**TF6311 TCP/UDP Realtime基本使用**

|  |  |
| --- | --- |
|  | 作者：云龙  职务：华东区 技术支持工程师  邮箱：long.yun@beckhoff.com.cn  日期：2022-12-19 |
| **摘 要：**  TF6311 TCP/UDP RealTime功能提供从实时环境直接访问网卡的功能，可以使用基于IEC61131-3的PLC程序或TwinCAT C++程序进行访问，后文将详细描述。 | |
| **附 件：**   |  |  |  | | --- | --- | --- | | 序 号 | 文件名 | 备注 | | 1 | TF6311 FunctionPack | 例程 | |  |  |  | |  |  |  | |  |  |  | | |
| **历史版本：**   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | | |
| **免责声明：**  我们已对本文档描述的内容做测试。但是差错在所难免，无法保证绝对正确并完全满足您的使用需求。本文档的内容可能随时更新，如有改动，恕不事先通知，也欢迎您提出改进建议。 | |
| **参考信息：** | |

目 录

[1. 概述 3](#_Toc122279539)

[1.1. 对比TF6310与TF6311 3](#_Toc122279540)

[1.2. 使用限制 4](#_Toc122279541)

[2. 准备工作 4](#_Toc122279542)

[2.1. 功能安装与授权 4](#_Toc122279543)

[2.2. 技术说明 4](#_Toc122279544)

[2.2.1. 通讯组件 4](#_Toc122279545)

[2.2.2. 编程接口 4](#_Toc122279546)

[2.2.3. 通讯性能 6](#_Toc122279547)

[2.2.4. 网络接口 7](#_Toc122279548)

[3. 测试环境 8](#_Toc122279549)

[3.1. Beckhoff 8](#_Toc122279550)

[3.1.1. 控制器硬件 8](#_Toc122279551)

[3.1.2. 控制软件 8](#_Toc122279552)

[3.2. 测试软件 8](#_Toc122279553)

[3.2.1. 通讯调试助手 8](#_Toc122279554)

[3.2.2. 通讯报文抓包软件 9](#_Toc122279555)

[3.3. 网络接线及IP配置 9](#_Toc122279556)

[4. 测试范例 10](#_Toc122279557)

[4.1. UDP功能测试 10](#_Toc122279558)

[4.1.1. PLC程序实现 10](#_Toc122279559)

[4.1.2. TcC++程序实现 19](#_Toc122279560)

[4.2. TCP客户端功能测试 23](#_Toc122279561)

[4.2.1. PLC程序实现 23](#_Toc122279562)

[4.2.2. TcC++程序实现 31](#_Toc122279563)

[4.3. TCP服务器功能测试 31](#_Toc122279564)

[4.3.1. PLC程序实现 31](#_Toc122279565)

[4.3.2. TcC++程序实现 39](#_Toc122279566)

# 概述

TF6311 TCP/UDP RealTime功能提供从实时环境直接访问网卡的功能，使用PLC程序或TwinCAT C++程序进行访问，后文将详细描述。而其他的通讯方式例如TCPIP、OPC UA等基于非实时以太网的通讯则是通过访问BECKHOFF提供的Windows服务间接访问网卡的数据，Windows服务运行在Windows非实时环境中，如此通讯也就不再具备实时性。

TF6311 TCP/UDP RealTime支持以下协议：

• TCP/IP

• UDP/IP

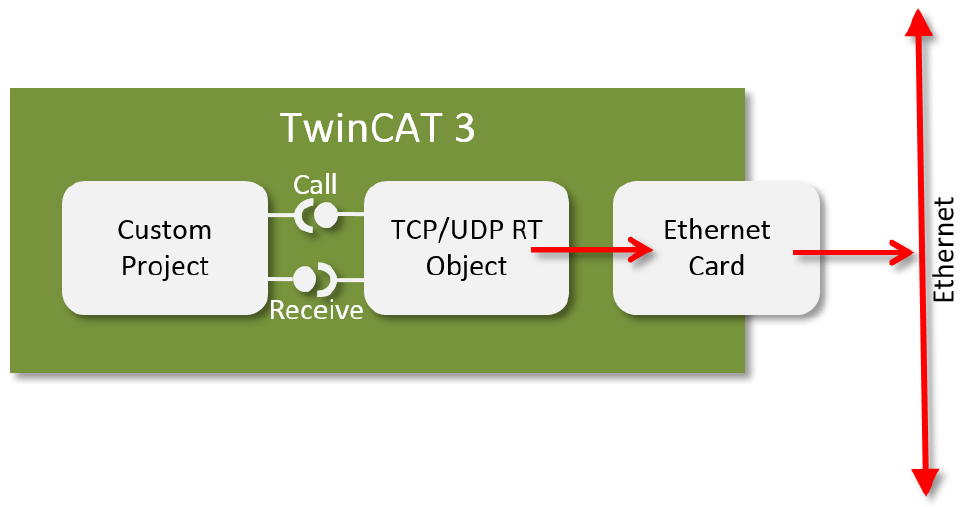
• ARP / Ping

无论以上哪种协议，使用该协议的项目与TCP/UDP RT组件之间的通信都是通过一对接口实现的，以TCP/IP通讯举例：

• ITcIoTcpProtocol接口，用作与TCP/UDP RT组件建立连接和发送数据等。

• ITcIoTcpProtocolRecv接口基于回调函数实现，用作以事件或数据的形式为项目提供反馈。

这些接口对的通信对象是一个TCP/UDP RT组件，它在添加时实例化并与网卡绑定。



## 对比TF6310与TF6311

产品TF6310 "TCP/IP "和TF6311"TCP/UDP RealTime "提供类似的功能。此处进行两种产品的对比：

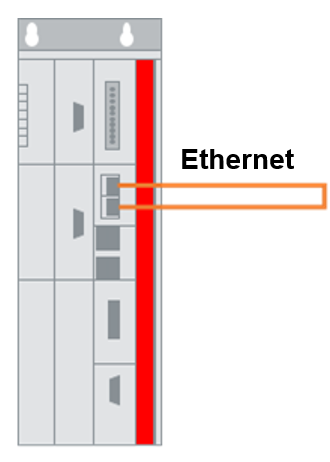
|  |  |  |
| --- | --- | --- |
|  | **TF 6310** | **TF 6311** |
| TwinCAT支持 | TwinCAT 2 / 3 | TwinCAT 3 |
| Client/Server | 两者都支持 | 两者都支持 |
| 大型/未知网络 | ++ | +[[1]](#footnote-1) |
| 确定性[[2]](#footnote-2) | + | ++ |
| 大容量的数据传输 | ++ | + |
| 编程语言 | PLC | PLC和C++ |
| 操作系统 | Win32/64, CE5/6/7 | Win32/64, CE7 |
| UDP多播 | 支持 | 不支持 |
| 试用授权 | 是 | 是 |
| 支持的协议 | TCP, UDP | TCP, UDP, ARP/Ping |
| 硬件要求 | 不限制 | 与TwinCAT兼容的网卡[[3]](#footnote-3) |

## 使用限制

该产品存在以下使用限制：

• 不能用作实现TwinCAT实时系统与Windows系统之间的本地环网(即127.0.0.1)通信。

(替代方案：通过设备的第二个网络接口组成环形网络进行通信，如下图所示)



• 不支持组播。

• EL6601和EL6614不能用于TF6311 TCP/UDP RealTime。

• 如需断点调试，建议调试使用不同的网络接口，因为断点会停止TwinCAT系统的某些部分，而这些部分可能与项目调试部分的通信有关。

# 准备工作

## 功能安装与授权

TF6311功能不需要单独安装，一旦安装了TwinCAT 3，TF6311所有功能都可以使用。

• 需要有"TC3 TCP UDP RT "授权。

• 可以使用7天试用授权。

## 技术说明

### 通讯组件

TCP/UDP RT组件是TF6311功能的主要组成部分，组件的实例化通常在某个具体的Device下进行，如下图在网络接口Device1设备下添加一个TCP/UDP RT组件实例。

图形用户界面, 文本

描述已自动生成

### 编程接口

TCP/UDP RT组件可以被不同的协议使用，程序端根据功能需要选择对应的组件接口实现：

* **实现TCP/IP实时通信：**ITcIoTcpProtocol和ITcIoTcpProtocolRecv接口

ITcIoTcpProtocol接口包含一个指向ITcIoTcpProtocol object的指针（该对象与TCP/UDP RT组件链接），并实现ITcIoTcpProtocolRecv本身。

ITcIoTcpProtocolRecv作为一个回调接口，用于程序端接收TCP/UDP RT组件的数据和事件。这些接口是基于一个套接字API，在使用一个套接字之前，必须用AllocSocket()分配它。

* **实现UDP/IP实时通信：**ITcIoUdpProtocol和ITcIoUdpProtocolRecv接口

ITcIoUdpProtocol接口包含一个指向ITcIoUdpProtocol object的指针（该对象与TCP/UDP RT组件链接）并实现ITcIoUdpProtocolRecv本身。

ITcIoUdpProtocolRecv作为回调接口，用于应用程序接收TCP/UDP RT模块的数据和事件。

* **实现ARP/Ping消息实时发送：**ITcIoArpPingProtocol和ITcIoArpPingProtocolRecv接口：

ITcIoArpPingProtocol接口包含一个指向ITcIoArpPingProtocol object的指针（该对象与TCP/UDP RT组件链接），并实现ITcIoArpPingProtocolRecv本身。

ITcIoArpPingProtocolRecv作为回调接口，用于应用程序接收TCP/UDP RT模块数据。

****编程接口实现范例：****

以实现TCP/IP实时通信为例：

1.添加TCP/UDP RT通讯组件

图形用户界面, 文本

描述已自动生成

2.添加功能块实现ITcIoTcpProtocolRecv接口

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

3.声明ITcIoTcpProtocol接口指针ipTcp与ObjectID变量oid

表格

中度可信度描述已自动生成

4.编译完成后将PLC程序中的oid与TCP/UDP RT通讯组件实例链接

图片包含 应用程序

描述已自动生成

5.在FB的Init方法中使用TC3\_Module函数库获取TCP/UDP RT组件的实例，并完成与PLC程序中声明的接口指针ipTcp的绑定

图形用户界面, 文本, 应用程序

描述已自动生成

6.以上实现了TCP/IP实时通讯的硬件接口和PLC程序编程接口的绑定，提供简略步骤仅供理解通讯实现原理

### 通讯性能

TCP/UDP RT组件以实时方式运行，该组件直接依赖于实时核的循环。因此，数据通信与组件的调用任务的循环周期有关系。

通过网络接口的数据通信取决于对CheckReceived()方法的周期性调用。

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

### 网络接口

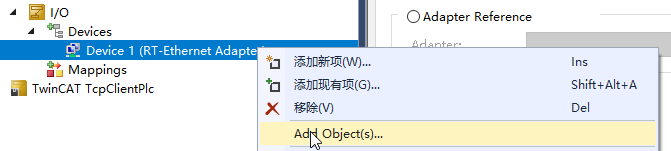
"TCP/UDP RT "组件被实例化和配置，配置则是为组件分配要使用的实时以太网卡。

1. 为项目添加实时以太网适配器

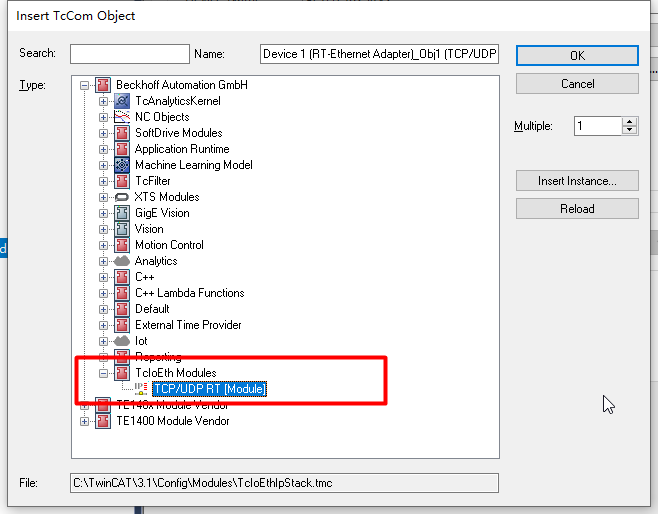
图形用户界面, 文本, 应用程序

描述已自动生成

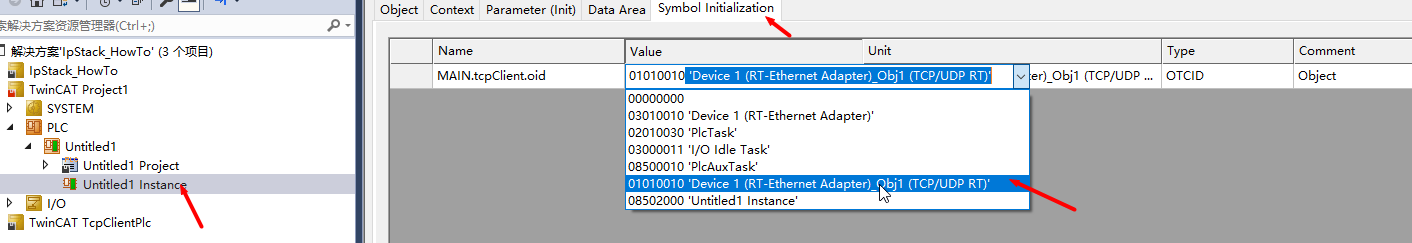
1. 在实时以太网适配器下创建 "TCP/UDP RT "组件，在右键中选择 "Add Object(s)..."



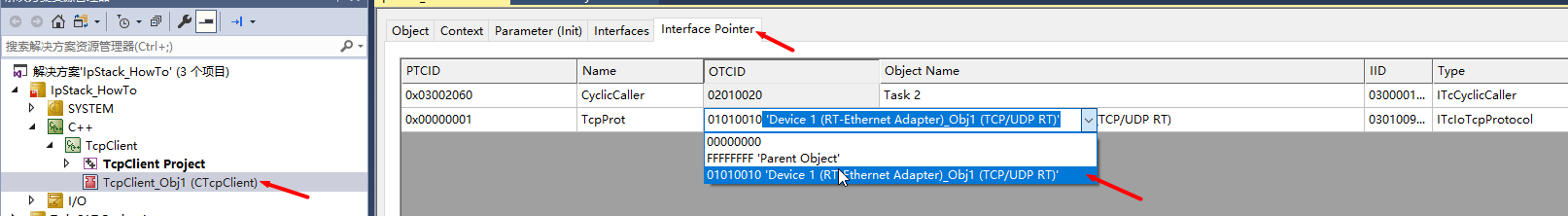
1. 在对话框中选择"TcIoEth Module"—>"TCP/UDP RT"组件添加



1. "TCP/UDP RT"组件与编程接口的绑定链接
   1. PLC环境中：



* 1. Tc C++环境中



* 这样，配置就完成了，需要注意因为TF6311直接集成在TwinCAT系统中，所以不能启用Windows防火墙否则会导致通讯无法实现。

# 测试环境

## Beckhoff

### 控制器硬件

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 型号 | 操作系统 | 适配器型号 | 适配器数量 |
| 嵌入式控制器 | CX5020 | WinCE 6.0 | Intel 82574L | 2 |
| 笔记本电脑 | Dell Latitude 5491 | Win10 企业版 | Intel i219-LM | 1 |
| 虚拟机 | 笔记本中VM实现 | Win10 WorkStation | E1000 | 1 |

倍福控制器、PC，包括：

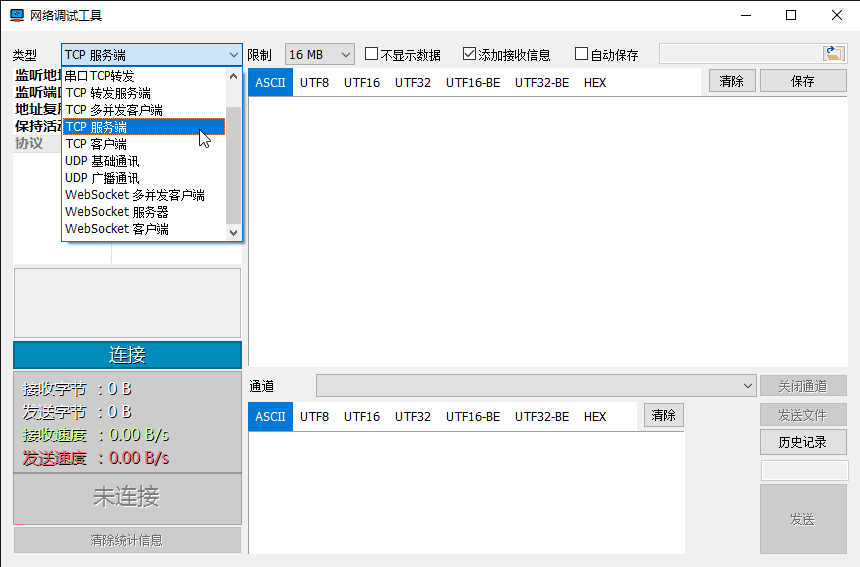
### 控制软件

控制器、笔记本和虚拟机都是基于TwinCAT 3.1 Build 4024.22版本

## 测试软件

### 通讯调试助手

使用开源网络调试工具进行网络调试中通讯服务建立与数据监控



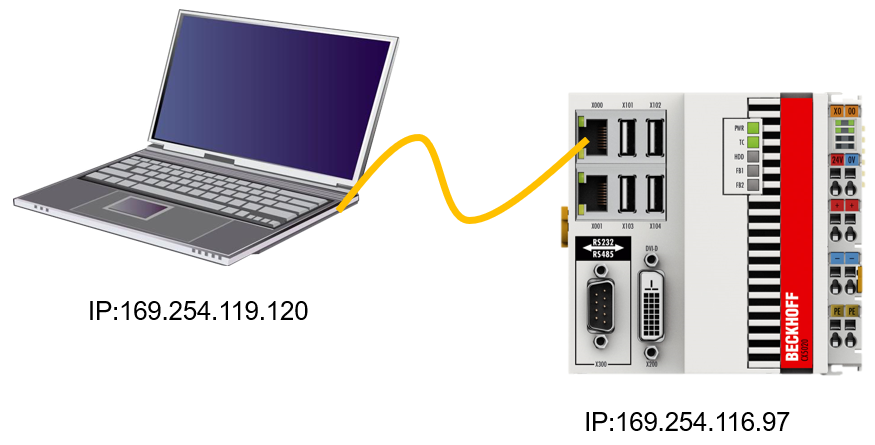
### 通讯报文抓包软件

Wireshark软件 v3.4.16.0，用作抓取数据通讯报文获取更详细信息

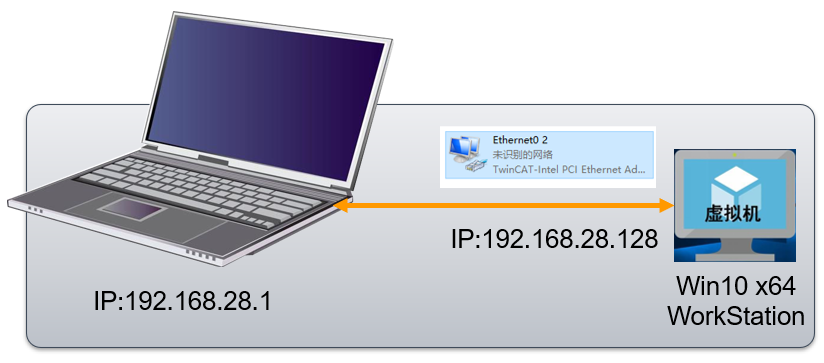


## 网络接线及IP配置

测试环境一：使用倍福控制器CX5020网络接口X000与PC网线连接



测试环境二：虚拟机PC与PC通过虚拟网络适配器内部连接



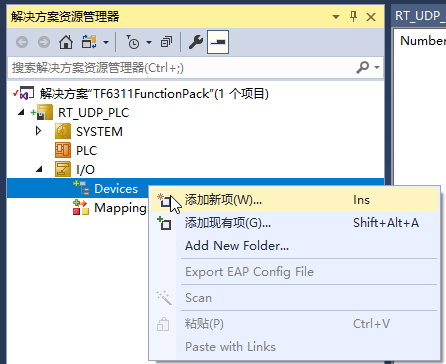
# 测试范例

## UDP功能测试

实现基于UDP/IP协议的通讯连接建立与数据接收发送的功能，需要使用ITcIoUdpProtocol和ITcIoUdpProtocolRecv接口。

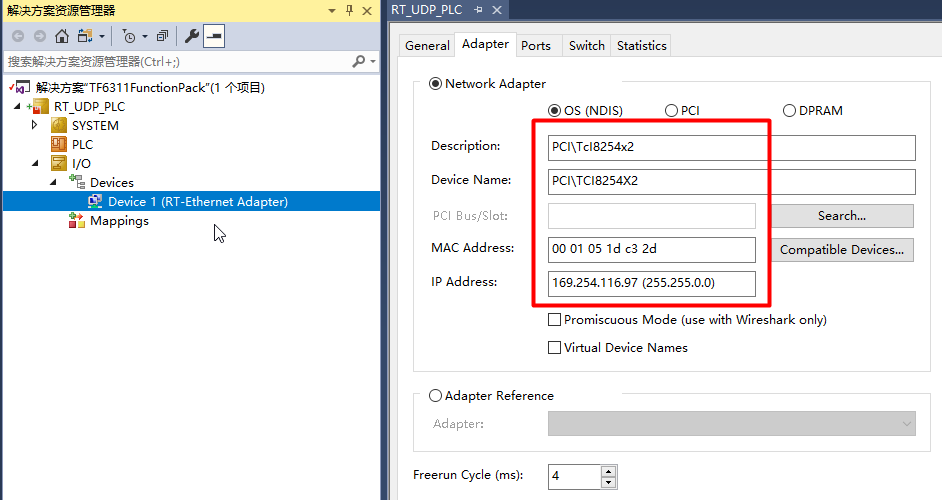
### PLC程序实现

1. 在现有的TwinCAT工程设备树中选择IO—>Devices，选中Devices右键添加新项—>选择Ethernet—>Real-Time Ethernet Adapter

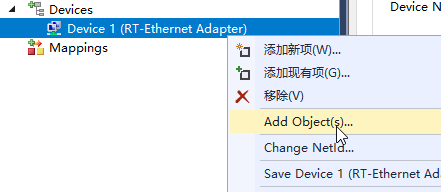
 图形用户界面, 文本, 应用程序

描述已自动生成

1. 确认设备已添加—>确认Adapter选项中Network Adapter内容非空



1. 选择Device1(Real-Time Ethernet Adapter)—>点击右键—>选择 "Add Object(s)..."，并完成"TCP/UDP RT "组件的添加

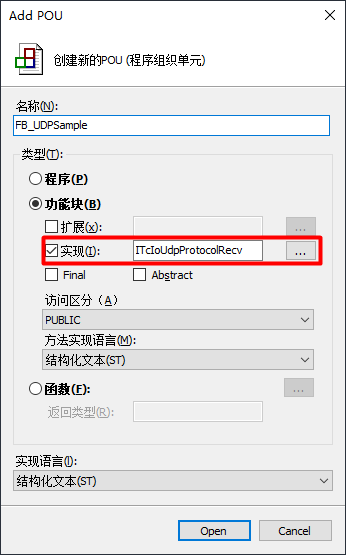
 图形用户界面, 应用程序

描述已自动生成

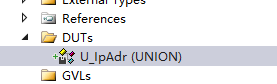
1. 添加完成后的设备树



1. 添加标准PLC工程—>添加功能块并实现ITcIoUdpProtocolRecv接口



1. 添加联合体U\_IpAdr用作IP地址转换



TYPE U\_IpAdr :

UNION

    ipadrInternal   : UDINT;

    ipadr           : ARRAY[0..3] OF **BYTE**;

//ex ip = 192.168.1.123—>

//ipadr[3] = 192，

//ipadr[2] = 168，

//ipadr[1] = 1，

//ipadr[0] = 123

END\_UNION

END\_TYPE

1. 功能块中变量声明及说明

{attribute 'c++\_compatible'}

**FUNCTION\_BLOCK PUBLIC** FB\_UDPSample **IMPLEMENTS** ITcIoUdpProtocolRecv

**VAR**

    //Object

    {attribute 'TcInitSymbol'}

    oid                 :OTCID;

    ipUdp               :ITcIoUdpProtocol;

    //Init

    hrInit              :**HRESULT**;

    uiLocalPort         :**UINT** := 6000;

    UdpTargetAdr        :U\_IpAdr;

    uiTargetPort        :**UINT** := 10000;

    //Recv

    hrRecv                          :**HRESULT**;

    UdpRecvAdr                      :U\_IpAdr;

    uiRecvPort                      :**UINT**;

    uiReceivedPakets                :**UINT**;

    udiLastReturnedMessageLength    :UDINT;

    sLastReturnedMessage            :STRING;

    //Send

    hrSend              :**HRESULT**;

    bSend               :**BOOL** := FALSE;

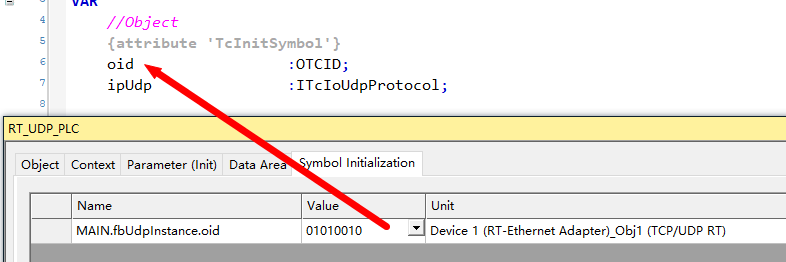
    sMessage            :STRING := '0123456789';

    udiSendCount        :UDINT;

**END\_VAR**

attribute ' c++\_compatible '：使PLC编译器生成的VTable与C++编译器的VTable二进制兼容，使得可以通过PLC中的接口方法，访问C++实现的TcCom组件，在本例中即是通过PLC中的接口方法访问"TCP/UDP RT "组件。

attribute 'TcInitSymbol'：使得定义的变量可以作为初始化符号使用，编译完成后包含该属性的变量在PLC实例的 " Symbol Initialization"标签中出现。这些变量的值，在代码执行开始前被复制到变量值中，并覆盖在变量声明时指定的初始值，在本例中用作传递"TCP/UDP RT "组件的ID信息，如下图所示。



1. 为功能块添加FB\_init方法，进行通讯组件的参数信息检查等

图形用户界面, 应用程序, 电子邮件

描述已自动生成

1. 添加代码实现

IF NOT bInCopyCode THEN // no online change

    IF ipUdp = 0 AND oid <> 0 THEN

        hrInit := FW\_ObjMgr\_GetObjectInstance(  oid:=oid,

                                                iid:=TC\_GLOBAL\_IID\_LIST.IID\_ITcIoUdpProtocol,

                                                pipUnk:=ADR(ipUdp) );

        IF SUCCEEDED(hrInit) THEN

            IF SUCCEEDED(ipudp.RegisterReceiver(uiLocalPort, THIS^)) THEN //Open Port

                FB\_init := TRUE;

            ELSE

                FB\_init := FALSE;

                FW\_SafeRelease(ADR(ipUdp));

            END\_IF

        ELSE

            FW\_SafeRelease(ADR(ipUdp));

        END\_IF

    ELSIF oid = 0 THEN

        FB\_init := FALSE;

        hrInit := ERR\_INVALID\_PARAM;

    END\_IF

END\_IF

//generate adress

//Remote IP 169.254.119.120

UdpTargetAdr.ipadr[3] := 169;

UdpTargetAdr.ipadr[2] := 254;

UdpTargetAdr.ipadr[1] := 119;

UdpTargetAdr.ipadr[0] := 120;

FW\_ObjMgr\_GetObjectInstance()：来自TC3\_Module库的函数，该函数通过输入Object ID，返回该Object ID指向的对象实例的接口指针，本例中返回"TCP/UDP RT "组件实例的接口指针。

RegisterReceiver()：来自ITcIoUdpProtocolRecv接口，用作在"TCP/UDP RT "组件注册功能，以接收通讯数据。

FW\_SafeRelease()：来自TC3\_Module库的函数，该函数减少该接口指针的引用计数，并同时将涉及的接口指针设置为零，如果没有有效管理可能导致内存崩溃或内存泄漏问题。

1. 为功能块添加FB\_exit方法，取消已注册的组件功能并安全释放接口指针



1. 添加代码实现

IF NOT bInCopyCode AND ipUdp <> 0 THEN //Shutdown

    ipUdp.UnregisterReceiver(uiLocalPort);

    FW\_SafeRelease(ADR(ipUdp));

    FB\_exit := TRUE;

ELSE

    FB\_exit := FALSE;

END\_IF

1. 为功能块添加FB\_reinit方法，在执行“在线更改”时为"TCP/UDP RT "组件提供新的回调地址等

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. 添加代码实现

//reregister THIS^ for online changes

IF (ipUdp <> 0) THEN

    ipUdp.RegisterReceiver(uiLocalPort, THIS^);

    FB\_reinit := TRUE;

END\_IF

1. 当接收到通讯数据时，ITcIoUdpProtocol.CheckReceived()开始处理，自动调用TcQueryInterface方法查询/验证当前接收到的是否是某个特定的接口，如果是QueryInterFace将返回一个指向该接口的指针。在本例中调用TcQueryInterface方法查询/验证"TCP/UDP RT "组件反馈的是否是ITcIoUdpProtocolRecv接口，如果是则返回指向该接口的指针，而在本例中该接口指针也就是“THIS^”，以下为代码实现：

方法中两个中间变量，用作接口指针的传递

METHOD TcQueryInterface : **HRESULT**

VAR\_INPUT

    iid : REFERENCE TO IID;

    pipItf  : POINTER TO **PVOID**;

END\_VAR

VAR

    ipUdpRecv : ITcIoUdpProtocolRecv;

    ipUnknown : ITcUnknown;

END\_VAR

实现代码：

IF GuidsEqual(ADR(iid), ADR(TC\_GLOBAL\_IID\_LIST.IID\_ITcIoUdpProtocolRecv)) THEN

    ipUdpRecv := THIS^; // cast to interface pointer

    pipItf^ := ITCUNKNOWN\_TO\_PVOID(ipUdpRecv);

    TcAddRef();

    TcQueryInterface := S\_OK;

ELSIF GuidsEqual(ADR(iid), ADR(TC\_GLOBAL\_IID\_LIST.IID\_ITcUnknown)) THEN

    ipUnknown := THIS^; // cast to interface pointer

    pipItf^ := ITCUNKNOWN\_TO\_PVOID(ipUnknown);

    TcAddRef();

    TcQueryInterface := S\_OK;

ELSE

    TcQueryInterface := E\_HRESULTAdsErr.NOINTERFACE ;

END\_IF

1. 在ReceiveData方法中处理接收到的通讯数据，实现代码如下：

//Recv Info

UdpRecvAdr.ipadrInternal := ipAddr;

uiRecvPort := udpSrcPort;

uiReceivedPakets := uiReceivedPakets + 1;

udiLastReturnedMessageLength := nData; //nonstripped

//stripped to max length of string

MEMCPY(ADR(sLastReturnedMessage), pData, MIN(nData, SIZEOF(sLastReturnedMessage)-1) );

//add a 00 Byte at end of message

MEMSET(ADR(sLastReturnedMessage)+MIN(nData, SIZEOF(sLastReturnedMessage)-1)+1, 0, 1);

1. 在功能块主体中添加CheckReceive()调用，同时增加条件发送功能，代码如下：

IF ipUdp <> 0 THEN

    ipUdp.CheckReceived();

    IF bSend THEN

        hrSend := ipUdp.SendData( ipDestAddr:= UdpTargetAdr.ipadrInternal,

                                  udpDestPort:= uiTargetPort,

                                  udpSrcPort:= uiLocalPort,

                                  nData:= SIZEOF(sMessage),

                                  pData:= ADR(sMessage),

                                  bCalcUdpCheckSum:= FALSE,

                                  pVlan:= 0);

        IF SUCCEEDED(hrSend) THEN

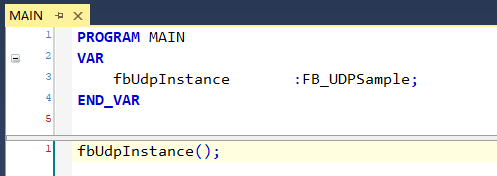
            udiSendCount := udiSendCount + 1;

        END\_IF

    END\_IF

END\_IF

1. 创建功能块实例，并在Main中调用实现，代码截图如下：

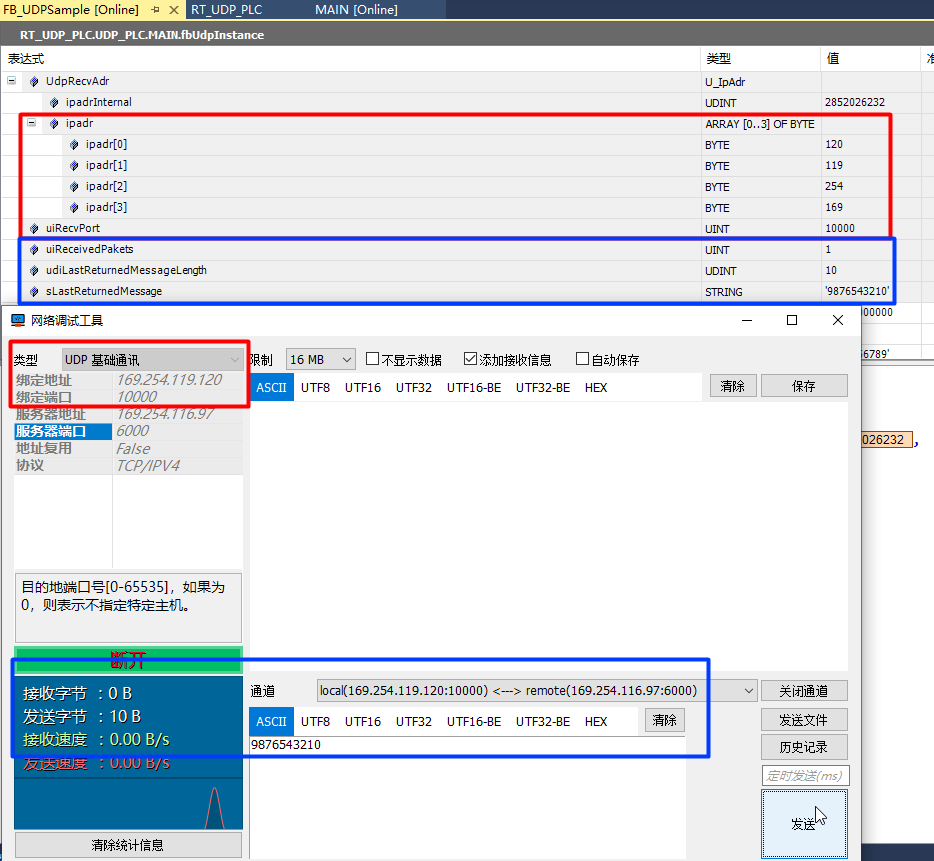


1. 程序编写完成，执行编译程序功能
2. 双击PLC对象实例—>选择" Symbol Initialization"标签—>为oid变量分配值

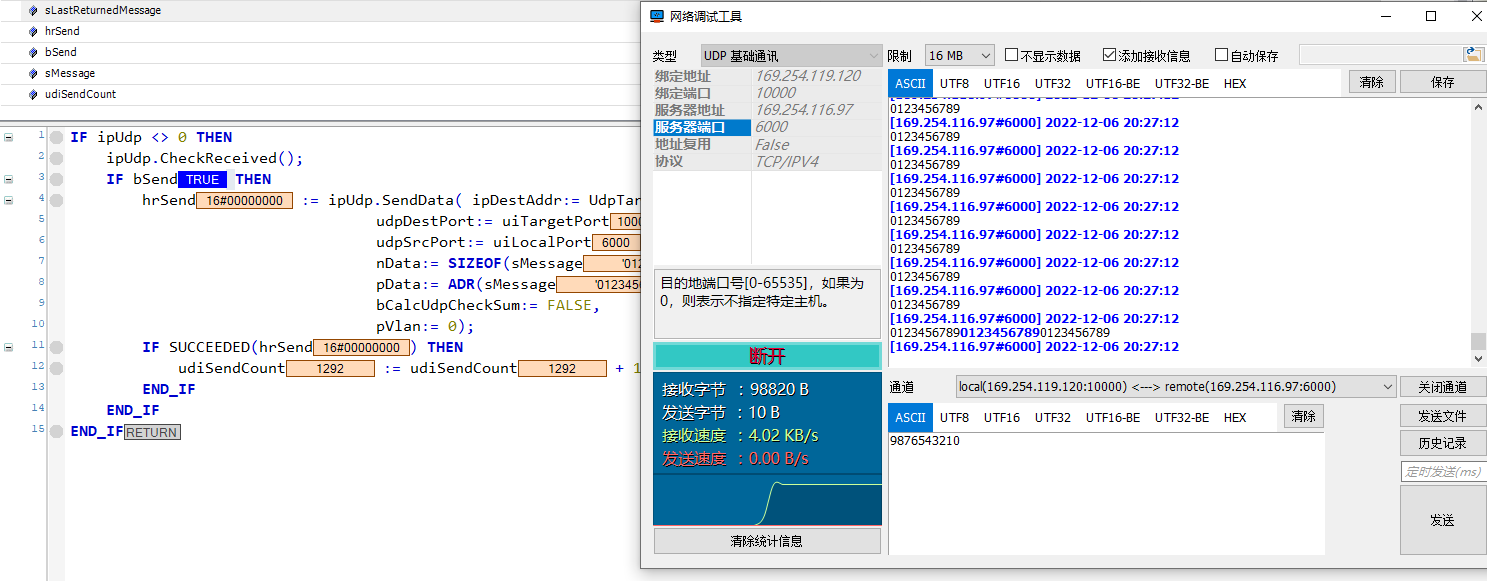
图形用户界面, 文本, 应用程序

描述已自动生成

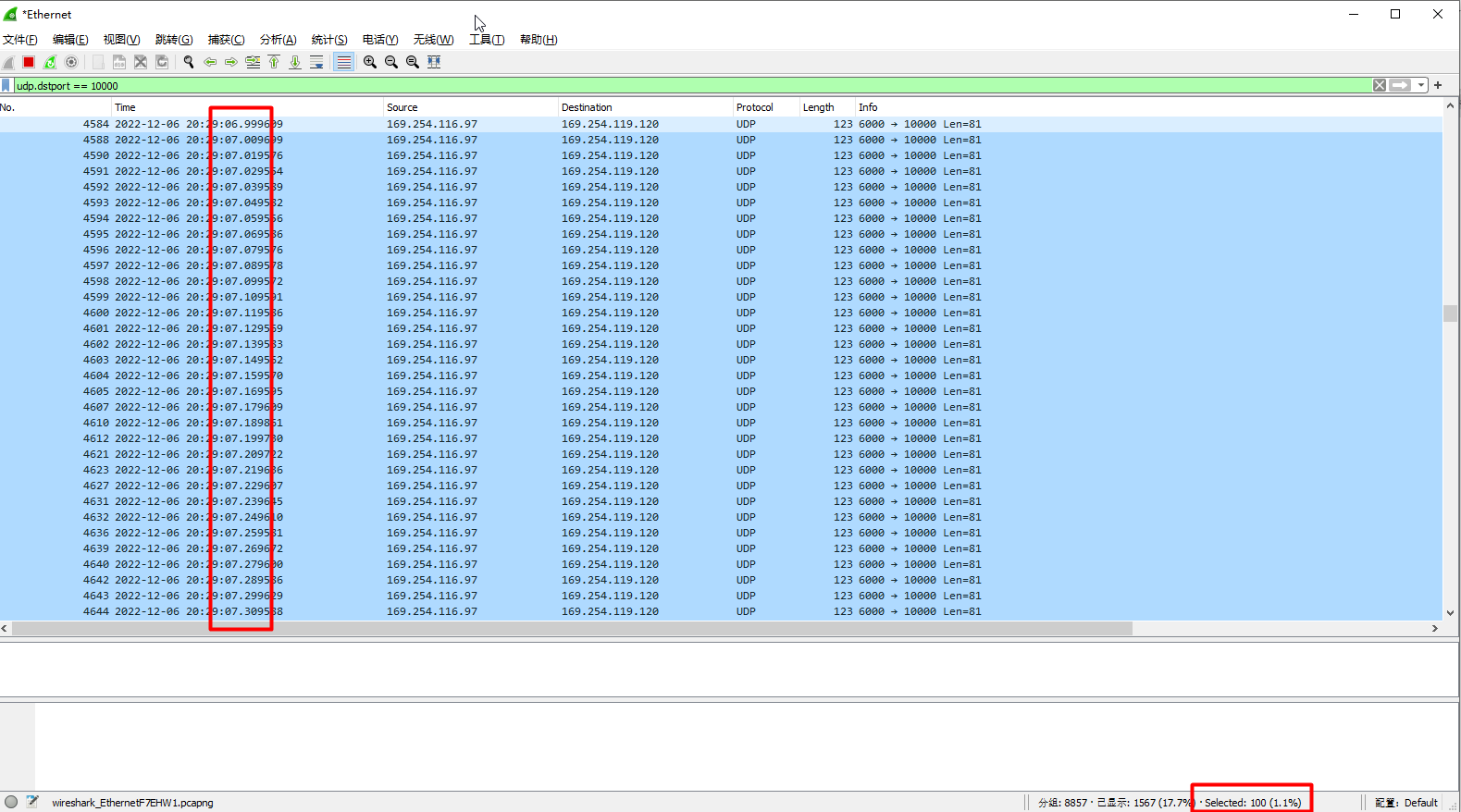
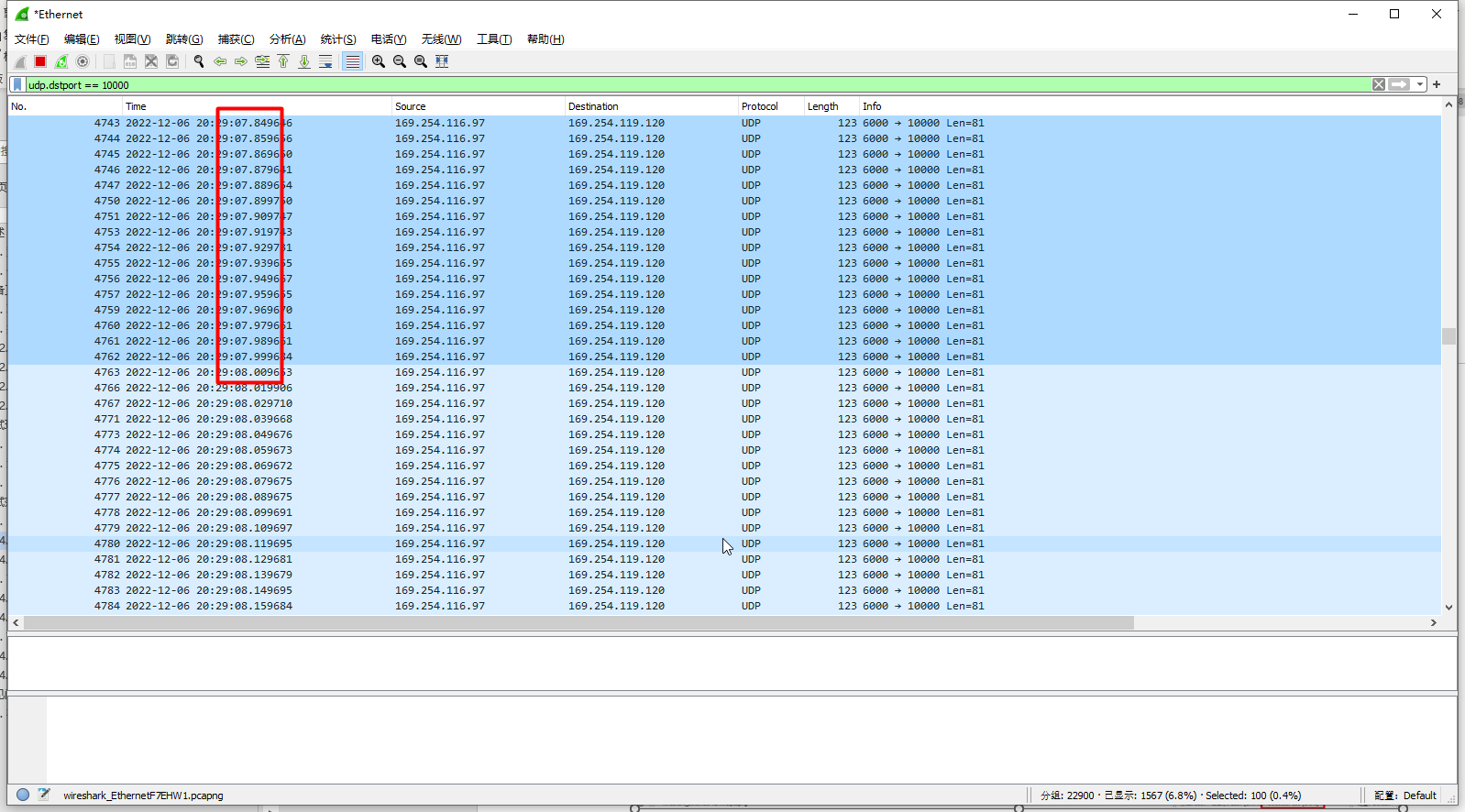
1. 激活配置—>通讯测试，测试环境为CX5020与PC调试助手
2. 接收来自调试助手的数据



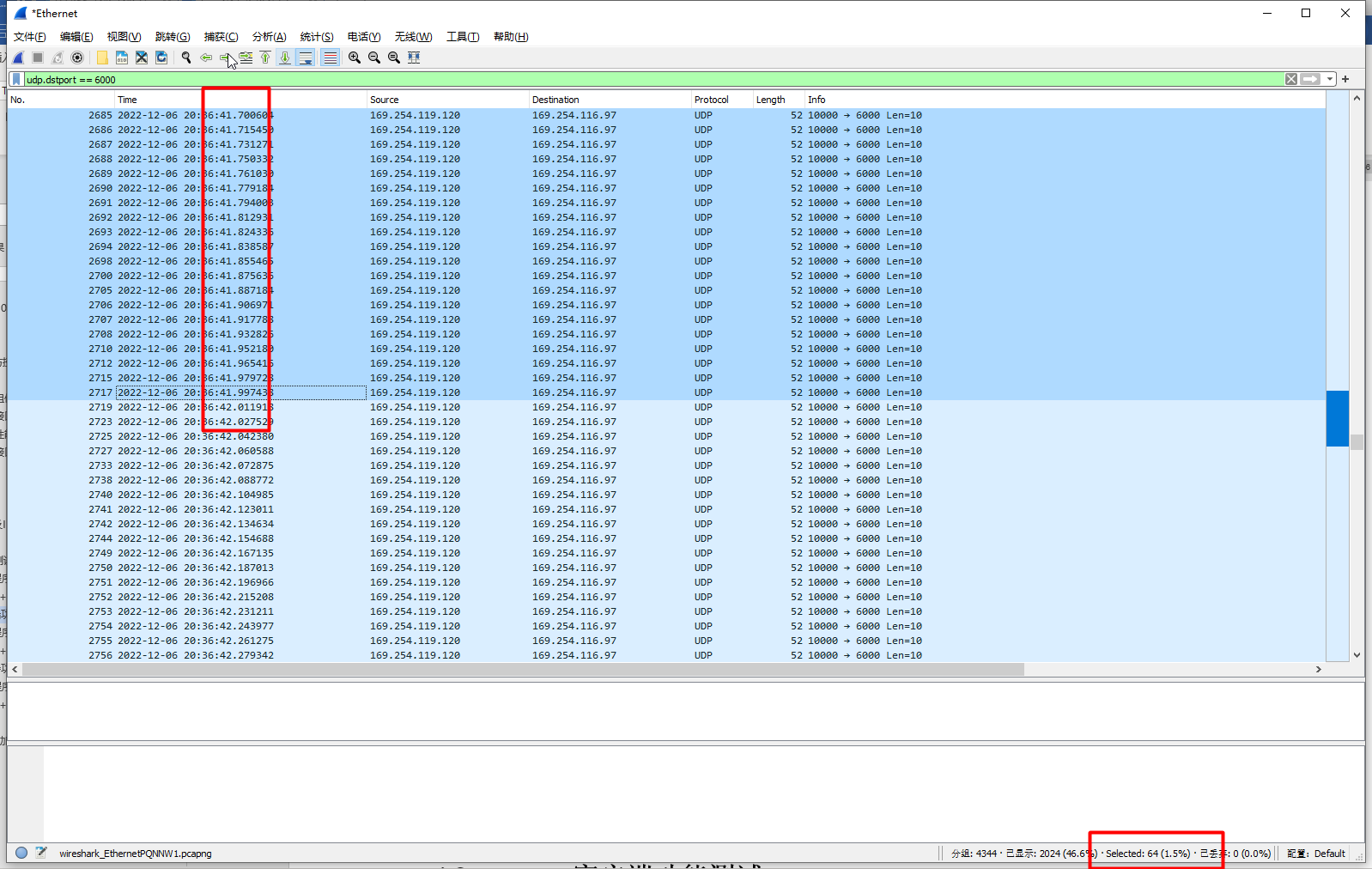
1. 将bSend变量置位TRUE发送数据



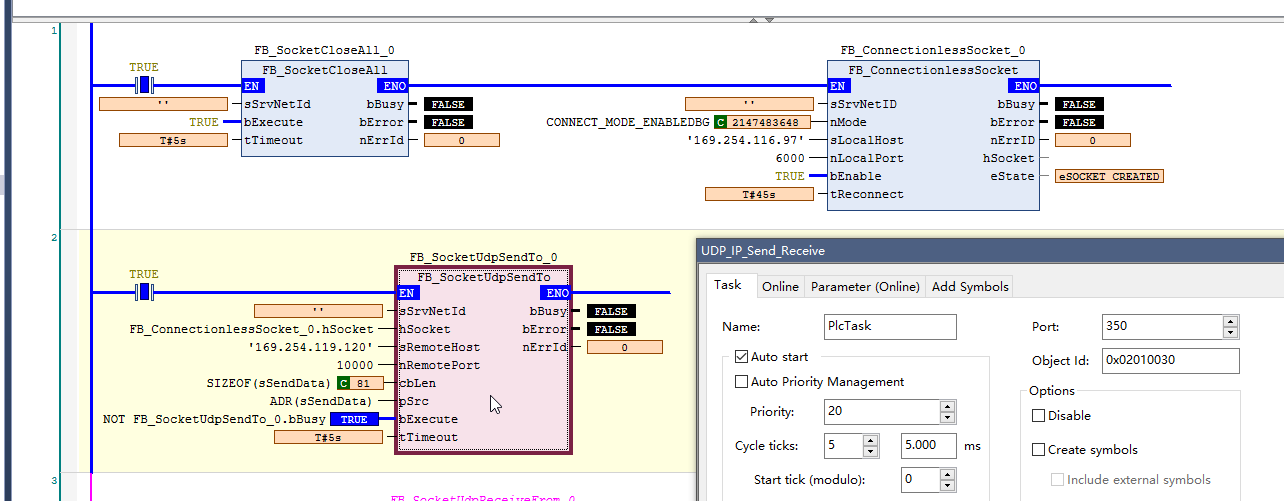
1. 使用wireshark展示通讯效能—>当控制器开始发送数据时进行报文抓取—>任选一秒的时间，周期内抓取到的报文数都为100条（PLC周期默认设置为10ms）



1. 当网络调试助手以10ms的周期发送数据—>报文的间隔周期不稳定且一秒时间没有发送出理论量的报文



1. 相同环境下使用TF6310功能库进行UDP通讯测试，因为是通过ADS间接与网卡通讯，需要将PLC task周期设置为5ms才能保证较为稳定每秒100条数据帧的通讯量



### TcC++程序实现

1. 创建另一个TwinCAT C++工程—>设备树中选择IO—>Devices，选中Devices右键添加新项—>选择Ethernet—>Real-Time Ethernet Adapter

图形用户界面, 文本, 应用程序

描述已自动生成 图形用户界面, 文本, 应用程序

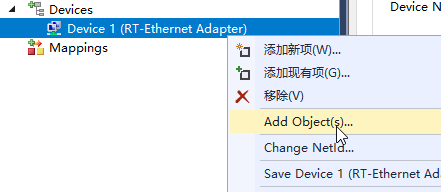
描述已自动生成

1. 确认设备已添加—>确认Adapter选项中Network Adapter内容非空

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. 选择Device1(Real-Time Ethernet Adapter)—>点击右键—>选择 "Add Object(s)..."，并完成"TCP/UDP RT "组件的添加

 图形用户界面, 应用程序

描述已自动生成

1. 添加完成后的设备树

文本, 应用程序

描述已自动生成

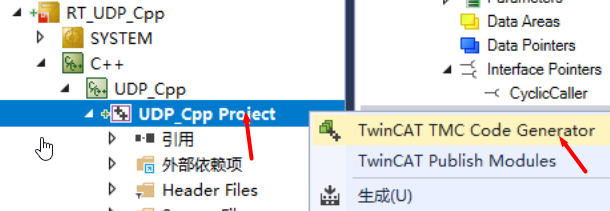
1. 添加TwinCAT C++工程，选择工程TMC文件添加需要实现的接口与配置接口指针

图形用户界面, 文本, 应用程序

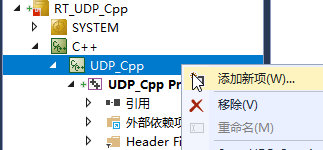
描述已自动生成 图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. 选择TwinCAT C++工程—>点击鼠标右键—>选择“TwinCAT TMC Code Generator”选项

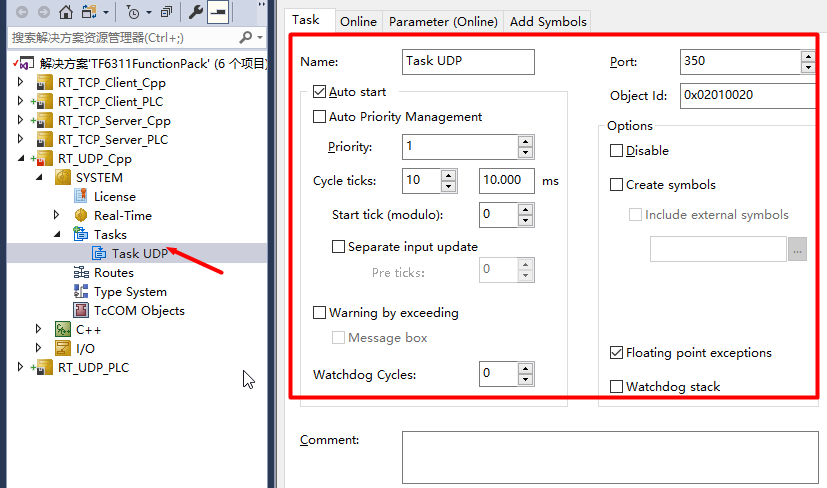


1. 编译成功后将TcCOM组件添加到工程中

 图形用户界面, 应用程序

描述已自动生成

1. 在System—>Task选项中添加新任务用作C++Module的刷新



1. 在Cpp文件中添加实现代码：SetObjStateSO()

**HRESULT** CModule1::SetObjStateSO()

{

    m\_Trace.Log(tlVerbose, FENTERA);

**HRESULT** hr = S\_OK;

    // TODO: Add any additional initialization

**if** (SUCCEEDED(hr) && m\_spipUdp.HasOID())

    {

        m\_Trace.Log(tlInfo, FLEAVEA "Register UdpProt");

**if** (SUCCEEDED\_DBG(hr = m\_spSrv->TcQuerySmartObjectInterface(m\_spipUdp)))

        {

            m\_Trace.Log(tlInfo, FLEAVEA "Server: UdpProt listen to Port: %d", 10000);

**if** (FAILED(hr = m\_spipUdp->RegisterReceiver(10000,

                THIS\_CAST(ITcIoUdpProtocolRecv))))

            {

                m\_Trace.Log(tlAlways, FLEAVEA "Server: UdpProtRegisterReceiver failed on Port:%d", 10000);

                m\_spipUdp = NULL;

            }

        }

    }

    // If following call is successful the CycleUpdate method will be called,

    // possibly even before this method has been left.

    hr = FAILED(hr) ? hr : AddModuleToCaller();

    // Cleanup if transition failed at some stage

**if** ( FAILED(hr) )

    {

**if** (m\_spipUdp != NULL)

            m\_spipUdp->UnregisterReceiver(10000);

        m\_spipUdp = NULL;

        RemoveModuleFromCaller();

    }

    m\_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);

**return** hr;

}

1. 在Cpp文件中添加实现代码：SetObjStateOS ()

**HRESULT** CModule1::SetObjStateOS()

{

    m\_Trace.Log(tlAlways, FENTERA);

**HRESULT** hr = S\_OK;

**if** (m\_spipUdp != NULL)

        m\_spipUdp->UnregisterReceiver(10000);

    m\_Trace.Log(tlAlways, FLEAVEA "hr=0x%08x", hr);

**return** hr;

}

1. 在Cpp文件中添加实现代码：ReceiveData ()

**HRESULT** hr = S\_OK;

ipRemoteAdr = ipAddr;

udpRemotePort = udpSrcPort;

udpLocalPort = udpDestPort;

m\_Trace.Log(tlAlways, FLEAVEA "UDP ReceiveData: IP: %d.%d.%d.%d udpSrcPort: %d DataSize: %d (hr2 = %x) \n",  ((**PBYTE**)&ipAddr)[3], ((**PBYTE**)&ipAddr)[2], ((**PBYTE**)&ipAddr)[1], ((**PBYTE**)&ipAddr)[0], udpSrcPort, nData, hr);

**return** hr;

1. 在Cpp文件中添加实现代码：CycleUpdate ()

**HRESULT** CModule1::CycleUpdate(ITcTask\* ipTask, ITcUnknown\* ipCaller, **ULONG\_PTR** context)

{

**HRESULT** hr = S\_OK, hrSend = S\_OK;

    // TODO: Add your cyclic code here

    m\_spipUdp->CheckReceived(); // ADDED

**if** (bSend == **true** && ipRemoteAdr != 0 && udpRemotePort != 0 && udpLocalPort != 0)

    {

        bSend = **false**;

        hrSend = m\_spipUdp->SendData(ipRemoteAdr, udpRemotePort, udpLocalPort, strlen(sMessage), sMessage, **true**);

**if** SUCCEEDED(hrSend) {

            m\_counter++;

        }

**else**

        {

            m\_Trace.Log(tlAlways, FLEAVEA "Send Message Failed hr=0x%08x", hr);

        }

    }

**return** hr;

}

1. 编译成功后配置C++Module—>分配任务与指针接口

图形用户界面

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

1. 调试时需要对程序中的变量状态进行修改，需要将C++的调试功能开启

图形用户界面, 应用程序

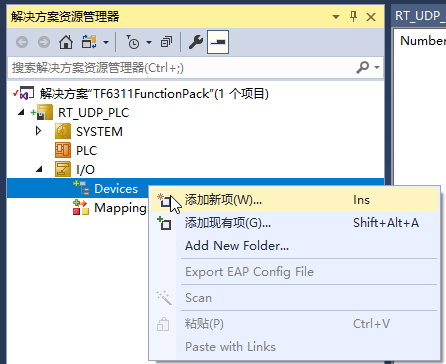
描述已自动生成

## TCP客户端功能测试

实现基于TCP/IP协议的通讯连接建立与数据接收发送的功能，需要使用ITcIoTcpProtocol和ITcIoTcpProtocolRecv接口。

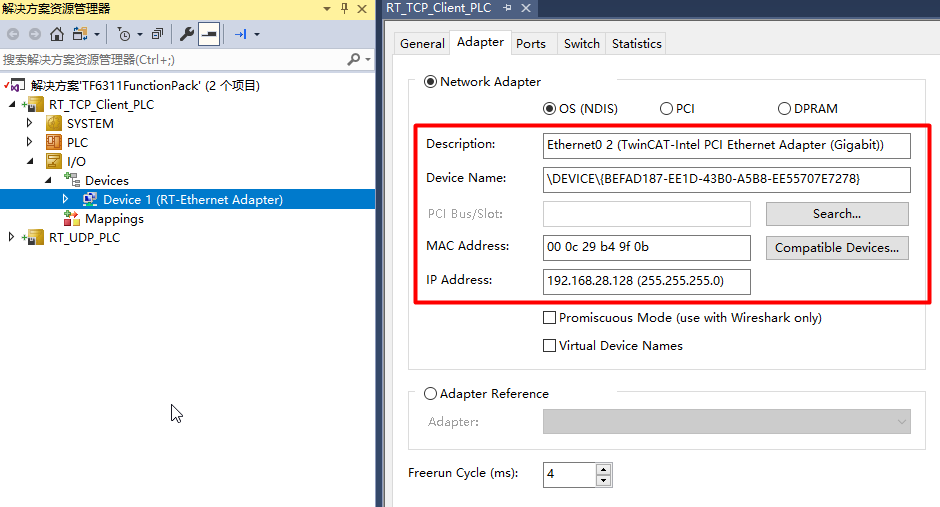
### PLC程序实现

1. 创建另一个TwinCAT工程—>设备树中选择IO—>Devices，选中Devices右键添加新项—>选择Ethernet—>Real-Time Ethernet Adapter

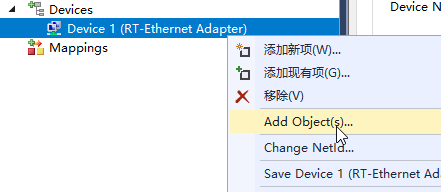
 图形用户界面, 文本, 应用程序

描述已自动生成

1. 确认设备已添加—>确认Adapter选项中Network Adapter内容非空

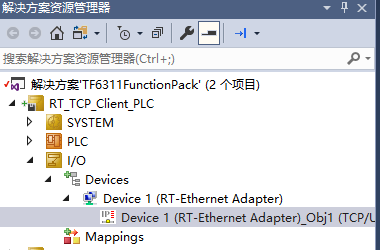


1. 选择Device1(Real-Time Ethernet Adapter)—>点击右键—>选择 "Add Object(s)..."，并完成"TCP/UDP RT "组件的添加

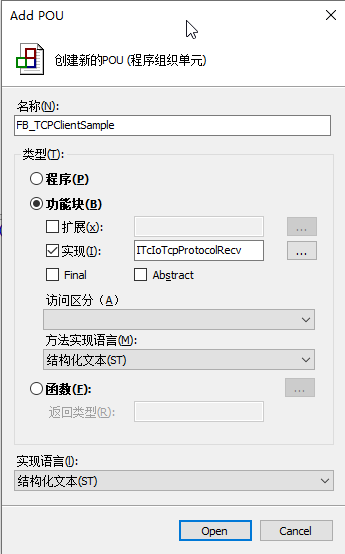
 图形用户界面, 应用程序

描述已自动生成

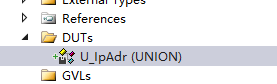
1. 添加完成后的设备树



1. 添加标准PLC工程—>添加功能块并实现ITcIoTcpProtocolRecv接口



1. 添加联合体U\_IpAdr用作IP地址转换



TYPE U\_IpAdr :

UNION

    ipadrInternal   : UDINT;

    ipadr           : ARRAY[0..3] OF **BYTE**;

END\_UNION

END\_TYPE

1. 功能块中变量声明及说明

{attribute 'c++\_compatible'}

FUNCTION\_BLOCK FB\_TCPClientSample IMPLEMENTS ITcIoTcpProtocolRecv

VAR

    {attribute 'TcInitSymbol'}

    //Object

    oid                 :OTCID;

    ipTcp               :ITcIoTcpProtocol;

    //Init

    hrInit              :**HRESULT**;

    tcpServerAdr        :U\_IpAdr;

    uiServerPort        :**UINT** := 6000;

    //Connect

    nSocketId       :UDINT;

    nConnections    :UDINT;

    //Recv

hrRecv                          :**HRESULT**;

lastTcpEvent :TCPIP\_EVENT;

    uiReceivedPakets                :**UINT**;

    udiLastReturnedMessageLength    :UDINT;

    sLastReturnedMessage            :STRING;

    //Send

    hrSend              :**HRESULT**;

    bSend               :**BOOL** := FALSE;

    sMessage            :STRING := '0123456789';

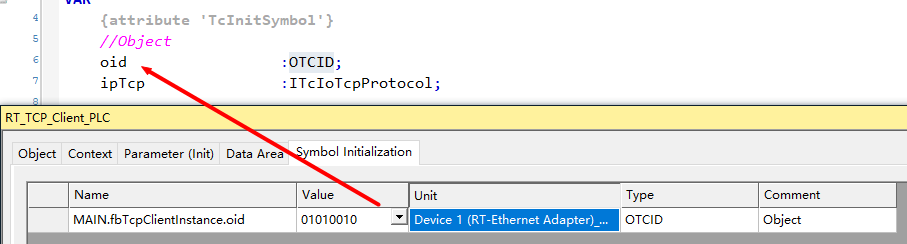
    udiSentData         :UDINT;

    udiSendCount        :UDINT;

END\_VAR

attribute 'TcInitSymbol'：使PLC编译器生成的VTable与C++编译器的VTable二进制兼容，使得可以通过PLC中的接口方法，访问C++实现的TcCom组件，在本例中即是通过PLC中的接口方法访问"TCP/UDP RT "组件。

attribute 'TcInitSymbol'：使得定义的变量可以作为初始化符号使用，编译完成后包含该属性的变量在PLC实例的 " Symbol Initialization"标签中出现。这些变量的值，在代码执行开始前被复制到变量值中，并覆盖在变量声明时指定的初始值，在本例中用作传递"TCP/UDP RT "组件的ID信息，如下图所示。



1. 依次添加FB\_init、FB\_exit、FB\_reinit方法

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. FB\_init方法中的代码实现：获取"TCP/UDP RT "组件实例的接口指针

IF NOT bInCopyCode THEN // no online change

IF ipTcp = 0 AND oid <> 0 THEN

hrInit := FW\_ObjMgr\_GetObjectInstance(  oid:=oid,

iid:=TC\_GLOBAL\_IID\_LIST.IID\_ITcIoTcpProtocol,

pipUnk:=ADR(ipTcp) );

IF SUCCEEDED(hrInit) THEN

FB\_init := TRUE;

ELSE

FB\_init := FALSE;

FW\_SafeRelease(ADR(ipTcp));

END\_IF

ELSIF oid = 0 THEN

FB\_init := FALSE;

hrInit := ERR\_INVALID\_PARAM;

END\_IF

END\_IF

////generate adress

//192.168.28.1

tcpServerAdr.ipadr[3] := 192;

tcpServerAdr.ipadr[2] := 168;

tcpServerAdr.ipadr[1] := 28;

tcpServerAdr.ipadr[0] := 1;

FW\_ObjMgr\_GetObjectInstance()：来自TC3\_Module库的函数，该函数通过输入Object ID，返回该Object ID指向的对象实例的接口指针，本例中返回"TCP/UDP RT "组件实例的接口指针。

FW\_SafeRelease()：来自TC3\_Module库的函数，该函数减少该接口指针的引用计数，并同时将涉及的接口指针设置为零，如果没有有效管理可能导致内存崩溃或内存泄漏问题。

1. FB\_exit方法中的代码实现：释放套接字并安全释放接口指针

IF NOT bInCopyCode THEN // no online change

    IF ipTcp <> 0 THEN

        ipTcp.FreeSocket(nSocketId); // Shutdown

        FW\_SafeRelease(ADR(ipTcp));

        FB\_exit := TRUE;

    END\_IF

END\_IF

FreeSocket()方法来自ITcIoTcpProtocol接口，用作释放一个套接字。

1. FB\_reinit方法中的代码实现：在线变更后重新调用AllocSocket()以更新回调目标

//reregister THIS^ for online changes

IF (ipTcp <> 0 AND nSocketId <> 0) THEN

    ipTcp.AllocSocket(THIS^, nSocketId);

    FB\_reinit := TRUE;

END\_IF

1. 当接收到通讯数据时，ITcIoUdpProtocol.CheckReceived()开始处理，自动调用TcQueryInterface方法查询/验证当前接收到的是否是某个特定的接口，如果是QueryInterFace将返回一个指向该接口的指针。在本例中调用TcQueryInterface方法查询/验证"TCP/UDP RT "组件反馈的是否是ITcIoUdpProtocolRecv接口，如果是则返回指向该接口的指针，而在本例中该接口指针也就是“THIS^”，以下为代码实现：

方法中两个中间变量，用作接口指针的传递

METHOD TcQueryInterface : **HRESULT**

VAR\_INPUT

    iid : REFERENCE TO IID;

    pipItf  : POINTER TO **PVOID**;

END\_VAR

VAR

    ipUdpRecv : ITcIoUdpProtocolRecv;

    ipUnknown : ITcUnknown;

END\_VAR

实现代码：

IF GuidsEqual(ADR(iid), ADR(TC\_GLOBAL\_IID\_LIST.IID\_ITcIoUdpProtocolRecv)) THEN

    ipUdpRecv := THIS^; // cast to interface pointer

    pipItf^ := ITCUNKNOWN\_TO\_PVOID(ipUdpRecv);

    TcAddRef();

    TcQueryInterface := S\_OK;

ELSIF GuidsEqual(ADR(iid), ADR(TC\_GLOBAL\_IID\_LIST.IID\_ITcUnknown)) THEN

    ipUnknown := THIS^; // cast to interface pointer

    pipItf^ := ITCUNKNOWN\_TO\_PVOID(ipUnknown);

    TcAddRef();

    TcQueryInterface := S\_OK;

ELSE

    TcQueryInterface := E\_HRESULTAdsErr.NOINTERFACE ;

END\_IF

1. 在ReceiveEvent()方法中添加TCP/IP连接事件处理代码：

CASE tcpEvent OF

    TCPIP\_EVENT\_ERROR:

        nSocketId := 0;

    TCPIP\_EVENT\_RESET:

        ipTcp.FreeSocket(socketId);

        nSocketId := 0;

    TCPIP\_EVENT\_TIMEOUT:

        ipTcp.FreeSocket(socketId);

        nSocketId := 0;

    TCPIP\_EVENT\_CONN\_CLOSED:

        ipTcp.FreeSocket(socketId);

        nSocketId := 0;

    TCPIP\_EVENT\_CONN\_INCOMING:

        ;

    TCPIP\_EVENT\_KEEP\_ALIVE:

        ;

    TCPIP\_EVENT\_CONN\_IDLE:

        ;

    TCPIP\_EVENT\_DATA\_SENT:

        ;

    TCPIP\_EVENT\_DATA\_RECEIVED:

        ;

    TCPIP\_EVENT\_LINKCONNECT:

        ;

    TCPIP\_EVENT\_LINKDISCONNECT:

        ipTcp.FreeSocket(socketId);

        nSocketId := 0;

END\_CASE;

lastTcpEvent := tcpEvent;

ReceiveEvent := S\_OK;

此方法主要用作响应TCP/IP通讯不同连接状态时客户端对应的动作

1. 在ReceiveData()方法中处理接收到的通讯数据，实现代码如下：

uiReceivedPakets := uiReceivedPakets+1;

udiLastReturnedMessageLength := nData; //nonstripped

//stripped to max length of string

MEMCPY(ADR(sLastReturnedMessage), pData, MIN(nData, SIZEOF(sLastReturnedMessage)-1) );

//add a 00 Byte at end of message

MEMSET(ADR(sLastReturnedMessage)+MIN(nData, SIZEOF(sLastReturnedMessage)-1)+1, 0, 1);

1. 在功能块主体中添加CheckReceived()方法调用，并连接到目标服务器设备与编辑触发发送数据给服务器的程序功能

IF(ipTcp <> 0) THEN

    ipTcp.CheckReceived();

    IF (nSocketId = 0) THEN

        //! THIS^ might get invalid due to online change! "Reallocate" by nSocketId <>0

        IF(ipTcp.AllocSocket(THIS^, nSocketId) = 0) THEN

            IF(ipTcp.Connect(nSocketId, tcpServerAdr.ipadrInternal, uiServerPort) <> 0) THEN

                ipTcp.FreeSocket(nSocketId);

                nSocketId := 0;

            ELSE

                nConnections := nConnections+1;

            END\_IF

        END\_IF

ELSIF bSend THEN

bSend := FALSE;

        hrSend := ipTcp.IsConnected(nSocketId);

        IF (hrSend = S\_OK) THEN

            hrSend := ipTcp.SendData(nSocketId, len(sMessage), ADR(sMessage), udiSentData);

            IF SUCCEEDED(hrSend) THEN

                udiSendCount := udiSendCount + 1;

            END\_IF

        ELSE

            ADSLOGDINT( msgCtrlMask := ADSLOG\_MSGTYPE\_HINT OR ADSLOG\_MSGTYPE\_MSGBOX,

                        msgFmtStr := 'Not yet connected socket %d',

                        dintArg := nSocketId);

        END\_IF

    END\_IF

END\_IF

AllocSocket()：该方法来自ITcIoTcpProtocol接口，用作给通讯组件分配一个套接字

Connect()：该方法来自ITcIoTcpProtocol接口，用作使通讯组件与服务器连接

IsConnected()：该方法来自ITcIoTcpProtocol接口，用作检查套接字是否被连接

SendData()：该方法来自ITcIoTcpProtocol接口，用作向服务器发送数据

ADSLOGDINT()：该函数来自Tc2\_System函数库，用作在屏幕上发出一个指定文本的消息框，并将该信息写入系统的日志

1. 创建功能块实例，并在Main函数中进行调用

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. 程序编写完成，执行编译程序功能
2. 双击PLC对象实例—>选择" Symbol Initialization"标签—>为oid变量分配值

表格

描述已自动生成

1. 激活配置—>通讯测试，测试环境为虚拟机PC与PC调试助手
2. 程序运行之后自动完成后PC端调试助手创建的服务器连接

图形用户界面, 文本, 应用程序

描述已自动生成

1. 从调试助手发送通讯数据到虚拟机PC中运行TwinCAT程序

图形用户界面, 应用程序

描述已自动生成

1. 触发PLC控制程序中的bSend变量，开始向调试助手发送数据

图形用户界面

描述已自动生成

TCP/IP通讯是可靠的通讯协议，在发送端发送数据后需要接收到远端设备的回复信息才会发送下一条通讯报文或数据，本例中发送端的发送命令周期为PLC默认周期10ms，由于远端反馈不及时会导致数据先暂存在发送端的发送缓存区，等待可以再次发送时再一起发送给远端设备，也就出现了我们测试所看到的粘包现象。

1. 当服务端关闭连接后出发发送功能，客户端程序报警弹出提示框

图形用户界面

描述已自动生成

### TcC++程序实现

配置步骤与UDP功能的TcC++程序实现相似，详细代码请参考配套例程。

## TCP服务器功能测试

实现基于TCP/IP协议的通讯连接建立与数据接收发送的功能，需要使用ITcIoTcpProtocol和ITcIoTcpProtocolRecv接口。

### PLC程序实现

1. 创建另一个TwinCAT工程—>设备树中选择IO—>Devices，选中Devices右键添加新项—>选择Ethernet—>Real-Time Ethernet Adapter

图形用户界面, 文本, 应用程序

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

1. 确认设备已添加—>确认Adapter选项中Network Adapter内容非空

图形用户界面, 应用程序

描述已自动生成

1. 选择Device1(Real-Time Ethernet Adapter)—>点击右键—>选择 "Add Object(s)..."，并完成"TCP/UDP RT "组件的添加

图形用户界面, 文本, 应用程序

描述已自动生成 图形用户界面, 应用程序

描述已自动生成

1. 添加完成后的设备树

文本

描述已自动生成

1. 添加标准PLC工程—>添加功能块并实现ITcIoTcpProtocolRecv接口

图形用户界面, 应用程序

描述已自动生成

1. 添加联合体U\_IpAdr用作IP地址转换

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

TYPE U\_IpAdr :

UNION

    ipadrInternal   : UDINT;

    ipadr           : ARRAY[0..3] OF **BYTE**;

END\_UNION

END\_TYPE

1. 功能块中变量声明及说明

{attribute 'c++\_compatible'}

FUNCTION\_BLOCK FB\_TCPServerSample IMPLEMENTS ITcIoTcpProtocolRecv

VAR

    {attribute 'TcInitSymbol'}

    //Object

    oid                 :OTCID;

    ipTcp               :ITcIoTcpProtocol;

    //Init

    hrInit              :**HRESULT**;

    nSocketId           :UDINT;

    uiServerPort        :**UINT** := 6000;

    //Connect

    udiAcceptedID       :UDINT;

    //Recv

    hrRecv                          :**HRESULT**;

    lastTcpEvent                    :TCPIP\_EVENT;

    tcpRemoteAdr                    :U\_IpAdr;

    uiReceivedPakets                :**UINT**;

    udiLastReturnedMessageLength    :UDINT;

    sLastReturnedMessage            :STRING;

    //Send

    hrSend              :**HRESULT**;

    bSend               :**BOOL** := FALSE;

    sMessage            :STRING := 'TestMessage';

    udiSentData         :UDINT;

    udiSendCount        :UDINT;

END\_VAR

attribute 'TcInitSymbol'：使PLC编译器生成的VTable与C++编译器的VTable二进制兼容，使得可以通过PLC中的接口方法，访问C++实现的TcCom组件，在本例中即是通过PLC中的接口方法访问"TCP/UDP RT "组件。

attribute 'TcInitSymbol'：使得定义的变量可以作为初始化符号使用，编译完成后包含该属性的变量在PLC实例的 " Symbol Initialization"标签中出现。这些变量的值，在代码执行开始前被复制到变量值中，并覆盖在变量声明时指定的初始值，在本例中用作传递"TCP/UDP RT "组件的ID信息，如下图所示。

图形用户界面, 应用程序

描述已自动生成

1. 依次添加FB\_init、FB\_exit、FB\_reinit方法

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. FB\_init方法中的代码实现：获取"TCP/UDP RT "组件实例的接口指针

IF NOT bInCopyCode THEN // no online change

    IF ipTcp = 0 THEN

        hrInit := FW\_ObjMgr\_GetObjectInstance(  oid:=oid,

                                                iid:=TC\_GLOBAL\_IID\_LIST.IID\_ITcIoTcpProtocol,

                                                pipUnk:=ADR(ipTcp) );

        IF SUCCEEDED(hrInit) THEN

            IF SUCCEEDED(ipTcp.AllocSocket(THIS^, nSocketId)) THEN  //startup -> open port

                IF SUCCEEDED(ipTcp.Listen(nSocketId, uiServerPort)) THEN

                    FB\_init := TRUE;

                ELSE

                    FW\_SafeRelease(ADR(ipTcp));

                END\_IF

            ELSE

                FW\_SafeRelease(ADR(ipTcp));

            END\_IF

        ELSE

            FW\_SafeRelease(ADR(ipTcp));

        END\_IF

    END\_IF

END\_IF

FW\_ObjMgr\_GetObjectInstance()：来自TC3\_Module库的函数，该函数通过输入Object ID，返回该Object ID指向的对象实例的接口指针，本例中返回"TCP/UDP RT "组件实例的接口指针。

FW\_SafeRelease()：来自TC3\_Module库的函数，该函数减少该接口指针的引用计数，并同时将涉及的接口指针设置为零，如果没有有效管理可能导致内存崩溃或内存泄漏问题。

1. FB\_exit方法中的代码实现：释放套接字与已建立的连接并安全释放接口指针

IF NOT bInCopyCode THEN // no online change

    IF ipTcp <> 0 THEN

        IF (nSocketId <> 0) THEN

            ipTcp.FreeSocket(nSocketId);

        END\_IF

        IF (udiAcceptedID <> 0) THEN

            ipTcp.FreeSocket(udiAcceptedID);

        END\_IF

        FW\_SafeRelease(ADR(ipTcp));

        FB\_exit := TRUE;

    END\_IF

END\_IF

1. FB\_reinit方法中的代码实现：在线变更后重新调用AllocSocket()以更新回调目标

//reregister THIS^ for online changes

IF (ipTcp <> 0 AND nSocketId <> 0) THEN

    ipTcp.AllocSocket(THIS^, nSocketId);

    FB\_reinit := TRUE;

END\_IF

IF (ipTcp <> 0 AND udiAcceptedID <> 0) THEN

    ipTcp.AllocSocket(THIS^, udiAcceptedID);

    FB\_reinit := TRUE;

END\_IF

1. 当接收到通讯数据时，ITcIoUdpProtocol.CheckReceived()开始处理，自动调用TcQueryInterface方法查询/验证当前接收到的是否是某个特定的接口，如果是QueryInterFace将返回一个指向该接口的指针。在本例中调用TcQueryInterface方法查询/验证"TCP/UDP RT "组件反馈的是否是ITcIoUdpProtocolRecv接口，如果是则返回指向该接口的指针，而在本例中该接口指针也就是“THIS^”，以下为代码实现：

METHOD TcQueryInterface : **HRESULT**

VAR\_INPUT

    iid : REFERENCE TO IID;

    pipItf  : POINTER TO **PVOID**;

END\_VAR

VAR

    ipUdpRecv : ITcIoUdpProtocolRecv;

    ipUnknown : ITcUnknown;

END\_VAR

实现代码：

IF GuidsEqual(ADR(iid), ADR(TC\_GLOBAL\_IID\_LIST.IID\_ITcIoUdpProtocolRecv)) THEN

    ipUdpRecv := THIS^; // cast to interface pointer

    pipItf^ := ITCUNKNOWN\_TO\_PVOID(ipUdpRecv);

    TcAddRef();

    TcQueryInterface := S\_OK;

ELSIF GuidsEqual(ADR(iid), ADR(TC\_GLOBAL\_IID\_LIST.IID\_ITcUnknown)) THEN

    ipUnknown := THIS^; // cast to interface pointer

    pipItf^ := ITCUNKNOWN\_TO\_PVOID(ipUnknown);

    TcAddRef();

    TcQueryInterface := S\_OK;

ELSE

    TcQueryInterface := E\_HRESULTAdsErr.NOINTERFACE ;

END\_IF

1. 在ReceiveEvent()方法中添加TCP/IP连接事件处理代码：

CASE tcpEvent OF

    TCPIP\_EVENT\_ERROR:

        ;

    TCPIP\_EVENT\_RESET:

        ipTcp.FreeSocket(socketId);

    TCPIP\_EVENT\_TIMEOUT:

        ipTcp.FreeSocket(socketId);

    TCPIP\_EVENT\_CONN\_CLOSED:

        ipTcp.FreeSocket(socketId);

    TCPIP\_EVENT\_CONN\_INCOMING:

        ipTcp.Accept(socketId);

        //store socketid; used at least in init during onlinechange

        udiAcceptedID := socketId;

    TCPIP\_EVENT\_KEEP\_ALIVE:

        ;

    TCPIP\_EVENT\_CONN\_IDLE:

        ;

    TCPIP\_EVENT\_DATA\_SENT:

        ;

    TCPIP\_EVENT\_DATA\_RECEIVED:

        ;

    TCPIP\_EVENT\_LINKCONNECT:

        ;

    TCPIP\_EVENT\_LINKDISCONNECT:

        ipTcp.FreeSocket(socketId);

END\_CASE;

lastTcpEvent := tcpEvent;

ReceiveEvent := S\_OK;

此方法主要用作响应TCP/IP通讯不同连接状态时客户端对应的动作

1. 在ReceiveData()方法中处理接收到的通讯数据，实现代码如下：

IF ipTcp <> 0 THEN

    ipTcp.GetRemoteIpAddr(socketId, tcpRemoteAdr.ipadrInternal);

END\_IF

uiReceivedPakets := uiReceivedPakets+1;

udiLastReturnedMessageLength := nData; //nonstripped

//stripped to max length of string

MEMCPY(ADR(sLastReturnedMessage), pData, MIN(nData, SIZEOF(sLastReturnedMessage)-1) );

//add a 00 Byte at end of message

MEMSET(ADR(sLastReturnedMessage)+MIN(nData, SIZEOF(sLastReturnedMessage)-1)+1, 0, 1);

1. 在功能块主体中添加CheckReceived()方法调用，并连接到目标服务器设备与编辑触发发送数据给服务器的程序功能

IF ipTcp <> 0 THEN

    ipTcp.CheckReceived();

    IF nSocketId <> 0

        AND\_THEN udiAcceptedID <> 0

        AND\_THEN bSend THEN

        bSend := FALSE;

        hrSend := ipTcp.SendData(udiAcceptedID, len(sMessage), ADR(sMessage), udiSentData);

        IF SUCCEEDED(hrSend) THEN

            udiSendCount := udiSendCount + 1;

        ELSE

            ADSLOGDINT(ADSLOG\_MSGTYPE\_HINT, 'Send Message failed', nSocketId);

        END\_IF

    END\_IF

END\_IF

SendData()：该方法来自ITcIoTcpProtocol接口，用作向客户端发送数据

ADSLOGDINT()：该函数来自Tc2\_System函数库，用作在屏幕上发出一个指定文本的消息框，并将该信息写入系统的日志

1. 创建功能块实例，并在Main函数中进行调用

图形用户界面, 文本, 应用程序

描述已自动生成

1. 程序编写完成，执行编译程序功能
2. 双击PLC对象实例—>选择" Symbol Initialization"标签—>为oid变量分配值

图形用户界面, 表格

描述已自动生成

1. 激活配置—>通讯测试，测试环境为虚拟机PC与PC调试助手
2. 程序运行之后自动完成后服务器的创建

图形用户界面, 应用程序

描述已自动生成

1. 使用调试助手建立与服务器的连接并进行收发数据测试

图形用户界面, 应用程序

描述已自动生成

### TcC++程序实现

配置步骤与UDP功能的TcC++程序实现相似，详细代码请参考配套例程。

**上海（ 中国区总部）**

中国上海市静安区汶水路 299 弄 9号（市北智汇园）

电话: 021-66312666

**北京分公司**

北京市西城区新街口北大街 3 号新街高和大厦 407 室

电话: 010-82200036 邮箱: beijing@beckhoff.com.cn

**广州分公司**

广州市天河区珠江新城珠江东路32号利通广场1303室

电话: 020-38010300/1/2 邮箱: guangzhou@beckhoff.com.cn

**成都分公司**

成都市锦江区东御街18号 百扬大厦2305 室

电话: 028-86202581 邮箱: chengdu@beckhoff.com.cn

|  |  |
| --- | --- |
| 请用微信扫描二维码  通过公众号与技术支持交流 | 倍福官方网站：  https://www.beckhoff.com.cn  在线帮助系统：  https://infosys.beckhoff.com/index\_en.htm |
| 倍福虚拟学院：  https://tr.beckhoff.com.cn/ |
| 招贤纳士：job@beckhoff.com.cn  技术支持：support@beckhoff.com.cn  产品维修：service@beckhoff.com.cn  方案咨询：sales@beckhoff.com.cn |
|  |

1. TF6311直接集成在TwinCAT系统中，所以不能使用Windows防火墙，在较大/未知的网络中，建议使用TF6310 [↑](#footnote-ref-1)
2. 预期能够传递数据的能力，时间上考虑也就是所说的实时性 [↑](#footnote-ref-2)
3. 网卡设备能够兼容并已安装EtherCAT驱动 [↑](#footnote-ref-3)