

Borland C++ Builder 例程说明

Mscomm 是微软提供给用户使用的,与串口设备进行通信的组件。本说明讲述如何使用 Mscomm 组件进行编程,实现与大连理工计算机控制工程有限公司 (简称: DCCE) 的 PLC 设备通信。主要包含 Mscomm 组件概述、modbus 协议介绍和在 Borland C++ Builder 下使用 Mscomm 等方面的内容。

环境准备

硬件需求: 大连理工计算机控制工程有限公司的 PLC 设备、有串口通信的计算机、串口数据线, RS232 RS485 转换器。

软件需求: Windows98/2000/XP 操作系统, 大连理工计算机控制工程有限公司 PLC 调试软件, Borland C++ Builder6.0。

准备流程:

1. 将设备通过 RS232-RS485 转换器以调试状态 (首先短接调试端与地,然后连接电源) 连接在计算机上, 运行与设备型号相对应的 PLC 调试软件设置串口参数, 由于本例程设置 Mscomm 参数为 N, 8, 1, 9600, 所以将设备的校验位, 数据位, 停止位, 波特率分别设置为: 无校验, 8, 1, 9600。同时要把将要与例程通信的端口设置为从口。最后要保存参数。
2. 设备断电, 将短接线移去, 重新通电, 直接运行本例程进行测试 (可能出现寄存器不能改写的情况, 解决方法是用调试软件清除 PLC 中的程序)。

Mscomm 概述

Microsoft Communications Control (以下简称 MSComm) 是 Microsoft 公司提供的简化 Windows 下串行通信编程的 ActiveX 控件, 它为应用程序提供了通过串行接口收发数据的简便方法。MSComm 控件在串口编程时非常方便, 程序员不必去花时间去了解较为复杂的 API 函数, 而且在 VC、VB、Delphi 等语言中均可使用。具体的来说, 它提供了两种处理通信问题的方法: 事件驱动(Event-driven)方法和查询法。

事件驱动通讯是处理串行端口交互作用的一种非常有效的方法。在许多情况下, 在事件发生时需要得到通知, 例如, 在串口接收缓冲区中有字符, 或者 Carrier Detect (CD) 或 Request To Send (RTS) 线上一个字符到达或一个变化发生时。在这些情况下, 可以利用 MSComm 控件的 OnComm 事件捕获并处理这些通讯事件。OnComm 事件还可以检查和处理通讯错误。所有通讯事件和通讯错误的列表, 参阅 CommEvent 属性。在编程过程中, 就可以在 OnComm 事件处理函数中加入自己的处理代码。这种方法的优点是程序响应及时, 可靠性高。每个 MSComm 控件对应着一个串行端口。如果应用程序需要访问多个串行端口, 必须使用多个 MSComm 控件。

常用属性

MSComm 编程序必须了解的属性:

属性	功能
CommPort	设置并返回通讯端口号，缺省为 COM1
Settings	以字符串的形式设置并返回波特率、奇偶校验、数据位、停止位
PortOpen	设置并返回通讯端口的状态。也可以打开和关闭端口
Input	从接收缓冲区返回和删除字符
Output	向传输缓冲区写一个字符串
InputLen	设置每次 Input 读入的字符个数，缺省值为 0，表明读取接收缓冲区中的全部内容
InBufferCount	返回接收缓冲区中已接收到的字符数，将其置 0 可以清除接收缓冲区
InputMode	定义 Input 属性获取数据的方式（为 0：文本方式；为 1：二进制方式）
RThreshold	和 SThreshold 属性，表示在 OnComm 事件发生之前，接收缓冲区或发送缓冲区中可以接收的字符数

CommPort 属性设置或返回通讯端口号。在设计时，value 可以设置成从 1 到 16 的任何数（缺省值为 1）。但是如果用 PortOpen 属性打开一个并不存在的端口时，MSComm 控件会产生错误 68（设备无效）。必须在打开端口之前设置 CommPort 属性。

Settings 属性设置或返回串口波特率、奇偶校验、数据位、停止位参数。参数格式为 "BBBB,P,D,S" ;其中,BBBB 为波特率, P 为奇偶校验, D 为数据位数, S 为停止位数。value 的缺省值是: "9600,N,8,1"。其它效验字符为: 奇效验----‘O’;偶效验----‘E’;

InputMode 属性设置或返回通信方式。comInputModeText(0, 缺省值) 通过 Input 属性以文本方式取回数据。comInputModeBinary(1) 通过 Input 属性以二进制方式检取回数据。

RThreshold 属性在 MSComm 控件发送数据前设置为要接收的字符数。当接收字符后，若 Rthreshold 属性设置为 0（缺省值）则不产生 OnComm 事件。若设置 Rthreshold 为 n，接收缓冲区收到 n 个字符都会使 MSComm 控件都将产生 OnComm 事件。

InputLen 属性设置或返回 Input 属性从接收缓冲区读取的字符数。Input 属性从接收缓冲区中读取的字符数。InputLen 属性的缺省值是 0。设置 InputLen 为 0 时,使用 Input 将使 MSComm 控件读取接收缓冲区中全部的内容。若接收缓冲区中 InputLen 字符无效，Input 属性返回一个零长度字符串 ("")。

InBufferCount 属性为缓冲区中已接收的字符数。该属性在从输出格式为定长数据的机器读取数据时非常有用。

使用方法

使用 MSComm 控件的一般使用方法为:

初始化串口: 通讯端口号 (CommPort)、串口配置(Settings)。打开串口 (PortOpen)。设置通信方式 (InputMode = 1)。

发送请求数据: 清空缓冲区(InBufferCount = 0)、设置读取的长度 (InputLen = 0)、设置触发事件的接收长度 (Rthreshold = n)、设置发送内容 (Output)。

接收事件处理： 取回串口中的字符长度(InBufferCount)、 取回返回的数据 (Input)。

Modbus 协议介绍

通讯命令

Modbus 协议是一种通信标准,包含 RTU 和 ASCII。我们只需要了解 RTU 的读写命令及其打包过程。与我们编程相关的 Modbus 协议大致格式是: 站地址 + 命令号 + 起始地址 + 操作数量 + 数据段 + 效验码。其回复格式为: 站地址 + 命令号 + 其它。

站地址一个 0~255 的设备标号,占用一个字节。命令号决定设备如何操作,如读位,写字,读寄存器,写寄存器等。起始地址是指寄存器在设备的地址编码,如 VW0 的绝对地址是 2336。操数量,是指操作(读或写)的单元格式。数据段只有写操作有,包含数据的字节长度和数据内容。 效验码用于检验传输数据的正确性。他们各自的命令格式及其回复见表 1 和表 2。

表 1 Modbus 读写命令格式

操作(命令号)	命令格式	字节数	举例（十六进制表示）
位 读 取 (01/02)	站地址(1) 功能号(1) 起始地址(2) 数量 (2) CRC(2)	8	01 01 00 01 00 02 CRC
字 读 取 (03/04)	站地址(1) 功能号(1) 起始地址(2) 数量 (2) CRC(2)	8	01 03 00 01 00 03 CRC
位写入(15)	站地址(1) 功能号(1) 起始地址(2) 操作位 数(2) 数据字节数(1) 数据(n) CRC(2)	9+n	01 0F 00 13 00 0A 02 CD 01 CRC
字写入(16)	站地址(1) 功能号(1) 起始地址(2) 操作字 节数(2) 数据字节数(1) 数据(n) CRC(2)	9+n	01 10 00 01 00 02 04 00 0A 01 02 CRC

表 2 Modbus 协议读写返回格式

操作(命令号)	返回数据格式	字节数	举例（十六进制表示）
位 读 取 (01/02)	站地址(1)-功能号(1)-字节数(1)-数 据(n)-CRC(2)	5+n	01 01 03 01 00 02 CRC
字 读 取 (03/04)	站地址(1)-功能号(1)-字节数(1)-数 据(n)-CRC(2)	5+n	01 03 04 01 00 03 01 CRC

位写入(15)	站地址(1)-功能号(1)-起始地址(2)- 8 操作位数(2)-CRC(2)	01 0F 00 13 00 0A CRC
字写入(16)	站地址(1)-功能号(1)-起始地址(2)- 8 操作字节数(2)-CRC(2)	01 10 00 01 00 02 CRC

数据格式

在读写过程中并不能直接传输数据。位数据需要将各个位从低到高的顺序排列。如读取v0.0-v0.4,返回的数据顺序是: (字节高端)-> v0.4 v0.3 ... v0.1 ->(字节低端)。即是说返回的字节为5,二进制为"0000 0101",那么V0.0和V0.2为1。V0.1、V0.3和V0.4为0。写入数据顺序与此相同。

字数据需要将数据进行交换。如图1所示。读取VW0返回的数据为: byte1 byte2,那么VW0的实际数据为 byte2 byte1。如果VW0表示的为一个word类型数据,那么它的真实值为: byte2+byte1*256,即交换为"byte2 byte1"后,强转为word型的值。VD0类似,需要将VW0和VW1分别交换强转为响应的数据类型。程序中ChangeVar函数可以改变这种字节顺序。这些都是由Modbus协议规定的。

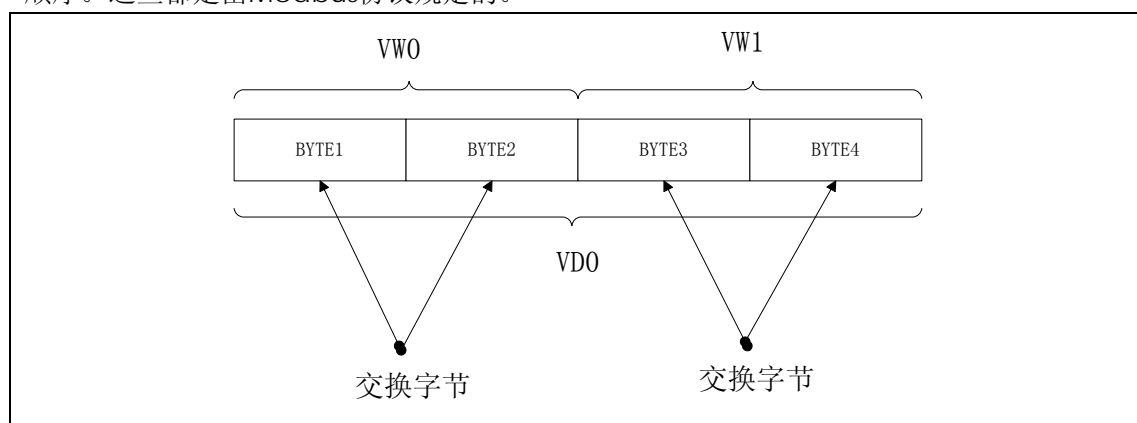


图 1 字节顺序

在 BCB 中使用 Mscomm

本例程使用的编辑器是 Borland C++ Builder6.0。

添加 Activex(mscomm32.ocx)组件

首先,选中菜单中的“组件”,点击“导入 ActiveX 控件(X)”。

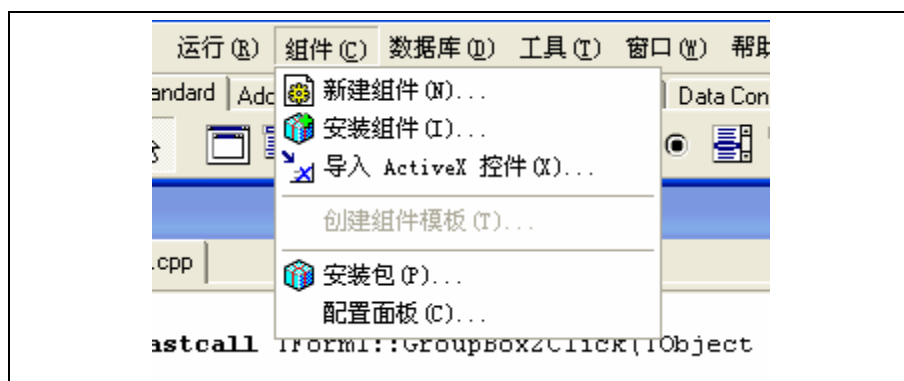


图 1 选择“导入 ActiveX 控件”菜单

然后在弹出对话框中，选中“Microsoft Comm Control 6.0”。如果没有找到该组件，可以通过“添加按钮把组件添加到列表中”。然后点击创建和安装。

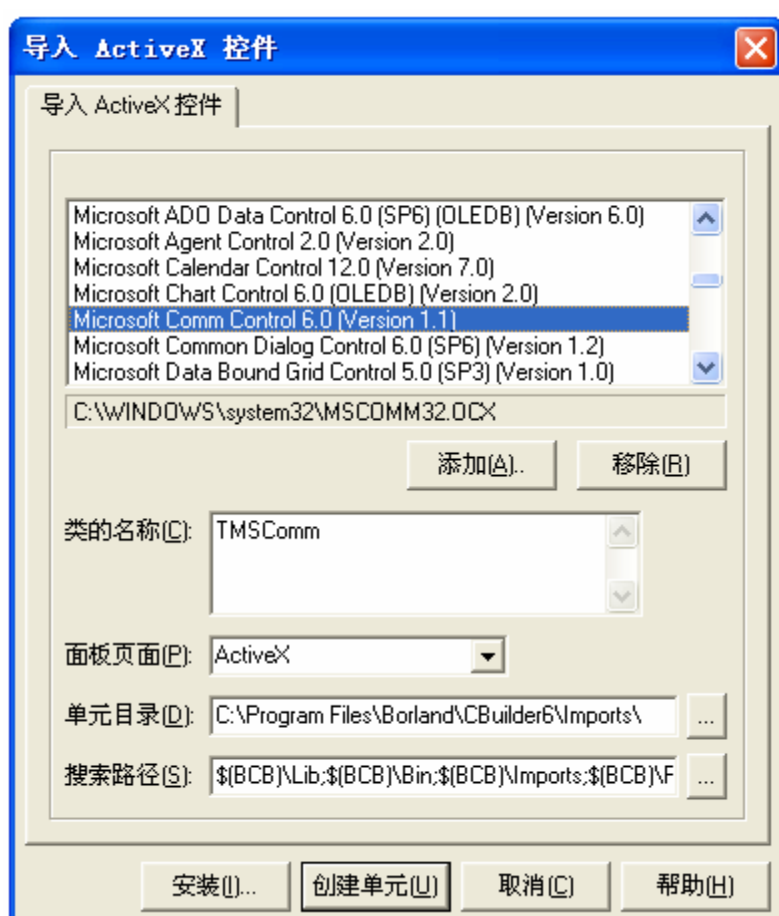


图 2 导入 Mscomm 组件



图 3 错误提示

如果已经安装和创建单元，程序会提示错误如图 3 所示。此时点击确定即可。等安装和创建完毕，点击取消即可回到主界面。



图 4 功能栏

选中工具栏中的“ActiveX”选项。可以看见在下边增加了一个“电话”摸样的图标。

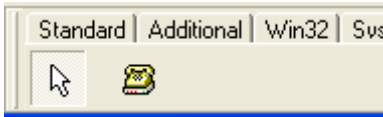


图 5 MSComm 组件图标

此时，我们可以编辑程序界面如图所示。同时增加 3 个定时器和串口通信组件。自此界面的操作完毕。只需要将相应按钮和事件的代码填入程序中即可运行。

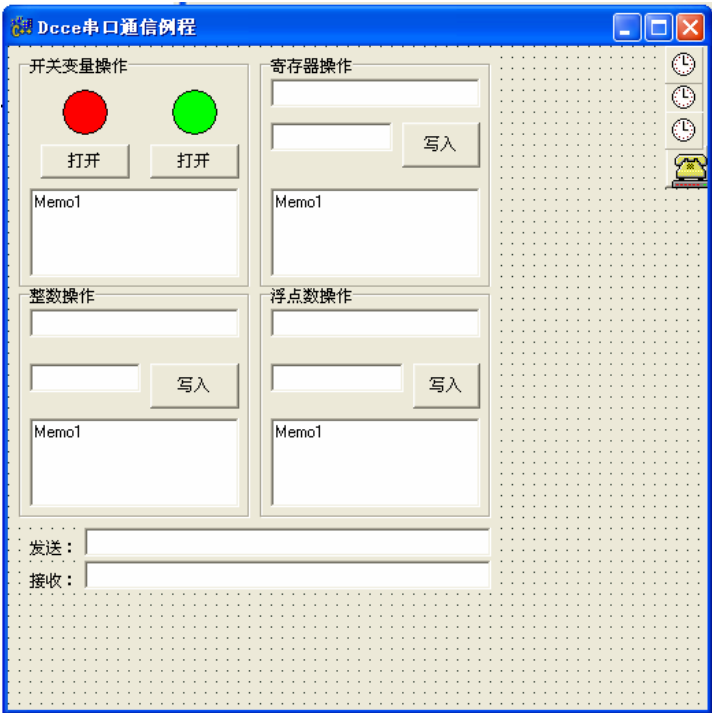


图 6 程序界面

编辑头文件

首先在头文件中填写一些数据结构和成员变量。SendTask 数据结构为通信请求的数据结构,其提供读写 4 数据的初始化函数(这些函数将自动完成数据打包)。的 CRC16 为 CRC 效验计算函数。

```
extern unsigned short CRC16(unsigned char* puchMsg, unsigned char usDataLen);

typedef struct _tagSendTask // 请求数据定义
```

```

{
    UINT    nID;           // 请求 ID 号 用于区别不同的请求
    int  nSendLen;         // 发送数据的长度
    int  nRevLen;          // 接收数据的长度
    BYTE  szSend[270];     // 发送的数据
    BYTE  szRev[270];      // 接收的数据

    _tagSendTask()         // 初始化
    {
        memset(this,0,sizeof(_tagSendTask));
    }
    // 初始化位读请求
    // ID 请求 ID 号, nStation 站地址,nAddr 操作起始地址,nLen 读取位长度
    void InitReadBit( UINT ID, BYTE nStation, UINT nAddr, UINT nLen );
    // 初始化寄存器读请求
    // ID 请求 ID 号, nStation 站地址,nAddr 操作起始地址,nLen 读取寄存器长度
    void InitReadRegister( UINT ID, BYTE nStation, UINT nAddr, UINT nLen);
    // 初始化写位请求
    // ID 请求 ID 号, nStation 站地址,nAddr 操作起始地址,pSend 发送的数据,nLen 读
    取寄存器长度
    void InitWriteBit( UINT ID, BYTE nStation, UINT nAddr, BYTE* pSend ,UINT nLen );
    // 初始化寄存器写操作
    // ID 请求 ID 号, nStation 站地址,nAddr 操作起始地址,pSend 发送的数据,nLen 读
    取寄存器长度
    void InitWriteRegister(UINT ID, BYTE nStation, UINT nAddr, BYTE *pSend,UINT
    nLen );
} SendTask,*PSendTask;

```

位读初始化函数如下所示：

```

// 初始化位读请求
// ID 请求 ID 号, nStation 站地址,nAddr 操作起始地址,nLen 读取位长度
void InitReadBit( UINT ID, BYTE nStation, UINT nAddr, UINT nLen )
{
    nID = ID;
    nSendLen = 8;           //设置发送请求字节长度为 8
    nRevLen = 5 + (nLen+7)/8; //设置接收字节
    szSend[0] = nStation;   //站地址
    szSend[1] = 1;          //命令号
    szSend[2] = (nAddr&0xFF00)>>8; //操作地址高字节
    szSend[3] = nAddr&0xFF;  //操作地址低字节
    szSend[4] = (nLen&0xFF00)>>8; //读取长度高字节
    szSend[5] = nLen&0xFF;   //读取长度低字节
    CRC16((BYTE*)szSend,6); //计算校验位
}

```

```
}
```

此外,还需要在 Form 中添加如下变量和函数。任务队列采用 stl 中的 list 管理。必须在头文件添加如下代码。m_pSend 记录当前操作的任务的指针。没有请求时为 NULL。m_bRedOpen, m_bGreenOpen 表示位变量的状态,其将与设备中对应变量存在一定程度上的延时。TChangeVar 主要简化字和双字的高低字节顺序,这是由 Modbus 协议决定。Modbus 规定传输数据高字在前,而电脑恰恰相反。SendPackage 主要完成发送请求。剩余的函数主要是读取返回处理。

```
#include <list>
using namespace std;
```

```
private: // User declarations
    list<PSendTask> m_SendList;           // 任务队列
    PSendTask      m_pSend;              // 当前发送的请求

    bool           m_bRedOpen;            // 红灯状态
    bool           m_bGreenOpen;          // 绿灯状态

public: // User declarations
    void __fastcall TChangeVar(BYTE* pByte,int nLen); // 交换字节顺序

    void SendPackage(); // 发送数据包
    void OnRedRead(BYTE value ); // 红灯返回处理
    void OnGreenRead(BYTE value ); // 绿灯返回处理
    void OnRegisterRead( BYTE* pV); // 读寄存器
    void OnIntegerRead(BYTE* pV); // 读整数
    void OnFloatRead(BYTE* pV); // 读浮点数
```

添加初始化代码

接下来,双击 Form 空白处进入 FormCreate 函数中。填写如下初始化代码完成初始化。

```
MSComm1->CommPort = 1;           // 设置端口号
    MSComm1->InputMode=1;          // 设置传入数据的格式,0 表示文本形式,1 表示 2 进制方式
    MSComm1->Settings="9600,n,8,1"; // 配置串口
    if ( MSComm1->PortOpen == false )
    {
```



```

        MComm1->PortOpen = true;    // 打开串口
        Timer_Read->Interval = READ_TIME;    // 读取定时器间隔时间

        Timer_Read->Enabled = true;    // 读取定时器启动
        SendPackage();
    }
    Memo1->Text = "说明：打开/关闭红灯即对 v0.0 写 1/0，通过定时器周期读 v0.0 改变灯色，绿灯对应 v0.1";
    Memo2->Text = "说明：将输入框中的数值写入 VW1，通过定时器周期读值，输出在输出框中";
    Memo3->Text = "说明：将输入框中的数值写入 VD4，通过定时器周期读取值，输出在输出框中";
    Memo4->Text = "说明：将输入框中的数值写入 VD2，通过定时器周期读取，输出在输出框中";
    m_bRedOpen = false;
    m_bGreenOpen = false;
    Button_Red->Caption = "打开";
    Button_Green->Caption = "打开";

```

读写操作及返回处理

要通信,首先要知道通信的内容是什么?然后才是怎么通信的问题。通信的数据就是 Modbus 协议内容。更直白点就是对串口设备的读写请求。以读寄存器为例,我们需要操作的地址是 VW1 (绝对地址为 2336)。那么要读取这个地址的命令就是 "?? 03 09 26 00 01 CRC", 写入的命令是 "?? 16 09 26 00 01 02 ** ** CRC"。其中第一个字节"??"表示站地址,第二个字节"03"和"16"表示读写寄存器命令号,第三个和第四个字节"09 26"为寄存器地址的十六进制表示,十进制即为 2336。接着"00 01"表示读取的长度。写命令紧跟着的是写入数据的字节长度和写入的数据。最后读写命令都需要加入 CRC 校验码。

读: 01 03 09 26 00 01 66 5D

写: 01 16 09 26 00 01 02 (00 10) F9 A5 括号中的表示写入的实际数据。

他们的返回为:

读: 01 03 02 ** ** CRC

写: 01 16 09 26 00 00 CRC

到这里,我们的需要发送的命令和接收数据都清楚了。关于数据组包,程序已经提供了对应函数,可直接调用。如果你对 Modbus 协议还不是很清楚,可以参考 Modbus 协议介绍。

现在需要解决的问题是怎么通信 (即读写)。

在 MComm 概述中我们讲过发送请求数据一般步骤是:

```
清空缓冲区( InBufferCount = 0 );
```

```
设置读取的长度 (InputLen = 0);  
设置触发事件的接收长度 (Rthreshold = n);  
设置发送内容 (Output)。
```

对于我们编写人机界面软件而言,需要不断读取 PLC 设备中的数值,来获取最新的数据。我们可以采用定时器。当程序启动或在用户点击开启按钮后,我们打开读取定时器,程序每过一段时间便调用我们编写好的读取函数。在用户点下“写入”,我们将用户的最新输入值发送到 PLC 设备中。

然而,还有一些问题需要引起注意。由于串口通信速度较慢,当我们发送一个请求后,需要等到返回或超时后才能发送下一个请求。而定时器并不关心这些,事件一到,便调用读取函数。这样就极有可能造成通信错误。为解决此问题,可以有俩种方法。一是设置一个全局变量(通信锁),需要通信时,首先检查变量锁是否有其它请求正在通信,若无,则锁定该变量并通信。等待接收数据或接收超时后解锁;否则放弃通信。这样可能造成有时不能写入的问题。另一种方法是维护一个请求队列,当需要读写请求时,将其添加到请求队列中。我们这个例程将使用这一方法。

请求队列的数据单元定义为一个结构体,并在其中增加几个常用 Modbus 协议打包方法。ID 表示操作标志符号,当接收数据后使用;nStation 为设备站地址;nAddr 为操作起始地址;nLen 表示操作的数据长度;pSend 表示发送的数据。每次请求时调用这些方法完成打包,并添加到队列末尾。当超时定时器发生、队列为空定时器发生或接收到数据时,再发送下一个请求。

```
typedef struct _tagSendTask  
{  
    UINT    nID;           // 任务 ID 号  
    int     nSendLen;      // 发送数据的长度  
    int     nRevLen;       // 接收数据的长度  
    char    szSend[270];   // 发送的数据  
    char    szRev[270];    // 接收的数据  
}  
  
void InitReadBit( UINT ID, BYTE nStation, UINT nAddr, UINT nLen )  
void InitReadRegister( UINT ID, BYTE nStation, UINT nAddr, UINT nLen)  
void InitWriteBit( UINT ID, BYTE nStation, UINT nAddr, BYTE* pSend ,UINT nLen )  
void InitWriteRegister(UINT ID, BYTE nStation, UINT nAddr, BYTE *pSend,UINT  
nLen )
```

1. 打包。

在读取请求填写一下类似代码。如下面的代码是添加读取V0.0的请求。

```
PSendTask pSend;  
pSend = new SendTask;  
pSend->InitReadBit( READ_RED_LED,MOD_ADDR,16768,1); //初始化读红灯位请求,读V0.0  
m_SendList.push_back(pSend);
```

2. 发送。

在SendPackage函数中添加发送代码。发送主要是从队列头取出发送请求，然后将发送数据转化成OleVariant发送到串口。

```
if ( m_SendList.empty())
{
    Timer_Send->Interval = SEND_TIME;
    Timer_Send->Enabled = true;
    return;          // 队列空，直接返回
}
m_pSend = (PSendTask)m_SendList.front();    // 取队列头,保存指针
m_SendList.pop_front();                    // 从队列头删除
MSComm1->InBufferCount = 0;               // 清空缓冲区
MSComm1->InputLen = 0;                     // 设置读取缓冲区的长度 0 表示读取
缓冲区全部数据
MSComm1->RThreshold=m_pSend->nRevLen;// 设置触发事件接收数
据的长度
int Bounds[2] = {0,m_pSend->nSendLen-1}; // 设置variant的上下限
OleVariant    _var;
_var = VarArrayCreate(Bounds,1, varByte); // 开辟空间
AnsiString strSend = "";
AnsiString strTemp = "";
for (int i = 0; i < m_pSend->nSendLen; ++i)
{
    _var.PutElement(m_pSend->szSend[i],i); // 赋值
    strTemp = Format("%0.2X ",&TVarRec(m_pSend->szSend[i]),0);
    strSend += strTemp;
}
Edit_Send->Text = strSend;
MSComm1->Output= _var;                // 发送数据
Timer_TimeOut->Interval = TIME_OUT;  // 开启超时定时器
Timer_TimeOut->Enabled = true;        // 启动超时定时器
```

3. 接收。

双击MSComm1组件事件处理函数，进入到处理函数处，然后添加以下代码。

```
OleVariant in;                // 返回数据
unsigned char  *pTemp;         // 以数组形式保存
AnsiString strRev = "";
AnsiString strTemp = "";
int nLen;
if ( m_pSend == NULL )        // 无上次发送请求 ， 返回。
                                // 一般不应进入该过程。
{
                                // 有可能为通信效果不佳，修改超时时间
```

```

// 这种情况应检查连接是否有其他通信数据
return; // 可能存在其他通信数据进入该过程
}
nLen = MSComm1->InBufferCount; // 接受数据长度
if ( MSComm1->CommEvent == 2 && nLen == m_pSend->nRevLen ) //
返回数据事件，且接受的数据长度和预定的字节长度相同
{
    in = MSComm1->Input;
    pTemp = new unsigned char[nLen]; // 根据返回长度开辟空间
    for ( int i = 0; i < nLen; i++ )
    {
        pTemp[i] = in.GetElement(i); // 取值
        strTemp = Format("%0.2X ", &TVarRec(pTemp[i]), 0);
        strRev += strTemp;
    }
    Edit_Rev->Text = strRev;
    switch(m_pSend->nID) // 注：此处可以根据设置ID进行区分命令，
也可以根据返回数据的命令号和地址区分
    {
        case READ_RED_LED:
            OnRedRead(pTemp[3]&0x01);
            break;
        case READ_GREEN_LED:
            OnGreenRead(pTemp[3]&0x01);
            break;
        case READ_REGISTER:
            OnRegisterRead(&pTemp[3]);
            break;
        case READ_INTERGER:
            OnIntegerRead(&pTemp[3]);
            break;
        case READ_FLOAT:
            OnFloatRead(&pTemp[3]);
            break;
        case WRITE:
            break;
        default:
            assert(false);
    }
    delete[] pTemp; // 释放空间
    SendPackage();
}

```

然后在OnRedRead等函数中将读取回来的数据显示到界面中。包含整数，浮点数等。

到此，通信程序便可以和设备通信了。

Dcce串口通信例程

开关变量操作

关闭

打开

说明：打开/关闭红灯即对v0.0写1/0，通过定时器周期读v0.0改变灯色，绿灯对应v0.1

寄存器操作

100

100

写入

说明：将输入框中的数值写入Vw1，通过定时器周期读取，输出在输出框中

整数操作

1314

1314

写入

说明：将输入框中的数值写入VD2，通过定时器周期读取，输出在输出框中

浮点数操作

518

518

写入

说明：将输入框中的数值写入VD4，通过定时器周期读取，输出在输出框中

发送：

01 01 41 80 00 01 E8 1E

接收：

01 03 04 80 00 44 01 21 33