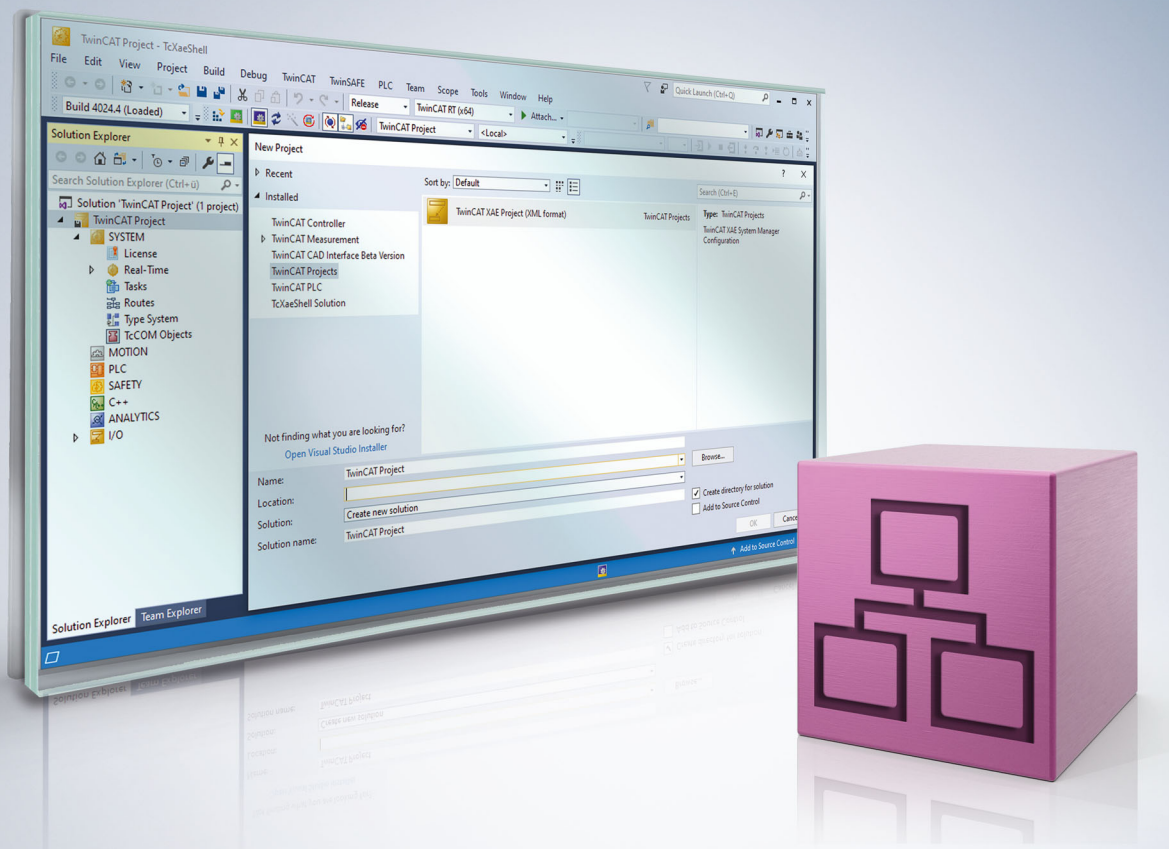
BECKHOFF New Automation Technology

Handbuch | DE

TF6310

TwinCAT 3 | TCP/IP



08.03.2023 | Version: 1.4.3

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 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

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.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

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 eines oder mehrerer TCP/IP Server/Clients in der TwinCAT SPS. Hierdurch erhält ein SPS-Programmierer die Möglichkeit, eigene Netzwerkprotokolle der Anwendungsschicht (OSI-Modell) direkt in einem SPS-Programm zu entwickeln. Die Kommunikationsverbindung kann hierbei optional auch über TLS abgesichert werden.

Produktkomponenten

Die Function TF6310 TCP/IP besteht aus den folgenden Komponenten, welche automatisch beim Setup installiert werden:

- **SPS-Bibliothek:** Tc2_TcpIp-Bibliothek (implementiert Basisfunktionalitäten wie TCP/IP und UDP/IP).
- **Hintergrundprogramm:** TwinCAT TCP/IP Connection Server (für Kommunikation).

2.1 Vergleich TF6310 TF6311

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

Auf dieser Seite werden Gemeinsamkeiten und Unterschiede der Produkte gegenübergestellt:

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

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

3 Installation

3.1 Systemvoraussetzungen

Die folgenden Systemvoraussetzungen müssen für eine ordnungsgemäße Funktion der Function TF6310 TCP/IP erfüllt sein.

Technische Daten	Beschreibung
Betriebssystem	Windows 7, 10 Windows CE 6/7 Windows Embedded Standard 2009 Windows Embedded 7 TwinCAT/BSD
Zielplattformen	PC-Architektur (x86, x64, ARM)
TwinCAT-Version	TwinCAT2, TwinCAT3
TwinCAT-Installationslevel	TwinCAT2 CP, PLC, NC-PTP TwinCAT3 XAE, XAR, ADS
Benötigte TwinCAT-Lizenz	TS6310 (für TwinCAT2) TF6310 (für TwinCAT3)

Unterstützung von TLS

Bitte beachten Sie, dass die TLS-Funktionsbausteine nicht unter Windows CE zur Verfügung stehen.

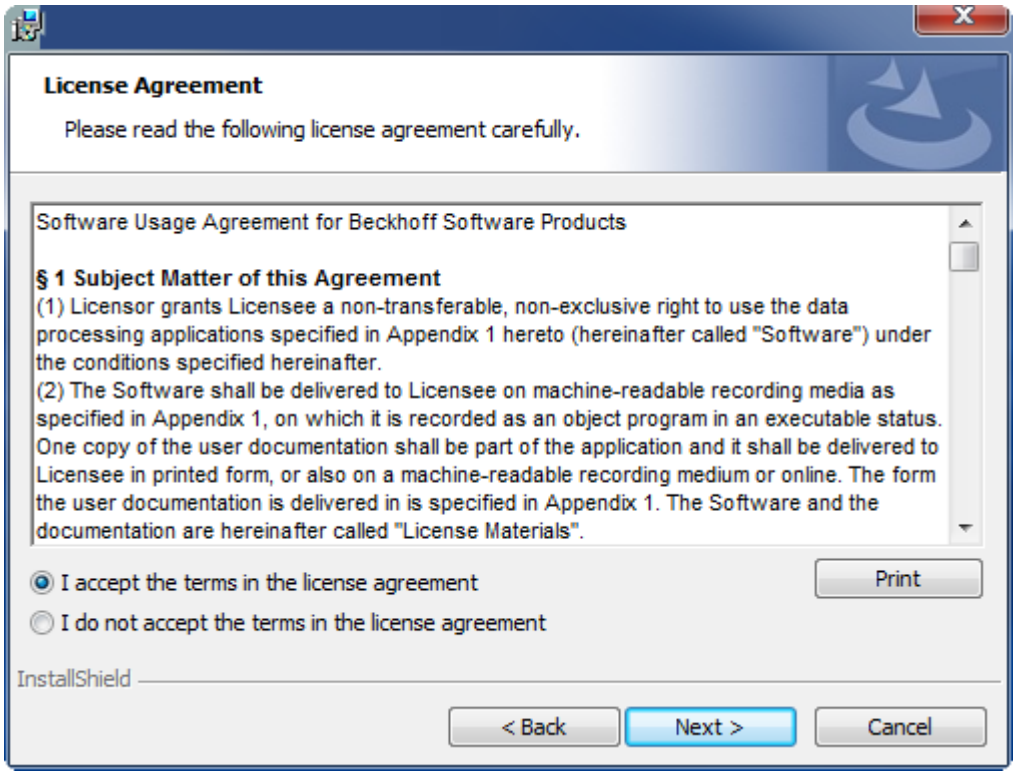
3.2 Installation

Nachfolgend wird beschrieben, wie die TwinCAT 3 Function für Windows-basierte Betriebssysteme installiert wird.

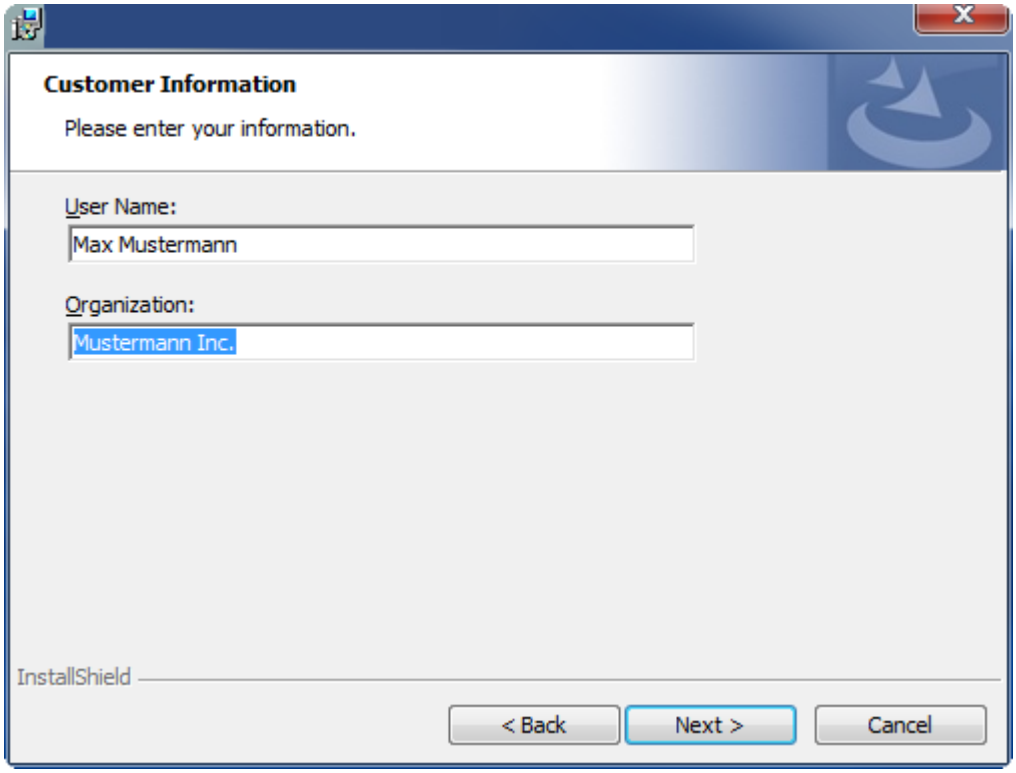
✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.

1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.
   ⇨ Der Installationsdialog öffnet sich.

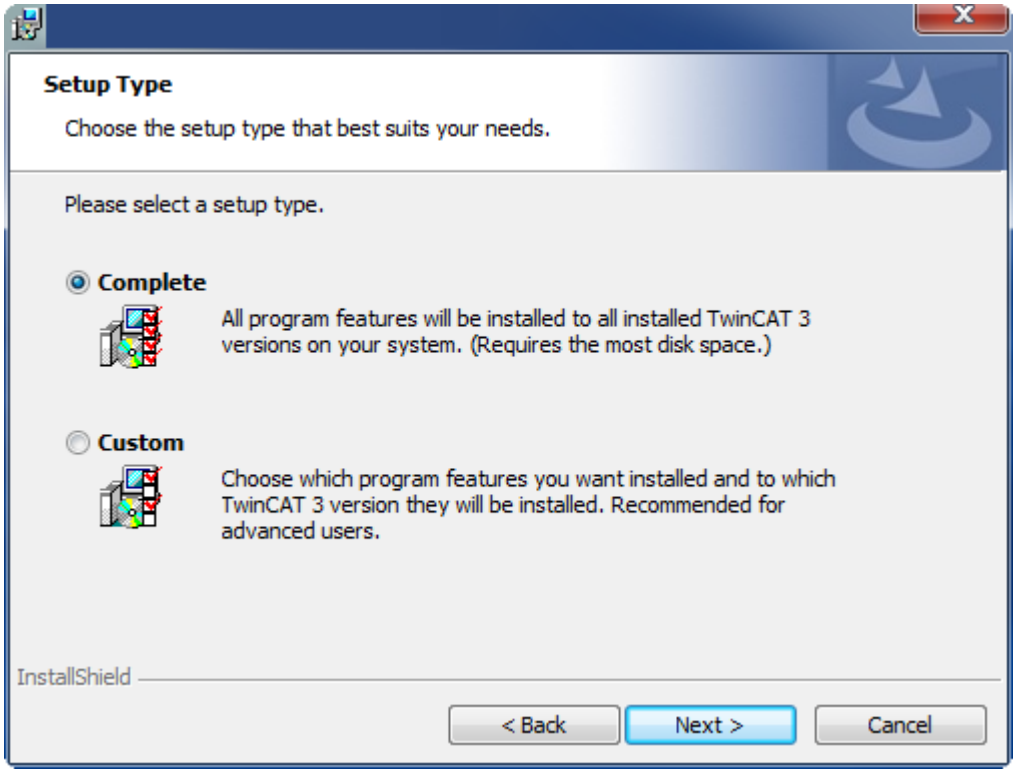
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.



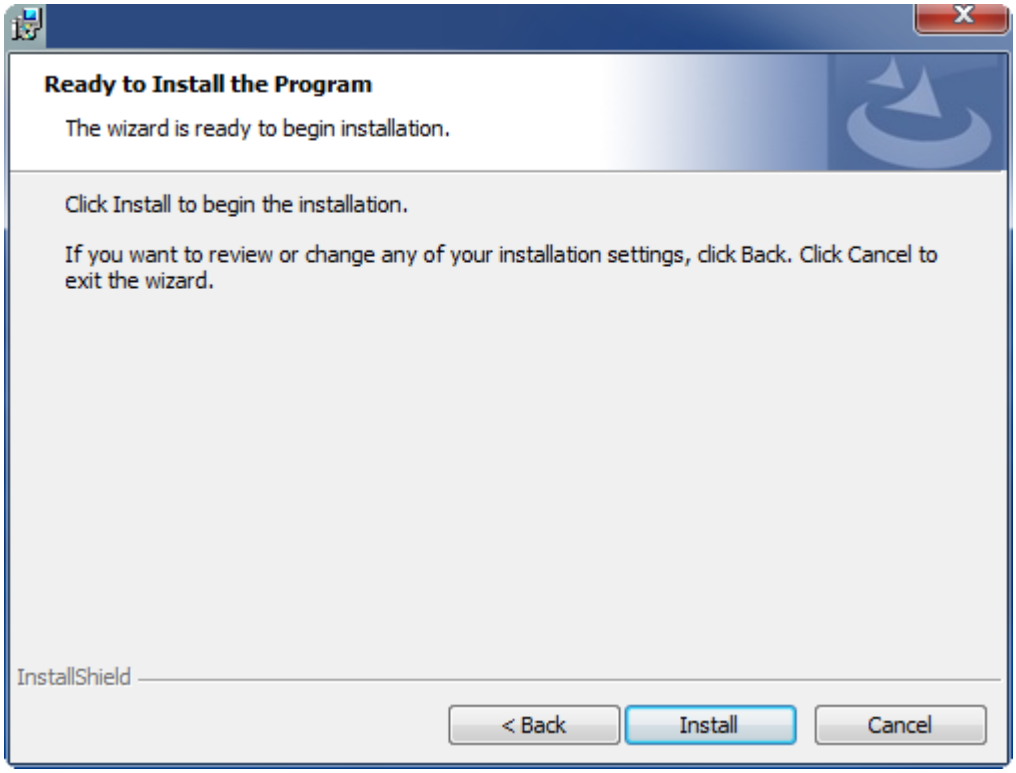
3. Geben Sie Ihre Benutzerdaten ein.



4. Wenn Sie die TwinCAT 3 Function vollständig installieren möchten, wählen Sie **Complete** als Installationstyp. Wenn Sie die Komponenten der TwinCAT 3 Function separat installieren möchten, wählen Sie **Custom**.

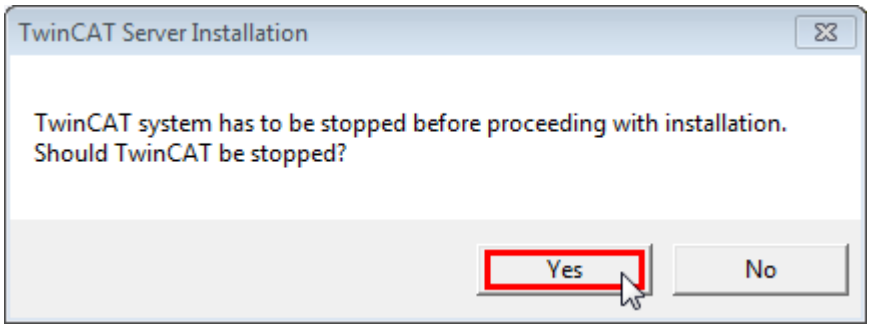


5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.

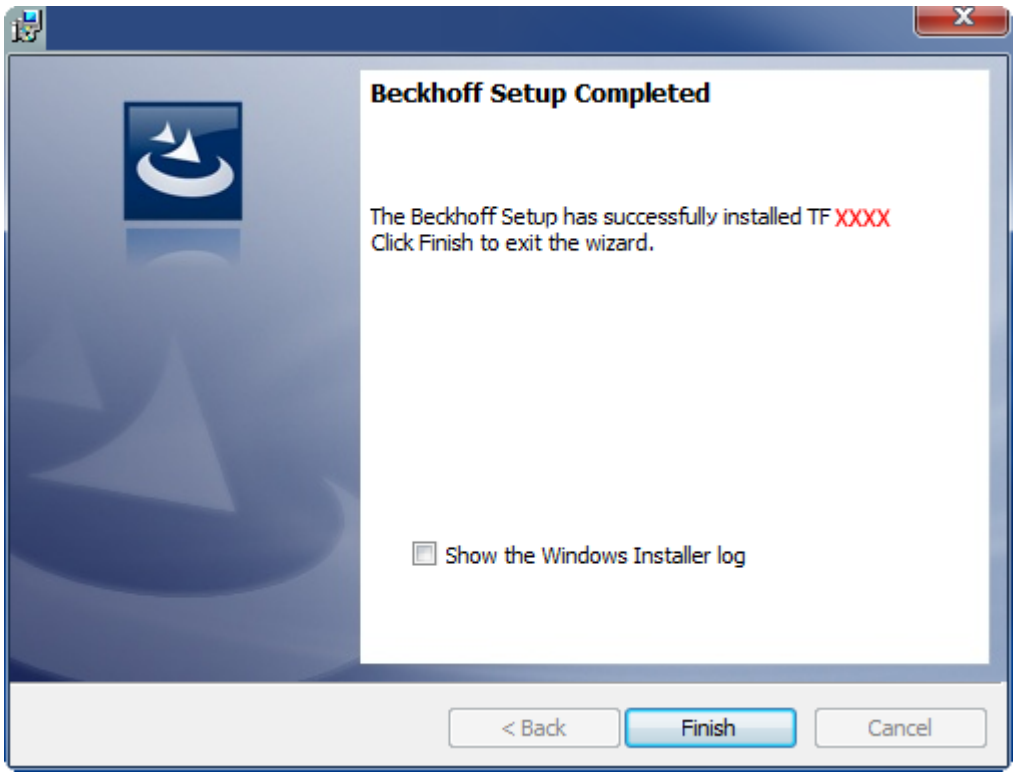


⇨ Ein Dialog weist Sie darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

6. Bestätigen Sie den Dialog mit **Yes.**



7. Wählen Sie **Finish**, um das Setup zu beenden.



⇨ Die TwinCAT 3 Function wurde erfolgreich installiert und kann lizenziert werden (siehe Lizenzierung [▶ 14]).

3.3 Installation Windows CE

In diesem Abschnitt wird beschrieben, wie die TwinCAT 3 Function TF6310 TCP/IP auf einem Beckhoff Embedded Controller mit Windows CE installiert werden kann.

Der Setup-Prozess besteht aus vier Schritten:

- Download der Setup-Datei [▶ 13]
- Installation auf einem Host-Computer [▶ 13]
- Übertragung der ausführbaren Datei auf das Windows-CE-Gerät [▶ 13]
- Installation der Software [▶ 13]

Im letzten Abschnitt finden Sie Hinweise zum Upgrade [▶ 14].

Download der Setup-Datei

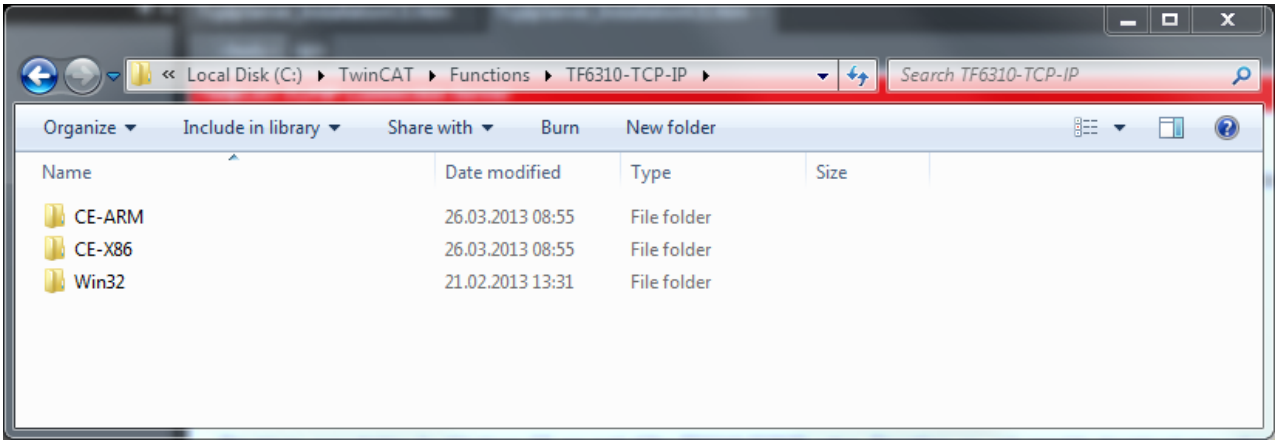
Die CAB-Installationsdatei für Windows CE ist Teil des TF6310 TCP/IP Setups. Daher müssen Sie nur das entsprechende Setup von www.beckhoff.com beziehen, welches automatisch alle Versionen für Windows XP, Windows 7 und Windows CE (x86 und ARM) enthält.

Die Installationsbeschreibung für das TF6310 TCP/IP Setup ist in unserer regulären Installationsbeschreibung enthalten (siehe Installation [▶ 9]).

Installation auf einem Host-Computer

Nach der Installation enthält der Installationsordner drei Verzeichnisse - jeweils ein Verzeichnis pro Hardwareplattform:

- **CE-ARM:** ARM-basierte Embedded Controller, welche unter Windows CE laufen, z. B. CX8090, CX9020
- **CE-X86:** X86-basierte Embedded Controller, welche unter Windows CE laufen, z. B. CX50xx. CX20x0
- **Win32:** Embedded Controller, welche unter Windows XP, Windows 7 oder Windows Embedded Standard laufen



Die Verzeichnisse CE-ARM und CE-X86 enthalten die CAB-Dateien der TF6310 Function für Windows CE in Bezug auf die jeweilige Hardwareplattform Ihres Windows-CE-Geräts. Die Datei muss auf das Windows-CE-Gerät kopiert werden.

Übertragung der ausführbaren Datei auf das Windows-CE-Gerät

Übertragen Sie die ausführbare Datei auf Ihr Windows-CE-Gerät. Zur Dateiübertragung stehen Ihnen mehrere Wege offen:

- über Netzwerkfreigaben
- über den integrierten FTP-Server
- über ActiveSync
- über CF/SD-Karten

Für weitere Informationen konsultieren Sie den Windows-CE-Bereich im Beckhoff Information System.

Installation der Software

Nachdem die CAB-Datei auf das Windows-CE-Gerät übertragen wurde, führen Sie die Datei dort aus. Den Installationsdialog können Sie mit **OK** bestätigen. Nachdem die Installation beendet wurde, starten Sie das CE-Gerät neu.

Nachdem das Gerät neu gestartet wurde, werden die ausführbaren Dateien der Function TF6310 automatisch im Hintergrund geladen.

Die Software wird in dem folgenden Verzeichnis auf dem CE-Gerät installiert:

\Hard Disk\TwinCAT\Functions\TF6310-TCP-IP

Hinweise zum Upgrade

Falls Sie schon eine ältere TF6310-Version auf dem Windows-CE-Gerät installiert haben, müssen Sie die folgenden Schritte auf dem Windows-CE-Gerät durchführen, um auf eine neuere Version zu upgraden:

1. Öffnen Sie den CE Explorer, indem Sie auf **Start > Run** klicken und „explorer“ eingeben.
2. Navigieren Sie nach *\Hard Disk\TwinCAT\Functions\TF6310-TCP-IP\Server.*
3. Benennen Sie die Datei *TcpIpServer.exe* in *TcpIpServer.old* um.
4. Starten Sie das Windows-CE-Gerät neu.
5. Übertragen Sie die neue CAB-Datei auf das Windows-CE-Gerät
6. Führen Sie die CAB-Datei auf dem CE-Gerät aus und installieren Sie die neue Version.
7. Löschen Sie die Datei *TcpIpServer.old*.
8. Starten Sie das Windows-CE-Gerät neu.

⇨ Nachdem der Neustart durchgeführt wurde, ist die neue Version aktiv.

3.4 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT 3 Lizenzierung“.

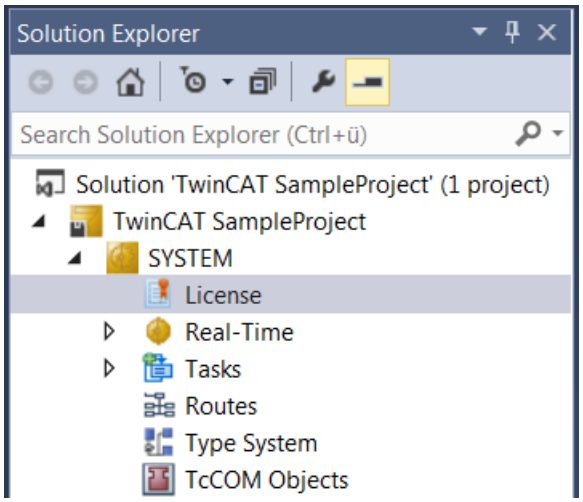
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

Eine 7-Tage-Testversion kann nicht für einen TwinCAT 3 Lizenzdongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.

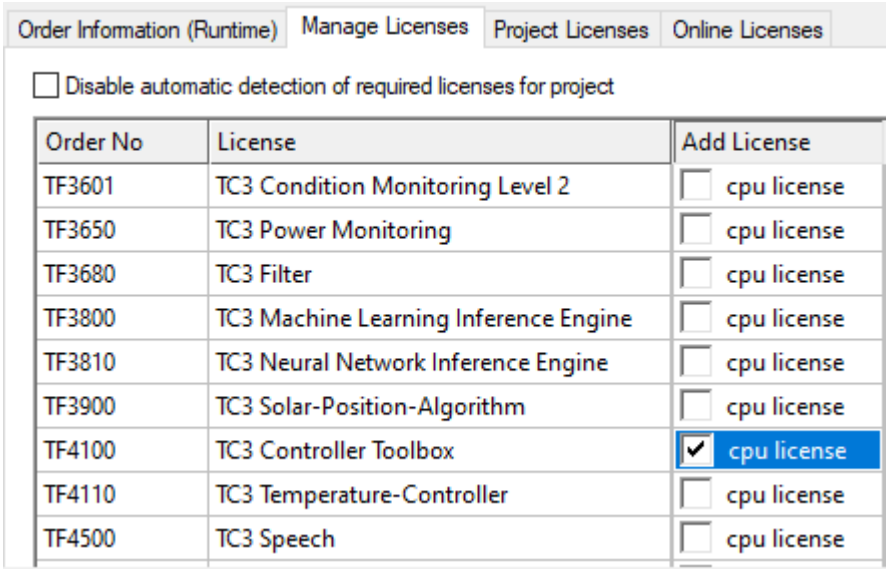
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.

4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

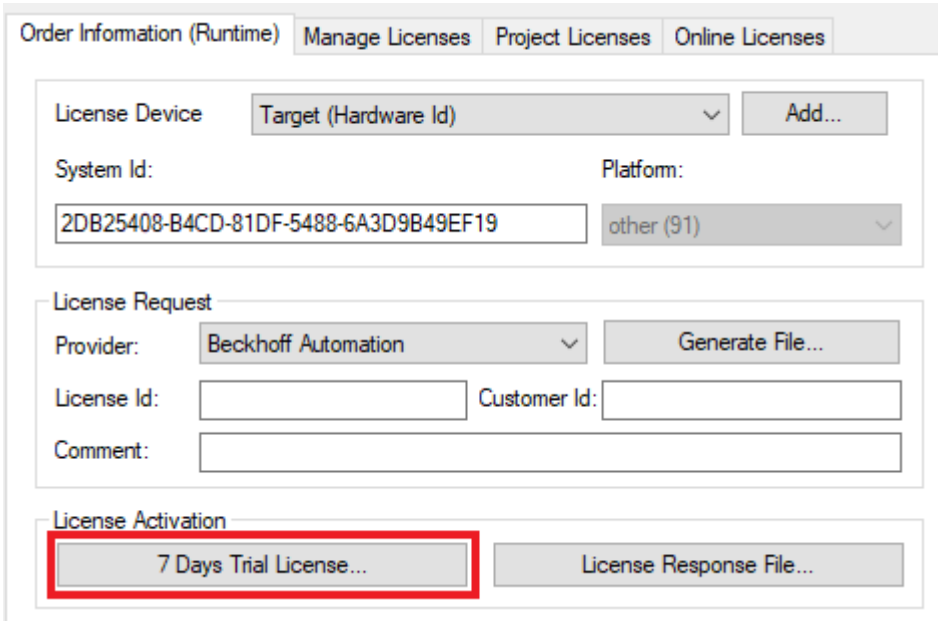
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



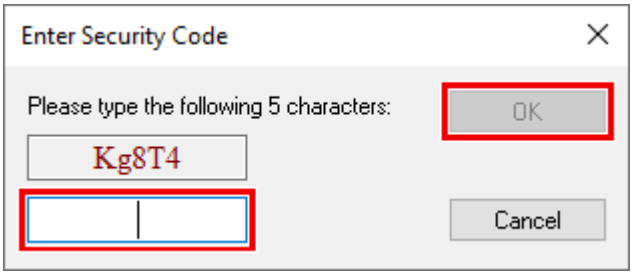
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

3.5 Migration von TwinCAT 2

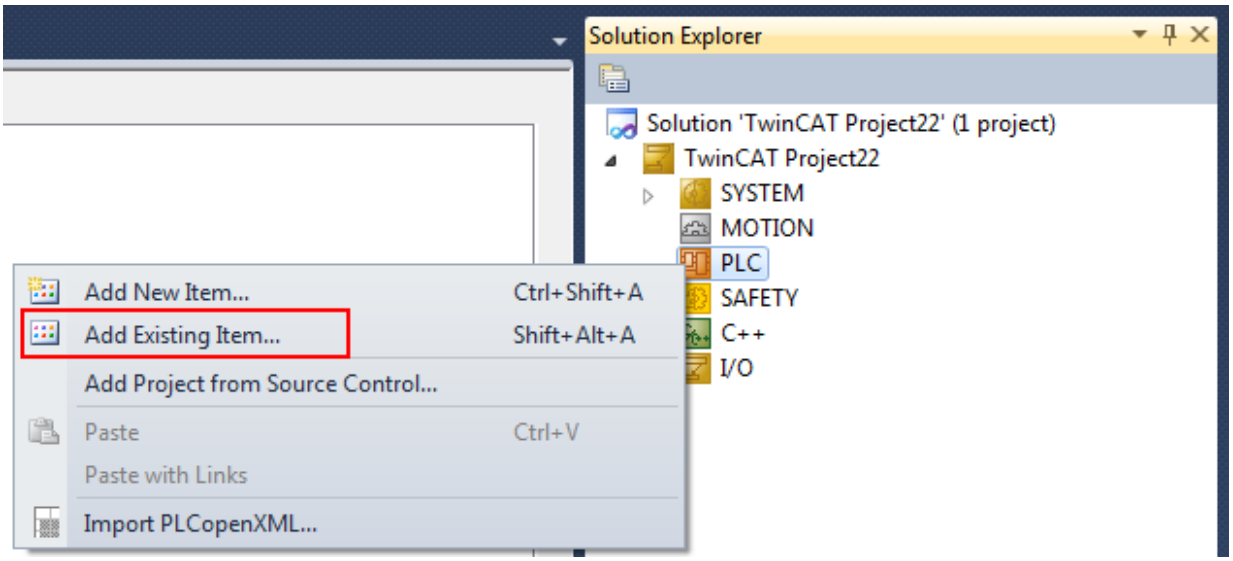
Wenn Sie ein existierendes TwinCAT-2-PLC-Projekt migrieren wollen, das eine der TCP/IP-Server-Bibliotheken der SPS nutzt, muss durch einige manuelle Schritte sichergestellt werden, dass der TwinCAT-3-SPS-Konverter die Projektdatei aus TwinCAT 2 (*.pro) verarbeiten kann. In TwinCAT 2 wird die Function TCP/IP Server mit drei SPS-Bibliotheken geliefert:

- TcpIp.lib
- TcSocketHelper.lib
- TcSnmp.lib

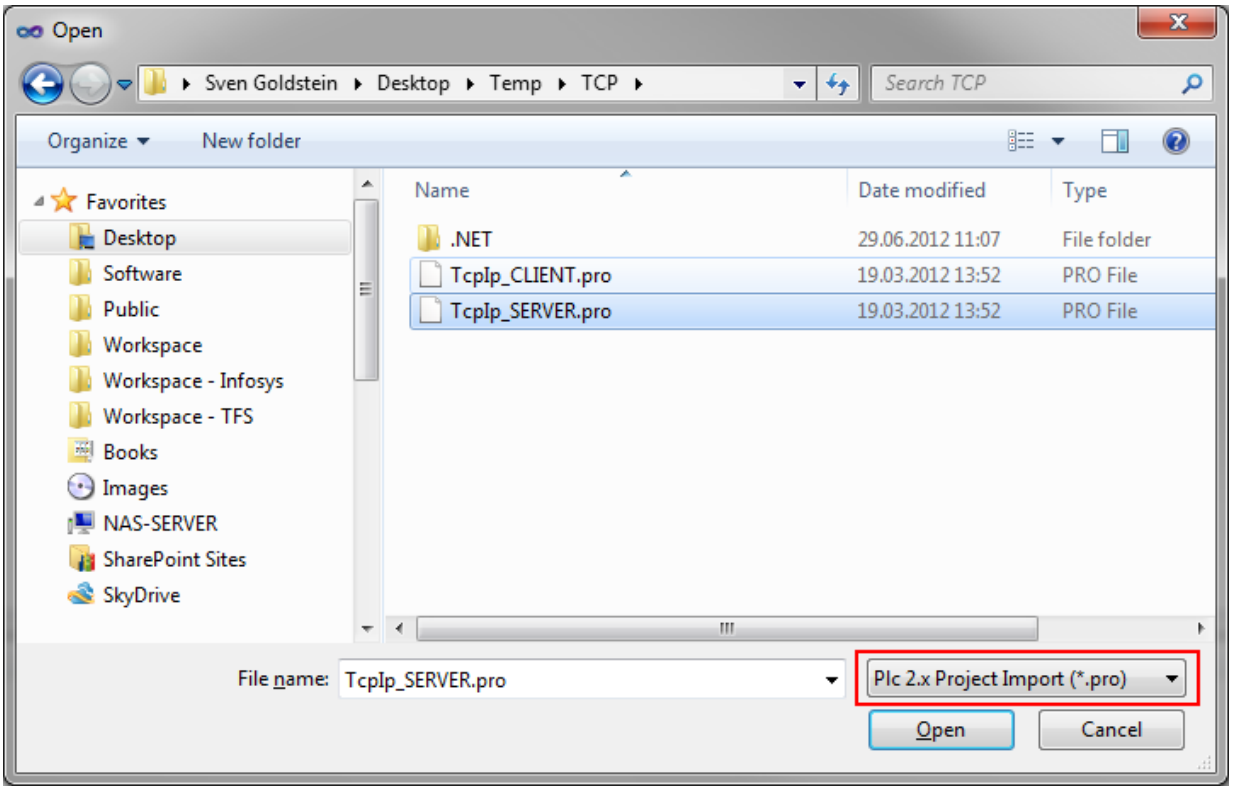
Normalerweise werden diese Dateien im Verzeichnis *C:\TwinCAT\Plc\Lib* installiert. Abhängig von der in Ihrem SPS-Projekt genutzten Bibliothek müssen Sie die entsprechende Datei nach *C:\TwinCAT3\Components\Plc\Converter\Lib* kopieren und folgende Schritte ausführen:

1. Öffnen Sie das TwinCAT Engineering.
2. Erzeugen Sie eine neue TwinCAT-3-Projektmappe.

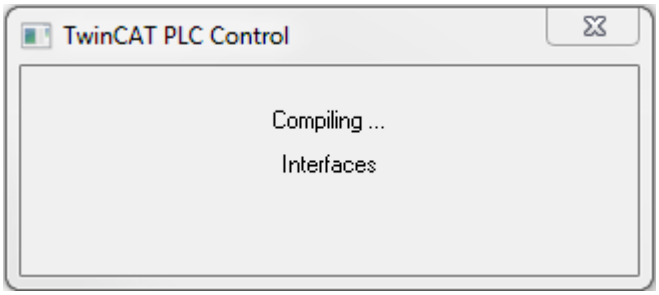
3. Klicken Sie mit der rechten Maustaste auf den Knoten „PLC“ und wählen Sie in dem sich öffnenden Kontextmenü **Vorhandenes Element hinzufügen**.

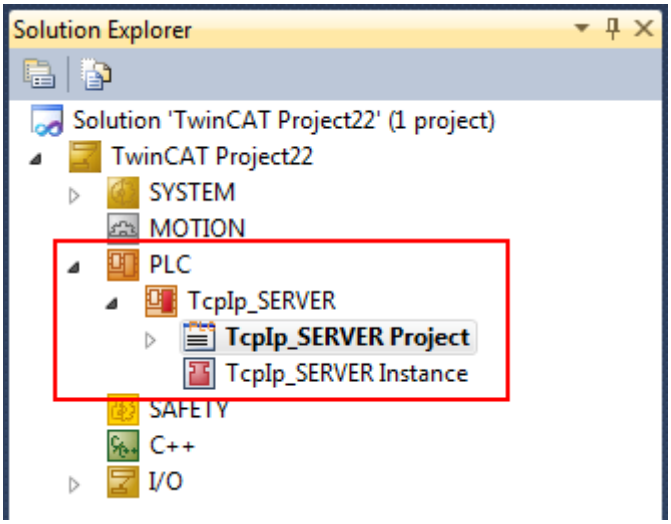


4. Wählen Sie im Dialog **Öffnen** den Dateityp „Plc 2.x Projektimport (*.pro)“ aus, öffnen Sie das Verzeichnis mit Ihrem TwinCAT-2-SPS-Projekt, wählen Sie die entsprechende .pro-Datei aus und klicken Sie auf **Öffnen.**



⇨ TwinCAT 3 startet die Konvertierung und zeigt anschließend das konvertierte SPS-Projekt im Knoten „PLC“ an.





4 Technische Einführung

In diesem Abschnitt finden Sie einen generellen Überblick zu den Übertragungsprotokollen TCP und UDP und einen Link zu den entsprechenden SPS-Bibliotheken, die zum Einbinden der Protokolle erforderlich sind. Beide Übertragungsprotokolle sind Bestandteil der Internet Protocol Suite und daher für unsere alltägliche Kommunikation, z. B. über Internet, von großer Bedeutung.

Transmission Control Protocol (TCP)

Bei dem TCP-Protokoll handelt es sich um ein verbindungsorientiertes Übertragungsprotokoll (OSI Layer 4), vergleichbar mit einer Telefonverbindung, wo Gesprächsteilnehmer erst eine Verbindung aufbauen müssen, bevor Daten übertragen werden können. Über TCP können Datenströme (Bytes) auf Anforderung zuverlässig übertragen werden, daher wird in diesem Zusammenhang auch von einem „Datenstrom-orientiertem Übertragungsprotokoll“ gesprochen. Das TCP-Protokoll wird in Netzwerken eingesetzt, wo für die von einem Client oder Server gesendeten Daten eine Bestätigung des gegenüberliegenden Gesprächspartners erforderlich ist. Das TCP-Protokoll ist gut geeignet, um größere Datenmengen oder Datenströme ohne definierte Start/Ende-Kennung zu übertragen. Für den Sender ist dies unproblematisch, da er weiß, wie viele Datenbytes er verschickt hat. 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. Eine mögliche Nachrichtenstruktur, die oft für die Übertragung von Binärdaten mit einer variablen Länge genutzt wird, kann wie folgt definiert werden: In den ersten Datenbytes wird ein spezielles Steuerzeichen (ein sogenannter start delimiter) und die Datenlänge der darauffolgenden Daten eingetragen. Der Empfänger kann dadurch den Nachrichtenanfang und das Ende erkennen.

TCP/IP Client

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

- Für das Aufbauen und Abbauen der Verbindung zum Remote-Server jeweils eine Instanz der Funktionsbausteine FB_SocketConnect [▶ 21] und FB_SocketClose [▶ 22] (Tipp: Der Funktionsbaustein FB_ClientServerConnection [▶ 45] vereint die Funktionalität beider Funktionsbausteine)
- Für den Datenaustausch (Senden und Empfangen) mit dem Remote-Server jeweils eine Instanz des Funktionsbausteins FB_SocketSend [▶ 27] und/oder FB_SocketReceive [▶ 28]

TCP/IP Server

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

- Für das Öffnen des Listener-Sockets eine Instanz des Funktionsbausteins FB_SocketListen [▶ 24]
- Für das Aufbauen und Abbauen der Verbindung/-en zu den Remote-Clients jeweils eine Instanz der Funktionsbausteine FB_SocketAccept [▶ 25] und FB_SocketClose [▶ 22] (Tipp: FB_ServerClientConnection [▶ 47] vereint die Funktionalität aller drei Funktionsblöcke)
- Für den Datenaustausch (Senden und Empfangen) mit den Remote-Clients jeweils eine Instanz des Funktionsbausteins FB_SocketSend [▶ 27] und/oder FB_SocketReceive [▶ 28]
- In jedem SPS-Runtime-System, in dem Sie ein Socket öffnen, jeweils eine Instanz des Funktionsbausteins FB_SocketCloseAll [▶ 23]

Die Instanzen der Funktionsbausteine FB_SocketAccept [▶ 25] und FB_SocketReceive [▶ 28] werden zyklisch (pollend) aufgerufen, alle anderen nach Bedarf.

User Datagram Protocol (UDP)

UDP ist ein verbindungsloses Protokoll, d. h. Daten werden ohne explizite Verbindung zwischen Netzwerkgeräten versendet. UDP nutzt ein einfaches Übertragungsmodell ohne spezielle Definition für Handshake, Zuverlässigkeit, Datenanforderung oder Stauüberwachung. Auch wenn die obige Beschreibung nahe legt, dass UDP-Datagramme unangefordert oder doppelt ankommen oder zu Staus in der Datenleitung führen, wird das Protokoll in einigen Fällen gegenüber TCP bevorzugt, besonders bei der Echtzeit-Kommunikation, da die TCP-Merkmale mehr Rechenleistung und damit auch mehr Zeit beanspruchen. Wegen der Verbindungslosigkeit ist das UDP-Protokoll gut geeignet, kleine Datenmengen zu verschicken. UDP ist ein „paketorientiertes/nachrichtenorientiertes Transportprotokoll“, d. h. der gesendete Datenblock wird auf der Empfängerseite auch als kompletter Datenblock empfangen.

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

- Für das Öffnen und Schließen eines UDP-Sockets jeweils eine Instanz der Funktionsbausteine FB_SocketUdpCreate [▶ 29] und FB_SocketClose [▶ 22] (Tipp: FB_ConnectionlessSocket [▶ 50] vereint die Funktionalität beider Funktionsbausteine)
- Für den Datenaustausch (Senden und Empfangen) mit anderen Teilnehmern jeweils eine Instanz des Funktionsbausteins FB_SocketUdpSendTo [▶ 31] und/oder FB_SocketUdpReceiveFrom [▶ 33]
- In jedem SPS-Runtime-System, in dem Sie ein UDP-Socket öffnen, jeweils eine Instanz des Funktionsbausteins FB_SocketCloseAll [▶ 23]

Die Instanzen des Funktionsbausteins FB_SocketUdpReceiveFrom [▶ 33] werden zyklisch (pollend) aufgerufen, alle anderen nach Bedarf.

Siehe auch: Beispiele [▶ 65]

5 SPS API

5.1 Funktionsbausteine

5.1.1 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. Beim Erfolg wird ein neuer Socket geöffnet und am hSocket-Ausgang das dazugehörige Verbindungshandle zurückgeliefert. Das Verbindungshandle wird dann z. B. von den Funktionsbausteinen FB_SocketSend [▶ 27] und FB_SocketReceive [▶ 28] benötigt, um mit einem Remote-Server Daten austauschen zu können. Eine nicht mehr benötigte Verbindung wird mit dem Funktionsbaustein FB_SocketClose [▶ 22] 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.

Eingänge

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    sRemoteHost : T_IPv4Addr := '';
    nRemotePort : UDINT;
    bExecute    : BOOL;
    tTimeout    : TIME := T#45s;(*!!!*)
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
sRemoteHost	T_IPv4Addr	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	UDINT	IP-Portnummer des Remote-Servers (z. B. 200).
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Maximale Ausführungszeit des Funktionsbausteins einstellen

Setzen Sie den Wert „tTimeout“ nicht zu niedrig, da bei einer Netzwerkunterbrechung Timeout-Zeiten 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.

Ausgänge

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
```

```
    nErrId     : UDINT;
    hSocket    : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	TCP/IP-Verbindungshandle [▶ 60] zum neu geöffneten Local-Client Socket.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.2 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 [▶ 24], ein Local-Client-Socket mit FB_SocketConnect [▶ 21] und ein Remote-Client-Socket mit FB_SocketAccept [▶ 25] geöffnet.

UDP: Der UDP-Socket wird mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 29] geöffnet.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	• TCP/IP: Verbindungshandle [▶ 60] des zu schließenden Listener-, Remote- oder Local-Client-Sockets. • UDP: Verbindungshandle des UDP-Socket.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.3 FB_SocketCloseAll

Beim TwinCAT-Restart oder TwinCAT-Stop wird auch der TwinCAT TCP/IP Connection Server gestoppt. Alle bereits geöffneten Sockets (TCP/IP- und UDP-Verbindungshandles) werden 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 Runtime-Systems (Port 801) aufrufen, werden alle Sockets geschlossen, die in dem ersten Runtime-System geöffnet wurden. In jedem SPS-Runtime-System, in dem die Socket-Funktionsbausteine benutzt werden, sollte eine Instanz von FB_SocketCloseAll beim SPS-Start aufgerufen werden.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Beispiel für eine Implementierung in ST

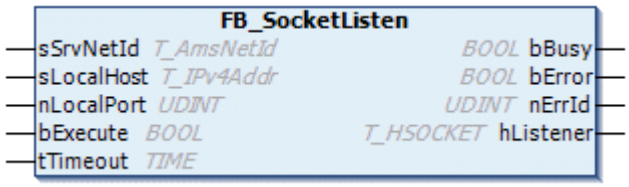
Durch den folgenden Programmcode werden die vor einem „SPS Reset“ oder „Download“ geöffneten Verbindungshandles (Sockets) beim erneuten SPS-Start ordnungsgemäß geschlossen.

```
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_IF
IF NOT fbSocketCloseAll.bBusy THEN
(*...
... continue program execution...
...*)
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.4 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 nach ankommenden Verbindungsanforderungen von Remote-Clients horchen. Beim Erfolg wird am hListner-Ausgang das dazugehörige Verbindungshandle zurückgeliefert. Dieses Handle wird von dem Funktionsbaustein FB_SocketAccept [▶ 25] benötigt. Ein nicht mehr benötigter Listener-Socket wird mit dem Funktionsbaustein FB_SocketClose [▶ 22] geschlossen. Auf einem Rechner kann nur ein Listener-Socket mit der gleichen IP-Portnummer geöffnet werden.

Eingänge

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    sLocalHost  : T_IPv4Addr := '';
    nLocalPort  : UDINT;
    bExecute    : BOOL;
    tTimeout    : TIME := T#5s;
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
sLocalHost	T_IPv4Addr	Local-Server IP-Adresse (Ipv4) als String (z. B. '172.13.15.2'). Für einen Server auf dem loka-len Rechner (default) kann auch ein Leerstring angegeben werden.
nLocalPort	UDINT	Local-Server IP-Port (z. B. 200).
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

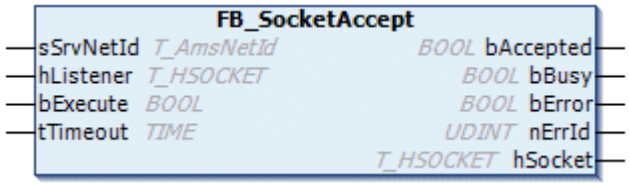
```
VAR_OUTPUT
    bBusy      : BOOL;
    bError     : BOOL;
    nErrId     : UDINT;
    hListener  : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hListener	T_HSOCKET	Verbindungshandle [▶ 60] zum neuen Listener-Socket.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.5 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 [▶ 27] und FB_SocketReceive [▶ 28] 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 [▶ 22] 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 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 einmal 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.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hListener	T_HSOCKET	Das Verbindungshandle [▶ 60] des Listener-Sockets. Dieses Handle muss vorher mit dem Funktionsbaustein FB_SocketListen [▶ 24] angefordert werden.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

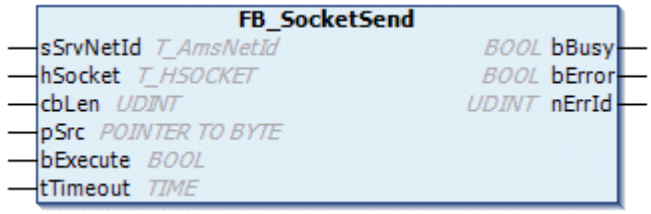
```
VAR_OUTPUT
    bAccepted : BOOL;
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    hSocket   : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
bAccepted	BOOL	Dieser Ausgang wird gesetzt, wenn eine neue Verbindung zu einem Remote-Client hergestellt wurde.
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] eines neuen Remote-Clients.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.6 FB_SocketSend



Mit dem Funktionsbaustein FB_SocketSend können über den TwinCAT TCP/IP Connection Server Daten zu Remote-Clients oder Remote-Servers gesendet werden. Eine Remote-Clientverbindung muss vorher mit dem Funktionsbaustein FB_SocketAccept [▶ 25] oder eine Remote-Serververbindung mit dem Funktionsbaustein FB_SocketConnect [▶ 21] aufgebaut werden.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Das Verbindungshandle [▶ 60] des Kommunikationspartners zu dem Daten gesendet werden sollen.
cbLen	UDINT	Anzahl der zu sendenden Daten in Bytes.
pSrc	POINTER TO BYTE	Adresse (Pointer) des Sendepuffers.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausführungszeit des Funktionsbausteins einstellen

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.

Ausgänge

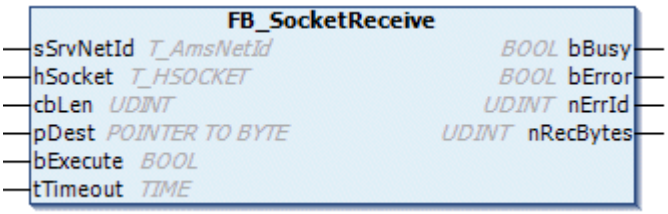
```
VAR_OUTPUT
    bBusy      : BOOL;
    bError     : BOOL;
    nErrId     : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.7 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 [▶ 25] und eine Remote-Serververbindung mit dem Funktionsbaustein FB_SocketConnect [▶ 21] 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 100 ms 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.

Timeout-Wert einstellen

Wenn der Remote-Teilnehmer vom TCP/IP-Netzwerk getrennt wurde (nur auf der Remote-Seite) und der lokale Teilnehmer noch mit dem TCP/IP-Neztwerk verbunden ist, dann liefert der FB_SocketReceive-Funktionsbaustein keinen Fehler und keine Daten. Der geöffnete Socket existiert immer noch, es werden nur keine Daten empfangen. Die Anwendung wartet in diesem Fall möglicherweise ewig auf Daten. Es wird empfohlen, in die SPS Anwendung eine Timeout-Überwachung zu implementieren. Wenn nach einer bestimmter Zeit z. B. 10 Sekunden immer noch nicht alle Daten empfangen wurde, muss die Verbindung geschlossen und neu initialisiert werden.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] des Kommunikationspartners, dessen Daten empfangen werden sollen.
cbLen	UDINT	Maximal verfügbare Puffergröße für die zu lesenden Daten in Bytes.
pDest	POINTER TO BYTE	Adresse (Pointer) des Empfangspuffers.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

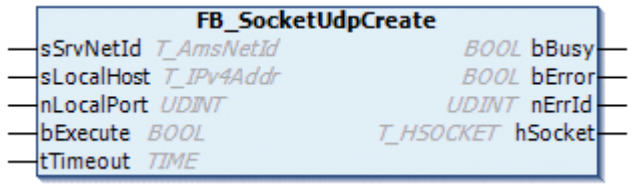
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    nRecBytes : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connec-tion Server Fehlernummer.
nRecBytes	UDINT	Die Anzahl der zuletzt erfolgreich empfangenen Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.8 FB_SocketUdpCreate



Mit dem Funktionsbaustein FB_SocketUdpCreate kann ein Client/Serverr-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 [▶ 31] und FB_SocketUdpReceiveFrom [▶ 33] benötigt, um mit einem Remote-Teilnehmer Daten austauschen zu können. Ein nicht mehr benötigter UDP-Socket kann mit dem Funktionsbaustein FB_SocketClose [▶ 22] 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.

Automatisch erstellte Netzwerkverbindungen

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'.

Automatisch erstellte Netzwerkverbindungen mit mehreren Netzwerkadaptern

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.

Festlegen einer Netzwerkadresse

Um das Öffnen der Sockets unter einer anderen IP-Adresse zu verhindern, können Sie die sLocalHost-Adresse explizit angeben oder die zurückgelieferte Adresse in der Handle-Variable (hSocket) überprüfen, den Socket schließen und erneut öffnen.

Eingänge

```
VAR_INPUT
    sSrvNetId  : T_AmsNetId := '';
    sLocalHost : T_IPv4Addr := '';
    nLocalPort : UDINT;
    bExecute   : BOOL;
    tTimeout   : TIME:= T#5s;
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
sLocalHost	T_IPv4Addr	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	UDINT	Lokale IP-Portnummer des UDP-Client/Server-Sockets (z. B. 200).
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

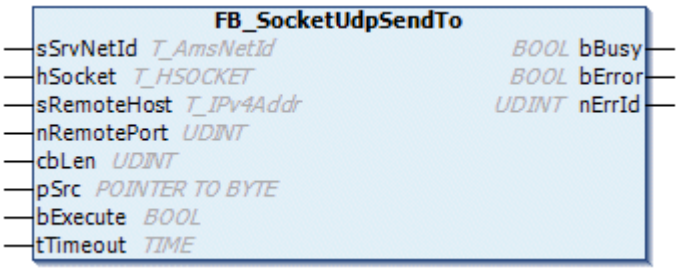
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    hSocket   : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	Handle des neu geöffneten UDP-Client/Server-Sockets [▶ 60].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.9 FB_SocketUdpSendTo



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 [▶ 29] geöffnet werden.

Eingänge

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId := '';
    hSocket     : T_HSOCKET;
    sRemoteHost : T_IPv4Addr;
    nRemotePort : UDINT;
    cbLen       : UDINT;
    pSrc        : POINTER TO BYTE;
    bExecute    : BOOL;
    tTimeout    : TIME := T#5s;
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Handle eines geöffneten UDP-Sockets [▶ 60].
sRemoteHost	T_IPv4Addr	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	UDINT	IP-Portnummer des Remote-Teilnehmers, an den Daten gesendet werden sollen (z. B. 200).
cbLen	UDINT	Anzahl der zu sendenden Daten in Bytes. Die maximale Anzahl der zu sendenden Datenbytes ist auf 8192 Bytes begrenzt (Konstante TCPADS_MAXUDP_BUFFSIZE in der Bibliothek, um den Speicherplatz zu schonen).
pSrc	POINTER TO BYTE	Adresse (Pointer) des Sendepuffers.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Größe der empfangenen Daten-Bytes einstellen

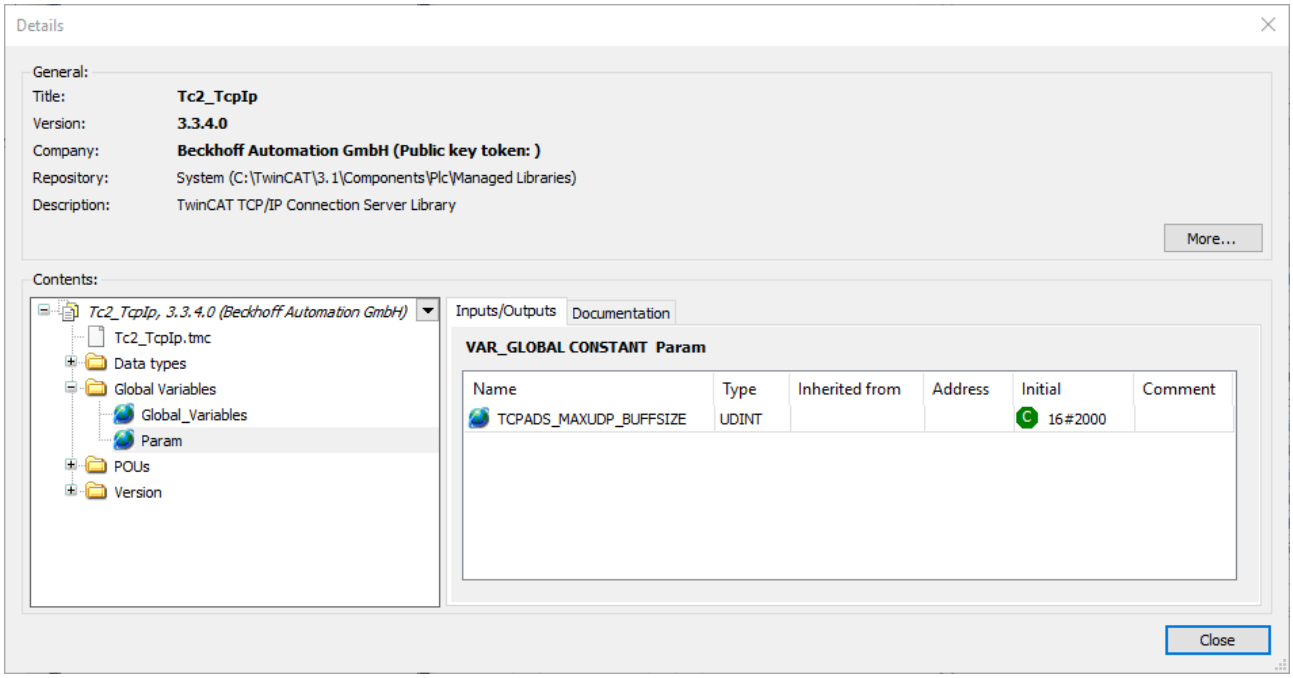
Verfügbar in Produktversion: TwinCAT TCP/IP Connection Server v1.0.50 oder höher: Die maximale Anzahl der zu empfangenden Datenbytes kann erhöht werden (nur wenn absolut unumgänglich).

TwinCAT 2

1. Globale Konstante im SPS-Projekt neu definieren (im Beispiel soll die maximale Anzahl der zu empfangenden Datenbytes auf 32000 erhöht werden):
```
VAR_GLOBAL CONSTANT
    TCPADS_MAXUDP_BUFFSIZE : UDINT := 32000;
END_VAR
```
2. Option **Replace constants** im Dialog der TwinCAT-SPS-Steuerung aktivieren (Project > Options ... > Build).
3. Projekt neu erstellen.

TwinCAT 3

In TwinCAT 3 kann dieser Wert über eine Parameterliste der SPS-Bibliothek (ab Version 3.3.4.0) editiert werden.



Ausgänge

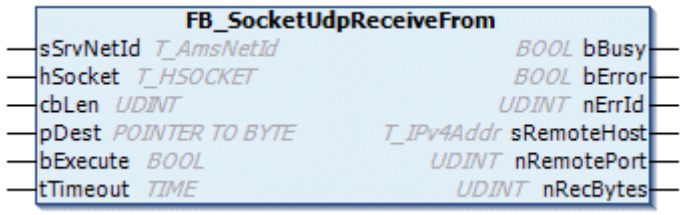
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.10 FB_SocketUdpReceiveFrom



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 [▶ 29] 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. 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.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Handle eines geöffneten UDP-Sockets [▶ 60], dessen Daten empfangen werden sollen.
cbLen	UDINT	Maximal verfügbare Puffergröße für die zu lesenden Daten in Bytes. Die maximale Anzahl der zu empfangenden Datenbytes ist auf 8192 begrenzt (Konstante TCPADS_MAXUDP_BUFFSIZE in der Bibliothek, um den Speicherplatz zu schonen).
pDest	POINTER TO BYTE	Adresse (Pointer) des Empfangspuffers.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Größe der empfangenen Daten-Bytes einstellen

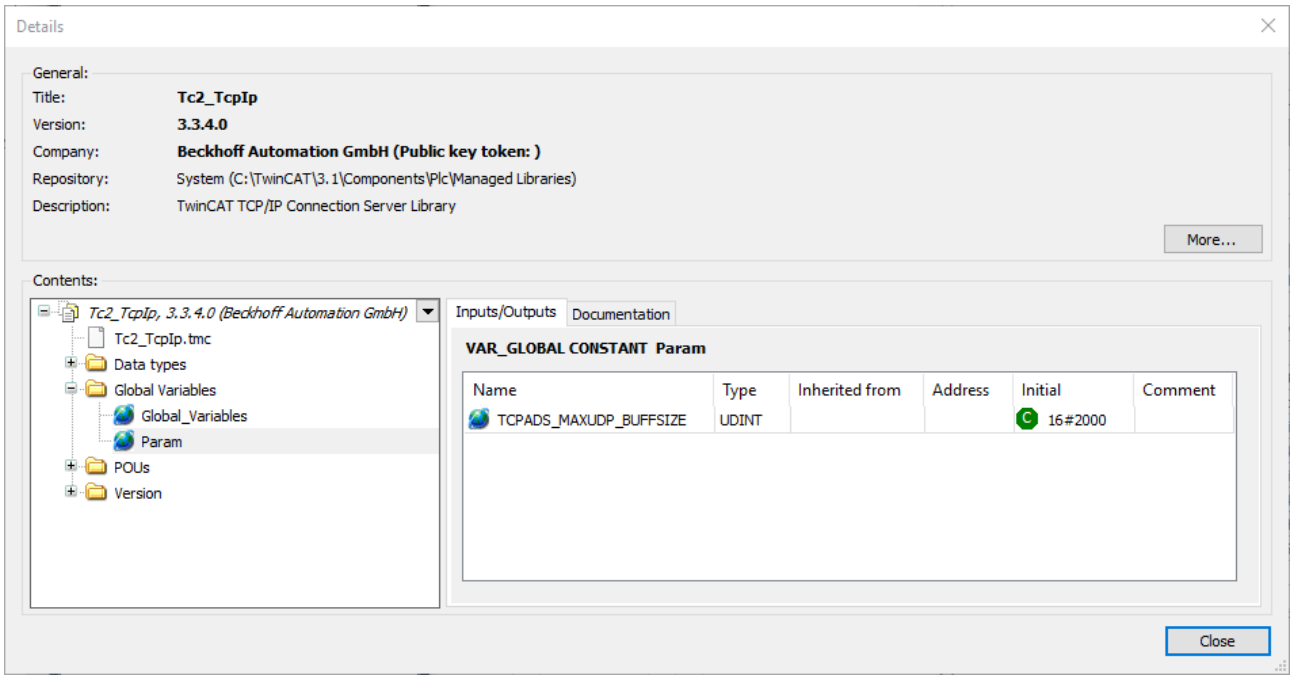
Verfügbar in Produktversion: TwinCAT TCP/IP Connection Server v1.0.50 oder höher: Die maximale Anzahl der zu empfangenden Datenbytes kann erhöht werden (nur wenn absolut unumgänglich).

TwinCAT 2

1. Globale Konstante im SPS-Projekt neu definieren (im Beispiel soll die maximale Anzahl der zu empfangenden Datenbytes auf 32000 erhöht werden):
```
VAR_GLOBAL CONSTANT
    TCPADS_MAXUDP_BUFFSIZE : UDINT := 32000;
END_VAR
```
2. Option **Replace constants** im Dialog der TwinCAT-SPS-Steuerung aktivieren (Project > Options ... > Build).
3. Projekt neu erstellen.

TwinCAT 3

In TwinCAT 3 kann dieser Wert über eine Parameterliste der SPS-Bibliothek (ab Version 3.3.4.0) editiert werden.



Ausgänge

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    sRemoteHost : T_IPv4Addr := '';
    nRemotePort : UDINT;
    nRecBytes   : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
sRemoteHost	T_IPv4Addr	Beim Erfolg die IP-Adresse (Ipv4) des Remote-Teilnehmers, dessen Daten empfangen wur-den.
nRemotePort	UDINT	Beim Erfolg die IP-Portnummer des Remote-Teilnehmers, dessen Daten empfangen wurden (z. B. 200).
nRecBytes	UDINT	Anzahl der zuletzt erfolgreich empfangen Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.11 FB_SocketUdpAddMulticastAddress

Bindet den Server an eine Multicast-IP-Adresse, sodass Multicast Pakete empfangen werden können. Dieser Funktionsbaustein erwartet eine bereits hergestellte UDP Socket-Verbindung, welche über den Funktionsbaustein FB_SocketUdpCreate [▶ 29] hergestellt werden kann.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] des Listener-Sockets. Dieses Handle muss vorher mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 29] angefordert werden.
sMulticastAddr	T_IPv4Addr	Multicast-IP-Adresse, an welche das Binding erfolgen soll.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.12 FB_SocketUdpDropMulticastAddress

Entfernt das Binding an eine Multicast-IP-Adresse, welches vorher über den Funktionsbaustein FB_SocketUdpAddMulticastAddress [▶ 35] eingerichtet wurde.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] des Listener-Sockets. Dieses Handle muss vorher mit dem Funktionsbaustein FB_SocketUdpCreate [▶ 29] angefordert werden.
sMulticastAddr	T_IPv4Addr	Multicast-IP-Adresse, an welche das Binding erfolgen soll.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

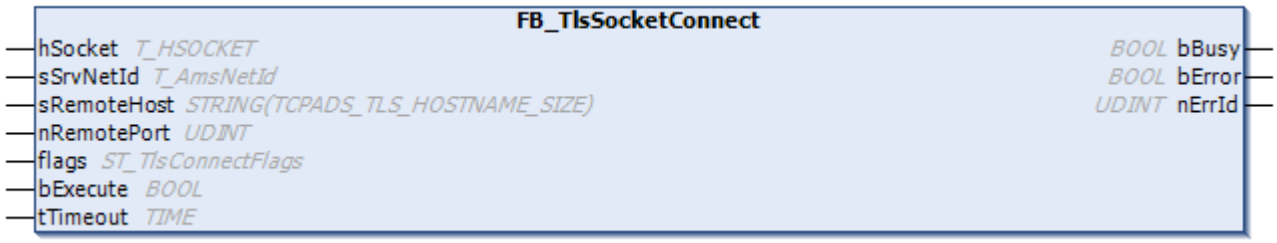
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.13 FB_TlsSocketConnect



Mit dem Funktionsbaustein FB_TlsSocketConnect kann ein Client über den TwinCAT TCP/IP Connection Server eine neue, über TLS abgesicherte TCP/IP-Verbindung zu einem Remote-Server aufbauen. Beim Erfolg wird ein neuer Socket geöffnet und am hSocket-Ausgang das dazugehörige Verbindungshandle zurückgeliefert. Das Verbindungshandle wird dann z. B. von den Funktionsbausteinen FB_SocketSend [▶ 27] und FB_SocketReceive [▶ 28] benötigt, um mit einem Remote-Server Daten austauschen zu können. Eine nicht mehr benötigte Verbindung wird mit dem Funktionsbaustein FB_SocketClose [▶ 22] 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. Die TLS-Parameter können über die Funktionsbausteine FB_TlsSocketAddCa [▶ 41], FB_TlsSocketAddCrl [▶ 42], FB_TlsSocketSetPsk [▶ 44] und FB_TlsSocketSetCert [▶ 43] definiert werden. Programmierbeispiele für deren Verwendung finden Sie in unseren Samples.

Eingänge

```
VAR_INPUT
    sSrvNetId   : T_AmsNetId:='';
    sRemoteHost : STRING(TCPADS_TLS_HOSTNAME_SIZE):='';
    nRemotePort : UDINT:=0;
    flags       : ST_TlsConnectFlags:=DEFAULT_TLSCONNECTFLAGS;
    bExecute    : BOOL;
    tTimeout    : TIME:=T#45s;(*!!!*)
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
sRemoteHost	STRING(TCPADS_TLS_HOSTNAME_SIZE)	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	UDINT	IP-Portnummer des Remote-Servers (z. B. 200).
flags	ST_TlsConnectFlags [▶ 59]	Zusätzliche (optionale) Client-Verbindungsparameter.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Maximale Ausführungszeit des Funktionsbausteins einstellen

Setzen Sie den Wert „tTimeout“ nicht zu niedrig, da bei einer Netzwerkunterbrechung Timeout-Zeiten 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.

Ein-/Ausgänge

```
VAR_IN_OUT
    hSocket : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
hSocket	T_HSOCKET	TCP/IP-Verbindungshandle [▶ 60] zum neu geöffneten Local-Client Socket

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.14 FB_TlsSocketListen

Mit dem Funktionsbaustein FB_TLsSocketListen kann über den TwinCAT TCP/IP Connection Server ein neuer, über TLS abgesicherter Listener-Socket geöffnet werden. Über einen Listener-Socket kann der TwinCAT TCP/IP Connection Server nach ankommenden Verbindungsanforderungen von Remote-Clients horchen. Das mit dem Funktionsbaustein FB_TlsSocketCreate [▶ 40] erzeugte Socket-Handle kann anschließend von dem Funktionsbaustein FB_SocketAccept [▶ 25] verwendet werden, um eine eingehende Clientanforderung zu akzeptieren. Ein nicht mehr benötigter Listener-Socket wird mit dem Funktionsbaustein

FB_SocketClose [▶ 22] geschlossen. Auf einem Rechner kann nur ein Listener-Socket mit der gleichen IP-Portnummer geöffnet werden. Programmierbeispiele zur Verwendung dieses Funktionsbausteins finden Sie in unseren Samples.

Eingänge

```
VAR_INPUT
    sSrvNetId  : T_AmsNetId:='';
    sLocalHost : T_IPv4Addr:='';
    nLocalPort : UDINT:=0;
    flags      : ST_TlsListenFlags:=DEFAULT_TLSLISTENFLAGS;
    bExecute   : BOOL;
    tTimeout   : TIME:=T#5s;
END_VAR
```

Name	Typ	Beschreibung
hListener	T_HSOCKET	Socket-Handle, welches über den Funktionsbaustein FB_TlsSocketCreate erzeugt wurde.
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
sLocalHost	T_IPv4Addr	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	UDINT	Local-Server IP-Port (z. B. 200).
flags	ST_TlsListenFlags [▶ 60]	Zusätzliche (optionale) Server-Verbindungseinstellungen.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ein-/Ausgänge

```
VAR_IN_OUT
    hListener : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
hListener	T_HSOCKET	Verbindungshandle [▶ 60] zum neuen Listener-Socket.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.15 FB_TlsSocketCreate

Mit dem Funktionsbaustein FB_TlsSocketCreate kann über den TwinCAT TCP/IP Connection Server ein neuer Socket erzeugt werden, entweder für eine Server- (bListener:=true) oder Client-Applikation (bListener:=false). Über einen Listener-Socket kann der TwinCAT TCP/IP Connection Server nach ankommenden Verbindungsanforderungen von Remote-Clients horchen. Beim Erfolg wird am hListner-Ausgang das dazugehörige Verbindungshandle (hSocket) zurückgeliefert. Dieses Handle wird von dem Funktionsbaustein FB_TlsSocketListen [▶ 38], sowie anschließend FB_SocketAccept [▶ 25] benötigt. Ein nicht mehr benötigter Listener-Socket wird mit dem Funktionsbaustein FB_SocketClose [▶ 22] geschlossen. Nach der Ausführung des Funktionsbausteins FB_TlsSocketCreate können TLS-Parameter zur Absicherung der Kommunikationsverbindung gesetzt werden. Dies geschieht über die Funktionsbausteine FB_TlsSocketAddCa [▶ 41], FB_TlsSocketAddCrl [▶ 42], FB_TlsSocketSetCert [▶ 43] und FB_TlsSocketSetPsk [▶ 44]. Programmierbeispiele hierzu finden Sie in unseren Samples.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
bListener	BOOL	Erzeugt ein neues Socket-Handle.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

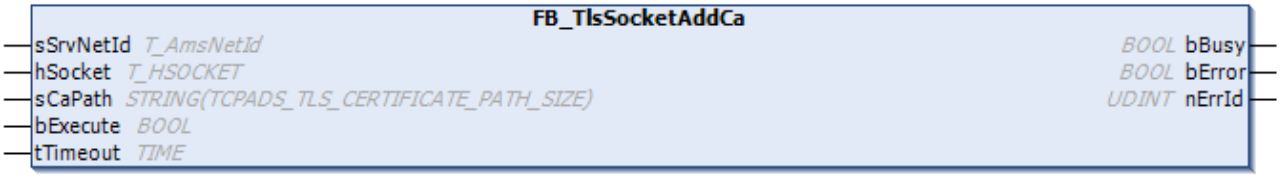
```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    hSocket : T_HSOCKET;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] zum neuen Socket.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.16 FB_TlsSocketAddCa



Mit dem Funktionsbaustein FB_TlsSocketAddCa wird der Pfad zu einem CA-Zertifikat für ein existierendes Socket-Handle konfiguriert. Die Zertifikatsdatei muss hierbei im PEM-Format vorliegen.
Programmierbeispiele zur Verwendung dieses Funktionsbausteins finden Sie in unseren Samples.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Socket-Handle.
sCaPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	Pfad zur Zertifikatsdatei der CA.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

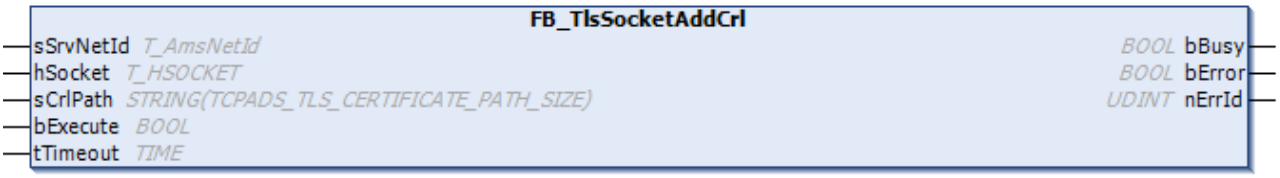
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.17 FB_TlsSocketAddCrl



Mit dem Funktionsbaustein FB_TlsSocketAddCrl wird der Pfad zu einer CRL-Datei für ein existierendes Socket-Handle angegeben. Die CRL muss hierbei im PEM Format vorliegen. Programmierbeispiele zur Verwendung dieses Funktionsbausteins finden Sie in unseren Samples.

Eingänge

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

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Socket-Handle.
sCrlPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	Pfad zur CRL-Datei.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.18 FB_TlsSocketSetCert

Mit dem Funktionsbaustein FB_TlsSocketSetCert kann ein Client/Server-Zertifikat konfiguriert werden, welches für ein bestimmtes Socket-Handle verwendet werden soll. Die Zertifikate müssen hierbei im PEM Format vorliegen. Programmierbeispiele zur Verwendung dieses Funktionsbausteins finden Sie in unseren Samples.

Eingänge

```
VAR_INPUT
    sSrvNetId : T_AmsNetId:='';
    hSocket   : T_HSOCKET;
    sCertPath : STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE):='';
    sKeyPath  : STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE):='';
    sKeyPwd   : STRING(TCPADS_TLS_KEY_PASSWORD_SIZE):='';
    bExecute  : BOOL;
    tTimeout  : TIME:=T#5s;
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Socket-Handle.
sCertPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	Pfad zum Datei mit dem Client/Server Zertifikat.
sKeyPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	Pfad zur Datei mit dem Client/Server Private Key.
sKeyPwd	STRING(TCPADS_TLS_KEY_PASSWORD_SIZE)	Optional, falls der Private Key mit einem Password gesichert ist.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.19 FB_TlsSocketSetPsk

Mit dem Funktionsbaustein FB_TlsSocketSetPsk kann ein Pre-Shared-Secret für ein existierendes Socket-Handle konfiguriert werden. Programmierbeispiele zur Verwendung dieses Funktionsbausteins finden Sie in unseren Samples.

Eingänge

```
VAR_INPUT
    sSrvNetId : T_AmsNetId:='';
    hSocket   : T_HSOCKET;
    sIdentity : STRING(TCPADS_TLS_PSK_IDENTITY_SIZE):='';
    pskKey    : PVOID:=0;
    pskKeyLen : UDINT(0..TCPADS_TLS_MAX_PSK_KEY_SIZE):=0;
    bExecute  : BOOL;
    tTimeout  : TIME:=T#5s;
END_VAR
```

Name	Typ	Beschreibung
sSrvNetId	T_AmsNetId	String mit der Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.
hSocket	T_HSOCKET	Socket-Handle.
sIdentity	STRING(TCPADS_TLS_PSK_IDENTITY_SIZE)	Eine frei wählbare Identity für das PSK.
pskKey	PVOID	Pointer auf ein Byte-Array, welches das PSK enthält.
pskKeyLen	UDINT(0..TCPADS_TLS_MAX_PSK_KEY_SIZE)	Länge von pskKey.
bExecute	BOOL	Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.
tTimeout	TIME	Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

Ausgänge

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

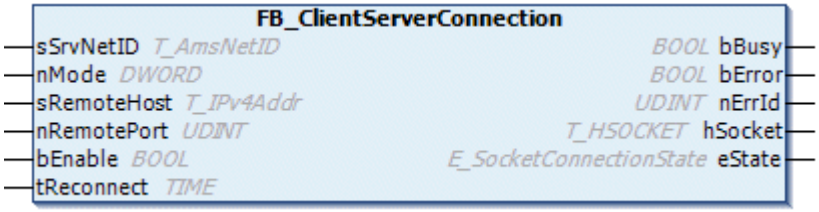
Name	Typ	Beschreibung
bBusy	BOOL	Bei aktiviertem Funktionsbaustein ist diese Ausgabe aktiv. Sie bleibt bis zur Quittierung aktiv.
bError	BOOL	Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.20 Erweitert

5.1.20.1 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 [▶ 21] und FB_SocketClose [▶ 22] 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 [▶ 27] und/oder eine Instanz vom FB_SocketReceive [▶ 28] 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.

Eingänge

```
VAR_INPUT
    sSrvNetID   : T_AmsNetID := '';
    nMode       : DWORD := 0;
    sRemoteHost : T_IPv4Addr := '';
    nRemotePort : UDINT;
    bEnable     : BOOL;
    tReconnect  : TIME := T#45s;(*!!!*)
END_VAR
```

Name	Typ	Beschreibung
sSrvNetID	T_AmsNetID	String mit der AMS-Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den loka-len Rechner (default) kann auch ein Leerstring angegeben werden.
nMode	DWORD	Parameter-Flags (Modi). Die zulässigen Parameter sind hier aufgeführt und können mit einer ODER-Verknüpfung kombiniert werden: CONNECT_MODE_ENABLEDBG: Aktiviert das Loggen von Debug-Meldungen im Application-Log. Um die Debug-Meldungen zu sehen öffnen Sie den Twin-CAT System Manager und aktivieren Sie die Loggeransicht.
sRemoteHost	T_IPv4Addr	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	UDINT	IP-Portnummer des Remote-Servers (z. B. 200).
bEnable	BOOL	Solange dieser Eingang TRUE ist, wird zyklisch versucht, eine neue Verbindung herzustellen, bis eine Verbindung hergestellt wurde. Mit FALSE kann eine hergestellte Verbindung wieder geschlossen werden.
tReconnect	TIME	Zykluszeit, mit der der Funktionsbaustein versucht, die Verbindung aufzubauen.

Zykluszeit für die Verbindung einstellen

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.

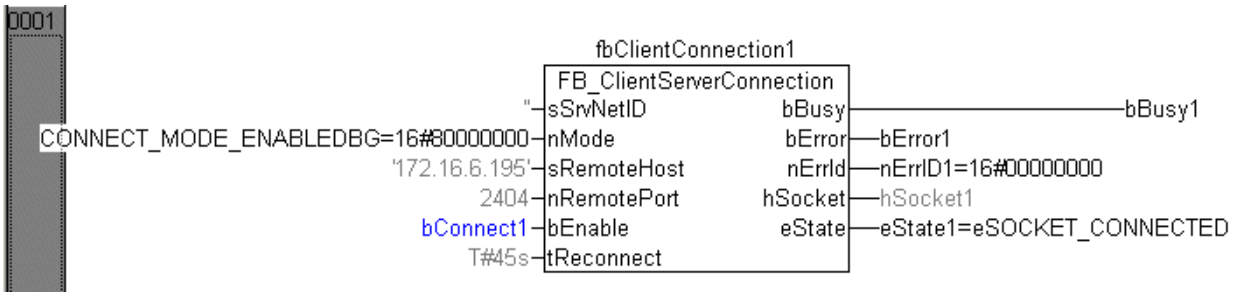
Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, solange der Funktionsbaustein aktiv ist.
bError	BOOL	Wird bei Auftreten eines Fehlercodes TRUE.
nErrID	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] zu dem neu geöffneten Local-Client Socket. Diese Variable wird bei Erfolg an die Instanzen der Funktionsbausteine FB_SocketSend [▶ 27] und/oder FB_SocketReceive [▶ 28] übergeben.
eState	E_SocketConnectionState	Liefert den aktuellen Verbindungsstatus [▶ 56].

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
```

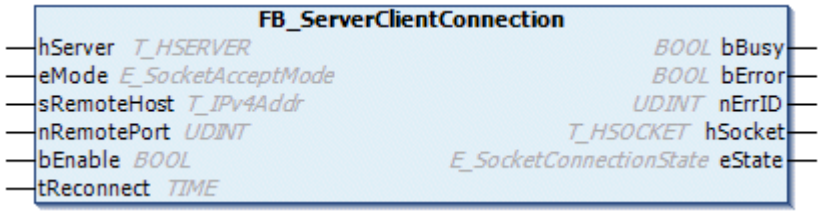


Hier finden Sie weitere Anwendungsbeispiele (und Quellcode): Beispiele [▶ 65]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.20.2 FB_ServerClientConnection



Mit dem Funktionsbaustein FB_ServerClientConnection kann eine Server-Verbindung verwaltet werden (auf- und abgebaut werden). FB_ServerClientConnection vereinfacht die Implementierung einer Server-Applikation in dem er die Funktionalität von den drei Funktionsbausteinen FB_SocketListen [▶ 24], FB_SocketAccept [▶ 25] und FB_SocketClose [▶ 22] 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 [▶ 27] und/oder eine Instanz vom FB_SocketReceive [▶ 28] 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.

Eingänge

```
VAR_INPUT
    eMode       : E_SocketAcceptMode := eACCEPT_ALL;
    sRemoteHost : T_IPv4Addr := '';
    nRemotePort : UDINT := 0;
    bEnable     : BOOL;
    tReconnect  : TIME := T#1s;
END_VAR
```

Name	Typ	Beschreibung
eMode	E_SocketAcceptMode	Legt fest, ob alle oder nur bestimmte Verbindungen akzeptiert [▶ 55] werden sollen.
sRemoteHost	T_IPv4Addr	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	UDINT	IP-Portnummer des Remote-Clients, dessen Verbindung akzeptiert werden soll (z. B. 200).
bEnable	BOOL	Solange dieser Eingang TRUE ist, wird zyklisch versucht, eine neue Verbindung herzustellen, bis eine Verbindung hergestellt wurde. Mit FALSE kann eine hergestellte Verbindung wieder geschlossen werden.
tReconnect	TIME	Zykluszeit, mit der der Funktionsbaustein versucht eine Verbindung aufzubauen.

Ein-/Ausgänge

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

Name	Typ	Beschreibung
hServer	hServer	Server-Handle [▶ 60]. Diese Eingangsvariable muss vorher mit der Funktion F_CreateServerHnd [▶ 52] initialisiert werden.

Ausgänge

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

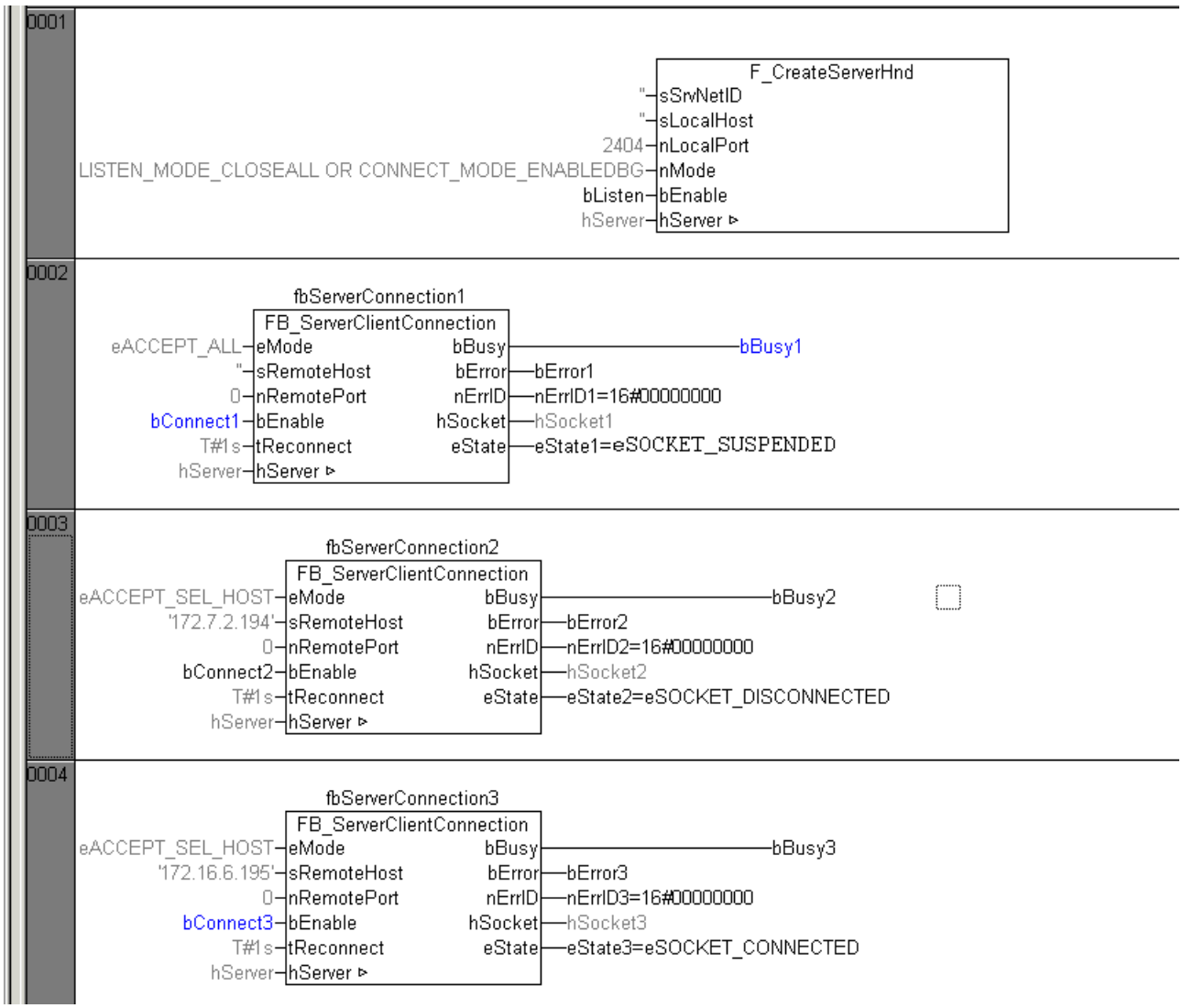
Name	Typ	Beschreibung
bBusy	BOOL	TRUE, solange der Funktionsbaustein aktiv ist.
bError	BOOL	Wird bei Auftreten eines Fehlercodes TRUE.
nErrId	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] zu dem neu geöffneten Remote-Client Socket. Diese Variable wird bei Erfolg an die Instanzen der Funktionsbausteine FB_SocketSend [▶ 27] und/oder FB_SocketReceive [▶ 28] übergeben.
eState	E_SocketConnectionState	Liefert den aktuellen Verbindungsstatus [▶ 56].

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).

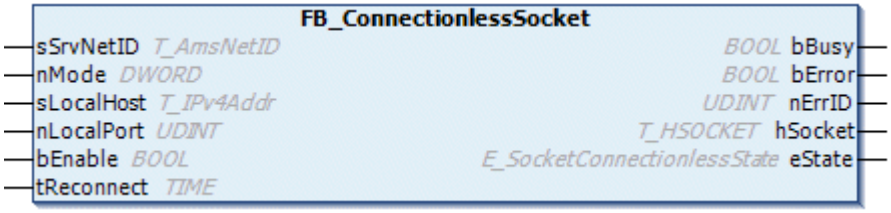
Die dritte Verbindung wurde aktiviert (`bConnect3 = TRUE`) und es wurde eine Verbindung zum Remote-Client hergestellt (Established).

Hier finden Sie weitere Anwendungsbeispiele (und Quellcode): Beispiele [▶ 65]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.1.20.3 FB_ConnectionlessSocket



Mit dem Funktionsbaustein FB_ConnectionlessSocket kann ein UDP-Socket verwaltet werden (geöffnet/ erzeugt und geschlossen werden). FB_ConnectionlessSocket vereinfacht die Implementierung einer UDP-Applikation indem er die Funktionalität von den zwei Funktionsbausteinen FB_SocketUdpCreate [▶ 29] und FB_SocketClose [▶ 22] bereits intern kapselt. Die integrierte Debug-Ausgabe des Socket-Status erleichtert die Fehlersuche bei Konfigurations- oder Kommunikationsfehlern. Eine minimale UDP-Applikation benötigt zusätzlich nur noch jeweils eine Instanz vom FB_SocketUdpSendTo [▶ 31] und/oder eine Instanz vom FB_SocketUdpReceiveFrom [▶ 33] Funktionsbaustein.

Eine typische UDP-Applikation öffnet im ersten Schritt mit dem FB_ConnectionlessSocket-Funktionsbaustein einen verbindungslosen UDP-Socket. Im nächsten Schritt können dann Instanzen von FB_SocketUdpSendTo und/oder FB_SocketUdpReceiveFrom benutzt werden, um Daten mit einem anderen Kommunikationsteilnehmer auszutauschen. Wann ein UDP-Socket geschlossen wird, hängt von den Anforderungen der Applikation ab (z. B. beim Kommunikationsfehler).

Eingänge

```
VAR_INPUT
    sSrvNetID    : T_AmsNetID := '';
    nMode        : DWORD := 0;
    sLocalHost   : T_Ipv4Addr := '';
    nLocalPort   : UDINT;
    bEnable      : BOOL;
    tReconnect   : TIME := T#45s;(*!!!*)
END_VAR
```

Name	Typ	Beschreibung
sSrvNetID	T_AmsNetID	String mit der AMS-Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den loka-len Rechner (default) kann auch ein Leerstring angegeben werden.
nMode	DWORD	Parameter-Flags (Modi). Die zulässigen Parameter sind hier aufgeführt und können mit einer ODER-Verknüpfung kombiniert werden. CONNECT_MODE_ENABLEDBG: Aktiviert das Loggen von Debug-Meldungen im Application-Log. Um die Debug-Meldungen zu sehen, öffnen Sie den Twin-CAT System Manager und aktivieren Sie die Loggeransicht.
sLocalHost	T_Ipv4Addr	IP-Adresse (Ipv4) des lokalen Netzwerkadapters als String (z. B. '172.33.5.1'). Für den Default-Netzwerkadapter kann auch ein Leerstring angegeben werden.
nLocalPort	UDINT	IP-Portnummer auf dem lokalen Rechner (z. B. 200).
bEnable	BOOL	Solange dieser Eingang TRUE ist, wird zyklisch versucht einen UDP-Socket zu öffnen, bis eine Verbindung hergestellt wurde. Mit FALSE kann ein geöffneter UDP-Socket wieder ge-schlossen werden.
tReconnect	TIME	Zykluszeit, mit der der Funktionsbaustein versucht den UDP-Socket zu öffnen.

Zykluszeit für die Verbindung einstellen

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.

Ausgänge

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    hSocket : T_HSOCKET;
    eState  : E_SocketConnectionlessState := eSOCKET_CLOSED;
END_VAR
```

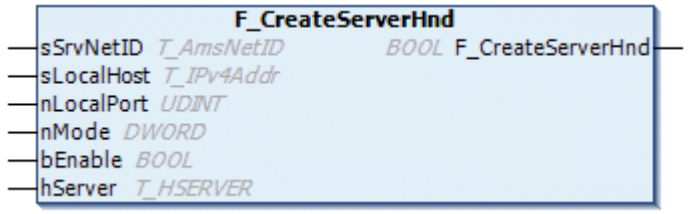
Name	Typ	Beschreibung
bBusy	BOOL	TRUE, solange der Funktionsbaustein aktiv ist.
bError	BOOL	Wird bei Auftreten eines Fehlercodes TRUE.
nErrID	UDINT	Dieser Parameter liefert bei einem gesetzten bError-Ausgang die TwinCAT TCP/IP Connection Server Fehlernummer [▶ 102].
hSocket	T_HSOCKET	Verbindungshandle [▶ 60] zu dem neu geöffneten UDP-Socket. Diese Variable wird bei Erfolg an die Instanzen der Funktionsbausteine FB_SocketUdpSendTo [▶ 31] und/oder FB_SocketUdpReceiveFrom [▶ 33] übergeben.
eState	E_SocketConnectionlessState	Liefert den aktuellen Verbindungsstatus [▶ 56].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.2 Funktionen

5.2.1 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 [▶ 47]-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.

Syntax

```
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
```

Rückgabewert

Name	Typ	Beschreibung
F_CreateServerHnd	BOOL	Liefert TRUE, wenn alles okay ist, FALSE, wenn ein falscher Parameterwert vorliegt.

Eingänge

Name	Typ	Beschreibung
sSrvNetID	T_AmsNetID	String mit der AMS-Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Für den loka-len Rechner (default) kann auch ein Leerstring angegeben werden.
sLocalHost	T_IPv4Addr	Local-Server IP-Adresse (Ipv4) als String (z.B. '172.13.15.2'). Für einen Server auf dem loka-len Rechner (default) kann auch ein Leerstring angegeben werden.
nLocalPort	UDINT	Local-Server IP-Port (z.B. 200).
nMode	DWORD	Parameter-Flags (Modi). Die zulässigen Parameter sind hier aufgeführt und können mit einer ODER-Verknüpfung kombiniert werden. 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 Twin-CAT System Manager und aktivieren Sie die Loggeransicht.
bEnable	BOOL	Dieser Eingang legt das Verhalten des Listener-Sockets fest. Ein vorher geöffneter Liste-ner 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.

/ Ein-/Ausgänge

Name	Typ	Beschreibung
hServer	T_HSERVER	Server-Handle-Variable, deren interne Parameter initialisiert werden sollen.

Beispiel:

Siehe FB_ServerClientConnection [▶ 47].

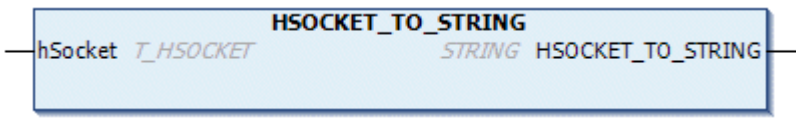
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

Sehen Sie dazu auch

T_HSERVER [▶ 60]

5.2.2 HSOCKET_TO_STRING



Die Funktion konvertiert das Verbindungshandle vom Typ T_HSOCKET in einen String (z. B. für Debug-Ausgaben).

Der zurückgelieferte String hat folgendes Format: "Handle:0xA[BCD] Local:a[aa].b[bb].c[cc].d[dd]:port Remote:a[aa].b[bb].c[cc].d[dd]:port".

Beispiel: "Handle:0x4001 Local:172.16.6.195:28459 Remote:172.16.6.180:2404"

Syntax

```
FUNCTION HSOCKET_TO_STRING : STRING
VAR_INPUT
    hSocket : T_HSOCKET;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
HSOCKET_TO_STRING	STRING	Enthält die STRING-Repräsentation des Verbindungshandles.

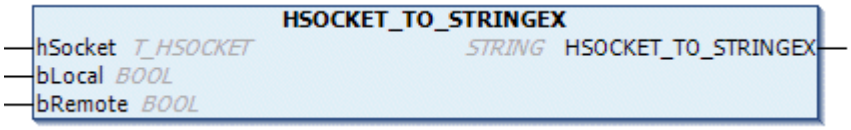
Eingänge

Name	Typ	Beschreibung
hSocket	T_HSOCKET	Das zu konvertierende Verbindungshandle [▶ 60].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.2.3 HSOCKET_TO_STRINGEX



Die Funktion konvertiert das Verbindungs-Handle vom Type T_HSOCKET in einen String (z. B. für Debug-Ausgaben).

Der zurückgelieferte String hat folgendes Format: "Handle:0xA[BCD] Local:a[aa].b[bb].c[cc].d[dd]:port Remote:a[aa].b[bb].c[cc].d[dd]:port".

Beispiel: "Handle:0x4001 Local:172.16.6.195:28459 Remote:172.16.6.180:2404"

Die Parameter bLocal und bRemote bestimmen, ob die lokale und/oder remote Adressinformation in dem zurückgelieferten String enthalten sein soll.

Syntax

```
FUNCTION HSOCKET_TO_STRINGEX : STRING
VAR_INPUT
    hSocket : T_HSOCKET;
    bLocal  : BOOL;
    bRemote : BOOL;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
HSOCKET_TO_STRINGEX	STRING	Enthält die Hex-basierte STRING-Repräsentation des Verbindungshandles.

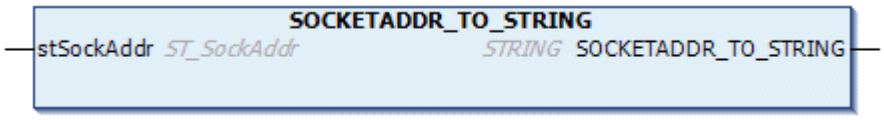
Eingänge

Name	Typ	Beschreibung
hSocket	T_HSOCKET	Das zu konvertierende Verbindungshandle [▶ 60].
bLocal	BOOL	TRUE: Inkludiere die lokale Adresse, FALSE: Exkludiere die lokale Adresse.
bRemote	BOOL	TRUE: Inkludiere die remote Adresse, FALSE: Exkludiere die remote Adresse.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.2.4 SOCKETADDR_TO_STRING



Die Funktion konvertiert eine Variable vom Typ ST_SockAddr in einen String (z. B. für Debug-Ausgaben).

Der zurückgelieferte String hat folgendes Format: "a[aa].b[bb].c[cc].d[dd]:port"

Beispiel: "172.16.6.195:80"

```
FUNCTION SOCKETADDR_TO_STRING : STRING
VAR_INPUT
    stSockAddr : ST_SockAddr;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
SOCKETADDR_TO_STRING	STRING	Enthält die STRING-Repräsentation der Socket Adresse.

Eingänge

Name	Typ	Beschreibung
stSockeAddr	ST_SockAddr	Die zu konvertierende Variable.

Siehe ST_SockAddr [▶ 59]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3 Datentypen

5.3.1 E_SocketAcceptMode

E_SocketAcceptMode legt fest, welche Verbindungen vom Server akzeptiert werden.

Syntax

```
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 *)
    eACCEPT_SEL_HOST_PORT (* Accept connection to selected host and port address *)
);
END_TYPE
```

Werte

Name	Beschreibung
eACCEPT_ALL	Verbindung zu allen remote-Clients akzeptieren.
eACCEPT_SEL_HOST	Verbindung zu ausgewählter Hostadresse akzeptieren.
eACCEPT_SEL_PORT	Verbindung zu ausgewählter Portadresse akzeptieren.
eACCEPT_SEL_HOST_PORT	Verbindung zu ausgewähltem Host und Portadresse akzeptieren.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.2 E_SocketConnectionState

TCP/IP Socket Connection Status (eSOCKET_SUSPENDED == der Status ändert sich z. B. von eSOCKET_CONNECTED => eSOCKET_DISCONNECTED).

Syntax

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

Werte

Name	Beschreibung
eSOCKET_DISCONNECTED	Die Verbindung ist unterbrochen.
eSOCKET_CONNECTED	Die Verbindung steht.
eSOCKET_SUSPENDED	Der Status der Verbindung wechselt von unterbrochen zu verbunden oder von verbunden zu unterbrochen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.3 E_SocketConnectionlessState

Statusinformation eines verbindungslosen UDP-Sockets (eSOCKET_TRANSIENT == der Status ändert sich z .B. von eSOCKET_CREATED => eSOCKET_CLOSED).

Syntax

```
TYPE E_SocketConnectionlessState:
(
    eSOCKET_CLOSED,
    eSOCKET_CREATED,
    eSOCKET_TRANSIENT
);
END_TYPE
```

Werte

Name	Beschreibung
eSOCKET_CLOSED	Das UDP-Socket ist geschlossen.
eSOCKET_CREATED	Das UDP-Socket ist erstellt.
eSOCKET_TRANSIENT	Das UDP-Socket wechselt von geschlossen zu offen oder von offen zu geschlossen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.4 E_WinsockError

Syntax

```
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 net
work 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_FOUND := 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 database, but it doe
s not have the correct associated data being resolved for. *)
 );
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.5 ST_SockAddr

Die Struktur enthält Adressinformationen eines geöffneten Sockets.

Syntax

```
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
```

Werte

Name	Typ	Beschreibung
nPort	UDINT	Internetprotokoll (IP) port
sAddr	STRING(15)	Durch Punkte getrennte Internetprotokolladresse (Ipv4) als String z. B.:"172.34.12.3"

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.6 ST_TlsConnectFlags

Zusätzliche (optionale) Client-Verbindungsparameter.

Syntax

```
TYPE ST_TlsConnectFlags :
STRUCT
    bNoServerCertCheck: BOOL;
    bIgnoreCnMismatch : BOOL;
END_STRUCT
END_TYPE
```

Werte

Name	Typ	Beschreibung
bNoServerCertCheck	BOOL	Deaktiviert die Validierung des Serverzertifikats.
bIgnoreCnMismatch	BOOL	Ignoriert, wenn der CommonName im Serverzertifikat nicht mit dem Hostname übereinstimmt, der als sRemoteHost angegeben wurde.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.7 ST_TlsListenFlags

Zusätzliche (optionale) Server-Verbindungsparameter.

Syntax

```
TYPE ST_TlsListenFlags :
STRUCT
    bNoClientCert : BOOL;
END_STRUCT
END_TYPE
```

Werte

Name	Typ	Beschreibung
bNoClientCert	BOOL	Client-Zertifikat wird nicht benötigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.8 T_HSERVER

Eine Variable von diesem Typ repräsentiert ein TCP/IP Server Handle. Das Handle muss vor der Nutzung mit F_CreateServerHnd [▶ 52] initialisiert werden. Damit werden die internen Parameter der Variablen T_HSERVER festgelegt.

Strukturelemente erhalten

Strukturelemente dürfen nicht überschrieben oder verändert werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.3.9 T_HSOCKET

Variablen von diesem Typ repräsentieren ein Verbindungshandle oder 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.

Syntax

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

Werte

Name	Typ	Beschreibung
handle	UDINT	Internes Socket-Handle des TwinCAT TCP/IP Connection Servers.
localAddr	ST_SockAddr	Lokale Socketadresse [▶ 59].
remoteAddr	ST_SockAddr	Remote Socketadresse [▶ 59].

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 Adressinformationen eingetragen.

Das Verbindungshandle auf der Serverseite

- Der Funktionsbaustein FB_SocketListen [▶ 24] öffnet einen Listener-Socket und liefert das Verbindungshandle des Listener-Sockets zurück.
- Das Verbindungshandle des Listener-Sockets wird an den Funktionsbaustein FB_SocketAccept [▶ 25] 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 [▶ 27] und/oder FB_SocketReceive [▶ 28] 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 [▶ 22] 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 [▶ 21] liefert das Verbindungshandle eines Local-Client-Sockets zurück.
- Dieses Verbindungshandle wird dann an die Funktionsbausteine FB_SocketSend [▶ 27] und FB_SocketReceive [▶ 28] übergeben, um Daten mit einem Remote-Server austauschen zu können.
- Das gleiche Verbindungshandle wird dann an den Funktionsbaustein FB_SocketClose [▶ 22] übergeben, um eine nicht mehr benötigte Verbindung zu schließen.

Mit dem Funktionsbaustein FB_SocketCloseAll [▶ 23] werden alle Sockets geschlossen, die von einem SPS-Runtime-System geöffnet wurden. D. h. wenn Sie FB_SocketCloseAll in einer der Tasks des ersten Runtime-Systems (Port 801) aufrufen, werden alle Sockets geschlossen die in dem ersten Runtime-System geöffnet wurden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.4 Globale Konstanten

5.4.1 Globale Variablen

Syntax

```
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;
    TCPADS_IGR_SENDTOBYHDL        : UDINT:=16#80000007;
    TCPADS_IGR_MULTICAST_ADDBYHDL : UDINT:=16#80000008;
    TCPADS_IGR_MULTICAST_DROPBYHDL: UDINT:=16#80000009;

    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;

    TLS_CONNECT_FLAG_INSECURE     : DWORD:=16#00000001;
    TLS_CONNECT_FLAG_IGNORE_CN    : DWORD:=16#00000002;
    TLS_LISTEN_FLAG_REQUIRES_CERT : DWORD:=16#00000001;

    TCPADS_NULL_HSOCKET           : T_HSOCKET:=(handle:=0, remoteAddr:=(nPort:=0, sAddr:=''), localA
ddr:=(nPort:=0, sAddr:=''));

    LISTEN_MODE_CLOSEALL          : DWORD:=16#00000001;
    LISTEN_MODE_USEOPENED         : DWORD:=16#00000002;
    CONNECT_MODE_ENABLEDBG        : DWORD:=16#80000000;
    DEFAULT_TLSLISTENFLAGS        : ST_TlsListenFlags:=(bNoClientCert:=FALSE);
    DEFAULT_TLSCONNECTFLAGS       : ST_TlsConnectFlags:=(bNoServerCertCheck:=FALSE, bIgnoreCnMismatc
h:=FALSE);
END_VAR
```

Parameter

Name	Typ	Beschreibung
AMSPORT_TCPIPSRV	UINT	
TCPADS_IGR_CONLIST	UDINT	
TCPADS_IGR_CLOSEBYHDL	UDINT	
TCPADS_IGR_SENDBYHDL	UDINT	
TCPADS_IGR_PEERBYHDL	UDINT	
TCPADS_IGR_RECVBYHDL	UDINT	
TCPADS_IGR_RECVFROMBYHDL	UDINT	
TCPADS_IGR_SENDTOBYHDL	UDINT	
TCPADS_IGR_MULTICAST_ADDBYHDL	UDINT	
TCPADS_IGR_MULTICAST_DROPBYHDL	UDINT	
TCPADSCONLST_IOF_CONNECT	UDINT	
TCPADSCONLST_IOF_LISTEN	UDINT	
TCPADSCONLST_IOF_CLOSEALL	UDINT	
TCPADSCONLST_IOF_ACCEPT	UDINT	
TCPADSCONLST_IOF_UDPBIND	UDINT	
TLS_CONNECT_FLAG_INSECURE	DWORD	Zertifikat des Servers wird nicht überprüft.
TLS_CONNECT_FLAG_IGNORE_CN	DWORD	Ungleichheit im “common name” des Servers wird ignoriert.
TLS_LISTEN_FLAG_REQUIRES_CERT	DWORD	Konfiguration des Client-Zertifikats wird benötigt und vorausgesetzt.
TCPADS_NULL_HSOCKET	T_HSOCKET	Leeres (nicht initialisiertes) Socket.
LISTEN_MODE_CLOSEALL	DWORD	FORCED close aller zuvor geöffneten Sockets.
LISTEN_MODE_USEOPENED	DWORD	Versuch, einen bereits geöffneten Listener-Socket zu verwenden.
CONNECT_MODE_ENABLEDBG	DWORD	Aktiviert/deaktiviert Debugging-Meldungen.
DEFAULT_TLSLISTENFLAGS	ST_TlsListenFlags [▶ 60]	Default (optional) TLS Server-Verbindungseinstellungen.
DEFAULT_TLSCONNECTFLAGS	ST_TlsConnectFlags [▶ 59]	Default (optional) TLS Client-Verbindungseinstellungen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.4.2 Bibliotheksversion

Alle Bibliotheken haben eine bestimmte Version. Diese Version wird im Repository der SPS-Bibliothek angezeigt.
Die Versionsnummer der Bibliothek ist in einer globalen Konstante gespeichert (Typ: ST_LibVersion).

Global_Version

```
VAR_GLOBAL CONSTANT
    stLibVersion_Tc2_TcpIp : ST_LibVersion;
END_VAR
```

Zum Vergleich zwischen vorhandener und erforderlicher Version dient die Funktion F_CmpLibVersion (in der Tc2_System Bibliothek).

Kompatibilität zu TwinCAT 2

Abfragemöglichkeiten von TwinCAT2 Bibliotheken sind nicht mehr verfügbar!

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

5.4.3 Parameterliste

Param

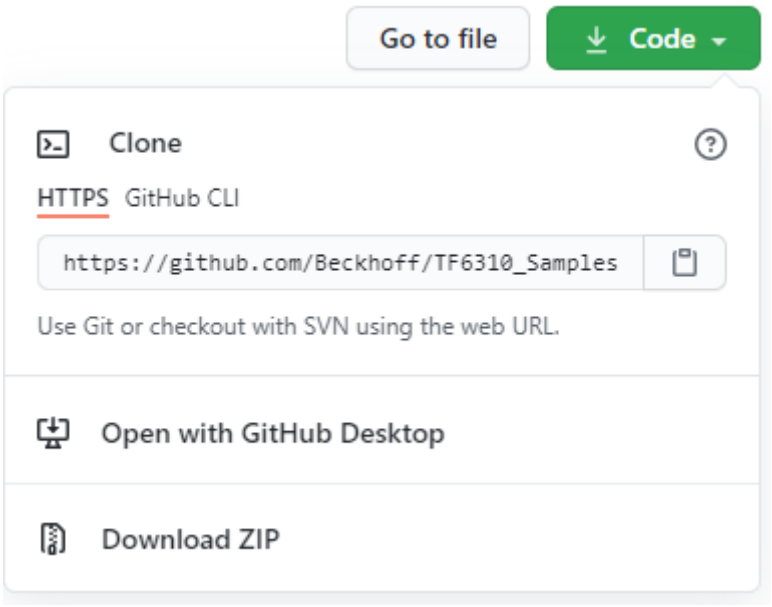
Name	Typ	Wert	Beschreibung
TCPADS_MAXUDP_BUFFSIZE	UDINT	16#2000	Max. Bytelänge des internen UDP send/receive Puffers (8192 bytes).
TCPADS_TLS_HOSTNAME_SIZE	UDINT	255	Max. Länge des Hostname-Strings.
TCPADS_TLS_CERTIFICATE_PATH_SIZE	UDINT	255	Max. Länge des Zertifikatpfad-Strings.
TCPADS_TLS_KEY_PASSWORD_SIZE	UDINT	255	Max. Länge des Zertifikatpasswordpfad-Strings.
TCPADS_TLS_PSK_IDENTITY_SIZE	UDINT	255	Max. Länge des PSK Identity-Strings.
TCPADS_TLS_MAX_PSK_KEY_SIZE	UDINT	128	Max. Bytelänge des PSK-Schlüssels.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken (Kategoriegruppe)
TF6310 v3.3.15.0 oder neuer TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_TcpIp (Communication)

6 Beispiele

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://github.com/Beckhoff/TF6310_Samples . Sie haben dort die Möglichkeit das Repository zu clonen oder ein ZIP File mit dem Sample herunterzuladen.



6.1 TCP

6.1.1 Beispiel01: "Echo" Client/Server (Basisbausteine)

6.1.1.1 Übersicht

Das folgende Beispiel zeigt eine beispielhafte Implementierung eines „Echo“-Client/Server-Systems. Der Client sendet in bestimmten Abständen (z. B. jede Sekunde) einen Test-String zum Server. Der Remote-Server sendet diesen String wieder zurück an den Client.

In diesem Beispiel ist der Client sowohl in der SPS implementiert als auch in einer .NET-Anwendung, die in C# geschrieben wurde. Der SPS-Client kann verschiedene Kommunikationsinstanzen erzeugen, die mehrere TCP-Verbindungen gleichzeitig simulieren. Der unter .NET entwickelte Beispiel-Client stellt nur eine Verbindung her. Der Server in der SPS kann mit mehreren Clients kommunizieren.

Vom Server können zusätzlich mehrere Instanzen angelegt werden. Jede Server-Instanz wird dann über eine eigene Portnummer angesprochen, die der Client zum Aufbau einer Verbindung zu einer spezifischen Instanz nutzen kann. Die Server-Implementierung ist schwieriger, wenn der Server mit mehr als nur einem Client kommunizieren soll.

Sie können das Beispiel beliebig nutzen und für Ihre Zwecke anpassen.

Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP
- Wird das Beispiel auf zwei Computern ausgeführt (ein Client und ein Server), muss die Function TF6310 auf beiden installiert sein.
- Wird das Beispiel auf einem Computer ausgeführt (z. B. Client und Server laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen.

- Zum Ausführen des .NET-Beispielclient ist nur .NET Framework 4.0 nötig.

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample01

https://github.com/Beckhoff/TF6310_Samples/tree/master/C%23/SampleClient

Projektbeschreibung

Für jede der erwähnten Komponenten ist eine Dokumentation verfügbar, welche im Folgenden verlinkt wurde. Ein separater Artikel erklärt Schritt für Schritt, wie die SPS- Beispiele eingerichtet und gestartet werden.

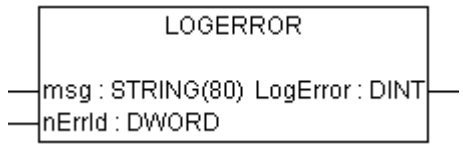
- Integration in TwinCAT und Test [▶ 67] (Start der SPS Beispiele)
- SPS-Client [▶ 70] (SPS-Client-Dokumentation: FB_LocalClient function block [▶ 70])
- SPS-Server [▶ 74] (SPS-Server-Dokumentation: FB_LocalServer function block [▶ 74])
- .NET-Client [▶ 80] (.NET-Client-Dokumentation: .NET sample client [▶ 80])

Zusätzliche Funktionen der SPS-Beispielprojekte

In den Beispielprojekten werden einige Funktionen, Konstanten und Funktionsbausteine benutzt, die im Folgenden kurz beschrieben werden sollen:

LogError-Funktion

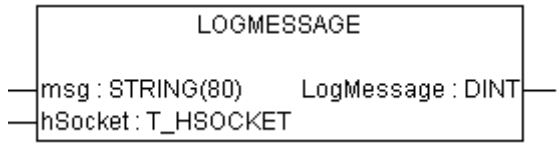
```
FUNCTION LogError : DINT
```



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

LogMessage-Funktion

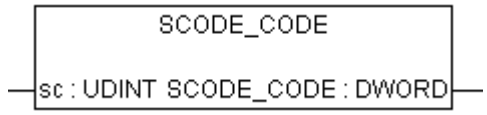
```
FUNCTION LogMessage : DINT
```



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 zuerst auf TRUE gesetzt werden.

SCODE_CODE-Funktion

```
FUNCTION SCODE_CODE : DWORD
```



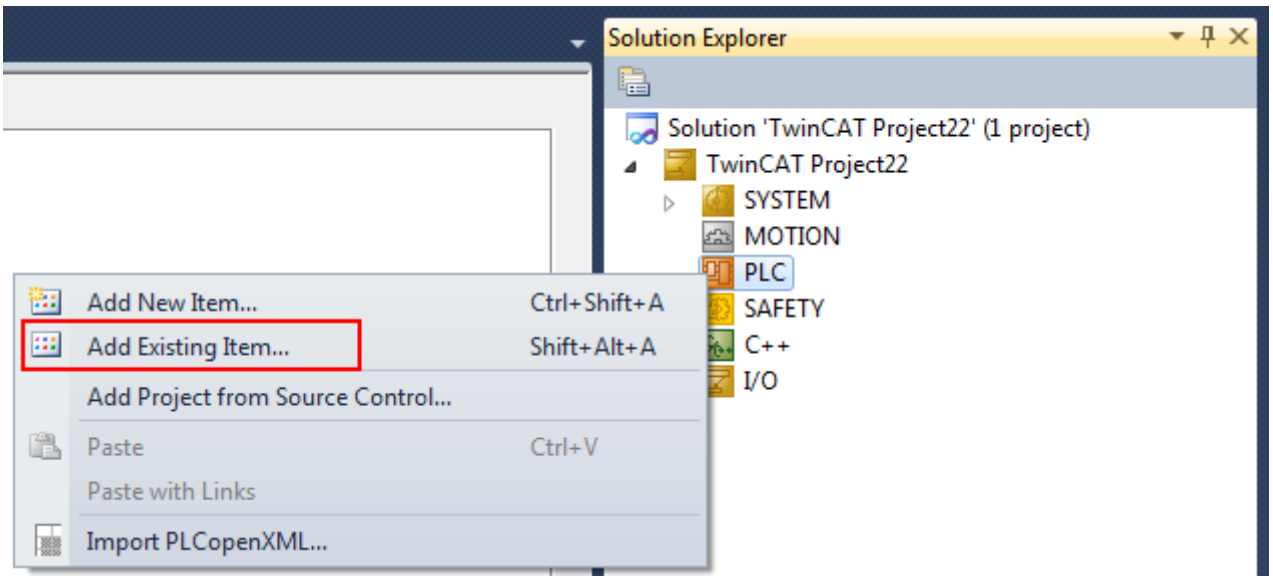
Die Funktion maskiert die niederwertigsten 16 Bits eines Win32-Fehlercodes aus und liefert diese zurück.

Globale Variablen

Name	Default-Wert	Beschreibung
bLogDebugMessages	TRUE	Aktiviert/deaktiviert das Schreiben von Nachrichten ins Logbuch des Betriebssystems
MAX_CLIENT_CONNECTIONS	5	Max. Anzahl der Remote-Clients, die eine Verbindung zum Server gleichzeitig aufbauen können;
MAX_PLCPRJ_RXBUFFER_SIZE	1000	Max. Länge des internen Empfangspuffers
PLCPRJ_RECONNECT_TIME	T#3s	Nach Ablauf dieser Zeit versucht der Local-Server den Listener-Socket neu zu öffnen
PLCPRJ_RECEIVE_POLLING_TIME	T#1s	In diesem Zyklus liest (pollt) der Server Daten
PLCPRJ_RECEIVE_TIMEOUT	T#50s	Nach Ablauf dieser Zeit bricht der Server den Empfang ab, wenn keine Datenbytes empfangen werden konnten
PLCPRJ_ACCEPT_POLLING_TIME	T#1s	In diesen Zeitabständen versucht der Local-Server die Verbindungsanforderungen des Remote-Clients anzunehmen (akzeptieren);
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	Fehlercode Beispielprojekt: Zu viele Zeichen ohne Nullterminierung empfangen
PLCPRJ_ERROR_RECEIVE_TIMEOUT	16#8102	Fehlercode Beispielprojekt: Bis zum Timeout konnten keine neuen Daten empfangen werden (PLCPRJ_RECEIVE_TIMEOUT)

6.1.1.2 Integration in TwinCAT und Test

Nachfolgend wird beschrieben, wie SPS-Server und -Client vorbereitet und gestartet werden. Die SPS-Beispiele werden als TwinCAT-3-SPS-Projektdateien zur Verfügung gestellt. Um ein SPS-Projekt in das TwinCAT XAE zu importieren, erstellen Sie zunächst eine neue TwinCAT 3 Solution. Wählen Sie anschließend im Kontextmenü des PLC-Knoten den Befehl **Add Existing Item** und in dem sich öffnenden Dialog die heruntergeladenen Beispieldatei (*Plc 3.x Project archive (*.tpzip)* als Dateityp auswählen). Nach Bestätigung des Dialogs wird das SPS-Projekt der Solution hinzugefügt.



Beispiel SPS-Server

Erstellen Sie im TwinCAT XAE eine neue TwinCAT 3 Solution und importieren Sie das TCP/IP-Server-Projekt. Wählen Sie ein Zielsystem. Auf dem Zielsystem muss die Function ebenfalls installiert sein und es müssen Lizenzen für TF6310 generiert worden sein. Lassen Sie die TwinCAT 3 Solution geöffnet.

```
PROGRAM MAIN
VAR
    fbServer         : FB_LocalServer := ( sLocalHost := '127.0.0.1' (*own IP address!
*), nLocalPort := 200 );
    bEnableServer    : BOOL := TRUE;
    fbSocketCloseAll : FB_SocketCloseAll := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    bCloseAll        : BOOL := TRUE;
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
    fbServer( bEnable := bEnableServer );
END_IF
```

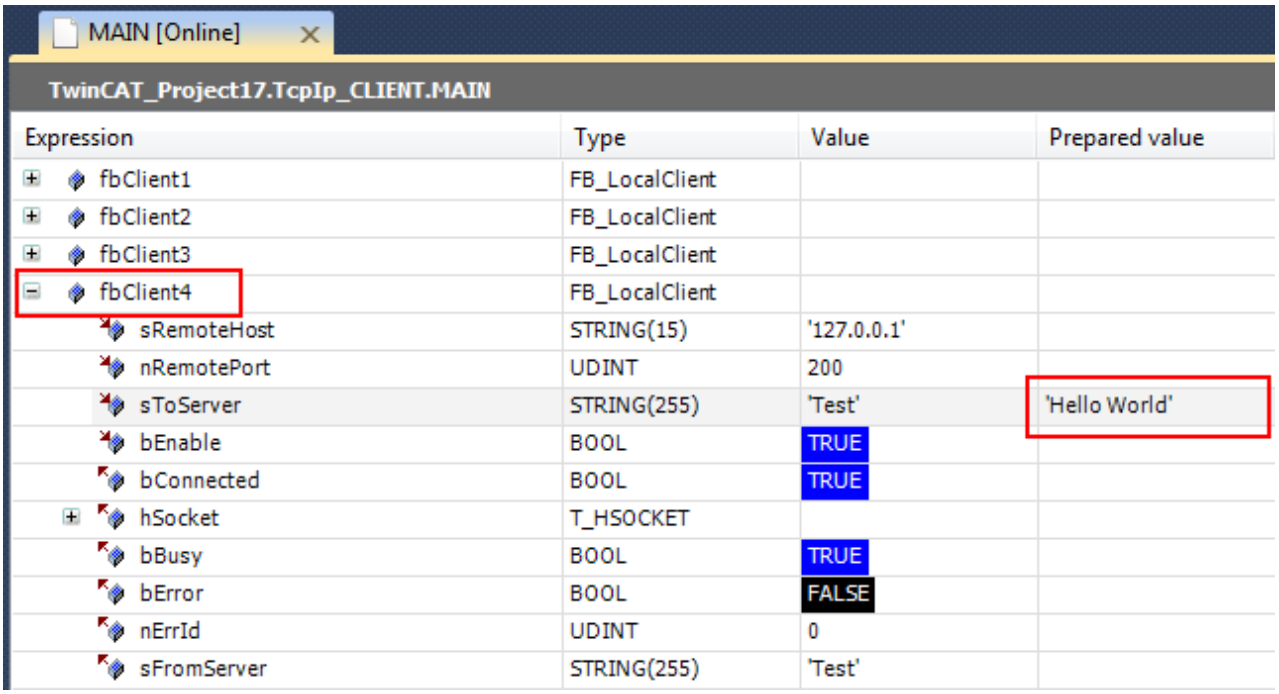
Beispiel SPS-Client

Importieren Sie das TCP/IP-Client-Projekt als zweites SPS-Projekt in die TwinCAT 3 Solution. Verlinken Sie dieses SPS-Projekt mit einer anderen Task als das Server-Beispiel. Die IP-Adresse des Servers muss an Ihr System angepasst werden (Initialisierungswerte der sRemoteHost-Variable). In diesem Fall ist der Server auf demselben PC, tragen Sie also 127.0.0.1. ein. Aktivieren Sie die Konfiguration, loggen sich ein und starten Sie das Server- und anschließend das Client-SPS-Projekt.

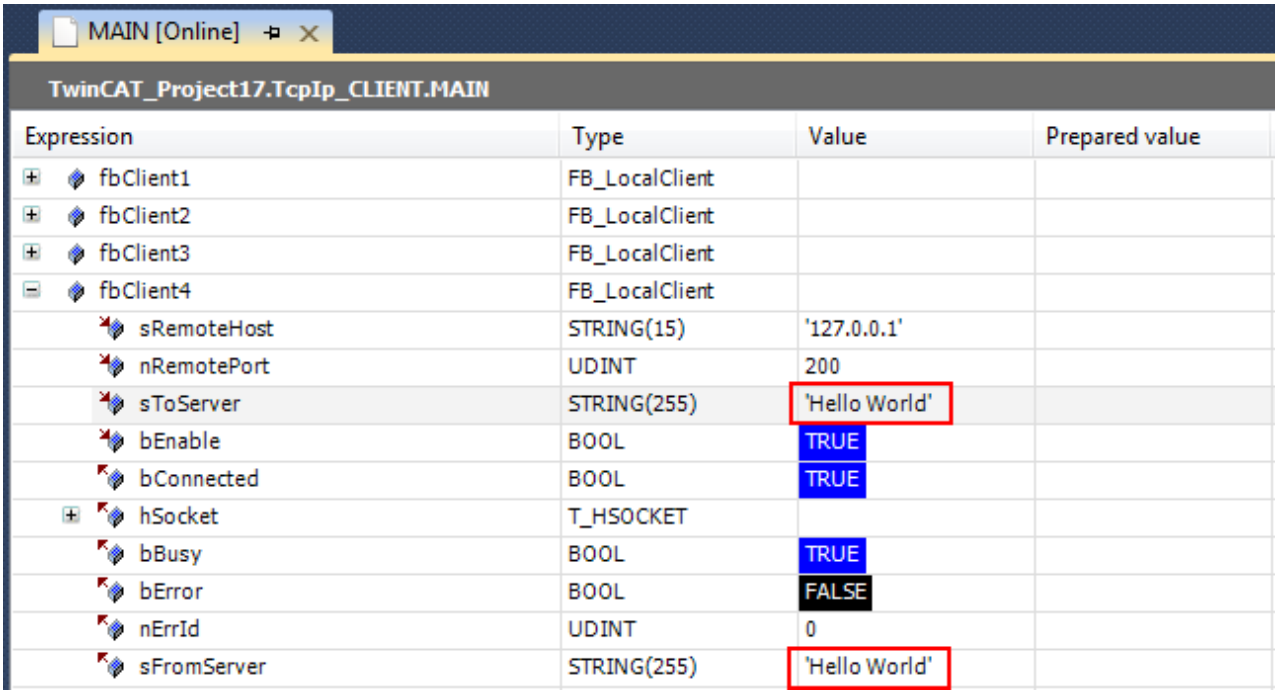
```
PROGRAM MAIN
VAR
    fbClient1     : FB_LocalClient := ( sRemoteHost:= '127.0.0.1' (* IP address of remote server! *)
, nRemotePort:= 200 );
    fbClient2     : FB_LocalClient := ( sRemoteHost:= '127.0.0.1', nRemotePort:= 200 );
    fbClient3     : FB_LocalClient := ( sRemoteHost:= '127.0.0.1', nRemotePort:= 200 );
    fbClient4     : FB_LocalClient := ( sRemoteHost:= '127.0.0.1', nRemotePort:= 200 );
    fbClient5     : FB_LocalClient := ( sRemoteHost:= '127.0.0.1', 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
```

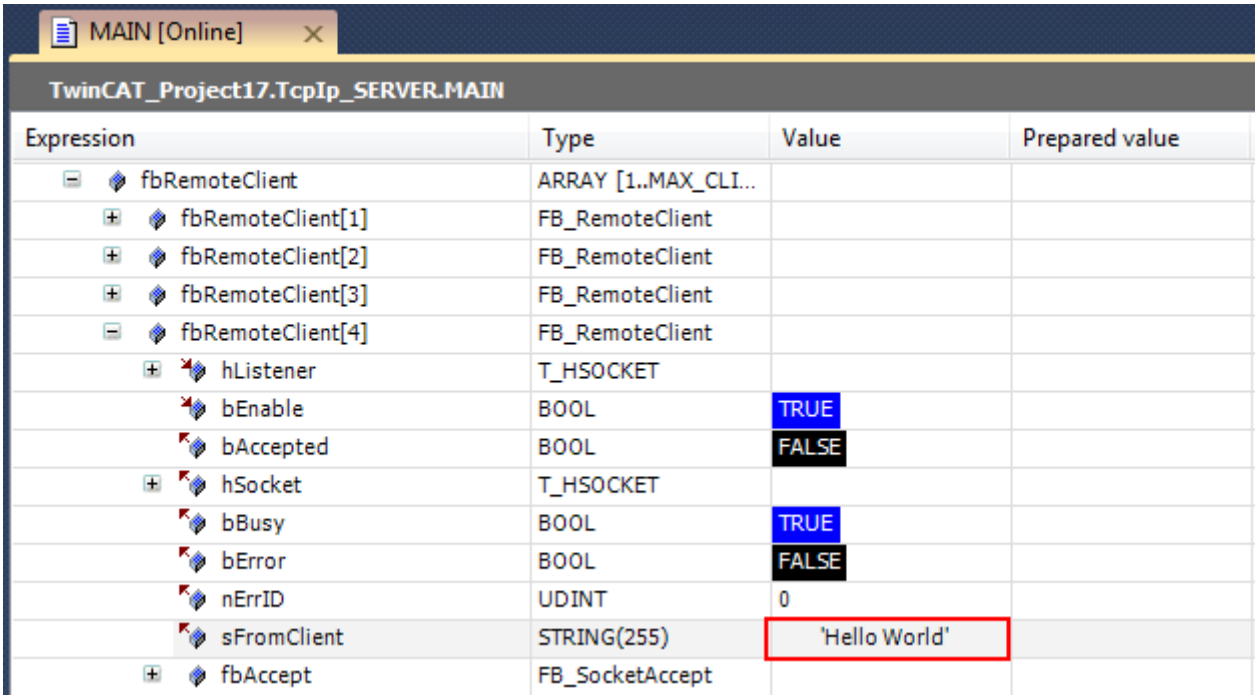
Beim Setzen einer der bEnableCientX-Variablen können bis zu fünf Client-Instanzen aktiviert werden. Jeder Client sendet pro Sekunde einen String zum Server (default: 'TEST'). Der gleiche String wird vom Server zum Client zurückgesendet (Echo-Server). Für den Test wird bei den ersten drei Instanzen ein String mit einem Zählerwert automatisch generiert. Der erste Client wird beim Programmstart automatisch aktiviert. Setzen Sie die bEnableCilent4-Variable im Client-Projekt auf TRUE. Der neue 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 und ändern Sie den Wert der Stringvariablen z. B. auf 'Hallo'.



Schließen Sie den Dialog mit **OK**. Forcen Sie den neuen Wert in die SPS. Kurz danach ist der vom Server zurückgegebene Wert online zu sehen.

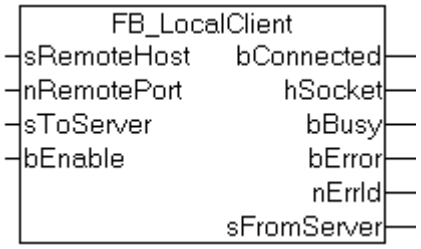


In dem Server-Projekt öffnen Sie jetzt die fbServer-Instanz des FB_LocalServer-Funktionsbausteins. Unser String: 'Hallo' kann in den Online-Daten des Servers gesehen werden.



6.1.1.3 SPS-Client

6.1.1.3.1 FB_LocalClient



Bei gesetztem bEnable-Eingang wird immer wieder versucht, nach Ablauf der CLIENT_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 [▶ 72] gekapselt. Der Datenaustausch erfolgt zyklisch, immer nach Ablauf von PLCPRJ_SEND_CYCLE_TIME. Dabei wird die sToServer-Stringvariable zum Server gesendet, der vom Server zurückgesandte String steht am Ausgang sFormServer zur Verfügung. Eine andere Implementierung, in der der Remote-Server bei Bedarf angesprochen wird, ist aber ebenfalls möglich. Bei einem Fehler wird die vorhandene Verbindung geschlossen und eine neue aufgebaut.

Schnittstelle

```
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;
```

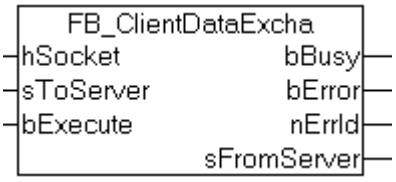
```
    sFromServer     : T_MaxString;
END_VAR
VAR
    fbConnect       : FB_SocketConnect := ( sSrvNetId := '' );
    fbClose         : FB_SocketClose := ( sSrvNetId := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbClientDataExcha : FB_ClientDataExcha;

    fbConnectTON     : TON := ( PT := PLCPRJ_RECONNECT_TIME );
    fbDataExchaTON   : TON := ( PT := PLCPRJ_SEND_CYCLE_TIME );
    eStep            : E_ClientSteps;
END_VAR
```

Realisierung

```
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
```

6.1.1.3.2 FB_ClientDataExcha



Bei einer steigenden Flanke am bExecute-Eingang wird ein Nullterminierter-String zum Remote-Server gesendet und ein vom Remote-Server zurückgelieferter String zurückgelesen. Der Funktionsbaustein versucht die Daten so lange zu lesen, bis eine Nullterminierung in dem empfangenen String erkannt wurde. Wenn die Timeout-Zeit PLCPRJ_RECEIVE_TIMEOUT überschritten wurde oder wenn ein Fehler auftritt, wird der Empfang abgebrochen. Der nächste Lesevorgang wird nach einer Verzögerungszeit ausgeführt, wenn beim letzten Lesevorgang keine neuen Daten gelesen werden konnten. Die Systemauslastung verringert sich dadurch.

Schnittstelle

```
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 := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbSocketReceive : FB_SocketReceive := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
```

```
    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
```

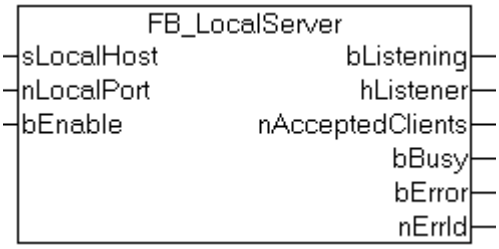
Realisierung

```
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_OVE
RFLOW );
                            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
```

6.1.1.4 SPS-Server

6.1.1.4.1 FB_LocalServer



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 SERVER_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 [▶ 76] 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-Sockets geschlossen. Der nAcceptedClients-Ausgang gibt Auskunft über die aktuelle Anzahl der verbundenen Clients.

Schnittstelle

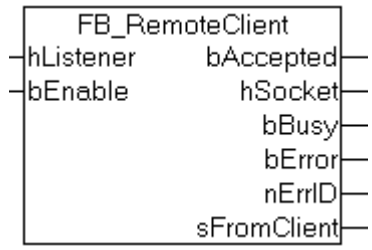
```
FUNCTION_BLOCK FB_LocalServer
VAR_INPUT
    sLocalHost       : STRING(15) := '127.0.0.1';(* own IP address! *)
    nLocalPort       : UDINT := 0;
    bEnable          : BOOL;
END_VAR
VAR_OUTPUT
    bListening       : BOOL;
    hListener        : T_HSOCKET;
    nAcceptedClients : UDINT;
    bBusy            : 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
```

Realisierung

```
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 r
eturned 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
```

6.1.1.4.2 FB_RemoteClient



Beim gesetzten bEnable-Eingang wird nach Ablauf der SERVER_ACCEPT_POOLING_TIME versucht, die Verbindungsanforderung eines Remote-Clients anzunehmen (zu akzeptieren). Der Datenaustausch zum Remote-Client wurde in einem separaten Funktionsbaustein FB_ServerDataExcha [▶ 78] 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.

Schnittstelle

```
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_TIMEOUT );
    fbClose             : FB_SocketClose := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbServerDataExcha     : FB_ServerDataExcha;
    fbAcceptTON     : TON := ( PT := PLCPRJ_ACCEPT_POLLING_TIME );
    eStep         : E_ClientSteps;
END_VAR
```

Realisierung

```
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
```

6.1.1.4.3 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. Wenn die Timeout-Zeit PLCPRJ_RECEIVE_TIMEOUT überschritten wurde oder wenn ein Fehler auftritt, wird der Empfang abgebrochen. Der nächste Lesevorgang wird nach einer Verzögerungszeit ausgeführt, wenn beim letzten Lesevorgang keine neuen Daten gelesen werden konnten. Die Systemauslastung verringert sich dadurch.

Schnittstelle

```
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
```

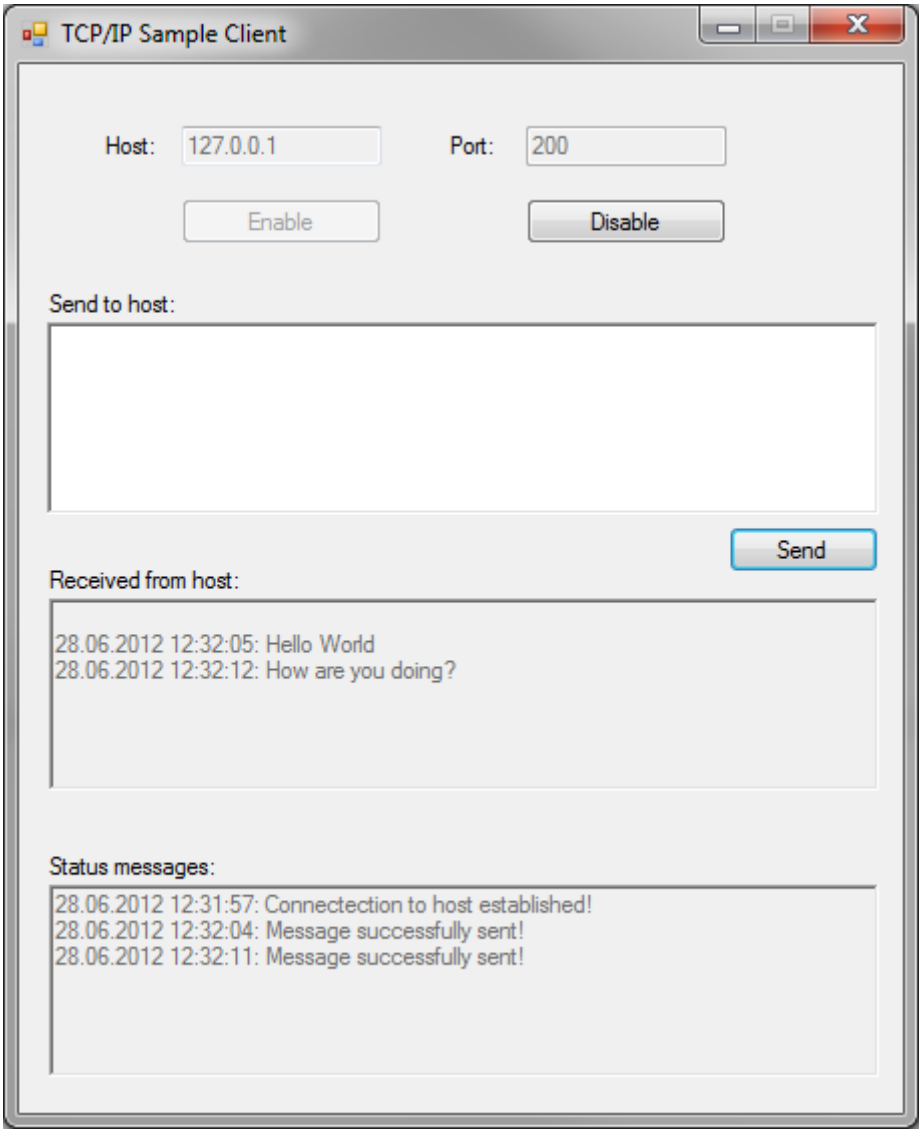
Realisierung

```
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_OV
ERFLOW );
                        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_TI
MEOUT );
                        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
```

6.1.1.5 .NET-Client

In diesem Beispielprojekt wird gezeigt, wie unter .NET4.0 in C# ein Client für den SPS-TCP/IP-Server realisiert werden kann.



Das Beispiel nutzt die .NET-Bibliotheken System.Net und System.Net.Sockets, mit denen ein Programmierer ganz einfach Socket-Funktionen nutzen kann. Durch Drücken auf **Enable** versucht die Anwendung zyklisch (je nach dem Wert von TIMERTICK in [ms]) eine Verbindung zum Server herzustellen. Beim Erfolg kann ein String mit einer maximalen Länge von 255 Zeichen über den Send-Button zum Server gesendet werden. Dieser String wird dann vom Server angenommen und an den Client zurückgesendet. Die Verbindung wird serverseitig nach Ablauf der im Serverbeispiel definierten SERVER_RECEIVE_TIMEOUT -Zeit, default: 50 Sekunden, automatisch geschlossen, wenn der Server innerhalb dieser Zeit keine neue Daten vom Client empfangen konnte.

```
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
{
 publicpartialclassForm1 : Form
 {
    /
* ######################################################################################
```

```
    * Constants
    * ########################################################################################## */
privateconstint RCVBUFFERSIZE = 256; // buffer size for receive bufferprivateconststring DEFAULTIP =
 "127.0.0.1";
    privateconststring DEFAULTPORT = "200";
    privateconstint TIMERTICK = 100;

    /
* ##########################################################################################
    * Global variables
    * ########################################################################################## */
privatestaticbool _isConnected; // signals whether socket connection is active or notprivatestaticSo
cket _socket; // object used for socket connection to TCP/IP-
ServerprivatestaticIPEndPoint _ipAddress; // contains IP address as entered in text fieldprivatestat
icbyte[] _rcvBuffer; // receive buffer used for receiving response from TCP/IP-Serverpublic Form1()
    {
     InitializeComponent();
    }

    privatevoid 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;
    }

    privatevoid 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
    * ########################################################################################## */
privatevoid timer1_Tick(object sender, EventArgs e)
    {
     if (!_isConnected)
        connect();
    }

    privatevoid connect()
    {
     /
```

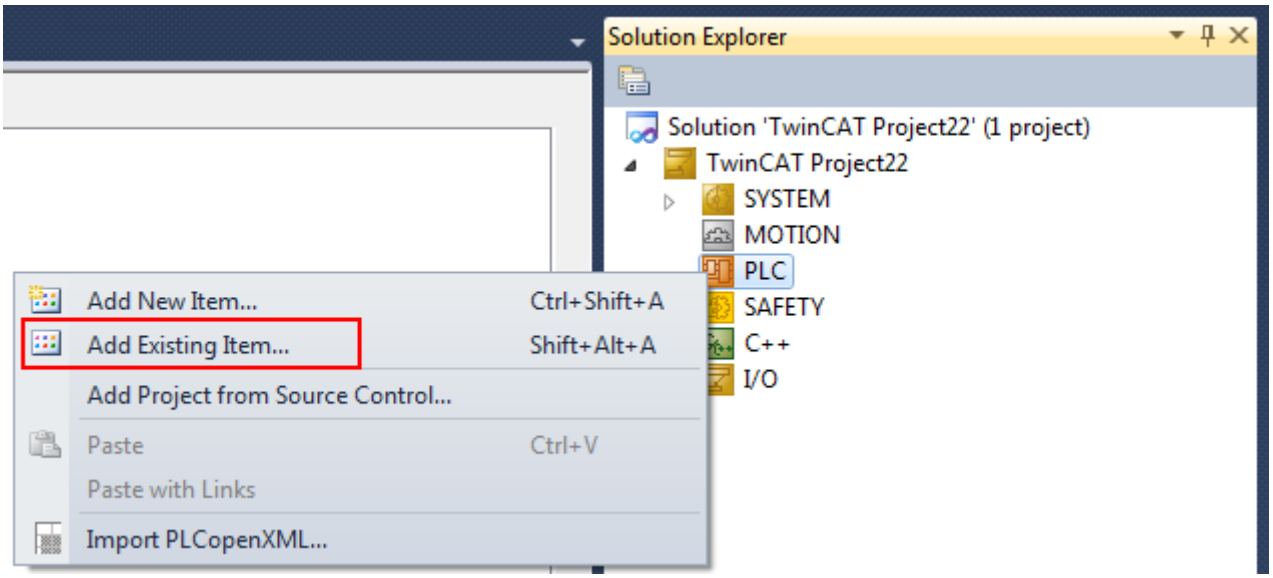
```
* ##########################################################################################
     * Connect to TCP/IP-Server using the IP address specified in the text field
     * ########################################################################################## */
try
     {
        _socket = newSocket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);
        _socket.Connect(_ipAddress);
        _isConnected = true;
        if (_socket.Connected)
         rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Connectection to host established!\n");
        else
         rtb_statMsg.AppendText(DateTime.Now.ToString() + ": A connection to the host could not be e
stablished!\n");
     }
     catch (Exception ex)
     {
        MessageBox.Show("An error occured while establishing a connection to the server: " + ex);
     }
    }

    privatevoid 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 = newASCIIEncoding();
     byte[] tempBuffer = enc.GetBytes(rtb_sendMsg.Text);
     byte[] sendBuffer = newbyte[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)
         thrownewException();
        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, null, nul
l);
         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);
     }
    }

    privatevoid 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();
     }
    }
 }
}
```

6.1.2 Beispiel02: "Echo" Client/Server (Einfachverbindung)

Dieses Beispiel nutzt die Funktionen der früheren TcSocketHelper.Lib, die nun in die Tc2_TcpIp-Bibliothek integriert ist. Es zeigt eine Client/Server-SPS-Anwendung auf Basis der Funktionen der früheren SocketHelper-Bibliothek.

Der Client sendet zyklisch einen Teststring (sToServer) zum Remote-Server. Der Server gibt diesen String unverändert an den Client zurück (sFromServer).



Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP
- Wird das Beispiel auf zwei Computern ausgeführt (ein Client und ein Server), muss die Function TF6310 auf beiden installiert sein.
- Wird das Beispiel auf einem Computer ausgeführt (z. B. Client und Server laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen.

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample02

Projektinformation

Die Standardeinstellungen für die Kommunikation in den obigen Beispielen lauten wie folgt:

- 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 Client- und Server-Anwendung auf zwei verschiedenen PCs getestet werden, müssen Port und IP Adresse entsprechend angepasst werden.

Sie können Client- und Server-Anwendung auch ohne Änderung der Einstellungen auf einem PC testen, indem Sie die Client-Anwendung in das erste SPS-Laufzeitsystem laden und die Server-Anwendung in das zweite.

Das Verhalten des SPS-Projektbeispiels wird von folgenden globalen Variablen/Konstanten bestimmt:

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.

Das SPS-Beispiel definiert und nutzt folgende interne Fehlercodes:

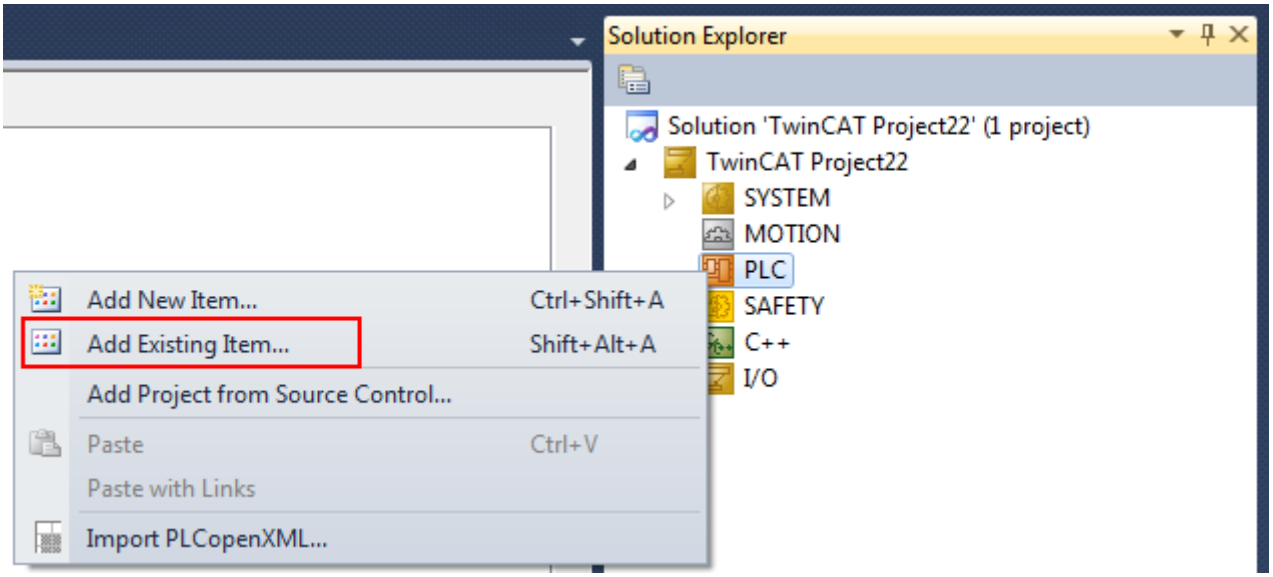
Fehlercode	Wert	Beschreibung
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	Der interne Empfangspuffer meldet einen Überlauf.
PLCPRJ_ERROR_SEND_BUFFER_OVERFLOW	16#8102	Der interne Sendepuffer meldet einen Überlauf.
PLCPRJ_ERROR_RESPONSE_TIMEOUT	16#8103	Der Server hat die Antwort in der angegebenen Timeoutzeit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

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

6.1.3 Beispiel03: "Echo" Client/Server (Mehrfachverbindung)

Dieses Beispiel nutzt die Funktionen der früheren TcSocketHelper.Lib, die nun in die Tc2_TcpIp-Bibliothek integriert ist. Es zeigt eine Client/Server-SPS-Anwendung auf Basis der Funktionen der früheren SocketHelper Bibliothek.

Der Client sendet zyklisch einen Teststring (sToServer) zum Remote-Server. Der Server gibt diesen String unverändert an den Client zurück (sFromServer). Der Unterschied zwischen diesem Beispiel und Beispiel02 ist, dass der Server bis zu fünf Verbindungen herstellen und die Client-Anwendung bis zu fünf Client-Instanzen starten kann. Jede Instanz baut eine Verbindung zum Server auf.



Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP
- Wird das Beispiel auf zwei Computern ausgeführt (ein Client und ein Server), muss die Function TF6310 auf beiden installiert sein.
- Wird das Beispiel auf einem Computer ausgeführt (z. B. Client und Server laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen.

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample03

Projektinformation

Die Standardeinstellungen für die Kommunikation in den obigen Beispielen:

- 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 Client- und Server-Anwendung auf zwei verschiedenen PCs getestet werden, müssen Port und IP Adresse entsprechend angepasst werden.

Sie können Client- und Server-Anwendung auch ohne Änderung der Einstellungen auf einem PC testen, indem Sie die Client-Anwendung in das erste SPS-Laufzeitsystem laden und die Server-Anwendung in das zweite.

Das Verhalten des SPS-Projektbeispiels wird von folgenden globalen Variablen/Konstanten bestimmt:

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.

Das SPS-Beispiel definiert und nutzt 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.).

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

6.1.4 Beispiel04: Binärdatenaustausch (Einfachverbindung)

Dieses Beispiel nutzt die Funktionen der früheren TcSocketHelper.Lib, die nunin die Tc2_TcpIp-Bibliothek integriert ist. Es zeigt eine Client/Server-SPS-Anwendung auf Basis der Funktionen der früheren SocketHelper-Bibliothek.

Dieses Beispiel bietet eine Client-Server-Anwendung für den Austausch binärer Daten. Dafür wurde ein einfaches Beispielprotokoll implementiert. Im Protokoll-Header wird die Länge der Binärdaten und ein Framezähler für die gesendeten und empfangenen Telegramme übertragen.

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.

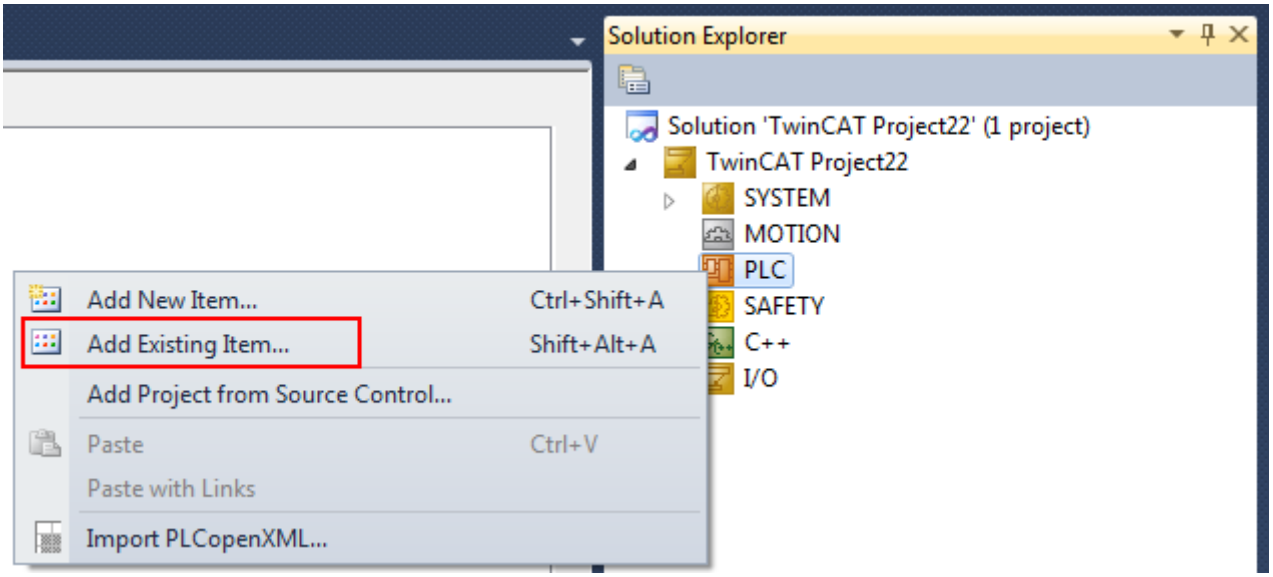
Sie können die Strukturdeklaration auf der Client und Server-Seite an Ihre Anforderungen anpassen. Die Strukturdeklaration muss aber auf beiden Seiten gleich sein.

Die maximale Größe der Struktur darf die maximale Puffergröße der Sende-/Empfangs-Fifos nicht überschreiten. Die maximale 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 bestimmte Funktion benötigen, müssen Sie die Funktionsbausteine FB_ServerApplication und FB_ClientApplication entsprechend modifizieren.



Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP
- Wird das Beispiel auf zwei Computern ausgeführt (ein Client und ein Server), muss die Function TF6310 auf beiden installiert sein.
- Wird das Beispiel auf einem Computer ausgeführt (z. B. Client und Server laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample04

Projektinformation

Die Standardeinstellungen für die Kommunikation in den obigen Beispielen:

- 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 Client- und Server-Anwendung auf zwei verschiedenen PCs getestet werden, müssen Port und IP Adresse entsprechend angepasst werden.

Sie können Client- und Server-Anwendung auch ohne Änderung der Einstellungen auf einem PC testen, indem Sie die Client-Anwendung in das erste SPS-Laufzeitsystem laden und die Server-Anwendung in das zweite.

Das Verhalten des SPS-Projektbeispiels wird von folgenden globalen Variablen/Konstanten bestimmt:

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.

Das SPS-Beispiel definiert und nutzt 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.).

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

6.1.5 Beispiel05: Binärdatenaustausch (Mehrfachverbindung)

Dieses Beispiel nutzt die Funktionen der früheren TcSocketHelper.Lib, die nun in die Tc2_TcpIp-Bibliothek integriert ist. Es zeigt eine Client/Server-SPS-Anwendung auf Basis der Funktionen der früheren SocketHelper-Bibliothek.

Dieses Beispiel bietet eine Client-Server-Anwendung für den Austausch binärer Daten. Dafür wurde ein einfaches Beispielprotokoll implementiert. Im Protokoll-Header wird die Länge der Binärdaten und ein Framezähler für die gesendeten und empfangenen Telegramme übertragen.

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.

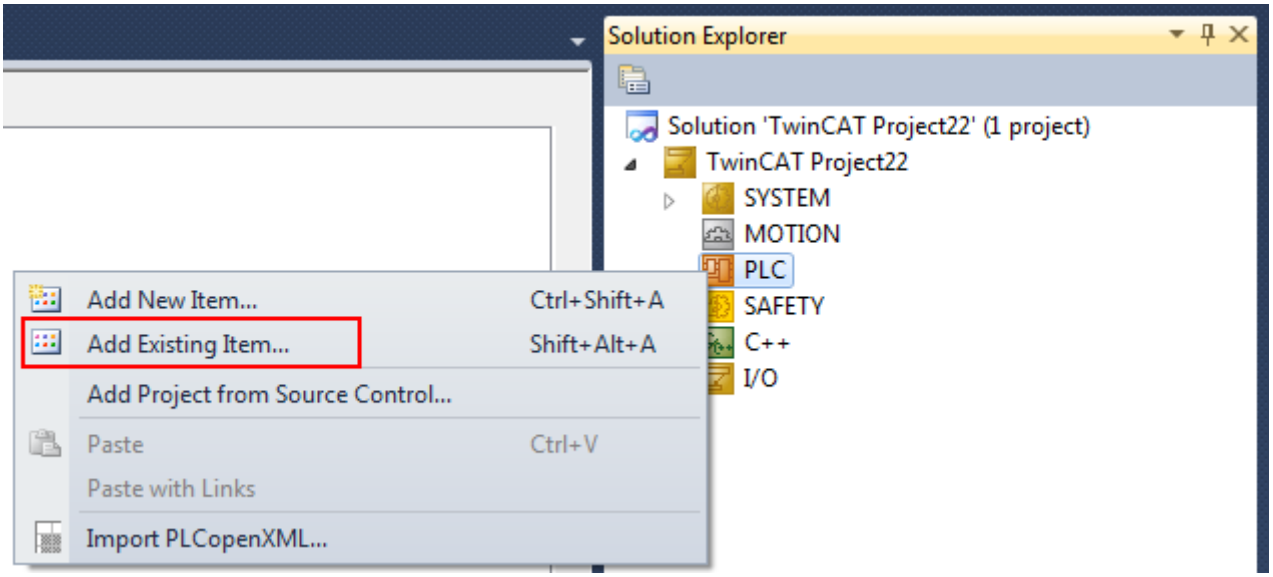
Die maximale Größe der Struktur darf die maximale Puffergröße der Sende-/Empfangs-Fifos nicht überschreiten. Die maximale Puffergröße ist durch eine Konstante festgelegt.

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 bestimmte Funktion benötigen, müssen Sie die Funktionsbausteine FB_ServerApplication und FB_ClientApplication entsprechend modifizieren.

Der Unterschied zwischen diesem Beispiel und Beispiel 04 ist, dass der Server bis zu 5 Verbindungen herstellen und die Client-Anwendung bis zu fünf Client-Instanzen haben kann. Jede Instanz baut eine Verbindung zum Server auf.



Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP
- Wird das Beispiel auf zwei Computern ausgeführt (ein Client und ein Server), muss die Function TF6310 auf beiden installiert sein.
- Wird das Beispiel auf einem Computer ausgeführt (z. B. Client und Server laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen.

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample05

Projektinformation

Die Standardeinstellungen für die Kommunikation in den obigen Beispielen:

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

Wenn Client- und Server-Anwendung auf zwei verschiedenen PCs getestet werden, müssen Port- und IP-Adresse entsprechend angepasst werden.

Sie können Client- und Server-Anwendung auch ohne Änderung der Einstellungen auf einem PC testen, indem Sie die Client-Anwendung in das erste SPS-Laufzeitsystem laden und die Server-Anwendung in das zweite.

Das Verhalten des SPS-Projektbeispiels wird von folgenden globalen Variablen/Konstanten bestimmt:

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.

Das SPS-Beispiel definiert und nutzt 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.).

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

6.1.6 Beispiel06: "Echo" Client/Server mit TLS (Basisbausteine)

Das folgende Beispiel basiert im Wesentlichen auf Sample01 und zeigt eine beispielhafte Implementierung eines „Echo“-Client/Server-Systems. Der Client sendet in bestimmten Abständen (z. B. jede Sekunde) einen Test-String zum Server. Der Remote-Server sendet diesen String wieder zurück an den Client.

Als Unterschied zu Sample01 wird die Kommunikationsverbindung in diesem Beispiel exemplarisch über TLS mit Client/Server Zertifikaten abgesichert. Die Zertifikate sind hierbei nicht Bestandteil des Samples und müssen vom Anwender erstellt werden.

Im Wesentlichen veranschaulicht dieses Beispiel somit die Verwendung der Funktionsbausteine FB_TlsSocketConnect [▶ 37], FB_TlsSocketCreate [▶ 40], FB_TlsSocketListen [▶ 38], FB_TlsSocketAddCa [▶ 41], FB_TlsSocketAddCrl [▶ 42] und FB_TlsSocketSetCert [▶ 43]. Diese wurden entsprechend in die State Machine des Client- und Server-Beispiels aus Sample01 integriert.

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample06

6.1.7 Beispiel07: "Echo" Client/Server mit TLS-PSK (Basisbausteine)

Das folgende Beispiel basiert im Wesentlichen auf Sample01 und zeigt eine beispielhafte Implementierung eines „Echo“-Client/Server-Systems. Der Client sendet in bestimmten Abständen (z. B. jede Sekunde) einen Test-String zum Server. Der Remote-Server sendet diesen String wieder zurück an den Client.

Als Unterschied zu Sample01 wird die Kommunikationsverbindung in diesem Beispiel exemplarisch über TLS mit einem Pre-Shared-Key (PSK) abgesichert.

Im Wesentlichen veranschaulicht dieses Beispiel somit die Verwendung der Funktionsbausteine FB_TlsSocketConnect [▶ 37], FB_TlsSocketCreate [▶ 40], FB_TlsSocketListen [▶ 38] und FB_TlsSocketSetPsk [▶ 44]. Diese wurden entsprechend in die State Machine des Client- und Server-Beispiels aus Sample01 integriert.

Projektdownloads

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/TCP/Sample07

6.2 UDP

6.2.1 Beispiel01: Peer-to-Peer Anwendung

6.2.1.1 Übersicht

Das folgende Beispiel zeigt die Implementierung einer einfachen Peer-to-Peer-Anwendung in der SPS. Es umfasst zwei SPS-Projekte (PeerA und PeerB) sowie eine .NET-Anwendung, die als separater Peer agiert. Alle Peer-Anwendungen senden einen Test-String zu einem Remote-Peer und erhalten gleichzeitig Strings von einem Remote-Peer. Die erhaltenen Strings werden auf dem Bildschirm in einer Messagebox angezeigt. Sie können das Beispiel beliebig nutzen und anpassen.

Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP
- Wenn Sie für die Ausführung des Beispiels zwei PCs nutzen, muss TF6310 auf beiden installiert sein.
- Wird das Beispiel auf einem Computer ausgeführt (z. B. PeerA und PeerB laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen.
- Zum Ausführen des .NET-Beispielclients ist nur .NET Framework 4.0 nötig.

Projektdownloads

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

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/UDP/Sample01

https://github.com/Beckhoff/TF6310_Samples/tree/master/C%23/SampleClientUdp

Projektbeschreibung

Unter den nachfolgenden Links finden Sie die Dokumentation der einzelnen Komponenten. Ein separater Artikel erklärt Schritt für Schritt, wie die SPS-Beispiele eingerichtet und gestartet werden.

- Integration in TwinCAT und Test [▶ 93] (Start der SPS-Beispiel)
- SPS-Teilnehmer A und B [▶ 95] (Peer-to-Peer SPS-Applikation)
- .NET-Teilnehmer [▶ 99] (.NET-Beispielclient)

Zusätzliche Funktionen der SPS Beispielprojekte

In den SPS-Beispielen werden einige Funktionen, Konstanten und Funktionsbausteine benutzt, die im Folgenden kurz beschrieben werden sollen:

Fifo-Funktionsbaustein

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

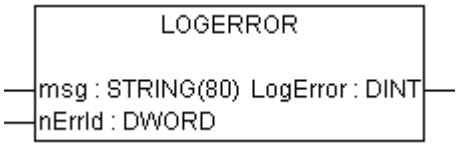
Hierbei handelt es sich um einen einfachen Fifo-Funktionsbaustein. Eine Instanz von diesem Baustein wird als „Sende-Fifo“ und eine als „Empfangs-Fifo“ benutzt. Die zu sendenden Nachrichten werden in den Sende-Fifo und die empfangenen in den Empfangs-Fifo abgelegt. Die bOk-Ausgangsvariable wird auf FALSE gesetzt, wenn bei der letzten 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 dotte
d address. *)
    nRemotePort : UDINT; (* Remote Internet Protocol (IP) port. *)
    msg         : STRING; (* Udp packet data *)
END_STRUCT
END_TYPE
```

LogError-Funktion

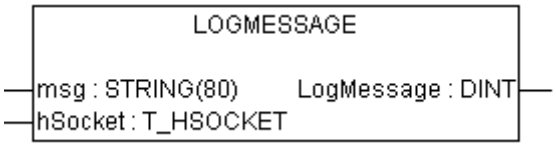
```
FUNCTION LogError : DINT
```



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

LogMessage-Funktion

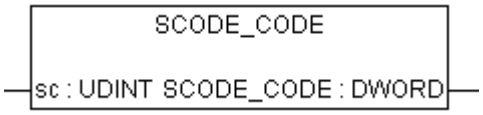
```
FUNCTION LogMessage : DINT
```



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 zuerst auf TRUE gesetzt werden.

SCODE_CODE-FUnktion

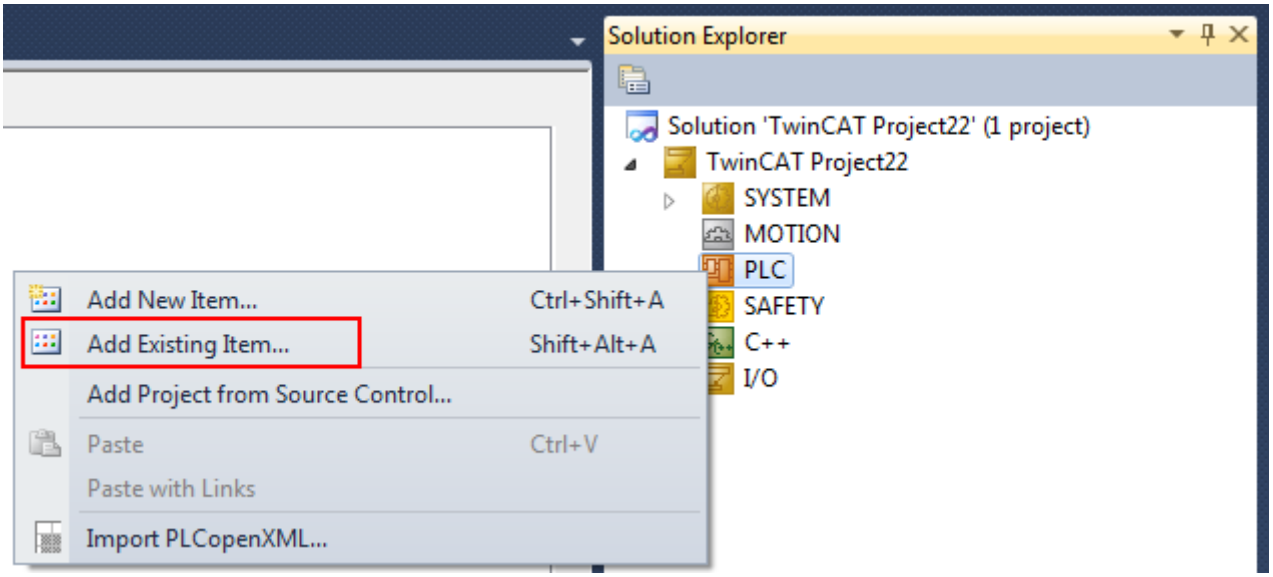
```
FUNCTION SCODE_CODE : DWORD
```



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

6.2.1.2 Integration in TwinCAT und Test

Die SPS-Beispiele werden als TwinCAT-3-SPS-Projektdateien zur Verfügung gestellt. Um ein SPS-Projekt in das TwinCAT XAE zu importieren, erstellen Sie zunächst eine neue TwinCAT 3 Solution. Wählen Sie anschließend im Kontextmenü des PLC-Knoten den Befehl **Add Existing Item** und in dem sich öffnenden Dialog die heruntergeladenen Beispieldatei (*Plc 3.x Project archive (*.tpzip)* als Dateityp auswählen). Nach Bestätigung des Dialogs wird das SPS-Projekt der Solution hinzugefügt.



Zum Start des Beispiels werden zwei Computer benötigt. 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 mit zwei Computern:

- Gerät A befindet sich auf dem lokalen PC und hat die IP-Adresse '10.1.128.21'
- Gerät B befindet sich auf dem Remote-PC und hat die IP-Adresse '172.16.6.195'.

Gerät A

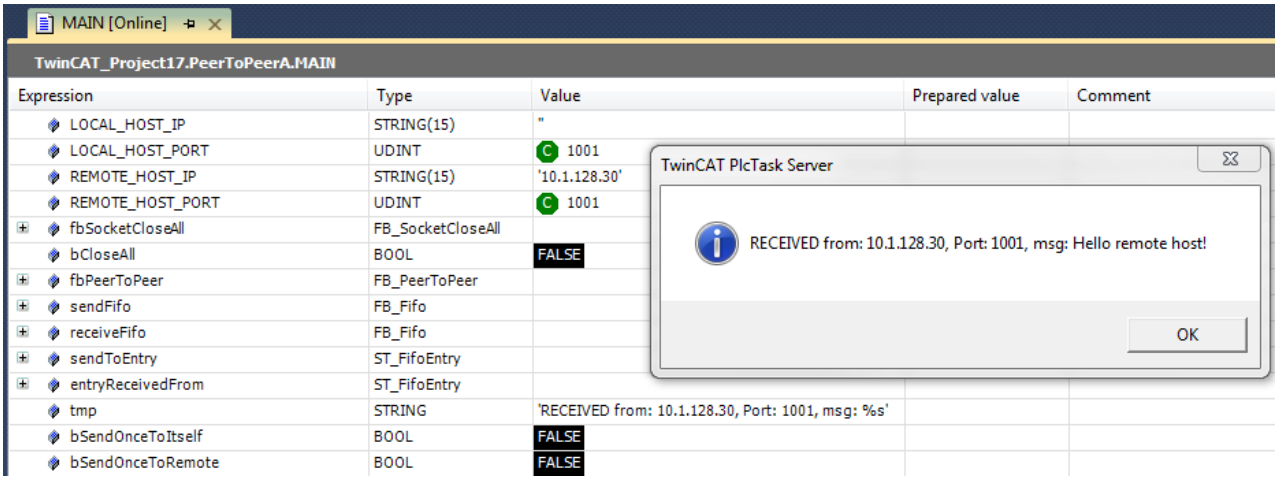
Zum Installieren des Beispiels auf Gerät A gehen Sie wie folgt vor:

- Erstellen Sie in TwinCAT XAE eine neue TwinCAT 3 Solution und importieren Sie das Peer-to-Peer SPS-Projekt für Gerät A.
- Passen Sie die Konstante REMOTE_HOST_IP in POU MAIN an die tatsächliche IP-Adresse Ihres Remote-Systems an (in unserem Beispiel '172.16.6.195').
- Aktivieren Sie die Konfiguration und starten die SPS-Laufzeit. (Vergessen Sie nicht vorher eine Lizenz für TF6310 TCP/IP zu erzeugen)

Gerät B

Zum Installieren des Beispiels auf Gerät B gehen Sie wie folgt vor:

- Erstellen Sie in TwinCAT XAE eine neue TwinCAT 3 Solution und importieren Sie das Peer-to-Peer SPS-Projekt für Gerät B.
- Passen Sie die Konstante REMOTE_HOST_IP in POU MAIN an die tatsächliche IP-Adresse des Gerätes A an (in unserem Beispiel '10.1.128.21').
- Aktivieren Sie die Konfiguration und starten die SPS-Laufzeit. (Vergessen Sie nicht vorher eine Lizenz für TF6310 TCP/IP zu erzeugen)
- Loggen Sie sich in die SPS-Laufzeit ein und setzen Sie in der POU "MAIN" die boolesche Variable bSendOnceToRemote auf TRUE.
- Kurz danach sollte eine Messagebox auf Gerät A erscheinen. Sie können diesen Schritt nun auf Gerät A wiederholen. Jetzt sollte die Messagebox entsprechend auf Gerät B erscheinen.

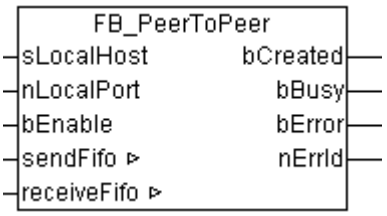


6.2.1.3 SPS-Teilnehmer A und B

Die benötigte Funktionalität wurde in dem Funktionsbaustein FB_PeerToPeer gekapselt. Jeder der Kommunikationspartner 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 Datenaustausch 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.

Name	Default-Wert	Beschreibung
g_sTcIpConnSvrAddr	''	Die Netzwerkadresse des TwinCAT TCP/IP Connection Servers. Default: Leerstring (der Server befindet sich auf dem lokalen PC)
bLogDebugMessages	TRUE	Aktiviert/deaktiviert das Schreiben von Nachrichten ins Logbuch des Betriebssystems
PLCPRJ_ERROR_SENDFIFO_OVERFLOW	16#8103	Fehlercode Beispielprojekt: Der Sende-Fifo ist voll.
PLCPRJ_ERROR_RECFIFO_OVERFLOW	16#8104	Fehlercode Beispielprojekt: Der Empfangs-Fifo ist voll.

FUNCTION_BLOCK FB_PeerToPeer



Schnittstelle

```
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
```

Realisierung

```
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
```

Programm MAIN

Nach einem Programm-Download oder SPS-Reset müssen die vorher geöffneten Sockets geschlossen werden. Dies geschieht beim SPS-Start durch den einmaligen Aufruf einer Instanz des FB_SocketCloseAll [▶ 23]-Funktionsbausteins. Bei einer steigender 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.

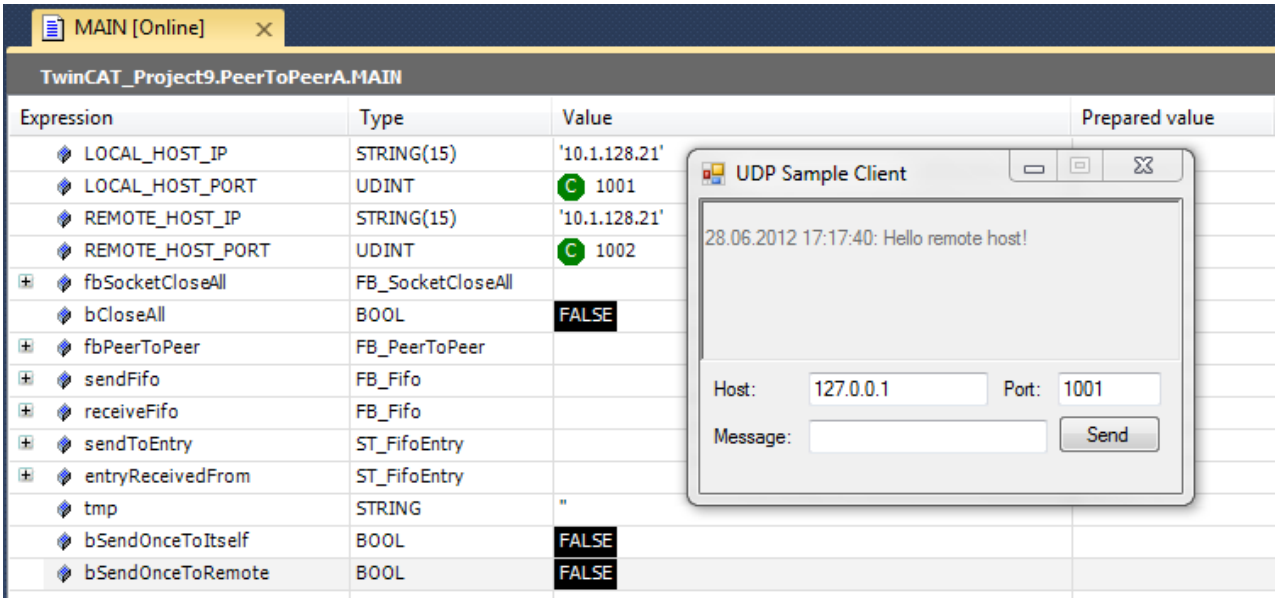
```
PROGRAM MAIN
VAR CONSTANT
    LOCAL_HOST_IP      : STRING(15)     := '';
    LOCAL_HOST_PORT    : UDINT         := 1001;
    REMOTE_HOST_IP     : STRING(15)     := '172.16.2.209';
    REMOTE_HOST_PORT   : UDINT         := 1001;
END_VAR
VAR
    fbSocketCloseAll    : FB_SocketCloseAll;
    bCloseAll     : 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 => sen
d it to itself *)
       sendToEntry.sRemoteHost         := LOCAL_HOST_IP;             (* sRemoteHost == sLocalHost =>
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
```

6.2.1.4 .NET-Teilnehmer

In diesem Beispiel wird demonstriert, wie ein passender .NET-Kommunikationspartner für das SPS-Peer-to-Peer-Gerät A realisiert werden kann. Verwenden Sie dieses Beispiel nur in Zusammenhang mit dem SPS-Projekt PeerToPeerA.



Der .NET Sample Client dient dazu, einzelne UDP-Datenpakete an einen UDP-Server, in diesem Fall das SPS-Projekt PeerToPeerA zu senden.

Download

Download des Test Clients.

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

Wie das Beispiel funktioniert

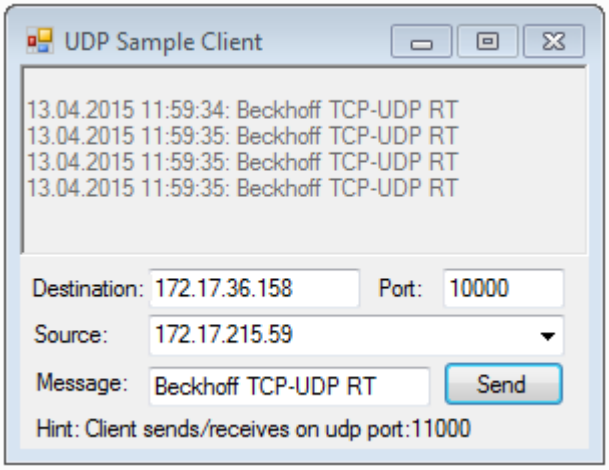
Das Beispiel nutzt die .NET-Bibliotheken System.Net und System.Net.Sockets zur Implementierung des UDP-Client (class UdpClient). Während ein Hintergrund-Thread auf eingehende UDP-Pakete achtet, kann durch Anklicken des Sende-Buttons und mittels Angabe von IP-Adresse und Portnummer ein String an ein Remote-Gerät geschickt werden.

Zum besseren Verständnis des Artikels stellen Sie sich im Verlauf dieses Artikels das folgende Setup vor:

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

Beschreibung

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



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

Der Client ist über die Oberfläche konfigurierbar:

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

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

6.2.2 Beispiel02: Multicast Anwendung

Dieses Beispiel demonstriert, wie Multicast-Pakete über UDP gesendet und empfangen werden können.

Client und Server senden sich zyklisch einen Wert über eine Multicast-IP-Adresse.

Client und Server sind durch zwei getrennte SPS-Anwendungen realisiert, die in einer einzelnen TwinCAT 3 Solution enthalten sind.

Systemvoraussetzungen

- TwinCAT 3 Build 3093 oder höher
- TwinCAT 3 Function TF6310 TCP/IP 1.0.64 oder höher
- TwinCAT 3 Library Tc2_TcpIp Version 3.2.64.0 oder höher
- Wird das Beispiel auf einem Computer ausgeführt (z. B. Client und Server laufen in zwei separaten SPS-Laufzeiten), müssen beide SPS-Laufzeiten in separaten Tasks laufen.

Projektdownload

https://github.com/Beckhoff/TF6310_Samples/tree/master/PLC/UDP/Sample02

7 Anhang

7.1 OSI-Modell

Der folgende Artikel beinhaltet eine kurze Einführung in das OSI-Modell und beschreibt, wie es Einfluss auf unsere alltägliche Netzwerkkommunikation nimmt. Beachten Sie, dass dieser Artikel eine detailliertere Dokumentation zu diesem Thema nicht überflüssig machen, sondern nur einen Einblick geben soll.

Das OSI-(Open Systems Interconnection)-Modell beschreibt die Standardisierung von Funktionen in einem Kommunikationssystem über ein abstraktes Schichtenmodell. Jede Schicht (engl. "Layer") beschreibt bestimmte Funktionen der Kommunikation zwischen Geräten im Netzwerk. Jede Schicht kommuniziert nur mit der direkt darüber- oder darunterliegenden Schicht.

OSI model

Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

Beispiel: Wenn Sie mit Ihrem Webbrowser die Adresse "http://www.beckhoff.com" aufrufen, werden dabei, beginnend ab Layer 7 folgende Protokolle genutzt: HTTP → TCP → IP → Ethernet. Bei Eingabe von "https://www.beckhoff.com" würden hingegen die Protokolle HTTP → SSL → TCP → IP → Ethernet genutzt.

Mit der TwinCAT 3 Function TF6310 TCP/IP können netzwerkfähige SPS-Programme entwickelt werden, die TCP oder UDP als Transportprotokoll nutzen. So können SPS-Programmierer eigene Netzwerkprotokolle in der Anwendungsschicht implementieren und eine eigene Nachrichtenstruktur zur Kommunikation mit Remote-Systemen definieren.

7.2 KeepAlive-Konfiguration

Die Versendung von KeepAlive-Nachrichten durch das TCP-Protokoll ermöglicht die Überprüfung, ob eine Verbindung im Leerlauf weiterhin aktiv ist. Seit Version 1.0.47 des TwinCAT TCP/IP Servers (TF6310 TCP/IP) wird die KeepAlive-Konfiguration des Windows-Betriebssystems verwendet, welche sich über die folgenden Windows-Registry-Einträge konfigurieren lässt.

Die folgende Dokumentation ist ein Auszug aus einem Microsoft Technet-Artikel.

KeepAliveTime

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

Data type	Range	Default value
REG_DWORD	0x1–0xFFFFFFFF (*milliseconds*)	0x6DDD00 (*7,200,000 milliseconds = 2 hours*)

Description

Determines how often TCP sends keep-alive transmissions. TCP sends keep-alive transmissions to verify that an idle connection is still active.This entry is used when the remote system is responding to TCP. Otherwise, the interval between transmissions is determined by the value of the KeepAliveInterval entry.By default, keep-alive transmissions are not sent. The TCP keep-alive feature must be enabled by a program, such as Telnet, or by an Internet browser, such as Internet Explorer.

KeepAliveInterval

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

Data type	Range	Default value
REG_DWORD	0x1–0xFFFFFFFF (*milliseconds*)	0x3E8 (*1,000 milliseconds = 1 second*)

Description

Determines how often TCP repeats keep-alive transmissions when no response is received. TCP sends keep-alive transmissions to verify that idle connections are still active. This prevents TCP from inadvertently disconnecting active lines.

7.3 Fehlercodes

7.3.1 Übersicht der Fehlercodes

Codes (hex)	Codes (dez)	Fehlerquelle	Beschreibung
0x00000000-0x00007800	0-30720	TwinCAT Systemfehler-Codes [▶ 105]	TwinCAT Systemfehler (ADS-Fehlercodes inklusive)
0x00008000-0x000080FF	32768-33023	Interne TwinCAT TCP/IP Connection Server Fehlercodes [▶ 103]	Interne Fehler des TwinCAT TCP/IP Connection Servers
0x80070000-0x8007FFFF	2147942400-2148007935	Fehlerquelle= Code - 0x80070000 = Win32 Systemfehler-Codes	Win32 Systemfehler (Windows Sockets Fehlercodes inklusive)

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1	PC, CX (x86) oder CX (ARM)	Tc2_TcpIp

7.3.2 Interne Fehlercodes des TwinCAT TCP/IP Connection Servers

Code (hex)	Code (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.
0x00008006	32774	TCPADSERROR_HOST_NOT_FOUND	Wird zurückgegeben, wenn das Zielsystem nicht erreichbar ist.
0x00008080	32896	TCPADSERROR_TLS_INVALID_STATE	Wird zurückgegeben, wenn FB_TlsSocketAddCa, FB_TlsSocketAddCrl , FB_TlsSocketSetCert oder FB_TlsSocketSetPsk aufgerufen werden und bereits ein Connect aufgerufen wurde.
0x00008081	32897	TCPADSERROR_TLS_CA_NOTFOUND	Wird zurückgegeben, wenn das angegebene CA Zertifikat nicht gefunden wurde.
0x00008082	32898	TCPADSERROR_TLS_CERT_NOTFOUND	Wird zurückgegeben, wenn die angegebene Zertifikatsdatei nicht gefunden wurde.
0x00008083	32899	TCPADSERROR_TLS_KEY_NOTFOUND	Wird zurückgegeben, wenn die angegebene Datei mit dem Private Key nicht gefunden wurde.
0x00008084	32900	TCPADSERROR_TLS_CA_INVALID	Wird zurückgegeben, wenn das angegebene CA Zertifikat nicht eingelesen werden konnte oder ungültig ist.
0x00008085	32901	TCPADSERROR_TLS_CERT_INVALID	Wird zurückgegeben, wenn die angegebene Zertifikatsdatei nicht eingelesen werden konnte oder ungültig ist.
0x00008086	32902	TCPADSERROR_TLS_KEY_INVALID	Wird zurückgegeben, wenn der angegebene Private Key nicht eingelesen werden konnte oder ungültig ist.
0x00008087	32903	TCPADSERROR_TLS_VERIFY_FAIL	Wird zurückgegeben, wenn beim TLS Handshake die Gegenstelle nicht verifiziert werden konnte.
0x00008088	32904	TCPADSERROR_TLS_SETUP	Wird zurückgegeben, wenn ein allgemeiner Fehler bei der Einrichtung der TLS Verbindung aufgetreten ist.
0x00008089	32905	TCPADSERROR_TLS_HANDSHAKE_FAIL	Wird zurückgegeben, wenn ein Fehler beim TLS Handshake aufgetreten ist. Üblicherweise funktioniert der Handshake immer. Wenn es jedoch zu Verbindungsproblemen während des Handshakes kommt, so kann dieser fehlschlagen.
0x0000808A	32906	TCPADSERROR_TLS_CIPHER_INVALID	Wird zurückgegeben, wenn eine ungültige Cipher Suite angegeben wurde.
0x0000808B	32907	TCPADSERROR_TLS_VERSION_INVALID	Wird zurückgegeben, wenn eine ungültige TLS Version angegeben wurde.
0x0000808C	32908	TCPADSERROR_TLS_CRL_INVALID	Wird zurückgegeben, wenn die angegebene Certificate Revocation List (CRL) ungültig ist.
0x0000808D	32909	TCPADSERROR_TLS_INTERNAL_ERROR	Wird zurückgegeben, wenn ein interner Fehler bei der Einrichtung der TLS Verbindung aufgetreten ist.
0x0000808E	32910	TCPADSERROR_TLS_PSK_SETUP_ERROR	Wird zurückgegeben, wenn ein Fehler bei der Verwendung eines PreSharedKey (PSK) für TLS aufgetreten ist.
0x0000808F	32911	TCPADSERROR_TLS_CN_MISMATCH	Wird zurückgegeben, wenn der CommonName im Zertifikat der Gegenstelle nicht zu dem verwendeten Hostname oder der IP-Adresse passt.

Code (hex)	Code (dez)	Symbolische Konstante	Beschreibung
0x00008090	32912	TCPADSERROR_TLS_CERT_EXPIRED	Wird zurückgegeben, wenn das Zertifikat der Gegenstelle abgelaufen ist.
0x00008091	32913	TCPADSERROR_TLS_CERT_REVOKED	Wird zurückgegeben, wenn das Zertifikat der Gegenstelle zurückgezogen wurde.
0x00008092	32914	TCPADSERROR_TLS_CERT_MISSING	Wird zurückgegeben, wenn die Gegenstelle kein Zertifikat übermittelt hat.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1	PC, CX (x86) oder CX (ARM)	Tc2_TcpIp

7.3.3 Fehlersuche/Diagnose

- Bei Verbindungsproblemen kann der PING-Befehl dazu benutzt werden, um festzustellen, ob die Fremdsteuerung über die Netzwerkverbindung erreichbar ist. Wenn dies nicht der Fall ist, überprüfen Sie Netzwerkkonfiguration und Firewall-Einstellungen.
- Eine komplette Aufzeichnung der Netzwerkkommunikation kann mit Sniffer-Tools wie Wireshark durchgeführt werden. Die Aufnahme kann dann vom Beckhoff-Supportpersonal analysiert werden.
- Überprüfen Sie die in dieser Dokumentation beschriebenen Hardware- und Softwareanforderungen (Versionen von TwinCAT und CE Image, usw.).
- Beachten Sie die Hinweise bezüglich der Softwareinstallation in dieser Dokumentation (z. B. Installation von CAB-Dateien auf einer CE Plattform).
- Überprüfen Sie, ob die Eingangsparameter, die Sie an die Funktionsbausteine übergeben, richtig sind (Verbindungshandle von Netzwerkadresse, Portnummer, Daten, usw..). Überprüfen Sie, ob der Funktionsbaustein einen Fehlercode ausgibt. Die Dokumentation zu den Fehlercodes finden Sie hier: Übersicht der Fehlercodes [▶ 102]
- Überprüfen Sie, ob der andere Kommunikationspartner/die Software/das Gerät einen Fehlercode ausgibt.
- Aktivieren Sie die Debug-Ausgabe in der TcSocketHelper.Lib beim Herstellen/Trennen der Verbindung (Keyword: CONNECT_MODE_ENABLEDBG). Öffnen Sie den TwinCAT System Manager und aktivieren Sie das LogView-Fenster. Analysieren/Prüfen Sie die Debug-Ausgabestrings.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1	PC, CX (x86) oder CX (ARM)	Tc2_TcpIp

7.3.4 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 105]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 106]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 107]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 109]... (0x9811_1000 ...)

Globale Fehlercodes

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

Router Fehlercodes

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

Allgemeine ADS Fehlercodes

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

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

RTime Fehlercodes

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

Spezifische positive HRESULT Return Codes:

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

TCP Winsock-Fehlercodes

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

7.4 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Mehr Informationen:
www.beckhoff.de/tf6310

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