGER;
        arrVarBind[1].iLength := 4;
        arrVarBind[1].pArrValue := ADR(iCounter);
        arrVarBind[1].sOID := '1.3.1.3.255.1.1';

        (*Variable Binding 2*)
        arrVarBind[2].iType := E_SNMP_OBJECTID;
        arrVarBind[2].iLength := LEN(sObjectId);
        arrVarBind[2].pArrValue: = ADR(sObjectId);
        arrVarBind[2].sOID:= '1.3.6.1.3.255.2';

        (*Variable Binding 3*)
        arrVarBind[3].iType := E_SNMP_OCTETSTRING;
        arrVarBind[3].iLength := LEN(sMessage);
        arrVarBind[3].pArrValue := ADR(sMessage);
        arrVarBind[3].sOID := '1.3.6.1.3.255.3';

        (*Variable Binding 4*)
        arrVarBind[4].iType := E_SNMP_GAUGE32;
        arrVarBind[4].iLength := 4;
        arrVarBind[4].pArrValue := ADR(iGauge);
        arrVarBind[4].sOID := '1.3.6.1.3.255.4';

        (*Variable Binding 5*)
        arrVarBind[5].iType := E_SNMP_TIMETICKS;
        arrVarBind[5].iLength := 4;
        arrVarBind[5].pArrValue := ADR(iTimeTicks);
        arrVarBind[5].sOID := '1.3.6.1.3.255.5';

        SendTrap.sCommunity := 'public';
        SendTrap.iGenericTrapNumber := E_SNMP_WarmStart;
        SendTrap.bExecute := TRUE;
        iState := 30;

    30:     (* reset *)
        SendTrap.bExecute := FALSE;
        SendTrap.bEnable := FALSE;
        IF NOT SendTrap.bBusy THEN
            iState := 0;
        END_IF

        IF SendTrap.bError THEN
            SendTrap.bEnable := FALSE;
            iState := 99;
        END_IF

    99:     (* Error case *)
        ;
END_CASE

SendTrap(
        (* TODO: add device IP of TwinCAT device *)
            sLocalHostIp := '',
        (* TODO: add SNMP Manager IP *)
            sManagerIP := '',
            pArrVarBinding := ADR(arrVarBind),
            nVarBindings := 5
    );
```

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcSnmp.lib (TcpIp.Lib; Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

8.3.3 Sample: SNMP Get request

This samples describes how a TwinCAT device can answer on a SNMP GET-request from SNMP manager. If a GET-Request arrived in the PLC, the requested information will be sent if the object id match.

Download: https://infosys.beckhoff.com/content/1031/tcpipserver/Resources/11383910155.zip

Variablen Declaration

```
PROGRAM MAIN
VAR
    GetSnmp:    FB_GetSnmp;
    iValue:     DINT;
    sValue:     T_MaxString;
    arrVarBind: ARRAY[0..1] OF ST_SNMP_VariableBinding;
    iState:     INT := 0;
END_VAR
```

PLC Program

```
CASE iState OF
    0:(* Enable *)
        GetSnmp.bEnable := TRUE;
        IF GetSnmp.bEnabled THEN
            iState := 10;
        END_IF

    10:(* Wait for SNMP GET request *)
        GetSnmp.bReceive := TRUE;
        IF GetSnmp.bReceived THEN
            GetSnmp.bReceive := FALSE;
            iState := 20;
        END_IF

    20: (* Compare oid's *)

        (* TODO: Assign your own ObjectID (oid) of the management information base (MIB) *)
        IF GetSnmp.sRecObjectID = '1.3.6.1.2.1.1.5.0' THEN
            sValue := 'BECKHOFF_DEVICE';
            arrVarBind[0].iType := E_SNMP_OCTETSTRING;
            arrVarBind[0].iLength :=LEN(sValue);
            arrVarBind[0].pArrValue := ADR(sValue);
            arrVarBind[0].sOID := GetSnmp.sRecObjectID;
            GetSnmp.pArrVarBinding := ADR(arrVarBind);
            GetSnmp.nVarBindings := 1;
            GetSnmp.iError := 0;
            GetSnmp.bSendResponse := TRUE;
        ELSE
        (* The requested ObjectID  was not found *)
            GetSnmp.nVarBindings := 0;
            GetSnmp.iError := 2;
            GetSnmp.bSendResponse := TRUE;
        END_IF
        iState := 30;

    30: (* reset *)
        GetSnmp.bSendResponse := FALSE;
        GetSnmp.bSendTrap := FALSE;
        IF NOT GetSnmp.bBusy THEN
            iState := 10;
        END_IF
        IF GetSnmp.bError THEN
            GetSnmp.bEnable := FALSE;
            iState := 0;
        END_IF
END_CASE

GetSnmp(
    (* TODO: check community string *)
        sCommunity := 'public',
        iGenericTrapNumber:= E_SNMP_WarmStart,
    (* TODO: add device IP of TwinCAT device *)
        sLocalHostIp := '',
    (* use SNMP Port 163, if default port 161 is in use by OS *)
        sLocalHostPort := 163,
    (* TODO: ADD SNMP Manager IP *)
        sManagerIP := '',
    );
```

Voraussetzungen

Development environment	Target system type	PLC libraries to be linked
TwinCAT version 2.8.0 or higher	PC or CX (x86)	TcSnmp.lib (TcpIp.Lib; Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

9 Fehlercodes

Voraussetzungen

Fehlercodes (Hex)	Fehlercodes (Dez)	Fehlerquelle	Beschreibung
0x00000000-0x0 0007800	0-30720	TwinCAT System Fehlercodes	TwinCAT System Fehler (ADS-Fehlercodes inklusive)
0x00008000-0x0 00080FF	32768-33023	Interne TwinCAT TCP/IP Connection Server Fehlecodes [▶ 88]	Interne Fehler des TwinCAT TCP/IP Connection Servers
0x00009000-0x0 00090FF	36864-37119	Interne SNMP Fehlercodes [▶ 89]	Interne SNMP Fehlercodes
0x80070000-0x8 007FFFF	2147942400-2148007 935	Fehlerquelle = Fehlercode - 0x80070000 = Win32 System Fehlercode	Win32 Systemfehler (Windows Sockets Fehlercodes inklusive)

9.1 Interne Fehlercodes des TwinCAT TCP/IP Connection Servers

Fehlercode (Hex)	Fehlercode (Dez)	Symbolische Konstante	Beschreibung
0x00008001	32769	TCPADSERROR_NOMOREENTRIES	Es können keine neuen Sockets mehr erstellt werden (bei FB_SocketListen und FB_SocketConnect).
0x00008002	32770	TCPADSERROR_NOTFOUND	Socket-Handle ist ungültig (bei FB_SocketReceive, FB_SocketAccept, FB_SocketSend etc.).
0x00008003	32771	TCPADSERROR_ALREADYEXISTS	Wird beim Aufruf von FB_SocketListen zurückgeliefert, wenn der Listener TcpIp-Port schon existiert.
0x00008004	32772	TCPADSERROR_NOTCONNECTED	Wird beim Aufruf von FB_SocketReceive zurückgeliefert, falls der Client Socket nicht mehr mit dem Server verbunden ist.
0x00008005	32773	TCPADSERROR_NOTLISTENING	Wird beim Aufruf von FB_SocketAccept zurückgeliefert, falls ein Fehler im Listener Socket registriert wurde.

9.2 Fehlersuche/Diagnose

1. Bei Verbindungsproblemen kann der PING-Befehl dazu benutzt werden, um festzustellen, ob der Kommunikationspartner über die Netzwerkverbindung erreichbar ist. Wenn dies nicht der Fall ist, überprüfen Sie die Netzwerkkonfiguration und die Firewall-Einstellungen.
2. Eine komplette Aufzeichnung der Netzwerkkommunikation kann mit Sniffer-Tools wie Wireshark durchgeführt werden. Die Aufnahme kann dann vom Beckhoff-Supportpersonal analysiert werden.

3. Überprüfen Sie die in dieser Dokumentation beschriebenen Hardware- und Softwareanforderungen (TwinCAT-Version, CE Image-Version usw.).
4. Überprüfen Sie die Installationshinweise (z.B. Installation der CAB-Files auf einem CE System).
5. Überprüfen Sie, ob die Eingangsparameter, die Sie an die Funktionsbausteine übergeben, richtig sind (Netzwerkadresse, Portnummer, Daten, Verbindungshandle usw.). Überprüfen Sie, ob der Funktionsbaustein einen Fehlercode ausgibt. Die Dokumentation zu den Fehlercodes finden Sie hier: Übersicht der Fehlercodes [▶ 88].
6. Überprüfen Sie ob der andere Kommunikationspartner/Software/Gerät einen Fehlercode ausgibt.
7. Aktivieren Sie die in der TcSocketHelper.Lib integrierte Debug-Ausgaben beim Aufbauen und Abbauen der Verbindung (Stichwort: CONNECT_MODE_ENABLEDBG). Öffnen Sie den TwinCAT System Manager und aktivieren das LogView-Fenster. Prüfen Sie die Debug-Ausgaben.

9.3 SNMP Fehlercodes

Hex	Dec	Description
0x9001	36865	INCORRECT PARAMETER SIZE
0x9002	36866	INVALID PARAMETER

Mehr Informationen:
www.beckhoff.de/ts6310

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