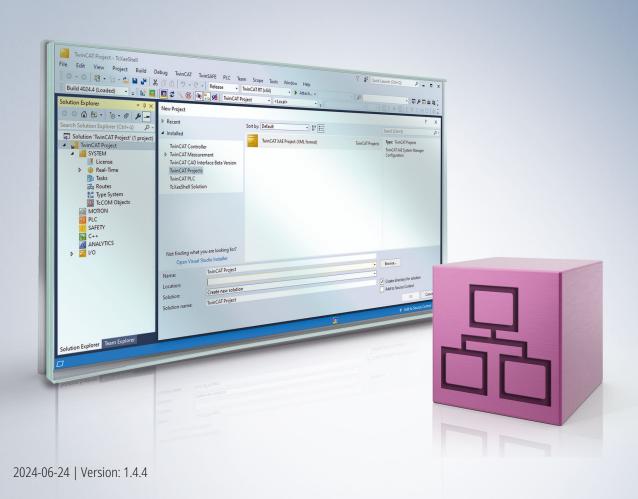
# **BECKHOFF** New Automation Technology

Manual | EN

## TF6310

TwinCAT 3 | TCP/IP





## **Table of contents**

1	Fore	word		5
	1.1	Notes o	on the documentation	5
	1.2	For you	r safety	5
	1.3	Notes o	on information security	7
2	Over	view		ε
	2.1	Compar	rison TF6310 TF6311	8
3	Insta	llation		ç
-	3.1		requirements	
	3.2	•	tion	
	3.3		tion from TwinCAT 4026	
	3.4		tion Windows CE	
	3.5		ng	
	3.6		on from TwinCAT 2	
4	Tech	•	roduction	
5				
	5.1		n blocks	
		5.1.1	FB_SocketConnect	
		5.1.2	FB_SocketClose	
		5.1.3	FB_SocketCloseAll	
		5.1.4	FB_SocketListen	
		5.1.5	FB_SocketAccept	
		5.1.6	FB_SocketSend	
		5.1.7	FB_SocketReceive	
		5.1.8	FB_SocketUdpCreate	
		5.1.9	FB_SocketUdpSendTo	
		5.1.10 5.1.11	FB_SocketUdpReceiveFrom  FB SocketUdpAddMulticastAddress	
		5.1.11	FB_SocketUdpDropMulticastAddress	
		5.1.12	FB_TIsSocketConnect	
		5.1.14	FB TIsSocketListen	
		5.1.15	FB_TIsSocketCreate	
		5.1.16	FB_TlsSocketAddCa	
		5.1.17	FB TIsSocketAddCrl	
		5.1.18	FB_TlsSocketSetCert	
		5.1.19	FB TlsSocketSetPsk	
		5.1.20	Helper	
	5.2	Function	ns	
		5.2.1	F_CreateServerHnd	
		5.2.2	HSOCKET_TO_STRING	
		5.2.3	HSOCKET_TO_STRINGEX	
		5.2.4	SOCKETADDR_TO_STRING	
	5.3	Data typ	pes	
		5.3.1	E_SocketAcceptMode	54



		5.3.2	E_SocketConnectionState	55
		5.3.3	E_SocketConnectionlessState	55
		5.3.4	E_WinsockError	56
		5.3.5	ST_SockAddr	57
		5.3.6	ST_TIsConnectFlags	58
		5.3.7	ST_TlsListenFlags	58
		5.3.8	T_HSERVER	59
		5.3.9	T_HSOCKET	59
	5.4	Global	onstants	60
		5.4.1	Library version	60
		5.4.2	Parameter list	61
6	Samp	oles		62
	6.1	TCP		62
		6.1.1	Sample01: "Echo" client/server (base blocks)	62
		6.1.2	Sample02: "Echo" client /server	81
		6.1.3	Sample03: "Echo" client/server	82
		6.1.4	Sample04: Binary data exchange	84
		6.1.5	Sample05: Binary data exchange	86
		6.1.6	Sample06: "Echo" client/server with TLS (basic modules)	87
		6.1.7	Sample07: "Echo" client/server with TLS-PSK (basic modules)	88
	6.2	UDP		88
		6.2.1	Sample01: Peer-to-peer communication	88
		6.2.2	Sample02: Multicast	96
7	Appe	ndix		97
	7.1	OSI mod	del	97
	7.2	KeepAli	ve configuration	97
	7.3	Error co	des	98
		7.3.1	Overview of the error codes	98
		7.3.2	Internal error codes of the TwinCAT TCP/IP Connection Server	99
		7.3.3	Troubleshooting/diagnostics	. 101
		7.3.4	ADS Return Codes	. 101
	7.4	Support	and Service	. 105



### 1 Foreword

### 1.1 Notes on the documentation

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

For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.

The qualified personnel is obliged to always use the currently valid documentation.

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

#### **Disclaimer**

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without notice.

No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

#### **Trademarks**

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

If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

#### **Patents**

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

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702 and similar applications and registrations in several other countries.



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

#### Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The distribution and reproduction of this document as well as the use and communication of its contents without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

### 1.2 For your safety

#### Safety regulations

Read the following explanations for your safety.

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



#### **Exclusion of liability**

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

#### Personnel qualification

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

#### Signal words

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

#### Personal injury warnings

#### **▲ DANGER**

Hazard with high risk of death or serious injury.

#### **⚠ WARNING**

Hazard with medium risk of death or serious injury.

#### **A** CAUTION

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

#### Warning of damage to property or environment

#### NOTICE

The environment, equipment, or data may be damaged.

#### Information on handling the product



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



### 1.3 Notes on information security

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

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

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

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



### 2 Overview

The TwinCAT TCP/IP Connection Server enables the implementation of one or several TCP/IP servers/ clients in the TwinCAT PLC. This gives a PLC programmer the possibility to develop own network protocols of the application layer (OSI model) directly in a PLC program. The communication connection can optionally be secured via TLS.

#### **Product components**

The product TF6310 TCP/IP consists of the following components, which will be delivered by the setup:

- PLC library: Tc2\_Tcplp library (implements basic TCP/IP and UDP/IP functionalities).
- Background program: TwinCAT TCP/IP Connection Server (process which is used for communication).

### **2.1 Comparison TF6310 TF6311**

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

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

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

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



### 3 Installation

### 3.1 System requirements

The following system requirements must be met for the function TF6310 TCP/IP to work properly.

Technical data	Description
Operating system	Windows 7, 10
	Windows CE 6/7
	Windows Embedded Standard 2009
	Windows Embedded 7
	TwinCAT/BSD
Target platforms	PC architecture (x86, x64, ARM)
TwinCAT Version	TwinCAT2, TwinCAT3
TwinCAT installation level	TwinCAT2 CP, PLC, NC-PTP
	TwinCAT3 XAE, XAR, ADS
Required TwinCAT license	TS6310 (for TwinCAT2)
	TF6310 (for TwinCAT3)



#### **Support of TLS**



Please note that the TLS function blocks are not available under Windows CE.

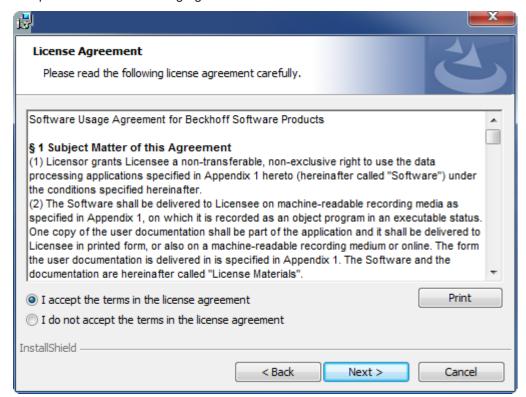
### 3.2 Installation

The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

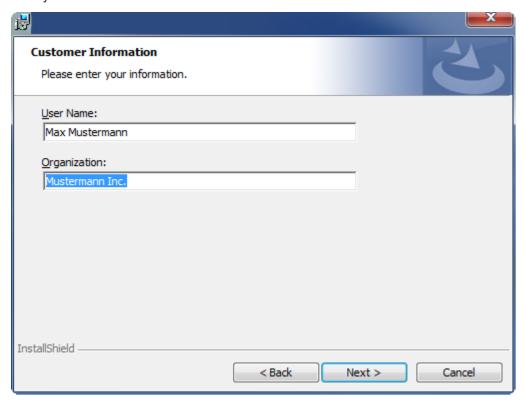
- √ The TwinCAT 3 Function setup file was downloaded from the Beckhoff website.
- 1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.
  - ⇒ The installation dialog opens.



2. Accept the end user licensing agreement and click Next.

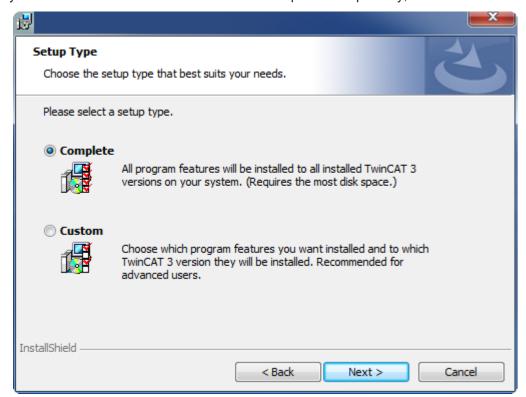


3. Enter your user data.

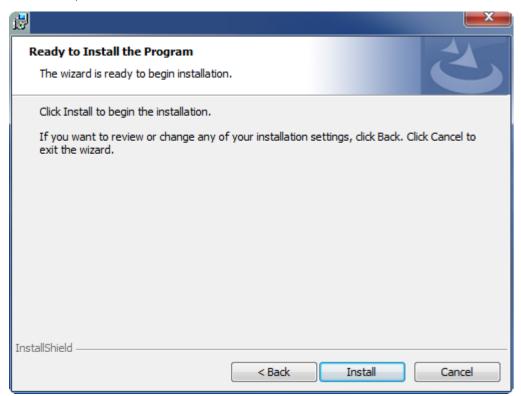




4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.



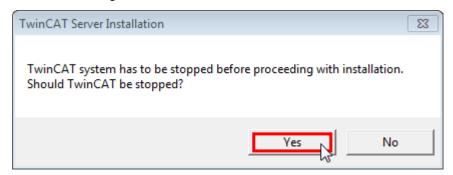
5. Select **Next**, then **Install** to start the installation.



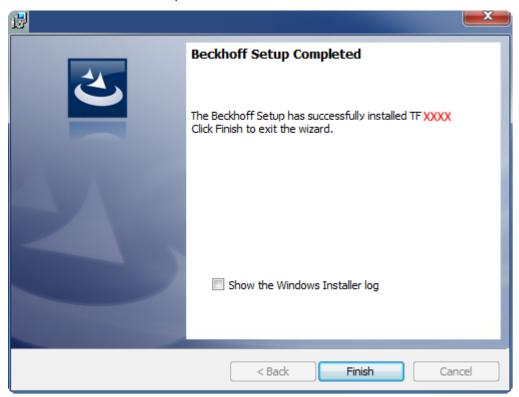
⇒ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.



6. Confirm the dialog with Yes.



7. Select **Finish** to exit the setup.



⇒ The TwinCAT 3 Function has been successfully installed.

### 3.3 Installation from TwinCAT 4026

#### TwinCAT Package Manager

If you are using TwinCAT 3.1 Build 4026 (and higher) on the Microsoft Windows operating system, you can install this function via the TwinCAT Package Manager, see <u>installation documentation</u>.

Normally you install the function via the corresponding workload; however, you can also install the packages contained in the workload individually. This documentation briefly describes the installation process via the workload.

#### **Command line program TcPkg**

You can use the TcPkg Command Line Interface (CLI) to display the available workloads on the system:

```
tcpkg list TF6310
```

You can use the following command to install the workload of the TF6770 IoT Websockets function.

```
tcpkg install TF6310.TcpIp.XAE tcpkg install TF6310.TcpIp.XAR
```

#### TwinCAT Package Manager UI



You can use the **U**ser Interface (UI) to display all available workloads and install them if required. To do this, follow the corresponding instructions in the interface.

### 3.4 Installation Windows CE

This section describes, how you can install the TwinCAT 3 Function TF6310 TCP/IP on a Beckhoff Embedded PC Controller based on Windows CE.

The setup process consists of four steps:

- Download of the setup file [▶ 13]
- Installation on a host computer [▶ 13]
- Transferring the executable to the Windows CE device [▶ 13]
- <u>Software installation [▶ 14]</u>

The last paragraph of this section describes the Software upgrade [ 14].

#### Download of the setup file

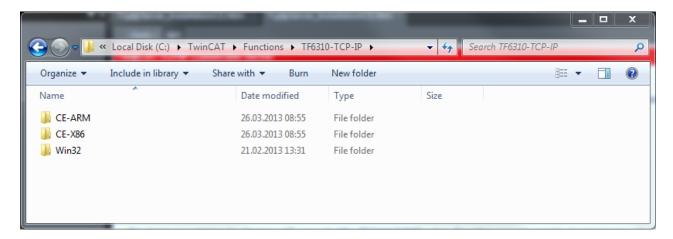
The CAB installation files for Windows CE are part of the TF6310 TCP/IP setup. Therefore you only need to download one setup file from <a href="https://www.beckhoff.com">www.beckhoff.com</a> which contains binaries for Windows XP, Windows 7 and Windows CE (x86 and ARM).

The installation procedure of the TF6310 TCP/IP setup is described in the regular installation article (see Installation [ > 9]).

#### Installation on a host computer

After installation, the install folder contains three directories - each one for a different hardware platform:

- CE-ARM: ARM-based Embedded Controllers running Windows CE, e.g. CX8090, CX9020
- CE-X86: X86-based Embedded Controllers running Windows CE, e.g. CX50xx. CX20x0
- Win32: Embedded Controllers running Windows XP, Windows 7 or Windows Embedded Standard



The CE-ARM and CE-X86 folders contain the TF6310 CAB files for Windows CE corresponding to the hardware platform of your Windows CE device. This file needs to be transferred to the Windows CE device.

#### Transferring the executable to the Windows CE device

Transfer the corresponding executable to you Windows CE device. This can be done via one of the following ways:

- · via a Shared Folder
- · via the integrated FTP-Server
- · via ActiveSync



· via a CF card

For more information, please consult the "Windows CE" section in the Beckhoff Information System.

#### Software installation

After the file has been transferred via one of the above methods, execute the file and acknowledge the following dialog with **Ok**. Restart your Windows CE device after the installation has finished.

After the restart has been completed, the executable files of TF6310 are started automatically in the background.

The software is installed in the following directory on the CE device:

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

#### **Upgrade instructions**

If you have already a version of TF6310 installed on your Windows CE device, you need to perform the following things on the Windows CE device to upgrade to a newer version:

- 1. Open the CE Explorer by clicking on **Start > Run** and entering "explorer".
- 2. Navigate to \Hard Disk\TwinCAT\Functions\TF6310-TCP-IP\Server.
- 3. Rename TcplpServer.exe to TcplpServer.old.
- 4. Restart the Windows CE device.
- 5. Transfer the new CAB-File to the CE device.
- 6. Execute the CAB-File and install the new version.
- 7. Delete TcplpServer.old.
- 8. Restart the Windows CE device.
- ⇒ After the restart is complete, the new version is active.

### 3.5 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

#### Licensing the full version of a TwinCAT 3 Function

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

#### Licensing the 7-day test version of a TwinCAT 3 Function

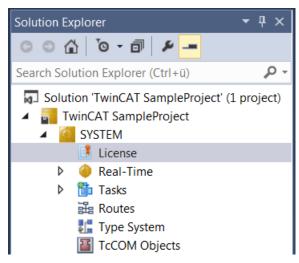


A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

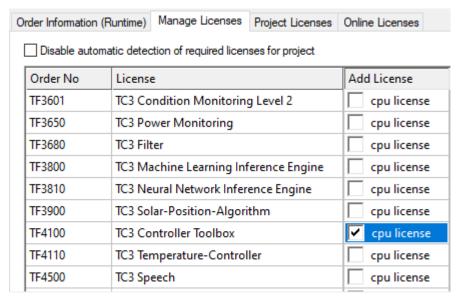
- 1. Start the TwinCAT 3 development environment (XAE).
- 2. Open an existing TwinCAT 3 project or create a new project.
- 3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
  - ⇒ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.



4. In the Solution Explorer, double-click License in the SYSTEM subtree.



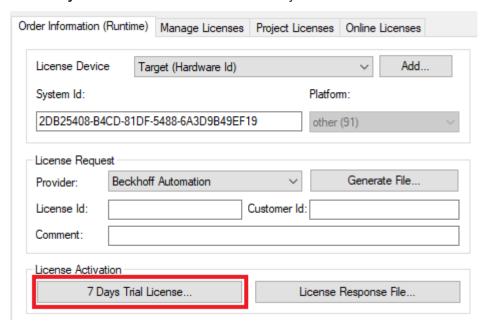
- ⇒ The TwinCAT 3 license manager opens.
- 5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



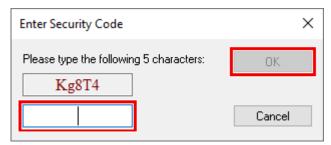
- 6. Open the Order Information (Runtime) tab.
  - ⇒ In the tabular overview of licenses, the previously selected license is displayed with the status "missing".



7. Click **7-Day Trial License...** to activate the 7-day trial license.



⇒ A dialog box opens, prompting you to enter the security code displayed in the dialog.



- 8. Enter the code exactly as it is displayed and confirm the entry.
- 9. Confirm the subsequent dialog, which indicates the successful activation.
  - ⇒ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
- 10. Restart the TwinCAT system.
- ⇒ The 7-day trial version is enabled.

### 3.6 Migration from TwinCAT 2

If you would like to migrate an existing TwinCAT 2 PLC project which uses one of the TCP/IP Server's PLC libraries, you need to perform some manual steps to ensure that the TwinCAT 3 PLC converter can process the TwinCAT 2 project file (\*.pro). In TwinCAT 2, the Function TCP/IP Server is delivered with three PLC libraries:

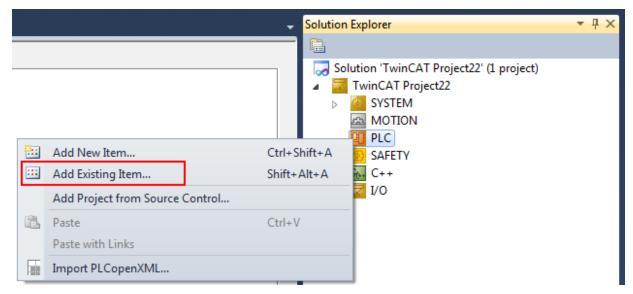
- Tcplp.lib
- · TcSocketHelper.lib
- TcSnmp.lib

By default, these library files are installed in C:\TwinCAT\Plc\Lib\. Depending on the library used in your PLC project, you need to copy the corresponding library file to C:\TwinCAT3\Components\Plc\Converter\Lib and then perform the following steps:

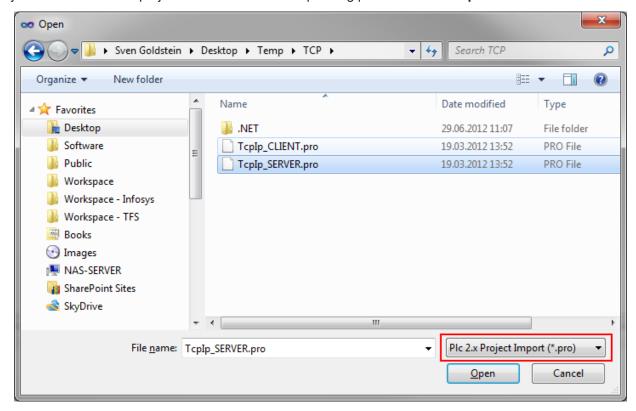
- 1. Open the TwinCAT Engineering.
- 2. Create a new TwinCAT 3 solution.



3. Right-click on the "PLC" node and select **Add Existing Item** in the context menu that opens.

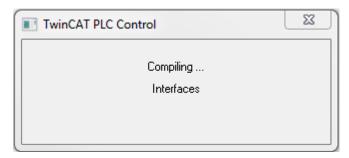


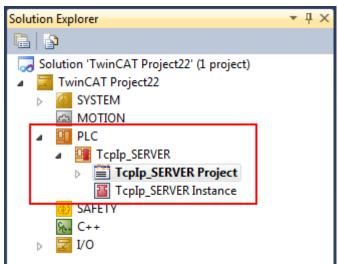
4. In the Open dialog, select the file type "Plc 2.x Project Import (\*.pro)", browse to the folder containing your TwinCAT 2 PLC project and select the corresponding.pro file and click **Open**.





⇒ TwinCAT 3 starts the converter process and finally displays the converted PLC project under the "PLC" node.







### 4 Technical introduction

This section will give a general overview about the transport protocols TCP and UDP and will also link to the corresponding PLC libraries needed to implement each protocol. Both transport protocols are part of the Internet Protocol suite and therefore an important part of our everyday communication, e.g. the Internet.

#### **Transmission Control Protocol (TCP)**

TCP is a connection-oriented transport protocol (OSI layer 4) that can be compared to a telephone connection, where participants have to establish the connection first before data can be transmitted. TCP provides a reliable and ordered delivery of a stream of bytes, therefore it is considered to be a "streamoriented transport protocol". The TCP protocol is used for network applications where a receive confirmation is required for the data sent by a client or server. The TCP protocol is well suited for sending larger data quantities and transports a data stream without a defined start and end. For the transmitter this is not a problem since he knows how many data bytes are transmitted. However, the receiver is unable to detect where a message ends within the data stream and where the next data stream starts. A read call on the receiver side only supplies the data currently in the receive buffer (this may be less or more than the data block sent by the other device). Therefore the transmitter has to specify a message structure that is known to the receiver and can be interpreted. In simple cases the message structure may consist of the data and a final control character (e.g. carriage return). The final control character indicates the end of a message. A possible message structure which is indeed often used for transferring binary data with a variable length could be defined as follows: The first data bytes contain a special control character (a so-called start delimiter) and the data length of the subsequent data. This enables the receiver to detect the start and end of the message.

#### TCP/IP client

A minimum TCP/IP client implementation within the PLC requires the following function blocks:

- An instance of the <u>FB SocketConnect</u> [▶ 21] and <u>FB SocketClose</u> [▶ 22] function blocks for
  establishing and closing the connection to the remote server (Hint: <u>FB ClientServerConnection</u> [▶ 44]
  encapsulates the functionality of both function blocks)
- An instance of the <u>FB SocketSend</u> [▶ <u>26</u>] and/or <u>FB SocketReceive</u> [▶ <u>28</u>] function block for the data exchange with the remote server

#### TCP/IP server

A minimum TCP/IP server implementation within the PLC requires the following function blocks:

- An instance of the <u>FB SocketListen [\*\* 24]</u> function block for opening the listener socket.
- An instance of the <u>FB SocketAccept</u> [▶ 25] and <u>FB SocketClose</u> [▶ 22] function blocks for establishing and closing the connection(s) to the remote clients (Hint: <u>FB ServerClientConnection</u> [▶ 46] encapsulates the functionality of all three function block)
- An instance of the <u>FB SocketSend</u> [▶ <u>26</u>] and/or <u>FB SocketReceive</u> [▶ <u>28</u>] function block for the data exchange with the remote clients
- An instance of the <u>FB\_SocketCloseAll\_[</u>) function block is required in each PLC runtime system, in which a socket is opened.

The instances of the <u>FB SocketAccept [ 25]</u> and <u>FB SocketReceive [ 28]</u> function blocks are called cyclically (polling), all others are called as required.

#### **User Datagram Protocol (UDP)**

UDP is a connection-less protocol, which means that data is sent between network devices without an explicit connection. UDP uses a simple transmission model without implicitly defining workflows for handshaking, reliability, data ordering or congestion control. However, even as this implies that UDP datagrams may arrive out of order, appear duplicated, or congest the wire, UDP is in some cases preffered to TCP, especially in realtime communication because all mentioned features (which are implemented in TCP) require processing power and therefore time. Because of its connection-less nature, the UDP protocol is well suited for sending small data quantities. UDP is a "packet-oriented/message-oriented transport protocol", i.e. the sent data block is received on the receiver side as a complete data block.



The following function blocks are required for a minimum UDP client/server implementation:

- An instance of the <u>FB SocketUdpCreate</u> [▶ 29] and <u>FB SocketClose</u> [▶ 22] function blocks for opening and closing an UDP socket (Hint: <u>FB ConnectionlessSocket</u> [▶ 49] encapsulates the functionality of both function)
- An instance of the <u>FB SocketUdpSendTo [▶ 30]</u> and/or <u>FB SocketUdpReceiveFrom [▶ 32]</u> function blocks for the data exchange with other devices;
- An instance of the <u>FB\_SocketCloseAll</u> [• <u>23]</u> function block in each PLC runtime system, in which a UDP socket is opened

The instances of the <u>FB SocketUdpReceiveFrom [▶ 32]</u> function block are called cyclically (polling), all others are called as required.

See also: <u>Samples [▶ 62]</u>



### 5 PLC API

### 5.1 Function blocks

### 5.1.1 FB\_SocketConnect

Using the function block FB\_SocketConnect, a local client can establish a new TCP/IP connection to a remote server via the TwinCAT TCP/IP Connection Server. If successful, a new socket is opened, and the associated connection handle is returned at the hSocket output. The connection handle is required by the function blocks <u>FB SocketSend [\rightarrow 26]</u> and <u>FB SocketReceive [\rightarrow 28]</u>, for example, in order to exchange data with a remote server. If a connection is no longer required, it can be closed with the function block <u>FB SocketClose [\rightarrow 22]</u>. Several clients can establish a connection with the remote server at the same time. For each new client, a new socket is opened and a new connection handle is returned. The TwinCAT TCP/IP Connection Server automatically assigns a new IP port number for each client.

#### Inputs

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

Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
sRemoteHost	T_IPv4Addr	IP address (Ipv4) of the remote server in the form of a string (e.g. '172.33.5.1'). An empty string can be entered on the local computer for a server.
nRemotePort	UDINT	IP port number of the remote server (e.g. 200).
bExecute	BOOL	The function block is activated by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

### •

#### Setting the maximum execution time of the function block

Do not set the value "tTimeout" too low, as timeout periods of > 30 s can occur in case of a network interruption. If the value is too low, command execution would be interrupted prematurely, and ADS error code 1861 (timeout elapsed) would be returned instead of the Winsocket error WSAETIMEDOUT.

#### Outputs

```
VAR_OUTPUT

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



Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrId	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hSocket	T_HSOCKET	TCP/IP connection handle [▶ 59] to the newly opened local client socket.

#### Requirements

Development environment	, , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

### 5.1.2 FB\_SocketClose



The function block FB\_SocketClose can be used to close an open TCP/IP or UDP socket.

**TCP/IP**: The listener socket is opened with the function block <u>FB SocketListen [ $\triangleright$  24]</u>, a local client socket with <u>FB SocketConnect [ $\triangleright$  21]</u> and a remote client socket with <u>FB SocketAccept [ $\triangleright$  25]</u>.

**UDP**: The UDP socket is opened with the function block <u>FB SocketUdpCreate</u> [▶ 29].

### Inputs

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

Name	Туре	Description	
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.	
hSocket	T_HSOCKET	TCP/IP: <u>Connection handle [▶ 59]</u> of the listener, remote or local clien socket to be closed.	
		UDP: Connection handle of the UDP socket.	
bExecute	BOOL	The function block is enabled by a positive edge at this input.	
tTimeout	TIME	Maximum time allowed for the execution of the function block.	

### Outputs

```
VAR_OUTPUT

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



Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [ &gt; 98].</u>

#### Requirements

Development environment	, ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.3 FB\_SocketCloseAll

		FB_SocketCloseAll			
_	sSrvNetId	T_AmsNetId	BOOL	bBusy	_
_	bExecute	BOOL	BOOL	bError	_
_	tTimeout	TIME	UDINT	nErrId	_

If TwinCAT is restarted or stopped, the TwinCAT TCP/IP Connection Server is also stopped. Any open sockets (TCP/IP and UDP connection handles) are closed automatically. The PLC program is reset after a "PLC reset", a "Rebuild all..." or a new "Download", and the information about already opened sockets (connection handles) is no longer available in the PLC. Any open connections can then no longer be closed properly.

The function block FB\_SocketCloseAll can be used to close all connection handles (TCP/IP and UDP sockets) that were opened by a PLC runtime system. This means that, if FB\_SocketCloseAll is called in one of the tasks of the first runtime systems (port 801), all sockets that were opened in the first runtime system are closed. In each PLC runtime system that uses the socket function blocks, an instance of FB\_SocketCloseAll should be called during the PLC start.

#### Inputs

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

Name	Туре	Description
sSrvNetId		String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

### Outputs

```
VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrid : UDINT;
END VAR
```



Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the TwinCAT TCP/
		IP Connection Server error number [▶ 98].

#### Sample of an implementation in ST

The following program code is used to properly close the connection handles (sockets) that were open before a "PLC reset" or "Download" before a PLC restart.

```
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
```

#### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (cate- gory group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

### 5.1.4 FB\_SocketListen

```
FB_SocketListen

sSrvNetId T_AmsNetId BOOL bBusy
sLocalHost T_IPv4Addr BOOL bError
nLocalPort UDINT UDINT NETID
bExecute BOOL T_HSOCKET hListener
tTimeout TIME
```

Using the function block FB\_SocketListen, a new listener socket can be opened via the TwinCAT TCP/IP Connection Server. Via a listener socket, the TwinCAT TCP/IP Connection Server can 'listen' for incoming connection requests from remote clients. If successful, the associated connection handle is returned at the hListner output. This handle is required by the function block <u>FB SocketAccept [\*25]</u>. If a listener socket is no longer required, it can be closed with the function block <u>FB SocketClose [\*22]</u>. The listener sockets on an individual computer must have unique IP port numbers.

#### Inputs

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



Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
sLocalHost	T_IPv4Addr	Local server IP address (Ipv4) in the form of a string (e.g. '172.13.15.2'). For a server on the local computer (default), an empty string may be entered.
nLocalPort	UDINT	Local server IP port (e.g. 200).
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Outputs

VAR\_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
hListener : T\_HSOCKET;

END VAF

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hListener	T_HSOCKE T	Connection handle [▶ 59] to the new listener socket.

#### Requirements

Development environment	Target system type	PLC libraries to include (cate- gory group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

### 5.1.5 FB\_SocketAccept

	FB_SocketAccept		
_	sSrvNetId	T_AmsNetId	BOOL bAccepted
_	hListener	T_H50CKET	BOOL bBusy
_	bExecute	BOOL	BOOL bError
_	tTimeout	TIME	UDINT nErrId
			T_HSOCKET hSocket

The remote client connection requests arriving at the TwinCAT TCP/IP Connection Server have to be acknowledged (accepted). The function block FB\_SocketAccept accepts the incoming remote client connection requests, opens a new remote client socket and returns the associated connection handle. The connection handle is required by the function blocks <u>FB\_SocketSend\_[\rightarrow\_26]</u> and <u>FB\_SocketReceive\_[\rightarrow\_28]</u>, for example, in order to exchange data with a remote client. All incoming connection requests first have to be accepted. If a connection is no longer required or undesirable, it can be closed with the function block <u>FB\_SocketClose\_[\rightarrow\_22]</u>.

A server implementation requires at least one instance of this function block. This instance has to be called cyclically (polling) from a PLC task. The function block can be activated via a positive edge at the bExecute input (e.g. every 5 seconds).

If successful, the bAccepted output is set, and the connection handle to the new remote client is returned at the hSocket output. No error is returned if there are no new remote client connection requests. Several remote clients can establish a connection with the server at the same time. The connection handles of



several remote clients can be retrieved sequentially via several function block calls. Each connection handle for a remote client can only be retrieved once. It is recommended to keep the connection handles in a list (array). New connections are added to the list, and closed connections must be removed from the list.

#### Inputs

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

Name	Туре	Description
sSrvNetId		String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hListener		Connection handle [▶ 59] of the listener socket. This handle must first be requested via the function block FB SocketListen [▶ 24].
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Outputs

```
VAR_OUTPUT

bAccepted: BOOL;

bBusy: BOOL;

bError: BOOL;

nErrId: UDINT;

hSocket: T_HSOCKET;

END VAR
```

Name	Туре	Description
bAccepted	BOOL	This output is set if a new connection to a remote client was established.
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrId	UDINT	If an bError output is set, this parameter returns the <a href="TwinCAT TCP/">TwinCAT TCP/</a> <a href="TwinCAT TCP/">IP Connection Server error number [ &gt; 98]</a> .
hSocket	T_HSOCKE T	Connection handle [▶ 59] of a new remote client.

#### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

### 5.1.6 FB\_SocketSend



Using the function block FB\_SocketSend, data can be sent to a remote client or remote server via the TwinCAT TCP/IP Connection Server. A remote client connection will first have to be established via the function block FB\_SocketAccept [ > 25], or a remote server connection via the function block FB\_SocketConnect [ > 21].

#### Inputs

```
VAR_INPUT

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

Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKET	<u>Connection handle [▶ 59]</u> of the communication partner to which data are to be sent.
cbLen	UDINT	Number of date to be sent in bytes.
pSrc	POINTER TO BYT	Address (pointer) of the send buffer.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

### •

#### Setting the execution time of the function block



If the transmit buffer of the socket is full, for example because the remote communication partner receives the transmitted data not quickly enough or large quantities of data are transmitted, the FB\_SocketSend function block will return ADS timeout error 1861 after the tTimeout time. In this case, the value of the tTimeout input variable has to be increased accordingly.

#### Outputs

```
VAR_OUTPUT

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

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <a href="TwinCAT TCP/">TwinCAT TCP/</a> <a href="TwinCAT TCP/">IP Connection Server error number [ 98]</a> .

#### Requirements

Development environment	, , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)



### 5.1.7 FB\_SocketReceive

```
FB_SocketReceive

sSrvNetId T_AmsNetId BOOL bBusy
hSocket T_HSOCKET BOOL bError
cbLen UDINT UDINT nErrId
pDest POINTER TO BYTE UDINT nRecBytes
bExecute BOOL
tTimeout TIME
```

Using the function block FB\_SocketReceive, data from a remote client or remote server can be received via the TwinCAT TCP/IP Connection Server. A remote client connection will first have to be established via the function block FB\_SocketAccept [\rightarrow 25], and a remote server connection via the function block FB\_SocketConnect [\rightarrow 21]. The data can be received or sent in fragmented form (i.e. in several packets) within a TCP/IP network. It is therefore possible that not all data may be received with a single call of the FB\_SocketReceive instance. For this reason, the instance has to be called cyclically (polling) within the PLC task, until all required data have been received. During this process, an rising edge is generated at the

bExecute input, e.g. every 100 ms. If successful, the data received last are copied into the receive buffer. The nRecBytes output returns the number of the last successfully received data bytes. If no new data could be read during the last call, the function block returns no error and nRecBytes == zero.

In a simple protocol for receiving, for example, a null-terminated string on a remote server, the function block FB\_SocketReceive, for example, will have to be called repeatedly until the null termination was detected in the data received.

## •

#### Set timeout value

If the remote device was disconnected from the TCP/IP network (on the remote side only) while the local device is still connected to the TCP/IP network, the function block FB\_SocketReceive returns no error and no data. The open socket still exists, but no data are received. The application may wait forever for data in this case. It is recommended to implement timeout monitoring in the PLC application. If not all data were received after a certain period, e.g. 10 seconds, the connection has to be closed and reinitialized.

#### Inputs

```
VAR_INPUT

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

Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKET	Connection handle [▶ 59] of the communication partner from which data are to be received.
cbLen	UDINT	Maximum available buffer size (in bytes) for the data to be read.
pDest	POINTER TO BY TE	Address (pointer) of the receive buffer.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Outputs

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



```
nErrid : UDINT;
nRecBytes : UDINT;
END VAR
```

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the TwinCAT TCP/IP Connection Server error number.
nRecBytes	UDINT	Number of the last successfully received data bytes.

#### Requirements

Development environment	, , ,,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.8 FB\_SocketUdpCreate

	FB_Socket	UdpCreate
	sSrvNetId T_AmsNetId	BOOL bBusy
_	sLocalHost <i>T_IPv4Addr</i>	BOOL bError
_	nLocalPort UDINT	UDINT nErrId
_	bExecute BOOL	T_HSOCKET hSocket
_	tTimeout TIME	

The function block FB\_SocketUdpCreate can be used to open a client/server socket for the User Datagram Protocol (UDP). If successful, a new socket is opened, and the associated socket handle is returned at the hSocket output. The handle is required by the function blocks FB SocketUdpSendTo [ > 30] and FB SocketUdpReceiveFrom [ > 32], for example, in order to exchange data with a remote device. If a UDP socket is no longer required, it can be closed with the function block FB SocketClose [ > 22]. The port address nLocalHost is internally reserved by the TCP/IP Connection Server for the UDP protocol (a "bind" is carried out). Several network adapters may exist in a PC. The input parameter sLocalHost determines the network adapter to be used. If the sLocalHost input variable is ignored (empty string), the TCP/IP Connection Server uses the default network adapter. This is usually the first network adapter from the list of the network adapters in the Control Panel.

### Automatically created network connections

If an empty string was specified for sLocalHost when FB\_SocketUdpCreate was called and the PC was disconnected from the network, the system will open a new socket under the software loopback IP address: '127.0.0.1'.

### Automatically created network connections with several network adapters

If two or more network adapters are installed in the PC and an empty string was specified as sLocalHost, and the default network adapter was then disconnected from the network, the new socket will be opened under the IP address of the second network adapter.

### Setting a network address

In order to prevent the sockets from being opened under a different IP address, you can specify the sLocalHost address explicitly or check the returned address in the handle variable (hSocket), close the socket and re-open it.

### Inputs

```
VAR_INPUT
sSrvNetId : T_AmsNetId := '';
sLocalHost : T_IPv4Addr := '';
nLocalPort : UDINT;
```



```
bExecute : BOOL;
tTimeout : TIME:= T#5s;
END_VAR
```

Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
sLocalHost	T_IPv4Addr	Local IP address (Ipv4) of the UDP client/server socket as a string (e.g. '172.33.5.1'). An empty string may be specified for the default network adapter.
nLocalPort	UDINT	Local IP port number of the UDP client/server socket (e.g. 200).
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Outputs

```
VAR_OUTPUT

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

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hSocket	T_HSOCKET	Handle of the newly opened UDP client/server socket [▶ 59].

#### Requirements

Development environment	, , ,,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.9 FB\_SocketUdpSendTo

The function block FB\_SocketUdpSendTo can be used to send UDP data to a remote device via the TwinCAT TCP/IP Connection Server. The UDP socket must first be opened with the function block FB\_SocketUdpCreate [\rightarrow 29].

### Inputs

```
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	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKE T	Handle of an opened UDP socket [▶ 59].
sRemoteHost	T_IPv4Addr	IP address (Ipv4) in string form (e.g. '172.33.5.1') of the remote device to which data is to be sent. An empty string can be entered on the local computer for a device.
nRemotePort	UDINT	IP port number (e.g. 200) of the remote device to which data is to be sent.
cbLen	UDINT	Number of date to be sent in bytes. The maximum number of data bytes to be sent is limited to 8192 bytes (constant TCPADS_MAXUDP_BUFFSIZE in the library in order to save storage space).
pSrc	POINTER TO BYTE	Address (pointer) of the send buffer.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Setting the size of the received data bytes



Available in product version: TwinCAT TCP/IP Connection Server v1.0.50 or higher: The maximum number of data bytes to be received can be increased (only if absolutely necessary).

#### TwinCAT 2

1. Redefine global constant in the PLC project (in the sample the maximum number of data bytes to be received is to be increased to 32000):

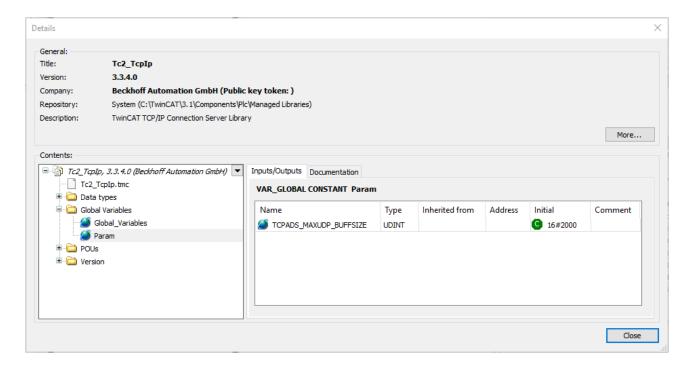
```
VAR_GLOBAL CONSTANT
        TCPADS_MAXUDP_BUFFSIZE : UDINT := 32000;
END VAR
```

- 2. Activate option **Replace constants** in the dialog of the TwinCAT PLC control (Project > Options ... > Build).
- 3. Rebuild Project.

#### TwinCAT 3

In TwinCAT 3, this value can be edited via a parameter list of the PLC library (from version 3.3.4.0).





#### Outputs

```
VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
END VAR
```

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> IP Connection Server error number [▶ 98].

#### Requirements

Development environment	, , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.10 FB\_SocketUdpReceiveFrom



Using the function block FB\_SocketUdpReceiveFrom, data from an open UDP socket can be received via the TwinCAT TCP/IP Connection Server. The UDP socket must first be opened with the function block FB\_SocketUdpCreate [▶ 29]. The instance of the FB\_SocketUdpReceive function block has to be called cyclically (polling) within the PLC task. During this process, an rising edge is generated at the bExecute



input, e.g. every 100 ms. If successful, the data received last are copied into the receive buffer. The nRecBytes output returns the number of the last successfully received data bytes. If no new data could be read during the last call, the function block returns no error and nRecBytes == zero.

#### Inputs

```
VAR_INPUT

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

Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKE T	Handle of an opened UDP socket [▶ 59], whose data are to be received.
cbLen	UDINT	Maximum available buffer size (in bytes) for the data to be read. The maximum number of data bytes to be received is limited to 8192 bytes (constant TCPADS_MAXUDP_BUFFSIZE in the library in order to save storage space).
pDest	POINTER TO BYTE	Address (pointer) of the receive buffer.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

### •

#### Setting the size of the received data bytes



Available in product version: TwinCAT TCP/IP Connection Server v1.0.50 or higher: The maximum number of data bytes to be received can be increased (only if absolutely necessary).

#### TwinCAT 2

1. Redefine global constant in the PLC project (in the sample the maximum number of data bytes to be received is to be increased to 32000):

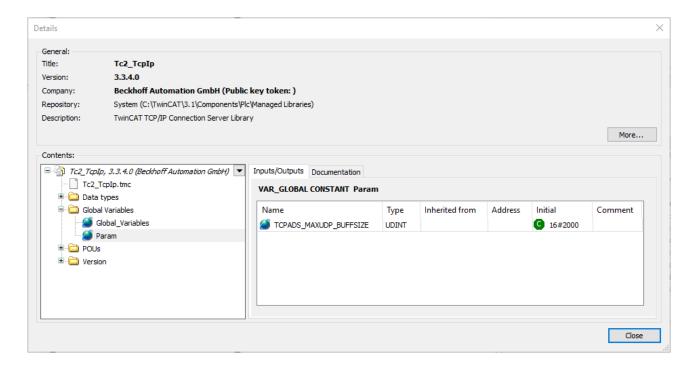
```
VAR_GLOBAL CONSTANT
        TCPADS_MAXUDP_BUFFSIZE : UDINT := 32000;
END VAR
```

- 2. Activate option **Replace constants** in the dialog of the TwinCAT PLC control (Project > Options ... > Build).
- 3. Rebuild Project.

#### **TwinCAT 3**

In TwinCAT 3, this value can be edited via a parameter list of the PLC library (from version 3.3.4.0).





### Outputs

```
VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
sRemoteHost : T_IPv4Addr := '';
nRemotePort : UDINT;
nRecBytes : UDINT;
END VAR
```

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <a href="IwinCAT TCP/">IwinCAT TCP/</a> <a href="IwinCAT TCP/">IP Connection Server error number [ &gt; 98]</a> .
sRemoteHos t	T_IPv4Addr	If successful, IP address (Ipv4) of the remote device whose data were received.
nRemotePort	UDINT	If successful, IP port number of the remote device whose data were received (e.g. 200).
nRecBytes	UDINT	Number of data bytes last successfully received.

#### Requirements

Development environment	Target system type	PLC libraries to include (cate- gory group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2 Tcplp (communication)

### 5.1.11 FB\_SocketUdpAddMulticastAddress





Binds the server to a multicast IP address so that multicast packages can be received. This function block expects an already established UDP socket connection, which can be established via the function block FB\_SocketUdpCreate [ > 29].

#### Inputs

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

Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKE T	Connection handle [▶ 59] of the listener socket. This handle must first be requested via the function block FB SocketUdpCreate [▶ 29].
sMulticastAddr	T_IPv4Addr	Multicast IP address to which the binding should take place.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Outputs

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

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError		If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrId	UDINT	If an bError output is set, this parameter returns the TwinCAT TCP/
		IP Connection Server error number [▶ 98].

#### Requirements

Development environment	<b>.</b> , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.12 FB\_SocketUdpDropMulticastAddress



Removes the binding to a multicast IP address that was previously set up via the function block <u>FB SocketUdpAddMulticastAddress [\rightarrow 34]</u>.



#### Inputs

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

Name	Туре	Description
sSrvNetId	T_AmsNetI d	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCK ET	<u>Connection handle [▶ 59]</u> of the listener socket. This handle must first be requested via the function block <u>FB SocketUdpCreate [▶ 29]</u> .
sMulticastAddr	T_IPv4Add r	Multicast IP address to which the binding should take place.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

#### Outputs

```
VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
END VAR
```

Name	Туре	Description	
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.	
bError		If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.	
nErrld		If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .	

#### Requirements

Development environment	Target system type	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

### 5.1.13 FB\_TIsSocketConnect

```
FB_TIsSocketConnect

hSocket T_HSOCKET

sSrvNetId T_AmsNetId

sRemoteHost STRING(TCPADS_TLS_HOSTNAME_SIZE)

nRemotePort UDINT

flags ST_TIsConnectFlags

bExecute BOOL

tTimeout TIME
```

The FB\_TIsSocketConnect function block enables a client to establish a new TCP/IP connection to a remote server via the TwinCAT TCP/IP Connection Server, secured via TLS. If successful, a new socket is opened, and the associated connection handle is returned at the hSocket output. The connection handle is required by the function blocks <u>FB SocketSend [\*\* 26]</u> and <u>FB SocketReceive [\*\* 28]</u>, for example, in order to exchange data with a remote server. If a connection is no longer required, it can be closed with the function block <u>FB SocketClose [\*\* 22]</u>. Several clients can establish a connection with the remote server at the same time.



For each new client, a new socket is opened and a new connection handle is returned. The TwinCAT TCP/IP Connection Server automatically assigns a new IP port number for each client. The TLS parameters can be defined via the function blocks FB\_TlsSocketAddCa [ \( \Delta 40 \)], FB\_TlsSocketAddCrl [ \( \Delta 41 \)],

<u>FB TlsSocketSetPsk</u> [▶ 43] and <u>FB TlsSocketSetCert</u> [▶ 42]. Programming samples for their use can be found in our samples.

# Inputs

```
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	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
sRemoteHost	STRING(TCPADS_TLS_HOS TNAME_SIZE)	IP address (Ipv4) of the remote server in the form of a string (e.g. 172.33.5.1). An empty string can be entered on the local computer for a server.
nRemotePort	UDINT	IP port number of the remote server (e.g. 200).
flags	ST TIsConnectFlags [▶ 58]	Additional (optional) client connection parameters.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

# •

### Setting the maximum execution time of the function block



Do not set the value "tTimeout" too low, as timeout periods of > 30 s can occur in case of a network interruption. If the value is too low, command execution would be interrupted prematurely, and ADS error code 1861 (timeout elapsed) would be returned instead of the Winsocket error WSAETIMEDOUT.

# Inputs/outputs

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

Name	Туре	Description
hSocket		TCP/IP connection handle [▶ 59] to the newly opened local client socket

# Outputs

```
VAR_OUTPUT

bBusy : BOOL;

bError : BOOL;

nErrid : UDINT;

END_VAR
```



Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError		If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the TwinCAT TCP/
		IP Connection Server error number [▶ 98].

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.1.14 FB\_TIsSocketListen

```
FB_TIsSocketListen

hListener T_HSOCKET BOOL bBusy
—sSrvNetId T_AmsNetId BOOL bError
—sLocalHost T_IPv4Addr UDINT nErrId
—nLocalPort UDINT
—flags ST_TIsListenFlags
—bExecute BOOL
—tTimeout TIME
```

The function block FB\_TLsSocketListen can be used to open a new listener socket secured via TLS via the TwinCAT TCP/IP Connection Server. Via a listener socket, the TwinCAT TCP/IP Connection Server can 'listen' for incoming connection requests from remote clients. The socket handle created with the function block FB\_TlsSocketCreate [\rightarrow 39] can then be used by the function block FB\_SocketAccept [\rightarrow 25] to accept an incoming client request. If a listener socket is no longer required, it can be closed with the function block FB\_SocketClose [\rightarrow 22]. The listener sockets on an individual computer must have unique IP port numbers. Programming samples for using this function block can be found in our samples.

# Inputs

```
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	Туре	Description
hListener	T_HSOCKET	Socket handle, which was created via the function block FB_TIsSocketCreate.
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
sLocalHost	T_IPv4Addr	Local server IP address (Ipv4) in the form of a string (e.g. 172.13.15.2). For a server on the local computer (default), an empty string may be entered.
nLocalPort	UDINT	Local server IP port (e.g. 200).
flags	ST TlsListenFlags [▶ 58]	Additional (optional) server connection settings.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.



# ✓ Inputs/outputs

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

Name	Туре	Description
hListener	T_HSOCKET	Connection handle [▶ 59] to the new listener socket.

# Outputs

```
VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrld : UDINT;
END_VAR
```

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_Tcplp (Communication)
TwinCAT v3.1.0		

# 5.1.15 FB\_TIsSocketCreate

The function block FB\_TlsSocketCreate can be used to create a new socket via the TwinCAT TCP/IP Connection Server, either for a server (bListener:=true) or client application (bListener:=false). Via a listener socket, the TwinCAT TCP/IP Connection Server can 'listen' for incoming connection requests from remote clients. If successful, the associated connection handle (hSocket) is returned at the hListner output. This handle is required by the function block FB\_TlsSocketListen [\rightarrow 38], and subsequently FB\_SocketAccept [\rightarrow 25]. If a listener socket is no longer required, it can be closed with the function block FB\_SocketClose [\rightarrow 22]. After the execution of the function block FB\_TlsSocketCreate TLS parameters can be set to secure the communication connection. This is done using the function blocks FB\_TlsSocketAddCa [\rightarrow 40], FB\_TlsSocketAddCrl [\rightarrow 41], FB\_TlsSocketSetCert [\rightarrow 42] and FB\_TlsSocketSetPsk [\rightarrow 43]. Programming samples for this can be found in our samples.

# Inputs

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



Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
bListener	BOOL	Creates a new socket handle.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

```
VAR_OUTPUT

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

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hSocket	T_HSOCKET	Connection handle [▶ 59] for the new socket.

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.1.16 FB\_TIsSocketAddCa

The FB\_TIsSocketAddCa function block is used to configure the path to a CA certificate for an existing socket handle. The certificate file must be in PEM format. Programming samples for using this function block can be found in our samples.

### Inputs

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



Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKET	Socket handle.
sCaPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	Path to the CA's certificate file.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

```
VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nerrid : UDINT;
END VAR
```

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <a href="TwinCAT TCP/">TwinCAT TCP/</a> <a href="TwinCAT TCP/">TCP/</a>

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.1.17 FB\_TIsSocketAddCrI

The function block FB\_TIsSocketAddCrl is used to specify the path to a CRL file for an existing socket handle. The CRL must be in PEM format. Programming samples for using this function block can be found in our samples.

# Inputs

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



Name	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/ IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKET	Socket handle.
sCrlPath	STRING(TCPADS_TLS_CER TIFICATE_PATH_SIZE)	Path to the CRL file.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

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

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the TwinCAT TCP/
		IP Connection Server error number [▶ 98].

#### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.1.18 FB\_TIsSocketSetCert

The function block FB\_TIsSocketSetCert can be used to configure a client/server certificate that is to be used for a specific socket handle. The certificates must be in PEM format. Programming samples for using this function block can be found in our samples.

### Inputs

```
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	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKET	Socket handle.
sCertPath	STRING(TCPADS_TLS_CERTIFI CATE_PATH_SIZE)	Path to the file with the client/server certificate.
sKeyPath	STRING(TCPADS_TLS_CERTIFI CATE_PATH_SIZE)	Path to the file with the client/server private key.
sKeyPwd	STRING(TCPADS_TLS_KEY_PA SSWORD_SIZE)	Optional, if the private key is secured with a password.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

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

Name	Туре	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrId	UDINT	If an bError output is set, this parameter returns the TwinCAT TCP/
		IP Connection Server error number [▶ 98].

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.1.19 FB\_TIsSocketSetPsk

```
### FB_TIsSocketSetPsk

### sSrvNetId T_AmsNetId

### hSocket T_HSOCKET

### sIdentity STRING(TCPADS_TLS_PSK_IDENTITY_SIZE)

### pskKey PVOID

### pskKey PVOID

### pskKeyLen UDINT (0..TCPADS_TLS_MAX_PSK_KEY_SIZE)

### bExecute BOOL

### tTimeout TIME
```

The function block FB\_TIsSocketSetPsk can be used to configure a pre-shared secret for an existing socket handle. Programming samples for using this function block can be found in our samples.

# Inputs

```
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	Туре	Description
sSrvNetId	T_AmsNetId	String containing the network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
hSocket	T_HSOCKET	Socket handle.
sldentity	STRING(TCPADS_TLS_PSK_IDENTI TY_SIZE)	A freely selectable identity for the PSK.
pskKey	PVOID	Pointer to a byte array containing the PSK.
pskKeyLen	UDINT(0TCPADS_TLS_MAX_PSK_K EY_SIZE)	Length of pskKey.
bExecute	BOOL	The function block is enabled by a positive edge at this input.
tTimeout	TIME	Maximum time allowed for the execution of the function block.

# Outputs

VAR OUTPUT

bBusy : BOOL; bError : BOOL; nErrId : UDINT;

END\_VAR

Name	Type	Description
bBusy	BOOL	This output is active if the function block is activated. It remains active until acknowledgement.
bError	BOOL	If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.
nErrld	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98].</u>

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# **5.1.20** Helper

# 5.1.20.1 FB\_ClientServerConnection

```
FB_ClientServerConnection

sSrvNetID T_AmsNetID BOOL bBusy

nMode DWORD BOOL bError

sRemoteHost T_IPv4Addr UDINT nErrId

nRemotePort UDINT T_HSOCKET hSocket

bEnable BOOL E_SocketConnectionState eState

tReconnect TIME
```

The function block FB\_ClientServerConnection can be used to manage (establish or remove) a client connection. FB\_ClientServerConnection simplifies the implementation of a client application by encapsulating the functionality of the two function blocks FB\_SocketConnect[\rightarrow\_21] and FB\_SocketClose[\rightarrow\_22]



internally. The integrated debugging output of the connection status facilitates troubleshooting in the event of configuration or communication errors. In addition, a minimum client application only requires an instance of the function block FB\_SocketSend [ > 26] and/or an instance of the function block FB\_SocketReceive [ > 28].

In the first step, a typical client application establishes the connection with the server via the FB\_ClientServerConnection function block. In the next step instances of FB\_SocketSend and/or FB\_SocketReceive can be used to exchange data with the server. When a connection is closed depends on the requirements of the application.

### Inputs

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

Name	Туре	Description
sSrvNetID	T_AmsNetI D	String containing the AMS network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
nMode	DWORD	Parameter flags (modes). The permissible parameters are listed here and can be combined by ORing:
		CONNECT_MODE_ENABLEDBG:
		Enables logging of debug messages in the application log. In order to view the debug messages open the TwinCAT System Manager and activate log view.
sRemoteHost	T_IPv4Add r	IP address (Ipv4) of the remote server in the form of a string (e.g. '172.33.5.1'). An empty string can be entered on the local computer for a server.
nRemotePort	UDINT	IP port number of the remote server (e.g. 200).
bEnable	BOOL	As long as this input is TRUE, the system attempts to establish a new connection at regular intervals until a connection was established successfully. Once established, a connection can be closed again with FALSE.
tReconnect	TIME	Cycle time used by the function block to try and establish the connection.

# •

### Setting the cycle time for the connection



The tReconnect value should not be set too low, since timeout periods of > 30 s may occur in the event of a network interruption. If the value is too low, command execution would be interrupted prematurely, and ADS error code 1861 (timeout elapsed) would be returned instead of the Winsocket error WSAETIMEDOUT.

### Outputs

```
VAR_OUTPUT

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

END VAR
```



Name	Туре	Description
bBusy	BOOL	TRUE, as long as the function block is active.
bError	BOOL	Becomes TRUE if an error code occurs.
nErrID	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hSocket	T_HSOCKET	Connection handle [▶ 59] to the newly opened local client socket. If successful, this variable is transferred to the instances of the function blocks FB SocketSend [▶ 26] and/or FB SocketReceive [▶ 28].
eState	E_SocketConnectionSt ate	Returns the current connection status [ > 55].

### Sample of a call in FBD

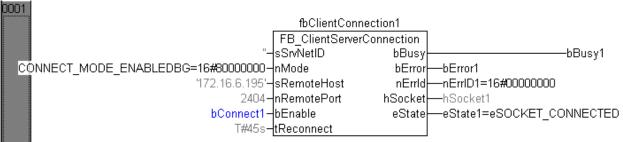
```
PROGRAM MAIN

VAR

fbClientConnection1 : FB_ClientServerConnection;
bConnect1 : B00L;
bBusy1 : B00L;
bError1 : B00L;
nErrID1 : UDINT;
hSocket1 : T_HSOCKET;
eState1 : E_SocketConnectionState;

END_VAR

fbClientConnection1
```



Here you can find more application examples (and source code): <u>Samples [▶ 62]</u>

### Requirements

Development environment	, , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.20.2 FB\_ServerClientConnection

```
FB_ServerClientConnection

hServer T_HSERVER BOOL bBusy

eMode E_SocketAcceptMode BOOL bError

sRemoteHost T_IPv4Addr UDINT nErrID

nRemotePort UDINT T_HSOCKET hSocket

bEnable BOOL E_SocketConnectionState eState

tReconnect TIME
```

The function block FB\_ServerClientConnection can be used to manage (establish or remove) a server connection. FB\_ServerClientConnection simplifies the implementation of a server application by encapsulating the functionality of the three function blocks <u>FB\_SocketListen [\rightarrow 24]</u>, <u>FB\_SocketAccept [\rightarrow 25]</u> and <u>FB\_SocketClose [\rightarrow 22]</u> internally. The integrated debugging output of the connection status facilitates troubleshooting in the event of configuration or communication errors. In addition, a minimum server application only requires an instance of the function block <u>FB\_SocketSend [\rightarrow 26]</u> and/or an instance of the function block <u>FB\_SocketReceive [\rightarrow 28]</u>.



In the first step a typical server application establishes the connection with the client via the FB\_ServerClientConnection function block (more precisely, the server application accepts the incoming connection request). In the next step instances of FB\_SocketSend and/or FB\_SocketReceive can be used to exchange data with the server. When a connection is closed depends on the requirements of the application.

### Inputs

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

Name	Туре	Description
eMode	E_SocketAcceptM ode	Defines whether all or only certain <u>connections [▶ 54]</u> are to be accepted.
sRemot eHost	T_IPv4Addr	IP address (Ipv4) in string form (e.g. '172.33.5.1') of the remote client whose connection is to be accepted. For a client on the local computer an empty string may be specified.
nRemot ePort	UDINT	IP port number (e.g. 200) of the remote client whose connection is to be accepted.
bEnable	BOOL	As long as this input is TRUE, the system attempts to establish a new connection at regular intervals until a connection was established successfully. Once established, a connection can be closed again with FALSE.
tReconn ect	TIME	Cycle time used by the function block to try to establish a connection.

# Inputs/outputs

```
VAR_IN_OUT
hserver : T_Hserver;
end var
```

Name	Туре	Description
hServer	hServer	Server handle [▶ 59]. This input variable has to be initialized via the
		F CreateServerHnd [▶ 51] function.

# Outputs

```
VAR_OUTPUT

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

Name	Туре	Description
bBusy	BOOL	TRUE, as long as the function block is active.
bError	BOOL	Becomes TRUE if an error code occurs.
nErrId	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hSocket		<u>Connection handle [▶ 59]</u> to the newly opened remote client socket. If successful, this variable is transferred to the instances of the function blocks <u>FB SocketSend [▶ 26]</u> and/or <u>FB SocketReceive [▶ 28]</u> .
eState	E_SocketConnectionSt ate	Returns the current <u>connection status [▶ 55]</u> .



### Sample in FBD

The following sample illustrates initialization of a server handle variable. The server handle is then transferred to three instances of the FB\_ServerClientConnection function block.

```
VAR
                           : T_HSERVER;
    hServer
    bListen
                           : BOOL;
    fbServerConnection1 : FB ServerClientConnection;
    bConnect1 : BOOL;
bBusy1 : BOOL;
bError1 : BOOL;
nErrID1 : UDINT;
hSocket1 : T_HSOCKET;
                  : T_HSUCRE1,
: E_SocketConnectionState;
    eState1
    fbServerConnection2 : FB ServerClientConnection;
    bConnect2 : BOOL;
bBusy2 : BOOL;
                           : BOOL;
    bError2
    nErrID2 : UDINT;
hSocket2 : T_HSOCKET;
eState2 : E_SocketConnectionState;
    fbServerConnection3 : FB ServerClientConnection;
    bConnect3 : BOOL;
bBusy3 : BOOL;
    DETTOT3
nETTID3
hSocket3
eState3
                           : BOOL;
    bError3
                            : UDINT;
                           : T_HSOCKET;
                     : E_SocketConnectionState;
    eState3
END_VAR
```

#### Online View:





The first connection is activated (bConnect1 = TRUE), but the connection has not yet been established (passive open).

The second connection has not yet been activated (bConnect2 = FALSE) (closed).

The third connection has been activated (bConnect3 = TRUE) and a connection to the remote client has been established.

Here you can find more application examples (and source code): <u>Samples [> 62]</u>

#### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

### 5.1.20.3 FB\_ConnectionlessSocket





A UDP socket can be managed (opened/generated and closed) with the function block FB\_ConnectionlessSocket. FB\_ConnectionlessSocket simplifies the implementation of a UDP application by encapsulating the functionality of the two function blocks <u>FB\_SocketUdpCreate [\rightarrow 29]</u> and <u>FB\_SocketClose [\rightarrow 22]</u> already internally. The integrated debugging output of the socket status facilitates troubleshooting in the event of configuration or communication errors. In addition, a minimum UDP application only requires an instance of the function block <u>SocketUdpSendTod [\rightarrow 30]</u> and/or an instance of the function block <u>FB\_SocketUdpReceiveFrom [\rightarrow 32]</u>.

In the first step a typical UDP application opens a connection-less UDP socket with the function block FB\_ConnectionlessSocket. In the next step instances of FB\_SocketUdpSendTo and/or FB\_SocketUdpReceiveFrom can be used for exchanging data with another communication device. When a UDP socket is closed depends on the requirements of the application (e.g. in the event of a communication error).

# Inputs

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

Name	Туре	Description
sSrvNetID	T_AmsNetI D	String containing the AMS network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
nMode	DWORD	Parameter flags (modes). The permissible parameters are listed here and can be combined by ORing.
		CONNECT_MODE_ENABLEDBG:
		Enables logging of debug messages in the application log. In order to view the debug messages open the TwinCAT System Manager and activate log view.
sLocalHost	T_lpv4Add r	IP address (Ipv4) in string form (e.g. '172.33.5.1') of the local network adapter. An empty string may be specified for the default network adapter.
nLocalPort	UDINT	IP port number (e.g. 200) on the local computer.
bEnable	BOOL	As long as this input is TRUE, attempts are made cyclically to open a UDP socket until a connection has been established. An open UDP socket can be closed again with FALSE.
tReconnect	TIME	Cycle time with which the function block tries to open the UDP socket.

### Setting the cycle time for the connection



The tReconnect value should not be set too low, since timeout periods of > 30 s may occur in the event of a network interruption. If the value is too low, command execution would be interrupted prematurely, and ADS error code 1861 (timeout elapsed) would be returned instead of the Winsocket error WSAETIMEDOUT.

# Outputs

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



Name	Туре	Description
bBusy	BOOL	TRUE, as long as the function block is active.
bError	BOOL	Becomes TRUE if an error code occurs.
nErrID	UDINT	If an bError output is set, this parameter returns the <u>TwinCAT TCP/</u> <u>IP Connection Server error number [▶ 98]</u> .
hSocket	T_HSOCKET	<u>Connection handle [▶ 59]</u> to the newly opened UDP socket. If successful, this variable is transferred to the instances of the function blocks <u>FB SocketUdpSendTo [▶ 30]</u> and/or <u>FB SocketUdpReceiveFrom [▶ 32]</u> .
eState	E_SocketConnectionlessS tate	Returns the current <u>connection status [▶ 55]</u> .

### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.2 Functions

# 5.2.1 F\_CreateServerHnd

```
F_CreateServerHnd

— sSrvNetID T_AmsNetID BOOL F_CreateServerHnd
— sLocalHost T_IPv4Addr
— nLocalPort UDINT
— nMode DWORD
— bEnable BOOL
— hServer T_HSERVER
```

The function F\_CreateServerHnd is used to initialize/set the internal parameters of a server handle variable hServer. The server handle is then transferred to the instances of the function block

<u>FB ServerClientConnection</u> [▶ 46] via VAR\_IN\_OUT. An instance of the FB\_ServerClientConnection function block can be used to manage (establish or remove) a sever connection in a straightforward manner. The same server handle can be transferred to several instances of the function block

FB ServerClientConnection, in order to enable the server to establish several concurrent connections.

### **Syntax**

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

### Return value

Name	Туре	Description
F_CreateServerHnd		Returns TRUE if everything is OK, FALSE if there is an incorrect parameter value.



### Inputs

Name	Туре	Description
sSrvNetID	T_AmsNetID	String containing the AMS network address of the TwinCAT TCP/IP Connection Server. For the local computer (default) an empty string may be specified.
sLocalHost	T_IPv4Addr	Local server IP address (Ipv4) in the form of a string (e.g. '172.13.15.2'). For a server on the local computer (default), an empty string may be entered.
nLocalPort	UDINT	Local server IP port (e.g. 200).
nMode	DWORD	Parameter flags (modes). The permissible parameters are listed here and can be combined by ORing.
		LISTEN_MODE_CLOSEALL:
		All previously opened socket connections are closed (default).
		CONNECT_MODE_ENABLEDBG:
		Enables logging of debug messages in the application log. In order to view the debug messages open the TwinCAT System Manager and activate log view.
bEnable	BOOL	This input determines the behavior of the listener socket. A listener socket opened beforehand remains open as long as this input is TRUE. If this input is FALSE, the listener socket is closed automatically, but only once the last (previously) accepted connection was also closed.

# / Inputs/outputs

Name	Туре	Description
hServer	T_HSERVER	Server handle variable whose internal parameters are to be
		initialized.

### Example:

See <u>FB ServerClientConnection</u> [▶ 46].

### Requirements

Development environment	Target system type	PLC libraries to include (cate- gory group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

# 5.2.2 HSOCKET\_TO\_STRING



The function converts the connection handle of type T\_HSOCKET to a string (e.g. for debug outputs).

The returned string has the following format: "Handle:0xA[BCD] Local:a[aa].b[bb].c[cc].d[dd]:port Remote:a[aa].b[bb].c[cc].d[dd]:port".

Example: "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



### Return value

Name	Туре	Description
HSOCKET TO STRI	STRING	Contains the STRING representation of the connection handle.
NG		

# Inputs

Name	Туре	Description
hSocket	T_HSOCKET	The <u>connection handle [▶ 59]</u> to be converted.

### Requirements

Development environment	, , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.2.3 HSOCKET\_TO\_STRINGEX

```
HSOCKET_TO_STRINGEX

— hSocket T_H5OCKET STRING HSOCKET_TO_STRINGEX
— bLocal BOOL
— bRemote BOOL
```

The function converts the connection handle of type T\_HSOCKET to a string (e.g. for debug outputs).

The returned string has the following format: "Handle:0xA[BCD] Local:a[aa].b[bb].c[cc].d[dd]:port Remote:a[aa].b[bb].c[cc].d[dd]:port".

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

The parameters bLocal and bRemote determine whether the local and/or remote address information should be included in the returned string.

### **Syntax**

```
FUNCTION HSOCKET_TO_STRINGEX : STRING

VAR_INPUT

hSocket : T_HSOCKET;
bLocal : BOOL;
bRemote : BOOL;
END_VAR
```

### Return value

Name	Туре	Description
HSOCKET_TO_ST RINGEX		Contains the hex-based STRING representation of the connection handle.

### Inputs

Name	Туре	Description
hSocket	T_HSOCKET	The <u>connection handle [▶ 59]</u> to be converted.
bLocal	BOOL	TRUE: Include the local address, FALSE: Exclude the local address.
bRemote	BOOL	TRUE: Include the remote address, FALSE: Exclude the remote address.



### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

# 5.2.4 SOCKETADDR\_TO\_STRING

```
SOCKETADDR_TO_STRING

stSockAddr ST_SockAddr STRING SOCKETADDR_TO_STRING
```

The function converts a variable of type ST\_SockAddr to a string (e.g. for debug outputs).

The returned string has the following format: "a[aa].b[bb].c[cc].d[dd]:port"

Example: "172.16.6.195:80"

```
FUNCTION SOCKETADDR_TO_STRING : STRING

VAR_INPUT

stSockAddr : ST_SockAddr;

END VAR
```

### Return value

Name	Туре	Description
SOCKETADDR_TO	STRING	Contains the STRING representation of the socket address.
_STRING		

# Inputs

Name	Туре	Description
stSockeAddr	ST_SockAddr	The variable to be converted.

See ST SockAddr [▶ 57]

### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.3 Data types

# 5.3.1 E\_SocketAcceptMode

E\_SocketAcceptMode specifies which connections are accepted by the server.

#### **Syntax**



#### **Values**

Name	Description
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.

### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

# 5.3.2 E\_SocketConnectionState

TCP/IP Socket Connection Status (eSOCKET\_SUSPENDED == the status changes e.g. from eSOCKET\_CONNECTED => eSOCKET\_DISCONNECTED).

### **Syntax**

#### **Values**

Name	Description
eSOCKET_DISCONNECTED	The connection is interrupted.
eSOCKET_CONNECTED	The connection exists.
	The status of the connection changes from disconnected to connected or from connected to disconnected.

### Requirements

Development environment	, , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.3.3 E\_SocketConnectionlessState

Status information of a connection-less UDP socket (eSOCKET\_TRANSIENT == the status changes from eSOCKET\_CREATED=>eSOCKET\_CLOSED, for example).

### **Syntax**

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



#### **Values**

Name	Description
eSOCKET_CLOSED	The UDP socket is closed.
eSOCKET_CREATED	The UDP socket is created.
eSOCKET_TRANSIENT	The UDP socket changes from closed to open or from open to closed.

#### Requirements

Development environment	, ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)

# 5.3.4 E WinsockError

### **Syntax**

```
TYPE E WinsockError :
    WSOK,
                  := 10004 ,
    WSAEINTR
(* A blocking operation was interrupted by a call to WSACancelBlockingCall. *)
                  := 10009 , (* The file handle supplied is not valid. *)
   WSAEBADF
   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. *)
                    := 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
   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. ^{\star})
   port) is normally permitted. *)
    WSAEADDRNOTAVAIL := 10049 ,(* The requested address is not valid in its context. *)
                   := 10050 , (* A socket operation encountered a dead network. *)
    WSAENETDOWN
   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. *)
                     := 10053
   WSAECONNABORTED
(* An established connection was aborted by the software in your host machine. *)
    WSAECONNRESET := 10054 ,(* An existing connection was forcibly closed by the remote host. *)
                  := 10055 ,
    WSAENOBUFS
(* 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. *)
                       := 10060 ,
     WSAETIMEDOUT
(* 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. *)
                    := 10062 ,(* Cannot translate name. *)
    WSAELOOP
    WSAENAMETOOLONG
                        := 10063 ,(* Name component or name was too long. *)
    WSAEHOSTDOWN
                    := 10064 ,
(* A socket operation failed because the destination host was down. *)
                       := 10065 , (* A socket operation was attempted to an unreachable host. *)
    WSAEHOSTUNREACH
                     := 10066 ,(* Cannot remove a directory that is not empty. *)
    WSAENOTEMPTY
    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. *)
    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. *)
                    := 10102 ,(* No more results can be returned by WSALookupServiceNext. *)
    WSAENOMORE
    WSAECANCELLED := 10103 ,
(* A call to WSALookupServiceEnd was made while this call was still processing. The call has been ca
    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. *)
    \label{eq:wsahost_not_found} \texttt{WSAHOST\_NOT\_FOUND} := 11001 \text{ ,(* No such host is known. *)}
                    := 11002
    WSATRY AGAIN
(* 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. ^{\star})
END TYPE
```

#### Requirements

Development environment		PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.3.5 ST\_SockAddr

The structure contains address information of an open socket.



### **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
```

#### **Values**

Name	Туре	Description
nPort	UDINT	Internet Protocol (IP) port
sAddr	\ ,	Internet Protocol address separated by periods (Ipv4) in the form of a string e.g.: "172.34.12.3"

### Requirements

Development environment	, , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.3.6 ST\_TIsConnectFlags

Additional (optional) client connection parameters.

### **Syntax**

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

### **Values**

Name	Туре	Description
bNoServerCertCheck	BOOL	Disables validation of the server certificate.
blgnoreCnMismatch	BOOL	Ignored if the CommonName in the server certificate does not match the host name specified as sRemoteHost.

### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.3.7 ST\_TIsListenFlags

Additional (optional) server connection parameters.

#### **Syntax**

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



#### Values

Name	Туре	Description
bNoClientCert	BOOL	Client certificate is not required.

#### Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		

# 5.3.8 T\_HSERVER

The variable of this type represents a TCP/IP Server Handle. The Handle has to be initialized with <u>F\_CreateServerHnd [▶ 51]</u> bevor it can be used. In doing so the internal parameters of variables T\_HSERVER are set.



#### Preserve the default structure elements



The structure elements are not to be written or changed.

### Requirements

Development environment	, , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.3.9 T\_HSOCKET

Variables of this type represent a connection handle or a handle of an open socket. Via this handle, data can be sent to or received from a socket. The handle can be used to close an open socket.

#### **Syntax**

```
TYPE T_HSOCKET

STRUCT

handle : UDINT;
localAddr : ST_SockAddr; (* Local address *)
remoteAddr : ST_SockAddr; (* Remote endpoint address *)

END_STRUCT
END_TYPE
```

### **Values**

Name	Туре	Description
handle	UDINT	Internal TwinCAT TCP/IP Connection Server socket handle.
localAddr	ST_SockAddr	Local socket address [▶ 57].
remoteAddr	ST_SockAddr	Remote socket address [ > 57].

The following sockets can be opened and closed via the TwinCAT TCP/IP Connection Server: Listener socket, Remote Client socket or Local Client socket. Depending on which of these sockets was opened by the TwinCAT TCP/IP Connection Server, suitable address information is entered into the localAddr and remoteAddr variables.

#### Connection handle on the server side

• The function block <u>FB SocketListen [▶ 24]</u> opens a listener socket and returns the connection handle of the listener socket.



- The connection handle of the listener sockets is transferred to the function block <u>FB SocketAccept</u>
   [<u>> 25</u>]. FB SocketAccept will then return the connection handles of the remote clients.
- The function block FB\_SocketAccept returns a new connection handle for each connected remote client.
- The connection handle is then transferred to the function blocks <u>FB SocketSend [▶ 26]</u> and/or <u>FB SocketReceive [▶ 28]</u>, in order to be able to exchange data with the remote clients.
- A connection handle of a remote client that is not desirable or no longer required is transferred to the function block <u>FB SocketClose</u> [ > 22], which closes the remote client socket.
- A listener socket connection handle that is no longer required is also transferred to the function block FB\_SocketClose, which closes the listener socket.

#### Connection handle on the client side

- The function block FB SocketConnect [ 21] returns the connection handle of a local client socket.
- The connection handle is then transferred to the function blocks <u>FB SocketSend [▶ 26]</u> and <u>FB SocketReceive [▶ 28]</u>, in order to be able to exchange data with a remote server.
- The same connection handle is then transferred to the function block <u>FB SocketClose [ 22]</u>, in order to close a connection that is no longer required.

The function block <u>FB SocketCloseAll [ 23]</u> can be used to close all connection handles (sockets) that were opened by a PLC runtime system. This means that, if FB\_SocketCloseAll is called in one of the tasks of the first runtime systems (port 801), all sockets that were opened in the first runtime system are closed.

### Requirements

Development environment	, ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_TcpIp (communication)

# 5.4 Global constants

# 5.4.1 Library version

All libraries have a certain version. This version is displayed in the repository of the PLC library. The version number of the library is stored in a global constant (type: ST\_LibVersion).

### Global\_Version

```
VAR_GLOBAL CONSTANT
stLibVersion_Tc2_TcpIp : ST_LibVersion;
END VAR
```

The F\_CmpLibVersion function (in the Tc2\_System library) is used to compare the existing and required version.



### Compatibility with TwinCAT 2

Query options for TwinCAT2 libraries are no longer available!

### Requirements

Development environment	J , , , , , , , , , , , , , , , , , , ,	PLC libraries to include (category group)
TwinCAT v3.1.0	PC, or CX (x86, X64, ARM)	Tc2_Tcplp (communication)



# 5.4.2 Parameter list

### **Param**

Name	Type	Value	Description
TCPADS_MAXUDP_BUFFSIZE	UDINT	16#2000	Max. byte length of the internal UDP send/receive buffer (8192 bytes).
TCPADS_TLS_HOSTNAME_SIZE	UDINT	255	Max. length of the host name string.
TCPADS_TLS_CERTIFICATE_ PATH_SIZE	UDINT	255	Max. length of the certificate path string.
TCPADS_TLS_KEY_PASSWO RD_SIZE	UDINT	255	Max. length of the certificate password path string.
TCPADS_TLS_PSK_IDENTITY _SIZE	UDINT	255	Max. length of the PSK identity string.
TCPADS_TLS_MAX_PSK_KEY _SIZE	UDINT	128	Max. byte length of the PSK key.

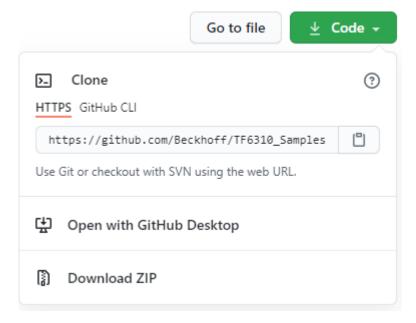
# Requirements

Development environment		PLC libraries to be integrated (category group)
TF6310 v3.3.15.0 or later	PC or CX (x86, x64, ARM)	Tc2_TcpIp (Communication)
TwinCAT v3.1.0		



# 6 Samples

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: <a href="https://github.com/Beckhoff/TF6310">https://github.com/Beckhoff/TF6310</a> Samples. There you have the option to clone the repository or download a ZIP file containing the sample.



# 6.1 TCP

# 6.1.1 Sample01: "Echo" client/server (base blocks)

### **6.1.1.1** Overview

The following example shows an implementation of an "echo" client/server. The client sends a test string to the server at certain intervals (e.g. every second). The remote server then immediately resends the same string to the client.

In this sample, the client is implemented in the PLC and as a .NET application written in C#. The PLC client can create several instances of the communication, simulating several TCP connections at once. The .NET sample client only establishes one concurrent connection. The server is able to communicate with several clients.

In addition, several instances of the server may be created. Each server instance is then addressed via a different port number which can be used by the client to connect to a specific server instance. The server implementation is more difficult if the server has to communicate with more than one client.

Feel free to use and customize this sample to your needs.

### System requirements

- TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP
- If two computers are used to execute the sample (one client and one server), the Function TF6310 needs to be installed on both computers
- If one computer is used to execute the sample, e.g. client and server running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks
- · To run the .NET sample client, only .NET Framework 4.0 is needed



#### **Project downloads**

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/TCP/Sample01

https://github.com/Beckhoff/TF6310\_Samples/tree/master/C%23/SampleClient

#### **Project description**

The following links provide documentation for the three components. Additionally, an own article explains how to start the PLC samples with step-by-step instructions.

- Integration in TwinCAT and Test [▶ 64] (Starting the PLC samples)
- PLC Client [▶ 67] (PLC client documentation: FB LocalClient function block [▶ 67])
- PLC Server [ > 71] (PLC serve documentation: FB\_LocalServer function block [ > 71])
- .NET client [▶ 77] (.NET client documentation: .NET sample client [▶ 77])

#### Additional functions of the PLC sample projects

Some functions, constants and function blocks are used in the sample projects, which are briefly described below:

#### LogError function

```
LOGERROR

msg:STRING(80) LogError:DINT
nErrid:DWORD
```

The function writes a message with the error code into the logbook of the operating system (Event Viewer). The global variable bLogDebugMessages must first be set to TRUE.

### LogMessage function

```
LOGMESSAGE

—msg:STRING(80) LogMessage:DINT—
hSocket:T_HSOCKET
```

The function writes a message into the logbook of the operating system (Event Viewer) if a new socket was opened or closed. The global variable bLogDebugMessages must first be set to TRUE.

### SCODE\_CODE function

```
FUNCTION SCODE_CODE : DWORD

SC : UDINT SCODE_CODE : DWORD
```

The function masks the least significant 16 bits of a Win32 error code returns them.

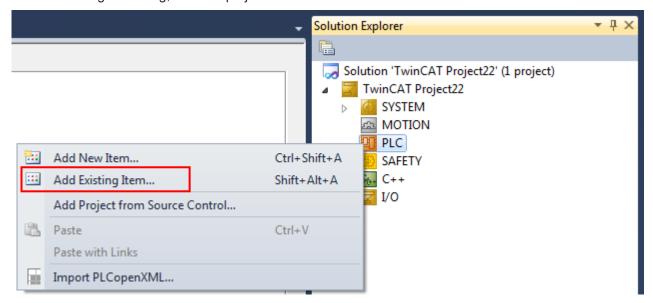


#### Global variables

Name	Default value	Description
bLogDebugMessages	TRUE	Activates/deactivates writing of messages into the log book of the operating system
MAX_CLIENT_CONNECTIONS	5	Max. number of remote clients, that can connect to the server at the same time.
MAX_PLCPRJ_RXBUFFER_SIZE	1000	Max. length of the internal receive buffer
PLCPRJ_RECONNECT_TIME	T#3s	Once this time has elapsed, the local client will attempt to re-establish the connection with the remote server
PLCPRJ_SEND_CYCLE_TIME	T#1s	The test string is sent cyclically at these intervals from the local client to the remote server
PLCPRJ_RECEIVE_POLLING_TI ME	T#1s	The client reads (polls) data from the server using this cycle
PLCPRJ_RECEIVE_TIMEOUT	T#10s	After this time has elapsed, the local client aborts the reception if no data bytes could be received during this time
PLCPRJ_ERROR_RECEIVE_BUF FER_OVERFLOW	16#8101	Sample project error code: Too many characters without zero termination were received
PLCPRJ_ERROR_RECEIVE_TIM EOUT	16#8102	Sample project error code: No new data could be received within the timeout time (PLCPRJ_RECEIVE_TIMEOUT)

# 6.1.1.2 Integration in TwinCAT and Test

The following section describes how to prepare and start the PLC server and client. The PLC samples are provided as TwinCAT 3 PLC project files. To import a PLC project into TwinCAT XAE, first create a new TwinCAT 3 Solution. Then select the **Add Existing Item** command in the context menu of the PLC node and select the downloaded sample file (*Plc 3.x Project archive (\*.tpzip)*) as the file type in the dialog that opens. After confirming the dialog, the PLC project is added to the Solution.



### Sample PLC server

Create a new TwinCAT 3 solution in TwinCAT XAE and import the TwinCAT TCP/IP server project. Select a target system. The function must also be installed on the target system and licenses for TF6310 must have been generated. Keep the TwinCAT 3 Solution open.

```
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
```

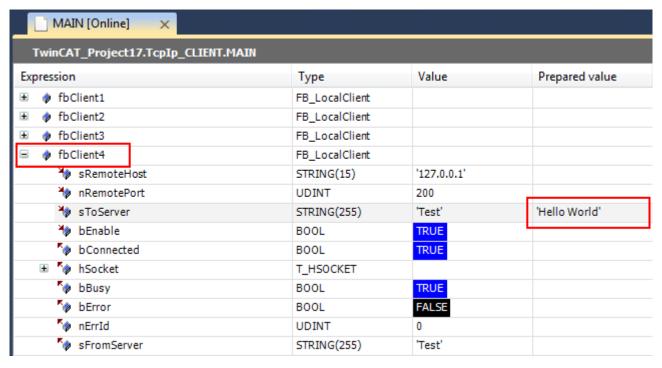
#### Sample PLC client

Import the TCP/IP client project into the TwinCAT 3 Solution as a second PLC project. Link this PLC project to a different task than the server sample. The IP address of the server must be adapted to your system (initialization values of the sRemoteHost variable). In this case, the server is on the same PC, so enter 127.0.0.1. Enable the configuration, log in and start the server and then the client PLC project.

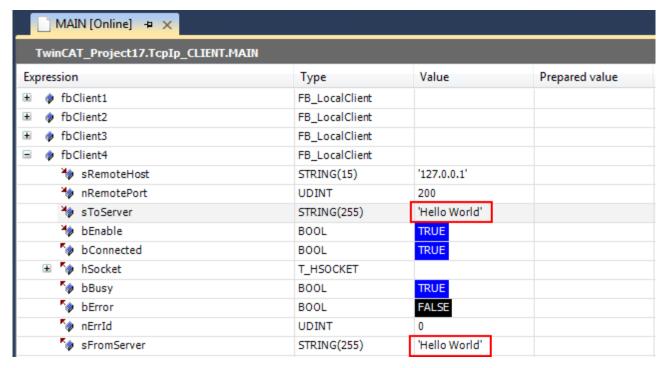
```
PROGRAM MAIN
VAR
    fbClient1
                     : FB LocalClient := ( sRemoteHost:= '127.0.0.1' (* IP address of remote server! *)
  nRemotePort:= 200 );
    fbClient2 : FB_LocalClient := ( sRemoteHost:= '127.0.0.1', nRemotePort:= 200 );
fbClient3 : FB_LocalClient := ( sRemoteHost:= '127.0.0.1',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

When setting one of the bEnableCientX variables, up to five client instances can be enabled. Each client sends one string per second to the server (default: 'TEST'). The server returns the same string to the client (echo server). For the test, a string with a counter value is generated automatically for the first three instances. The first client is enabled automatically when the program is started. Set the bEnableCilent4 variable in the client project to TRUE. The new client then attempts to establish a connection to the server. If successful, the 'TEST' string is sent cyclically. Now open the fbClient4 instance of the FB\_LocalClient function block. Double-click to open the dialog for writing the sToString variable and change the value of the string variable to 'Hello', for example.





Close the dialog with **OK**. Force the new value into the PLC. Shortly afterwards, the value returned by the server can be viewed online.



Now open the fbServer instance of the FB\_LocalServer function block in the server project. The string: 'Hello' can be seen in the online data of the server.



MAIN [Online] ×					
TwinCAT_Project17.TcpIp_SERVER.MAIN					
Expression	Туре	Value	Prepared value		
	ARRAY [1MAX_CLI				
	FB_RemoteClient				
	FB_RemoteClient				
	FB_RemoteClient				
fbRemoteClient[4]	FB_RemoteClient				
⊞ 🧤 hListener	T_HSOCKET				
¥≱ bEnable	BOOL	TRUE			
<b>™</b> bAccepted	BOOL	FALSE			
⊞ 🍫 hSocket	T_HSOCKET				
<b>™</b>	BOOL	TRUE			
<b>™</b> bError	BOOL	FALSE			
🏇 nErrID	UDINT	0			
🏇 sFromClient	STRING(255)	'Hello World'			
⊞ 🧼 fbAccept	FB_SocketAccept				

### 6.1.1.3 PLC Client

# 6.1.1.3.1 FB\_LocalClient

```
FB_LocalClient
-sRemoteHost bConnected—
-nRemotePort hSocket—
-sToServer bBusy—
-bEnable bError—
nErrld—
sFromServer—
```

If the bEnable input is set, the system will keep trying to establish the connection to the remote server once the CLIENT\_RECONNECT\_TIME has elapsed. The remote server is identified via the sRemoteHost IP address and the nRemotePort IP port address. The data exchange with the server was encapsulated in a separate function block <u>FB ClientDataExcha</u> [\*\*<u>69</u>]. The data exchange is cyclic, always after PLCPRJ\_SEND\_CYCLE\_TIME has expired. The sToServer string variable is sent to the server and the string returned by the server is available at the sFormServer output. Another implementation, in which the remote server is addressed as required is also possible. In the event of an error, the existing connection is closed, and a new connection is established.

#### Interface

```
FUNCTION BLOCK FB LocalClient
VAR INPUT
    sRemoteHost : STRING(15) := '127.0.0.1'; (* IP adress of remote server *)
    nRemotePort : UDINT := 0;
sToServer : T_MaxString:= 'TEST';
    sToServer
    bEnable
                    : BOOL;
END VAR
VAR OUTPUT
    bConnected
                     : BOOL;
    hSocket
                     : T HSOCKET;
    bBusy
                     : BOOL;
                     : BOOL;
    bError
    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
```

#### Implementation

```
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;
           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;
                          := CLIENT_STATE_IDLE;
                eStep
                LogMessage( 'LOCAL client CONNECTED!', hSocket );
                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;
                         := CLIENT STATE IDLE;
                (* 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;
           bBusy := FALSE;
            eStep := CLIENT STATE IDLE;
        END IF
END CASE
```

### 6.1.1.3.2 FB\_ClientDataExcha

```
FB_ClientDataExcha
-hSocket bBusy--
-sToServer bError--
-bExecute nErrld--
sFromServer--
```

In the event of an rising edge at the bExecute input, a null-terminated string is sent to the remote server, and a string returned by the remote server is read. The function block will try reading the data until null termination was detected in the string received. If the PLCPRJ\_RECEIVE\_TIMEOUT time-out period is exceeded or if an error occurs, reception is aborted. Data are attempted to be read again after a certain delay time, if no new data could be read during the last read attempt. This reduces the system load.

#### Interface

```
FUNCTION BLOCK FB ClientDataExcha
VAR_INPUT
               : T HSOCKET;
   hSocket
    sToServer : T_MaxString;
   bExecute : BOOL;
END VAR
VAR_OUTPUT
   bBusy
               : BOOL;
           : BOOL;
: UDINT;
   nErrId
   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
```



#### Implementation

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

# 6.1.1.4.1 FB\_LocalServer

```
FB_LocalServer
-sLocalHost bListening
-nLocalPort hListener
-bEnable nAcceptedClients
-bBusy
-bError
-bError
```

The server must first be allocated a unique sLocalHost IP address and an nLocaPort IP port number. If the bEnable input is set, the local server will repeatedly try to open the listener socket once the SERVER\_RECONNECT\_TIME has elapsed. The listener socket can usually be opened at the first attempt, if the TwinCAT TCP/IP Connection Server is located on the local PC. The functionality of a remote client was encapsulated in the function block FB\_RemoteClient [\rightarrow 73]. The remote client instances are activated once the listener socket was opened successfully. Each instance of the FB\_RemoteClient corresponds to a remote client, with which the local server can communicate simultaneously. The maximum number of remote clients communicating with the server can be modified via the value of the MAX\_CLIENT\_CONNECTIONS constant. In the event of an error, all remote client connections are closed first and then the listener socket. The nAcceptedClients output provides information about the current number of connected clients.

### Interface

```
FUNCTION BLOCK FB LocalServer
VAR INPUT
                     : STRING(15) := '127.0.0.1'; (* own IP address! *)
   sLocalHost
   nLocalPort
                    : UDINT := 0;
                     : BOOL;
   bEnable
END VAR
VAR_OUTPUT
   bListening
                    : BOOL;
   hListener
                    : T_HSOCKET;
   nAcceptedClients : UDINT;
  bBusv
                   : BOOL;
```



```
bError
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
```

### Implementation

```
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 );
                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;
           bBusy := FALSE;
            eStep := SERVER STATE IDLE;
        END IF
END CASE
```

#### 6.1.1.4.2 FB RemoteClient

```
FB_RemoteClient
-hListener bAccepted-
-bEnable hSocket-
bBusy-
bError-
nErrID-
sFromClient
```

If the bEnable input is set, an attempt is made to accept the connection request of a remote client, once the SERVER\_ACCEPT\_POOLING\_TIME has elapsed. The data exchange with the remote client was encapsulated in a separate function block <u>FB\_ServerDataExcha[\rightarrow\_75]</u>. Once the connection was established successfully, the instance is enabled via the FB\_ServerDataExcha function block. If an error occurs, the accepted connection is closed and a new one is established.

#### Interface

```
FUNCTION BLOCK FB RemoteClient
VAR INPUT
               : T HSOCKET;
   hListener
   bEnable
                : BOOL;
END VAR
VAR OUTPUT
               : BOOL;
   bAccepted
                : T_HSOCKET;
   hSocket
   bBusy
                : BOOL;
               : BOOL;
   nErrID
                : UDINT;
   sFromClient : T_MaxString;
END_VAR
              : FB SocketAccept := ( sSrvNetID := '', tTimeout := DEFAULT ADS TIMEOUT );
fbAccept
```



#### Implementation

```
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;
               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;
            eStep := CLIENT STATE IDLE;
                bBusy := FALSE;
        END IF
END CASE
```

# 6.1.1.4.3 FB ServerDataExcha

```
FB_ServerDataExcha
-hSocket bBusy-
-bExecute bError-
nErrID-
sFromClient-
```

In the event of an rising edge at the bExecute input, a zero-terminated string is read by the remote client and returned to the remote client, if zero termination was detected. The function block will try reading the data until zero termination was detected in the string received. Reception is aborted in the event of an error, and if no new data were received within the PLCPRJ\_RECEIVE\_TIMEOUT timeout time. Data are attempted to be read again after a certain delay time, if no new data could be read during the last read attempt. This reduces the system load.

#### Interface

```
FUNCTION BLOCK FB ServerDataExcha
VAR INPUT
   hSocket
               : T HSOCKET;
   bExecute : BOOL;
END VAR
VAR OUTPUT
   bBusv
               : BOOL;
            : BOOL;
: UDINT;
   bError
    nErrID
    sFromClient : T MaxString;
END VAR
VAR
   fbSocketReceive : FB_SocketReceive := ( sSrvNetId := '', tTimeout := DEFAULT ADS TIMEOUT );
    fbSocketSend := FB SocketSend := ( sSrvNetId := '', tTimeout := DEFAULT ADS TIMEOUT );
             : E DataExchaSteps;
   RisingEdge : R TRIG;
    fbReceiveTON : TON;
    fbDisconnectTON
                       : TON;
    cbReceived, startPos, endPos, idx : UDINT;
            : UDINT;
    cbFrame
    rxBuffer
                   : ARRAY[0..MAX PLCPRJ RXBUFFER SIZE] OF BYTE;
END_VAR
```

#### Implementation

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

DATAEXCHA STATE IDLE:
```



```
IF RisingEdge.Q THEN
           bBusy := TRUE;
           bError := FALSE;
           nErrId := 0;
           \texttt{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.O 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 *)
                                  := cbReceived + fbSocketReceive.nRecBytes - 1;
                   endPos
(* rxBuffer array index of last data byte *)
                   cbReceived := cbReceived + fbSocketReceive.nRecBytes;
(* calculate the number of received data bytes *)
                              := 0; (* reset frame length *)
                   cbFrame
                   IF cbReceived < SIZEOF( sFromClient ) THEN(* no overflow *)</pre>
                       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) *)
                               {\tt 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 *)
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
```

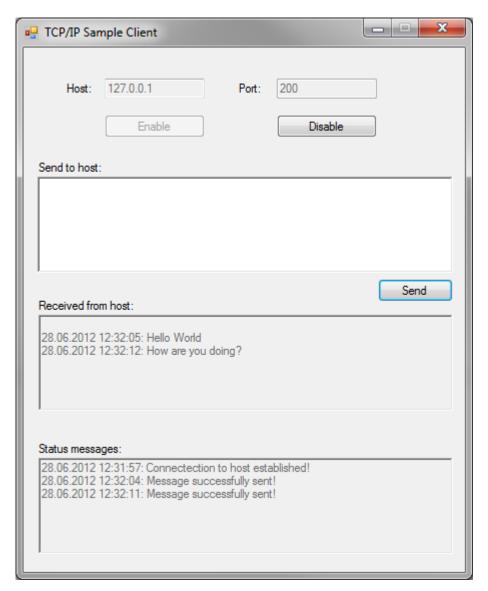


```
END IF
            ELSE(* receive error *)
                LogError( 'FB_SocketReceive (remote client)', fbSocketReceive.nErrId );
nErrId := fbSocketReceive.nErrId;
                eStep := DATAEXCHA_STATE_ERROR;
            END IF
        END IF
    DATAEXCHA STATE SEND START:
        fbSocketSend( 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;
                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

This sample project shows how a client for the PLC TCP/IP server can be realized under .NET4.0 in C#.





The sample uses the .NET libraries System.Net and System.Net.Sockets, with which a programmer can easily use socket functions. Pressing **Enable** causes the application to cyclically (depending on the value of TIMERTICK in [ms]) attempt to establish a connection to the server. If successful, a string with a maximum length of 255 characters can be sent to the server via the Send button. This string is then accepted by the server and sent back to the client. The connection is automatically closed on the server side after the SERVER\_RECEIVE\_TIMEOUT time defined in the server sample has expired, default: 50 seconds, if the server was unable to receive any new data from the client within this time.

```
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
\star 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 rcvMsq.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
);
^{\star} 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
   ^{\star} character in the buffer needs to be a termination character, so that the TCP/IP-
Server knows
    ^{\star} 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++)</pre>
      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
1);
      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 rcvMsq.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
```

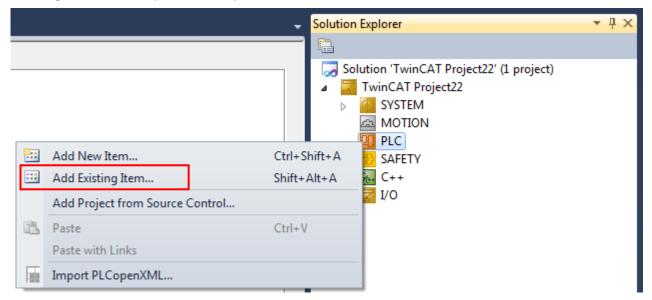


```
timer1.Enabled = false;
    _socket.Disconnect(true);
if (!_socket.Connected)
{
        _isConnected = false;
        cmd_disable.Enabled = false;
        cmd_enable.Enabled = true;
        txt_host.Enabled = true;
        txt_port.Enabled = true;
        rtb_sendMsg.Enabled = false;
        cmd_send.Enabled = false;
        rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Connectection to host closed!\n");
        rtb_rcvMsg.Clear();
        rtb_statMsg.Clear();
    }
}
```

# 6.1.2 Sample02: "Echo" client /server

This sample uses the functions of the former TcSocketHelper.Lib, which is now integrated into the Tc2\_TcpIp library. It shows a client/server PLC application based on the functions of the former SocketHelper library.

The client cyclically sends a test string (sToServer) to the remote server. The server returns this string unchanged to the client (sFromServer).



## System requirements

- · TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP
- If two computers are used to execute the sample (one client and one server), the Function TF6310 needs to be installed on both computers
- If one computer is used to execute the sample, e.g. client and server running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks.

#### **Project downloads**

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/TCP/Sample02

#### **Project information**

The default communication settings used in the above samples are as follows:

• PLC client application: Port and IP address of the remote server: 200, '127.0.0.1'



• PLC server application: Port and IP address of the local server: 200, '127.0.0.1'

To test the client and server application on two different PCs, you have to adjust the port and the IP address accordingly.

However, you can also test the client and server samples with the default values on a single computer by loading the client application into the first PLC runtime system and the server application into the second PLC runtime system.

The behavior of the PLC project sample is determined by the following global variables/constants.

Constant	Value	Description
PLCPRJ_MAX_CONNECTIONS	5	Max. number of server $\rightarrow$ client connections. A server can establish connections to more than one client. A client can establish a connection to only one server at a time.
PLCPRJ_SERVER_RESPONSE_ TIMEOUT	T#10s	Max. delay time (timeout time) after which a server should send a response to the client.
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	Cycle time based on which a client sends send data (TX) to the server.
PLCPRJ_RECEIVER_POLLING_C YCLE_TIME	T#200ms	Cycle time based on which a client or server polls for receive data (RX).
PLCPRJ_BUFFER_SIZE	10000	Max. internal buffer size for RX/TX data.

The PLC samples define and use the following internal error codes:

Error code	Value	Description
PLCPRJ_ERROR_RECEIVE_BUF FER_OVERFLOW	16#8101	The internal receive buffer reports an overflow.
PLCPRJ_ERROR_SEND_BUFFE R_OVERFLOW	16#8102	The internal send buffer reports an overflow.
PLCPRJ_ERROR_RESPONSE_TI MEOUT	16#8103	The server has not sent the response within the specified timeout time.
PLCPRJ_ERROR_INVALID_FRA ME_FORMAT	16#8104	The telegram formatting is incorrect (size, faulty data bytes etc.).

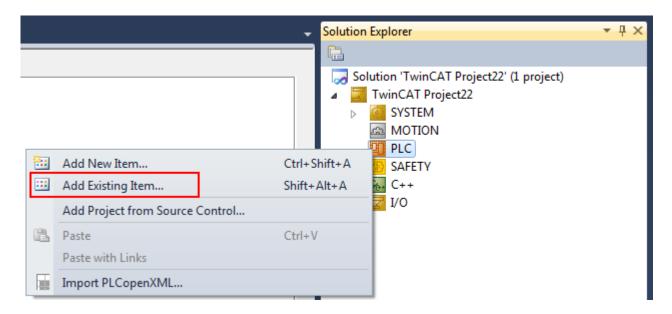
The client and server applications (FB\_ServerApplication, FB\_ClientApplication) were implemented as function blocks. The application and the connection can thus be instanced repeatedly.

# 6.1.3 Sample03: "Echo" client/server

This sample uses the functions of the former TcSocketHelper.Lib, which is now integrated into the Tc2\_Tcplp library. It shows a client/server PLC application based on the functions of the former SocketHelper library.

The client cyclically sends a test string (sToServer) to the remote server. The server returns this string unchanged to the client (sFromServer). The difference between this sample and Sample02 is that the server can establish up to five connections and the client application can start up to five client instances. Each instance establishes a connection to the server.





#### System requirements

- · TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP
- If two computers are used to execute the sample (one client and one server), the Function TF6310 needs to be installed on both computers
- If one computer is used to execute the sample, e.g. client and server running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks

#### **Project downloads**

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/TCP/Sample03

#### **Project information**

The default communication settings used in the above samples are as follows:

- PLC client application: Port and IP address of the remote server: 200, '127.0.0.1'
- PLC server application: Port and IP address of the local server: 200, '127.0.0.1'

To test the client and server application on two different PCs, you have to adjust the port and the IP address accordingly.

However, you can also test the client and server samples with the default values on a single computer by loading the client application into the first PLC runtime system and the server application into the second PLC runtime system.

The behavior of the PLC project sample is determined by the following global variables/constants.

Constant	Value	Description
PLCPRJ_MAX_CONNECTIONS	5	Max. number of server->client connections. A server can establish connections to more than one client. A client can establish a connection to only one server at a time.
PLCPRJ_SERVER_RESPONSE_ TIMEOUT	T#10s	Max. delay time (timeout time) after which a server should send a response to the client.
PLCPRJ_CLIENT_SEND_CYCLE_ TIME	T#1s	Cycle time based on which a client sends send data (TX) to the server.
PLCPRJ_RECEIVER_POLLING_C YCLE_TIME	T#200ms	Cycle time based on which a client or server polls for receive data (RX).
PLCPRJ_BUFFER_SIZE	10000	Max. internal buffer size for RX/TX data.



The PLC samples define and use the following internal error codes:

Error code	Value	Description
PLCPRJ_ERROR_RECEIVE_BUF FER_OVERFLOW	16#8101	The internal receive buffer reports an overflow.
PLCPRJ_ERROR_SEND_BUFFE R_OVERFLOW	16#8102	The internal send buffer reports an overflow.
PLCPRJ_ERROR_RESPONSE_TI MEOUT	16#8103	The server has not sent the response within the specified timeout time.
PLCPRJ_ERROR_INVALID_FRA ME_FORMAT	16#8104	The telegram formatting is incorrect (size, faulty data bytes etc.).

The client and server applications (FB\_ServerApplication, FB\_ClientApplication) were implemented as function blocks. The application and the connection can thus be instanced repeatedly.

# 6.1.4 Sample04: Binary data exchange

This sample uses the functions of the former TcSocketHelper.Lib, which is now integrated into the Tc2\_Tcplp library. It shows a client/server PLC application based on the functions of the former SocketHelper library.

This sample provides a client-server application for the exchange of binary data. A simple sample protocol has been implemented for this purpose. The length of the binary data and a frame counter for the sent and received telegrams are transferred in the protocol header.

The structure of the binary data is defined by the PLC structure ST\_ApplicationBinaryData. The binary data are appended to the header and transferred. The instances of the binary structure are called toServer, fromServer on the client side and toClient, fromClient on the server side.

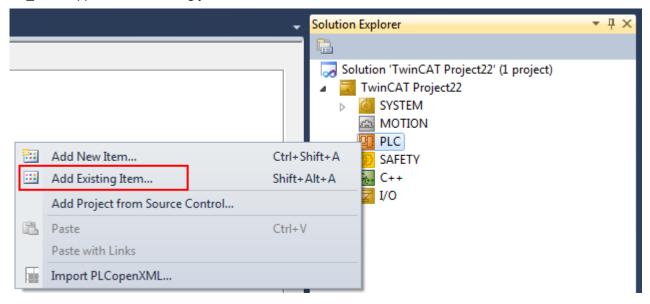
The structure declaration on the client and server sides can be adapted as required. The structure declaration must be identical on both sides.

The maximum size of the structure must not exceed the maximum buffer size of the send/receive Fifos. The maximum buffer size is defined by a constant.

The server functionality is implemented in the function block FB\_ServerApplication and the client functionality in the function block FB ClientApplication.

In the standard implementation the client cyclically sends the data of the binary structure to the server and waits for a response from the server. The server modifies some data and returns them to the client.

If you require a specific function, you must modify the function blocks FB\_ServerApplication and FB\_ClientApplication accordingly.





#### System requirements

- · TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP
- If two computers are used to execute the sample (one client and one server), the Function TF6310 needs to be installed on both computers
- If one computer is used to execute the sample, e.g. client and server running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks.

#### **Project downloads**

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/TCP/Sample04

#### **Project information**

The default communication settings used in the above samples are as follows:

- PLC client application: Port and IP address of the remote server: 200, '127.0.0.1'
- PLC server application: Port and IP address of the local server: 200, '127.0.0.1'

To test the client and server application on two different PCs, you have to adjust the port and the IP address accordingly.

However, you can also test the client and server samples with the default values on a single computer by loading the client application into the first PLC runtime system and the server application into the second PLC runtime system.

The behavior of the PLC project sample is determined by the following global variables/constants.

Constant	Value	Description
PLCPRJ_MAX_CONNECTIONS	5	Max. number of server->client connections. A server can establish connections to more than one client. A client can establish a connection to only one server at a time.
PLCPRJ_SERVER_RESPONSE_ TIMEOUT	T#10s	Max. delay time (timeout time) after which a server should send a response to the client.
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	Cycle time based on which a client sends send data (TX) to the server.
PLCPRJ_RECEIVER_POLLING_C YCLE_TIME	T#200ms	Cycle time based on which a client or server polls for receive data (RX).
PLCPRJ_BUFFER_SIZE	10000	Max. internal buffer size for RX/TX data.

The PLC samples define and use the following internal error codes:

Error code	Value	Description
PLCPRJ_ERROR_RECEIVE_BUF FER_OVERFLOW	16#8101	The internal receive buffer reports an overflow.
PLCPRJ_ERROR_SEND_BUFFE R_OVERFLOW	16#8102	The internal send buffer reports an overflow.
PLCPRJ_ERROR_RESPONSE_TI MEOUT	16#8103	The server has not sent the response within the specified timeout time.
PLCPRJ_ERROR_INVALID_FRA ME_FORMAT	16#8104	The telegram formatting is incorrect (size, faulty data bytes etc.).

The client and server applications (FB\_ServerApplication, FB\_ClientApplication) were implemented as function blocks. The application and the connection can thus be instanced repeatedly.



# 6.1.5 Sample05: Binary data exchange

This sample uses the functions of the former TcSocketHelper.Lib, which is now integrated into the Tc2\_Tcplp library. It shows a client/server PLC application based on the functions of the former SocketHelper library.

This sample provides a client-server application for the exchange of binary data. A simple sample protocol has been implemented for this purpose. The length of the binary data and a frame counter for the sent and received telegrams are transferred in the protocol header.

The structure of the binary data is defined by the PLC structure ST\_ApplicationBinaryData. The binary data are appended to the header and transferred. The instances of the binary structure are called toServer, fromServer on the client side and toClient, fromClient on the server side.

The structure declaration on the client and server sides can be adapted as required. The structure declaration must be identical on both sides.

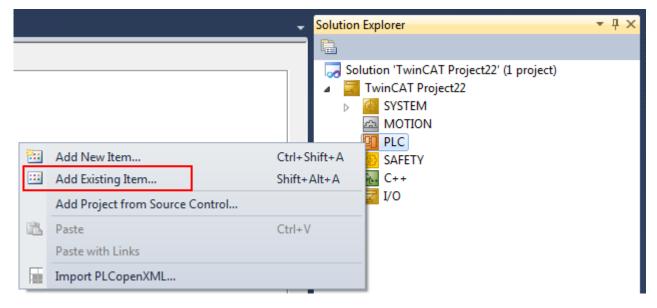
The maximum size of the structure must not exceed the maximum buffer size of the send/receive Fifos. The maximum buffer size is defined by a constant.

The server functionality is implemented in the function block FB\_ServerApplication and the client functionality in the function block FB ClientApplication.

In the standard implementation the client cyclically sends the data of the binary structure to the server and waits for a response from the server. The server modifies some data and returns them to the client.

If you require a specific function, you must modify the function blocks FB\_ServerApplication and FB\_ClientApplication accordingly.

The difference between this sample and Sample04 is that the server can establish up to 5 connections and the client application can have up to five client instances. Each instance establishes a connection to the server.



#### System requirements

- TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP
- If two computers are used to execute the sample (one client and one server), the Function TF6310 needs to be installed on both computers
- If one computer is used to execute the sample, e.g. client and server running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks.

#### **Project downloads**

https://github.com/Beckhoff/TF6310 Samples/tree/master/PLC/TCP/Sample05



#### **Project information**

The default communication settings used in the above samples are as follows:

- PLC client application: Port and IP address of the remote server: 200, '127.0.0.1'
- PLC server application: Port and IP address of the local server: 200, '127.0.0.1'

To test the client and server application on two different PCs, you have to adjust the port and the IP address accordingly.

However, you can also test the client and server samples with the default values on a single computer by loading the client application into the first PLC runtime system and the server application into the second PLC runtime system.

The behavior of the PLC project sample is determined by the following global variables/constants.

Constant	Value	Description
PLCPRJ_MAX_CONNECTIONS	5	Max. number of server->client connections. A server can establish connections to more than one client. A client can establish a connection to only one server at a time.
PLCPRJ_SERVER_RESPONSE_ TIMEOUT	T#10s	Max. delay time (timeout time) after which a server should send a response to the client.
PLCPRJ_CLIENT_SEND_CYCLE_ TIME	T#1s	Cycle time based on which a client sends send data (TX) to the server.
PLCPRJ_RECEIVER_POLLING_C YCLE_TIME	T#200ms	Cycle time based on which a client or server polls for receive data (RX).
PLCPRJ_BUFFER_SIZE	10000	Max. internal buffer size for RX/TX data.

The PLC samples define and use the following internal error codes:

Error code	Value	Description
PLCPRJ_ERROR_RECEIVE_BUF FER_OVERFLOW	16#8101	The internal receive buffer reports an overflow.
PLCPRJ_ERROR_SEND_BUFFE R_OVERFLOW	16#8102	The internal send buffer reports an overflow.
PLCPRJ_ERROR_RESPONSE_TI MEOUT	16#8103	The server has not sent the response within the specified timeout time.
PLCPRJ_ERROR_INVALID_FRA ME_FORMAT	16#8104	The telegram formatting is incorrect (size, faulty data bytes etc.).

The client and server applications (FB\_ServerApplication, FB\_ClientApplication) were implemented as function blocks. The application and the connection can thus be instanced repeatedly.

# 6.1.6 Sample06: "Echo" client/server with TLS (basic modules)

The following sample is essentially based on Sample01 and shows an exemplary implementation of an "Echo" client/server system. The client sends a test string to the server at certain intervals (e.g. every second). The remote server sends this string back to the client.

In contrast to Sample01, the communication connection in this sample is secured via TLS with client/server certificates. The certificates are not part of the sample and must be created by the user.

In essence, this sample thus illustrates the use of the function blocks <u>FB\_TlsSocketConnect</u> [ $\triangleright$ \_36], <u>FB\_TlsSocketCreate</u> [ $\triangleright$ \_39], <u>FB\_TlsSocketListen</u> [ $\triangleright$ \_38], <u>FB\_TlsSocketAddCa</u> [ $\triangleright$ \_40], <u>FB\_TlsSocketAddCrl</u> [ $\triangleright$ \_41], and <u>FB\_TlsSocketSetCert</u> [ $\triangleright$ \_42]. These were integrated accordingly into the state machine of the client and server sample from Sample01.

#### **Project downloads**

https://github.com/Beckhoff/TF6310 Samples/tree/master/PLC/TCP/Sample06



# 6.1.7 Sample07: "Echo" client/server with TLS-PSK (basic modules)

The following sample is essentially based on Sample01 and shows an exemplary implementation of an "Echo" client/server system. The client sends a test string to the server at certain intervals (e.g. every second). The remote server sends this string back to the client.

In contrast to Sample01, the communication connection in this sample is secured via TLS with a pre-shared key (PSK).

In essence, this sample thus illustrates the use of the function blocks <u>FB TlsSocketConnect [\rightarrow 36]</u>, <u>FB TlsSocketListen [\rightarrow 38]</u>, and <u>FB TlsSocketSetPsk [\rightarrow 43]</u>. These were integrated accordingly into the state machine of the client and server sample from Sample01.

#### **Project downloads**

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/TCP/Sample07

# 6.2 UDP

# 6.2.1 Sample01: Peer-to-peer communication

#### 6.2.1.1 Overview

The following example demonstrates the implementation of a simple Peer-to-Peer application in the PLC and consists of two PLC projects (PeerA and PeerB) plus a .NET application which also acts as a separate peer. All peer applications send a test string to a remote peer and at the same time receive strings from a remote peer. The received strings are displayed in a message box on the monitor of the target computer. Feel free to use and customize this sample to your needs.

#### System requirements

- TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP
- If two computers are used to execute the sample, the Function TF6310 needs to be installed on both computers
- If one computer is used to execute the sample, e.g. Peer A und Peer B running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks
- To run the .NET sample client, only .NET Framework 4.0 is needed

#### **Project downloads**

The sources of the two PLC devices only differ in terms of different IP addresses of the remote communication partners.

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/UDP/Sample01

https://github.com/Beckhoff/TF6310\_Samples/tree/master/C%23/SampleClientUdp

#### **Project description**

The following links provide documentation for each component. Additionally, an own article explains how to start the PLC samples with step-by-step instructions.

- Integration in TwinCAT and Test [▶ 90] (Starting the PLC samples)
- PLC devices A and B [▶ 91] (Peer-to-Peer PLC application)
- .NET communication [▶ 94] (.NET sample client)



#### Additional functions of the PLC sample projects

Some functions, constants and function blocks are used in the PLC samples, which are briefly described below:

#### Fifo function block

```
FUNCTION_BLOCK FB_Fifo

VAR_INPUT

new: ST_FifoEntry;

END_VAR

VAR_OUTPUT

bok: BOOL;

old: ST_FifoEntry;

END_VAR
```

This is a simple Fifo function block. One instance of this function block is used as "send Fifo", another one as "receive Fifo". The messages to be sent are stored in the send Fifo, the received messages are stored in the receive Fifo. The bOk output variable is set to FALSE if errors occurred during the last action (AddTail or RemoveHead) (Fifo empty or overfilled).

A Fifo entry consists of the following components:

```
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 function

```
LOGERROR

msg:STRING(80) LogError:DINT

nErrid:DWORD
```

The function writes a message with the error code into the logbook of the operating system (Event Viewer). The global variable bLogDebugMessages must first be set to TRUE.

#### LogMessage function

```
LOGMESSAGE

msg:STRING(80) LogMessage:DINT
hSocket:T_HSOCKET
```

The function writes a message into the logbook of the operating system (Event Viewer) if a new socket was opened or closed. The global variable bLogDebugMessages must first be set to TRUE.

#### SCODE\_CODE function

```
SCODE_CODE : DWORD

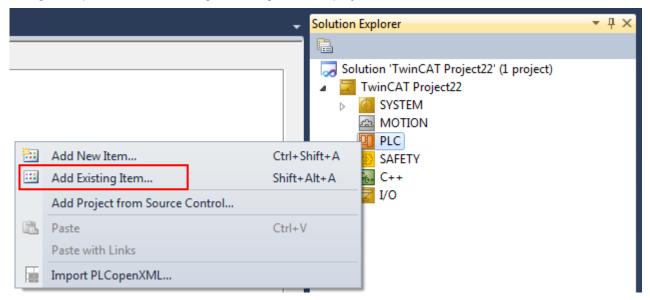
SC : UDINT SCODE_CODE : DWORD
```

The function masks the least significant 16 bits of a Win32 error code returns them.



# 6.2.1.2 Integration in TwinCAT and Test

The PLC samples are provided as TwinCAT 3 PLC project files. To import a PLC project into TwinCAT XAE, first create a new TwinCAT 3 Solution. Then select the **Add Existing Item** command in the context menu of the PLC node and select the downloaded sample file (*Plc 3.x Project archive (\*.tpzip)*) as the file type in the dialog that opens. After confirming the dialog, the PLC project is added to the Solution.



Two computers are required to start the sample. Alternatively, the test may be carried out with two runtime systems on a single PC. The constants with the port numbers and the IP addresses of the communication partners have to be modified accordingly.

#### Sample configuration with two computers:

- Device A is located on the local PC and has the IP address '10.1.128.21'
- Device B is located on the remote PC and has the IP address '172.16.6.195'.

#### **Device A**

Please perform the following steps to configure the sample on device A:

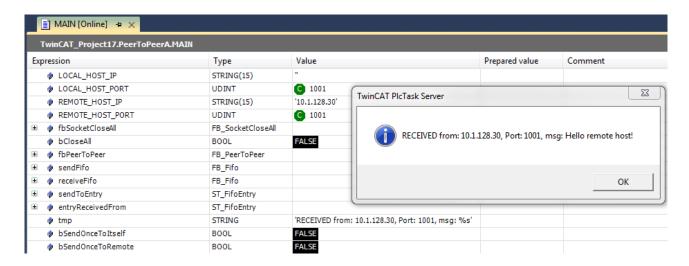
- Create a new TwinCAT 3 solution in TwinCAT XAE and import the Peer-to-Peer PLC project for device A.
- Set the constant REMOTE\_HOST\_IP in POU MAIN to the real IP address of the remote system (device B in our example: '10.1.128.').
- Activate the configuration and start the PLC runtime. (Don't forget to create a license for TF6310 TCP/IP)

#### **Device B**

To install the sample on device B, proceed as follows:

- Create a new TwinCAT 3 Solution in TwinCAT XAE and import the peer-to-peer PLC project for device B.
- Adapt the constant REMOTE\_HOST\_IP in POU MAIN to the actual IP address of device A (in our sample '10.1.128.21').
- Enable the configuration and start the PLC runtime. (Do not forget to generate a license for TF6310 TCP/IP beforehand)
- Log in to the PLC runtime and set the Boolean variable bSendOnceToRemote to TRUE in the POU "MAIN".
- A message box should appear on device A shortly afterwards. You can now repeat this step on device A. The message box should now appear on device B.



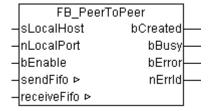


#### 6.2.1.3 PLC devices A and B

The required functionality was encapsulated in the function block FB\_PeerToPeer. Each of the communication partners uses an instance of the FB\_PeerToPeer function block. The block is activated through a rising edge at the bEnable input. A new UDP socket is opened, and data exchange commences. The socket address is specified via the variables sLocalHost and nLocalPort. A falling edge stops the data exchange and closes the socket. The data to be sent are transferred to the block through a reference (VAR\_IN\_OUT) via the variable sendFifo. The data received are stored in the variable receiveFifo.

Name	Default value	Description
g_sTclpConnSvrAddr	"	Network address of the TwinCAT TCP/IP Connection Server. Default: Empty string (the server is located on the local PC);
bLogDebugMessages	TRUE	Activates/deactivates writing of messages into the log book of the operating system;
PLCPRJ_ERROR_SENDFIFO_OV ERFLOW	16#8103	Sample project error code: The send Fifo is full.
PLCPRJ_ERROR_RECFIFO_OVE RFLOW	16#8104	Sample project error code: The receive Fifo is full.

#### FUNCTION\_BLOCK FB\_PeerToPeer



#### Interface

```
VAR IN OUT
    sendFifo
               : FB Fifo;
    receiveFifo : FB Fifo;
END VAR
VAR INPUT
    sLocalHost : STRING(15);
    nLocalPort : UDINT;
    bEnable
                : BOOL;
END VAR
VAR OUTPUT
    bCreated : BOOL;
              : BOOL;
    bBusy
               : BOOL;
: UDINT;
    bError
   nErrId
END 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
```

#### Implementation

```
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;
               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 );
                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;
                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;
            bBusy := FALSE;
            eStep := UDP STATE IDLE;
        END IF
END CASE
```

#### **MAIN** program

Previously opened sockets must be closed after a program download or a PLC reset. During PLC start-up, this is done by calling an instance of the <u>FB SocketCloseAll [ 23]</u> function block. If one of the variables bSendOnceToItself or bSendOnceToRemote has an rising edge, a new Fifo entry is generated and stored in the send Fifo. Received messages are removed from the receive Fifo and displayed in a message box.

```
PROGRAM MAIN

VAR CONSTANT

LOCAL_HOST_IP : STRING(15) := '';

LOCAL_HOST_PORT : UDINT := 1001;
```

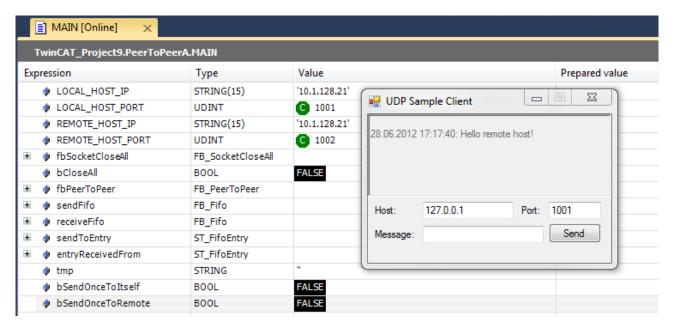


```
REMOTE_HOST_IP : STRING(15) := '172.16.2.209';
REMOTE_HOST_PORT : UDINT := 1001;
END VAR
VAR
    fbSocketCloseAll : FB SocketCloseAll;
    bCloseAll : BOOL := TRUE;
    fbPeerToPeer
                    : FB PeerToPeer;
                   : FB Fifo;
    sendFifo
                 . FB_Fifo;
. ST FifoEntry;
    receiveFifo
    sendToEntry
    entryReceivedFrom : ST_FifoEntry;
            : 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 sendToEntry.nRemotePort
     sendToEntry.sRemoteHost
                                                                   (* remote host IP address *)
     sendToEntry.msg
                                 := 'Hello remote host!';
                                                                      (* message text*);
     sendToEntry.msg
sendFifo.AddTail( new := sendToEntry ); (* add new entry to the check for fifo overflow*)
                                                            (* add new entry to the send queue*)
         LogError( 'Send fifo overflow!', PLCPRJ ERROR SENDFIFO OVERFLOW );
    END IF
    IF bSendOnceToItself THEN
    bSendOnceToItself := FALSE;
sendToEntry.nRemotePort := LOCAL_HOST_PORT;
                                                           (* clear flag *)
                                                                 (* nRemotePort == nLocalPort => sen
d it to itself *)
      sendToEntry.sRemoteHost
                                      := LOCAL HOST IP;
                                                                      (* sRemoteHost == sLocalHost =>
send it to itself *)
                                 := 'Hello itself!';
    sendToEntry.msg
                                                              (* message text*);
     sendFifo.AddTail( new := sendToEntry );
                                                             (* add new entry to the send queue*)
                                                         (* check for fifo overflow*)
     IF NOT sendFifo.bOk THEN
        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, ', ms\overline{g}: 8s');
            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 communication

This sample demonstrates how a suitable .NET communication partner for the PLC peer-to-peer device A can be realized. Only use this sample in conjunction with the PLC project PeerToPeerA.





The .NET Sample Client can be used to send single UPD data packages to a UPD Server, in this case the PLC project PeerToPeerA.

#### **Download**

Download the test client.

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

#### How it works

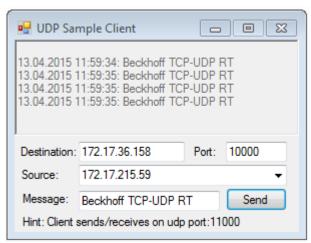
The sample uses the .Net libraries System.Net and System.Net.Sockets to implement a UDP client (class UdpClient). While listening for incoming UDP packets in a background thread, a string can be sent to a remote device by specifying its IP address and port number and clicking the Send button.

For a better understanding of this article, imagine the following setup:

- The PLC project Peer-to-Peer device A is running on a computer with IP address 10.1.128.21
- The .NET application is running on a computer with IP address 10.1.128.30

#### **Description**

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



Together with the PLC / C++ samples, this results in an echo sample:

A UDP message is sent from the client port 11000 to the server port 10000, which sends the same data back to the sender.



The client can be configured via the interface:

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

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

# 6.2.2 Sample02: Multicast

This sample demonstrates how to send and receive Multicast packages via UDP.

Client and Server cyclically send a value to each other via a Multicast IP address.

Client and Server are realized by two PLC applications and delivered within a single TwinCAT 3 solution.

#### **System requirements**

- · TwinCAT 3 Build 3093 or higher
- TwinCAT 3 Function TF6310 TCP/IP version 1.0.64 or higher
- TwinCAT 3 Library Tc2\_Tcplp version 3.2.64.0 or higher
- If one computer is used to execute the sample, e.g. client and server running in two separate PLC runtimes, both PLC runtimes need to run in separate tasks.

#### **Project download**

https://github.com/Beckhoff/TF6310\_Samples/tree/master/PLC/UDP/Sample02

# 7 Appendix

# 7.1 OSI model

The following article contains a brief introduction to the OSI model and describes how it influences our everyday network communication. Please note that this article is not intended to supersede more detailed documentation on the subject, but only to provide an overview.

The OSI (Open Systems Interconnection) model describes the standardization of functions in a communication system via an abstract layer model. Each layer describes certain functions of communication between devices in the network. Each layer only communicates with the layer directly above or below it.

OSI model			
Layer	yer 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	

**Sample:** If you call up the address "http://www.beckhoff.com" with your web browser, the following protocols are used, starting from layer 7: HTTP  $\rightarrow$  TCP  $\rightarrow$  IP  $\rightarrow$  Ethernet. If you enter "https://www.beckhoff.com", however, the protocols HTTP  $\rightarrow$  SSL  $\rightarrow$  TCP  $\rightarrow$  IP  $\rightarrow$  Ethernet would be used.

The TwinCAT 3 function TF6310 TCP/IP can be used to develop network-capable PLC programs that use TCP or UDP as the transport protocol. This allows PLC programmers to implement their own network protocols in the application layer and define their own message structure for communication with remote systems.

# 7.2 KeepAlive configuration

The transmission of TCP KeepAlive messages verifies if an idle TCP connection is still active. Since version 1.0.47 of the TwinCAT TCP/IP Server (TF6310), the KeepAlive configuration of the Windows operating system is used, which can be configured via the following registry keys:

The following documentation is an excerpt of a Microsoft Technet article.

#### 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 <a href="KeepAliveInterval">KeepAliveInterval</a> 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 Error codes

## 7.3.1 Overview of the error codes

Codes (hex)	Codes (dec)	Error source	Description
0x00000000-0x00007800	0-30720	codes [ 101]	TwinCAT system error (including ADS error codes)
0x00008000-0x000080FF	32768-33023	Internal TwinCAT TCP/IP Connection Server error codes [ • 99]	Internal error of the TwinCAT TCP/IP Connection Server
0x80070000-0x8007FFFF	2147942400-2148007935		Win32 system error (including Windows sockets error codes)

#### Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1	PC, CX (x86) or CX (ARM)	Tc2_Tcplp



# 7.3.2 Internal error codes of the TwinCAT TCP/IP Connection Server



Code (hex)	Code (dec)	Symbolic constant	Description
0x0000800	32769	TCPADSERROR_NOMO REENTRIES	No new sockets can be created (for FB_SocketListen and FB_SocketConnect).
0x0000800 2	32770	TCPADSERROR_NOTF OUND	Socket handle is invalid (for FB_SocketReceive, FB_SocketAccept, FB_SocketSend etc.).
0x0000800 3	32771	TCPADSERROR_ALREA DYEXISTS	Is returned when FB_SocketListen is called, if the TcpIp port listener already exists.
0x0000800 4	32772	TCPADSERROR_NOTC ONNECTED	Is returned when FB_SocketReceive is called, if the client socket is no longer connected with the server.
0x0000800 5	32773	TCPADSERROR_NOTLI STENING	Is returned when FB_SocketAccept is called, if an error was registered in the listener socket.
0x0000800 6	32774	TCPADSERROR_HOST_ NOT_FOUND	Returned if the target system is not reachable.
0x0000808 0	32896	TCPADSERROR_TLS_I NVALID_STATE	Returned if FB_TlsSocketAddCa, FB_TlsSocketAddCrl , FB_TlsSocketSetCert or FB_TlsSocketSetPsk are called and a Connect has already been called.
0x0000808 1	32897	TCPADSERROR_TLS_C A_NOTFOUND	Returned if the specified CA certificate was not found.
0x0000808 2	32898	TCPADSERROR_TLS_C ERT_NOTFOUND	Returned if the specified certificate file was not found.
0x0000808 3	32899	TCPADSERROR_TLS_K EY_NOTFOUND	Returned if the specified file with the private key was not found.
0x0000808 4	32900	TCPADSERROR_TLS_C A_INVALID	Returned if the specified CA certificate could not be read or is invalid.
0x0000808 5	32901	TCPADSERROR_TLS_C ERT_INVALID	Returned if the specified certificate file could not be read or is invalid.
0x0000808 6	32902	TCPADSERROR_TLS_K EY_INVALID	Returned if the specified private key could not be read or is invalid.
0x0000808 7	32903	TCPADSERROR_TLS_V ERIFY_FAIL	Returned if the remote terminal could not be verified during the TLS handshake.
0x0000808 8	32904	TCPADSERROR_TLS_S ETUP	Returned if a general error occurred while setting up the TLS connection.
0x0000808 9	32905	TCPADSERROR_TLS_H ANDSHAKE_FAIL	Returned if an error occurred during the TLS handshake. Usually the handshake always works. However, if there are connection problems during the handshake, it may fail.
0x0000808 A	32906	TCPADSERROR_TLS_C IPHER_INVALID	Returned if an invalid cipher suite was specified.
0x0000808 B	32907	TCPADSERROR_TLS_V ERSION_INVALID	Returned if an invalid TLS version was specified.
0x0000808 C	32908	TCPADSERROR_TLS_C RL_INVALID	Returned if the specified Certificate Revocation List (CRL) is invalid.
0x0000808 D	32909	TCPADSERROR_TLS_I NTERNAL_ERROR	Returned if an internal error occurred while setting up the TLS connection.
0x0000808 E	32910	TCPADSERROR_TLS_P SK_SETUP_ERROR	Returned if an error occurred when using a PreSharedKey (PSK) for TLS.
0x0000808 F	32911	TCPADSERROR_TLS_C N_MISMATCH	Returned if the CommonName in the certificate of the remote terminal does not match the host name or IP address used.
0x0000809 0	32912	TCPADSERROR_TLS_C ERT_EXPIRED	Returned when the certificate of the remote terminal has expired.
0x0000809 1	32913	TCPADSERROR_TLS_C ERT_REVOKED	Returned when the certificate of the remote terminal has been revoked.
0x0000809 2	32914	TCPADSERROR_TLS_C ERT_MISSING	Returned when the remote terminal did not submit a certificate.



#### Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1	PC, CX (x86) or CX (ARM)	Tc2_TcpIp

# 7.3.3 Troubleshooting/diagnostics

- In the event of connection problems the PING command can be used to ascertain whether the external communication partner can be reached via the network connection. If this is not the case, check the network configuration and firewall settings.
- Sniffer tools such as Wireshark enable logging of the entire network communication. The log can then be analysed by Beckhoff support staff.
- Check the hardware and software requirements described in this documentation (TwinCAT version, CE image version etc.).
- Check the software installation hints described in this documentation (e.g. installation of CAB files on CE plattform).
- Check the input parameters that are transferred to the function blocks (network address, port number, data etc, connection handle.) for correctness. Check whether the function block issues an error code. The documentation for the error codes can be found here: Overview of error codes [ > 98].
- Check if the other communication partner/software/device issues an error code.
- Activate the debug output integrated in the TcSocketHelper.Lib during connection establishment/ disconnect process (keyword: CONNECT\_MODE\_ENABLEDBG). Open the TwinCAT System Manager and activate the LogView window. Analyze/check the debug output strings.

#### Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1	PC, CX (x86) or CX (ARM)	Tc2_TcpIp

#### 7.3.4 ADS Return Codes

Grouping of error codes:

Global error codes: <u>ADS Return Codes [▶ 101]</u>... (0x9811\_0000 ...) Router error codes: <u>ADS Return Codes [▶ 102]</u>... (0x9811\_0500 ...) General ADS errors: <u>ADS Return Codes [▶ 102]</u>... (0x9811\_0700 ...) RTime error codes: <u>ADS Return Codes [▶ 104]</u>... (0x9811\_1000 ...)

#### Global error codes



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

## **Router error codes**

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

## **General ADS error codes**



Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR DEVICE ERROR	General device error.
0x700	1793	0x98110701	ADSERR DEVICE SRVNOTSUPP	Service is not supported by the server.
0x701	1794	0x98110701	ADSERR DEVICE INVALIDGRP	Invalid index group.
0x702	1795	0x98110703	ADSERR DEVICE INVALIDOFFSET	Invalid index offset.
0x703	1796	0x98110704	ADSERR DEVICE INVALIDACCESS	Reading or writing not permitted.
0x704	1797	0x98110704	ADSERR DEVICE INVALIDACESS  ADSERR DEVICE INVALIDACESS	Parameter size not correct.
0x705	1798	0x98110706	ADSERR DEVICE INVALIDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR DEVICE NOTREADY	Device is not ready to operate.
0x707	1800	0x98110708	ADSERR DEVICE BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR DEVICE INVALIDCONTEXT	Invalid operating system context. This can result
OX 7 GG	1001	0,000110100	ABBERRY_BEVIOL_INVALIDOGNIEXI	from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files,).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.
			<u> </u>	



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

## RTime error codes

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

# Specific positive HRESULT Return Codes:

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

# **TCP Winsock error codes**



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

# 7.4 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

#### **Download finder**

Our <u>download finder</u> contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

#### Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for <u>local support and service</u> on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

#### **Beckhoff Support**

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- · design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline: +49 5246 963-157 e-mail: support@beckhoff.com

#### **Beckhoff Service**

The Beckhoff Service Center supports you in all matters of after-sales service:

- · on-site service
- · repair service
- · spare parts service
- hotline service

Hotline: +49 5246 963-460 e-mail: service@beckhoff.com

#### **Beckhoff Headquarters**

Beckhoff Automation GmbH & Co. KG



Huelshorstweg 20 33415 Verl Germany

Phone: +49 5246 963-0
e-mail: info@beckhoff.com
web: www.beckhoff.com

More Information: www.beckhoff.com/tf6310

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

