My first ACS.NET Application

Microsoft C#

1 Introduction

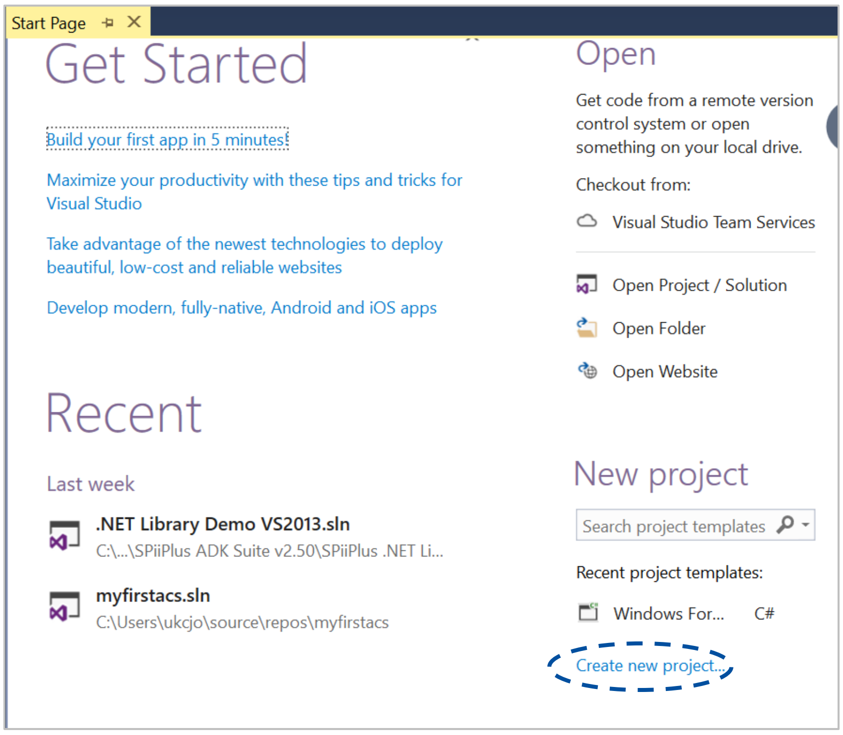
This whitepaper contains a step-by-step explanation on how to create a very simple application in Microsoft C#, which is one of the .NET high-level software languages for interfacing to an ACS Motion Controller. The application connects to an ACS controller (can be physical or simulator), enables the axis and produces an example of motion using jogging.

2 Downloading Visual Studio Community

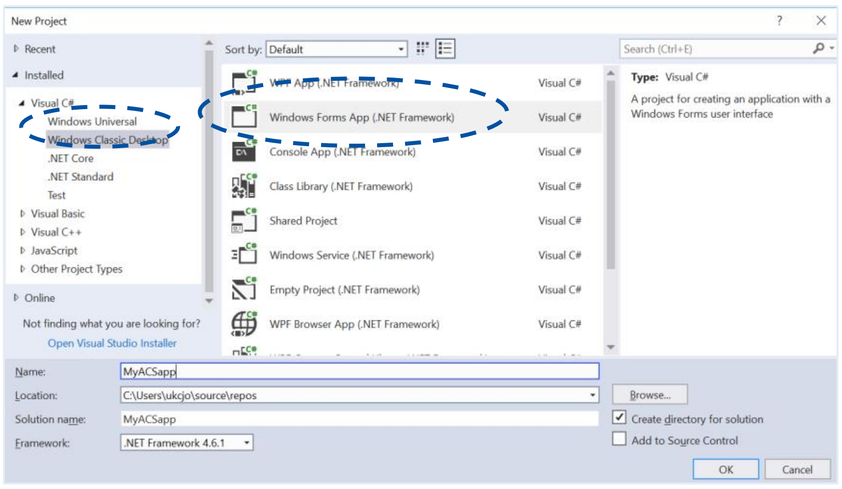
1. Browse to https://www.visualstudio.com/vs/community/
2. Download VS Community by pressing the download button
3. Follow the onscreen instruction to install the software

3 Launching Visual Studio and Creating a Project

1. Launch Visual Studio or your copy of C# and open a new Windows Forms Application
2. You will see the following screen. Click Create new project


Start Page
Get Started
Build your first app in 5 minutes!
Maximize your productivity with these tips and tricks for Visual Studio
Take advantage of the newest technologies to deploy beautiful, low-cost and reliable websites
Develop modern, fully-native, Android and iOS apps
Recent
Last week
.NET Library Demo VS2013.sln
C:\...\SPiiPlus ADK Suite v2.50\SPiiPlus .NET Li...
myfirstacs.sln
C:\Users\ukcjo\source\repos\myfirstacs
Open
Get code from a remote version control system or open something on your local drive.
Checkout from:
Visual Studio Team Services
Open Project / Solution
Open Folder
Open Website
New project
Search project templates
Recent project templates:
Windows For... C#
Create new project...

3. The following dialog box will appear


New Project
Recent
Installed
Visual C#
Windows Universal
Windows Classic Desktop
.NET Core
.NET Standard
Test
Visual Basic
Visual C++
JavaScript
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Framework) Visual C#
Shared Project Visual C#
Windows Service (.NET Framework) Visual C#
Empty Project (.NET Framework) Visual C#
WPF Browser App (.NET Framework) Visual C#
Search (Ctrl+E)
Browse...
OK
Cancel

Type: Visual C#

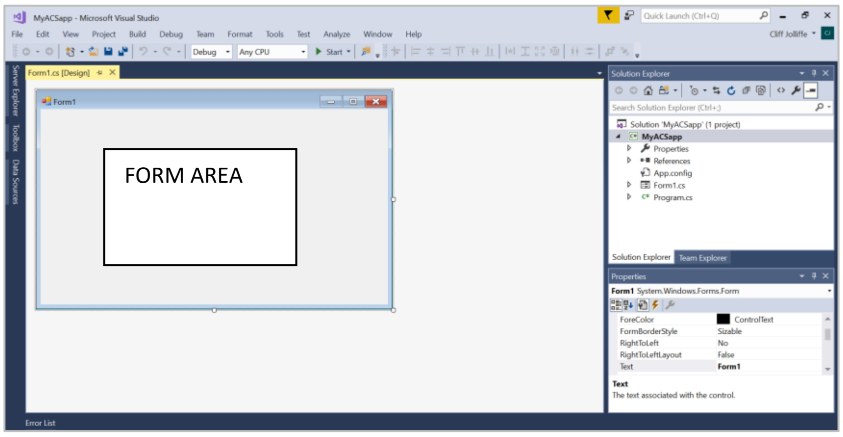
A project for creating an application with a Windows Forms user interface

Name: MyACSapp
Location: C:\Users\ukcjo\source\repos
Solution name: MyACSapp
Framework: .NET Framework 4.6.1
[✓] Create directory for solution
[ ] Add to Source Control

4. Browse the project types on the left hand of the dialog and find Visual C# and Windows Classic Desktop. Select **Windows Forms App** and change the name to MyACSapp.
5. Click **OK** to create the project.

4 Building the Interface

The user interface will consist of five Buttons, one NumericUpDown and two TextBoxes. The buttons will be used to operate different functions typical of a motion application. One TextBox will act as an output to the user in case there are any errors present while running the motion application. The second TextBox will be used to enter the IP address of the controller that we wish to connect to. The NumericUpDown allows the user to enter the speed when jogging. You will see the following screen:

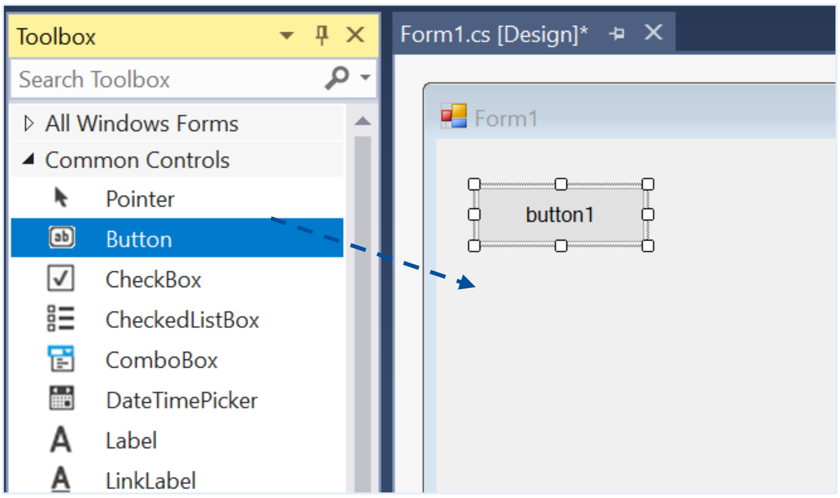


ForeColor	ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1

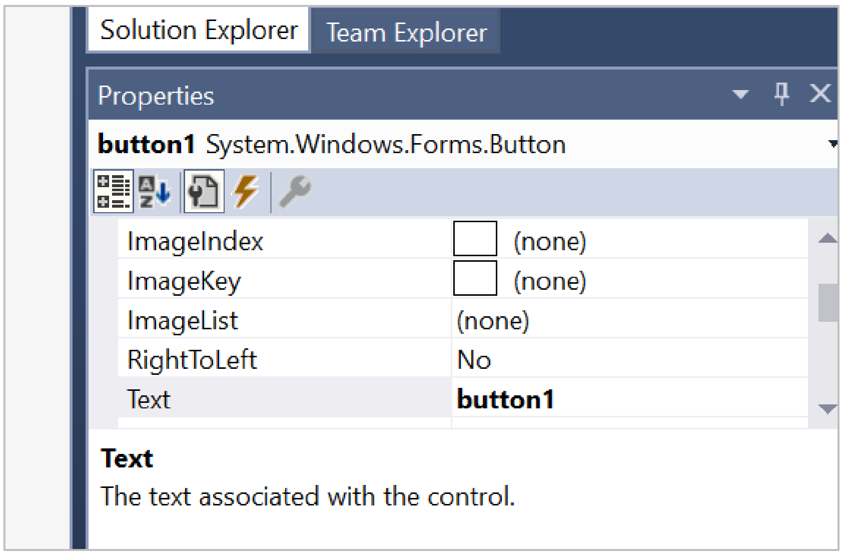
Text
The text associated with the control.

The components that will make the user interface have to be dropped into the form area shown above Form1[Design]. These components are available from the toolbox available on the left hand side of the Visual Studio environment. To keep this open press the at the top right of the Toolbox window.

1. **Click the button control** in the Toolbox and drag it into the form area window as shown below.



2. **Click on this new button.** This will display the properties area at the bottom right of the screen:



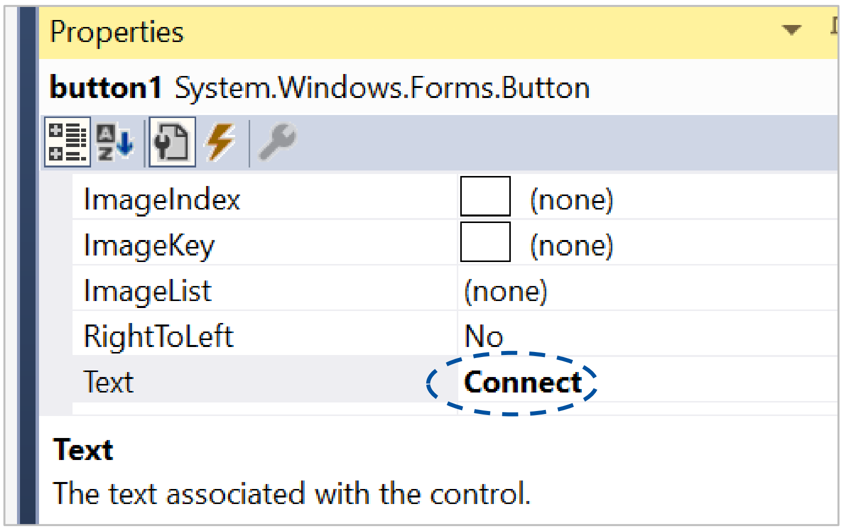
button1 System.Windows.Forms.Button

ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	**button1**

Text
The text associated with the control.

The properties listed here are only for the button1 object that you have selected.

3. **Change the text property to 'Connect'**. The Text property changes what is displayed in the user interface.



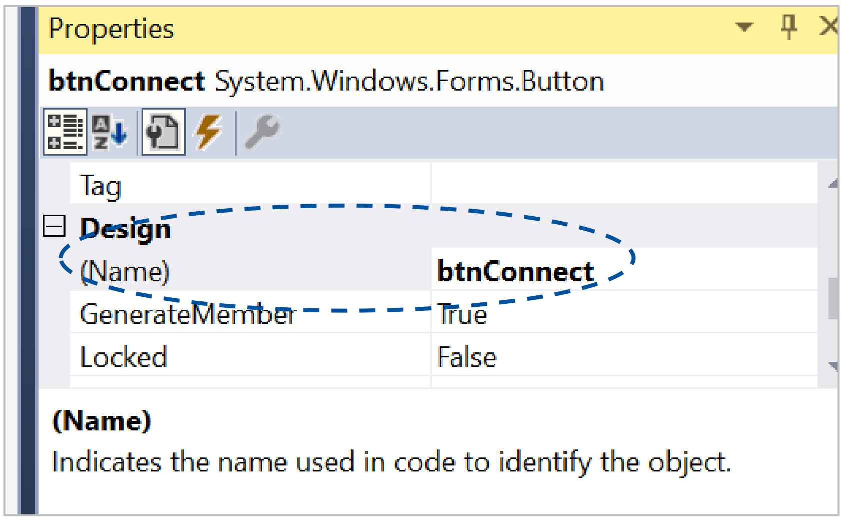
Properties

button1 System.Windows.Forms.Button

ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	**Connect**

Text
The text associated with the control.

4. Scroll down the list, **change the (Name) property** to **'btnConnect'**. The (Name) property changes the reference name used in the underlying code. This is found under the Design Section



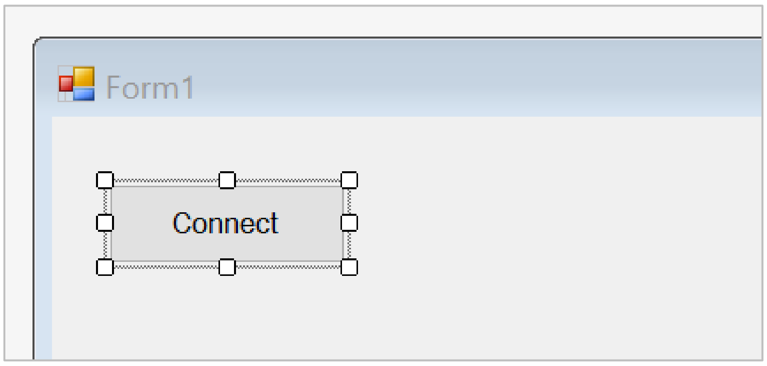
Properties

btnConnect System.Windows.Forms.Button

Tag	
Design	
(Name)	**btnConnect**
GenerateMember	True
Locked	False

(Name)
Indicates the name used in code to identify the object.

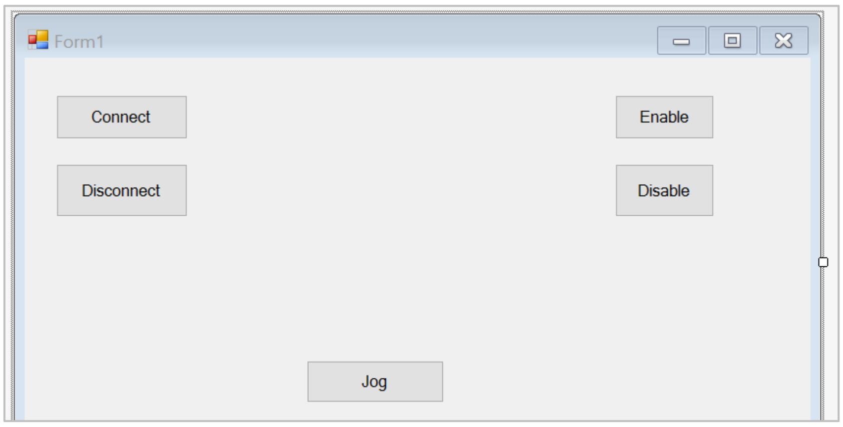
5. You should now see the following in the form area:



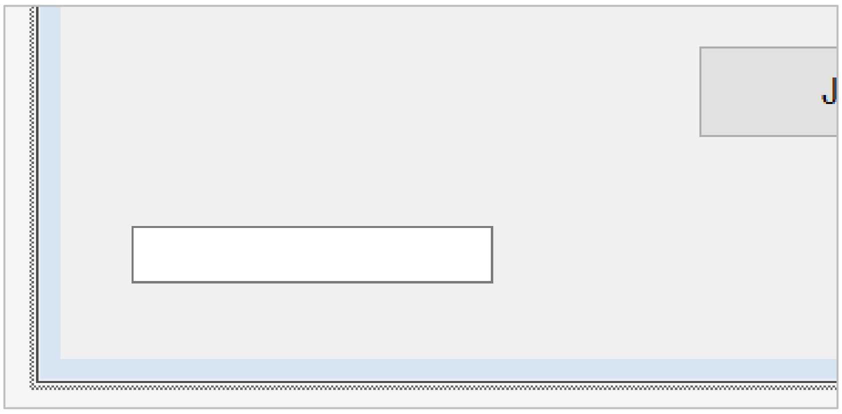
6. **Repeat the same process for following Button Controls** as shown in the table below. It's best to repeat the above from step 1, and not copy and paste.

Control Type	Design Name property	Text property
Button	btnConnect	Connect
Button	btnDisconnect	Disconnect
Button	btnEnable	Enable
Button	btnDisable	Disable
Button	btnJog	Jog

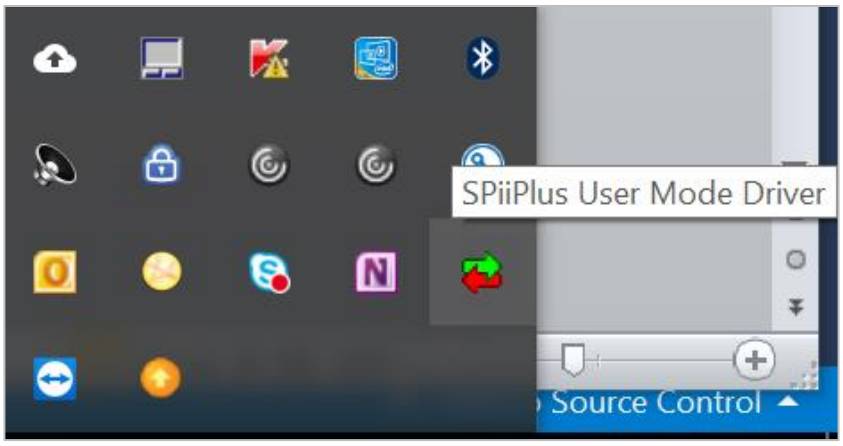
7. Arrange the form as shown below:



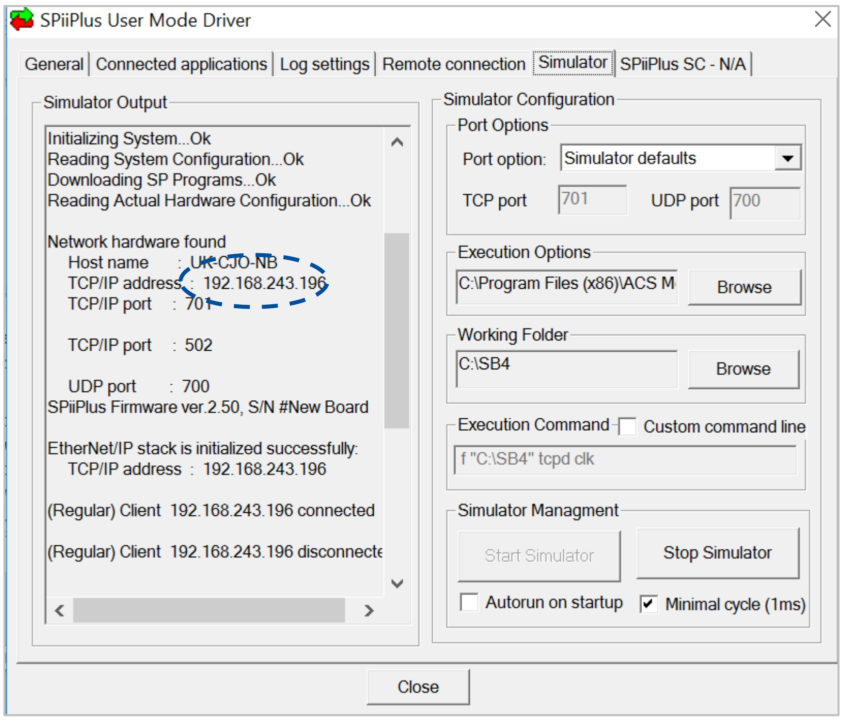
8. A text box will show the user if an error occurs while running the application. **Add TextBox** from the Toolbox by onto the form as before.



9. As with the buttons previously, **Change (Design) Name property to txtErrorInfo**. The Text property does not need to be changed because we want to start the application with an empty error indicator.
10. We will add a second text box that will allow entry of the IP address of the controller we wish to connect to. **Change (Design) Name property to RemoteAddressTB**. The Text property can hold the IP address of the controller. For our example we will use the IP address that ACS uses for its motion simulator.
11. To find the Simulator IP address, **double-click the green and red arrow icon** found in the windows application tray.



12. This will bring up the following window. **Click on the simulator tab.**



SPiiPlus User Mode Driver

General | Connected applications | Log settings | Remote connection | Simulator | SPiiPlus SC - N/A

Simulator Output

```
Initializing System...Ok
Reading System Configuration...Ok
Downloading SP Programs...Ok
Reading Actual Hardware Configuration...Ok

Network hardware found
    Host name       : UK-CJO-NB
    TCP/IP address  : 192.168.243.196
    TCP/IP port     : 701

    TCP/IP port     : 502

    UDP port        : 700
SPiiPlus Firmware ver.2.50, S/N #New Board

EtherNet/IP stack is initialized successfully:
    TCP/IP address  : 192.168.243.196

(Regular) Client  192.168.243.196 connected

(Regular) Client  192.168.243.196 disconnecte
```

Simulator Configuration

Port Options
Port option: Simulator defaults
TCP port: 701 UDP port: 700

Execution Options
C:\Program Files (x86)\ACS M Browse

Working Folder
C:\SB4 Browse

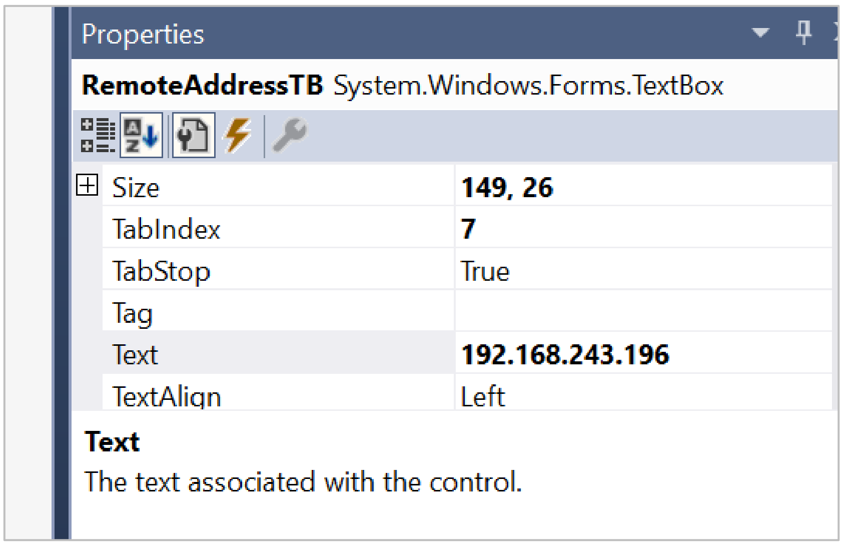
Execution Command ☐ Custom command line
f "C:\SB4" tcpd clk

Simulator Managment
Start Simulator Stop Simulator
☐ Autorun on startup ☑ Minimal cycle (1ms)

Close

This shows the IP address of the simulator

13. **Add the value of the IP address to the Text field of the RemoteAddressTB TextBox**.

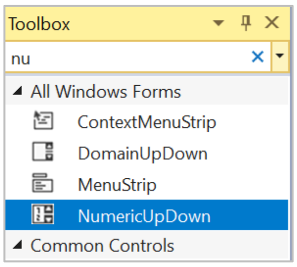

Properties
RemoteAddressTB System.Windows.Forms.TextBox

Size	149, 26
TabIndex	7
TabStop	True
Tag	
Text	192.168.243.196
TextAlign	Left

Text
The text associated with the control.

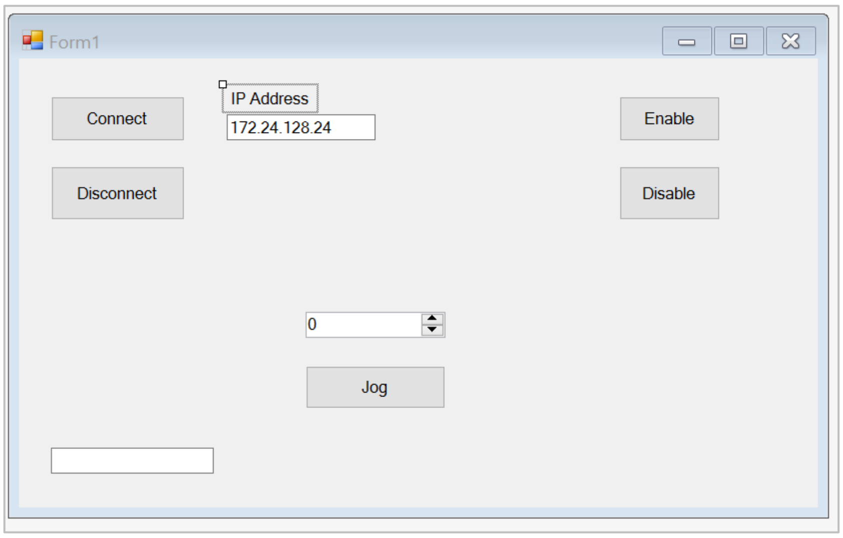
Just for clarity we **will Add a Label** from the Toolbox to add the words 'IP address' above the TextBox. We can change the **Text property** to 'IP Address'.

14. **Add a NumericUpDown indicator** just above the Jog button. This will allow the user of the application to select the desired jog speed. You may need to search for this control.



15. **Change (Design) Name property** to **numFeedrate**. The Text property does not need filled.

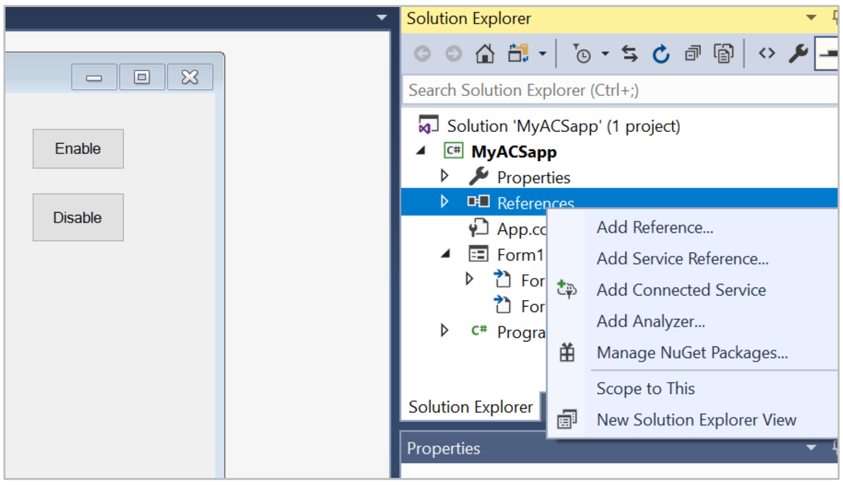
You should have your form layout looking like below.



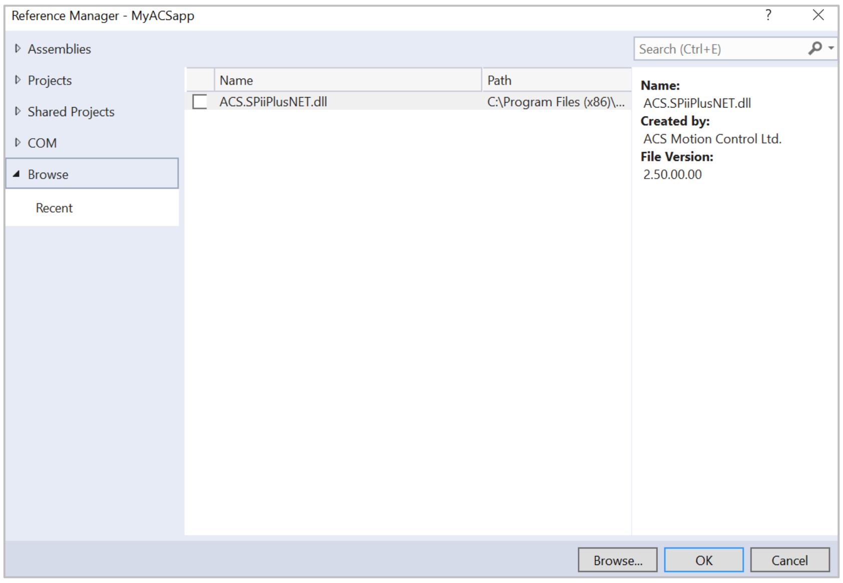
5 Adding References

At this point the user interface components are all in place and we are ready to add the code. You will have to a make a 'reference' to the ACS .NET libraries which are installed with the ACS SPiiPlus MMI Application Studio software.

1. From the Solution Explorer on the right hand side of the C# environment. **Expand References Folder as below.**



2. **Right-click References**, select **Add Reference** which will open the following screen



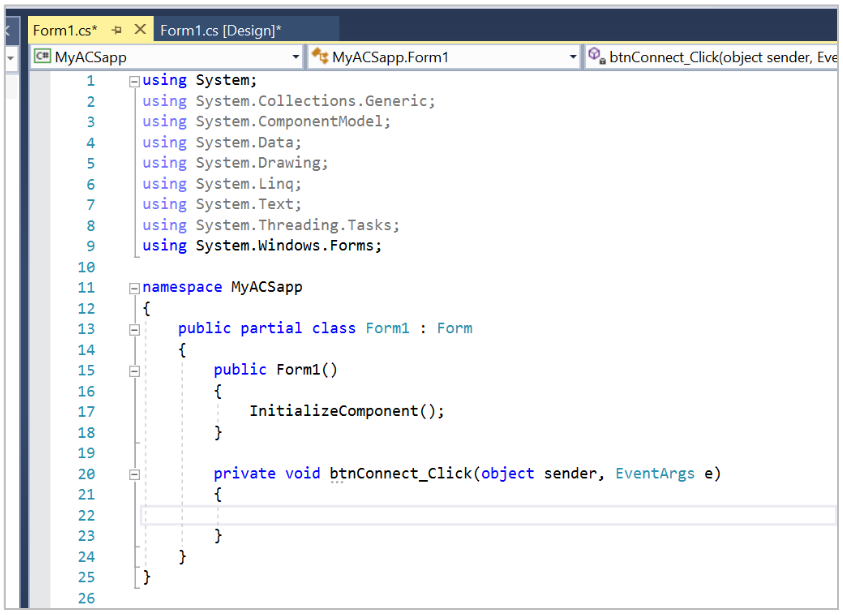
Name	Path
ACS.SPiiPlusNET.dll	C:\Program Files (x86)\...

Name: ACS.SPiiPlusNET.dll
Created by: ACS Motion Control Ltd.
File Version: 2.50.00.00

3. **Click Browse** and go to the folder: C:\Program Files (x86)\ACS Motion Control\CommonFiles
4. **Select 'ACS.SPiiPlusNET.dll' and press OK**

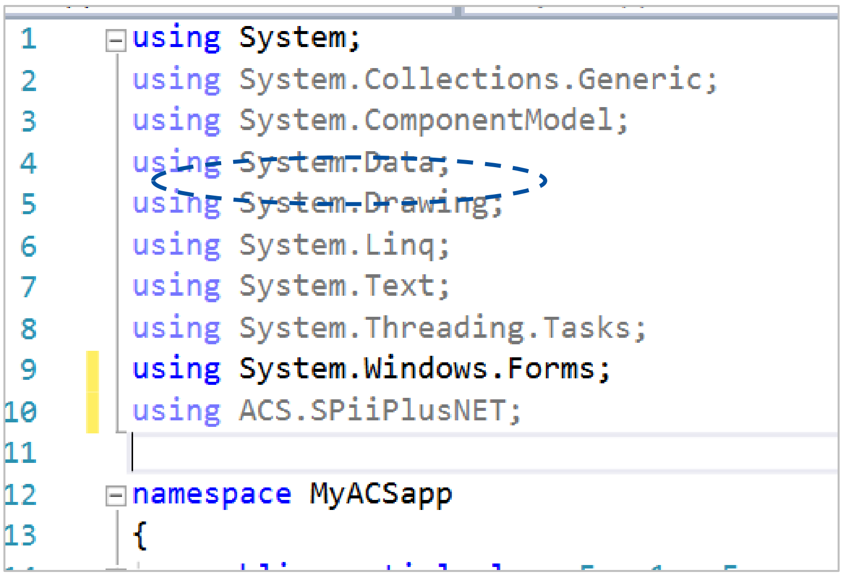
6 Adding the Code

1. In the Form1, **double-click→ Connect button**, the environment will automatically create a code where you can place your own functionality.



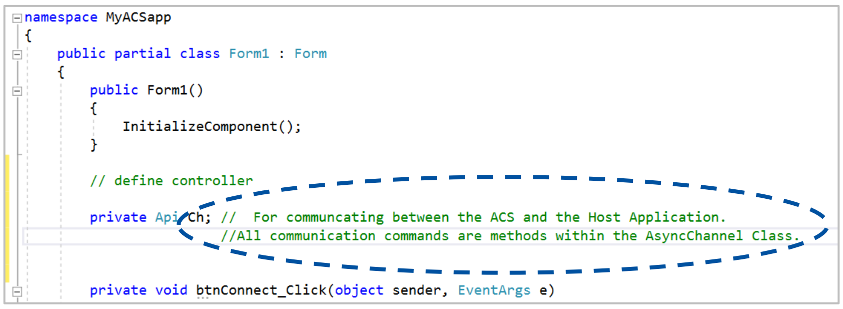
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyACSapp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnConnect_Click(object sender, EventArgs e)
        {

        }
    }
}

```

2. Under the Using Section at the top of the file add the reference as shown below: `using ACS.SPiiPlusNET;`



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
~~using System.Data;~~
~~using System.Drawing;~~
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ACS.SPiiPlusNET;

namespace MyACSapp
{
```

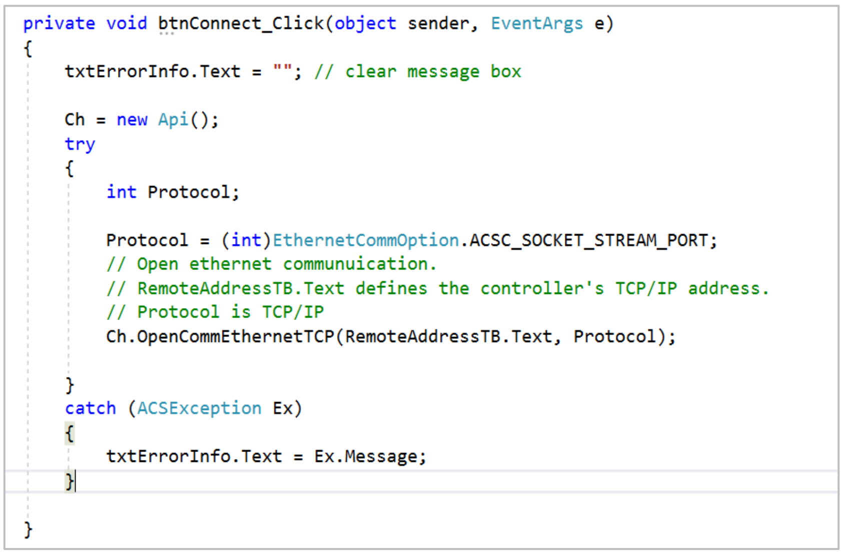
3. Next we need to declare 'Ch' an Object of type API. **Add the line of code** after the **Public Form1() function** as shown below.



```
namespace MyACSapp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // define controller

        private Api Ch; //  For communcating between the ACS and the Host Application.
                        //All communication commands are methods within the AsyncChannel Class.


        private void btnConnect_Click(object sender, EventArgs e)
```

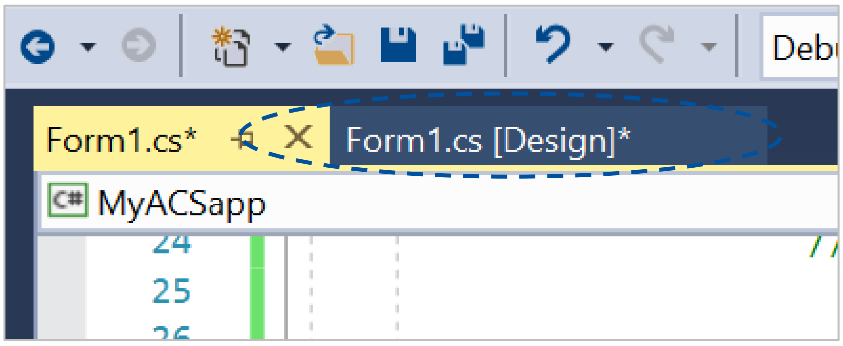
We want the Connect button to establish communication to the controller and if there are any errors to alert the user using our Text-Box txtErrorInfo.

4. **Add the following section of code** within the **Connect** event handler function. Notice with the editor that as you start typing the code it may give you the option to auto-complete. Press tab if a correct suggestion shows and the word will auto-complete. Also notice that any characters following two forward slashes "//" is a comment that the compiler will ignore and is therefore for documentation purposes.

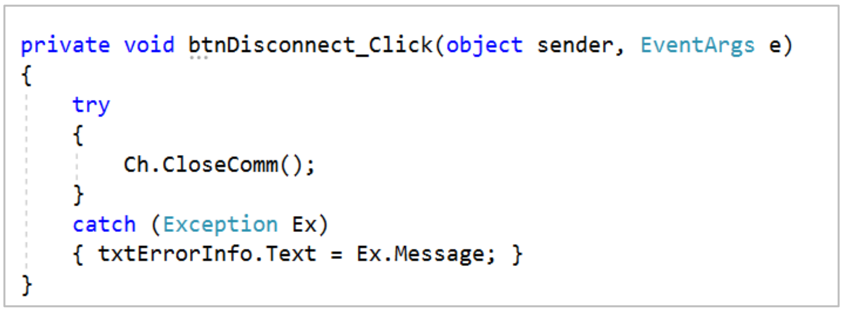


```
private void btnConnect_Click(object sender, EventArgs e)
{
    txtErrorInfo.Text = ""; // clear message box

    Ch = new Api();
    try
    {
        int Protocol;

        Protocol = (int)EthernetCommOption.ACSC_SOCKET_STREAM_PORT;
        // Open ethernet communication.
        // RemoteAddressTB.Text defines the controller's TCP/IP address.
        // Protocol is TCP/IP
        Ch.OpenCommEthernetTCP(RemoteAddressTB.Text, Protocol);

    }
    catch (ACSException Ex)
    {
        txtErrorInfo.Text = Ex.Message;
    }

}
```

5. To get back to the Windows Form **click on the tab "Form1.cs[Design]"** at the top, as shown below.

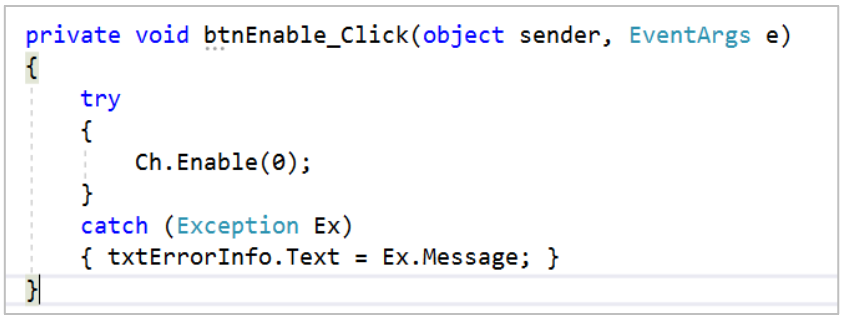


6. **Double-click the Disconnect** button, and add the following code:



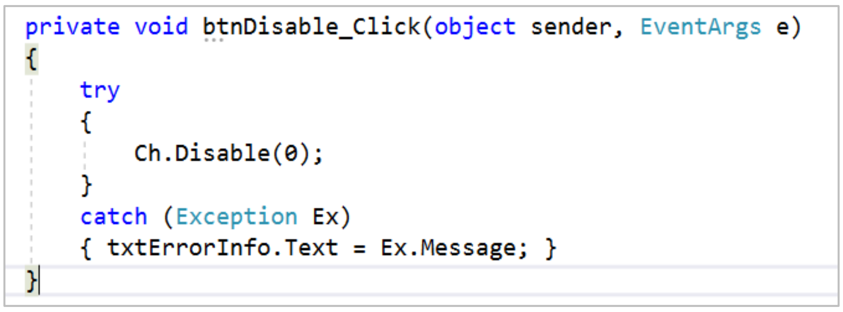
```
private void btnDisconnect_Click(object sender, EventArgs e)
{
    try
    {
        Ch.CloseComm();
    }
    catch (Exception Ex)
    { txtErrorInfo.Text = Ex.Message; }
}
```

7. **Click the tab “Form1.cs[Design]”** at the top again
8. **Double-click the Enable button**, and add the following code:



```
private void btnEnable_Click(object sender, EventArgs e)
{
    try
    {
        Ch.Enable(0);
    }
    catch (Exception Ex)
    { txtErrorInfo.Text = Ex.Message; }
}
```

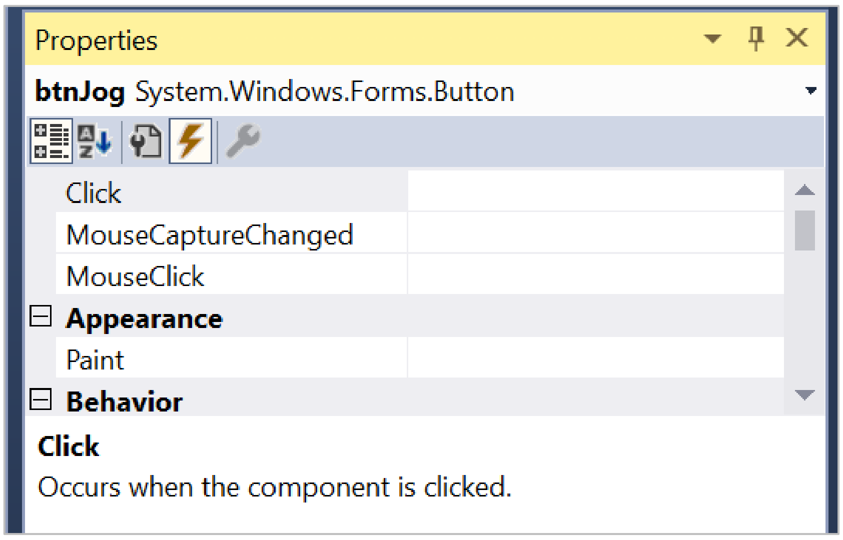
9. **Click the tab “Form1.cs[Design]”** at the top
10. **Double-click the Disable button**, and add the following code:



```
private void btnDisable_Click(object sender, EventArgs e)
{
    try
    {
        Ch.Disable(0);
    }
    catch (Exception Ex)
    { txtErrorInfo.Text = Ex.Message; }
}
```

You have seen that by double-clicking any of the buttons the environment automatically creates the code that becomes the event handler for the event “Click” of the object that you selected. There are other events available, depending on the control that can be used to execute other functionality. In our application the Jog button will use the “MouseDown” and “MouseUp” events. We want to jog the motor while the Jog button is being pressed (MouseDown) but we want to stop the motor from jogging when the button is released (MouseUp event). The next step is to create the code for the MouseDown event for the Jog button. Carry out the following set of steps:

1. **Return to the Form view and click the Jog button** and refer to the **Properties** in the bottom right of the screen
2. **Click →** ⚡ button and the dialog box will show you all of the possible events that the Jog button control can respond to as below:



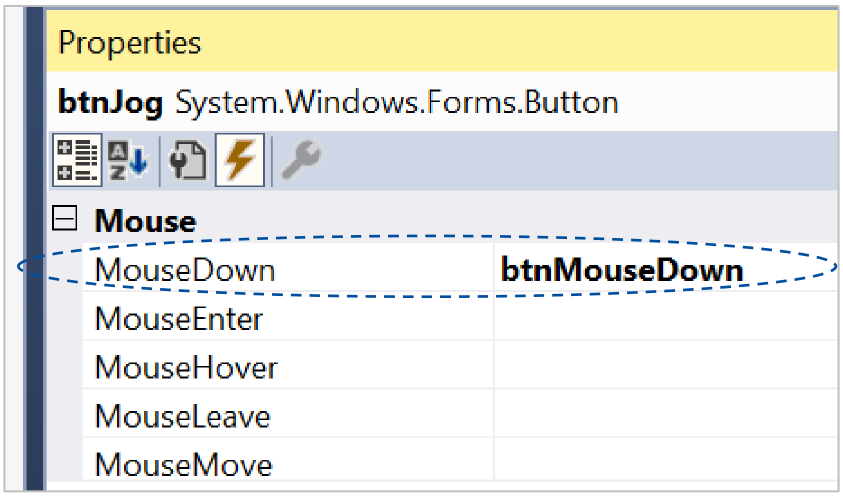
Properties

btnJog System.Windows.Forms.Button

Click	
MouseCaptureChanged	
MouseClick	
Appearance	
Paint	
Behavior	

Click
Occurs when the component is clicked.

3. Find the MouseDown event.
4. **Enter the name of function** to handle the mouse down event as btnJog_MouseDown (as shown below).

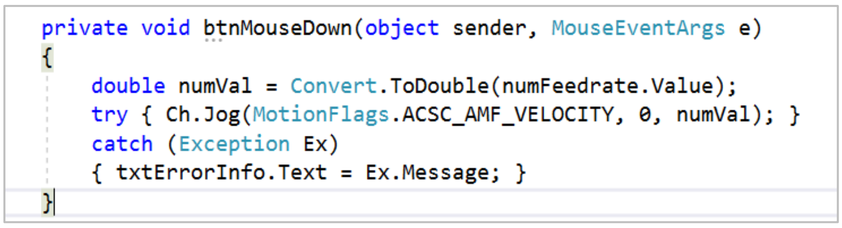


Properties

btnJog System.Windows.Forms.Button

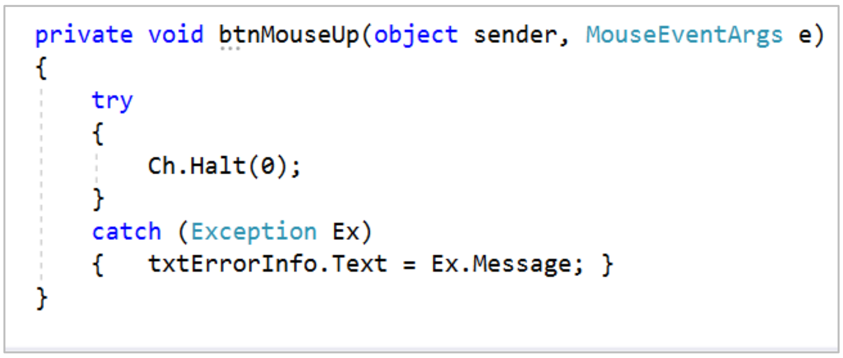
Mouse	
MouseDown	**btnMouseDown**
MouseEnter	
MouseHover	
MouseLeave	
MouseMove	

5. **Press Enter** and code will be generated:
6. Add the following code inside the function:



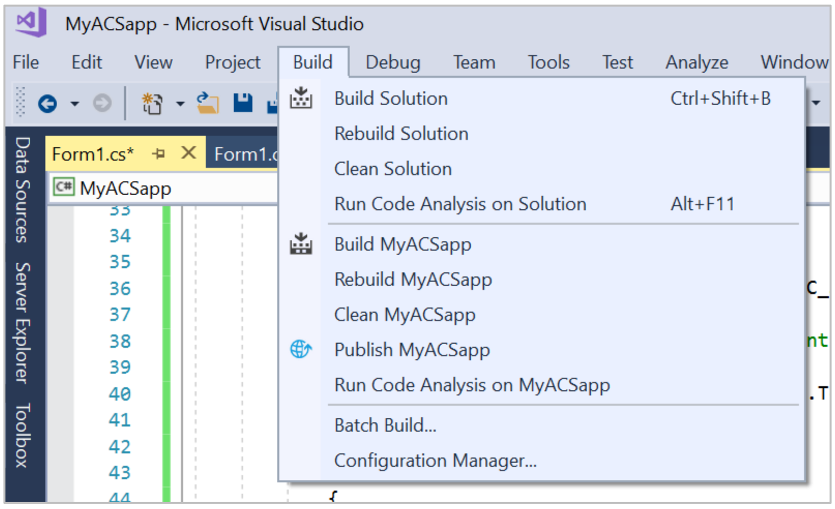
```
private void btnMouseDown(object sender, MouseEventArgs e)
{
    double numVal = Convert.ToDouble(numFeedrate.Value);
    try { Ch.Jog(MotionFlags.ACSC_AMF_VELOCITY, 0, numVal); }
    catch (Exception Ex)
    { txtErrorInfo.Text = Ex.Message; }
}
```

7. **Click the tab "Form1.cs[Design]"** at the top
8. **Repeat from step 1 and add code for btnJog_MouseUp event**. The MouseUp events will halt the motion because we want to make the motion come to a rest after the key is released by the user.



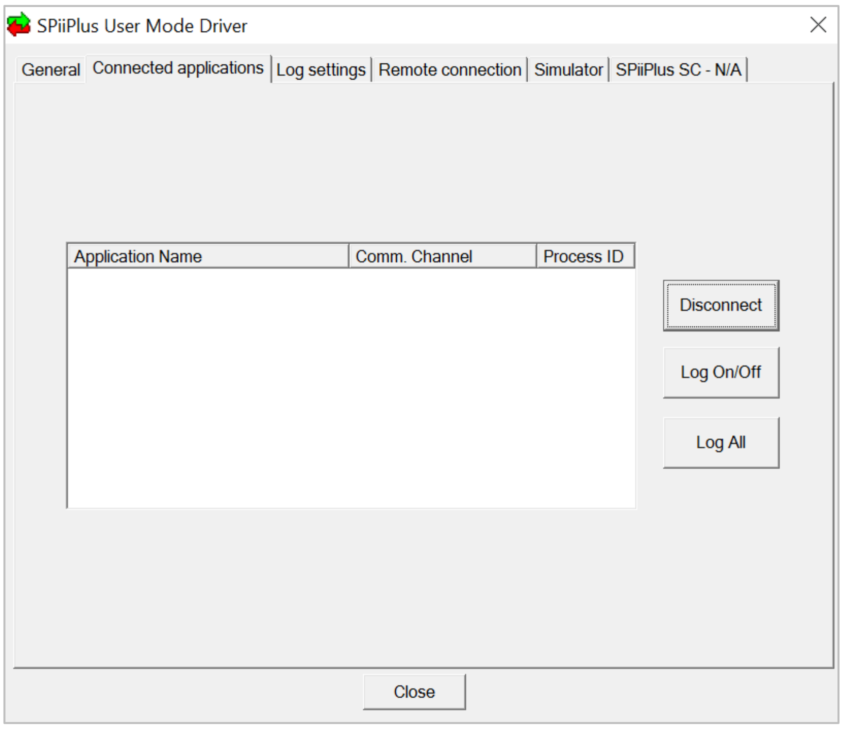
```
private void btnMouseUp(object sender, MouseEventArgs e)
{
    try
    {
        Ch.Halt(0);
    }
    catch (Exception Ex)
    {   txtErrorInfo.Text = Ex.Message; }
}
```

9. Build the application by selecting Build Solution from the Build toolbar.

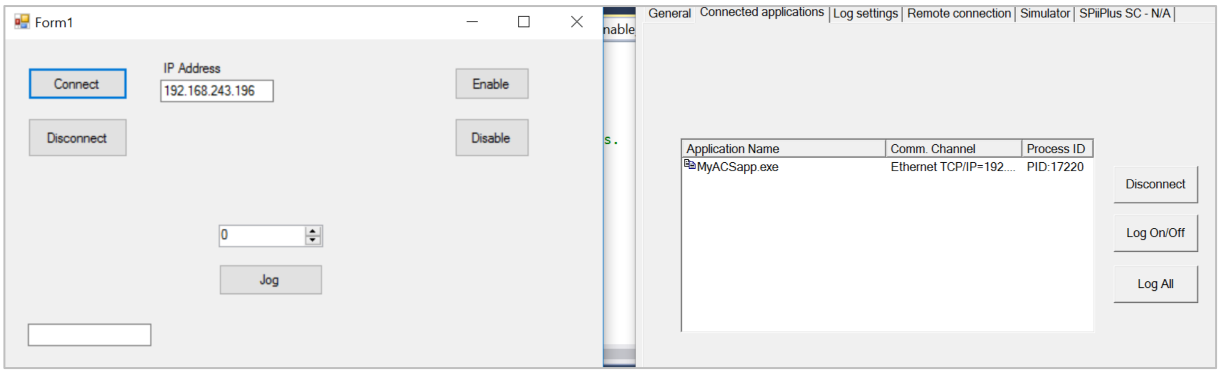


7 Testing your .NET Application

1. Find the SPiiPlus User Mode Driver application you used to find the simulator IP address and select the connected applications.

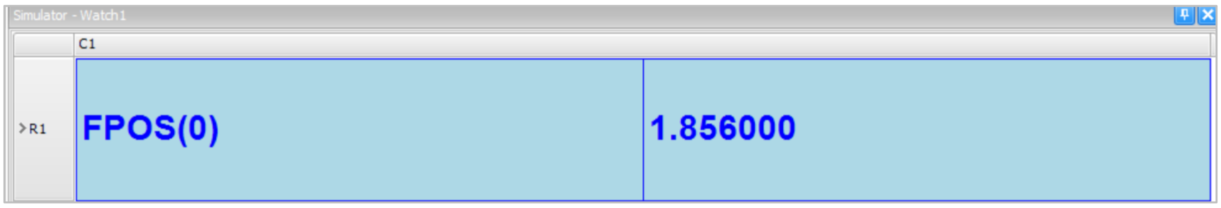


2. Run the C# application now by pressing the button at the top of Visual studio environment.
3. When connect is pressed you should see the application connect in the SPiiPlus User Mode Driver application. When you press disconnect you should see it disappear.



Application Name	Comm. Channel	Process ID
MyACSapp.exe	Ethernet TCP/IP=192....	PID:17220

4. Run the SpiiPlus MMI Software and create a work space that connects to the motion simulator.
5. Bring up the Motion Manager and use the 'create motion' to bring up the Jog Motion.
6. Switch back to your .NET application and make sure you are connected to the simulator. Press the Enable Button (and Disable) and you should see the 'Motor State' change in the Motion Manager.
7. Within the SPiiPlus MMI create a watch component for a variable that reference motion distance for example FPOS(0)



```
FPOS(0)    1.856000
```

8. By entering a jog value within your .NET application and pressing the jog button you can see the distance change.

Author

Dr. Cliff Jolliffe, Head of Segment Marketing Automation at Physik Instrumente (PI) GmbH & Co. KG.

About PI

For many years, PI (Physik Instrumente), founded in 1970, has been a market and technology leader for high precision positioning technology and piezo applications in the semiconductor industry, life sciences, photonics, and in industrial automation. In close cooperation with customers from all over the world, and for 50 years now, PI's specialists (approx. 1,300) have been pushing, again and again, the boundaries of what is technically possible, and developing customized solutions from scratch. Technologies from PI achieve reproducible accuracies in the millionth of a millimeter range. More than 350 granted and registered patents underline the company's claim to innovation.

PI develops, manufactures, and qualifies all core technologies in house, thereby constantly setting new standards for precision positioning: Piezoceramic patch transducers and actuators, electromagnetic drives, and sensors working in the nanometer range. As the majority owner of ACS Motion Control, PI is also a leading global manufacturer of modular motion control systems for multi-axis drive systems, and develops customized complete systems for industrial applications with the highest precision and dynamics.

With six manufacturing sites and 15 sales and service offices in Europe, North America, and Asia, PI is represented wherever high tech solutions are developed and manufactured.