|  |  |  |
| --- | --- | --- |
| 作者： | 曹志强 | [Logo_Beckhoff_Red](http://www.beckhoff.com.cn/)上海市江场三路市北工业园区  163号5楼（200436）  TEL: 021-66312666  FAX: 021-66315696 |
| 日期： |  |
| 版本： | V1.0 |
| E\_mail: | Alvin\_ensi@163.com |

|  |
| --- |
| **使用Microsoft Visual C# 实现与TWINCAT PLC control变量的通讯** |

# 

|  |
| --- |
| 概 述 |

|  |
| --- |
| 使用Microsoft Visual C# 软件，实现通过对TWINCAT PLC control中的变量（bool int string类型）进行读取和修改 |

|  |
| --- |
| 文档中包含的文件 |

文件名称 文件说明

|  |  |
| --- | --- |
| PLC源文件 | PLC control程序源文件 |
| TwinCAT.Ads.dll | C#中需要导入的库文件 |
| C#源文件 | C#程序源文件 |

|  |
| --- |
| 备 注 |

|  |
| --- |
| 关键字：通过高级语言对PLC程序的读和写 |

|  |
| --- |
| 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，也欢迎您提出改进建议。  *文档内容可能随时更新*  *如有改动，恕不事先通知* |

**目录**

[**通过c#程序读取和赋值PLC程序中的1个布尔量 3**](#_Toc332019831)

[**通过c#程序读取和赋值PLC程序中的1个整形变量 3**](#_Toc332019832)

[**通过c#程序读取和赋值PLC程序中的1个双精度浮点数 3**](#_Toc332019833)

[**通过c#程序读取和赋值PLC程序中的1个字符串 3**](#_Toc332019834)

[**通过c#程序读取和赋值PLC程序中的1个结构体 3**](#_Toc332019835)

[**通过c#程序读取和赋值PLC程序中的1个数组 3**](#_Toc332019836)

[**通过定时方式来控制PLC程序中变量的读入 3**](#_Toc332019837)

[**通过通知方式来控制PLC程序中变量的读入 3**](#_Toc332019838)

# 通过c#程序读取和赋值PLC程序中的1个布尔量

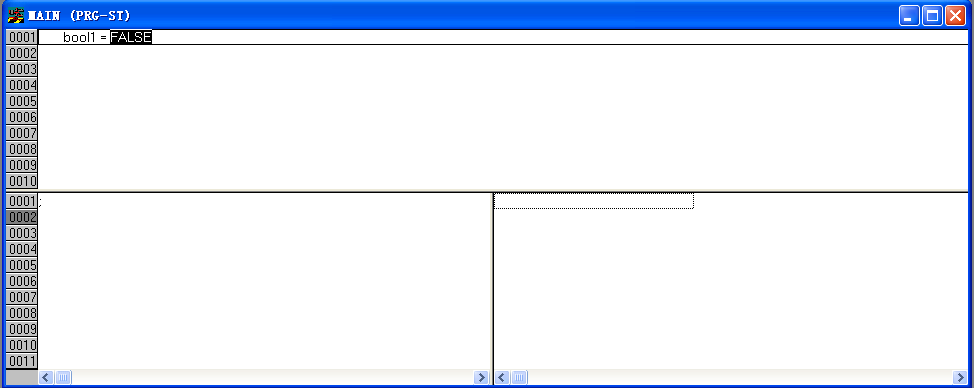
1. 打开PLC CONTROL

在变量定义区定义一个名称为Bool1的bool变量



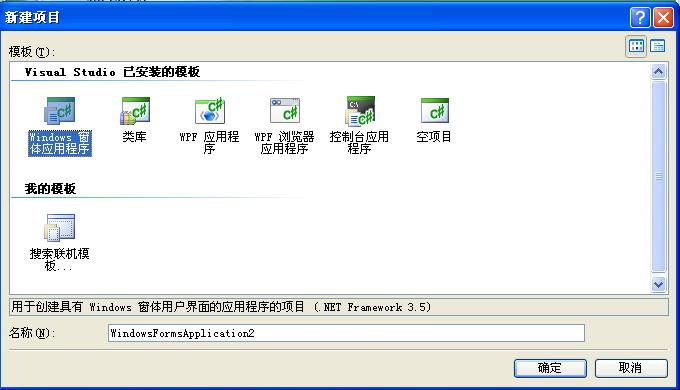
2.运行 PLC程序

效果如下：



3..打开Microsoft Visual C# 2008 Express Edition

新建Windows窗体应用程序

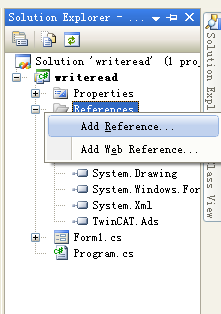


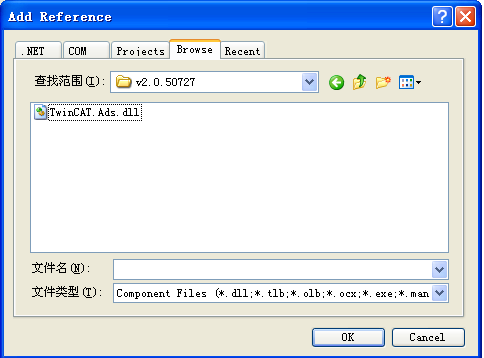
4. .在资源管理器中引用twincat.ads

这个库文件是BECKHOFF公司已经做好的用于通讯连接的库文件

C#中的程序通过该库中提供的方法对PLC进行连接和修改

在资源管理器中导入库文件，点击引用，如图1

图1

图2

其具体路径为“安装盘:\TwinCAT\ADS Api\.NET\v2.0.50727”；对于ce系统其路径为“安装盘:\TwinCAT\ADS Api\CompactFramework\v2.0”。引用完毕后如图一所示出现“TwinCAT.Ads”

如图2

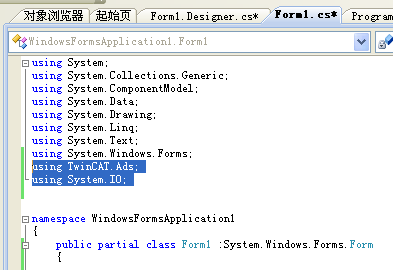
此文件在本文档的附件中也可以找到

5. 输入

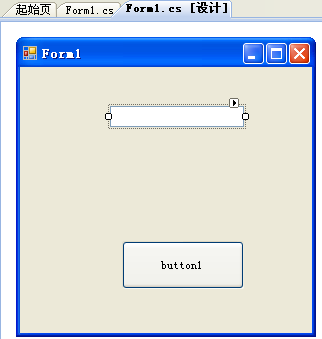
using TwinCAT.Ads;

using System.IO;

实现对导入库文件的调用



6.在winform界面设计上画一个textbox和一个botton



10.在c#编程区声明2个变量

public partial class Form1 : Form

{

private int hbool1;

private TcAdsClient adsClient;

11.双击设计的程序界面

输入：

private void Form1\_Load(object sender, System.EventArgs e)

{

adsClient = new TcAdsClient();

try

{

adsClient.Connect(801);

hbool1 = adsClient.CreateVariableHandle("MAIN.bool1");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

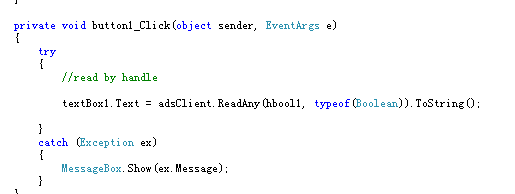
}

}

该段程序作用是新建一个TcAdsClient的实例 adsclient并调用该connect方法与801（run-time1）进行连接。并使用句柄hbool1接受PLC程序中的”BOOL1

12.将BUTTON1的控件的TEXT属性改为“read” 并双击设计的程序界面中的botton 控件在弹出程序段中

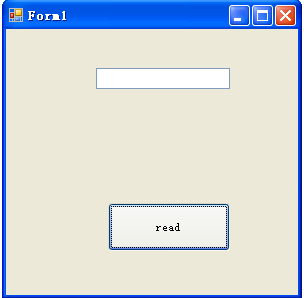
输入：



该句的作用是读入指定句柄对应变量，并将其转化后显示在TEXTBOX控件上

13.点击运行工具栏中的运行

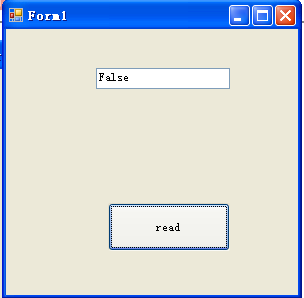




14.切到plc control中观察当前bool1的值



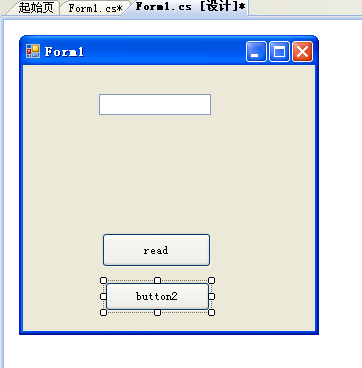
15..切回C#程序点击read



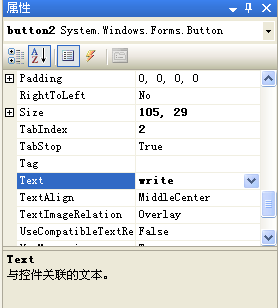
Text中显示的bool状态应该与Plc程序中的bool1相同.

关闭程序，返回C#编程环境

16.在winform程序界面设计中添加一个按钮

****

17将这个button2的text属性改成write



18.完成后双击write按钮 切换到编程界面，添加如下语句：

该句的作用是读入指定句柄对应变量，并将其转化后显示在TEXTBOX控件上

private void button2\_Click(object sender, EventArgs e)

{

try

{

adsClient.WriteAny(hbool1, Boolean.Parse(textBox1.Text));

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

19.完成后，点击运行。

同时将PLC程序也切换到RUN状态下





可以看到当前的PLC状态时TRUE

这个时候我们在C#程序中的框格内输入FALSE

点击Write看一下效果



切换到PLC程序后，我们发现PLC程序中的BOOL变量的值由TRUE转换成FALSE了。

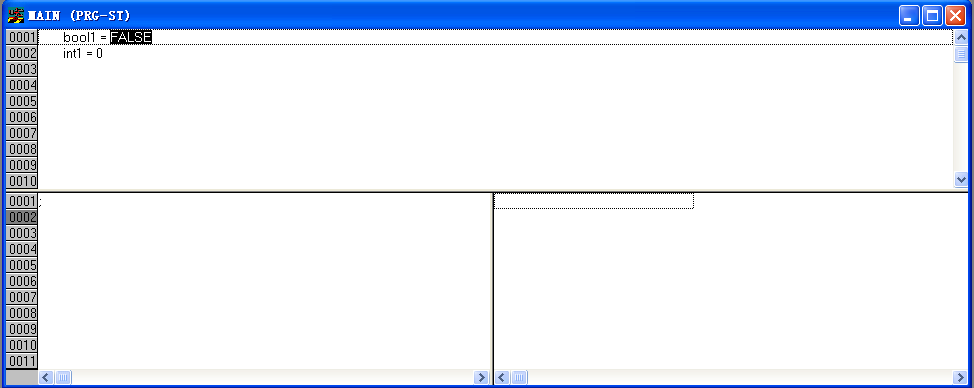
# 通过c#程序读取和赋值PLC程序中的1个整形变量

1. 在PLC程序中定义1个int类型的变量int1

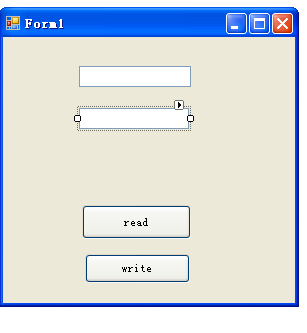


运行PLC程序

效果如下：



3.在C# 程序界面设计中添加1个textbox



4.在c#程序编写中添加1个句柄 hint1



5. 并使用句柄hint1接受PLC程序中的”int1



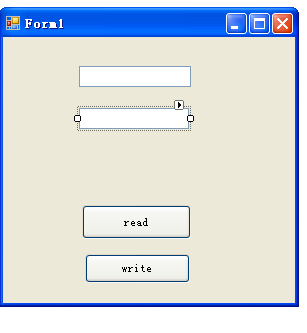
6. 该句的作用是读入指定句柄对应变量，并将其转化后显示在TEXTBOX控件上



7.点击运行

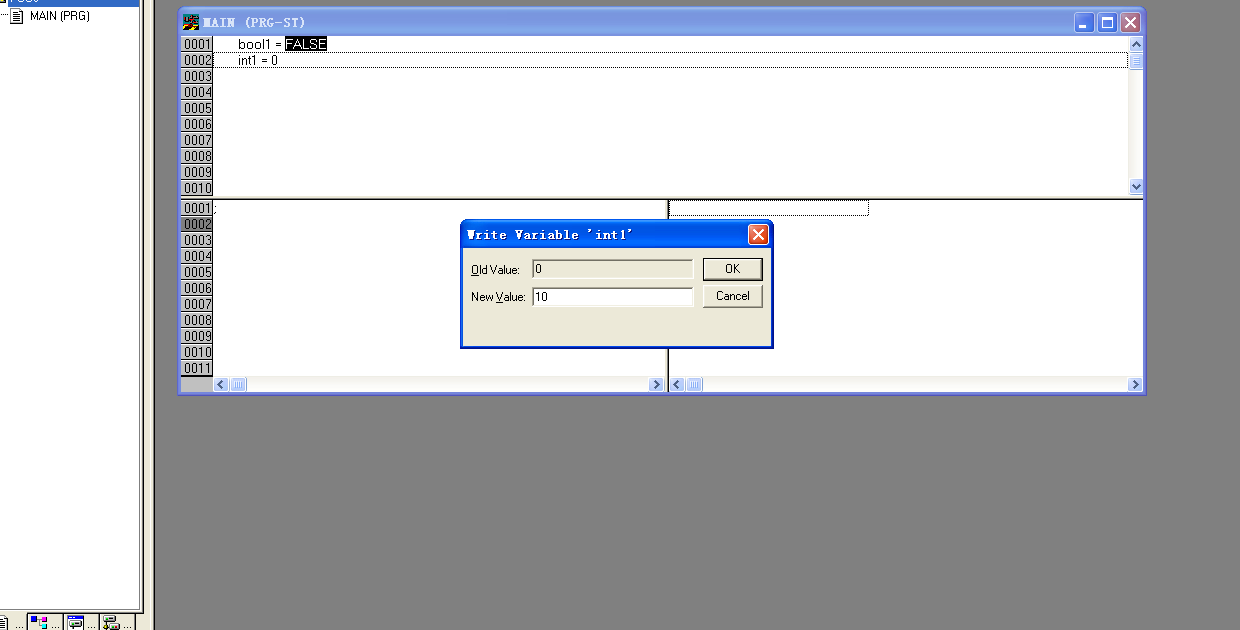


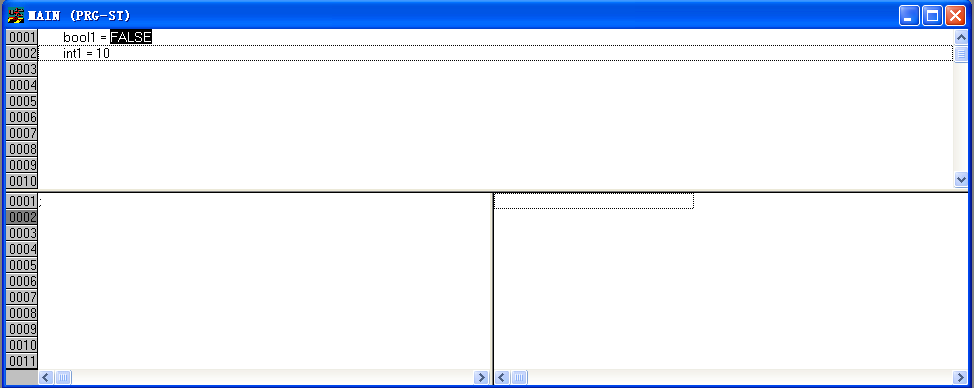
效果如下：



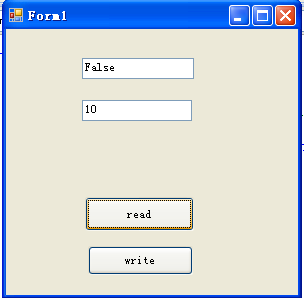
8.切回PLC CONTROL

对int1进行赋值





9.切回C#程序，点击READ



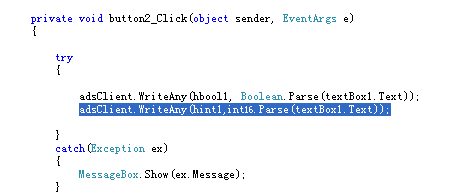
Text2中的数值应该和我们在PLC control中设置的值一样

10.成功后关闭程序，在C#程序

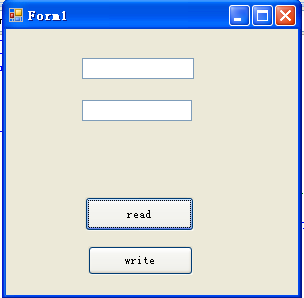
界面编写中添加一个切换到C#编程模式下，添加语句：

通过句柄将值赋入到plc程序里的in1中

adsClient.WriteAny(hint1, int.Parse(textBox2.Text));

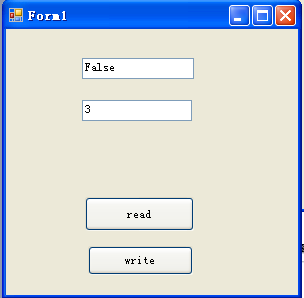


11.完成后点击运行。16





12.这时候PLC中INT变量的值是10，我们通过C#程序进行修改

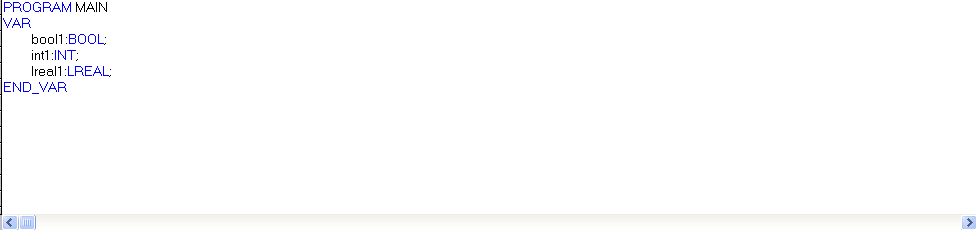


点击WRITE后，可以看到PLC程序中INT类型的变量值发成了改变，值变为C#程序中设定的值



# 通过c#程序读取和赋值PLC程序中的1个双精度浮点数

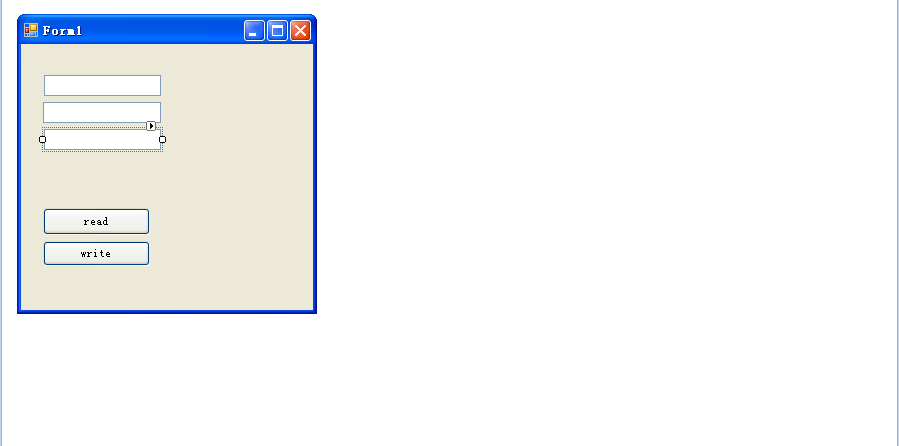
1. 在PLC CONTROL中添加一个LREAL类型的变量LREAL1



运行PLC程序，效果如下：



2．在C#程序界面中添加一个TEXTBOX控件，效果如下：

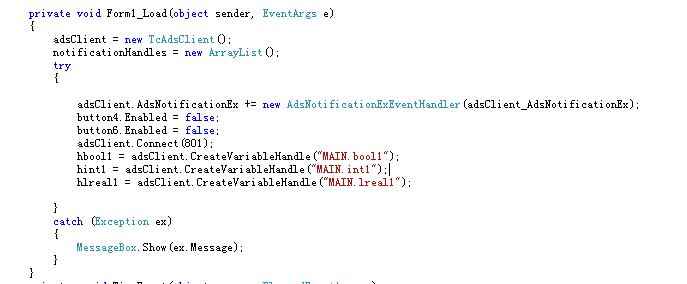


3，在C#程序编写中定义一个句柄变量：

private int hlreal1;

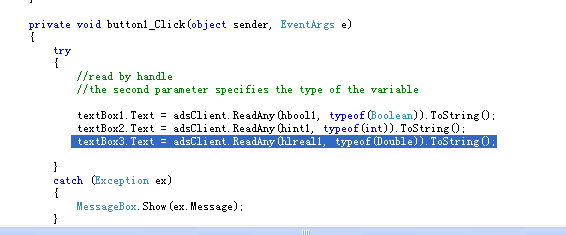
并使用句柄hreal1接受PLC程序中的”lreal1

hlreal1 = adsClient.CreateVariableHandle("MAIN.lreal1");

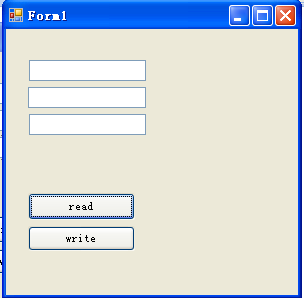


4．该句的作用是读入指定句柄对应变量，并将其转化后显示在TEXTBOX控件上

textBox3.Text = adsClient.ReadAny(hlreal1, typeof(Double)).ToString();

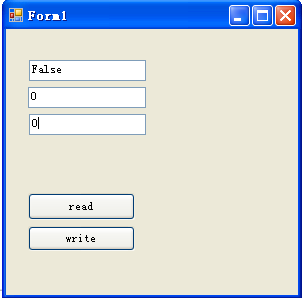


1. 点击运行，效果如下：





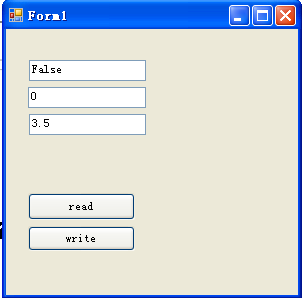
这时候点击C#程序中的read按键，显示如下：



修改PLC程序中LREAL类型的值：



再次按下C#程序中的READ按键

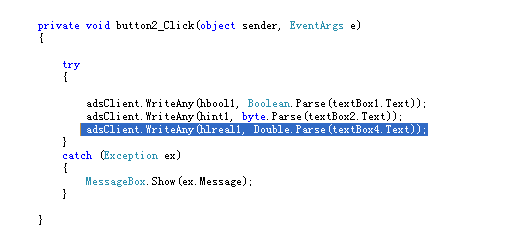


可以看到对应的变量值已经发现改变

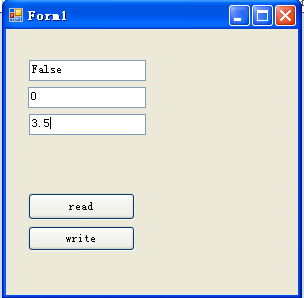
下面我们将对LREAL类型的变量值进行输入

1. 在C#程序中添加如下语句对LREAL类型变量赋值的语句

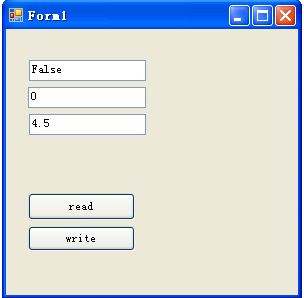
adsClient.WriteAny(hlreal1, Double.Parse(textBox4.Text));

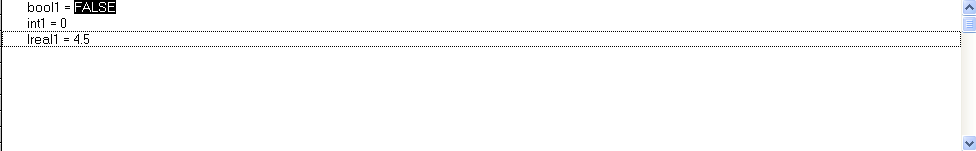


1. 完成之后点击运行。效果如下：



在C#程序中。修改LREAL类型的值：





可以看到PLC程序中LREAL类型变量的值已经被修改

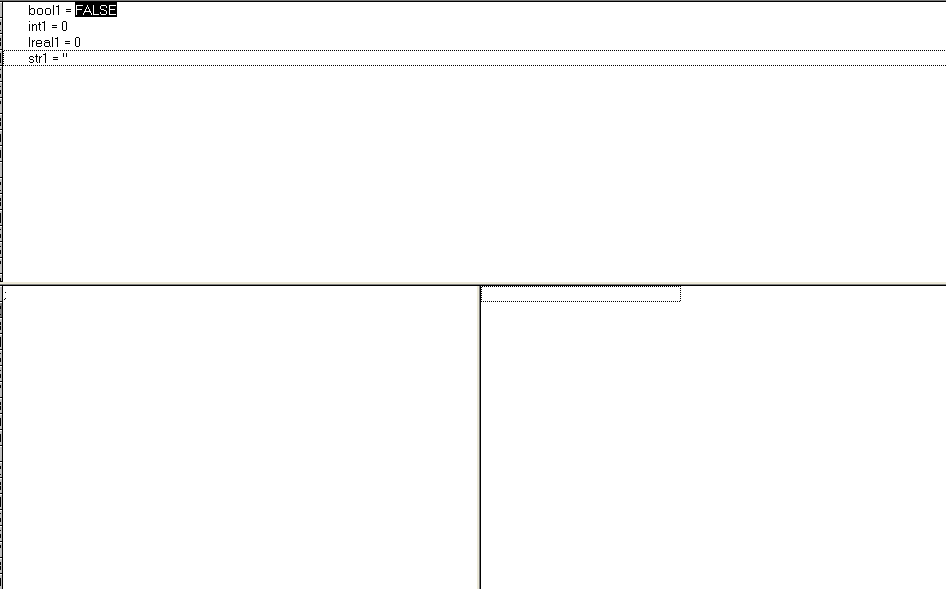
# 通过c#程序读赋值PLC程序中的1个字符串

1. 在Plc control 中定义一个string 类型的变量 str1



2.运行PLC程序

效果如下：



1. 在窗体界面设计中添加一个textbox控件



4.在c#程序编写中添加1个句柄 hstr1



5.使用句柄对PLC程序中的hstr1进行通讯



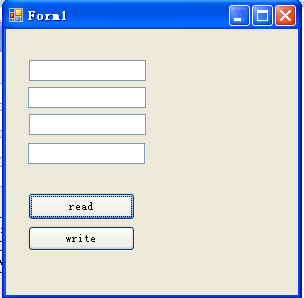
6. 该句的作用是读入指定句柄对应变量，并将其转化后显示在TEXTBOX控件上

textBox3.Text = adsClient.ReadAny(hstr1, typeof(String), new int[] { 5 }).ToString();

7..点击运行

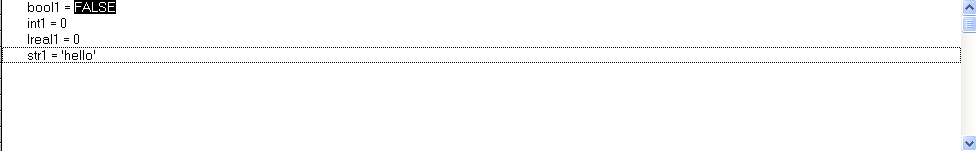


.效果如下：

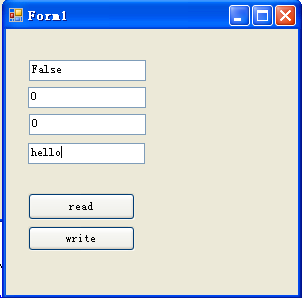


8.在PLC程序

对str1进行赋值

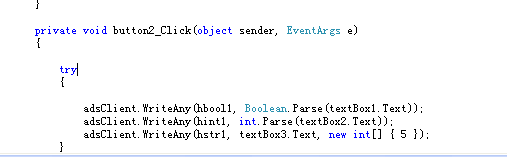


9.切到C#程序中 按read查看效果



10.成功之后，切换到C#的编程界面，输入如下代码：

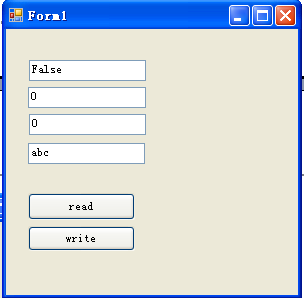
adsClient.WriteAny(hstr1, textBox3.Text, new int[] { 5 });



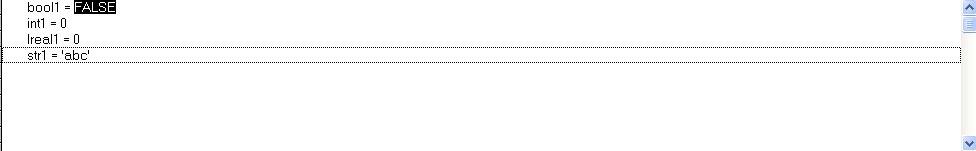
该语句的作用是使用句柄将C#程序中textbox3中的字符串输入到PLC程序中

完成之后点击运行，

11.我们在C#程序中修改TEXTBOX3中的字符，看一下效果：



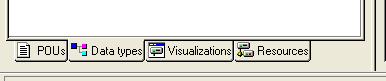
点击write后，可以看到PLC程序中STRING的变量值变成了我们修改的值：

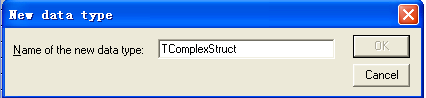


# 通过c#程序读取和赋值PLC程序中的1个结构体

1. 在PLC CONTROL中切换到Data types选项卡，新建一个结构体

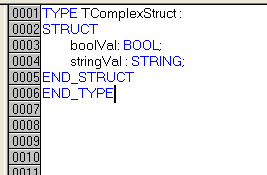
命名TcomplexStruct，如图所示：



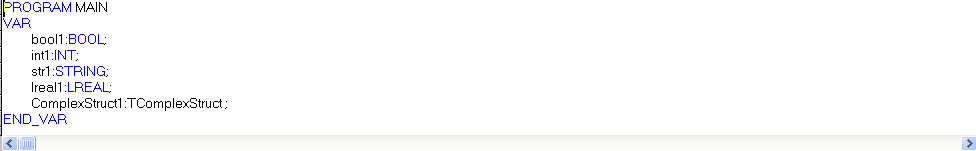


1. 在这个结构体中，赋予2个元素，分别是BOOL类型的BOOLVAL和STRING类型的

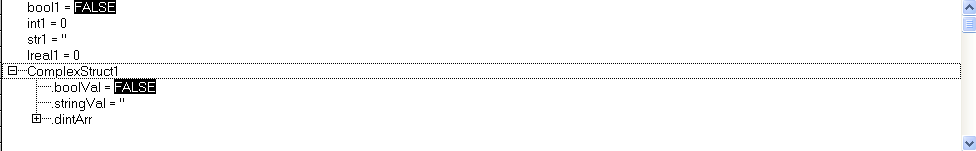
STRINGVAL



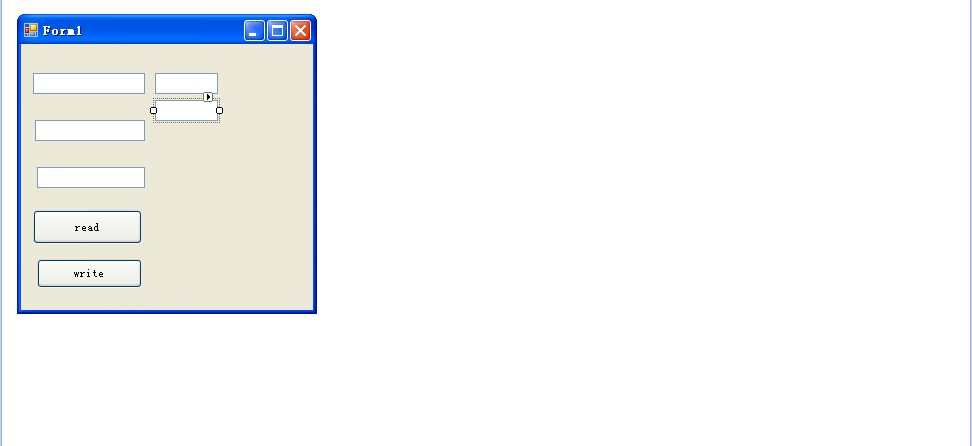
3，在PLC的MAIN函数中引用这个结构体：



4，LOG-IN,运行PLC 程序：



5．切换到C#编程，在界面绘制中添加2个TEXTBOX，如图所示：



1. 在C#程序编写中添加添加一个引用，并一个结构体类型的类:

[StructLayout(LayoutKind.Sequential, Pack = 1)] //分布结构体内内存地址的位置

public class ComplexStruct

{

[MarshalAs(UnmanagedType.I1)]

public bool boolVal;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 6)]

public string stringVal = "";

[MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]

public int[] dintArr = new int[4];

}

7. 添加一个结构体元素显示在C#程序界面中的一个方法：

private void FillStructControls(ComplexStruct structure)

{

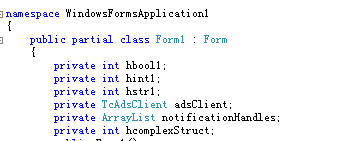
textBox4.Text= structure.boolVal.ToString();

textBox5.Text = structure.stringVal;

}

8．定义一个接受结构体变量的句柄

private int hcomplexStruct;



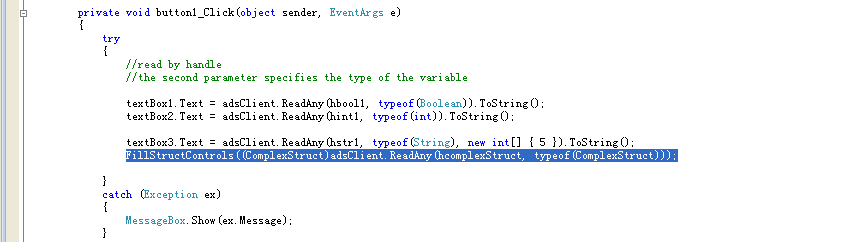
9.在Form1\_Load方法中对该句柄赋值：

hcomplexStruct = adsClient.CreateVariableHandle("MAIN.ComplexStruct1");

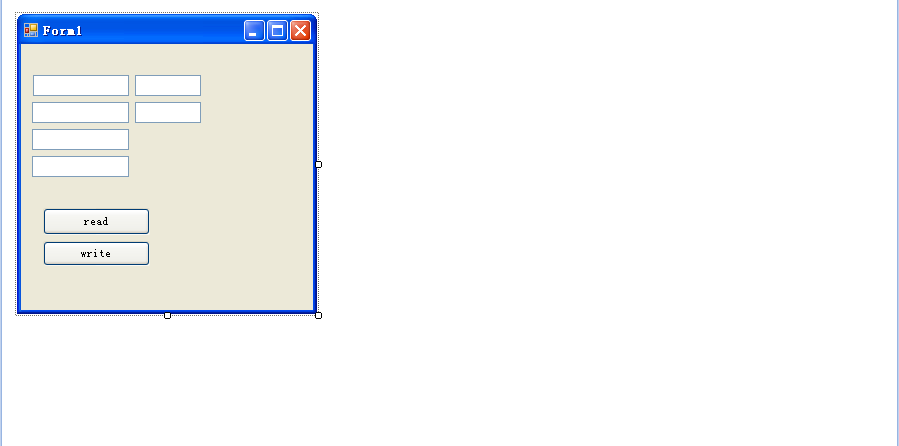


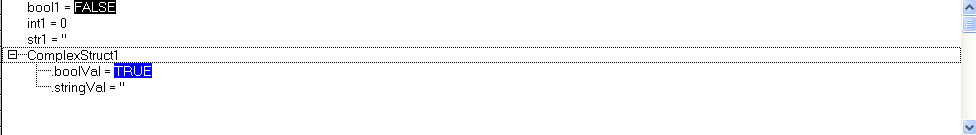
10. 编写从PLC中读取结构体元素值的C#语句：

FillStructControls((ComplexStruct)adsClient.ReadAny(hcomplexStruct, typeof(ComplexStruct)));



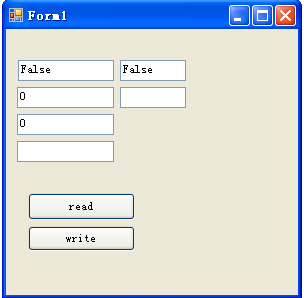
11.点击运行C#程序，效果如下：



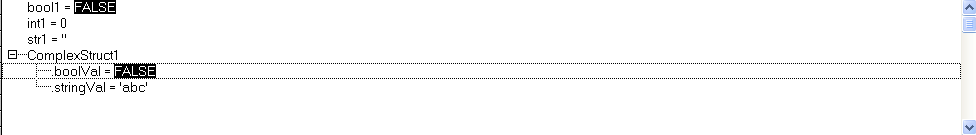


这时候点击READ按钮，

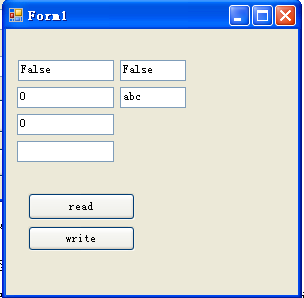
可以看到如下的效果：



12.这时候我们可以再PLC 程序中改变结构体中变量的值，



在C#程序中重新点击read,可以看到如下效果：



以上就是结构体的写入部分。

接着要介绍的通过C#程序实现PLC程序中结构体的写入。

13.在C#程序中编写一个将读入的结构体元素写到C#程序界面上的方法：

private ComplexStruct GetStructFromControls()

{

ComplexStruct structure = new ComplexStruct();

structure.boolVal = Boolean.Parse(textBox4.Text);

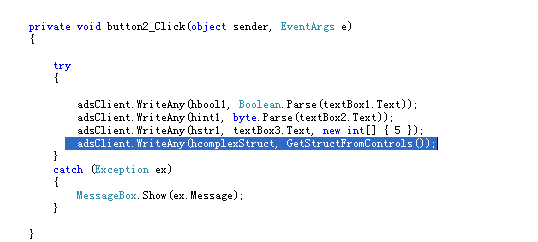
structure.stringVal = textBox5.Text;

return structure;

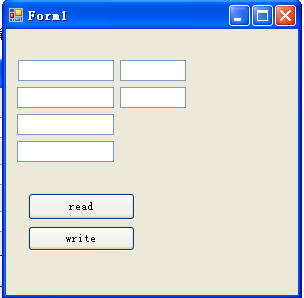
}

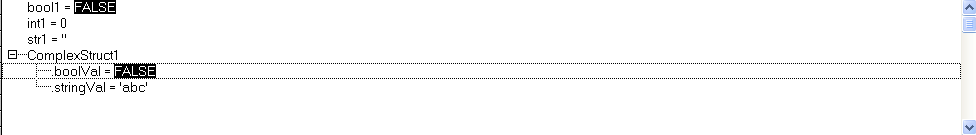
14.编写写入PLC中结构体元素值的C#语句：

adsClient.WriteAny(hcomplexStruct, GetStructFromControls());

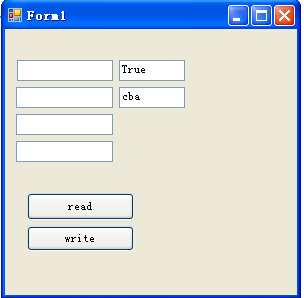


15.完成之后，点击运行：

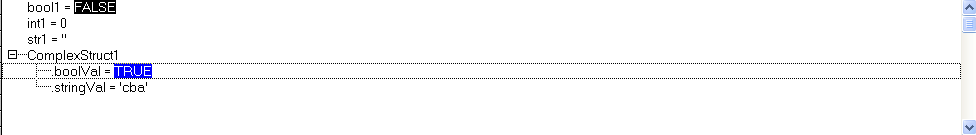




16.在C#程序中输入要赋予结构体元素的值，点击WRITE:

\

可以看到PLC程序中结构体的相应元素也发生了变化：

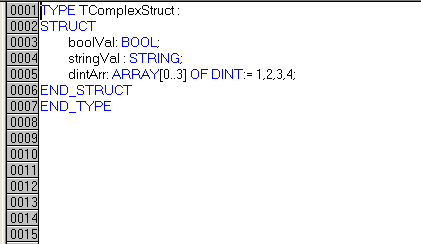


结构体的值被成功的修改了。

# 通过c#程序读取和赋值PLC程序中的1个数组

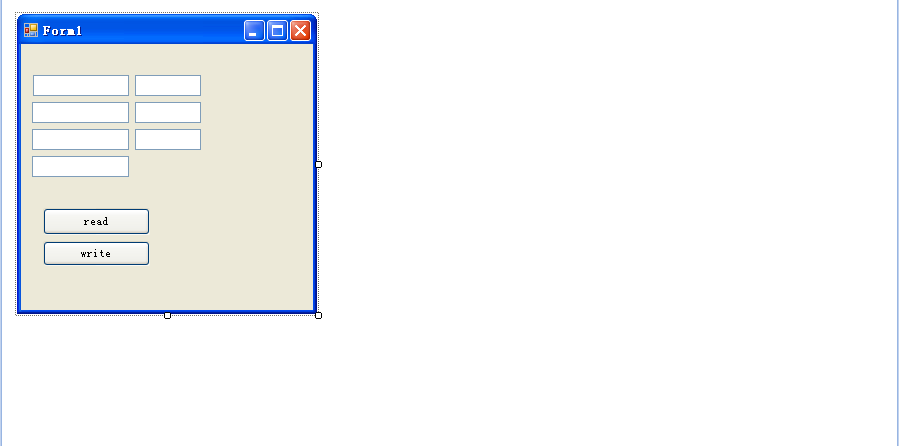
1.在PLC程序中添加1个有4个元素的数组变量作为结构体的元素，并赋予初值。

如下图所示：



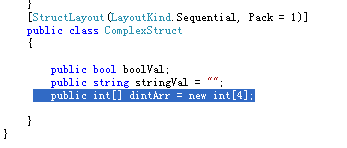
2．在C#程序界面绘制中，添加一个TEXTBOX控件，

如图所示：



3.切换到C#编程界面，在先前所定义的结构体中添加一个数组变量：

public int[] dintArr = new int[4];

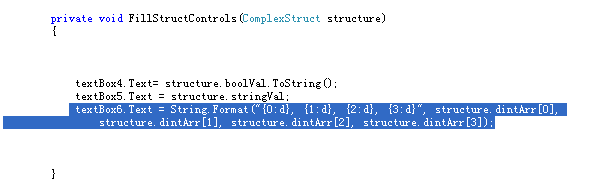


.

1. 编写从PLC中读取结构体数组值的C#语句：

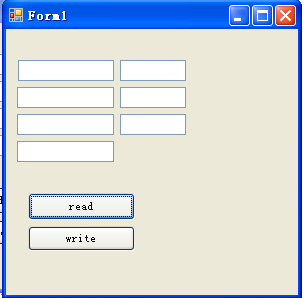
textBox6.Text = String.Format("{0:d}, {1:d}, {2:d}, {3:d}", structure.dintArr[0],

structure.dintArr[1], structure.dintArr[2], structure.dintArr[3]);

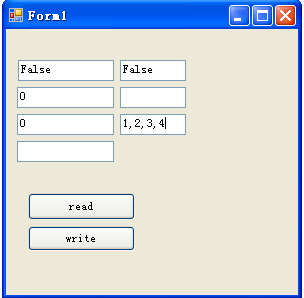


1. 点击运行程序，

效果如下：



点击READ，可以看到如下效果：



说明数组已经成功读入。

接下来编写写入数组的语句：

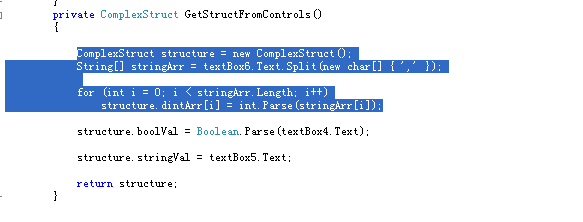
1. 在C#程序中编写一个将读入的数组元素写到C#程序界面上的方法：

ComplexStruct structure = new ComplexStruct();

String[] stringArr = textBox6.Text.Split(new char[] { ',' });

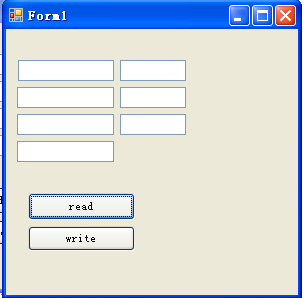
for (int i = 0; i < stringArr.Length; i++)

structure.dintArr[i] = int.Parse(stringArr[i]);



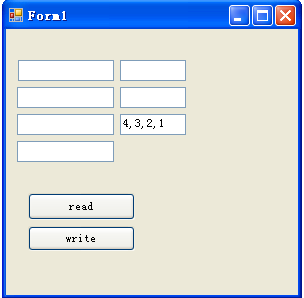
1. 完成之后，点击运行，

效果如下：





在C#程序中输入所需要数组的值：



点击WRITE

可以看到如下效果：



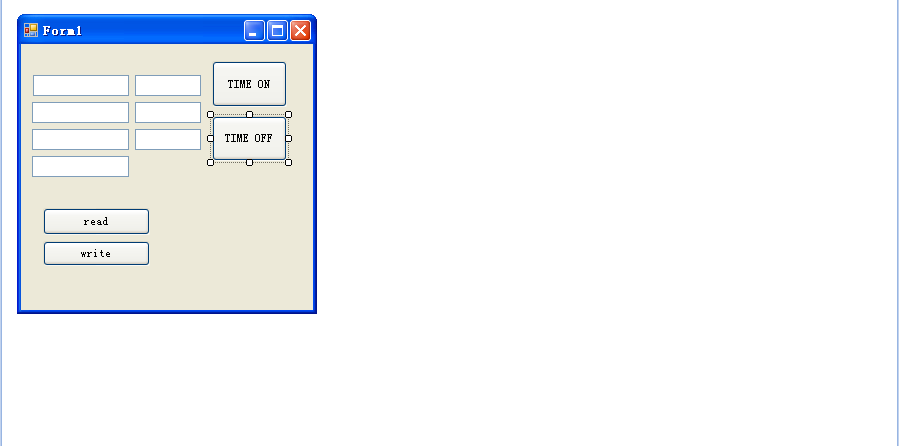
数组的值被成功的修改。

# 通过定时方式来控制PLC程序中变量的读入

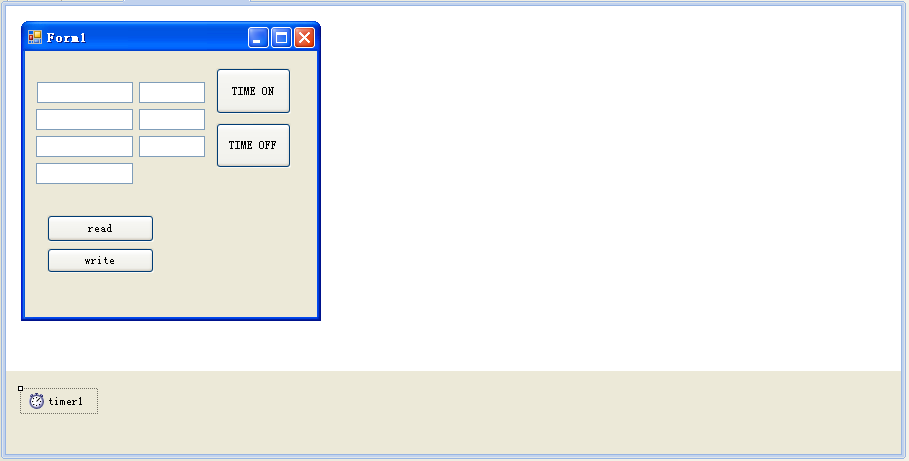
定时方式就是我们设定一个接受的间隔时间，程序每过一段时间自动的进行一次数据的更新。

在接受数据的过程中，我们碰到这种情况，需要数据实时的变化，而不是每按一次READ改变一下，这种时候我们就需要使用定时器功能块来完成。

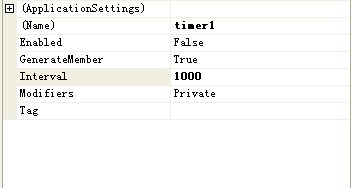
1. 在C#程序界面设计中，添加2个按键



2.添加一个定时器控件：



3.设置定时器的默认状态和定时时间：



4.在程序语句编写中添加对定时功能能按键的设置的语句

private void button3\_Click(object sender, EventArgs e)

{

timer1.Enabled = true;

button3.Enabled = false;

button4.Enabled = true;

}

private void button4\_Click(object sender, EventArgs e)

{

timer1.Enabled = false;

button3.Enabled = true;

button4.Enabled = false;

}

5.设置定时器的功能：

private void timer1\_Tick(object sender, EventArgs e)

{

textBox1.Text = adsClient.ReadAny(hbool1, typeof(Boolean)).ToString();

textBox2.Text = adsClient.ReadAny(hint1, typeof(int)).ToString();

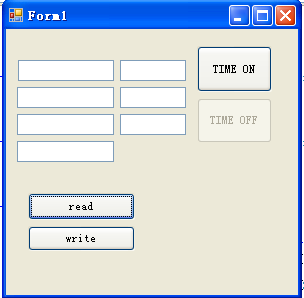
textBox4.Text = adsClient.ReadAny(hlreal1, typeof(Double)).ToString();

textBox3.Text = adsClient.ReadAny(hstr1, typeof(String), new int[] { 5 }).ToString();

FillStructControls((ComplexStruct)adsClient.ReadAny(hcomplexStruct, typeof(ComplexStruct)));

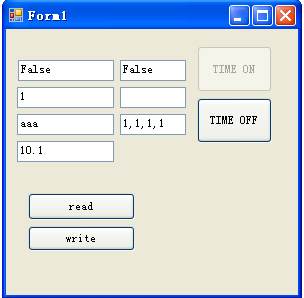
}

6．完成之后点击运行：





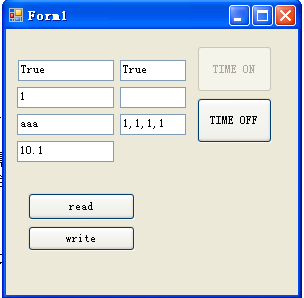
1. 点击TIME ON



修改PLC程序中变量的值。



切回C#程序中观察：



可以发现在下一个周期内C#程序已经更新了我们所需要的数据

# 通过通知方式来控制PLC程序中变量的读入

--------通知方式数据

在有些情况下我们需要使用到定时方式，但是定时方式在不改变的情况下会产生空的数据交换，浪费了数据的流量。这个时候我们需要另一种控制方式：通知方式。

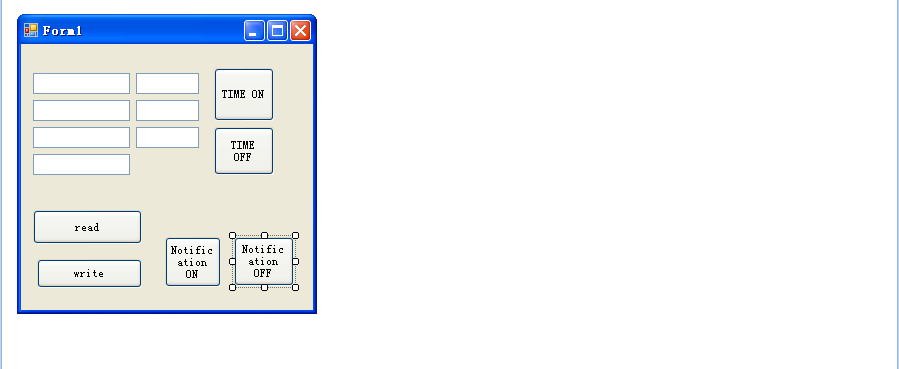
通知方式（Notification）

ADS 客户端向ADS 服务器发送ADS 请求， 当客户端接受到新的信息时改变客户端的值。

1.在C#程序界面中，添加2个BOTTON控件

在2个控件的TEXT属性中分别命名Notification ON和Notification OFF

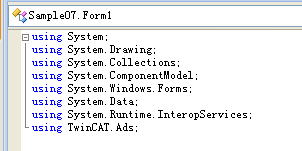
如图所示：



完成之后，切换到编程界面

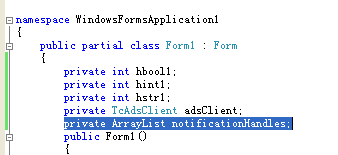
2.调用System.Collections库

using System.Collections;



3.使用该库新建一个方法

private ArrayList notificationHandles;



4.创建一个adsClient\_AdsNotificationEx方法用来处理变量参数

private void adsClient\_AdsNotificationEx(object sender, AdsNotificationExEventArgs e)

{

TextBox textBox = (TextBox)e.UserData;

Type type = e.Value.GetType();

if (type == typeof(string) || type.IsPrimitive)

textBox.Text = e.Value.ToString();

else if (type == typeof(ComplexStruct))

FillStructControls((ComplexStruct)e.Value);

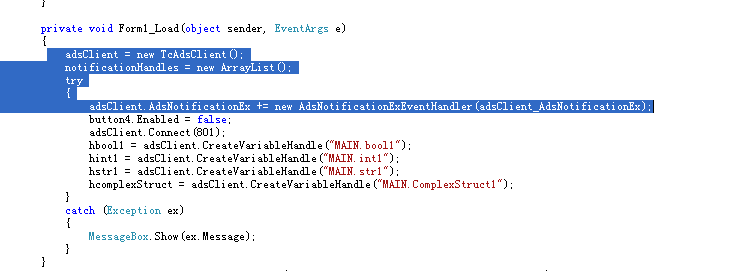
}

5.在FORM\_LOAD中注册事件处理的函数

adsClient.AdsNotificationEx += new AdsNotificationExEventHandler(adsClient\_AdsNotificationEx);

button4.Enabled = false;

并在程序启动时将BUTTON4的值默认调为FALSE



6.在通知中将变量注册到ADS客户端

private void button3\_Click(object sender, EventArgs e)

{

notificationHandles.Clear();

try

{

//register notification

notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.int1", AdsTransMode.OnChange, 100, 0, textBox2, typeof(int)));

notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.bool1", AdsTransMode.OnChange, 100, 0, textBox1, typeof(Boolean)));

notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.str1", AdsTransMode.OnChange, 100, 0, textBox3, typeof(String), new int[] { 5 }));

notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.complexStruct1", AdsTransMode.OnChange, 100, 0, textBox5, typeof(ComplexStruct)));

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

button4.Enabled = true;

button3.Enabled = false;

}

并设置当BUTTON3按下时，BUTTON3按键不可用，BUTTON4按键可用

7.以相同的方式设置BUTTON4（取消通知方式）的按键响应

private void button4\_Click(object sender, EventArgs e)

{

//delete registered notifications.

try

{

foreach (int handle in notificationHandles)

adsClient.DeleteDeviceNotification(handle);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

notificationHandles.Clear();

button3.Enabled = true;

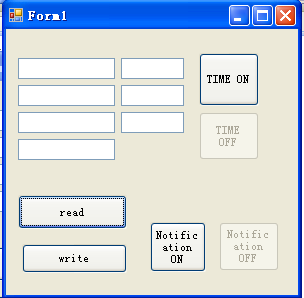
button4.Enabled = false;

}

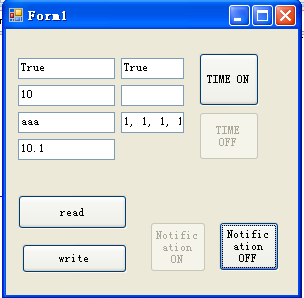
并设置当BUTTON4按下时，BUTTON4按键不可用，BUTTON3按键可用

8.完成之后点击运行程序

效果如下：



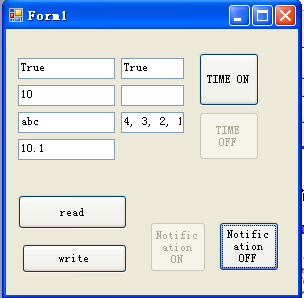
9.点击Notification ON按键，可以看到如下效果：



10.这时候我们在PLC程序中，对变量进行修改：

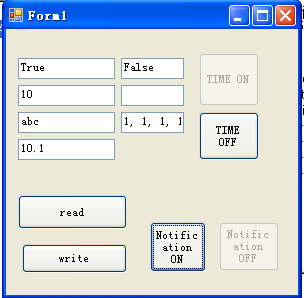


11.切回C#程序，可以看到，C#程序中的TEXTBOX框中也发生了响应的变化



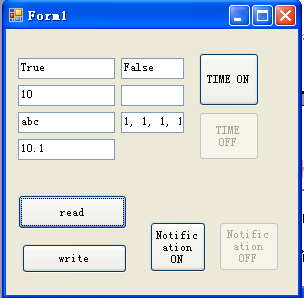
12.再次改变PLC 程序中变量的值：





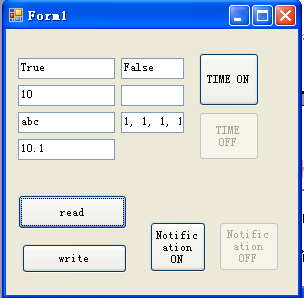
可以看到C#程序的值自动的改变了。

13.这时候点击Notification OFF 按键



这时候再次改变PLC 程序中的值：





可以看到C#程序中的值不会再发生变化了。