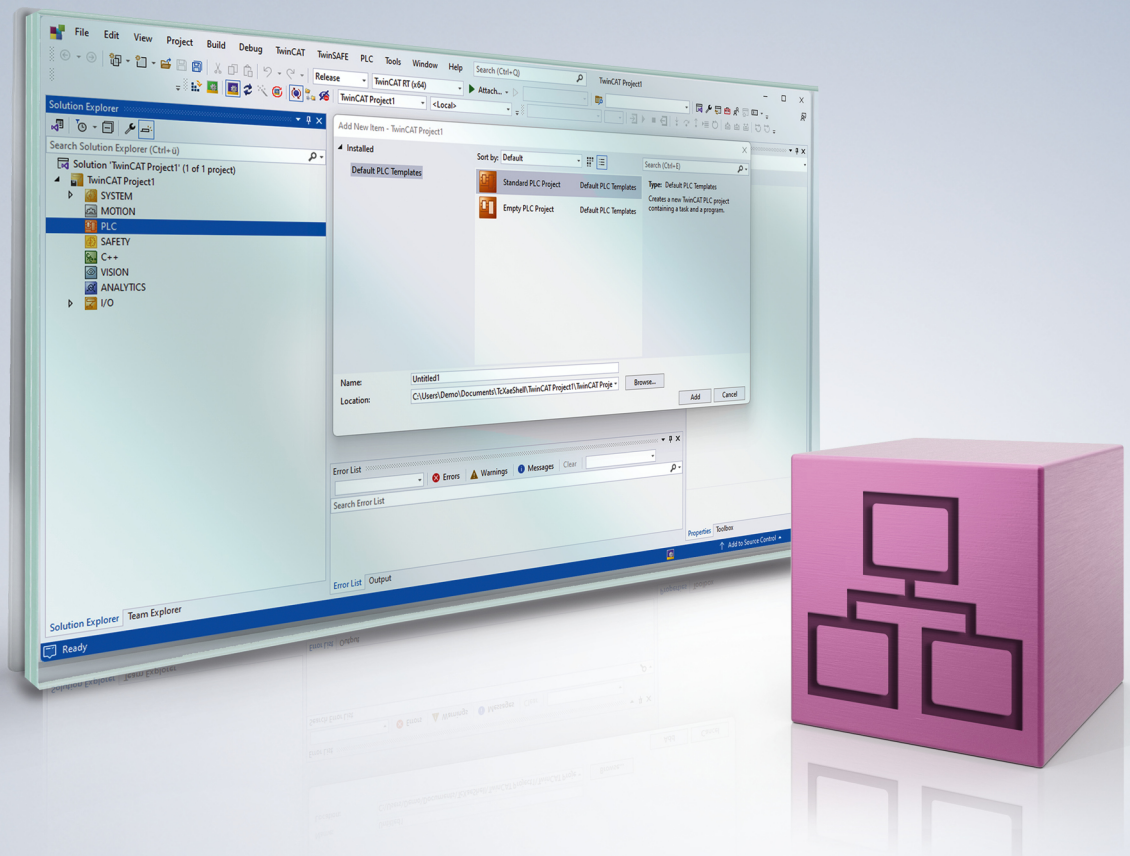
BECKHOFF New Automation Technology

手册 | ZH

TF6310

TwinCAT 3 | TCP/IP



2026-07-17 | 版本： 1.5.2

目录

1 前言

1.1 文档说明

本说明仅适用于熟悉国家标准且经过培训的控制和自动化工程专家。
在安装和调试组件时，必须遵循文档和以下说明及解释。
操作人员应具备相关资质，并始终使用最新的生效文档。

相关负责人员必须确保所述产品的应用或使用符合所有安全要求，包括所有相关法律、法规、准则和标准。

免责声明

本文档经过精心准备。然而，所述产品正在不断开发中。
我们保留随时修改和更改本文档的权利，恕不另行通知。
不得依据本文档中的数据、图表和说明对已供货产品的修改提出赔偿。

商标

Beckhoff®、ATRO®、EtherCAT®、EtherCAT G®、EtherCAT G10®、EtherCAT P®、MX-System®、Safety over EtherCAT®、TC/BSD®、 TwinCAT®、TwinCAT/BSD®、TwinSAFE®、XFC®、XPlanar® 和 XTS®是 Beckhoff Automation GmbH
的注册商标并由其授权使用。本出版物中所使用的其它名称可能是商标名称，任何第三方出于其自身目的使用它们可能会侵犯商标所有者的权利。

EtherCAT®是注册商标和专利技术，由 Beckhoff Automation GmbH 授权使用。

版权所有

第三方商标

本文档可能使用了第三方商标。有关商标信息，可以访问：https://www.beckhoff.com/trademarks。

1.2 安全信息

安全规范

为了确保您的使用安全，请务必仔细阅读
并遵守本文档中每个产品的安全使用说明。

责任免除

所有组件在供货时都配有适合应用的特定硬件和软件配置。严禁未按文档所述修改硬件或软件配置，否则，德国倍福自动化有限公司对由此产生的后果不承担责任。

人员资格

本说明仅供熟悉适用国家标准的控制、自动化和驱动工程专家使用。

警示性词语

文档中使用的警示信号词分类如下。为避免人身伤害和财产损失，请阅读并遵守安全和警告注意事项。

人身伤害警告

⚠ 危险
存在死亡或重伤的高度风险。

⚠ 警告
存在死亡或重伤的中度风险。

⚠ 谨慎
存在可能导致中度或轻度伤害的低度风险。

财产或环境损害警告

注意
可能会损坏环境、设备或数据。

操作产品的信息

这些信息包括：
有关产品的操作、帮助或进一步信息的建议。

1.3 信息安全说明

Beckhoff Automation GmbH & Co.KG (简称 Beckhoff) 的产品，只要可以在线访问，都配备了安全功能，支持工厂、系统、机器和网络的安全运行。尽管配备了安全功能，但为了保护相应的工厂、系统、机器和网络免受网络威胁，必须建立、实施和不断更新整个操作安全概念。Beckhoff 所销售的产品只是整个安全概念的一部分。客户有责任防止第三方未经授权访问其设备、系统、机器和网络。它们只有在采取了适当的保护措施的情况下，方可与公司网络或互联网连接。

此外，还应遵守 Beckhoff 关于采取适当保护措施的建议。关于信息安全和工业安全的更多信息，请访问本公司网站 https://www.beckhoff.com/secguide。

Beckhoff 的产品和解决方案持续进行改进。这也适用于安全功能。鉴于持续进行改进，Beckhoff 明确建议始终保持产品的最新状态，并在产品更新可用后马上进行安装。使用过时的或不支持的产品版本可能会增加网络威胁的风险。

如需了解 Beckhoff 产品信息安全的信息，请订阅 https://www.beckhoff.com/secinfo 上的 RSS 源。

1.4 文档发行状态

版本	修改
1.6.x	**新增：** TwinCAT 3 UserMode Runtime [▶ 11]
1.5.x	**新增：** 快速入门 [▶ 18] 协议 [▶ 19]

2 概述

TwinCAT TCP/IP Connection Server 可以在 TwinCAT PLC 内搭建单个或多个 TCP/IP 服务器/客户端。这样，PLC 程序员就可以直接在 PLC 程序中自主开发应用层（OSI 模型）的网络协议。通信连接可选择通过 TLS 进行加密。

产品组件

TF6310 TCP/IP 产品由以下组件组成，这些组件将随安装程序一并提供：

- **PLC 库：**Tc2_TcpIp 库（实现基本的 TCP/IP 和 UDP/IP 功能）。
- **后台程序：**TwinCAT TCP/IP Connection Server（用于通信的进程）。

2.1 对比 TF6310 TF6311

TF6310 “TCP/IP”和 TF6311 ”TCP/UDP Realtime”产品可提供类似功能。

本页概述了这些产品的异同之处：

	TF 6310	**TF 6311**
TwinCAT	TwinCAT 2/3	TwinCAT 3
客户端/服务器	都有	都有
大型/未知网络	++	+
确定性	+	++
大容量数据传输	++	+
编程语言	PLC	PLC 和 C++
UDP-Mutlicast	是	否
协议	TCP、UDP	TCP、UDP、Arp/Ping
硬件要求	变量	兼容 TwinCAT 的网卡
插槽配置	请参见操作系统 (WinSock)	

由于 TF6311 直接集成在 TwinCAT 系统中，因此不能使用 Windows 防火墙。在大型/未知网络中，我们建议使用 TF6310。

3 安装

3.1 系统要求

为确保 TF6310 TCP/IP 功能正常运行，必须满足以下系统要求。

技术数据	描述
操作系统	Windows 10、11 Windows CE 6/7 Windows Embedded Standard 2009 TwinCAT/BSD
目标平台	PC 架构（x86、x64、Arm®）
TwinCAT 版本	TwinCAT 2、TwinCAT 3
TwinCAT 安装级别	TwinCAT 2 CP、PLC、NC-PTP TwinCAT 3 XAE、XAR、ADS
所需 TwinCAT 授权	TS6310（适用于 TwinCAT 2） TF6310（适用于 TwinCAT 3）

支持 TLS

请注意，TLS 功能块在 Windows CE 下不可用。

3.2 安装

安装 (TwinCAT 3.1 Build 4024)

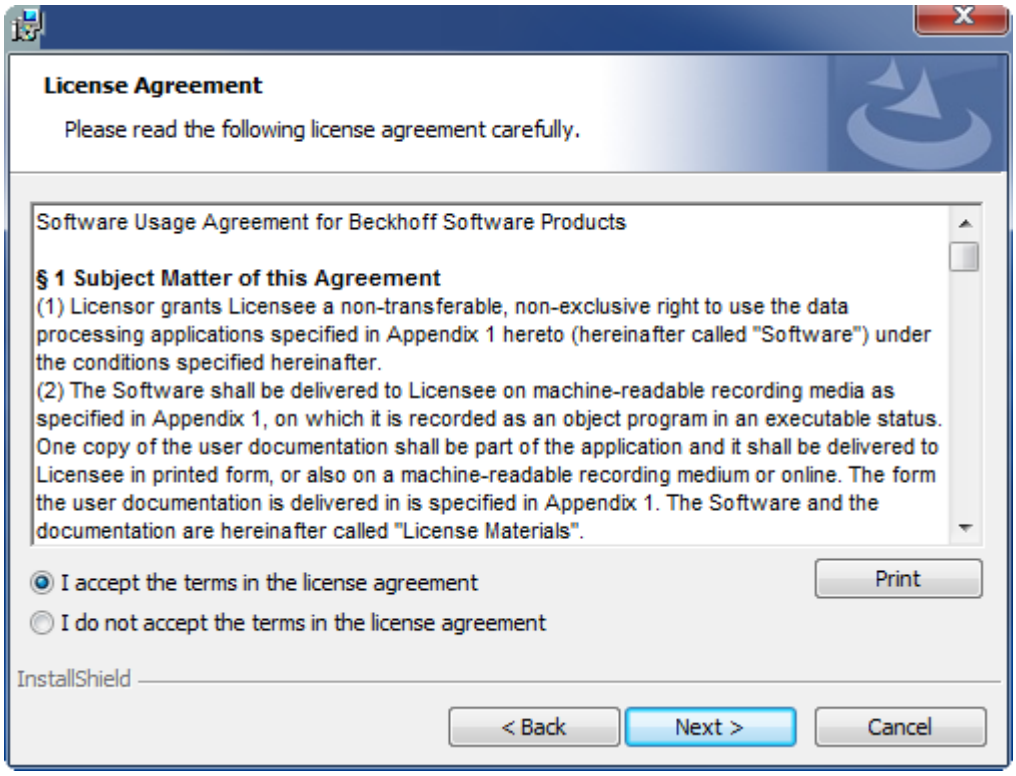
下文将介绍如何在基于 Windows 的操作系统上安装 TwinCAT 3 功能组件。

✓ 从倍福官网下载 TwinCAT 3 功能组件的安装文件。

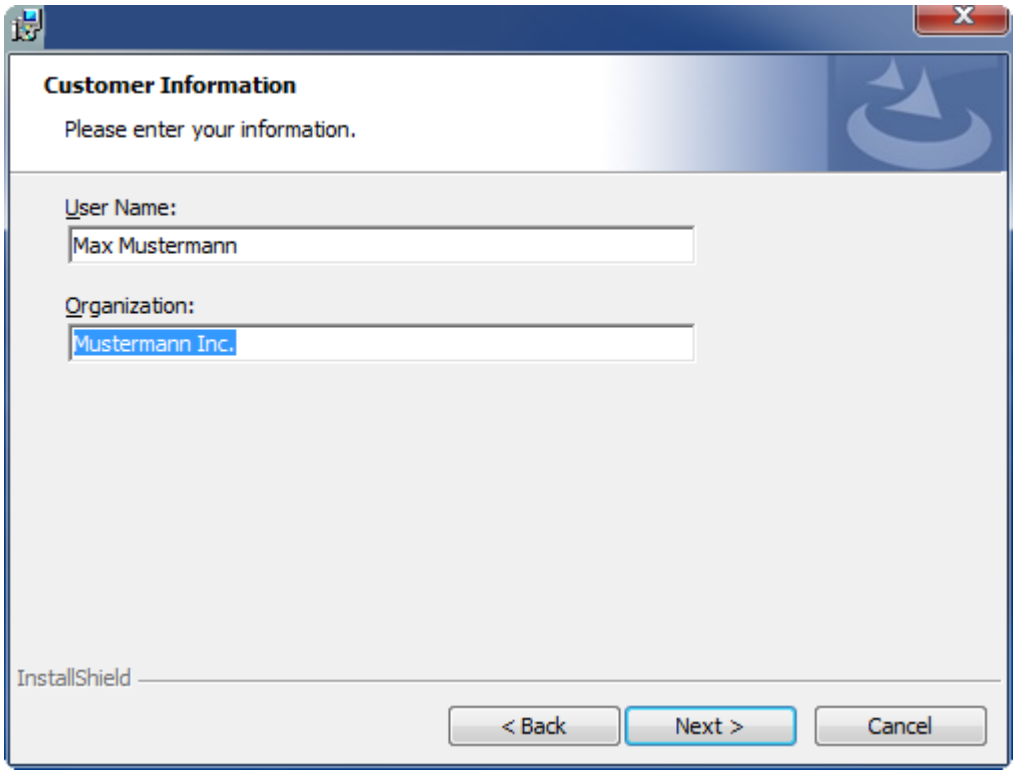
1. 以管理员身份运行安装文件。为此，请在该文件的上下文菜单中选择“**Run As Admin（以管理员身份运行）**”命令。

   ⇨ 安装对话框打开。

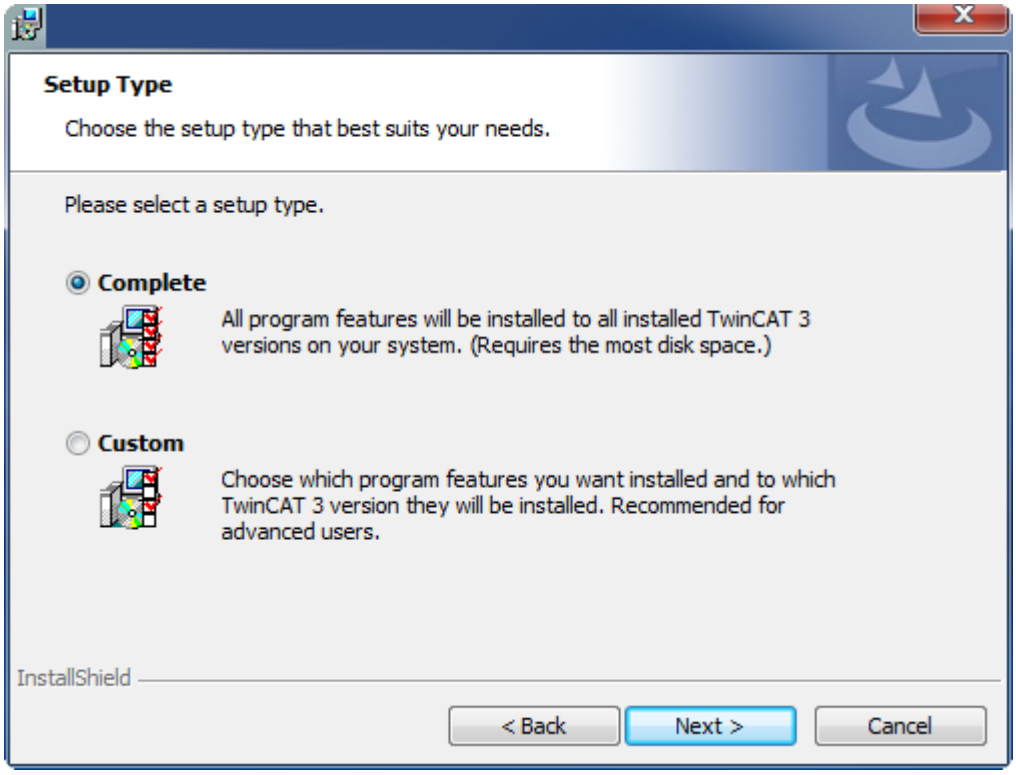
2. 接受最终用户许可协议，然后点击“**Next（下一步）**”。



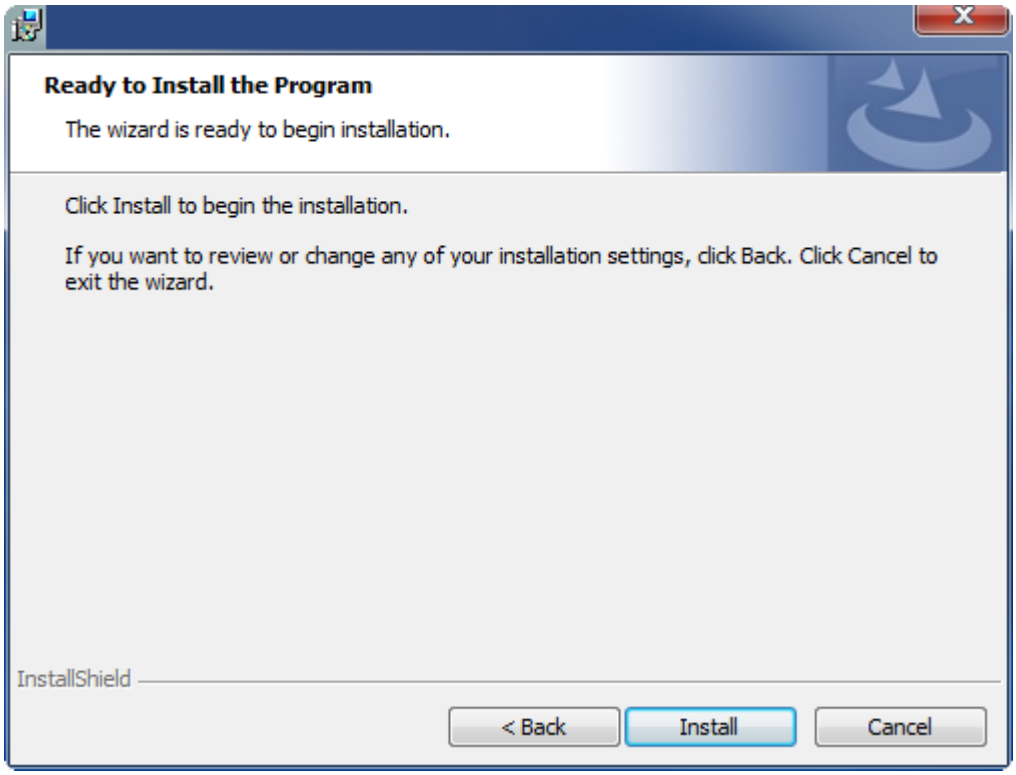
3. 输入用户数据。



4. 若要安装完整版 TwinCAT 3 功能组件，请选择“**Complete（完整版）**”作为安装类型。若要单独安装 TwinCAT 3 功能组件，请选择“**Custom（自定义）**”。

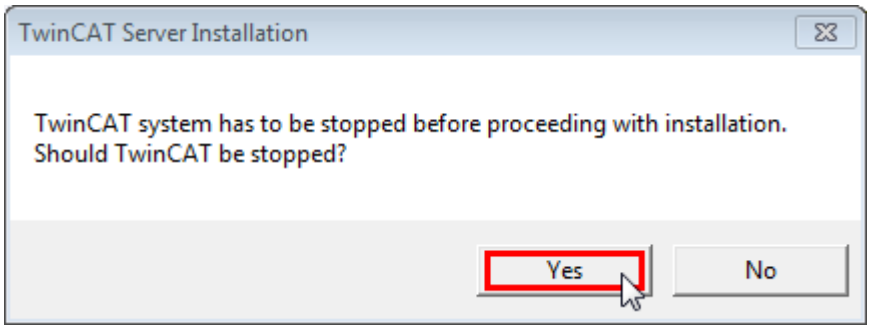


5. 点击“**Next（下一步）**”，然后点击“**Install（安装）**”开始进行安装。

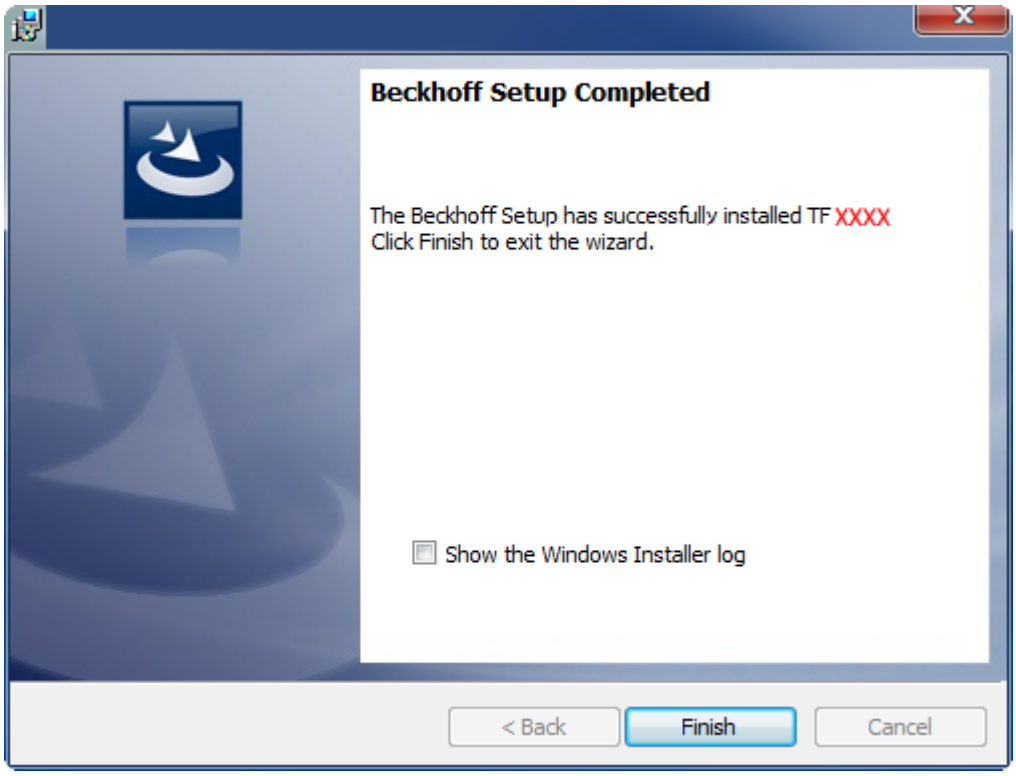


⇨ 将显示一个对话框，提示您必须关闭 TwinCAT 系统才能继续安装。

6. 点击“**Yes（是）**”进行确认。



7. 点击“**Finish（完成）**”退出安装。



⇨ TwinCAT 3 功能组件安装成功。

3.3 TwinCAT 4026 的安装操作

TwinCAT Package Manager

如果在 Microsoft Windows 操作系统上使用 TwinCAT 3.1 Build 4026（及更高版本），则可以通过 TwinCAT Package Manager 安装，请参见安装文档。

通常情况下，可通过相应的工作量安装此功能；然而，您也可以单独安装工作量中包含的软件包。本文档简要介绍了通过工作负载进行安装的过程。

命令行程序 TcPkg

您可以使用 TcPkg **C**ommand **L**ine **I**nterface（命令行界面，CLI）来显示系统上的可用工作负载：

```
tcpkg list TF6310
```

您可以使用以下命令安装 TF6310 TwinCAT 3 TCP/IP 功能组件的工作负载。

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

TwinCAT Package Manager UI

您可以使用 **U**ser **I**nterface（用户界面，UI）显示所有可用的工作负载，并根据需要进行安装。
为此，请按照界面中的相应说明操作。

TwinCAT 3 Usermode Runtime

本产品支持安装在 TwinCAT 3 Usermode Runtime 上。在此类环境中安装时，会自动部署相应的 UM 软件包。

UM 软件包安装了一个配置文件，用于说明 TwinCAT Usermode 系统服务在启动阶段会启动哪些应用程序。默认情况下，该配置文件位于 TwinCAT Usermode Runtime 目录中：

C:\ProgramData\Beckhoff\TwinCAT\3.1\Runtimes\UmRT_Default\3.1\Target\StartManConfig

每个产品都会提供各自的配置文件，因此在该目录中发现多个配置文件是正常现象。每个配置文件都会为其相关的 TwinCAT 3 功能组件定义与启动相关的信息，如：

- 要启动的可执行文件的安装路径
- 启动 TwinCAT 3 功能组件所需的参数

TwinCAT Usermode 系统服务会将这些条目用于确定在运行时初始化过程中启动哪些应用程序以及何时执行这些应用程序。

3.4 Windows CE 的安装操作

本节将介绍如何在基于 Windows CE 的 Beckhoff Embedded PC Controller 上安装 TwinCAT 3 功能组件 TF6310 TCP/IP。

安装过程包括 4 个步骤：

- 下载安装文件 [▶ 12]
- 在主机上安装 [▶ 12]
- 将可执行文件传输到 Windows CE 设备上 [▶ 13]
- 软件安装 [▶ 13]

本节最后一段将介绍软件升级 [▶ 13]。

下载安装文件

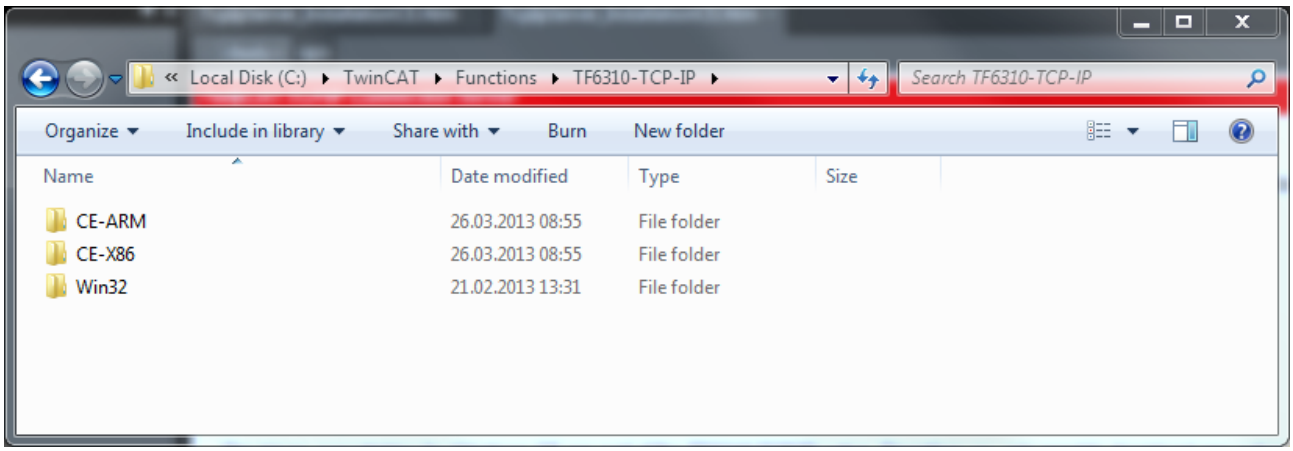
Windows CE 的 CAB 安装文件是 TF6310 TCP/IP 安装程序的一部分。因此，您只需从 www.beckhoff.com 获取相应的安装程序，该程序自动包含 Windows 和 Windows CE（x86 和 Arm®）的所有版本。

TF6310 TCP/IP 安装程序的安装说明整合在我们的常规安装说明中（见 安装 [▶ 8]）。

在主机上安装

安装完成后，安装文件夹包含 3 个目录（每个硬件平台 1 个目录）：

- **CE-Arm®：**基于 Arm® 的嵌入式控制器，可在 Windows CE 上运行，例如 CX8090、CX9020
- **CE-X86：**基于 X86 的嵌入式控制器，可在 Windows CE 上运行，例如 CX50xx。CX20x0
- **Win32：**可在 Windows 上运行的嵌入式控制器。



CE-Arm® 和 CE-X86 目录包含与 Windows CE 设备各自硬件平台相关的 Windows CE TF6310 功能组件的 CAB 文件。必须将文件复制到 Windows CE 设备上。

将可执行文件传输到 Windows CE 设备上

将相应的可执行文件传输到 Windows CE 设备上。可以通过以下任一方式实现：

- 通过 Shared Folder（共享文件夹）
- 通过集成式 FTP 服务器
- 通过 ActiveSync
- 通过 CF 卡

更多详细信息，请查阅 Beckhoff Information System 中的“Windows CE”部分。

软件安装

通过以上任一方式传输文件后，执行该文件并点击 **OK（确定）**确认以下对话框。安装完成后，重启 Windows CE 设备。

重启完成后，TF6310 的可执行文件将在后台自动启动。

软件安装在 CE 设备的以下目录中：

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

升级说明

如果 Windows CE 设备上已经安装了 TF6310 版本，则需要在 Windows CE 设备上执行以下操作升级至更高版本：

1. 点击 **Start > Run（开始 > 运行）**，输入“Explorer”（资源管理器），打开 CE Explorer（CE 资源管理器）。
2. 导航至 \HardDisk\TwinCAT\Functions\TF6310-TCP-IP\Server。
3. 将 TcpIpServer.exe 重命名为 TcpIpServer.old。
4. 重启 Windows CE 设备。
5. 将新的 CAB 文件传输至 CE 设备。
6. 执行 CAB 文件并安装新版本。
7. 删除 TcpIpServer.old。
8. 重启 Windows CE 设备。

⇨ 重启完成后，新版本生效。

3.5 授权

TwinCAT 3 功能可以作为完整版或 7 天测试版激活。这两种许可证类型都可以通过 TwinCAT 3 开发环境（XAE）激活。

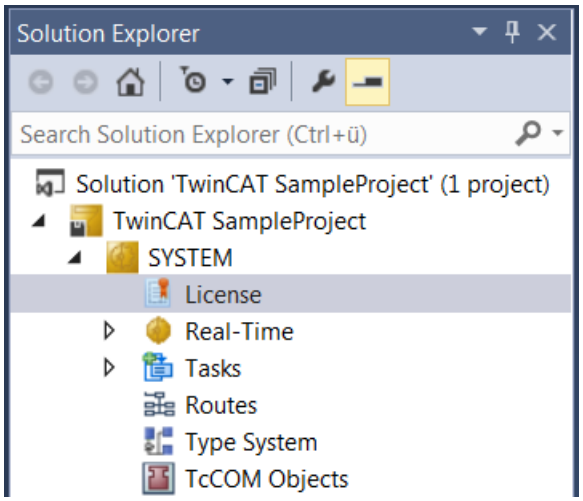
授权使用 TwinCAT 3 功能的完整版本

关于许可证完整版本的程序描述，可参见倍福信息系统的文档“TwinCAT 3 授权”。

授权一个 TwinCAT 3 功能的 7 天测试版本

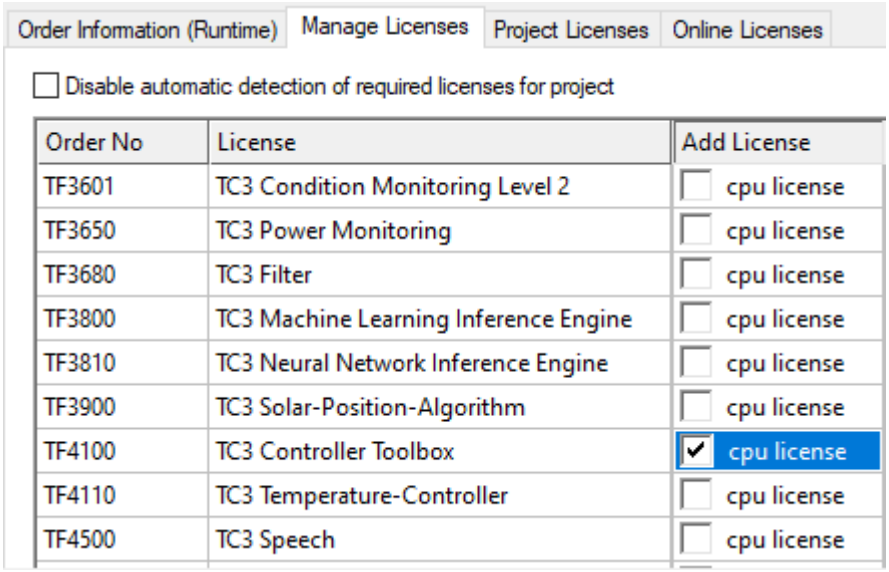
对于TwinCAT 3 许可证加密狗无法启用 7 天测试版本。

1. 启动 TwinCAT 3 开发环境（XAE）。
2. 打开一个现有的 TwinCAT 3 项目或创建一个新项目。
3. 如果想为一个远程设备激活授权，请设置所需的目标系统。为此，从工具栏的**选择目标系统**下拉列表中选择目标系统。
   ⇨ 授权设置总是指选定的目标系统。在目标系统上激活项目时，自动将相应的 TwinCAT 3 许可证复制到该系统中。
4. 在**解决方案资源管理器**中，双击**系统**子目录的**许可证**。

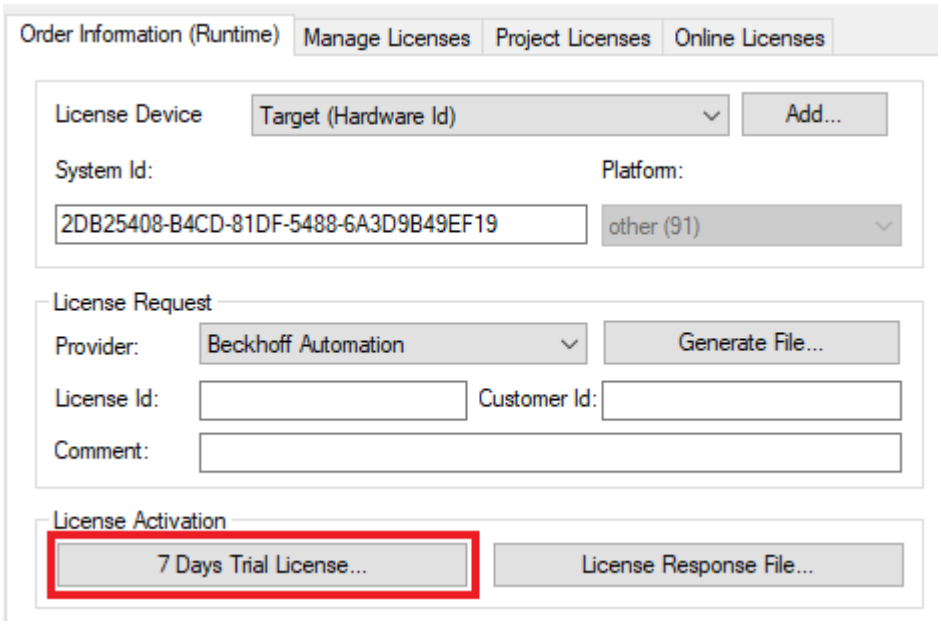


⇨ TwinCAT 3 许可证管理器打开。

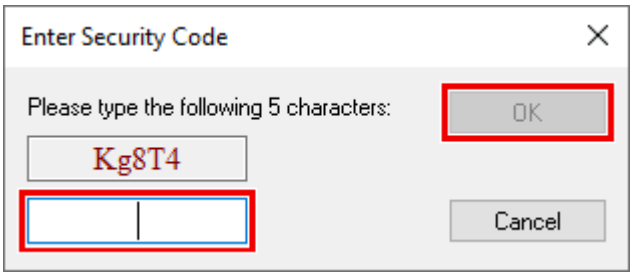
5. 打开**管理许可证**选项卡。在**添加许可证**栏中，选中希望添加到项目的许可证的复选框（例如“TF4100 TC3 控制器工具箱”）。



6. 打开**订单信息（运行时间）**选项卡。
   ⇨ 在许可证的表格概览中，以前选择的许可证显示为“缺失”状态。
7. 点击**7 天试用许可证...**，以激活 7 天试用许可证。



   ⇨ 一个对话框打开，提示输入对话框中显示的安全代码。



8. 准确输入显示的代码并确认输入。
9. 确认随后的对话框，表明激活成功。
   ⇨ 在许可证的表格概览中，许可证状态现在显示许可证的到期日。
10. 重新启动 TwinCAT 系统。

⇨ 7 天试用版启用。

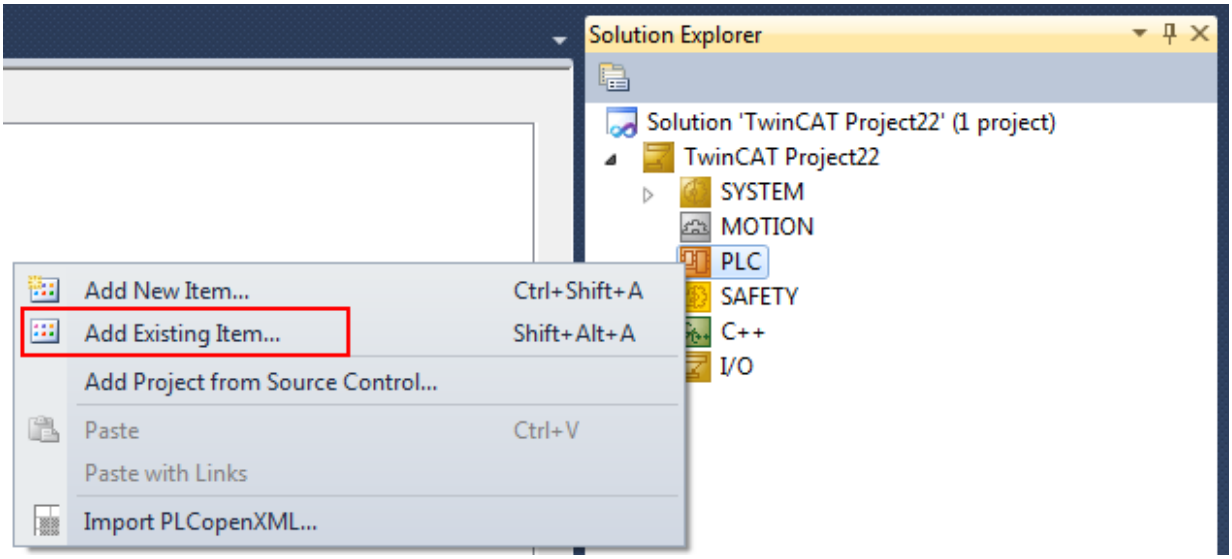
3.6 从 TwinCAT 2 迁移

若要迁移现有 TwinCAT 2 PLC 项目，且该项目调用了 TCP/IP 服务器中的一个 PLC 库，则需执行一些手动步骤，以确保 TwinCAT 3 PLC 转换器能够处理 TwinCAT 2 项目文件 (*.pro)。在 TwinCAT 2 中，功能 TCP/IP 服务器随附 3 个 PLC 库：

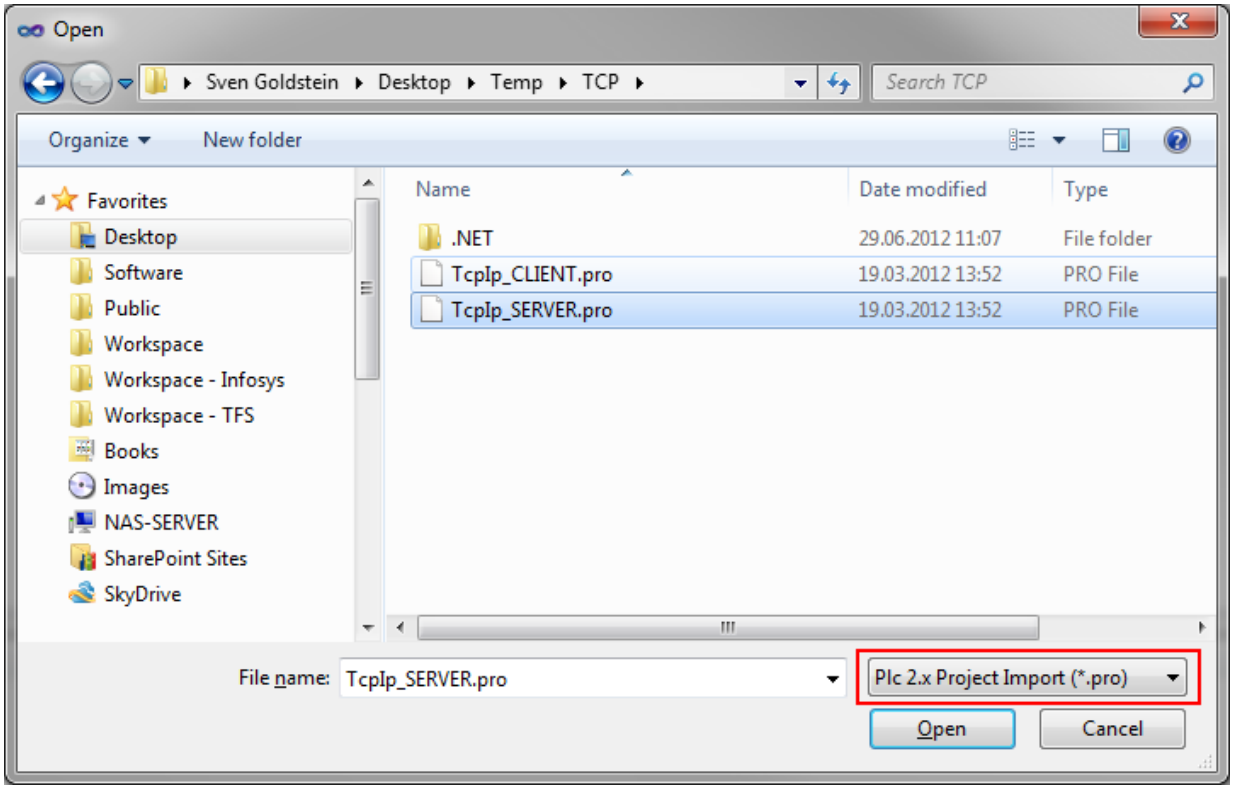
- TcpIp.lib
- TcSocketHelper.lib
- TcSnmp.lib

默认情况下，这些库文件安装在 C:\TwinCAT\Plc\Lib\ 中。根据 PLC 项目所使用的库，您需要将相应的库文件复制到C:\TwinCAT3\Components\Plc\Converter\Lib，然后执行以下步骤：

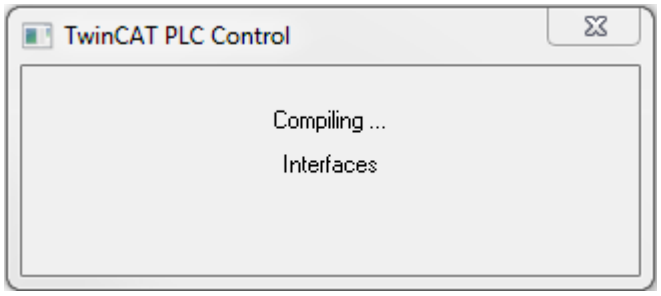
1. 打开 TwinCAT Engineering。
2. 创建新的 TwinCAT 3 解决方案。
3. 右键单击“PLC”节点，在打开的上下文菜单中选择 **Add Existing Item（添加现有项）**。

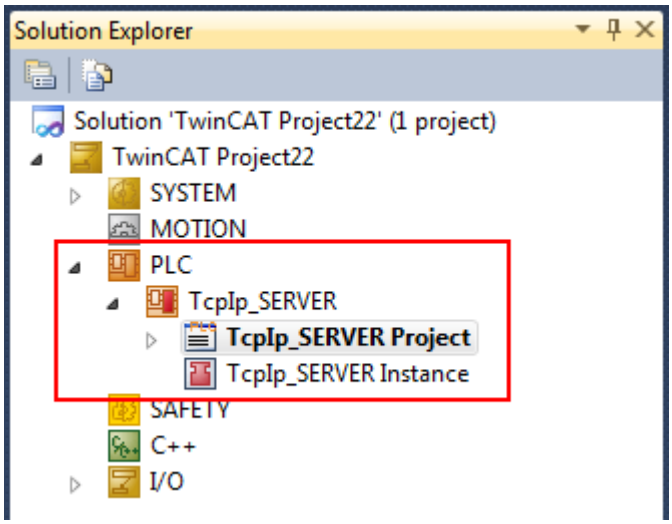


4. 在 Open（打开）对话框中，选择文件类型“Plc 2.x Project Import（PLC 2.x 项目导入）(*.pro)”，定位到包含 TwinCAT 2 PLC 项目的文件夹并选择 corresponding.pro 文件，然后点击 **Open（打开）**。



⇨ TwinCAT 3 启动转换过程，最后在“PLC”节点下显示转换后的 PLC 项目。





4 技术简介

4.1 快速入门

本章将简要介绍 TwinCAT TCP/IP 产品。本操作说明基于我们示例库 [▶ 57]中对应的下载例程，可作为完整的 TwinCAT 项目进行下载。下文将详细介绍应用程序的各个组件。

TwinCAT 项目搭建了 TCP/IP 客户端/服务器应用程序，该程序会以客户端身份向服务器发送消息，并相应地接收相同的消息返回。TwinCAT 项目可直接激活并立即开始执行程序。客户端与服务器之间通过本地主机进行通信。

MAIN

在 MAIN 程序中，首先要为客户端和服务器进行相应的变量声明。客户端和服务器由 2 个功能块封装，其使用 Tc2_TcpIp 库中的相应功能块来调用套接字函数（Send（发送）、Receive（接收）、Listen（监听）、Connect（连接）等）。

为客户端声明以下变量：

```
fbTcpClient1: FB_TCPClient;
sClient1SendData : STRING(255);
nServer1Port : UDINT := 12000;
sServer1Host : T_IPv4Addr := '';
bStartClient1Communication : BOOL := TRUE;
tClient1CycleTime : TIME := T#0.5S;
bClient1SendTrigger : BOOL;
bClient1Connected : BOOL;
sClient1ReceivedData : STRING(255);
bClient1Busy : BOOL;
bClientError1 : BOOL;
nClient1ErrorID1 : UDINT;
```

为服务器声明以下变量：

```
fbTcpEchoServer1 : FB_TCPServer;
sServerReceivedData : STRING(255);
bStartServer1Communication : BOOL := TRUE;
bServer1Connected : BOOL;
sServerData : STRING(255);
bServer1Busy : BOOL;
bServer1Error : BOOL;
nServer1ErrorID : UDINT;
```

待发送的消息保存在变量 sClient1SendData 中。在程序后续运行中，该变量的值发生周期性改变。具体来说，通过字符串拼接操作将计数器数值追加到变量中。

```
IF bAutogenerateData AND bClient1Connected THEN
  fbTimer1(IN:= NOT fbTimer1.Q, PT:= tClient1CycleTime, Q=> , ET=> );
  IF fbTimer1.Q THEN
    sClient1SendData := Concat('TestString No.', UDINT_TO_STRING(nCnt));
    fbTcpClient1.bSendTrigger := TRUE;
    nCnt := nCnt + 1;
  ELSE
    fbTcpClient1.bSendTrigger := FALSE;
  END_IF
END_IF
```

设置并发送该变量值的周期由定时器功能块 fbTimer1 定义。通过设置变量 fbTcpClient1.bSendTrigger 启动发送过程。

服务器应用程序由功能块实例 fbTcpEchoServer1 表示。该功能在程序流程中周期性触发，以便通过服务器的状态机接收消息。

```
fbTcpEchoServer1(
  sLocalHost := '',
  nLocalPort := nServer1Port,
  bStartCommunication := bStartServer1Communication,
  bConnected => bServer1Connected,
  sSendData => sServerData,
  bBusy=> bServer1Busy,
  bError=> bServer1Error,
  nErrorID=> nServer1ErrorID);
```

服务器的 TCP 端口通过 nLocalPort 输入定义。客户端与该端口相连，以便与服务器交换数据。如果要修改该示例，使客户端与服务器之间通过网络进行通信，请确保在系统防火墙中已打开服务器端口。

关闭套接字

TwinCAT 项目重启时，变量 bInit 会导致所有已激活的套接字连接关闭。此操作在程序运行之初执行。

FB_TCPServer

此功能块封装了服务器应用程序，并使用 Tc2_TcpIp 库中的功能块为服务器建立套接字连接、监听传入的消息并发回相应的响应。具体来说，功能块 FB_SocketAccept [▶ 25]、FB_SocketListen [▶ 24]、FB_SocketReceive [▶ 27]、FB_SocketSend [▶ 26]、FB_SocketClose [▶ 22] 和 FB_SocketCloseAll [▶ 23] 用于此用途。

功能块的内部状态机基于以下步骤：

状态	描述
0	初始状态。该过程通过变量 bStartCommunication 启动。
10	在这种状态下，启动套接字监听器，即服务器应用程序与定义的 TCP 端口相连。
20	在这种状态下，会接受传入的套接字连接。
30	在这种状态下，会接收来自已连接客户端的消息。
35	在这种状态下，会向已连接客户端返回消息。
40–42	在这些状态下，套接字连接会关闭。

FB_TCPClient

该功能块封装了客户端应用程序，并使用 Tc2_TcpIp 库中的功能块建立与服务器的连接、向服务器发送消息并接收相应的响应。具体来说，功能块 FB_SocketConnect [▶ 21]、FB_SocketSend [▶ 26]、FB_SocketReceive [▶ 27] 和 FB_SocketClose [▶ 22] 可用于此用途。

功能块的内部状态机基于以下步骤：

状态	描述
0	初始状态。该过程通过变量 bStartCommunication 启动。
10	在这种状态下，会与服务器建立连接。
15	在这种状态下，会向服务器发送消息并处理响应。
20	在这种状态下，会关闭与服务器的连接。

4.2 协议

本节将概述 TCP 和 UDP 传输协议，并提供集成协议所需的相应 PLC 库的链接。这两种传输协议均属于 Internet Protocol Suite（互联网协议群），因此，对我们的日常通信非常重要（如通过互联网）。

传输控制协议 (Transmission Control Protocol, TCP)

TCP 协议是一种面向连接的传输协议（OSI 第 4 层），相当于电话连接，其中呼叫方必须首先建立连接，然后才能传输数据。数据流（字节）可以根据请求通过 TCP 进行可靠传输，因此 TCP 也被称为“面向数据流的传输协议”。 当客户端或服务器发送的数据需要对方应答确认时，网络中采用 TCP 协议。TCP 协议非常适合传输大量数据或没有确定开始/结束标识符的数据流。对发送方而言，这并不是问题，因为他知道传输了多少个数据字节。然而，接收方却无法检测到数据流中消息的结束位置，以及下一条数据流的起始位置。接收方的读取调用仅提供当前接收缓冲区中的数据（数量可能少于或多于另一台设备发送的数据块）。发送方必须指定一种接收方已知并可解读的消息结构。在简单的情况下，消息结构可能由数据和结束控制字符（如回车符）组成。结束控制字符表示消息结束。一种可能的消息结构通常用于传输长度可变的二进制数据，可定义如下：在第一组数据字节中输入特殊控制字符（即所谓的起始分隔符）和后续数据的数据长度。这使得接收方能检测到消息的起始和结束。

TCP/IP 客户端

在 PLC 中实现 TCP/IP 客户端至少需要使用以下功能块：

- 功能块 FB_SocketConnect [▶ 21] 和 FB_SocketClose [▶ 22] 的实例，用于建立和终止与远程服务器的连接（提示：功能块 FB_ClientServerConnection [▶ 41] 整合了这两个功能块的功能）。
- 功能块 FB_SocketSend [▶ 26] 和/或 FB_SocketReceive [▶ 27] 的实例，用于与远程服务器进行数据交换（发送和接收）。

TCP/IP 服务器

在 PLC 中实现 TCP/IP 服务器至少需要使用以下功能块：

- FB_SocketListen [▶ 24] 功能块的实例用于开启监听套接字：
- 功能块 FB_SocketAccept [▶ 25] 和 FB_SocketClose [▶ 22] 的实例（提示：FB_ServerClientConnection [▶ 43] 整合了全部 3 个功能块的功能），用于建立和终止与远程客户端的连接：
- 为了与远程客户端进行数据交换（发送和接收），需要使用功能块 FB_SocketSend [▶ 26] 和/或 FB_SocketReceive [▶ 27] 的实例：
- 在每个用于打开套接字的 PLC Runtime 系统中，都有功能块 FB_SocketCloseAll [▶ 23] 的实例：

功能块 FB_SocketAccept [▶ 25] 和 FB_SocketReceive [▶ 27] 的实例会进行周期性调用（轮询），所有其他功能块则按需调用。

用户数据报协议 (User Datagram Protocol, UDP)

UDP 是一种无连接的协议，即在网络设备之间发送数据时不需要明确的连接。UDP 使用简单的传输模型，未隐式定义用于握手、可靠性、数据排序或拥塞控制的工作流。即使从上述描述来看，UDP 数据报会乱序到达或重复到达，或造成数据线路拥塞，但在某些情况下，特别是在实时通信中，该协议比 TCP 更受青睐，因为 TCP 需要更强的算力，因而需要更多的时间。UDP 协议具有无连接的特性，非常适合发送少量数据。UDP 是一种“面向数据包/面向消息的传输协议”，即接收方收到的发送数据块是完整的数据块。

最简实现 UDP 客户端/服务器需要使用以下功能块：

- 要打开和关闭 UDP 套接字，需要使用功能块 FB_SocketUdpCreate [▶ 28] 和 FB_SocketClose [▶ 22] 的实例（提示：FB_ConnectionlessSocket [▶ 45] 整合了这两个功能块的功能）：
- 功能块 FB_SocketUdpSendTo [▶ 29] 和/或 FB_SocketUdpReceiveFrom [▶ 31] 的实例，用于与其他设备进行数据交换（发送和接收）：
- 在每个用于打开 UDP 套接字的 PLC 运行时系统中，都有功能块 FB_SocketCloseAll [▶ 23] 的实例：

功能块 FB_SocketUdpReceiveFrom [▶ 31] 的实例会进行周期性调用（轮询），所有其他实例则按需调用。

另请参见：示例 [▶ 57]

5 PLC API

5.1 功能块

5.1.1 FB_SocketConnect

利用功能块 FB_SocketConnect，本地客户端可以通过 TwinCAT TCP/IP Connection Server 与远程服务器建立新的 TCP/IP 连接。如果连接成功，则会打开新的套接字，并在 hSocket 输出端返回相应的连接句柄。例如，功能块 FB_SocketSend [▶ 26] 和 FB_SocketReceive [▶ 27] 都需要使用连接句柄，以便与远程服务器交换数据。如果不再需要连接，则可使用功能块 FB_SocketClose [▶ 22] 关闭该连接。多个客户端可以同时与远程服务器建立连接。对于每个新的客户端，都会打开新的套接字，并返回新的连接句柄。TwinCAT TCP/IP Connection Server 会自动为每个客户端分配新的 IP 端口号。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机，可以指定“127.0.0.1”。
sRemoteHost	T_IPv4Addr	采用字符串形式的远程服务器 IP 地址 (Ipv4)（如“172.33.5.1”）。对于服务器，可以在本地计算机上输入空字符串。
nRemotePort	UDINT	远程服务器的 IP 端口号（如 200）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

设置功能块的最长执行时间

不要将“tTimeout”值设得过低，因为如果网络中断，超时时间可能大于 30 s。如果该值过低，则会提前中断执行命令，并返回 ADS 错误代码 1861（超时时间已过），而非 Winsocket 错误 WSAETIMEDOUT。

输出

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

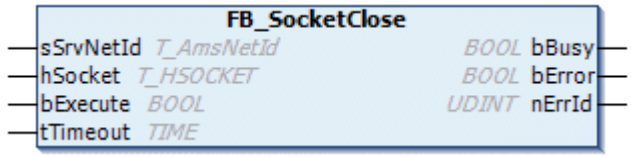
名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBusy 输出被重置后，将会设置该输出。

名称	类型	描述
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新打开的本地客户端套接字的 TCP/IP 连接句柄 [▶ 54]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.2 FB_SocketClose



功能块 FB_SocketClose 可用于关闭打开的 TCP/IP 或 UDP 套接字。

TCP/IP：监听套接字通过功能块 FB_SocketListen [▶ 24] 打开，本地客户端套接字通过 FB_SocketConnect [▶ 21] 打开，远程客户端套接字通过 FB_SocketAccept [▶ 25] 打开。

UDP：通过功能块 FB_SocketUdpCreate [▶ 28] 打开 UDP 套接字。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	• TCP/IP：要关闭的监听器、远程客户端或本地客户端套接字的连接句柄 [▶ 54]。 • UDP：UDP 套接字的连接句柄。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

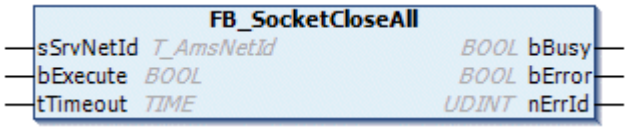
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.3 FB_SocketCloseAll



如果重新启动或停止 TwinCAT，TwinCAT TCP/IP Connection Server 也会停止。打开的套接字（TCP/IP 和 UDP 连接句柄）都会自动关闭。执行“PLC reset（PLC 复位）”、“Rebuild all...（全部重建）”或重新“Download（下载）”后，会重置 PLC 程序，此时 PLC 将无法再获取已打开的套接字（连接句柄）的相关信息。 此后，任何已建立的连接都将无法正常关闭。

功能块 FB_SocketCloseAll 可用于关闭 PLC Runtime 系统打开的所有连接句柄（TCP/IP 和 UDP 套接字）。这意味着，如果在第一个 Runtime 系统（端口 801）的某个任务中调用了 FB_SocketCloseAll，那么该 Runtime 系统中所有已打开的套接字都会关闭。在每个使用套接字功能块的 PLC Runtime 系统中，应在 PLC 启动期间调用 FB_SocketCloseAll 的实例。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

ST 语言实施示例

以下程序代码用于在 PLC 重启前，正确关闭在“PLC reset（PLC 复位）”或“Download（下载）”之前已打开的连接句柄（套接字）。

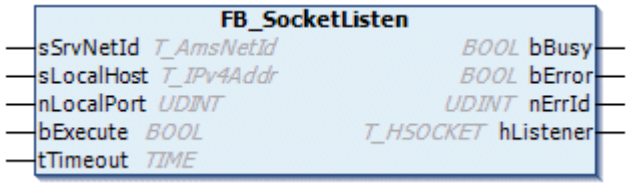
```
PROGRAM MAIN
VAR
    fbSocketCloseAll : FB_SocketCloseAll;
    bCloseAll        : BOOL := TRUE;
END_VAR
IF bCloseAll THEN(*On PLC reset or program download close all old connections *)
    bCloseAll := FALSE;
    fbSocketCloseAll( sSrvNetId:= '', bExecute:= TRUE, tTimeout:= T#10s );
ELSE
    fbSocketCloseAll( bExecute:= FALSE );
END_IF
```

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

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.4 FB_SocketListen



使用功能块 FB_SocketListen，可以通过 TwinCAT TCP/IP Connection Server 打开新的监听套接字。通过监听套接字，TwinCAT TCP/IP Connection Server 可以“监听”远程客户端发出的连接请求。如果连接成功，则会在 hListner 输出端返回相应的连接句柄。功能块 FB_SocketAccept [▶ 25] 需要使用该句柄。如果不再需要监听套接字，可使用功能块 FB_SocketClose [▶ 22] 关闭该套接字。每台计算机上的监听套接字必须有唯一的 IP 端口号。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
sLocalHost	T_IPv4Addr	采用字符串形式的本地服务器 IP 地址 (Ipv4)（如“172.13.15.2”）。“0.0.0.0”可用于列出所有可用的网络适配器。
nLocalPort	UDINT	本地服务器 IP 端口（如 200）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

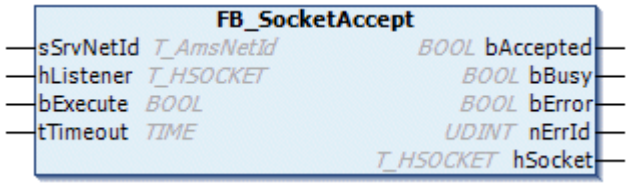
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hListener   : T_HSOCKET;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBusy 输出被重置后，将会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hListener	T_HSOCKET	新的监听套接字的连接句柄 [▶ 54]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.5 FB_SocketAccept



发送到 TwinCAT TCP/IP Connection Server 的远程客户端连接请求必须予以确认（接受）。功能块 FB_SocketAccept 会接受传入的远程客户端连接请求，打开新的远程客户端套接字，并返回相关连接句柄。例如，功能块 FB_SocketSend [▶ 26] 和 FB_SocketReceive [▶ 27] 需要使用连接句柄，以便与远程客户端交换数据。首先必须接受所有传入的连接请求。如果不再需要或不需要某个连接，可以使用功能块 FB_SocketClose [▶ 22] 将其关闭。

服务器实现需要至少 1 个该功能块的实例。必须从 PLC 任务周期性地调用（轮询）该实例。功能块可通过 bExecute 输入端的上升沿激活（例如每 5 秒一次）。

如果成功，则会设置 bAccepted 输出，并在 hSocket 输出端返回新的远程客户端的连接句柄。如果没有新的远程客户端连接请求，则不会返回错误。多个远程客户端可以同时与服务器建立连接。多个远程客户端的连接句柄可通过多个功能块调用依次获取。远程客户端的每个连接句柄只能获取一次。建议将连接句柄保存在列表（数组）中。新的连接会添加到列表中，关闭的连接必须从列表中删除。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hListener	T_HSOCKET	监听套接字的连接句柄 [▶ 54]。必须首先通过功能块 FB_SocketListen [▶ 24] 申请该句柄。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

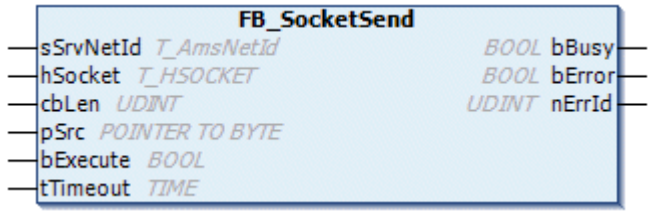
```
VAR_OUTPUT
    bAccepted : BOOL;
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    hSocket   : T_HSOCKET;
END_VAR
```

名称	类型	描述
bAccepted	BOOL	如果已与远程客户端建立新的连接，则会设置此输出。
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新的远程客户端的连接句柄 [▶ 54]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.6 FB_SocketSend



利用功能块 FB_SocketSend，可以通过 TwinCAT TCP/IP Connection Server 向远程客户端或远程服务器发送数据。首先要通过功能块 FB_SocketAccept [▶ 25] 建立远程客户端连接，或通过功能块 FB_SocketConnect [▶ 21] 建立远程服务器连接。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	待发送数据的通信伙伴的 连接句柄 [▶ 54]。
cbLen	UDINT	以字节为单位的待发送数据。
pSrc	POINTER TO BYTE	发送缓冲区的地址（指针）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

设置功能块的执行时间

如果套接字的发送缓冲区已满，例如，由于远程通信伙伴接收传输数据的速度不够快或传输大量数据，则 FB_SocketSend 功能块会在 tTimeout 时间过后返回 ADS 超时错误 1861。在这种情况下，必须相应增加 tTimeout 输入变量的值。

输出

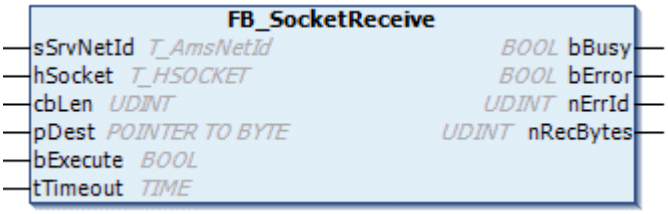
```
VAR_OUTPUT
    bBusy      : BOOL;
    bError     : BOOL;
    nErrId     : UDINT;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.7 FB_SocketReceive



利用功能块 FB_SocketReceive，可以通过 TwinCAT TCP/IP Connection Server 接收远程客户端或远程服务器发送的数据。首先要通过功能块 FB_SocketAccept [▶ 25] 建立远程客户端连接，然后通过功能块 FB_SocketConnect [▶ 21] 建立远程服务器连接。可以在 TCP/IP 网络中以分片形式（即多个数据包）接收或发送数据。因此，单次调用 FB_SocketReceive 实例可能无法接收全部数据。为此，必须在 PLC 任务中周期性调用该实例（轮询），直至收到全部所需数据。在此过程中，bExecute 输入端会生成上升沿，例如每 100 ms 一次。如果成功，会将最后收到的数据复制到接收缓冲区。nRecBytes 输出会返回最后一次成功接收的数据字节数。如果在最后一次调用期间无法读取新数据，则功能块不会返回错误，且 nRecBytes == 0。

在一套简单接收协议中，例如，接收远程服务器中以空字符结尾的字符串，必须反复调用功能块 FB_SocketReceive，直至在接收到的数据中检测到空终止符为止。

设置超时值

如果远程设备已从 TCP/IP 网络断开连接（仅限远程端），而本地设备仍连接在 TCP/IP 网络上，则功能块 FB_SocketReceive 不会返回错误和数据。打开的套接字仍然存在，但没有接收到数据。在这种情况下，应用程序可能会一直等待数据。建议在 PLC 应用程序中实施超时监控。如果在一定时间（如 10 秒）后仍未接收到所有数据，则必须关闭并重新初始化连接。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	待接收数据的通信伙伴的连接句柄 [▶ 54]。
cbLen	UDINT	要读取的数据的最大可用缓冲器大小（以字节为单位）。
pDest	POINTER TO BYTE	接收缓冲区的地址（指针）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

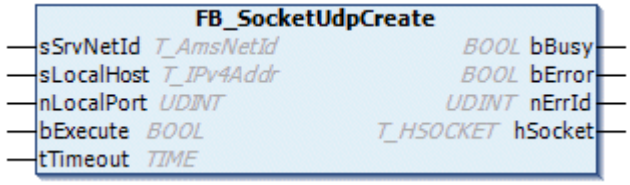
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    nRecBytes : UDINT;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号。
nRecBytes	UDINT	最后成功接收的数据字节数。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.8 FB_SocketUdpCreate



功能块 FB_SocketUdpCreate 可用于打开用户数据报协议 (User Datagram Protocol, UDP) 的客户端/服务器套接字。如果成功，则会打开新的套接字，并在 hSocket 输出端返回相关套接字句柄。例如，功能块 FB_SocketUdpSendTo [▶ 29] 和 FB_SocketUdpReceiveFrom [▶ 31] 都需要使用该句柄，以便与远程设备交换数据。如果不再需要 UDP 套接字，则可使用功能块 FB_SocketClose [▶ 22] 关闭该套接字。端口地址 nLocalPort 由 TCP/IP Connection Server 内部为 UDP 协议预留（已进行“绑定”）。一台 PC 中可能有多个网络适配器。输入参数 sLocalHost 决定了要使用的网络适配器。如果忽略 sLocalHost 输入变量（空字符串），那么 TCP/IP Connection Server 会使用默认网络适配器。通常是控制面板网络适配器列表中的第一个网络适配器。

自动创建的网络连接

如果在调用 FB_SocketUdpCreate 时为 sLocalHost 指定的是空字符串，并且 PC 已断开网络连接，那么系统会使用软件环回 IP 地址“127.0.0.1”打开新的套接字。

通过多个网络适配器自动创建网络连接

如果 PC 中安装了 2 个或多个网络适配器，且将空字符串指定为 sLocalHost，并且默认网络适配器已断开网络连接，则会在第二个网络适配器的 IP 地址下打开新的套接字。

设置网络地址

为了防止套接字在不同的 IP 地址下打开，可以明确指定 sLocalHost 地址，或者在句柄变量 (hSocket) 中检查返回的地址，关闭套接字并重新打开。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
sLocalHost	T_IPv4Addr	UDP 客户端/服务器套接字的本地 IP 地址 (Ipv4) 用作字符串（如“172.33.5.1”）。可为默认网络适配器指定空字符串。

名称	类型	描述
nLocalPort	UDINT	UDP 客户端/服务器套接字的本地 IP 端口号（如 200）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

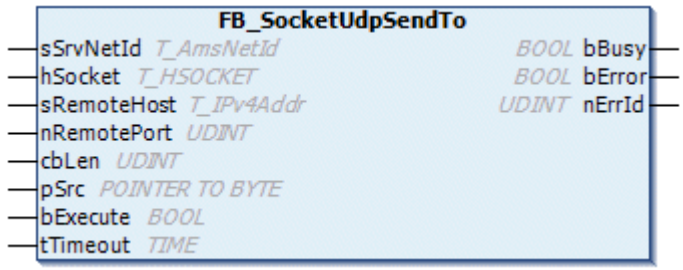
```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    hSocket   : T_HSOCKET;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新打开的 UDP 客户端/服务器套接字的句柄 [▶ 54]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.9 FB_SocketUdpSendTo



功能块 FB_SocketUdpSendTo 可用于通过 TwinCAT TCP/IP Connection Server 向远程设备发送 UDP 数据。必须先通过功能块 FB_SocketUdpCreate [▶ 28] 打开 UDP 套接字。

输入

```
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
```

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	已打开的 UDP 套接字的句柄 [▶ 54]。
sRemoteHost	T_IPv4Addr	要向其发送数据的远程设备的 IP 地址 (Ipv4)，以字符串形式表示（如“172.33.5.1”）。在本地计算机上，可以为设备输入空字符串。
nRemotePort	UDINT	要向其发送数据的远程设备的 IP 端口号（如 200）。

名称	类型	描述
cbLen	UDINT	以字节为单位的待发送数据。发送数据的最大字节数限为 8192 字节（为了节省存储容量，为库中的 TCPADS_MAXUDP_BUFFSIZE 常量）。
pSrc	POINTER TO BYTE	发送缓冲区的地址（指针）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

设置接收数据字节的大小

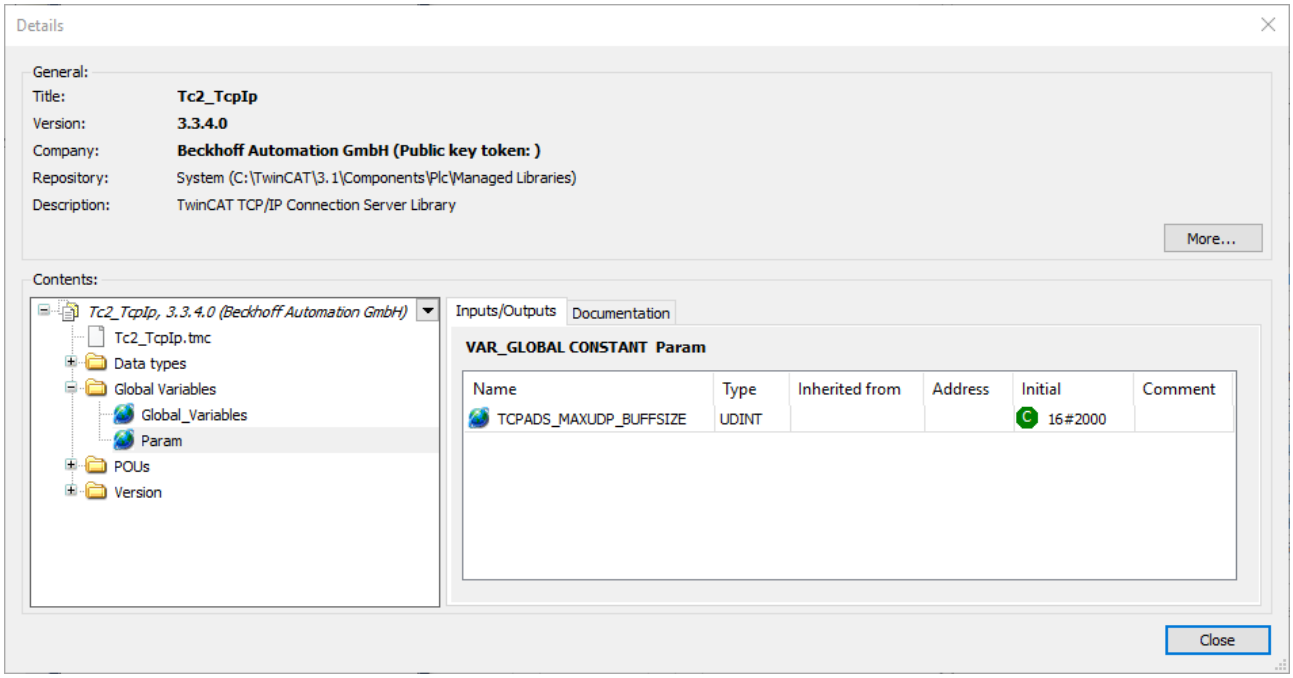
在产品版本 TwinCAT TCP/IP Connection Server v1.0.50 或更高版本中：可增加待接收数据的最大字节数（仅在绝对必要时）。

TwinCAT 2

1. 重新定义 PLC 项目中的全局常量（在示例中，需将可接收数据的最大字节数增加到 32000）：
```
VAR_GLOBAL CONSTANT
    TCPADS_MAXUDP_BUFFSIZE : UDINT := 32000;
END_VAR
```
2. 在 TwinCAT PLC 控制的对话框中激活 **Replace constants（替换常量）**选项（Project > Options ... > Build（项目 > 选项 > 构建））。
3. 重建项目。

TwinCAT 3

在 TwinCAT 3 中，可以通过 PLC 库（3.3.4.0 以上版本）的参数列表编辑该值。



输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。

名称	类型	描述
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.10 FB_SocketUdpReceiveFrom

利用功能块 FB_SocketUdpReceiveFrom，可以通过 TwinCAT TCP/IP Connection Server 从打开的 UDP 套接字接收数据。必须先通过功能块 FB_SocketUdpCreate [▶ 28] 打开 UDP 套接字。FB_SocketUdpReceive 功能块的实例必须在 PLC 任务中进行周期性调用（轮询）。在此过程中，bExecute 输入端会生成上升沿，例如每 100 ms 一次。如果成功，会将最后收到的数据复制到接收缓冲区。nRecBytes 输出会返回最后一次成功接收的数据字节数。如果在最后一次调用期间无法读取新数据，则功能块不会返回错误，且 nRecBytes == 0。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	已打开的 UDP 套接字句柄 [▶ 54]，将通过该句柄接收数据。
cbLen	UDINT	要读取的数据的最大可用缓冲器大小（以字节为单位）。可接收数据的最大字节数为 8192 字节（库中的 TCPADS_MAXUDP_BUFFSIZE 常量，用于节省存储空间）。
pDest	POINTER TO BYTE	接收缓冲区的地址（指针）。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

设置接收数据字节的大小

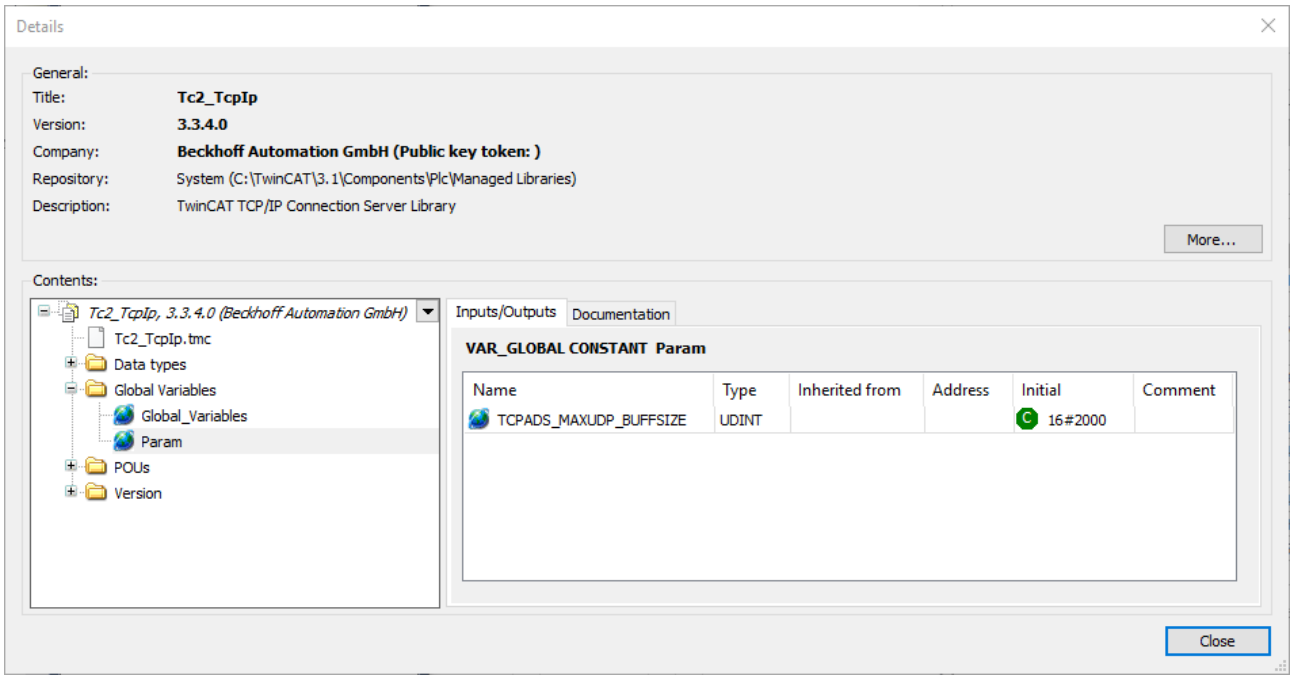
在产品版本 TwinCAT TCP/IP Connection Server v1.0.50 或更高版本中：可增加待接收数据的最大字节数（仅在绝对必要时）。

TwinCAT 2

1. 重新定义 PLC 项目中的全局常量（在示例中，需将可接收数据的最大字节数增加到 32000）：
```
VAR_GLOBAL CONSTANT
    TCPADS_MAXUDP_BUFFSIZE : UDINT := 32000;
END_VAR
```
2. 在 TwinCAT PLC 控制的对话框中激活 **Replace constants（替换常量）**选项（Project > Options ... > Build（项目 > 选项 > 构建））。
3. 重建项目。

TwinCAT 3

在 TwinCAT 3 中，可以通过 PLC 库（3.3.4.0 以上版本）的参数列表编辑该值。



输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
sRemoteHost	T_IPv4Addr	如果成功，则返回接收数据的远程设备的 IP 地址 (Ipv4)。
nRemotePort	UDINT	如果成功，则返回接收数据的远程设备的 IP 端口号（如 200）。
nRecBytes	UDINT	最后成功接收的数据字节数。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.11 FB_SocketUdpAddMulticastAddress

将服务器绑定到多播 IP 地址，以便接收多播数据包。该功能块要求已建立 UDP 套接字连接，该连接可通过功能块 FB_SocketUdpCreate [▶ 28] 建立。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	监听套接字的连接句柄 [▶ 54]。必须先通过功能块 FB_SocketUdpCreate [▶ 28] 申请该句柄。
sMulticastAddr	T_IPv4Addr	应与之绑定的多播 IP 地址。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.12 FB_SocketUdpDropMulticastAddress

解除此前通过功能块 FB_SocketUdpAddMulticastAddress [▶ 33] 设置的多播 IP 地址的绑定。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	监听套接字的连接句柄 [▶ 54]。必须先通过功能块 FB_SocketUdpCreate [▶ 28] 申请该句柄。
sMulticastAddr	T_IPv4Addr	应与之绑定的多播 IP 地址。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.13 FB_TlsSocketConnect

通过 FB_TlsSocketConnect 功能块，客户端可以通过 TwinCAT TCP/IP Connection Server 与远程服务器建立新的 TCP/IP 连接，并通过 TLS 加密。如果连接成功，则会打开新的套接字，并在 hSocket 输出端返回相应的连接句柄。例如，功能块 FB_SocketSend [▶ 26] 和 FB_SocketReceive [▶ 27] 都需要使用连接句柄，以便与远程服务器交换数据。如果不再需要连接，则可使用功能块 FB_SocketClose [▶ 22] 关闭该连接。多个客户端可以同时与远程服务器建立连接。对于每个新的客户端，都会打开新的套接字，并返回新的连接句柄。TwinCAT TCP/IP Connection Server 会自动为每个客户端分配新的 IP 端口号。TLS 参数可通过功能块 FB_TlsSocketAddCa [▶ 38]、FB_TlsSocketAddCrl [▶ 39]、FB_TlsSocketSetPsk [▶ 40] 和 FB_TlsSocketSetCert [▶ 39] 进行定义。关于其用法的编程示例，请参见我们的示例。

输入

```
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
```

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机，可以指定“127.0.0.1”。
sRemoteHost	STRING(TCPADS_TLS_HOSTNAME_SIZE)	采用字符串形式的远程服务器 IP 地址 (Ipv4)（如 172.33.5.1）。对于服务器，可以在本地计算机上输入空字符串。
nRemotePort	UDINT	远程服务器的 IP 端口号（如 200）。
flags	ST_TlsConnectFlags [▶ 53]	其他（可选）客户端连接参数。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

设置功能块的最长执行时间

不要将“tTimeout”值设得过低，因为如果网络中断，超时时间可能大于 30 s。如果该值过低，则会提前中断执行命令，并返回 ADS 错误代码 1861（超时时间已过），而非 Winsocket 错误 WSAETIMEDOUT。

输入/输出

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

名称	类型	描述
hSocket	T_HSOCKET	新打开的本地客户端套接字的 TCP/IP 连接句柄 [▶ 54]

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBusy 输出被重置后，将会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.14 FB_TlsSocketListen

功能块 FB_TLsSocketListen 可用于通过 TwinCAT TCP/IP Connection Server，打开一个通过 TLS 加密的新监听套接字。通过监听套接字，TwinCAT TCP/IP Connection Server 可以“监听”远程客户端发出的连接请求。由功能块 FB_TlsSocketCreate [▶ 37] 创建的套接字句柄可由功能块 FB_SocketAccept [▶ 25] 用于接受传入的客户端请求。如果不再需要监听套接字，可使用功能块 FB_SocketClose [▶ 22] 关闭该套接字。每台计算机上的监听套接字必须有唯一的 IP 端口号。有关使用该功能块的编程示例，请参见我们的示例。

输入

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

名称	类型	描述
hListener	T_HSOCKET	通过功能块 FB_TlsSocketCreate 创建的套接字句柄。
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。可以为本地计算机指定空字符串。
sLocalHost	T_IPv4Addr	采用字符串形式的本地服务器 IP 地址 (Ipv4)（如“172.13.15.2”）。“0.0.0.0”可用于列出所有可用的网络适配器。
nLocalPort	UDINT	本地服务器 IP 端口（如 200）。
flags	ST_TlsListenFlags [▶ 54]	其他（可选）服务器连接设置。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输入/输出

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

名称	类型	描述
hListener	T_HSOCKET	新的监听套接字的连接句柄 [▶ 54]。

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBusy 输出被重置后，将会设置该输出。

名称	类型	描述
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.15 FB_TlsSocketCreate

功能块 FB_TlsSocketCreate 可用于通过 TwinCAT TCP/IP Connection Server 创建新的套接字，既可用于服务器 (bListener:=true)，也可用于客户端应用程序 (bListener:=false)。通过监听套接字，TwinCAT TCP/IP Connection Server 可以“监听”远程客户端发出的连接请求。如果成功，相关连接句柄 (hSocket) 可在 hListner 输出端返回。功能块 FB_TlsSocketListen [▶ 36] 和后续的 FB_SocketAccept [▶ 25] 都需要使用该句柄。如果不再需要监听套接字，可使用功能块 FB_SocketClose [▶ 22] 关闭该套接字。执行功能块 FB_TlsSocketCreate 之后，可以设置 TLS 参数，以确保通信连接的安全。这是通过功能块 FB_TlsSocketAddCa [▶ 38]、FB_TlsSocketAddCrl [▶ 39]、FB_TlsSocketSetCert [▶ 39] 和 FB_TlsSocketSetPsk [▶ 40] 来实现的。相关编程示例，请参见我们的示例。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
bListener	BOOL	创建新的套接字句柄。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

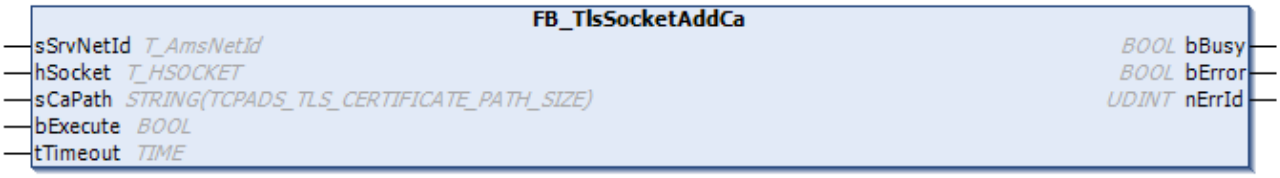
```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    hSocket : T_HSOCKET;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新套接字的连接句柄 [▶ 54]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.16 FB_TlsSocketAddCa



FB_TlsSocketAddCa 功能块用于为现有套接字句柄配置 CA 证书的路径。证书文件必须采用 PEM 格式。有关使用该功能块的编程示例，请参见我们的示例。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	套接字句柄。
sCaPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	CA 证书文件的路径。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

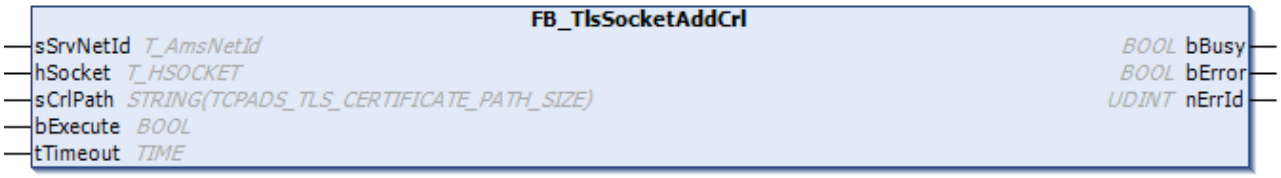
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.17 FB_TlsSocketAddCrl



功能块 FB_TlsSocketAddCrl 用于为现有套接字句柄指定 CRL 文件的路径。CRL 必须是 PEM 格式。有关使用该功能块的编程示例，请参见我们的示例。

输入

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

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	套接字句柄。
sCrlPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	CRL 文件的路径。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.18 FB_TlsSocketSetCert

功能块 FB_TlsSocketSetCert 可用于配置用于特定套接字句柄的客户端/服务器证书。证书必须采用 PEM 格式。有关使用该功能块的编程示例，请参见我们的示例。

输入

```
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
```

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	套接字句柄。
sCertPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	客户端/服务器证书文件的路径。
sKeyPath	STRING(TCPADS_TLS_CERTIFICATE_PATH_SIZE)	客户端/服务器私钥文件的路径。
sKeyPwd	string(tcpads_tls_key_password_size)	可选项：如果私钥设置密码保护，则选择此项。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

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

名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.19 FB_TlsSocketSetPsk

功能块 FB_TlsSocketSetPsk 可用于为现有套接字句柄配置预共享密文。有关使用该功能块的编程示例，请参见我们的示例。

输入

```
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
```

名称	类型	描述
sSrvNetId	T_AmsNetId	包含 TwinCAT TCP/IP Connection Server 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
hSocket	T_HSOCKET	套接字句柄。
sIdentity	string(tcpads_tls_psk_identity_size)	可自行选定的 PSK 标识。
pskKey	PVOID	指向包含 PSK 的字节数组的指针。
pskKeyLen	udint(0..tcpads_tls_max_psk_key_size)	pskKey 的长度。
bExecute	BOOL	功能块由该输入的上升沿启用。
tTimeout	TIME	允许执行功能块的最长时间。

输出

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

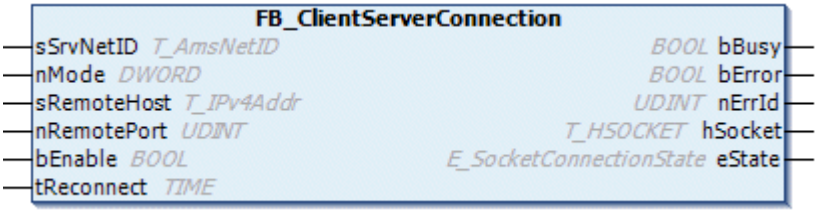
名称	类型	描述
bBusy	BOOL	在功能块激活时，该输出生效。在确认之前，该输出保持有效。
bError	BOOL	如果在命令传输过程中出现错误，则在 bBUSY 输出复位后，会设置该输出。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.20 辅助功能块

5.1.20.1 FB_ClientServerConnection



功能块 FB_ClientServerConnection 可用于管理（建立或删除）客户端连接。FB_ClientServerConnection 在内部封装了 FB_SocketConnect [▶ 21] 和 FB_SocketClose [▶ 22] 这 2 个功能块的功能，从而简化了客户端应用程序的实现。连接状态的集成调试输出便于在出现配置或连接错误时排除故障。此外，最简客户端应用程序只需要功能块 FB_SocketSend [▶ 26] 和/或功能块 FB_SocketReceive [▶ 27] 的实例。

第一步，典型的客户端应用程序通过 FB_ClientServerConnection 功能块与服务器建立连接。下一步，可以使用 FB_SocketSend 和/或 FB_SocketReceive 实例与服务器交换数据。何时关闭连接取决于应用程序的要求。

输入

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

名称	类型	描述
sSrvNetID	T_AmsNetID	包含 TwinCAT TCP/IP Connection Server AMS 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
nMode	DWORD	参数标志（模式）。此处列有允许使用的参数，可以通过 ORing 进行组合： CONNECT_MODE_ENABLEDBG: 启用在应用程序日志中对调试消息进行日志记录的功能。要查看调试消息，请打开 TwinCAT System Manager 并激活日志视图。
sRemoteHost	T_IPv4Addr	采用字符串形式的远程服务器 IP 地址 (Ipv4)（如“172.33.5.1”）。对于服务器，可以在本地计算机上输入空字符串。
nRemotePort	UDINT	远程服务器的 IP 端口号（如 200）。
bEnable	BOOL	只要该输入为 TRUE，系统就会定期尝试建立新的连接，直到成功建立连接。连接建立后，可以用 FALSE 再次关闭。
tReconnect	TIME	功能块尝试建立连接所用的周期时间。

设置连接的周期时间

tReconnect 值不得设置过低，因为网络中断时可能会出现 30 s 以上的超时。如果该值过低，则会提前中断执行命令，并返回 ADS 错误代码 1861（超时时间已过），而非 Winsocket 错误 WSAETIMEDOUT。

输出

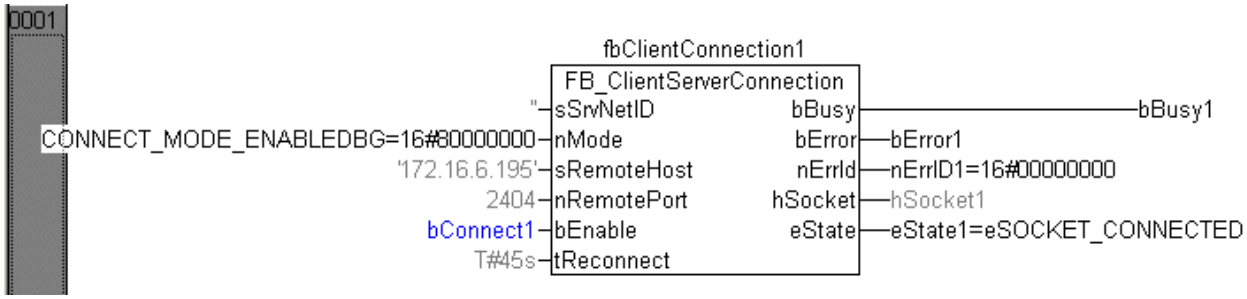
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hSocket     : T_HSOCKET;
    eState      : E_SocketConnectionState := eSOCKET_DISCONNECTED;
END_VAR
```

名称	类型	描述
bBusy	BOOL	TRUE，只要该功能块处于活动状态。
bError	BOOL	如果出现错误代码，则变为 TRUE。
nErrID	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新打开的本地客户端套接字的连接句柄 [▶ 54]。如果成功，会将该变量传输到功能块 FB_SocketSend [▶ 26] 和/或 FB_SocketReceive [▶ 27] 的实例中。
eState	E_SocketConnectionState	返回当前的连接状态 [▶ 50]。

FBD 调用示例

```
PROGRAM MAIN
VAR
    fbClientConnection1 : FB_ClientServerConnection;
    bConnect1           : BOOL;
    bBusy1              : BOOL;
    bError1             : BOOL;
```

```
    nErrID1             : UDINT;
    hSocket1            : T_HSOCKET;
    eState1             : E_SocketConnectionState;
END_VAR
```

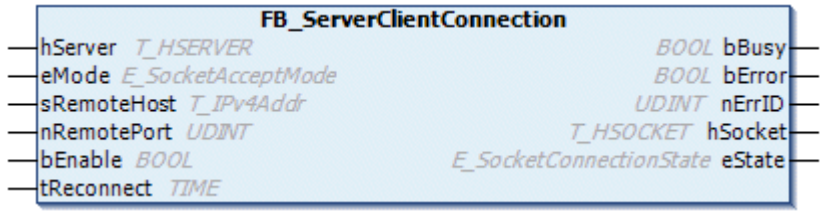


您可以在此查阅更多应用示例（和源代码）：示例 [▶ 57]

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.20.2 FB_ServerClientConnection



功能块 FB_ServerClientConnection 可用于管理（建立或删除）服务器连接。FB_ServerClientConnection 在内部封装了 FB_SocketListen [▶ 24]、FB_SocketAccept [▶ 25] 和 FB_SocketClose [▶ 22] 这 3 个功能块的功能，从而简化了服务器应用程序的实现。连接状态的集成调试输出便于在出现配置或连接错误时排除故障。此外，最简服务器应用程序只需要功能块 FB_SocketSend [▶ 26] 和/或功能块 FB_SocketReceive [▶ 27] 的实例。

第一步，典型的服务器应用程序通过 FB_ServerClientConnection 功能块与客户端建立连接（更精确地说，服务器应用程序接受传入的连接请求）。下一步，可以使用 FB_SocketSend 和/或 FB_SocketReceive 实例与服务器交换数据。何时关闭连接取决于应用程序的要求。

输入

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

名称	类型	描述
eMode	E_SocketAcceptMode	定义是接受所有连接 [▶ 50]还是只接受某些连接。
sRemoteHost	T_IPv4Addr	要接受其连接的远程客户端的 IP 地址 (Ipv4)，以字符串形式表示（如“172.33.5.1”）。对于本地计算机上的客户端，可以指定空字符串。
nRemotePort	UDINT	要接受其连接的远程客户端的 IP 端口号（如 200）。
bEnable	BOOL	只要该输入为 TRUE，系统就会定期尝试建立新的连接，直到成功建立连接。连接建立后，可以用 FALSE 再次关闭。
tReconnect	TIME	功能块尝试建立连接所用的周期时间。

/ 输入/输出

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

名称	类型	描述
hServer	hServer	服务器句柄 [▶ 54]。该输入变量必须通过 F_CreateServerHnd [▶ 47] 功能进行初始化。

输出

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

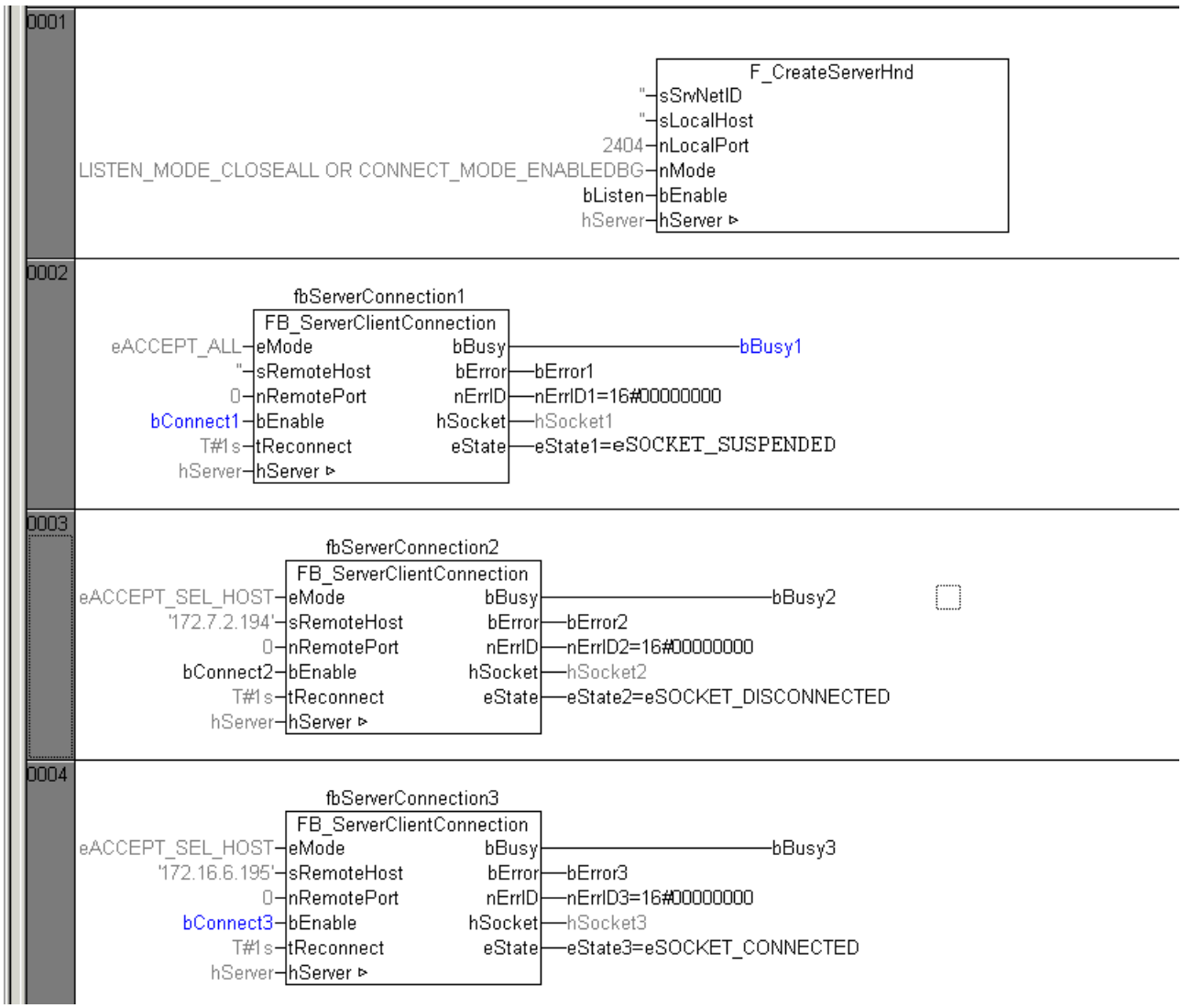
名称	类型	描述
bBusy	BOOL	TRUE，只要该功能块处于活动状态。
bError	BOOL	如果出现错误代码，则变为 TRUE。
nErrId	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新打开的远程客户端套接字的连接句柄 [▶ 54]。如果成功，会将该变量传输到功能块 FB_SocketSend [▶ 26] 和/或 FB_SocketReceive [▶ 27] 的实例中。
eState	E_SocketConnectionState	返回当前的连接状态 [▶ 50]。

FBD 中的示例

下面的示例演示了服务器句柄变量的初始化过程。然后，会将服务器句柄传输到 FB_ServerClientConnection 功能块的 3 个实例中。

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

    fbServerConnection1 : FB_ServerClientConnection;
    bConnect1           : BOOL;
    bBusy1              : BOOL;
    bError1             : BOOL;
    nErrID1             : UDINT;
    hSocket1            : T_HSOCKET;
    eState1             : E_SocketConnectionState;

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

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

在线视图：



第一个连接已激活 (`bConnect1 = TRUE`)，但连接尚未建立（被动打开）。

第二个连接尚未激活 (`bConnect2 = FALSE`)（已关闭）。

第三个连接已激活 (`bConnect3 = TRUE`) 并与远程客户端建立了连接。

您可以在此查阅更多应用示例（和源代码）：示例 [▶ 57]

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.1.20.3 FB_ConnectlessSocket

可以使用功能块 FB_ConnectionlessSocket 管理（打开/生成和关闭）UDP 套接字。FB_ConnectionlessSocket 在内部封装了 FB_SocketUdpCreate [▶ 28] 和 FB_SocketClose [▶ 22] 这 2 个功能块的功能，从而简化了 UDP 应用程序的实现。套接字状态的集成调试输出便于在出现配置或通信错误时排除故障。此外，最简 UDP 应用程序只需要功能块 SocketUdpSendTod [▶ 29] 和/或功能块 FB_SocketUdpReceiveFrom [▶ 31] 的实例。

第一步，典型 UDP 应用程序通过功能块 FB_ConnectionlessSocket 打开无连接的 UDP 套接字。下一步，FB_SocketUdpSendTo 和/或 FB_SocketUdpReceiveFrom 实例可用于与另一台通信设备交换数据。UDP 套接字何时关闭取决于应用程序的要求（例如，在通信出错时）。

输入

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

名称	类型	描述
sSrvNetID	T_AmsNetID	包含 TwinCAT TCP/IP Connection Server AMS 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
nMode	DWORD	参数标志（模式）。此处列有允许使用的参数，可以通过 ORing 进行组合。 CONNECT_MODE_ENABLEDBG: 启用在应用程序日志中对调试消息进行日志记录的功能。要查看调试消息，请打开 TwinCAT System Manager 并激活日志视图。
sLocalHost	T_Ipv4Addr	本地网络适配器的 IP 地址 (Ipv4)，以字符串形式表示（如“172.33.5.1”）。可为默认网络适配器指定空字符串。
nLocalPort	UDINT	本地计算机的 IP 端口号（如 200）。
bEnable	BOOL	只要该输入为 TRUE，就会周期性尝试打开 UDP 套接字，直到建立连接。打开的 UDP 套接字可以用 FALSE 重新关闭。
tReconnect	TIME	功能块尝试打开 UDP 套接字的周期时间。

设置连接的周期时间

tReconnect 值不得设置过低，因为网络中断时可能会出现 30 s 以上的超时。如果该值过低，则会提前中断执行命令，并返回 ADS 错误代码 1861（超时时间已过），而非 Winsocket 错误 WSAETIMEDOUT。

输出

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

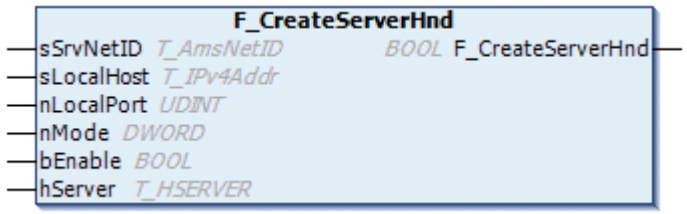
名称	类型	描述
bBusy	BOOL	TRUE，只要该功能块处于活动状态。
bError	BOOL	如果出现错误代码，则变为 TRUE。
nErrID	UDINT	如果已设置 bError 输出，则该参数会返回 TwinCAT TCP/IP Connection Server 的错误编号 [▶ 90]。
hSocket	T_HSOCKET	新打开的 UDP 套接字的连接句柄 [▶ 54]。如果成功，会将该变量传输到功能块 FB_SocketUdpSendTo [▶ 29] 和/或 FB_SocketUdpReceiveFrom [▶ 31] 的实例中。
eState	E_SocketConnectionlessState	返回当前的连接状态 [▶ 51]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.2 功能

5.2.1 F_CreateServerHnd



功能 F_CreateServerHnd 用于初始化/设置服务器句柄变量 hServer 的内部参数。然后，服务器句柄通过 VAR_IN_OUT 传输到功能块 FB_ServerClientConnection [▶ 43] 的实例。FB_ServerClientConnection 功能块的实例可用于直接管理（建立或删除）服务器连接。可将同一服务器句柄传输到功能块 FB_ServerClientConnection 的多个实例中，以便使服务器能够建立多个并发连接。

语法

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

返回值

名称	类型	描述
F_CreateServerHnd	BOOL	如果一切正常，则返回 TRUE；如果参数值不正确，则返回 FALSE。

输入

名称	类型	描述
sSrvNetID	T_AmsNetID	包含 TwinCAT TCP/IP Connection Server AMS 网络地址的字符串。对于本地计算机（默认），可以指定空字符串。
sLocalHost	T_IPv4Addr	采用字符串形式的本地服务器 IP 地址 (Ipv4)（如“172.13.15.2”）。对于本地计算机上的服务器（默认），可以输入空字符串。
nLocalPort	UDINT	本地服务器 IP 端口（如 200）。
nMode	DWORD	参数标志（模式）。此处列有允许使用的参数，可以通过 ORing 进行组合。 LISTEN_MODE_CLOSEALL: 所有之前打开的套接字连接都会关闭（默认）。 CONNECT_MODE_ENABLEDBG: 启用在应用程序日志中对调试消息进行日志记录的功能。要查看调试消息，请打开 TwinCAT System Manager 并激活日志视图。

名称	类型	描述
bEnable	BOOL	该输入决定监听套接字的行为。只要该输入为 TRUE，则事先打开的监听套接字就会保持打开状态。如果该输入为 FALSE，则监听套接字会自动关闭，但只有在最后（之前）接受的连接也关闭后才会关闭。

/ 输入/输出

名称	类型	描述
hServer	T_HSERVER	服务器句柄变量，其内部参数需要初始化。

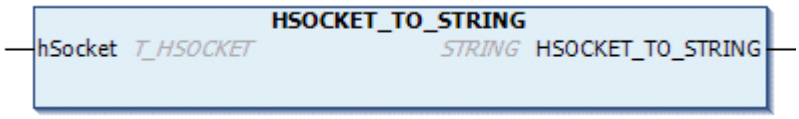
示例：

请参见 FB_ServerClientConnection [▶ 43]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.2.2 HSOCKET_TO_STRING



该函数会将 T_HSOCKET 类型的连接句柄转换为字符串（例如用于调试输出）。

返回的字符串格式如下：“Handle:0xA[BCD] Local:a[aa].b[bb].c[cc].d[dd]:port Remote:a[aa].b[bb].c[cc].d[dd]:port”。

示例：“Handle:0x4001 Local:172.16.6.195:28459 Remote:172.16.6.180:2404”

语法

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

返回值

名称	类型	描述
HSOCKET_TO_STRING	STRING	包含连接句柄的字符串形式。

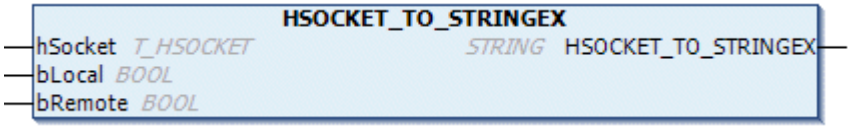
输入

名称	类型	描述
hSocket	T_HSOCKET	待转换的连接句柄 [▶ 54]。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.2.3 HSOCKET_TO_STRINGEX



该函数会将 T_HSOCKET 类型的连接句柄转换为字符串（例如用于调试输出）。

返回的字符串格式如下：“Handle:0xA[BCD] Local:a[aa].b[bb].c[cc].d[dd]:port Remote:a[aa].b[bb].c[cc].d[dd]:port”。

示例：“Handle:0x4001 Local:172.16.6.195:28459 Remote:172.16.6.180:2404”

参数 bLocal 和 bRemote 决定返回的字符串中是否包含本地和/或远程地址信息。

语法

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

返回值

名称	类型	描述
HSOCKET_TO_STRINGEX	STRING	包含连接句柄的十六进制字符串形式。

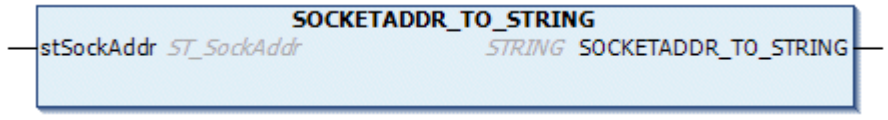
输入

名称	类型	描述
hSocket	T_HSOCKET	待转换的连接句柄 [▶ 54]。
bLocal	BOOL	TRUE：包含本地地址，FALSE：不包含本地地址。
bRemote	BOOL	TRUE：包含远程地址，FALSE：不包含远程地址。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.2.4 SOCKETADDR_TO_STRING



该函数会将 ST_SockAddr 类型的变量转换为字符串（例如用于调试输出）。

返回的字符串格式如下：“a[aa].b[bb].c[cc].d[dd]:port”

示例：“172.16.6.195:80”

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

返回值

名称	类型	描述
SOCKETADDR_TO_STRING	STRING	包含套接字地址的字符串形式。

输入

名称	类型	描述
stSockeAddr	ST_SockAddr	待转换的变量。

请参见 ST_SockAddr [▶ 53]

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3 数据类型

5.3.1 E_SocketAcceptMode

E_SocketAcceptMode 会指定服务器接受哪些连接。

语法

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

值

名称	描述
eACCEPT_ALL	接受与所有远程客户端的连接。
eACCEPT_SEL_HOST	接受与所选主机地址的连接。
eACCEPT_SEL_PORT	接受与所选端口地址的连接。
eACCEPT_SEL_HOST_PORT	接受与所选主机和端口地址的连接。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.2 E_SocketConnectionState

TCP/IP 套接字连接状态（eSOCKET_SUSPENDED == 状态发生变化，例如从 eSOCKET_CONNECTED => eSOCKET_DISCONNECTED）。

语法

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

值

名称	描述
eSOCKET_DISCONNECTED	连接中断。
eSOCKET_CONNECTED	连接已存在。
eSOCKET_SUSPENDED	连接状态从断开变为已连接，或从已连接变为断开。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.3 E_SocketConnectionlessState

无连接 UDP 套接字的状态信息（例如，eSOCKET_TRANSIENT == 状态从 eSOCKET_CREATED=>eSOCKET_CLOSED 改变）。

语法

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

值

名称	描述
eSOCKET_CLOSED	UDP 套接字已关闭。
eSOCKET_CREATED	UDP 套接字已创建。
eSOCKET_TRANSIENT	UDP 套接字从已关闭变为已打开，或从已打开变为已关闭。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.4 E_WinsockError

语法

```
TYPE E_WinsockError :
 (
    WSOK,
    WSAEINTR        := 10004 ,
(* A blocking operation was interrupted by a call to WSACancelBlockingCall. *)
    WSAEBADF        := 10009 ,(* The file handle supplied is not valid. *)
    WSAEACCES     := 10013 ,
(* An attempt was made to access a socket in a way forbidden by its access permissions. *)
    WSAEFAULT     := 10014 ,
(* The system detected an invalid pointer address in attempting to use a pointer argument in a call.
 *)
    WSAEINVAL     := 10022 ,(* An invalid argument was supplied. *)
    WSAEMFILE     := 10024 ,(* Too many open sockets. *)
    WSAEWOULDBLOCK     := 10035 ,(* A non-
```

```
blocking socket operation could not be completed immediately. *)
    WSAEINPROGRESS     := 10036 ,(* A blocking operation is currently executing. *)
    WSAEALREADY     := 10037 ,(* An operation was attempted on a non-
blocking socket that already had an operation in progress. *)
    WSAENOTSOCK     := 10038 ,(* An operation was attempted on something that is not a socket. *)
    WSAEDESTADDRREQ     := 10039 ,
(* A required address was omitted from an operation on a socket. *)
    WSAEMSGSIZE     := 10040 ,
(* A message sent on a datagram socket was larger than the internal message buffer or some other net
work limit, or the buffer used to receive a datagram into was smaller than the datagram itself. *)
    WSAEPROTOTYPE     := 10041 ,
(* A protocol was specified in the socket function call that does not support the semantics of the s
ocket type requested. *)
    WSAENOPROTOOPT     := 10042 ,
(* An unknown, invalid, or unsupported option or level was specified in a getsockopt or setsockopt c
all. *)
    WSAEPROTONOSUPPORT := 10043 ,
(* The requested protocol has not been configured into the system, or no implementation for it exist
s. *)
    WSAESOCKTNOSUPPORT := 10044 ,
(* The support for the specified socket type does not exist in this address family. *)
    WSAEOPNOTSUPP     := 10045 ,
(* The attempted operation is not supported for the type of object referenced. *)
    WSAEPFNOSUPPORT     := 10046 ,
(* The protocol family has not been configured into the system or no implementation for it exists. *
)
    WSAEAFNOSUPPORT     := 10047 ,
(* An address incompatible with the requested protocol was used. *)
    WSAEADDRINUSE     := 10048 ,(* Only one usage of each socket address (protocol/network address/
port) is normally permitted. *)
    WSAEADDRNOTAVAIL    := 10049 ,(* The requested address is not valid in its context. *)
    WSAENETDOWN     := 10050 ,(* A socket operation encountered a dead network. *)
    WSAENETUNREACH     := 10051 ,(* A socket operation was attempted to an unreachable network. *)
    WSAENETRESET     := 10052 ,(* The connection has been broken due to keep-
alive activity detecting a failure while the operation was in progress. *)
    WSAECONNABORTED     := 10053 ,
(* An established connection was aborted by the software in your host machine. *)
    WSAECONNRESET     := 10054 ,(* An existing connection was forcibly closed by the remote host. *)
    WSAENOBUFS     := 10055 ,
(* An operation on a socket could not be performed because the system lacked sufficient buffer space
 or because a queue was full. *)
    WSAEISCONN     := 10056 ,(* A connect request was made on an already connected socket. *)
    WSAENOTCONN     := 10057 ,
(* A request to send or receive data was disallowed because the socket is not connected and (when se
nding on a datagram socket using a sendto call) no address was supplied. *)
    WSAESHUTDOWN     := 10058 ,
(* A request to send or receive data was disallowed because the socket had already been shut down in
 that direction with a previous shutdown call. *)
    WSAETOOMANYREFS     := 10059 ,(* Too many references to some kernel object. *)
      WSAETIMEDOUT     := 10060 ,
(* A connection attempt failed because the connected party did not properly respond after a period o
f time, or established connection failed because connected host has failed to respond. *)
    WSAECONNREFUSED     := 10061 ,
(* No connection could be made because the target machine actively refused it. *)
    WSAELOOP        := 10062 ,(* Cannot translate name. *)
    WSAENAMETOOLONG     := 10063 ,(* Name component or name was too long. *)
    WSAEHOSTDOWN     := 10064 ,
(* A socket operation failed because the destination host was down. *)
    WSAEHOSTUNREACH     := 10065 ,(* A socket operation was attempted to an unreachable host. *)
    WSAENOTEMPTY     := 10066 ,(* Cannot remove a directory that is not empty. *)
    WSAEPROCLIM     := 10067 ,
(* A Windows Sockets implementation may have a limit on the number of applications that may use it s
imultaneously. *)
    WSAEUSERS     := 10068 ,(* Ran out of quota. *)
    WSAEDQUOT     := 10069 ,(* Ran out of disk quota. *)
    WSAESTALE     := 10070 ,(* File handle reference is no longer available. *)
    WSAEREMOTE     := 10071 ,(* Item is not available locally. *)
    WSASYSNOTREADY     := 10091 ,
(* WSAStartup cannot function at this time because the underlying system it uses to provide network
services is currently unavailable. *)
    WSAVERNOTSUPPORTED := 10092 ,(* The Windows Sockets version requested is not supported. *)
    WSANOTINITIALISED := 10093 ,
(* Either the application has not called WSAStartup, or WSAStartup failed. *)
    WSAEDISCON     := 10101 ,
(* Returned by WSARecv or WSARecvFrom to indicate the remote party has initiated a graceful shutdown
 sequence. *)
    WSAENOMORE     := 10102 ,(* No more results can be returned by WSALookupServiceNext. *)
    WSAECANCELLED     := 10103 ,
(* A call to WSALookupServiceEnd was made while this call was still processing. The call has been ca
nceled. *)
    WSAEINVALIDPROCTABLE := 10104 ,(* The procedure call table is invalid. *)
    WSAEINVALIDPROVIDER := 10105 ,(* The requested service provider is invalid. *)
    WSAEPROVIDERFAILEDINIT := 10106 ,
(* The requested service provider could not be loaded or initialized. *)
    WSASYSCALLFAILURE := 10107 ,(* A system call that should never fail has failed. *)
    WSASERVICE_NOT_FOUND := 10108 ,
(* No such service is known. The service cannot be found in the specified name space. *)
    WSATYPE_NOT_FOUND := 10109 ,(* The specified class was not found. *)
    WSA_E_NO_MORE     := 10110 ,(* No more results can be returned by WSALookupServiceNext. *)
    WSA_E_CANCELLED     := 10111 ,
(* A call to WSALookupServiceEnd was made while this call was still processing. The call has been ca
nceled. *)
    WSAEREFUSED     := 10112 ,(* A database query failed because it was actively refused. *)
    WSAHOST_NOT_FOUND := 11001 ,(* No such host is known. *)
```

```
    WSATRY_AGAIN     := 11002 ,
(* This is usually a temporary error during hostname resolution and means that the local server did
not receive a response from an authoritative server. *)
    WSANO_RECOVERY     := 11003 ,(* A non-recoverable error occurred during a database lookup. *)
    WSANO_DATA     := 11004 (* The requested name is valid and was found in the database, but it doe
s not have the correct associated data being resolved for. *)
 );
END_TYPE
```

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.5 ST_SockAddr

该结构包含已打开套接字的地址信息。

语法

```
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
```

值

名称	类型	描述
nPort	UDINT	互联网协议 (Internet Protocol, IP) 端口
sAddr	STRING(15)	以字符串形式分隔的互联网协议地址 (Ipv4)，例如“172.34.12.3”

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.6 ST_TlsConnectFlags

其他（可选）客户端连接参数。

语法

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

值

名称	类型	描述
bNoServerCertCheck	BOOL	禁用服务器证书验证功能。
bIgnoreCnMismatch	BOOL	如果服务器证书中的 CommonName 与指定为 sRemoteHost 的主机名不匹配，则忽略。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.7 ST_TlsListenFlags

其他（可选）服务器连接参数。

语法

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

值

名称	类型	描述
bNoClientCert	BOOL	无需客户证书。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.8 T_HSERVER

该类型的变量代表 TCP/IP 服务器句柄。必须使用 F_CreateServerHnd [▶ 47] 对句柄进行初始化后才能使用。在此过程中，会设置变量 T_HSERVER 的内部参数。

保留默认结构元素

不得写入或更改结构元素。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.3.9 T_HSOCKET

该类型的变量表示连接句柄或已打开的套接字句柄。借助该句柄，可以通过套接字发送或接收数据。该句柄可用于关闭已打开的套接字。

语法

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

值

名称	类型	描述
句柄	UDINT	TwinCAT TCP/IP Connection Server 内部套接字句柄。
localAddr	ST_SockAddr	本地套接字地址 [▶ 53]。
remoteAddr	ST_SockAddr	远程套接字地址 [▶ 53]。

可以通过 TwinCAT TCP/IP Connection Server 打开和关闭以下套接字：监听套接字、远程客户端套接字或本地客户端套接字。根据这些套接字由 TwinCAT TCP/IP Connection Server 打开的具体情况，将相应的地址信息填入 localAddr 和 remoteAddr 变量中。

服务器端的连接句柄

- 功能块 FB_SocketListen [▶ 24] 会打开监听套接字，并返回监听套接字的连接句柄。
- 会将监听套接字的连接句柄传输到功能块 FB_SocketAccept [▶ 25]。然后，FB_SocketAccept 将返回远程客户端的连接句柄。
- 功能块 FB_SocketAccept 会为每个已连接的远程客户端返回新的连接句柄。
- 然后，会将连接句柄传输到功能块 FB_SocketSend [▶ 26] 和/或 FB_SocketReceive [▶ 27]，以便与远程客户端交换数据。
- 会将不需要或不再需要的远程客户端的连接句柄传输到功能块 FB_SocketClose [▶ 22]，该功能块会关闭远程客户端套接字。
- 同时会将不再需要的监听套接字连接句柄传输到用于关闭监听套接字的功能块 FB_SocketClose。

客户端的连接句柄

- 功能块 FB_SocketConnect [▶ 21] 会返回本地客户端套接字的连接句柄。
- 接着，会将连接句柄传输到功能块 FB_SocketSend [▶ 26] 和 FB_SocketReceive [▶ 27]，以便与远程服务器交换数据。
- 然后，会将同一连接句柄传输到功能块 FB_SocketClose [▶ 22]，以便关闭不再需要的连接。

功能块 FB_SocketCloseAll [▶ 23] 可用于关闭由 PLC Runtime 系统打开的所有连接句柄（套接字）。这意味着，如果在第一个 Runtime 系统（端口 801）的某个任务中调用了 FB_SocketCloseAll，那么该 Runtime 系统中所有已打开的套接字都会关闭。

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.4 全局常量

5.4.1 库版本

所有库都有特定版本。该版本信息会显示在 PLC 库的存储库中。
库版本号存储在全局常量中（类型：ST_LibVersion）。

Global_Version

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

函数 F_CmpLibVersion（在 Tc2_System 库中）可用于比较现有版本和所需版本。

与 TwinCAT 2 兼容

不再提供 TwinCAT 2 库的查询选项！

要求

开发环境	目标系统类型	需包含的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

5.4.2 参数列表

Param

名称	类型	值	描述
TCPADS_MAXUDP_BUFFSIZE	UDINT	16#2000	内部 UDP 发送/接收缓冲区的最大字节长度（8192 字节）。
TCPADS_TLS_HOSTNAME_SIZE	UDINT	255	主机名称最大字符串长度。
TCPADS_TLS_CERTIFICATE_PATH_SIZE	UDINT	255	证书路径最大字符串长度。
TCPADS_TLS_KEY_PASSWORD_SIZE	UDINT	255	证书密码路径最大字符串长度。
TCPADS_TLS_PSK_IDENTITY_SIZE	UDINT	255	PSK 标识最大字符串长度。
TCPADS_TLS_MAX_PSK_KEY_SIZE	UDINT	128	PSK 密钥最大字节长度。

要求

开发环境	目标平台	待集成的 PLC 库（类别组）
TF6310 v3.3.15.0 或更高版本 TwinCAT v3.1.0	PC 或 CX（x86、x64、Arm®）	Tc2_TcpIp（通信）

6 示例

概述

以下将产品用作 TCP/IP 客户端/服务器的示例可在我们的 GitHub 存储库中下载（见下文）。

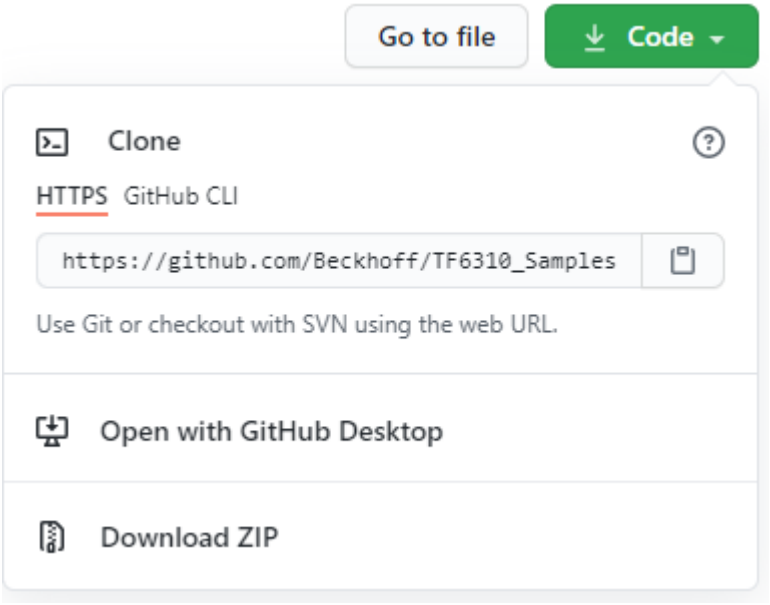
链接	描述
快速入门 [▶ 18]	启用快速入门功能，将产品用作 TCP/IP 客户端/服务器。
Sample01：“Echo”客户端/服务器（基本功能块） [▶ 58]	演示了一个 TCP/IP 客户端/服务器应用程序的示例实现，该程序可在客户端和服务器之间周期性交换消息。以基本功能块为基础。
Sample02：“Echo”客户端/服务器 [▶ 74]	演示了一个 TCP/IP 客户端/服务器应用程序的示例实现，该程序可在客户端和服务器之间周期性交换消息。以辅助功能块为基础。该应用程序最多允许建立 1 个连接。
Sample03：“Echo”客户端/服务器 [▶ 76]	演示了一个 TCP/IP 客户端/服务器应用程序的示例实现，该程序可在客户端和服务器之间周期性交换消息。以辅助功能块为基础。该应用程序允许建立多个连接。
Sample04：二进制数据交换 [▶ 77]	演示了一个 TCP/IP 客户端/服务器应用程序的示例实现，该程序可实现自身二进制数据交换协议。该应用程序最多允许建立 1 个连接。
Sample05：二进制数据交换 [▶ 79]	演示了一个 TCP/IP 客户端/服务器应用程序的示例实现，该程序可实现自身二进制数据交换协议。该应用程序允许建立多个连接。
Sample06：基于 TLS 的“Echo”客户端/服务器（基本模块） [▶ 80]	该示例基本上以 Sample01 为基础，并通过 TLS 进行扩展。
Sample07：基于 TLS-PSK 的“Echo”客户端/服务器（基本模块） [▶ 81]	该示例基本上以 Sample01 为基础，并通过 TLS-PSK 进行扩展。

以下将产品用作 UDP/IP 客户端/服务器的示例可在我们的 GitHub 存储库中下载（见下文）。

链接	描述
Sample01：点对点通信 [▶ 81]	该示例演示了在 PLC 中实现简单的点对点应用程序的过程。
Sample02：多播 [▶ 88]	该示例演示了如何通过 UDP 发送和接收多播信息。

下载

本产品的示例代码和配置可从 GitHub 上的相应存储库获取：https://github.com/Beckhoff/TF6310_Samples。您可以选择克隆存储库或下载包含示例的 ZIP 文件。



6.1 TCP

6.1.1 Sample01：“Echo”客户端/服务器（基本功能块）

6.1.1.1 概述

下方示例展示的是“Echo”客户端/服务器的实现。客户端每隔一定时间（如每秒）向服务器发送 1 个测试字符串。然后，远程服务器会立即向客户端重新发送相同的字符串。

在本示例中，客户端分别通过 PLC 程序与 C# 编写的.NET 应用程序两种方式实现。PLC 客户端可以创建多个通信实例，同时模拟多个 TCP 连接。.NET 示例客户端只能建立一个并发连接。服务器可与多个客户端通信。

此外，还可以创建多个服务器实例。然后，每个服务器实例通过不同的端口号寻址，客户端可使用这些端口号与特定的服务器实例连接。如果服务器必须与多个客户端通信，那么服务器的实现便会更加困难。

请根据自身需求自由使用并自定义该示例。

系统要求

- TwinCAT 3 Build 3093 或更高版本
- TwinCAT 3 功能组件 TF6310 TCP/IP
- 如果使用 2 台计算机执行本示例（1 台客户端和 1 台服务器），则需要在 2 台计算机上安装功能组件 TF6310
- 如果使用 1 台计算机执行示例，例如，客户端和服务器分别运行在 2 个不同的 PLC Runtime 中，则这两个 PLC Runtime 都需要在不同的任务中运行
- 运行 .NET 示例客户端只需 .NET Framework 4.0

项目下载

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

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

项目描述

以下链接提供了这 3 个组件的文档。此外，另有一篇文章通过分步说明，详细介绍了如何运行 PLC 示例程序。

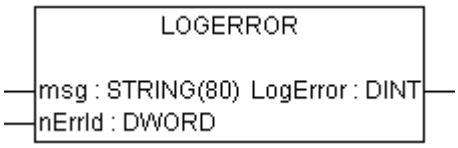
- TwinCAT 集成与测试 [▶ 60]（启动 PLC 示例）
- PLC 客户端 [▶ 62]（PLC 客户端文档：FB_LocalClient 功能块 [▶ 62]）
- PLC 服务器 [▶ 66]（PLC 服务器文档：FB_LocalServer 功能块 [▶ 66]）
- .NET 客户端 [▶ 71]（.NET 客户端文档：.NET 示例客户端 [▶ 71]）

PLC 示例项目的附加功能

下面简要介绍示例项目中使用的一些函数、常量和功能块：

LogError 函数

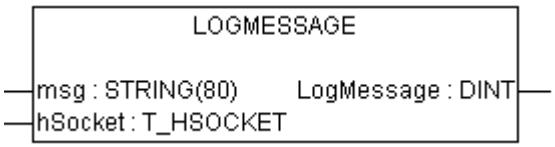
```
FUNCTION LogError : DINT
```



该函数会将带有错误代码的消息写入操作系统的日志（事件查看器）中。必须先将全局变量 bLogDebugMessages 设置为 TRUE。

LogMessage 函数

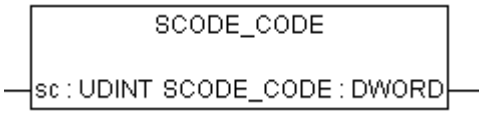
```
FUNCTION LogMessage : DINT
```



如果已经打开或关闭新的套接字，则该函数会向操作系统的日志（事件查看器）中写入消息。必须先将全局变量 bLogDebugMessages 设置为 TRUE。

SCODE_CODE 函数

```
FUNCTION SCODE_CODE : DWORD
```



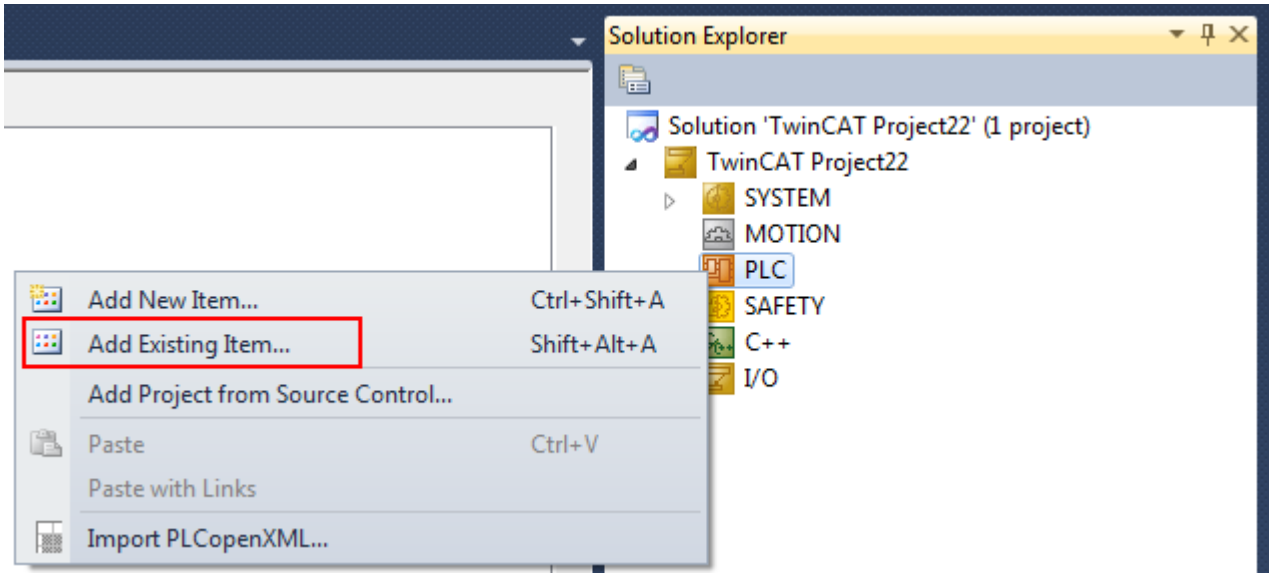
该函数会将 Win32 错误代码的最低 16 位进行掩码处理，然后返回这些位。

全局变量

名称	默认值	描述
bLogDebugMessages	TRUE	启用/禁用将消息写入操作系统日志的功能
MAX_CLIENT_CONNECTIONS	5	可同时与服务器连接的远程客户端的最大数量。
MAX_PLCPRJ_RXBUFFER_SIZE	1000	内部接收缓冲区的最大长度
PLCPRJ_RECONNECT_TIME	T#3s	一旦超时，本地客户端便会尝试重新建立与远程服务器的连接
PLCPRJ_SEND_CYCLE_TIME	T#1s	会将测试字符串定期从本地客户端周期性地发送到远程服务器
PLCPRJ_RECEIVE_POLLING_TIME	T#1s	客户端通过此循环从服务器读取（轮询）数据
PLCPRJ_RECEIVE_TIMEOUT	T#10s	如果在此期间未能接收到任何数据字节，则该时间段结束后，本地客户端将中止接收操作
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	示例项目错误代码：收到过多不带零终止符的字符
PLCPRJ_ERROR_RECEIVE_TIMEOUT	16#8102	示例项目错误代码：超时时间内无法接收新数据 (PLCPRJ_RECEIVE_TIMEOUT)

6.1.1.2 TwinCAT 集成与测试

下节将介绍如何准备和启动 PLC 服务器和客户端。PLC 示例以 TwinCAT 3 PLC 项目文件的形式提供。要将 PLC 项目导入 TwinCAT XAE，首先要创建新的 TwinCAT 3 Solution（解决方案）。然后在 PLC 节点的上下文菜单中选择 **Add Existing Item（添加现有项目）**命令，并在打开的对话框中选择已下载的示例文件（PLC 3.x 项目压缩文件 (*.tpzip)）作为文件类型。确认对话框后，则会将 PLC 项目添加到 Solution（解决方案）中。



PLC 服务器示例

在 TwinCAT XAE 中创建新的 TwinCAT 3 Solution（解决方案），并导入 TwinCAT TCP/IP Server 项目。选择目标系统。目标系统上也必须安装该功能组件，并已生成 TF6310 的授权。不要关闭 TwinCAT 3 Solution（解决方案）。

```
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
```

PLC 客户端示例

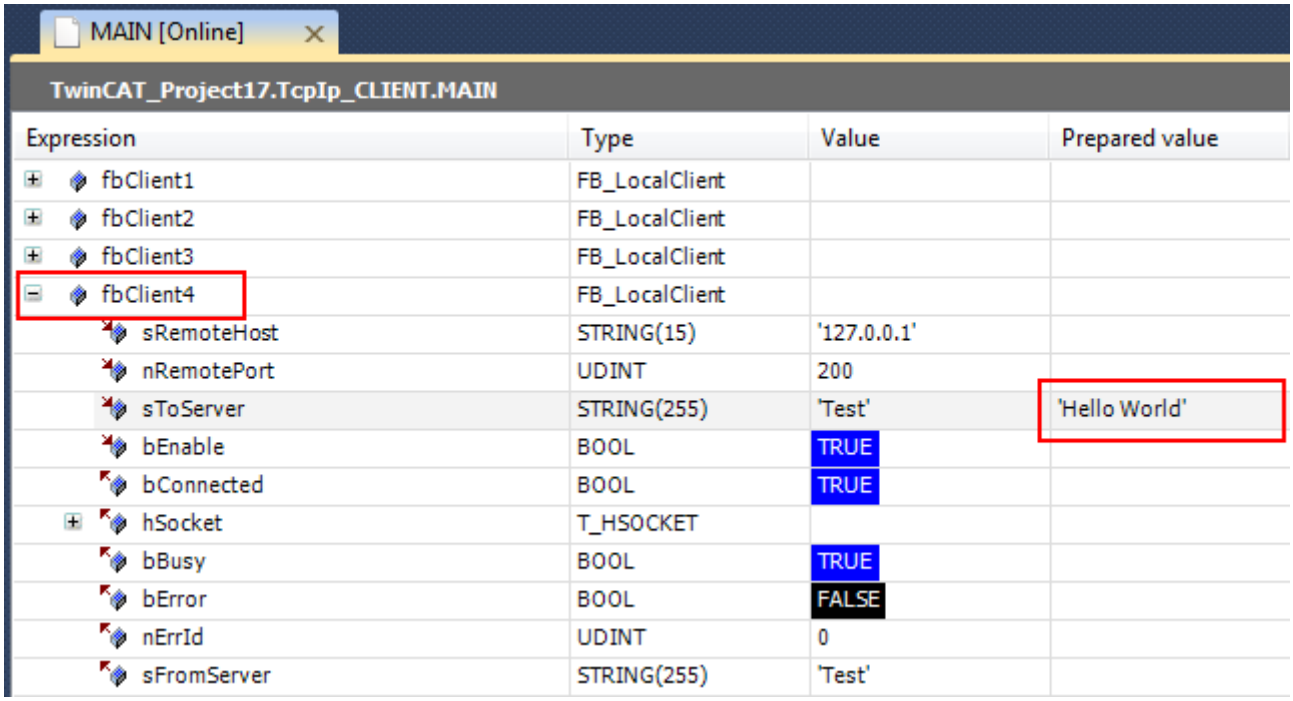
将 TCP/IP 客户端项目作为第二个 PLC 项目导入 TwinCAT 3 Solution（解决方案）。将此 PLC 项目关联到与服务器示例不同的任务。服务器的 IP 地址必须与系统相匹配（sRemoteHost 变量的初始化值）。在本例中，服务器位于同一 PC 上，因此输入 127.0.0.1。启用配置，然后登录并启动服务器，接着启动客户端 PLC 项目。

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

    bEnableClient1 : BOOL := TRUE;
    bEnableClient2 : BOOL := FALSE;
    bEnableClient3 : BOOL := FALSE;
    bEnableClient4 : BOOL := FALSE;
```

```
    bEnableClient5 : BOOL := FALSE;

    fbSocketCloseAll    : FB_SocketCloseAll := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    bCloseAll     : BOOL := TRUE;

    nCount     : UDINT;
END_VAR
```

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

IF NOT fbSocketCloseAll.bBusy THEN
    nCount := nCount + 1;
    fbClient1( bEnable := bEnableClient1, sToServer := CONCAT( 'CLIENT1-', UDINT_TO_STRING( nCount )
 ) );
    fbClient2( bEnable := bEnableClient2, sToServer := CONCAT( 'CLIENT2-', UDINT_TO_STRING( nCount )
 ) );
    fbClient3( bEnable := bEnableClient3, sToServer := CONCAT( 'CLIENT3-', UDINT_TO_STRING( nCount )
 ) );
    fbClient4( bEnable := bEnableClient4 );
    fbClient5( bEnable := bEnableClient5 );
END_IF
```

当设置其中一个 bEnableClientX 变量时，最多可启用 5 个客户端实例。每个客户端每秒向服务器发送一个字符串（默认：“TEST”）。服务器会向客户端（Echo服务器）返回相同的字符串。在测试中，前 3 个实例会自动生成 1 个带有计数器值的字符串。第一个客户端在程序启动时自动启用。将客户端项目中的 bEnableCilent4 变量设置为 TRUE。然后，新的客户端会尝试与服务器建立连接。如果成功，则会周期性发送“TEST”字符串。现在打开 FB_LocalClient 功能块的 fbClient4 实例。双击打开用于写入 sToString 变量的对话框，例如，将字符串变量的值改为“Hello”。



选择 **OK**（确定），关闭对话框。将新值强制输入 PLC。稍后便可以在线查看服务器返回的值。

MAIN [Online]

TwinCAT_Project17.TcpIp_CLIENT.MAIN

Expression	Type	Value	Prepared value
fbClient1	FB_LocalClient		
fbClient2	FB_LocalClient		
fbClient3	FB_LocalClient		
fbClient4	FB_LocalClient		
sRemoteHost	STRING(15)	'127.0.0.1'	
nRemotePort	UDINT	200	
sToServer	STRING(255)	'Hello World'	
bEnable	BOOL	TRUE	
bConnected	BOOL	TRUE	
hSocket	T_HSOCKET		
bBusy	BOOL	TRUE	
bError	BOOL	FALSE	
nErrId	UDINT	0	
sFromServer	STRING(255)	'Hello World'	

现在打开服务器项目中 FB_LocalServer 功能块的 fbServer 实例。在服务器的在线数据中可以看到字符串：“Hello”。

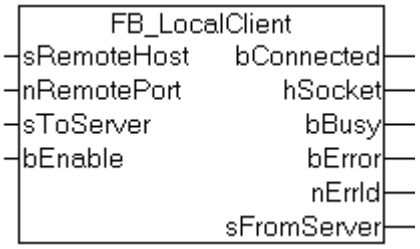
MAIN [Online]

TwinCAT_Project17.TcpIp_SERVER.MAIN

Expression	Type	Value	Prepared value
fbRemoteClient	ARRAY [1..MAX_CLI...		
fbRemoteClient[1]	FB_RemoteClient		
fbRemoteClient[2]	FB_RemoteClient		
fbRemoteClient[3]	FB_RemoteClient		
fbRemoteClient[4]	FB_RemoteClient		
hListener	T_HSOCKET		
bEnable	BOOL	TRUE	
bAccepted	BOOL	FALSE	
hSocket	T_HSOCKET		
bBusy	BOOL	TRUE	
bError	BOOL	FALSE	
nErrID	UDINT	0	
sFromClient	STRING(255)	'Hello World'	
fbAccept	FB_SocketAccept		

6.1.1.3 PLC 客户端

6.1.1.3.1 FB_LocalClient



如果已设置 bEnable 输入，则在 CLIENT_RECONNECT_TIME 计时结束后，系统会继续尝试建立与远程服务器的连接。远程服务器通过 sRemoteHost IP 地址和 nRemotePort IP 端口地址来识别。与服务器的数据交换封装在单独的功能块 FB_ClientDataExcha [▶ 64] 中。数据交换是循环的，始终在 PLCPRJ_SEND_CYCLE_TIME 计时结束后进行。sToServer 字符串变量已发送到服务器，服务器返回的字符串可在 sFormServer 输出端获得。另外，还可以根据需要对远程服务器进行寻址。如果出现错误，则会关闭现有连接，并建立新的连接。

接口

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

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

实施

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

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

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

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

```
                    bExecute    := TRUE );
            eStep := CLIENT_STATE_DATAEXCHA_WAIT;
        END_IF

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

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

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

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

6.1.1.3.2 FB_ClientDataExcha

如果 bExecute 输入出现上升沿，则会向远程服务器发送以空字符结尾的字符串，并读取远程服务器返回的字符串。功能块会尝试读取数据，直至检测到接收到的字符串为空终止符为止。如果超过 PLCPRJ_RECEIVE_TIMEOUT 超时时间或发生错误，则终止接收。如果在上一次读取尝试过程中无法读取新数据，则会在一定延迟时间过后，再次尝试读取数据。这样可以减少系统负载。

接口

```
FUNCTION_BLOCK FB_ClientDataExcha
VAR_INPUT
    hSocket     : T_HSOCKET;
    sToServer   : T_MaxString;
    bExecute    : BOOL;
END_VAR
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    sFromServer : T_MaxString;
END_VAR
VAR
    fbSocketSend : FB_SocketSend := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbSocketReceive : FB_SocketReceive := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbReceiveTON : TON;
    fbDisconnectTON : TON;
    RisingEdge : R_TRIG;
    eStep     : E_DataExchaSteps;
```

```
    cbReceived, startPos, endPos, idx : UDINT;
    cbFrame     : UDINT;
    rxBuffer    : ARRAY[0..MAX_PLCPRJ_RXBUFFER_SIZE] OF BYTE;
END_VAR
```

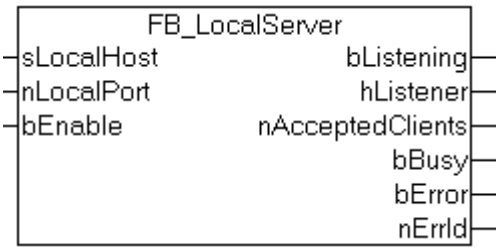
实施

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

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

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

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

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

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

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

6.1.1.4 PLC 服务器

6.1.1.4.1 FB_LocalServer



首先必须为服务器分配唯一的 sLocalHost IP 地址和 nLocaPort IP 端口号。如果已设置 bEnable 输入，则本地服务器会在 SERVER_RECONNECT_TIME 计时结束后反复尝试打开监听套接字。如果 TwinCAT TCP/IP Connection Server 定位在本地计算机上，则监听套接字通常可以在首次尝试时打开。远程客户端的功能封装在功能块 FB_RemoteClient [▶ 68] 中。一旦成功打开监听套接字，就会激活远程客户端实例。
FB_RemoteClient 的每个实例都对应 1 个远程客户端，本地服务器可同时与之通信。与服务器通信的远程客户端的最大数量可通过 MAX_CLIENT_CONNECTIONS 常量的值进行修改。如果出现错误，首先会关闭所有远程客户端连接，然后再关闭监听套接字。nAcceptedClients 输出提供了当前已连接客户端数量的信息。

接口

```
FUNCTION_BLOCK FB_LocalServer
VAR_INPUT
    sLocalHost       : STRING(15) := '127.0.0.1';(* own IP address! *)
    nLocalPort       : UDINT := 0;
    bEnable          : BOOL;
END_VAR
VAR_OUTPUT
    bListening       : BOOL;
    hListener        : T_HSOCKET;
    nAcceptedClients : UDINT;
    bBusy            : BOOL;
    bError           : BOOL;
    nErrId           : UDINT;
END_VAR
VAR
    fbListen         : FB_SocketListen := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbClose          : FB_SocketClose := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbConnectTON     : TON := ( PT := PLCPRJ_RECONNECT_TIME );
    eStep            : E_ServerSteps;
    fbRemoteClient   : ARRAY[1..MAX_CLIENT_CONNECTIONS ] OF FB_RemoteClient;
    i                : UDINT;
END_VAR
```

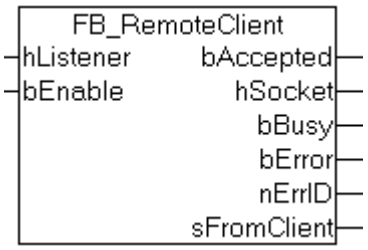
实施

```
CASE eStep OF

    SERVER_STATE_IDLE:
        IF bEnable XOR bListening THEN
```

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

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

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

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

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

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

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

    SERVER_STATE_ERROR:
        bError := TRUE;
```

```
            IF bListening THEN
                eStep := SERVER_STATE_REMOTE_CLIENTS_CLOSE;
            ELSE
                bBusy := FALSE;
                eStep := SERVER_STATE_IDLE;
            END_IF

END_CASE
```

6.1.1.4.2 FB_RemoteClient



如果设置了 bEnable 输入，则在 SERVER_ACCEPT_POOLING_TIME 计时结束后，会尝试接受远程客户端的连接请求。与远程客户端的数据交换封装在单独的功能块 FB_ServerDataExcha [▶ 69] 中。一旦成功建立连接，就会通过 FB_ServerDataExcha 功能块启用实例。如果出现错误，则会关闭已接受的连接，并建立新的连接。

接口

```
FUNCTION_BLOCK FB_RemoteClient
VAR_INPUT
    hListener       : T_HSOCKET;
    bEnable         : BOOL;
END_VAR
VAR_OUTPUT
    bAccepted       : BOOL;
    hSocket         : T_HSOCKET;
    bBusy           : BOOL;
    bError          : BOOL;
    nErrID          : UDINT;
    sFromClient     : T_MaxString;
END_VAR
VAR
    fbAccept            : FB_SocketAccept := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbClose             : FB_SocketClose := ( sSrvNetID := '', tTimeout := DEFAULT_ADS_TIMEOUT );
    fbServerDataExcha     : FB_ServerDataExcha;
    fbAcceptTON     : TON := ( PT := PLCPRJ_ACCEPT_POLLING_TIME );
    eStep           : E_ClientSteps;
END_VAR
```

实施

```
CASE eStep OF

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

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

    CLIENT_STATE_CONNECT_WAIT:
        fbAccept( bExecute := FALSE );
        IF NOT fbAccept.bBusy THEN
            IF NOT fbAccept.bError THEN
```

```
                IF fbAccept.bAccepted THEN
                    bAccepted := TRUE;
                    hSocket := fbAccept.hSocket;
                    LogMessage( 'REMOTE client ACCEPTED!', hSocket );
                END_IF
                eStep := CLIENT_STATE_IDLE;
            ELSE
                LogError( 'FB_SocketAccept', fbAccept.nErrId );
                nErrId := fbAccept.nErrId;
                eStep := CLIENT_STATE_ERROR;
                END_IF
        END_IF

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

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

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

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

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

END_CASE
```

6.1.1.4.3 FB_ServerDataExcha

如果 bExecute 输入端出现上升沿，则远程客户端会读取零终止字符串，并在检测到零终止符时将其返回给远程客户端。功能块会尝试读取数据，直至检测到接收到的字符串为零终止符为止。如果出现错误，或者在 PLCPRJ_RECEIVE_TIMEOUT 超时时间内没有收到新数据，则会中止接收。如果在上一次读取尝试过程中无法读取新数据，则会在一定延迟时间过后，再次尝试读取数据。这样可以减少系统负载。

接口

```
FUNCTION_BLOCK FB_ServerDataExcha
VAR_INPUT
    hSocket     : T_HSOCKET;
    bExecute    : BOOL;
```

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

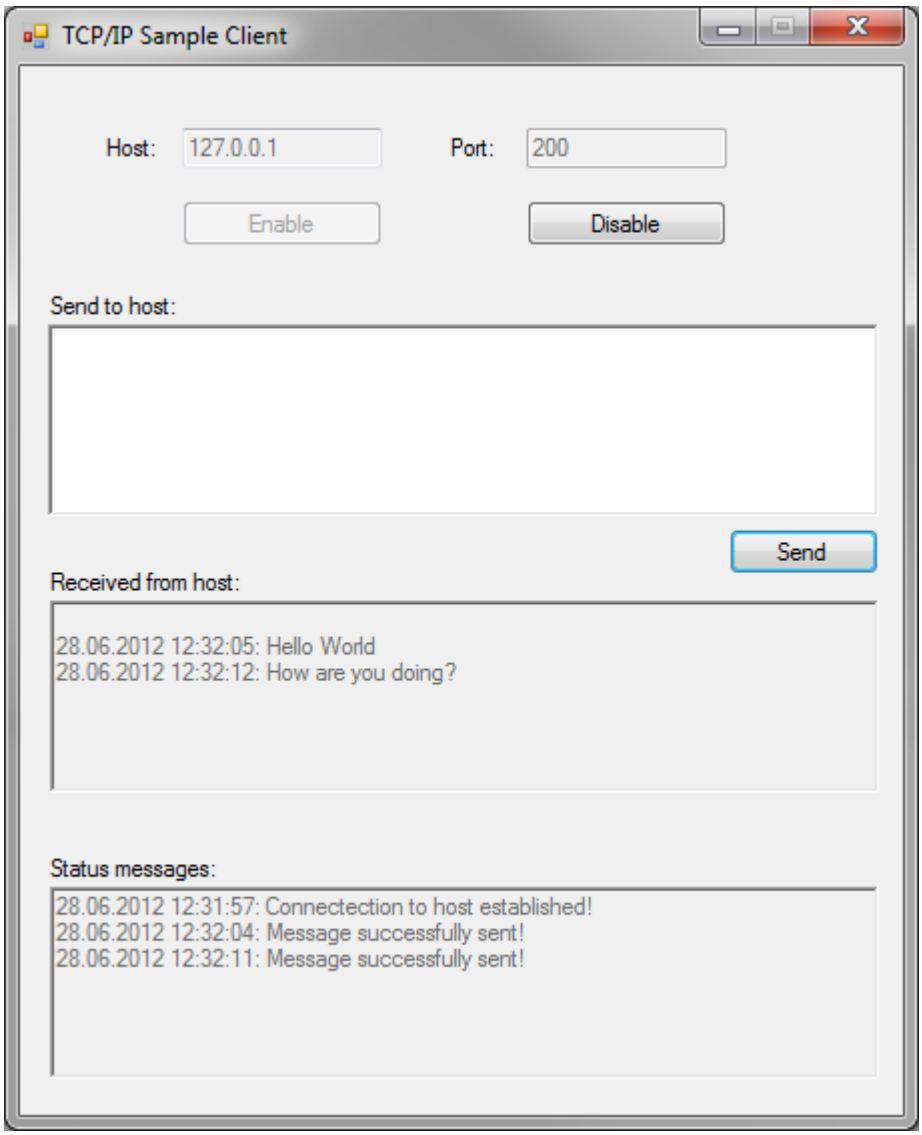
实施

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

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

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

    DATAEXCHA_STATE_RECEIVE_WAIT:
        fbSocketReceive( bExecute := FALSE );
           IF NOT fbSocketReceive.bBusy THEN
            IF NOT fbSocketReceive.bError THEN

                IF (fbSocketReceive.nRecBytes > 0) THEN(* bytes received *)

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

                    IF cbReceived < SIZEOF( sFromClient ) THEN(* no overflow *)

                        fbReceiveTON( IN := FALSE, PT := T#0s ); (* bytes received => increase the r
ead (polling) speed *)
                        fbDisconnectTON( IN := FALSE, PT := PLCPRJ_RECEIVE_TIMEOUT );
(* bytes received => disable timeout check *)

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

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

                ELSE(* no bytes received *)
                    fbReceiveTON( IN := FALSE, PT := PLCPRJ_RECEIVE_POLLING_TIME );
```

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

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

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

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

6.1.1.5 .NET 客户端

该示例项目展示了如何在 .NET4.0 下用 C# 实现 PLC TCP/IP 服务器客户端。



该示例使用的是 .NET 库 System.Net 和 System.Net.Sockets，程序员可以轻松使用套接字函数。按下 **Enable（启用）** 键会促使应用程序周期性地尝试与服务器连接建立（具体取决于 TIMERTICK 的值，单位为 [ms]）。如果成功，可通过 Send（发送）按钮向服务器发送最大长度为 255 个字符的字符串。然后，服务器会接受该字符串并将其发送回客户端。在服务器示例中定义的 SERVER_RECEIVE_TIMEOUT 时间（默认值：50 秒）超时后，如果服务器在此时间内无法从客户端接收到任何新数据，则服务器端会自动关闭连接。

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

/* ######################################################################################
 * This sample TCP/IP client connects to a TCP/IP-Server, sends a message and waits for the
 * response. It is being delivered together with our TCP-Sample, which implements an echo server
 * in PLC.
 * ###################################################################################### */
namespace TcpIpServer_SampleClient
{
 publicpartialclassForm1 : Form
 {
    /
* ######################################################################################
    * Constants
    * ###################################################################################### */
privateconstint RCVBUFFERSIZE = 256; // buffer size for receive bufferprivateconststring DEFAULTIP =
 "127.0.0.1";
    privateconststring DEFAULTPORT = "200";
    privateconstint TIMERTICK = 100;

    /
* ######################################################################################
```

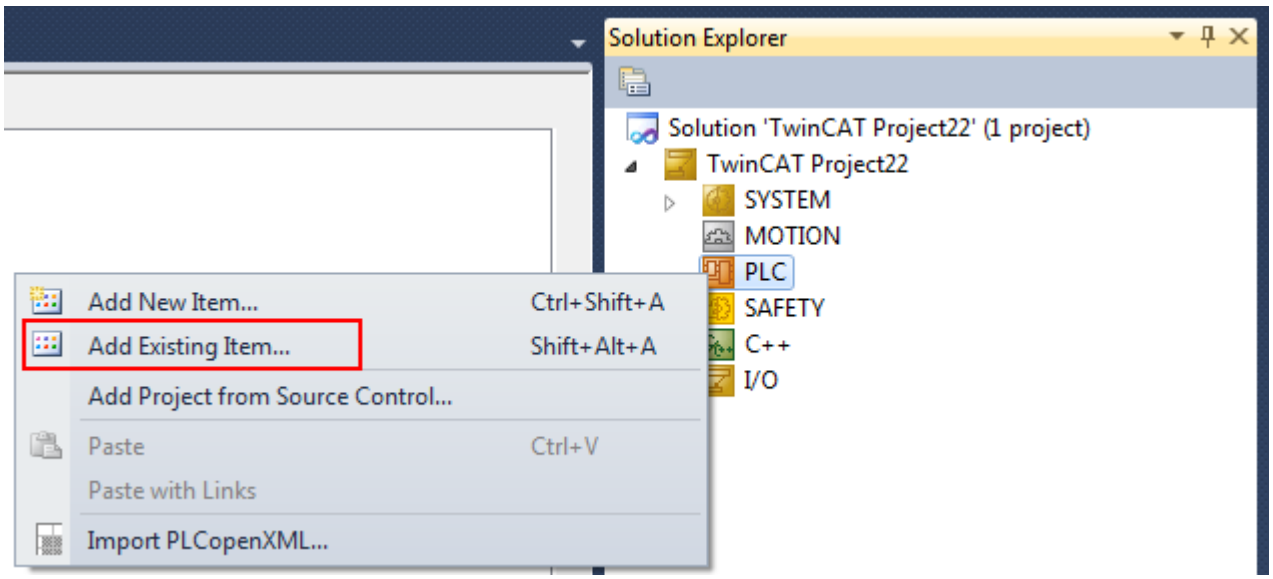
```
    * Global variables
    * ########################################################################################## */
privatestaticbool _isConnected; // signals whether socket connection is active or notprivatestaticSo
cket _socket; // object used for socket connection to TCP/IP-
ServerprivatestaticIPEndPoint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

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

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

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

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

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

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

    privatevoid cmd_send_Click(object sender, EventArgs e)
    {
     /
```

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

     /
* ##########################################################################################
       * Send buffer content via TCP/IP connection
     * ########################################################################################## */
try
     {
        int send = _socket.Send(sendBuffer);
        if (send == 0)
         thrownewException();
        else
        {
         /
* ##########################################################################################

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

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

6.1.2 Sample02：“Echo”客户端/服务器

本示例会使用原 TcSocketHelper.Lib 的函数，该函数现已集成到 Tc2_TcpIp 库中。该示例展示的是基于原 SocketHelper 库函数的客户端/服务器 PLC 应用程序。

客户端会向远程服务器周期性地发送测试字符串 (sToServer)。服务器会将该字符串原封不动地返回给客户端 (sFromServer)。



系统要求

- TwinCAT 3 Build 3093 或更高版本
- TwinCAT 3 功能组件 TF6310 TCP/IP
- 如果使用 2 台计算机执行本示例（1 台客户端和 1 台服务器），则需要在 2 台计算机上安装功能组件 TF6310
- 如果使用 1 台计算机执行示例，例如，客户端和服务器分别运行在 2 个不同的 PLC Runtime 中，则这两个 PLC Runtime 都需要在不同的任务中运行。

项目下载

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

项目信息

上述示例中使用的默认通信设置如下：

- PLC 客户端应用程序：远程服务器的端口和 IP 地址：200、“127.0.0.1”
- PLC 服务器应用程序：本地服务器的端口和 IP 地址：200、“127.0.0.1”

要在 2 台不同的 PC 上测试客户端和服务器应用程序，必须相应调整端口和 IP 地址。

然而，您也可以在单一计算机上用默认值测试客户端和服务器示例，方法是将客户端应用程序加载到第一个 PLC Runtime 系统中，将服务器应用程序加载到第二个 PLC Runtime 系统中。

PLC 项目示例的行为由以下全局变量/常量决定。

常量	值	描述
PLCPRJ_MAX_CONNECTIONS	5	服务器 → 客户端的最大连接数。1 个服务器可与多个客户端建立连接。1 个客户端一次只能与 1 个服务器建立连接。
PLCPRJ_SERVER_RESPONSE_TIMEOUT	T#10s	服务器应向客户端发送响应的最长延迟时间（超时时间）。
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	客户端向服务器发送数据 (TX) 所依据的周期时间。
PLCPRJ_RECEIVER_POLLING_CYCLE_TIME	T#200ms	客户端或服务器轮询接收数据 (RX) 所依据的周期时间。
PLCPRJ_BUFFER_SIZE	10000	RX/TX 数据的最大内部缓冲区大小。

PLC 示例定义并使用以下内部错误代码：

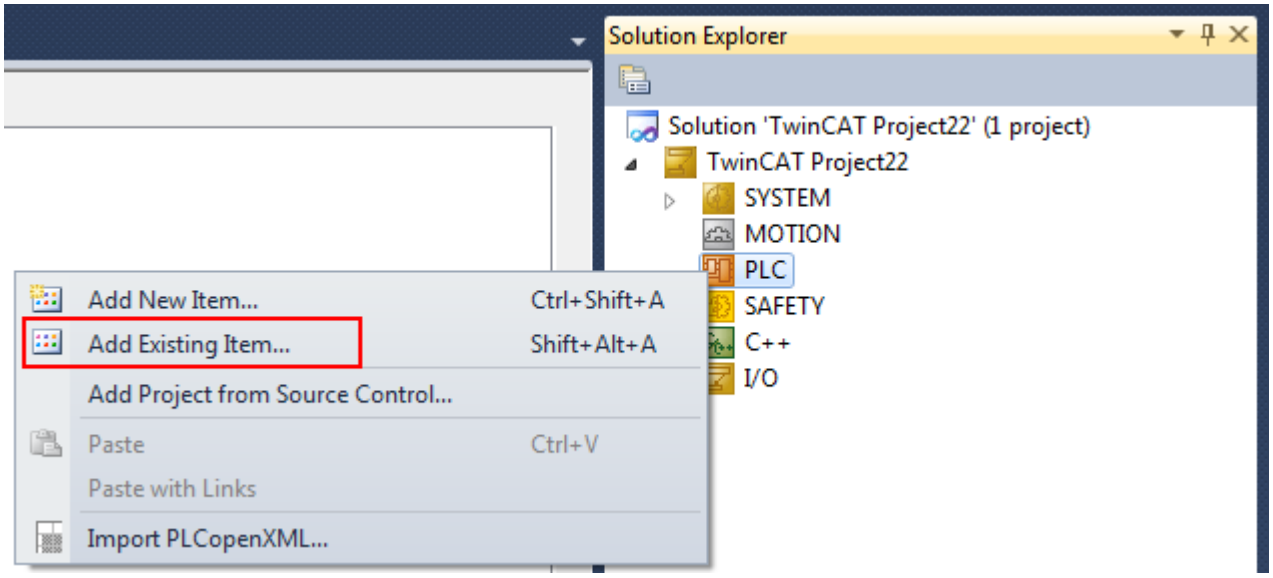
错误代码	值	描述
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	内部接收缓冲区报告溢出。
PLCPRJ_ERROR_SEND_BUFFER_OVERFLOW	16#8102	内部发送缓冲区报告溢出。
PLCPRJ_ERROR_RESPONSE_TIMEOUT	16#8103	服务器在指定超时时间内未发送响应。
PLCPRJ_ERROR_INVALID_FRAME_FORMAT	16#8104	报文格式不正确（大小、数据字节错误等）。

客户端和服务器应用程序（FB_ServerApplication、FB_ClientApplication）以功能块的形式实现。因此，应用程序和连接可以反复实例化。

6.1.3 Sample03：“Echo”客户端/服务器

本示例会使用原 TcSocketHelper.Lib 的函数，该函数现已集成到 Tc2_TcpIp 库中。该示例展示的是基于原 SocketHelper 库函数的客户端/服务器 PLC 应用程序。

客户端会向远程服务器周期性地发送测试字符串 (sToServer)。服务器会将该字符串原封不动地返回给客户端 (sFromServer)。该示例与 Sample02 的区别在于，服务器最多可建立 5 个连接，客户端应用程序最多可启动 5 个客户端实例。每个实例都会与服务器建立连接。



系统要求

- TwinCAT 3 Build 3093 或更高版本
- TwinCAT 3 功能组件 TF6310 TCP/IP
- 如果使用 2 台计算机执行本示例（1 台客户端和 1 台服务器），则需要在 2 台计算机上安装功能组件 TF6310
- 如果使用 1 台计算机执行示例，例如，客户端和服务器分别运行在 2 个不同的 PLC Runtime 中，则这两个 PLC Runtime 都需要在不同的任务中运行

项目下载

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

项目信息

上述示例中使用的默认通信设置如下：

- PLC 客户端应用程序：远程服务器的端口和 IP 地址：200、“127.0.0.1”

- PLC 服务器应用程序：本地服务器的端口和 IP 地址：200、“127.0.0.1”

要在 2 台不同的 PC 上测试客户端和服务器应用程序，必须相应调整端口和 IP 地址。

然而，您也可以在单一计算机上用默认值测试客户端和服务器示例，方法是将客户端应用程序加载到第一个 PLC Runtime 系统中，将服务器应用程序加载到第二个 PLC Runtime 系统中。

PLC 项目示例的行为由以下全局变量/常量决定。

常量	值	描述
PLCPRJ_MAX_CONNECTIONS	5	服务器->客户端的最大连接数。1 个服务器可与多个客户端建立连接。1 个客户端一次只能与 1 个服务器建立连接。
PLCPRJ_SERVER_RESPONSE_TIMEOUT	T#10s	服务器应向客户端发送响应的最长延迟时间（超时时间）。
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	客户端向服务器发送数据 (TX) 所依据的周期时间。
PLCPRJ_RECEIVER_POLLING_CYCLE_TIME	T#200ms	客户端或服务器轮询接收数据 (RX) 所依据的周期时间。
PLCPRJ_BUFFER_SIZE	10000	RX/TX 数据的最大内部缓冲区大小。

PLC 示例定义并使用以下内部错误代码：

错误代码	值	描述
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	内部接收缓冲区报告溢出。
PLCPRJ_ERROR_SEND_BUFFER_OVERFLOW	16#8102	内部发送缓冲区报告溢出。
PLCPRJ_ERROR_RESPONSE_TIMEOUT	16#8103	服务器在指定超时时间内未发送响应。
PLCPRJ_ERROR_INVALID_FRAME_FORMAT	16#8104	报文格式不正确（大小、数据字节错误等）。

客户端和服务器应用程序（FB_ServerApplication、FB_ClientApplication）以功能块的形式实现。因此，应用程序和连接可以反复实例化。

6.1.4 Sample04：二进制数据交换

本示例会使用原 TcSocketHelper.Lib 的函数，该函数现已集成到 Tc2_TcpIp 库中。该示例展示的是基于原 SocketHelper 库函数的客户端/服务器 PLC 应用程序。

该示例提供了用于交换二进制数据的客户端 – 服务器应用程序。为此已实现一个简单的示例协议。已发送和已接收报文的二进制数据长度和帧计数器会在协议头中传输。

二进制数据的结构由 PLC 结构 ST_ApplicationBinaryData 定义。二进制数据追加至协议头后一并传输。二进制结构的实例在客户端称为 toServer、fromServer，在服务器端则称为 toClient、fromClient。

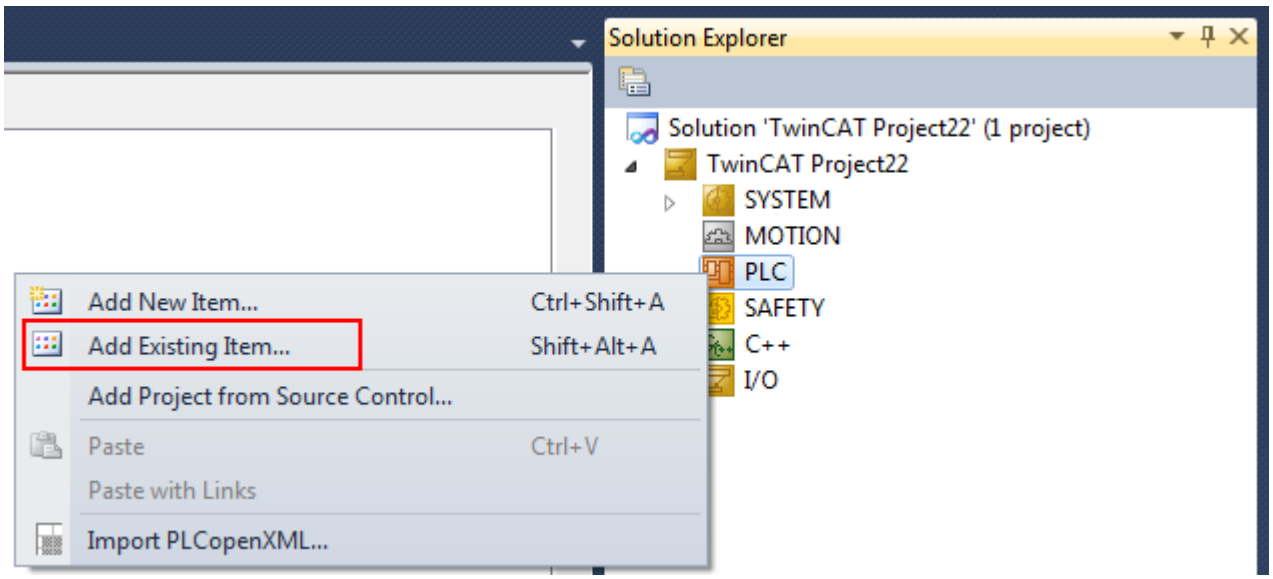
客户端和服务器端的结构声明可根据需要进行调整。两端的结构声明必须完全一致。

结构的大小不得超过收发 FIFO 缓冲区的最大容量。最大缓冲区大小由常量定义。

服务器功能会在功能块 FB_ServerApplication 中实现，而客户端功能则在功能块 FB_ClientApplication 中实现。

在标准实现中，客户端会周期性地向服务器发送二进制结构的数据，并等待服务器的响应。服务器会修改一些数据并将其返回至客户端。

如果需要特定功能，则必须相应修改功能块 FB_ServerApplication 和 FB_ClientApplication。



系统要求

- TwinCAT 3 Build 3093 或更高版本
- TwinCAT 3 功能组件 TF6310 TCP/IP
- 如果使用 2 台计算机执行本示例（1 台客户端和 1 台服务器），则需要在 2 台计算机上安装功能组件 TF6310
- 如果使用 1 台计算机执行示例，例如，客户端和服务器分别运行在 2 个不同的 PLC Runtime 中，则这两个 PLC Runtime 都需要在不同的任务中运行。

项目下载

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

项目信息

上述示例中使用的默认通信设置如下：

- PLC 客户端应用程序：远程服务器的端口和 IP 地址：200、“127.0.0.1”
- PLC 服务器应用程序：本地服务器的端口和 IP 地址：200、“127.0.0.1”

要在 2 台不同的 PC 上测试客户端和服务器应用程序，必须相应调整端口和 IP 地址。

然而，您也可以在单一计算机上用默认值测试客户端和服务器示例，方法是将客户端应用程序加载到第一个 PLC Runtime 系统中，将服务器应用程序加载到第二个 PLC Runtime 系统中。

PLC 项目示例的行为由以下全局变量/常量决定。

常量	值	描述
PLCPRJ_MAX_CONNECTIONS	5	服务器->客户端的最大连接数。1 个服务器可与多个客户端建立连接。1 个客户端一次只能与 1 个服务器建立连接。
PLCPRJ_SERVER_RESPONSE_TIMEOUT	T#10s	服务器应向客户端发送响应的最长延迟时间（超时时间）。
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	客户端向服务器发送数据 (TX) 所依据的周期时间。
PLCPRJ_RECEIVER_POLLING_CYCLE_TIME	T#200ms	客户端或服务器轮询接收数据 (RX) 所依据的周期时间。
PLCPRJ_BUFFER_SIZE	10000	RX/TX 数据的最大内部缓冲区大小。

PLC 示例定义并使用以下内部错误代码：

错误代码	值	描述
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	内部接收缓冲区报告溢出。
PLCPRJ_ERROR_SEND_BUFFER_OVERFLOW	16#8102	内部发送缓冲区报告溢出。
PLCPRJ_ERROR_RESPONSE_TIMEOUT	16#8103	服务器在指定超时时间内未发送响应。
PLCPRJ_ERROR_INVALID_FRAME_FORMAT	16#8104	报文格式不正确（大小、数据字节错误等）。

客户端和服务器应用程序（FB_ServerApplication、FB_ClientApplication）以功能块的形式实现。因此，应用程序和连接可以反复实例化。

6.1.5 Sample05：二进制数据交换

本示例会使用原 TcSocketHelper.Lib 的函数，该函数现已集成到 Tc2_TcpIp 库中。该示例展示的是基于原 SocketHelper 库函数的客户端/服务器 PLC 应用程序。

该示例提供了用于交换二进制数据的客户端 – 服务器应用程序。为此已实现一个简单的示例协议。已发送和已接收报文的二进制数据长度和帧计数器会在协议头中传输。

二进制数据的结构由 PLC 结构 ST_ApplicationBinaryData 定义。二进制数据追加至协议头后一并传输。二进制结构的实例在客户端称为 toServer、fromServer，在服务器端则称为 toClient、fromClient。

客户端和服务器端的结构声明可根据需要进行调整。两端的结构声明必须完全一致。

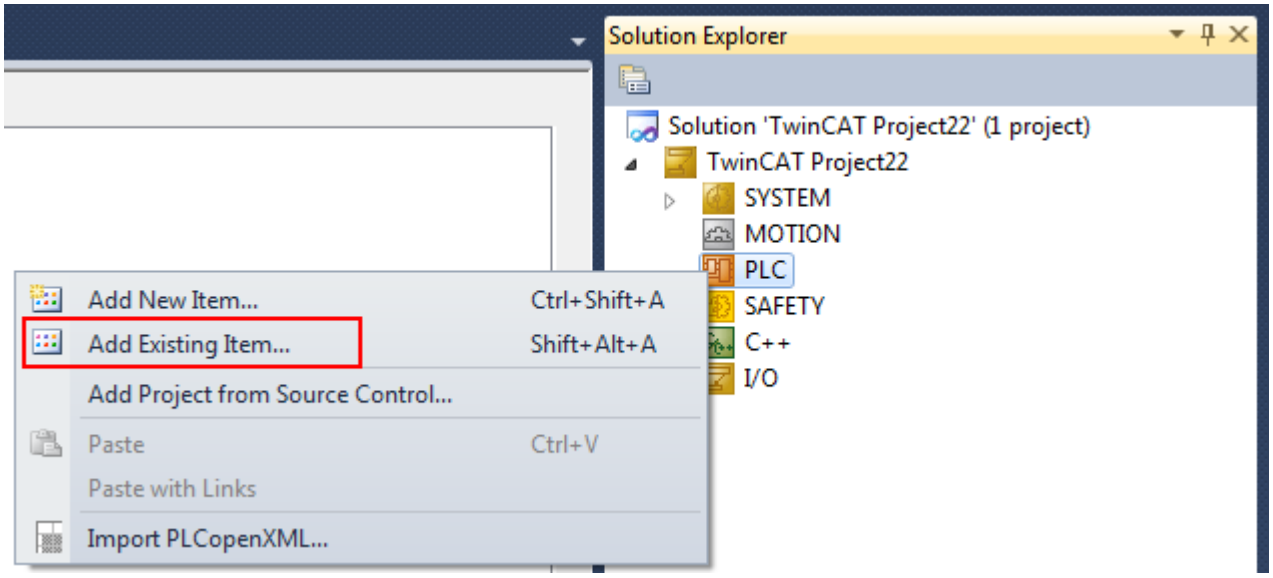
结构的大小不得超过收发 FIFO 缓冲区的最大容量。最大缓冲区大小由常量定义。

服务器功能会在功能块 FB_ServerApplication 中实现，而客户端功能则在功能块 FB_ClientApplication 中实现。

在标准实现中，客户端会周期性地向服务器发送二进制结构的数据，并等待服务器的响应。服务器会修改一些数据并将其返回至客户端。

如果需要特定功能，则必须相应修改功能块 FB_ServerApplication 和 FB_ClientApplication。

该示例与 Sample04 的区别在于，服务器最多可建立 5 个连接，客户端应用程序最多可拥有 5 个客户端实例。每个实例都会与服务器建立连接。



系统要求

- TwinCAT 3 Build 3093 或更高版本

- TwinCAT 3 功能组件 TF6310 TCP/IP
- 如果使用 2 台计算机执行本示例（1 台客户端和 1 台服务器），则需要在 2 台计算机上安装功能组件 TF6310
- 如果使用 1 台计算机执行示例，例如，客户端和服务器分别运行在 2 个不同的 PLC Runtime 中，则这两个 PLC Runtime 都需要在不同的任务中运行。

项目下载

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

项目信息

上述示例中使用的默认通信设置如下：

- PLC 客户端应用程序：远程服务器的端口和 IP 地址：200、“127.0.0.1”
- PLC 服务器应用程序：本地服务器的端口和 IP 地址：200、“127.0.0.1”

要在 2 台不同的 PC 上测试客户端和服务器应用程序，必须相应调整端口和 IP 地址。

然而，您也可以在单一计算机上用默认值测试客户端和服务器示例，方法是将客户端应用程序加载到第一个 PLC Runtime 系统中，将服务器应用程序加载到第二个 PLC Runtime 系统中。

PLC 项目示例的行为由以下全局变量/常量决定。

常量	值	描述
PLCPRJ_MAX_CONNECTIONS	5	服务器->客户端的最大连接数。1 个服务器可与多个客户端建立连接。1 个客户端一次只能与 1 个服务器建立连接。
PLCPRJ_SERVER_RESPONSE_TIMEOUT	T#10s	服务器应向客户端发送响应的最长延迟时间（超时时间）。
PLCPRJ_CLIENT_SEND_CYCLE_TIME	T#1s	客户端向服务器发送数据 (TX) 所依据的周期时间。
PLCPRJ_RECEIVER_POLLING_CYCLE_TIME	T#200ms	客户端或服务器轮询接收数据 (RX) 所依据的周期时间。
PLCPRJ_BUFFER_SIZE	10000	RX/TX 数据的最大内部缓冲区大小。

PLC 示例定义并使用以下内部错误代码：

错误代码	值	描述
PLCPRJ_ERROR_RECEIVE_BUFFER_OVERFLOW	16#8101	内部接收缓冲区报告溢出。
PLCPRJ_ERROR_SEND_BUFFER_OVERFLOW	16#8102	内部发送缓冲区报告溢出。
PLCPRJ_ERROR_RESPONSE_TIMEOUT	16#8103	服务器在指定超时时间内未发送响应。
PLCPRJ_ERROR_INVALID_FRAME_FORMAT	16#8104	报文格式不正确（大小、数据字节错误等）。

客户端和服务器应用程序（FB_ServerApplication、FB_ClientApplication）以功能块的形式实现。因此，应用程序和连接可以反复实例化。

6.1.6 Sample06：基于 TLS 的“Echo”客户端/服务器（基本模块）

下方示例基本上以 Sample01 为基础，展示了“Echo”客户端/服务器系统的示例实现。客户端每隔一定时间（如每秒）向服务器发送 1 个测试字符串。远程服务器会将此字符串发回客户端。

与 Sample01 不同的是，本示例中的通信连接是通过 TLS 与客户端/服务器端证书来确保安全的。证书并不属于示例内容，必须由用户创建。

简言之，本示例实质上说明了功能块 FB_TlsSocketConnect [▶ 34]、FB_TlsSocketCreate [▶ 37]、FB_TlsSocketListen [▶ 36]、FB_TlsSocketAddCa [▶ 38]、FB_TlsSocketAddCrl [▶ 39] 和 FB_TlsSocketSetCert [▶ 39] 的使用方法。这些功能块都相应地集成到了 Sample01 中客户端和服务器示例的状态机中。

项目下载

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

6.1.7 Sample07：基于 TLS-PSK 的“Echo”客户端/服务器（基本模块）

下方示例基本上以 Sample01 为基础，展示了“Echo”客户端/服务器系统的示例实现。客户端每隔一定时间（如每秒）向服务器发送 1 个测试字符串。远程服务器会将此字符串发回客户端。

与 Sample01 不同的是，本示例中的通信连接是通过带有预共享密钥 (PSK) 的 TLS 加密的。

简言之，本示例实质上说明了功能块 FB_TlsSocketConnect [▶ 34]、FB_TlsSocketCreate [▶ 37]、FB_TlsSocketListen [▶ 36] 和 FB_TlsSocketSetPsk [▶ 40] 的使用方法。这些功能块都相应地集成到了 Sample01 中客户端和服务器示例的状态机中。

项目下载

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

6.2 UDP

6.2.1 Sample01：点对点通信

6.2.1.1 概述

下方示例展示的是如何在 PLC 中实现简单点对点应用程序。该示例包含 2 个 PLC 项目（PeerA 和 PeerB），以及 1 个同样作为独立节点运行的 .NET 应用程序。所有点对点应用程序都会向远程点对点程序发送测试字符串，同时从远程点对点程序接收字符串。接收到的字符串会显示在目标计算机显示器的消息框中。请根据自身需求自由使用并自定义该示例。

系统要求

- TwinCAT 3 Build 3093 或更高版本
- TwinCAT 3 功能组件 TF6310 TCP/IP
- 如果使用两台计算机执行示例，则需要在两台计算机上安装功能组件 TF6310
- 如果使用一台计算机来执行示例，例如在两个独立的 PLC Runtime 中分别运行 Peer A 和 Peer B，则这两个 PLC Runtime 需要在不同的任务中运行
- 运行 .NET 示例客户端只需 .NET Framework 4.0

项目下载

这两台 PLC 设备的信号源仅在远程通信伙伴的 IP 地址配置上有所不同。

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

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

项目描述

以下链接提供了各组件的文档。此外，另有一篇文章通过分步说明，详细介绍了如何运行 PLC 示例程序。

- TwinCAT 集成与测试 [▶ 83]（启动 PLC 示例）

- PLC 设备 A 和 B [▶ 84]（点对点 PLC 应用程序）
- .NET 通信 [▶ 87]（.NET 示例客户端）

PLC 示例项目的附加功能

PLC 示例中使用了一些函数、常量和功能块，现简要介绍如下：

Fifo 功能块

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

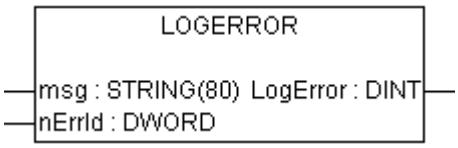
这是一个简单的 Fifo 功能块。该功能块的其中一个实例用作“发送 Fifo”，另一个实例则用作“接收 Fifo”。要发送的消息存储在“发送 Fifo”中，而接收到的消息则存储在“接收 Fifo 中”。如果在最后一次操作（AddTail 或 RemoveHead）过程中发生错误（Fifo 为空或过满），则 bOk 输出变量会设为 FALSE。

Fifo 条目由以下组件组成：

```
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 函数

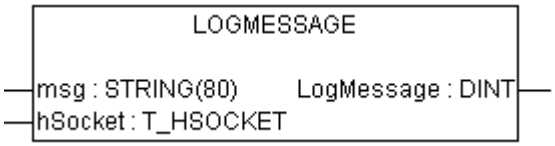
```
FUNCTION LogError : DINT
```



该函数会将带有错误代码的消息写入操作系统的日志（事件查看器）中。必须先将全局变量 bLogDebugMessages 设置为 TRUE。

LogMessage 函数

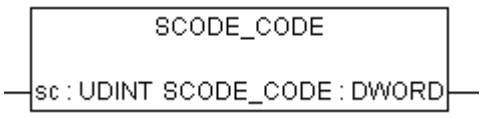
```
FUNCTION LogMessage : DINT
```



如果已经打开或关闭新的套接字，则该函数会向操作系统的日志（事件查看器）中写入消息。必须先将全局变量 bLogDebugMessages 设置为 TRUE。

SCODE_CODE 函数

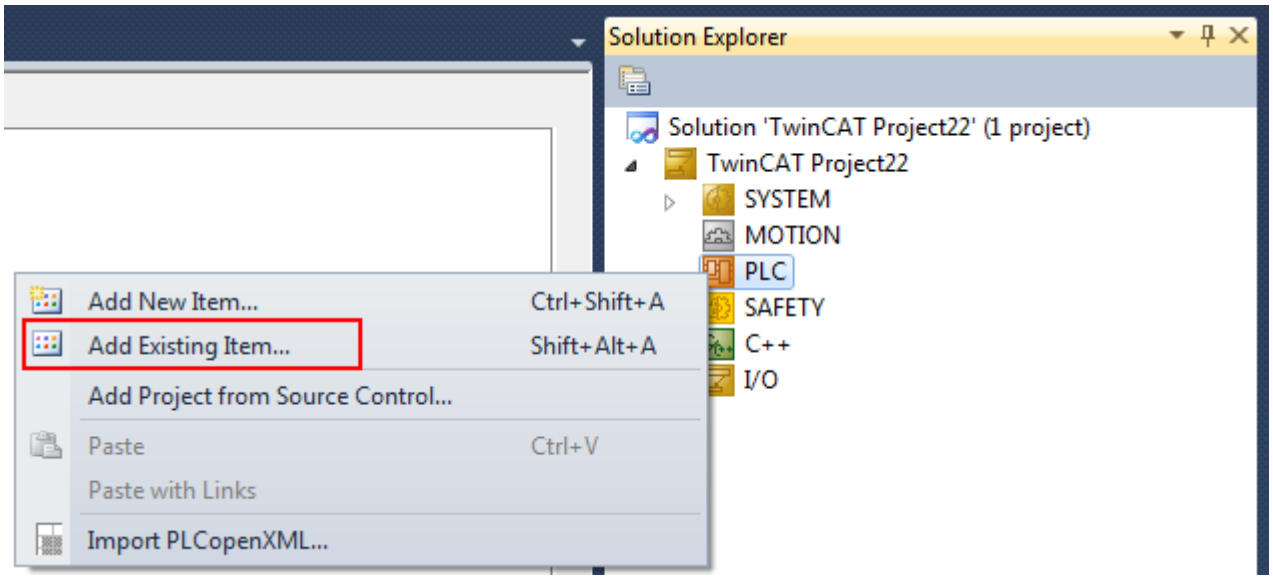
```
FUNCTION SCODE_CODE : DWORD
```



该函数会将 Win32 错误代码的最低 16 位进行掩码处理，然后返回这些位。

6.2.1.2 TwinCAT 集成与测试

PLC 示例以 TwinCAT 3 PLC 项目文件的形式提供。要将 PLC 项目导入 TwinCAT XAE，首先要创建新的 TwinCAT 3 Solution（解决方案）。然后在 PLC 节点的上下文菜单中选择 **Add Existing Item（添加现有项目）**命令，并在打开的对话框中选择已下载的示例文件（PLC 3.x 项目压缩文件 (*.tpzip)）作为文件类型。确认对话框后，则会将 PLC 项目添加到 Solution（解决方案）中。



启动示例需要 2 台计算机。另外，也可以在 1 台电脑上使用 2 种 Runtime 系统进行测试。包含通信伙伴端口号和 IP 地址的常量必须进行相应修改。

2 台计算机的配置示例：

- 设备 A 定位在本地 PC 上，其 IP 地址为“10.1.128.21”
- 设备 B 定位在远程 PC 上，其 IP 地址为“172.16.6.195”。

设备 A

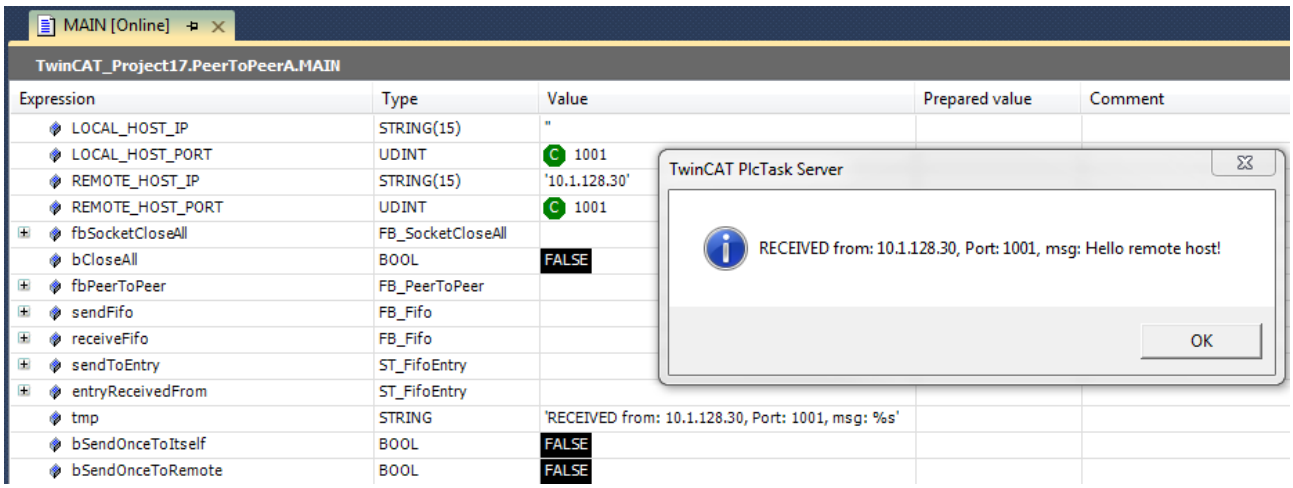
请执行以下步骤，在设备 A 上配置示例：

- 在 TwinCAT XAE 中创建新的 TwinCAT 3 解决方案，并导入设备 A 的 点对点 PLC 项目。
- 将 POU MAIN 中的常量 REMOTE_HOST_IP 设置为远程系统（设备 B – 在我们的示例中为“10.1.128.”）的真实 IP 地址。
- 激活配置，启动 PLC Runtime。(不要忘记为 TF6310 TCP/IP 创建授权)

设备 B

在设备 B 上安装示例的步骤如下：

- 在 TwinCAT XAE 中创建新的 TwinCAT 3 解决方案，并导入设备 B 的点对点 PLC 项目。
- 将 POU MAIN 中的 REMOTE_HOST_IP 常量调整为设备 A 的实际 IP 地址（在我们的示例中为“10.1.128.21”）。
- 启用配置，启动 PLC Runtime。(不要忘记提前为 TF6310 TCP/IP 生成授权)
- 登录 PLC 运行时，在 POU “MAIN”中将 Boolean 变量 bSendOnceToRemote 设为 TRUE。
- 随后设备 A 上会出现消息框。现在可以在设备 A 上重复此步骤。现在设备 B 上应该会出现消息框。

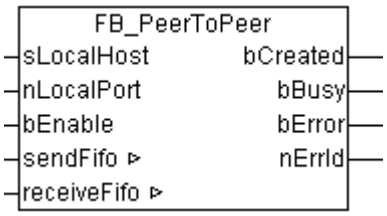


6.2.1.3 PLC 设备 A 和 B

所需功能已封装在功能块 FB_PeerToPeer 中。每个通信伙伴都使用 1 个 FB_PeerToPeer 功能块实例。功能块通过 bEnable 输入端的上升沿激活。新的 UDP 套接字打开，开启数据交换。套接字地址通过变量 sLocalHost 和 nLocalPort 指定。下降沿停止数据交换并关闭套接字。待发送的数据通过引用 (VAR_IN_OUT) 经由变量 sendFifo 传输到程序块。接收到的数据存储在变量 receiveFifo 中。

名称	默认值	描述
g_sTcIpConnSvrAddr	''	TwinCAT TCP/IP Connection Server 的网络地址。默认值：空字符串（服务器定位在本地 PC 上）；
bLogDebugMessages	TRUE	启用/禁用将消息写入操作系统日志的功能；
PLCPRJ_ERROR_SENDFIFO_OVERFLOW	16#8103	示例项目错误代码：“发送 Fifo”已满。
PLCPRJ_ERROR_RECFIFO_OVERFLOW	16#8104	项目错误代码示例：“接收 Fifo”已满。

FUNCTION_BLOCK FB_PeerToPeer



接口

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

```
    sendTo      : ST_FifoEntry;
    receivedFrom : ST_FifoEntry;
END_VAR
```

实施

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

    UDP_STATE_CREATE_START:
        fbCreate( bExecute := FALSE );
        fbCreate( sSrvNetId:= g_sTcIpConnSvrAddr,
                sLocalHost:= sLocalHost,
            nLocalPort:= nLocalPort,
            bExecute:= TRUE );
        eStep := UDP_STATE_CREATE_WAIT;

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

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

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

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

    UDP_STATE_RECEIVE_WAIT:
        fbReceiveFrom( bExecute := FALSE );
        IF NOT fbReceiveFrom.bBusy THEN
            IF NOT fbReceiveFrom.bError THEN
                IF fbReceiveFrom.nRecBytes > 0 THEN
                    receivedFrom.nRemotePort := fbReceiveFrom.nRemotePort;
                    receivedFrom.sRemoteHost := fbReceiveFrom.sRemoteHost;
                    receiveFifo.AddTail( new := receivedFrom );
                    IF NOT receiveFifo.bOk THEN(* Check for fifo overflow *)
```

```
                    LogError( 'Receive fifo overflow!', PLCPRJ_ERROR_RECFIFO_OVERFLOW );
                END_IF
            END_IF
            eStep := UDP_STATE_IDLE;
        ELSIF fbReceiveFrom.nErrId = 16#80072746 THEN
                LogError( 'The connection is reset by remote side.', fbReceiveFrom.nErrId );
            eStep := UDP_STATE_IDLE;
        ELSE
            LogError( 'FB_SocketUdpReceiveFrom (UDP client/server)', fbReceiveFrom.nErrId );
            nErrId := fbReceiveFrom.nErrId;
            eStep := UDP_STATE_ERROR;
        END_IF
    END_IF

UDP_STATE_CLOSE_START:
    fbClose( bExecute := FALSE );
    fbClose(    sSrvNetId:= g_sTcIpConnSvrAddr,
        hSocket:= hSocket,
        bExecute:= TRUE );
    eStep := UDP_STATE_CLOSE_WAIT;

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

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

MAIN 程序

程序下载或 PLC 复位后，必须关闭之前打开的套接字。在 PLC 启动过程中，可通过调用 FB_SocketCloseAll [▶ 23] 功能块的实例来实现。如果 bSendOnceToItself 或 bSendOnceToRemote 变量之一出现上升沿，便会生成新的 Fifo 条目并存储在“发送 Fifo”中。接收到的消息会从“接收 Fifo”中移除，并显示在消息框中。

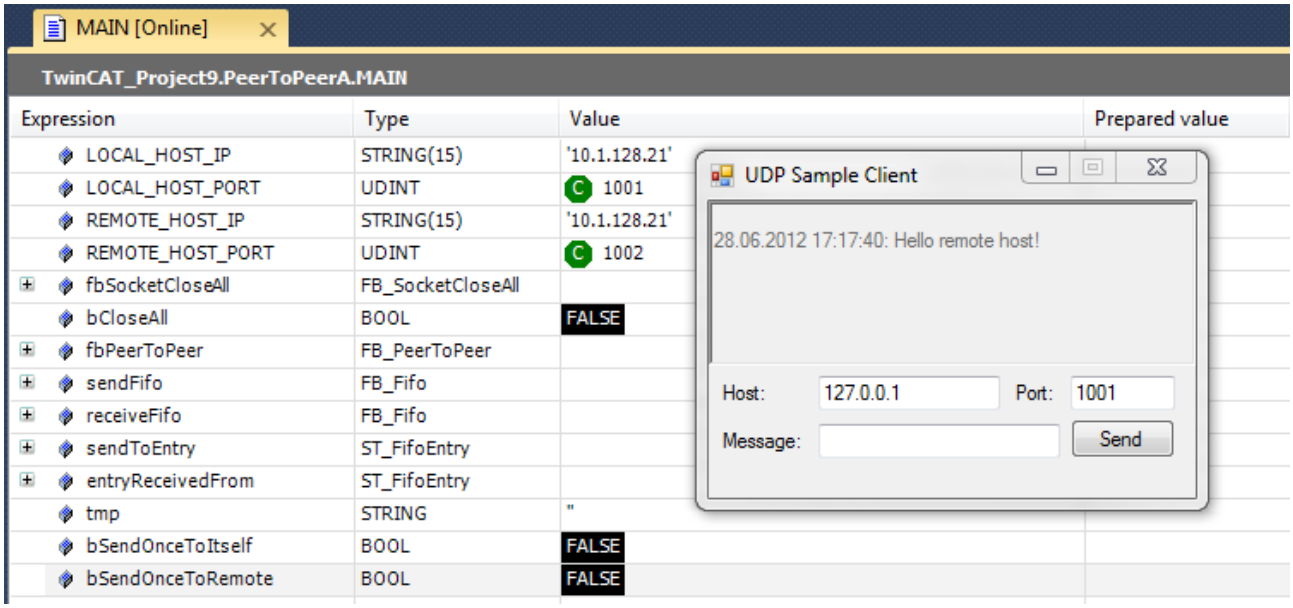
```
PROGRAM MAIN
VAR CONSTANT
    LOCAL_HOST_IP       : STRING(15)     := '';
    LOCAL_HOST_PORT     : UDINT          := 1001;
    REMOTE_HOST_IP      : STRING(15)     := '172.16.2.209';
    REMOTE_HOST_PORT    : UDINT          := 1001;
END_VAR
VAR
    fbSocketCloseAll    : FB_SocketCloseAll;
    bCloseAll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

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

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

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

IF NOT fbSocketCloseAll.bBusy AND NOT fbSocketCloseAll.bError THEN

    IF bSendOnceToRemote THEN
        bSendOnceToRemote          := FALSE;                  (* clear flag *)
     sendToEntry.nRemotePort          := REMOTE_HOST_PORT;          (* remote host port number*)
     sendToEntry.sRemoteHost          := REMOTE_HOST_IP;              (* remote host IP address *)
     sendToEntry.msg             := 'Hello remote host!';             (* message text*);
     sendFifo.AddTail( new := sendToEntry );                  (* add new entry to the send queue*)
```

```
    IF NOT sendFifo.bOk THEN                              (* check for fifo overflow*)
        LogError( 'Send fifo overflow!', PLCPRJ_ERROR_SENDFIFO_OVERFLOW );
      END_IF
  END_IF

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

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

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

END_IF
```

6.2.1.4 .NET 通信

该示例展示的是如何为 PLC 点对点设备 A 实现适配的 .NET 通信合作伙伴。本示例只能与 PLC 项目 PeerToPeerA 结合使用。



.NET 示例客户端可用于向 UPD 服务器（本例中为 PLC 项目 PeerToPeerA）发送单个 UPD 数据包。

下载

下载测试客户端。

解压 ZIP 文件；.exe 文件可在 Windows 系统上运行。

工作原理

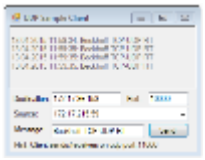
该示例使用 .Net 库 System.Net 和 System.Net.Sockets 来实现 UDP 客户端（类 UdpClient）。在后台线程中监听传入的 UDP 数据包时，可通过指定远程设备的 IP 地址和端口号并点击 Send（发送）按钮，将字符串发送到远程设备。

为了更好地理解本文，请设想以下设置：

- PLC 项目点对点设备 A 运行在 IP 地址为 10.1.128.21 的计算机上
- .NET 应用程序运行在 IP 地址为 10.1.128.30 的计算机上

描述

客户端本身使用端口 11000 进行发送。与此同时，它还会打开该端口，并在界面上部以日志形式显示接收到的消息：



结合 PLC / C++ 示例代码，最终得到一个回显示例：
从客户端 11000 端口向服务器 10000 端口发送 UDP 消息，而服务器 10000 端口会将相同的数据发送回发送方。

可通过界面配置客户端：

- 目的地：目的地 IP 地址
- 端口：目标中寻址的端口
- 来源：发送方网卡（IP 地址）。
  “基于 OS”的操作系统会处理选择适配网卡的问题。
- 消息

TF6311 “TCP/UDP Realtime”不允许执行本地通信。然而，出于测试目的，可以通过“Source（源）”选择不同的网络接口，这样 UDP 数据包便可通过其中一个网卡离开计算机，到达另一个网卡（"环形电缆"）。

6.2.2 Sample02：多播

本示例演示如何通过 UDP 发送和接收多播数据包。

客户端和服务器通过多播 IP 地址周期性地发送一个值给对方。

客户端和服务器由 2 个 PLC 应用程序实现，并集成在单一 TwinCAT 3 解决方案中。

系统要求

- TwinCAT 3 Build 3093 或更高版本
- TwinCAT 3 功能组件 TF6310 TCP/IP 1.0.64 或更高版本
- TwinCAT 3 Library Tc2_TcpIp 版本 3.2.64.0 或更高版本
- 如果使用 1 台计算机执行示例，例如，客户端和服务器分别运行在 2 个不同的 PLC Runtime 中，则这两个 PLC Runtime 都需要在不同的任务中运行。

项目下载

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

7 附录

7.1 OSI 模型

下文简要介绍 OSI 模型，并阐述该模型如何影响我们的日常网络通信。请注意，本文无意取代有关该主题的详尽文档，而只是提供概述。

OSI（Open Systems Interconnection，开放式系统互联）模型通过抽象分层模型，对通信系统的各项功能进行标准化定义。每一层都描述了网络中各设备之间通信的特定功能。每一层只与其正上层或正下层通信。

OSI model

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

示例：如果使用网络浏览器调用地址“http://www.beckhoff.com”，则从第七层开始使用以下协议：HTTP → TCP → IP → 以太网。但是，如果输入“https://www.beckhoff.com”，则会使用协议 HTTP → SSL → TCP → IP → 以太网。

TwinCAT 3 功能组件 TF6310 TCP/IP 可用于开发以 TCP 或 UDP 作为传输协议的网络功能 PLC 程序。这样，PLC 程序员就可以在应用层实施自己的网络协议，并定义自己的信息结构，以便与远程系统进行通信。

7.2 KeepAlive 配置

TCP KeepAlive 消息的报文可验证空闲 TCP 连接是否仍处于有效状态。自 TwinCAT TCP/IP Server (TF6310) 1.0.47 版起，将使用 Windows 操作系统的 KeepAlive 配置，可通过以下注册表键值进行配置：

以下文档摘自 Microsoft Technet。

KeepAliveTime

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

数据类型	范围	默认值
REG_DWORD	0x1-0xFFFFFFFF（毫秒）	0x6DDD00（7,200,000 毫秒 = 2 小时）

描述

设定 TCP 发送保活报文的频率。TCP 会发送保活报文，以验证空闲连接是否仍处于有效状态。该条目会在远程系统响应 TCP 时使用。否则，报文发送间隔由 KeepAliveInterval 项的值决定。默认情况下，不发送保活报文。TCP 保活功能必须由 Telnet 等程序或 Internet 浏览器（如 Internet Explorer）启用。

KeepAliveInterval

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

数据类型	范围	默认值
REG_DWORD	0x1-0xFFFFFFFF（毫秒）	0x3E8（1000 毫秒 = 1 秒）

描述

设置 TCP 在未收到响应时重复发送保活报文的频率。TCP 会发送保活报文，以验证空闲连接是否仍处于有效状态。从而防止 TCP 意外断开有效连接线路。

7.3 错误代码

7.3.1 错误代码的概述

代码（十六进制）	代码（十进制）	错误源	描述
0x00000000-0x00007800	0–30720	TwinCAT 系统错误代码 [▶ 92]	TwinCAT 系统错误（包括 ADS 错误代码）
0x00008000-0x000080FF	32768–33023	TwinCAT TCP/IP Connection Server 内部错误代码 [▶ 90]	TwinCAT TCP/IP Connection Server 内部错误
0x80070000-0x8007FFFF	2147942400–2148007935	错误源 = 代码 - 0x80070000 = Win32 系统错误代码	Win32 系统错误（包括 Windows 套接字错误代码）

要求

开发环境	目标系统类型	待集成的 PLC 库
TwinCAT v3.1	PC、CX (x86) 或 CX (Arm®)	Tc2_TcpIp

7.3.2 TwinCAT TCP/IP Connection Server 的内部错误代码

代码（十六进制）	代码（十进制）	符号常量	描述
0x00008001	32769	TCPADSERROR_NOMOREENTRIES	无法创建新套接字（FB_SocketListen 和 FB_SocketConnect）。
0x00008002	32770	TCPADSERROR_NOTFOUND	套接字句柄无效（适用于 FB_SocketReceive、FB_SocketAccept、FB_SocketSend 等）。
0x00008003	32771	TCPADSERROR_ALREADYEXISTS	如果 TcpIp 端口监听器已经存在，则会在调用 FB_SocketListen 时返回该值。
0x00008004	32772	TCPADSERROR_NOTCONNECTED	在调用 FB_SocketReceive 时，如果客户端套接字不再与服务器连接，则返回该值。
0x00008005	32773	TCPADSERROR_NOTLISTENING	当调用 FB_SocketAccept 时，如果监听套接字发生错误，则返回该值。
0x00008006	32774	TCPADSERROR_HOST_NOT_FOUND	目标系统不可访问时返回该值。
0x00008080	32896	TCPADSERROR_TLS_INVALID_STATE	如果调用了 FB_TlsSocketAddCa、FB_TlsSocketAddCrl、FB_TlsSocketSetCert 或 FB_TlsSocketSetPsk，且已调用 Connect（连接），则返回该值。
0x00008081	32897	TCPADSERROR_TLS_CA_NOTFOUND	如果未找到指定的 CA 证书，则返回该值。
0x00008082	32898	TCPADSERROR_TLS_CERT_NOTFOUND	如果未找到指定的证书文件，则返回该值。

代码（十六进制）	代码（十进制）	符号常量	描述
0x00008083	32899	TCPADSERROR_TLS_KEY_NOTFOUND	如果未找到包含私钥的指定文件，则返回该值。
0x00008084	32900	TCPADSERROR_TLS_CA_INVALID	如果指定的 CA 证书无法读取或无效，则返回该值。
0x00008085	32901	TCPADSERROR_TLS_CERT_INVALID	如果指定的证书文件无法读取或无效，则返回该值。
0x00008086	32902	TCPADSERROR_TLS_KEY_INVALID	如果指定的私钥无法读取或无效，则返回该值。
0x00008087	32903	TCPADSERROR_TLS_VERIFY_FAIL	如果在 TLS 握手过程中无法验证远程终端，则返回该值。
0x00008088	32904	TCPADSERROR_TLS_SETUP	如果在设置 TLS 连接时发生一般性错误，则返回该值。
0x00008089	32905	TCPADSERROR_TLS_HANDSHAKE_FAIL	如果在 TLS 握手过程中发生错误，则返回该值。通常情况下，握手始终有效。但是，如果在握手过程中出现连接问题，则可能会导致失败。
0x0000808A	32906	TCPADSERROR_TLS_CIPHER_INVALID	如果指定的密码套件无效，则返回该值。
0x0000808B	32907	TCPADSERROR_TLS_VERSION_INVALID	如果指定的 TLS 版本无效，则返回该值。
0x0000808C	32908	TCPADSERROR_TLS_CRL_INVALID	如果指定的证书撤销列表 (CRL) 无效，则返回该值。
0x0000808D	32909	TCPADSERROR_TLS_INTERNAL_ERROR	如果在设置 TLS 连接时发生内部错误，则返回该值。
0x0000808E	32910	TCPADSERROR_TLS_PSK_SETUP_ERROR	如果在为 TLS 使用预共享密钥 (PSK) 时发生错误，则返回该值。
0x0000808F	32911	TCPADSERROR_TLS_CN_MISMATCH	如果远程终端证书中的 CommonName（通用名称）与所用主机名或 IP 地址不匹配，则返回该值。
0x00008090	32912	TCPADSERROR_TLS_CERT_EXPIRED	当远程终端证书过期时返回该值。
0x00008091	32913	TCPADSERROR_TLS_CERT_REVOKED	当远程终端证书撤销时返回该值。
0x00008092	32914	TCPADSERROR_TLS_CERT_MISSING	当远程终端未提交证书时返回该值。

要求

开发环境	目标系统类型	待集成的 PLC 库
TwinCAT v3.1	PC、CX (x86) 或 CX (Arm®)	Tc2_TcpIp

7.3.3 故障排除/诊断

- 如果出现连接问题，可以使用 PING 命令来确定是否可以通过网络连接与外部通信伙伴取得联系。若非如此，请检查网络配置和防火墙设置。
- Wireshark® 等抓包工具可以实现整个网络通信的日志记录。然后，倍福支持人员可对日志进行分析。
- 检查本文档所述硬件和软件要求（TwinCAT 版本、CE 图像版本等）。
- 检查本文档所述软件安装提示（例如，在 CE 平台上安装 CAB 文件）。
- 检查传输到功能块的输入参数（网络地址、端口号、数据等、连接句柄）是否校正。检查功能块是否发出错误代码。可在此处查阅错误代码相关文档：错误代码概览 [▶ 90]。
- 检查其他通信伙伴/软件/设备是否发出错误代码。

- 在连接建立/断开过程中激活 TcSocketHelper.Lib 中集成的调试输出（关键词：CONNECT_MODE_ENABLEDBG）。打开 TwinCAT System Manager 并激活 LogView 窗口。分析/检查调试输出字符串。

要求

开发环境	目标系统类型	待集成的 PLC 库
TwinCAT v3.1	PC、CX (x86) 或 CX (Arm®)	Tc2_TcpIp

7.3.4 ADS 返回代码

错误代码分组：
全局错误代码：0x0000 [▶ 92]... （0x9811_0000 ...）
路由器错误代码：0x0500 [▶ 92]... （0x9811_0500 ...）
一般 ADS 错误：0x0700 [▶ 93]... （0x9811_0700 ...）
RTime 错误代码：0x1000 [▶ 94]... （0x9811_1000 ...）

全局错误代码

Hex	Dec	HRESULT	名称	描述
0x0	0	0x98110000	ERR_NOERROR	无错误。
0x1	1	0x98110001	ERR_INTERNAL	内部错误。
0x2	2	0x98110002	ERR_NORTIME	不具有实时性。
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	分配已锁定 – 内存错误。
0x4	4	0x98110004	ERR_INSERTMAILBOX	邮箱已满 – 无法发送 ADS 消息。减少每个周期的 ADS 消息数量将有所帮助。
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	HMSG 错误。
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	未找到目标端口 – ADS 服务器未启动或无法访问。
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	未找到目标计算机 – 未找到 AMS 路由。
0x8	8	0x98110008	ERR_UNKNOWNCMDID	未知命令 ID。
0x9	9	0x98110009	ERR_BADTASKID	任务 ID 无效。
0xA	10	0x9811000A	ERR_NOIO	No IO。
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	未知 AMS 命令。
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 错误。
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	端口未连接。
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	AMS 长度无效。
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	AMS Net ID 无效。
0x10	16	0x98110010	ERR_LOWINSTLEVEL	安装级别 – TwinCAT 2 授权错误。
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	无调试可用。
0x12	18	0x98110012	ERR_PORTDISABLED	端口已禁用 – TwinCAT 系统服务未启动。
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	端口已连接。
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 错误。
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync 超时。
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync 错误。
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	不存在适用于 AMS Sync 的索引映射。
0x18	24	0x98110018	ERR_INVALIDAMSPORT	AMS 端口无效。
0x19	25	0x98110019	ERR_NOMEMORY	无内存。
0x1A	26	0x9811001A	ERR_TCPSEND	TCP 发送错误。
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	主机无法访问。
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	AMS 片段无效。
0x1D	29	0x9811001D	ERR_TLSSEND	TLS 发送错误 – ADS 安全连接失败。
0x1E	30	0x9811001E	ERR_ACCESSDENIED	拒绝访问 – 拒绝 ADS 安全访问。

路由器错误代码

Hex	Dec	HRESULT	名称	描述
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	无法分配锁定的内存。

Hex	Dec	HRESULT	名称	描述
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	路由器内存大小无法更改。
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	邮箱已达到最大消息数。
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	调试邮箱已达到最大消息数。
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	端口类型未知。
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	路由器未初始化。
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	端口号已分配。
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	端口未注册。
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	已达到最大端口数。
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	端口无效。
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	路由未激活。
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	邮箱已达到最大片段消息数。
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	发生片段超时。
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	端口已移除。

一般性 ADS 错误代码

Hex	Dec	HRESULT	名称	描述
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	一般性设备错误。
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	服务器不支持该服务。
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	索引组无效。
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	索引偏移量无效。
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	不允许读取或写入。 可能有几种原因。例如，创建路由时输入了错误的密码。
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	参数大小不正确。
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	无效数据值。
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	设备尚未准备好运行。
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	设备正忙。
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	操作系统上下文无效。这可能是由于在不同的任务中使用 ADS 函数块造成的。可以通过在 PLC 中进行多任务同步解决这个问题。
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	内存不足。
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	参数值无效。
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	未找到（文件…）。
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	文件或命令中存在语法错误。
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	对象不匹配。
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	对象已存在。
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	符号未找到。
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	符号版本无效。这可能是由于联机更改造成的。创建新句柄。
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	设备（服务器）处于无效状态。
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	不支持 AdsTransMode。
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	通知句柄无效。
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	通知客户端未注册。
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	没有更多的句柄可用。
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	通知大小过大。
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	设备未初始化。
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	设备超时。
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	接口查询失败。
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	请求的接口错误。
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID 无效。
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID 无效。

Hex	Dec	HRESULT	名称	描述
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	请求待定。
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	请求已中止。
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	警告信号。
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	数组索引无效。
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	符号未激活。
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	拒绝访问。 有几种可能的原因。例如，单向 ADS 路由用于相反方向。
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	缺少授权。
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	授权已过期。
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	超出授权。
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	授权无效。
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	授权问题：系统 ID 无效。
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	授权不受时间限制。
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	授权问题：未来的时间。
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	授权期限太长。
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	系统启动异常。
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	授权文件读取了两次。
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	签名无效。
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	证书无效。
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	OEM未知公钥。
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	此系统 ID 授权无效。
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	演示授权已禁止。
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	无效函数 ID。
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	超出有效范围。
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	无效对齐。
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	无效平台级别。
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	上下文 – 转到被动级别。
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	上下文 – 转到调度级别。
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	上下文 – 转到实时。
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	客户端错误。
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	服务包含无效参数。
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	轮询列表为空。
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var 连接已在使用中。
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	调用的 ID 已在使用中。
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	已发生超时 – 远程终端在指定的 ADS 超时中未响应。远程终端的路由设置可能配置不正确。
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Win32 子系统中发生错误。
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	客户端超时值无效。
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	端口未打开。
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	无 AMS 地址。
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Ads sync 中发生内部错误。
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	哈希表溢出。
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	未在表格中找到密钥。
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	缓存中没有符号。
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	收到无效响应。
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	同步端口已锁定。
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	请求被取消。

RTime 错误代码

Hex	Dec	HRESULT	名称	描述
0x1000	4096	0x98111000	RTERR_INTERNAL	实时系统中发生内部错误。
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	计时器值无效。
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	任务指针提供了无效值 0（零）。
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	堆栈指针提供了无效值 0（零）。

Hex	Dec	HRESULT	名称	描述
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	请求任务优先级已分配。
0x1005	4101	0x98111005	RTERR_NOMORETCB	没有可用的空闲 TCB（任务控制块）。TCB 的最大数量为 64。
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	没有可用的空闲信号。信号的最大数量为 64。
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	队列中没有可用的空闲空间。队列中的最大位置数为 64。
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	已应用外部同步中断。
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	未应用外部同步中断。
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	外部同步中断应用失败。
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	在错误的上下文中调用服务函数
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	不支持 Intel® VT-x 扩展。
0x1018	4120	0x98111018	RTERR_VMXDISABLED	BIOS 中未启用 Intel® VT-x 扩展。
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Intel® VT-x 扩展中缺少函数。
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Intel® VT-x 激活失败。

具体的正向 HRESULT 返回代码：

HRESULT	名称	描述
0x0000_0000	S_OK	无错误。
0x0000_0001	S_FALSE	无错误。 示例：处理成功，但有一个负面或不完整的结果。
0x0000_0203	S_PENDING	无错误。 示例：处理成功，但还没有结果。
0x0000_0256	S_WATCHDOG_TIMEOUT	无错误。 示例：处理成功，但发生了超时。

TCP Winsock 错误代码

Hex	Dec	名称	描述
0x274C	10060	WSAETIMEDOUT	发生连接超时 - 在创建连接时发生错误，因为远程终端在特定时间后未正确响应，或者所建立连接因所连接主机未响应而无法维持。
0x274D	10061	WSAECONNREFUSED	连接遭到拒绝 - 无法建立连接，因为目标计算机明确拒绝了该连接。此错误通常由尝试连接到外部主机上处于非活动状态的服务（即没有运行服务器应用程序的服务）导致。
0x2751	10065	WSAEHOSTUNREACH	不存在至主机的路由 - 套接字操作引用了不可用的主机。
更多 Winsock 错误代码：Win32 错误代码			

7.4 技术支持和服务

倍福公司及其合作伙伴在世界各地提供全面的技术支持和服务，对与倍福产品和系统解决方案相关的所有问题提供快速有效的帮助。

下载搜索器

我们的下载搜索器包含我们供您下载的所有文件。您可以通过它搜索我们的应用案例、技术文档、技术图纸、配置文件等等。

可供下载的文件格式多种多样。

倍福分公司和代表处

若需要倍福产品的本地支持和服务，请联系倍福分公司或代表处！

倍福遍布世界各地的分公司和代表处地址可在倍福官网上找到：http://www.beckhoff.com.cn

该网页还提供更多倍福产品组件的文档。

倍福技术支持

技术支持部门为您提供全面的技术援助，不仅帮助您应用各种倍福产品，还提供其他广泛的服务：

- 技术支持
- 复杂自动化系统的设计、编程和调试
- 以及倍福系统组件的各种培训课程

热线电话：	+49 5246 963-157
电子邮箱：	support@beckhoff.com

倍福售后服务

倍福服务中心提供所有售后服务：

- 现场服务
- 维修服务
- 备件服务
- 热线服务

热线电话：	+49 5246 963-460
电子邮箱：	service@beckhoff.com

倍福公司总部

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

电话：	+49 5246 963-0
电子邮箱：	info@beckhoff.com
网址：	www.beckhoff.com

Trademark statements

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

Third-party trademark statements

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

DALI, DALI-2, D4i, DALI+, and DiiA are trademarks in various countries in the exclusive use of the Digital Illumination Interface Alliance.

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

MAX®, Stratix®, Cyclone®, Altera®, Agilex™, Arria®, Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

Wireshark is a registered trademark of Sysdig, Inc.

更多信息:
www.beckhoff.com/tf6310

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