



## School of Information Technology

---

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

---

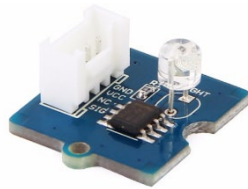
Raspberry Pi Practical : Programming Raspberry Pi Data Communications

### Objectives:

- Learn how to send and receive data between Raspberry Pi and Windows form
- Learn how to save data from sensor into database
- Learn how to create program with state machines to work with Light Sensor and push button

## Exercise 1: Creating DataCommsRpi

1. Today we will be creating a Raspberry Pi program to send light sensor data and push button data to Windows Form and as well receive command from Window form.
2. We shall use Analog Input port **A1** of the GrovePi for Light sensor and Digital Input port **D4** for the push button.

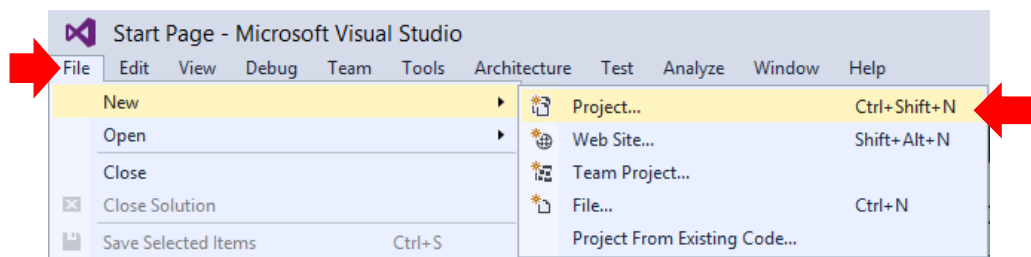


Light Sensor

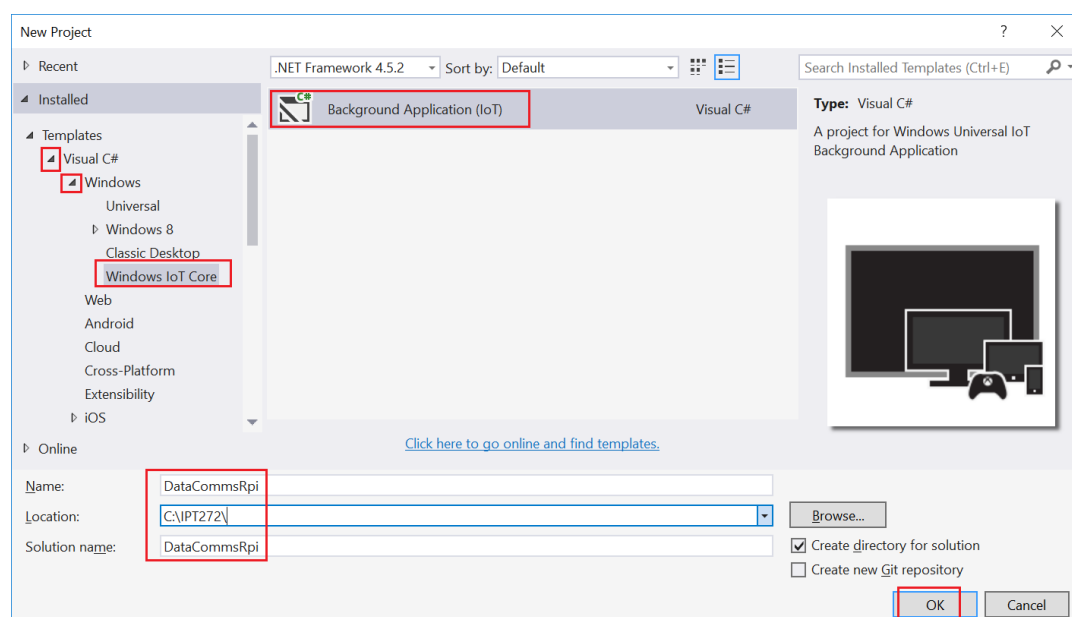


Push Button

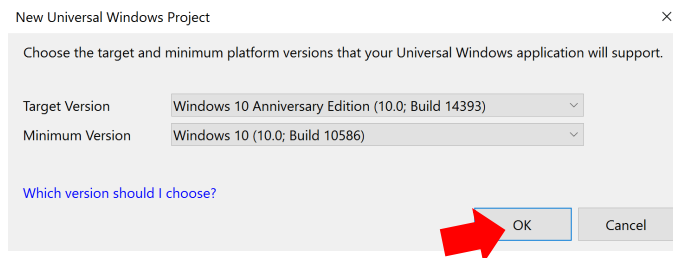
3. Start Visual Studio 2015, Click on File – New - Project.



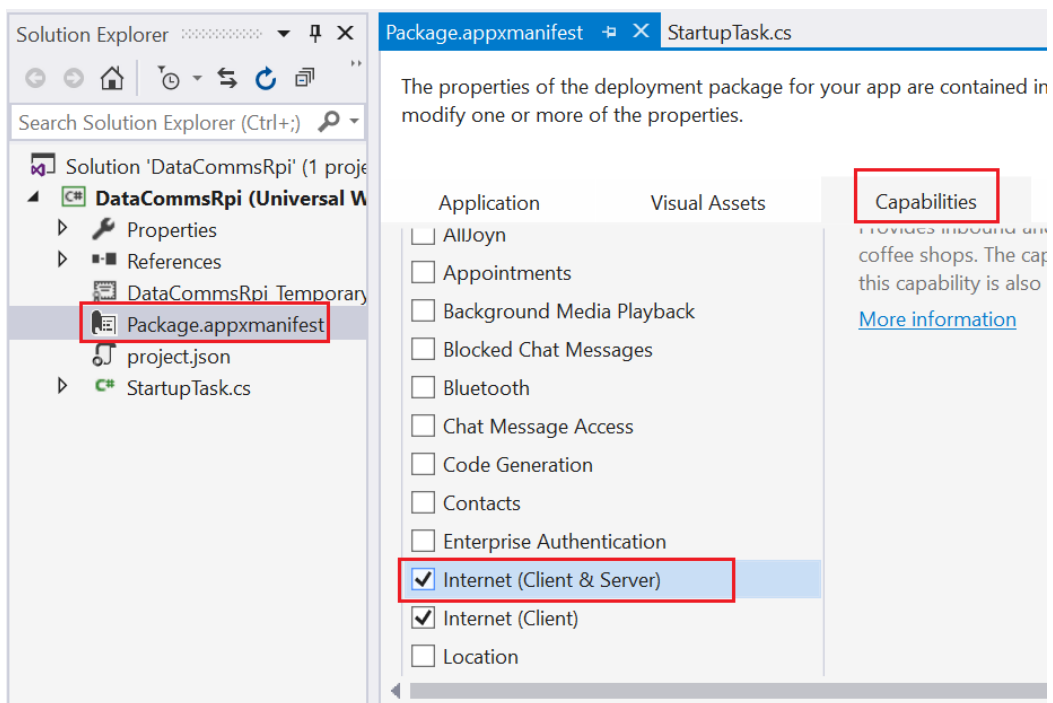
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **DataCommsRpi**, save it in your ITP272 folder and Click OK.



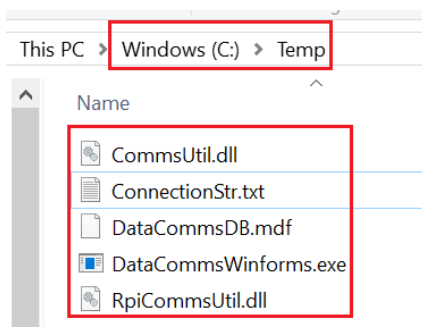
5. You should see this pop-up. Click OK.



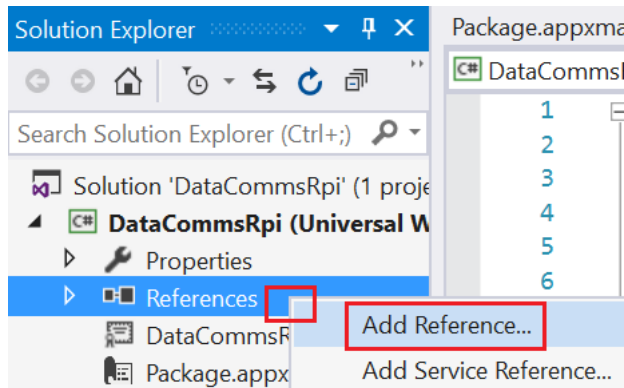
6. To allow data transmission in and out of the Raspberry Pi, double click on **Package.appxmanifest**, click on **Capabilities** Tab and enable **Internet (Client & Server)**



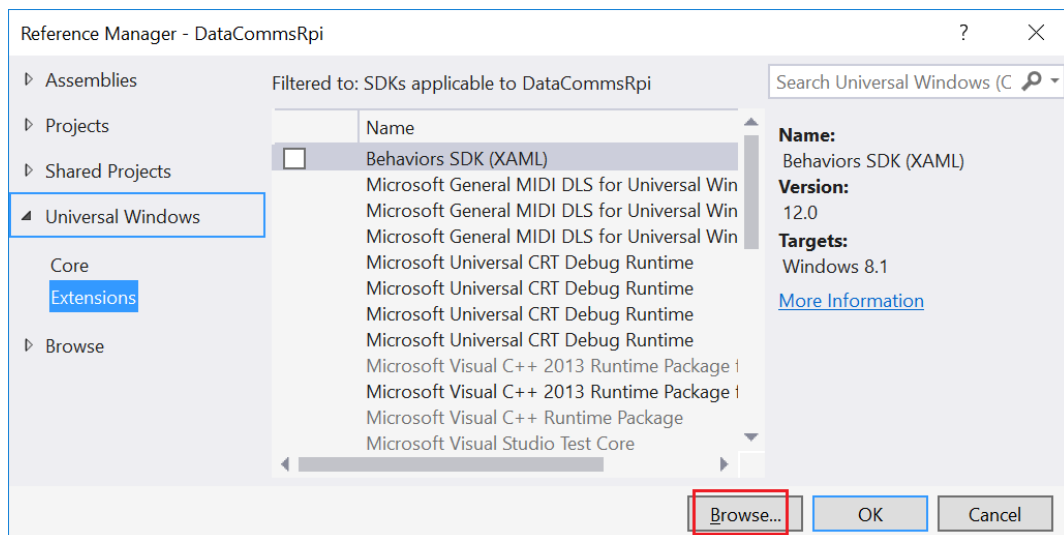
7. Download the helper file **DataCommsStarters.zip** from Blackboard and unzip it to C:/Temp. You should see the following files.



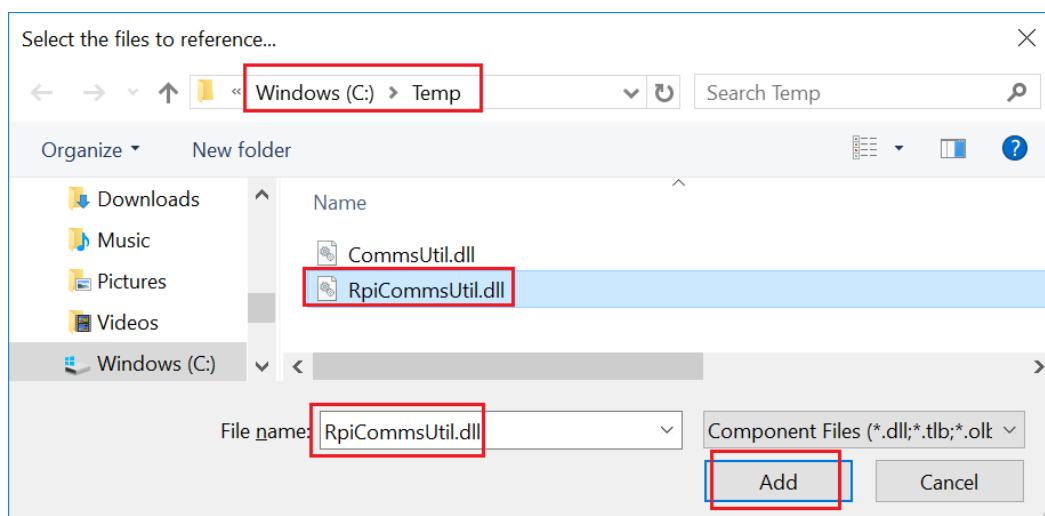
8. Add **RpiCommsUtil.dll** to your workspace. **Right Click** on References on Solution Explorer and Click on Add Reference.



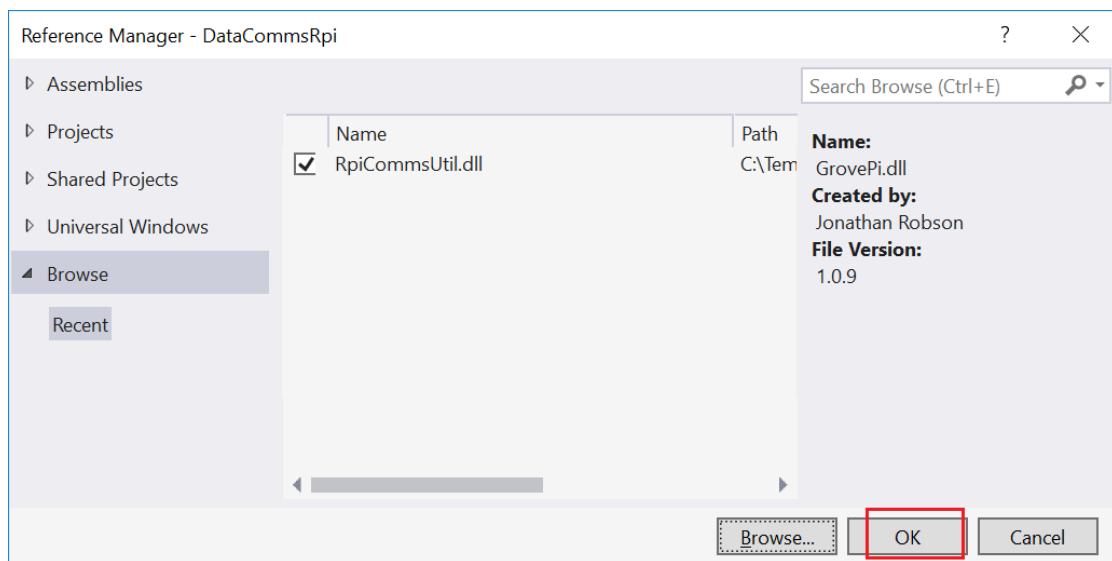
Click on **Browse**



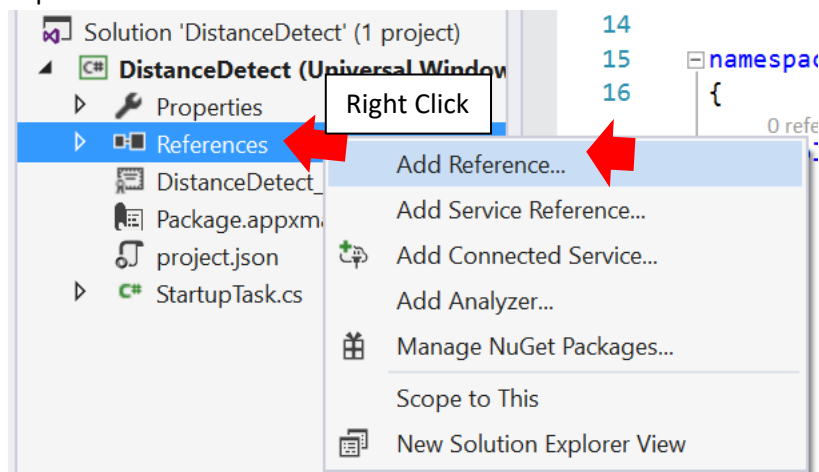
Select **RpiCommsUtil.dll** and click on **Add**.



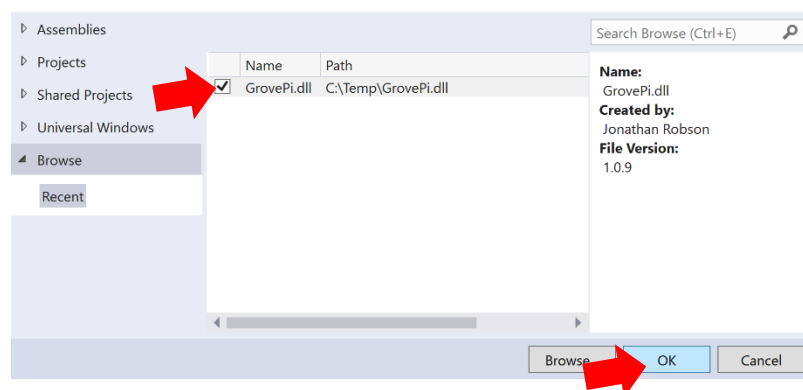
Verify the RpiCommsUtil.dll is checked and click **OK**.



9. Add Reference for the GrovePi as well. Right Click on References on Solution Explorer and Click on Add Reference.



10. Download and select GrovePi.dll as usual. It should appear in your Recent if you've used it before. Check on it and Click OK.



11. Go to your "StartupTask.cs" and type in the following codes above your namespace to include required libraries. It is fine to be in gray initially since you have not used them in the codes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Http;
using Windows.ApplicationModel.Background;

// The Background Application template is documented at

using System.Diagnostics;
using System.Threading.Tasks;
using GrovePi;
using GrovePi.Sensors;
```

12. Add in these codes in the StartupTask.cs

```
public sealed class StartupTask : IBackgroundTask
{
    //State Machine vairables to control different mode of operation
    const int MODE_SENDLIGHT = 1;
    const int MODE_SENDBUTTON = 2;
    static int curMode;    //stores the current mode the program is at

    // Use A1 for light sensor and D4 for Push button
    Pin lightPin = Pin.AnalogPin1;
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);

    //for Data Comms
    DataComms dataComms;
    //This is used to check for data coming in from Winform
    string strDataReceived = "";

    //used by sensor for internal processing
    int lightAdcValue = 800;
    int iPrevAdcValue = 800, iReadAdcValue, iDiff = 0;

    //This is for main logic controller to know that a button is pressed
    private bool buttonPressed = false;
    private bool prevButtonStatus = false;

    int sensorLightAdcValue;
    private bool lightDark = false;
    private bool prevLightDark = false;

    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    } //End Sleep()
}
```

```
//End Sleep()

private int getLight()
{
    iReadAdcValue = DeviceFactory.Build.GrovePi().AnalogRead(lightPin);

    if (iPrevAdcValue > iReadAdcValue)
        iDiff = iPrevAdcValue - iReadAdcValue;
    else
        iDiff = iReadAdcValue - iPrevAdcValue;

    iPrevAdcValue = iReadAdcValue;
    if (iDiff < 100)
        lightAdcValue = iReadAdcValue;

    return lightAdcValue;
}

private async void startButtonMonitoring()
{
    await Task.Delay(100);
    while (true)
    {
        Sleep(100);
        string buttonState = button.CurrentState.ToString();
        if (buttonState.Equals("On"))
        {
            Sleep(100);
            buttonState = button.CurrentState.ToString();
            if (buttonState.Equals("On"))
            {
                buttonPressed = true;
            }
        }
    }
}

private void startLightMonitoring()
{
    while (true)
    {
        getLight();
        if (lightAdcValue > 100)
        {
            lightPressed = true;
        }
        else
        {
            lightPressed = false;
        }
    }
}

//End of Program
```

14. Create the following methods below startButtonMonitoring() to facilitate data transfer between raspberry pi and windows form.

```
//End of startButtonMonitoring()

//this method is automatically called when data comes from Winform
public void commsDataReceive(string dataReceived)
{
    //You can use strDataReceived anywhere in your codes to
    //check for any data coming in from Winform
    strDataReceived = dataReceived;
    Debug.WriteLine("Data Received : " + strDataReceived);
}

//use this method to send data out to Winforms
private void sendDataToWindows(string strDataOut)
{
    try
    {
        dataComms.sendData(strDataOut);
        Debug.WriteLine("Sending Msg : " + strDataOut);
    }
    catch(Exception)
    {
        Debug.WriteLine("ERROR. Did you forget to initComms()?");
    }
}

//This is to setup the comms for data transfer with Winforms
private void initComms()
{
    dataComms = new DataComms();
    dataComms.dataReceiveEvent += new DataComms.DataReceivedDelegate(commsDataReceive);
}
//End InitComms
```

15. Create a handler method for the operation to send light status to windows form

```
//End InitComms

private void handleModeSendLight()
{
    // 1. Define Behaviour in this mode
    if (sensorLightAdcValue <= 500) //must be same threshold as Winform
        lightDark = true;
    else
        lightDark = false;

    if (prevLightDark != lightDark) //send only when change of status
        sendDataToWindows("LIGHT=" + sensorLightAdcValue);

    prevLightDark = lightDark; //must always update

    // 2. Must write the condition to move on to other modes
    if (strDataReceived.Equals("SENDERBUTTON"))
    {
        //Move on to Mode Send Button status when button is pressed
        curMode = MODE_SENDBUTTON;
        Debug.WriteLine("===Entering MODE_SENDBUTTON===");
    }
    strDataReceived = ""; //must always clear after processing
}
//End handleModeSendLight()
```



16. Create a handler method for the operation to send button status to windows form

```
//End handleModeSendLight()

private void handleModeSendButton()
{
    // 1. Define Behaviour in this mode
    if(buttonPressed != prevButtonStatus) //send only when change of status
    {
        sendDataToWindows("BUTTON=" + buttonPressed);
    }

    prevButtonStatus = buttonPressed;
    buttonPressed = false; //clear after processing

    // 2. Must write the condition to move on to other modes
    if (strDataReceived.Equals("SENDLIGHT"))
    {
        //Move on to Mode Send Button when button is pressed
        curMode = MODE_SENDBUTTON;
        Debug.WriteLine("===Entering MODE_SENDBUTTON===");
    }
    strDataReceived = ""; //must always clear after processing
} //End handleModeSendButton()
```

The program is written to have 2 modes of operation. It starts in MODE\_SENDBUTTON where it will send the light sensor values to window form. Depending on the design of the project idea, it is usually not feasible to send all values to the window forms to be saved into database as it is a waste of storage space and often not easy to make sense of the data saved. Thus the codes in handleModeSendLight() only send the light value when there is a change of status.

The program is also designed to receive command from Windows form so that windows form can tell Raspberry Pi to send Button status or Light status. (This behaviour is just used to illustrate how you can process data coming in from windows form. You do not need to follow this behaviour in your project) When the raspberry Pi receives command "SENDBUTTON", it will switch to MODE\_SENDBUTTON whereby it will send the button values to windows form. Similar to the send light values, it only send the data over when there is a change of button status.

When you're sending data over to windows form, you need to let window form know what data it is so that windows form knows how to process the data. Thus you need to add an identifier to the data values so that window form can use the identifier to know what data is coming in. For light value, we use "LIGHT=" and so over at windows form, you also need to check for "LIGHT=" to process it as light sensor values. Both Raspberry Pi and Windows form must use the **SAME** identifier and **CASE SENSITIVE**. For button status, the identifier is "BUTTON=". So over at windows, you need to check for all the various identifiers to process the different data. Likewise when you send data from window forms to raspberry Pi, you need make sure both window form and raspberry pi are using some identifiers. In this Lab, the window forms send 2 different commands which are "SENDBUTTON" and "SENDLIGHT" and raspberry Pi need to handle them.

17. Lastly, add the codes for the Run() method

```
public void Run(IBackgroundTaskInstance taskInstance)
{
    //
    // TODO: Insert code to perform background work
    //
    // If you start any asynchronous methods here, prevent the task
    // from closing prematurely by using BackgroundTaskDeferral as
    // described in http://aka.ms/backgroundtaskdeferral
    //

    initComms(); //Must start before data transfer can work

    //Start the button self monitoring method
    startButtonMonitoring();

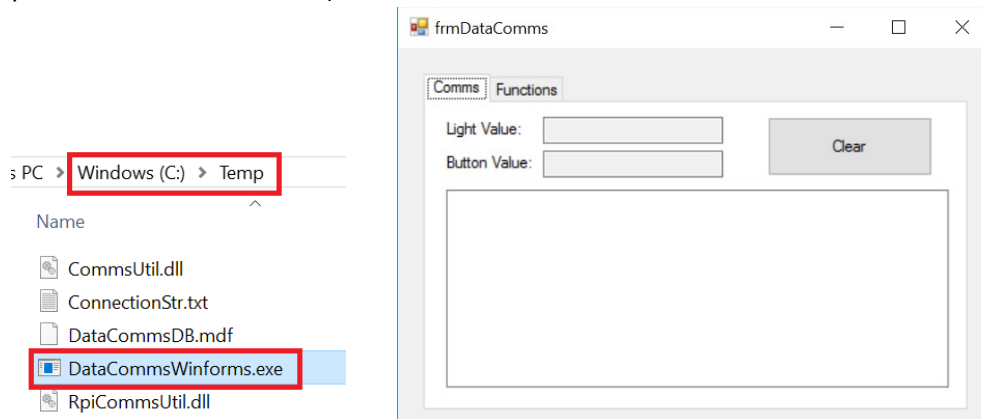
    //Init Mode
    curMode = MODE_SENDLIGHT;
    Debug.WriteLine("===Entering  MODE_SENDLIGHT===");

    //This makes sure the main program runs indefinitely
    while (true)
    {
        Sleep(300);
        sensorLightAdcValue = getLight(); //get light from sensor
        Debug.WriteLine("Sensor light = " + sensorLightAdcValue);

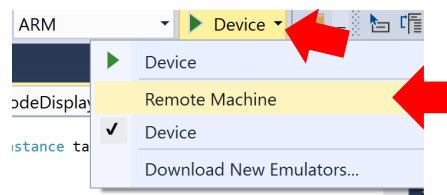
        //state machine
        if (curMode == MODE_SENDLIGHT)
            handleModeSendLight();
        else if (curMode == MODE_SENDBUTTON)
            handleModeSendButton();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

We have now completed the codes for Raspberry Pi to perform data transfer. We shall now use the **DataCommsWinforms.exe** to test with our program. You'll be learning how to code the DataCommsWinforms later in the 2<sup>nd</sup> part of your practical

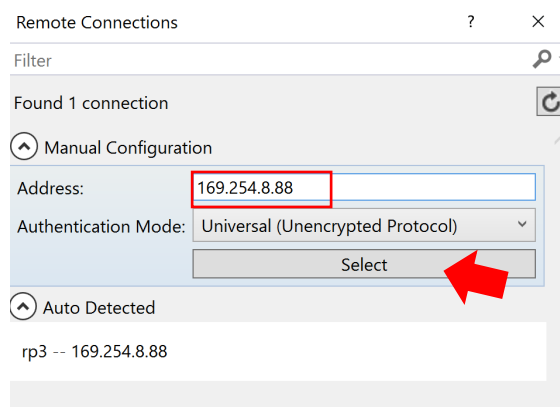
18. Double click on the **DataCommsWinforms.exe** (make sure the **CommsUtil.dll** is present in the same folder)



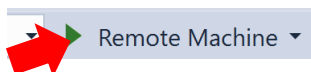
19. In order to deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



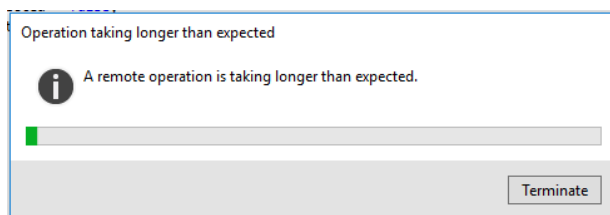
20. Key in the address manually like shown below. Click on **Select** after that.



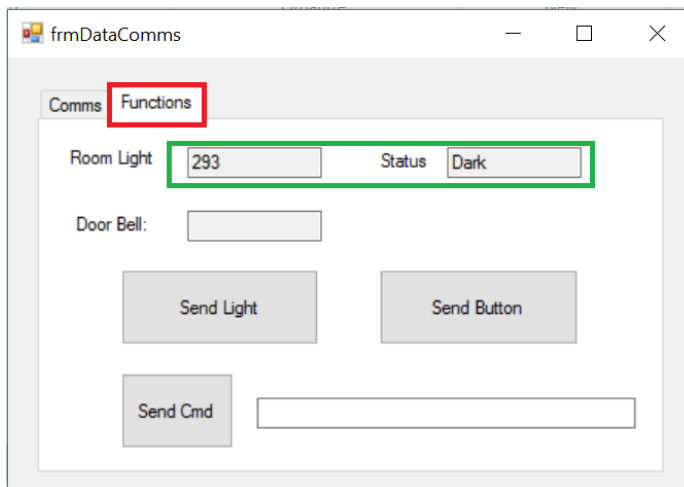
21. Click on Remote Machine to Run the Program.



22. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.

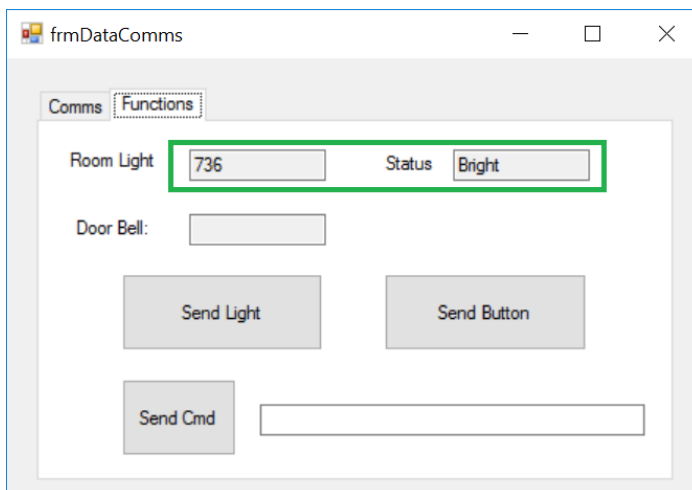


23. Upon successful deployment, you should see “Remote Host Connected” on the Output windows. On the Windows form, click on the “Functions” Tab and then cover your light sensor with your hand now.



You should see the Room Light value and status is **Dark**

Now remove the hand and expose the light sensor to room light.

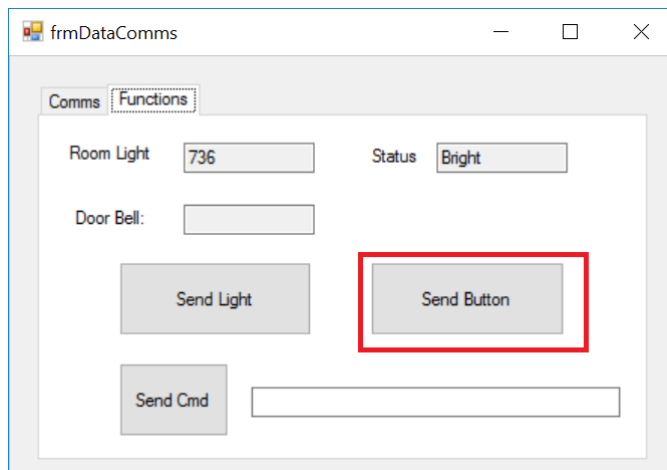


You should see the Room Light value and status is **Bright**

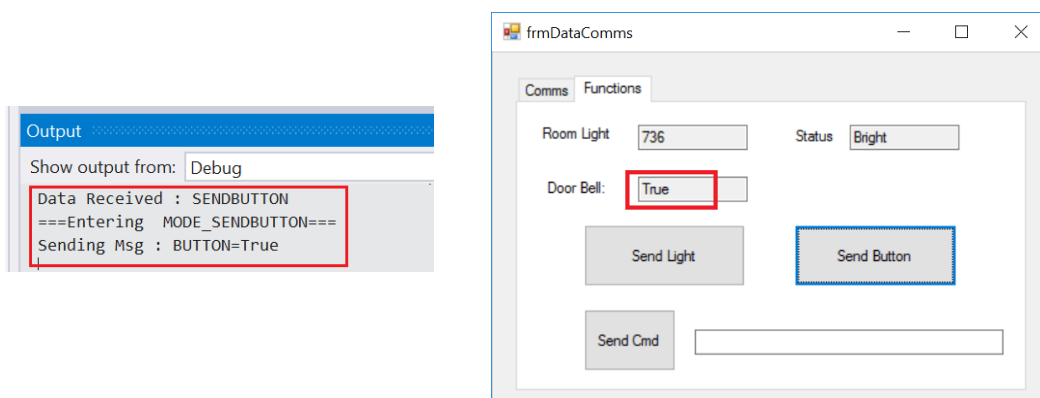
**(Note:** The Threshold has been set to > 500 for Bright. If you room light is dim, you can lower the threshold to suitable value or shine your phone torch)  
You should also see the light value sent over to windows form on your Raspberry Pi Output.

```
Output
Show output from: Debug
backgroundTaskHost.exe (CoreCL
'backgroundTaskHost.exe' (CoreCL
'backgroundTaskHost.exe' (CoreCL
===Entering MODE_SENDLIGHT===
Remote Host Connected
Sending Msg : LIGHT=293
Sending Msg : LIGHT=736
```

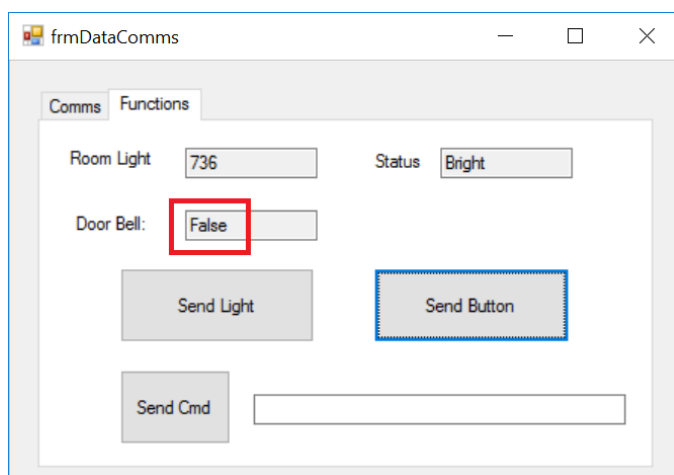
Click on “Send Button” button to ask Raspberry Pi to send button status over.



You'll see that the Raspberry pi received the command “**SENDERBUTTON**” and switch to **MODE\_SENDERBUTTON**. Now press and hold on the push button and you should see push button status of **True** is sent to window form.



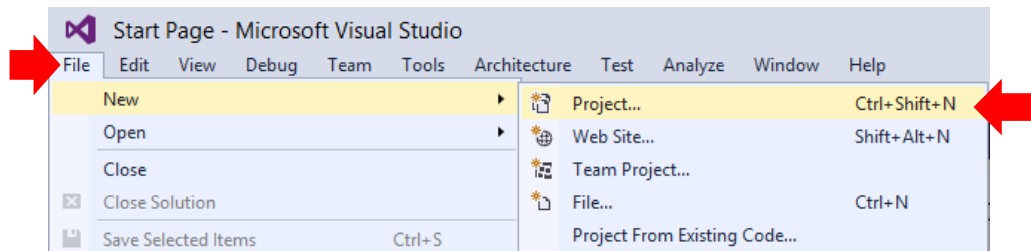
Release the push button and you will see the button status **False** is sent.



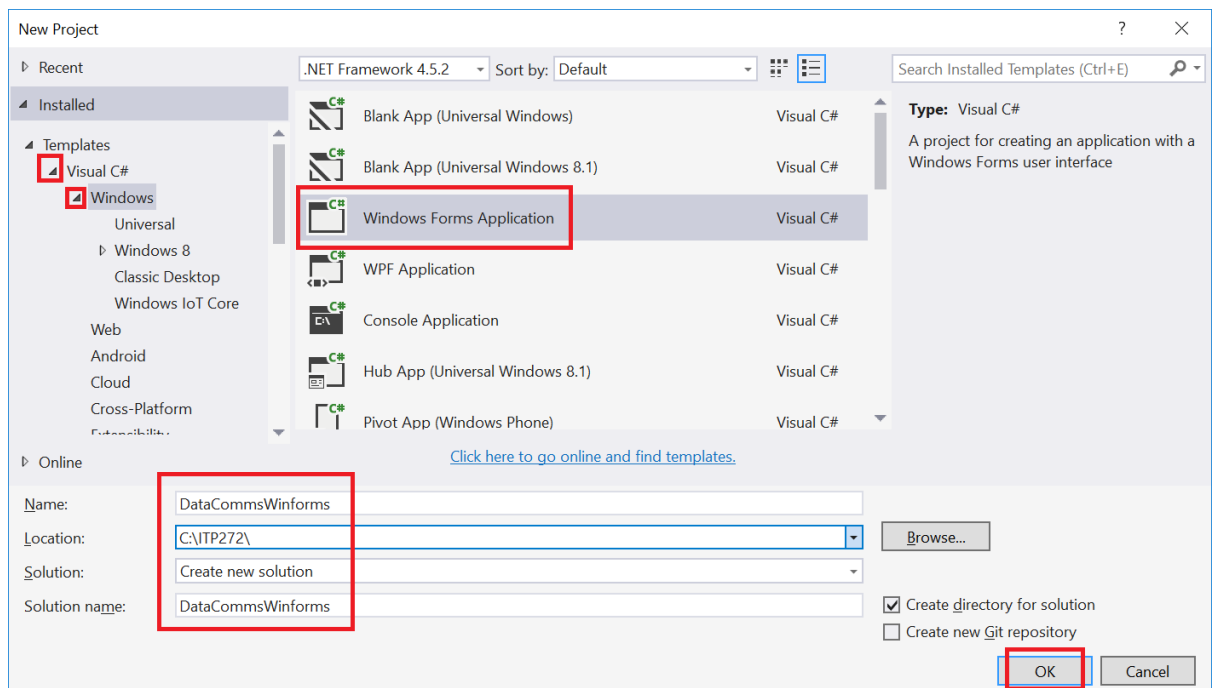
You can send “Send Light” command to switch back to receive Light sensor values.

## Exercise 2: Creating DataCommsWinforms

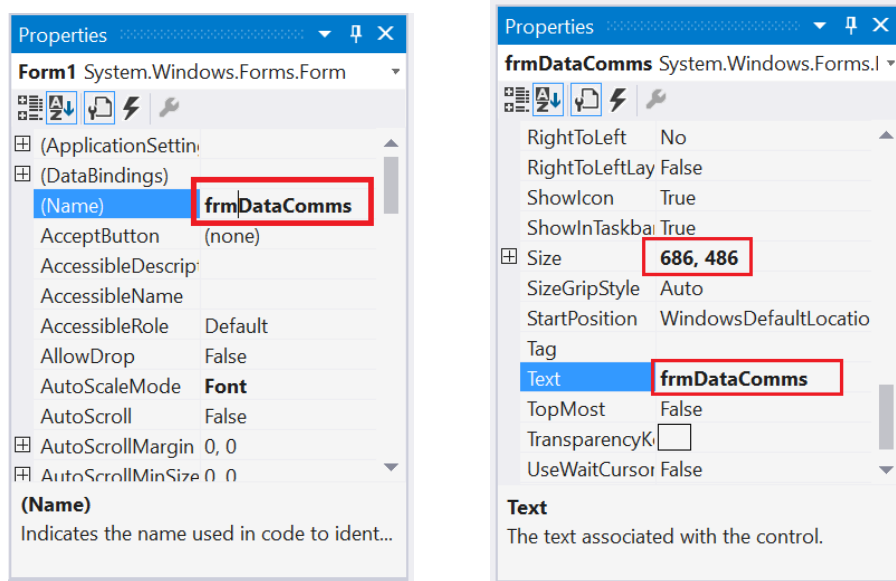
1. In this exercise, we will be creating a program **DataCommsWinforms** as stated in the previous page.
2. Start Visual Studio 2015, Click on File – New - Project.



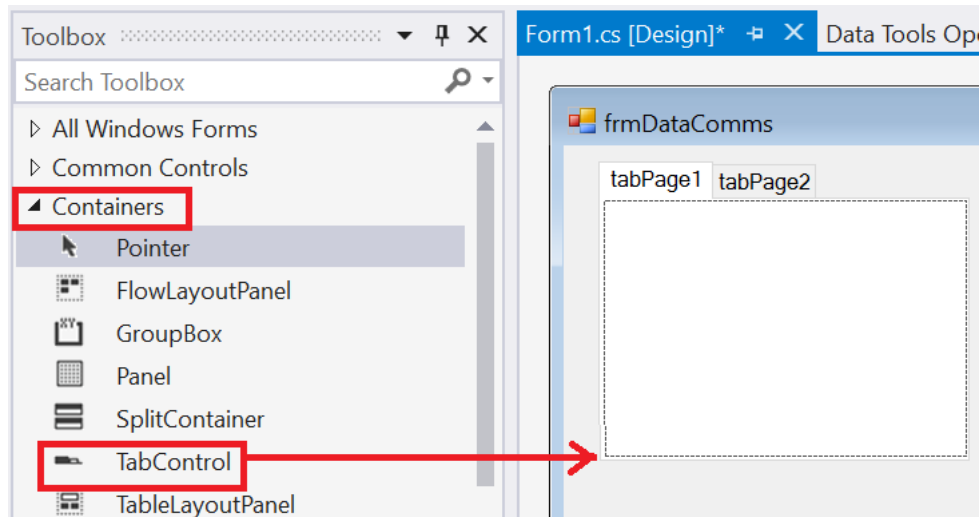
3. From the Installed Templates, select Visual C# – Windows – Windows Form Application. Name your project as **DataCommsWinforms**, save it in you ITP272 folder and Click OK.



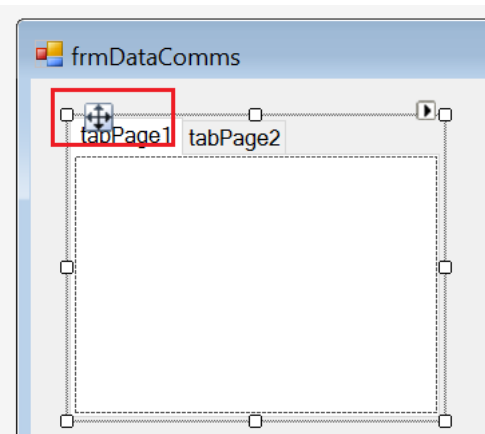
4. Click on the form, Under it's properties, change the name, size and text



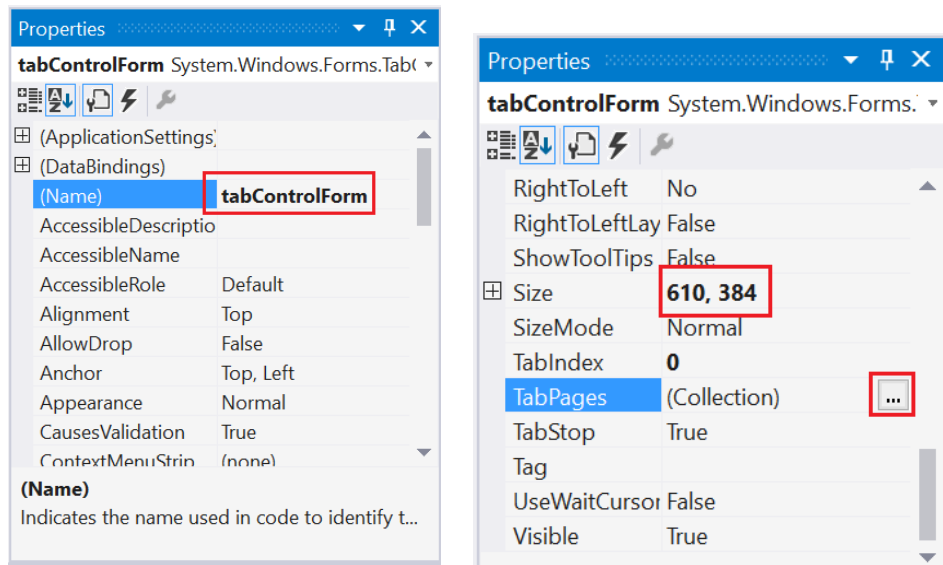
5. At the Toolbox, Under **Containers**, Drag a **TabControl** over to your form



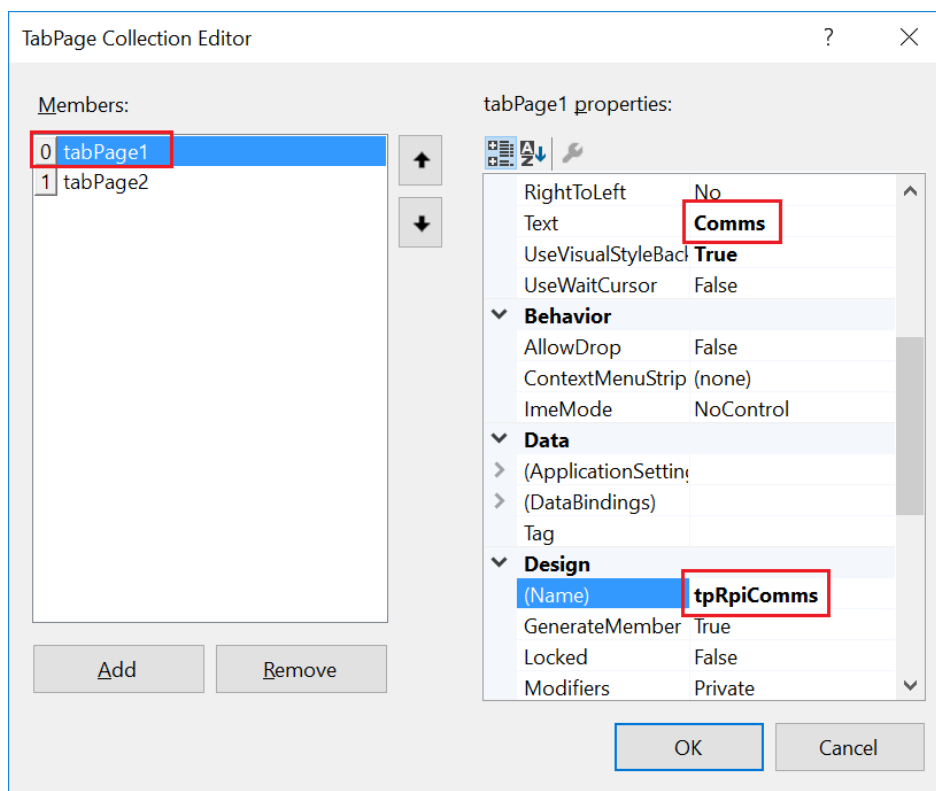
6. Click on the TabControl on the form to select it. (Note not to select it's tab Page instead)



- At it's properties, change the Name and size as follows and Edit the Tab Pages properties by selecting TabPages as shown

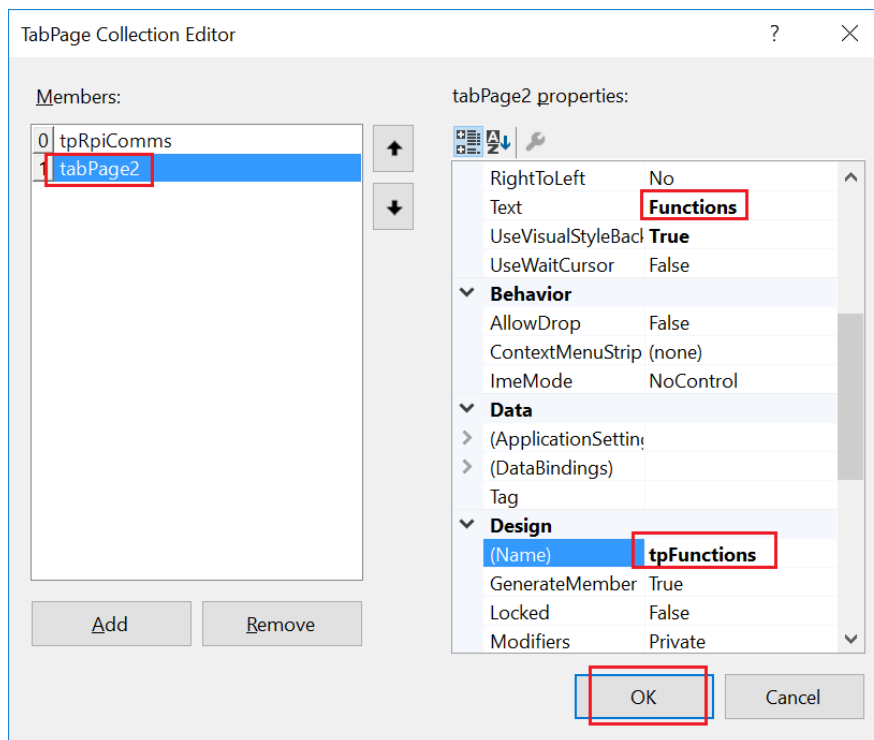


- Name first Tab Page as `tpRpiComms` and the text as `Comms`



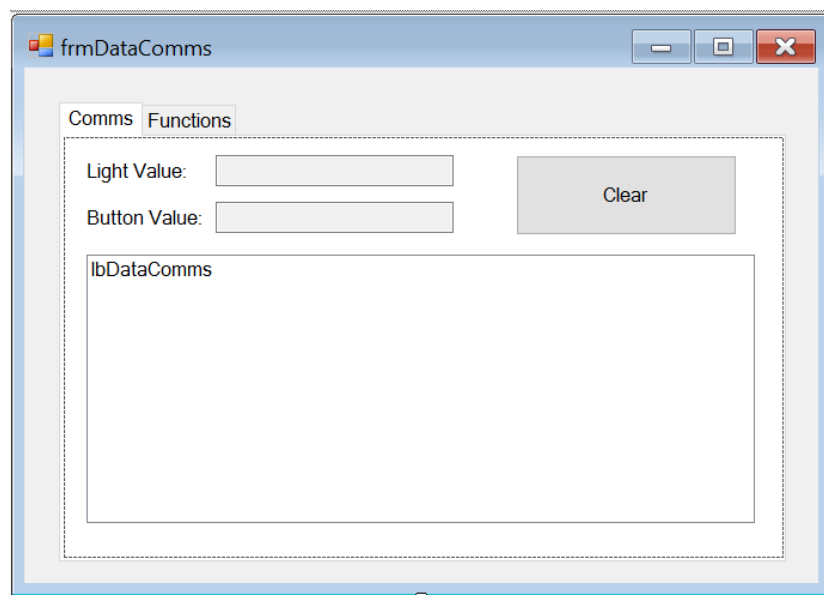


## 9. Continue to change properties of Tab Page 2



## 10. Add in the following components to the “Comms” Page

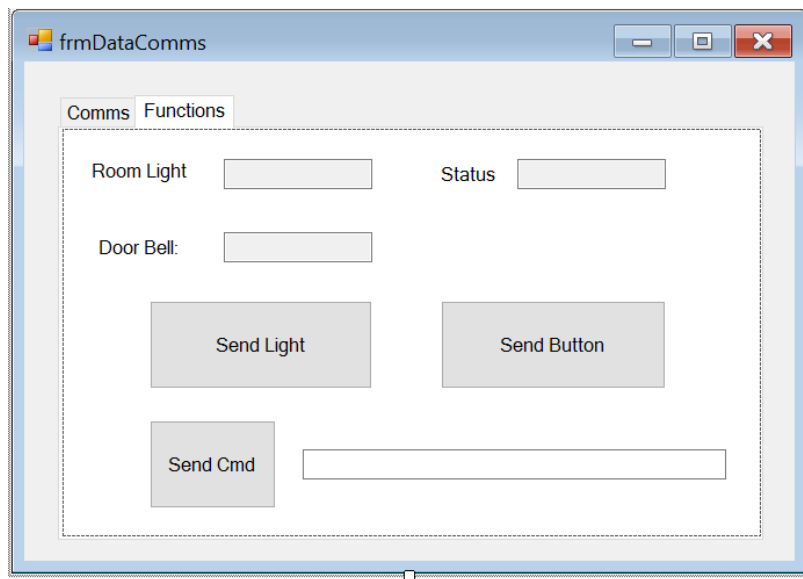
(Labels are excluded for brevity)



| Control | Name          | Text  | Other properties                   |
|---------|---------------|-------|------------------------------------|
| TextBox | tbLightValue  |       | <b>ReadOnly</b> set to <b>True</b> |
| TextBox | tbButtonValue |       | <b>ReadOnly</b> set to <b>True</b> |
| Listbox | lbDataComms   |       |                                    |
| Button  | btnClear      | Clear |                                    |

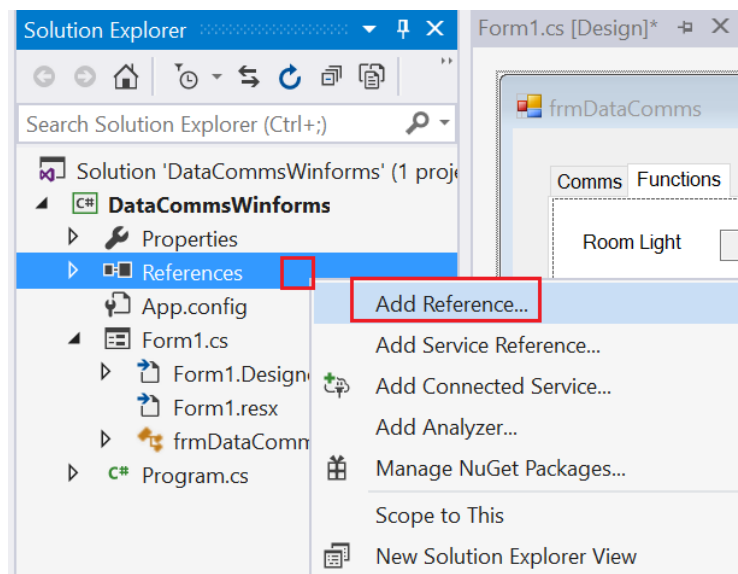
## 11. Add in the following components to the “Functions” Page

(Labels are excluded for brevity)

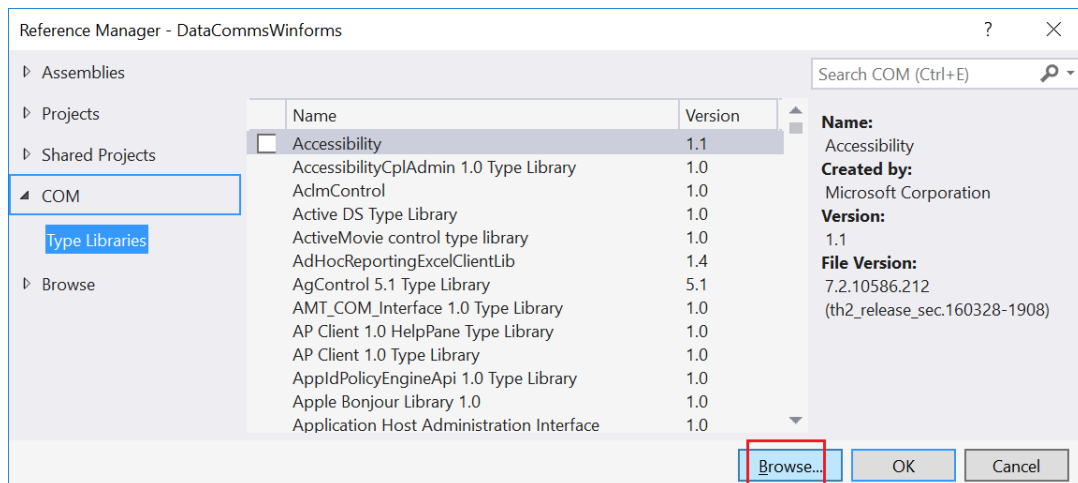


| Control | Name          | Text        | Other properties                   |
|---------|---------------|-------------|------------------------------------|
| TextBox | tbRoomLight   |             | <b>ReadOnly</b> set to <b>True</b> |
| TextBox | tbRoomStatus  |             | <b>ReadOnly</b> set to <b>True</b> |
| TextBox | tbDoorBell    |             | <b>ReadOnly</b> set to <b>True</b> |
| Button  | btnSendLight  | Send Light  |                                    |
| Button  | btnSendButton | Send Button |                                    |
| Button  | btnCmdTest    | Send Cmd    |                                    |
| TextBox | tbCmdTest     |             |                                    |

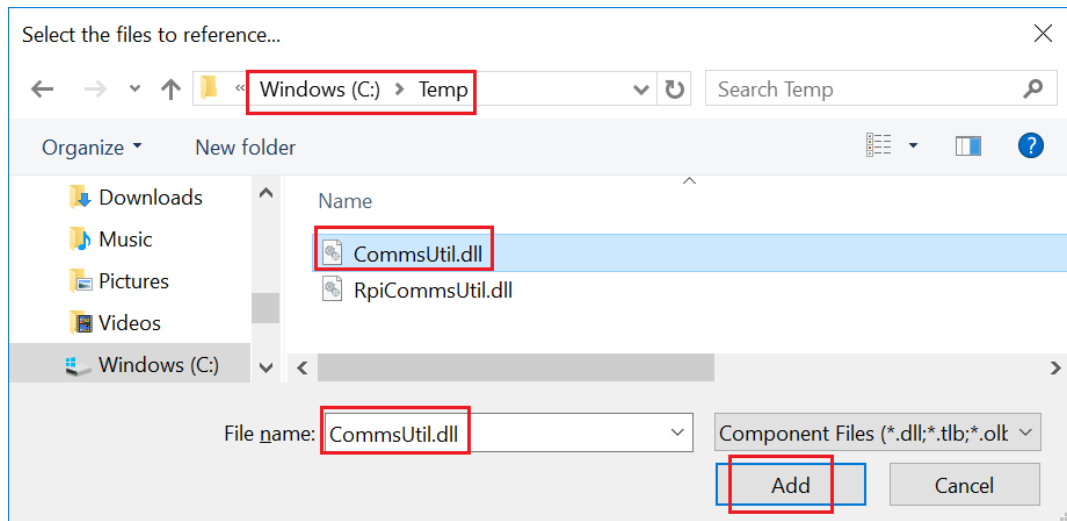
## 12. Add **CommsUtil.dll** to your workspace. Right Click on References on Solution Explorer and Click on Add Reference.



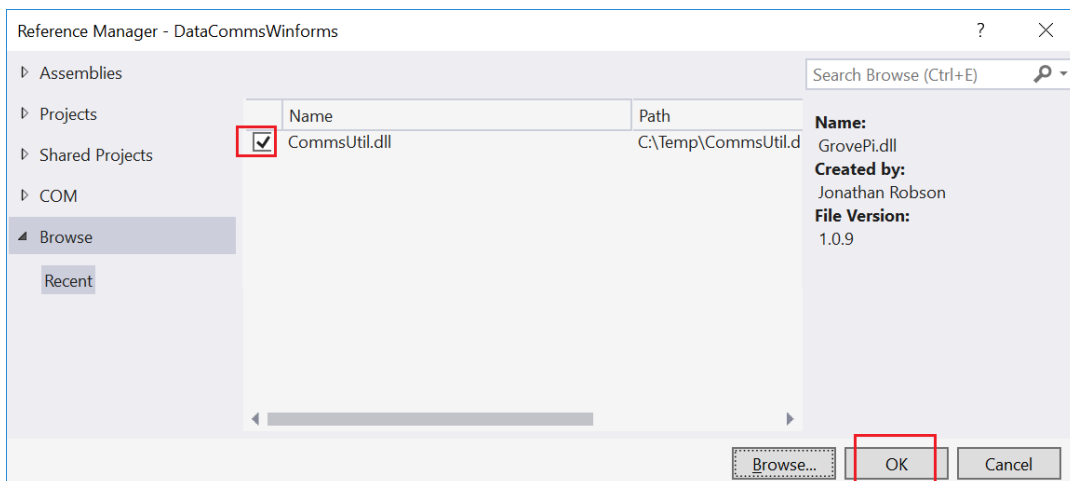
Click on **Browse** button



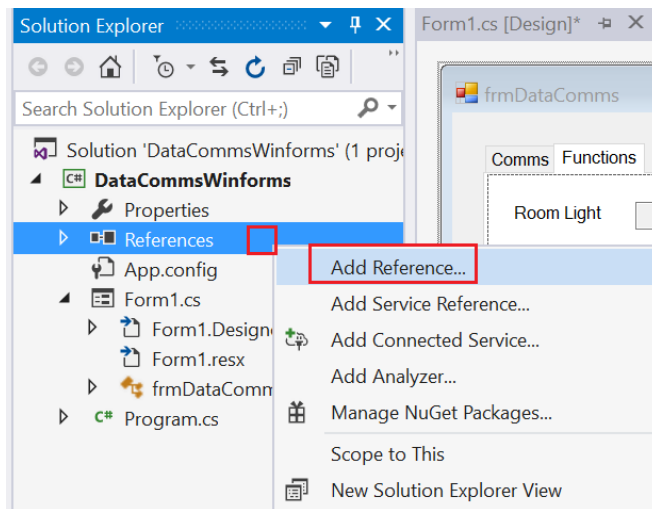
Select the **CommsUtil.dll** that you've downloaded just now to **C:\Temp** and click **Add**



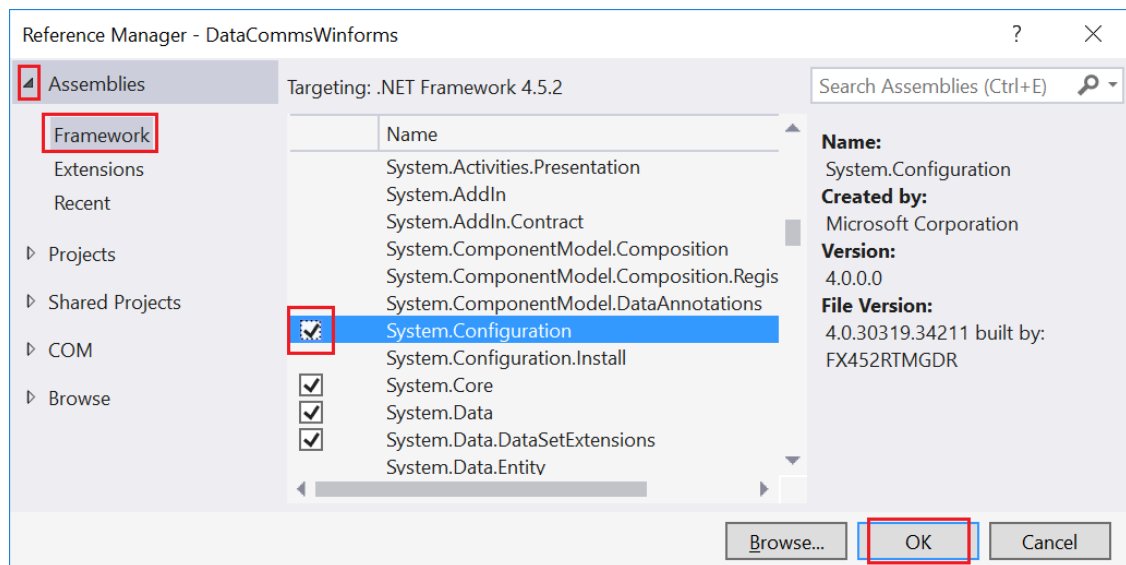
Verify the CommsUtil.dll is checked and click **OK**.



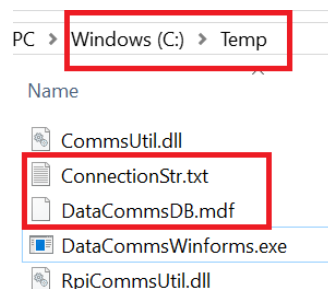
13. As we're going to add codes to write to database as well, we need to include `System.Configuration` into the workspace as well. Right click to Add Reference



Select **Framework**, checked **System.Configuration** and click **OK**.



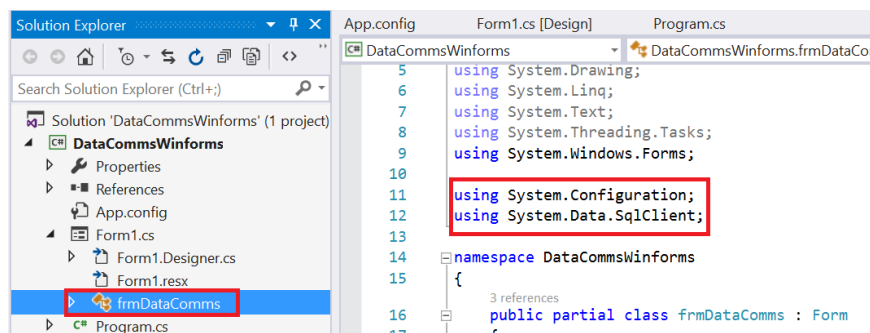
14. Checked that from the previous helper files you've downloaded, the **DataCommsDB.mdf** is in `C:\Temp` and open up **ConnectionStr.txt**



15. At your Solution Explorer, double click on the App.config. Replace the content inside with the content from **ConnectionStr.txt**. Your App.config should look like this after you've replaced the content.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="DataCommsDBConnection"
      connectionString="Data Source=(LocalDB)\MSSQLLocalDB; AttachDbFilename=C:\temp\DataCommsDB.mdf;
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

16. Click on the frmDataComms to show the codes. Type in the codes as highlighted.



17. Add the following codes, Comments are added to help you understand

```
public partial class frmDataComms : Form
{
    string strConnectionString =
        ConfigurationManager.ConnectionStrings["DataCommsDBConnection"].ConnectionString;

    DataComms dataComms;

    public delegate void myprocessDataDelegate(String strData);

    //To save sensor data to DB, you need to change to suite your project needs
    private void saveLightSensorDataToDB(string strTime, string strlightValue, string strStatus)
    {
        //Step 1: Create connection
        SqlConnection myConnect = new SqlConnection(strConnectionString);

        //Step 2: Create command
        String strCommandText =
            "INSERT MySensor (TimeOccurred, SensorValue, SensorStatus) " +
            " VALUES (@time, @value, @status)";

        SqlCommand updateCmd = new SqlCommand(strCommandText, myConnect);
        updateCmd.Parameters.AddWithValue("@time", strTime);
        updateCmd.Parameters.AddWithValue("@value", strlightValue);
        updateCmd.Parameters.AddWithValue("@status", strStatus);

        //Step 3: Open Connection
        myConnect.Open();

        //Step 4: ExecuteCommand
        int result = updateCmd.ExecuteNonQuery();

        //Step 5: Close connection
        myConnect.Close();
    } //End saveLightSensorDataToDB
}
```

```
//utility method, you should not need to edit this
private string extractStringValue(string strData, string ID)
{
    string result = strData.Substring(strData.IndexOf(ID) + ID.Length);
    return result;
}

//utility method, you should not need to edit this
private float extractFloatValue(string strData, string ID)
{
    return (float.Parse(extractStringValue(strData, ID)));
}

//create your own data handler for your project needs
private void handleLightSensorData(string strData, string strTime, string ID)
{
    string strlightValue = extractStringValue(strData, ID);

    //update GUI component in any tabs
    tbLightValue.Text = strlightValue;
    tbRoomLight.Text = strlightValue;

    float fLightValue = extractFloatValue(strData, ID);
    string status = "";
    if (fLightValue <= 500)
        status = "Dark";
    else
        status = "Bright";
    tbRoomStatus.Text = status;

    //update database
    saveLightSensorDataToDB(strTime, strlightValue, status);
}

//create your own data handler for your project needs
private void handleButtonData(string strData, string strTime, string ID)
{
    string strbuttonValue = extractStringValue(strData, ID);

    //update GUI component in any tabs
    tbButtonValue.Text = strbuttonValue;
    tbDoorBell.Text = strbuttonValue;

    //do your own update of database
}

//You need to edit here to suite your project needs
private void extractSensorData(string strData, string strTime)
{
    //Any type of data may be send over by hardware
    //so you need t always check what data is received
    //before extracting the information

    //check whether Light value is send
    if (strData.IndexOf("LIGHT=") != -1)
        handleLightSensorData(strData, strTime, "LIGHT=");
    else if (strData.IndexOf("BUTTON=") != -1) //check button status
        handleButtonData(strData, strTime, "BUTTON=");
}
```

```
//Raw data received from Hardware comes here
public void handleSensorData(String strData)
{
    string dt = DateTime.Now.ToString("s"); //get current time
    extractSensorData(strData, dt); //get sensor values out

    //update raw data received to listbox
    string strMessage = dt + ":" + strData;
    lbDataComms.Items.Insert(0, strMessage);
}

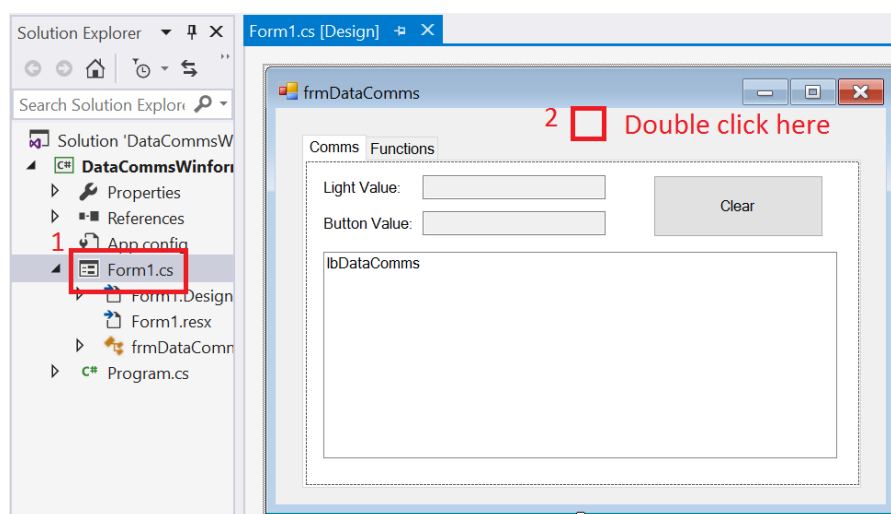
//This method is automatically called when data is received
public void processDataReceive(String strData)
{
    myprocessDataDelegate d = new myprocessDataDelegate(handleSensorData);
    lbDataComms.Invoke(d, new object[] { strData });
}

//This method is automatically called when data is received
public void commsDataReceive(string dataReceived)
{
    processDataReceive(dataReceived);
}

//This method is automatically called when there is error
public void commsSendError(string errMsg)
{
    MessageBox.Show(errMsg);
    processDataReceive(errMsg);
}

//This method must be called right at the start for data communications
private void InitComms()
{
    dataComms = new DataComms();
    dataComms.dataReceiveEvent += new DataComms.DataReceivedDelegate(commsDataReceive);
    dataComms.dataSendErrorEvent += new DataComms.DataSendErrorDelegate(commsSendError);
}
```

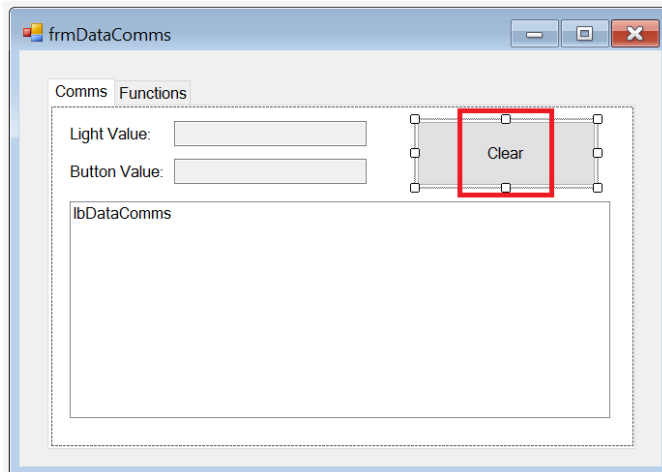
18. Double click on Form1.cs to see the form and double click anywhere on the form to create the frmDataComms\_Load() method



19. Add the highlighted codes to the frmDataComms\_Load() created by the system.

```
private void frmDataComms_Load(object sender, EventArgs e)
{
    InitComms();
}
```

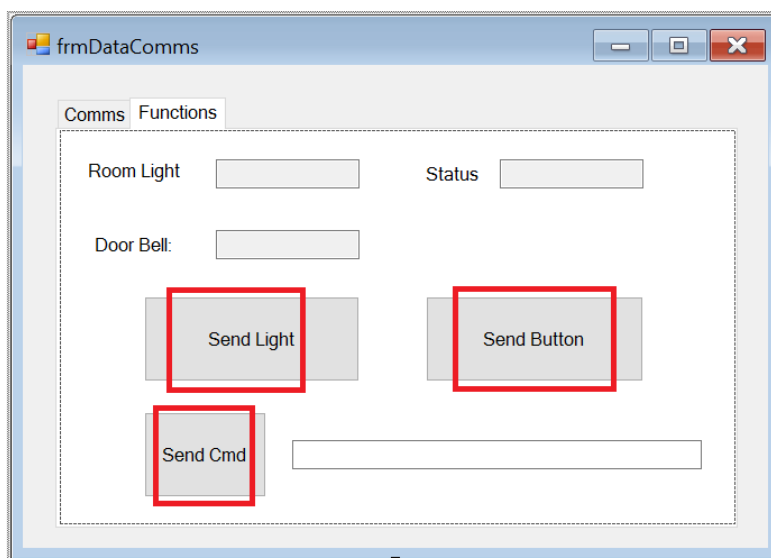
20. At the “Comms” Tab, double click on the btnClear to create it’s click handler



21. Add the codes to clear the listbox when btnClear is clicked

```
private void btnClear_Click(object sender, EventArgs e)
{
    lbDataComms.Items.Clear();
}
```

22. At the “Functions” Tab, double click on the 3 buttons to create all it’s 3 button click handlers





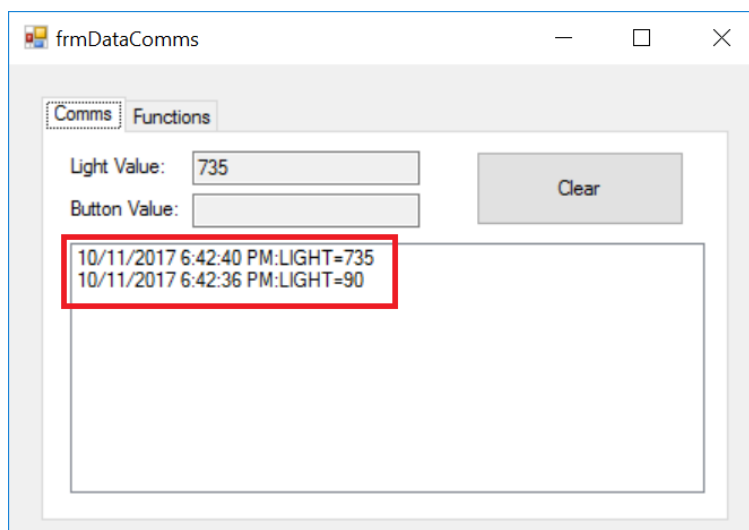
23. Add the codes for them to send data over from windows form to Raspberry Pi

```
private void btnSendLight_Click(object sender, EventArgs e)
{
    dataComms.sendData("SENDLIGHT");
}

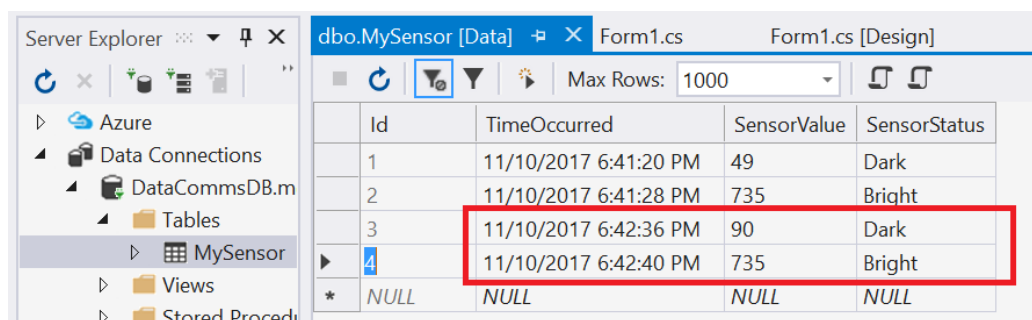
private void btnSendButton_Click(object sender, EventArgs e)
{
    dataComms.sendData("SENDERBUTTON");
}

private void btnCmdTest_Click(object sender, EventArgs e)
{
    dataComms.sendData(tbCmdTest.Text);
}
```

24. You've completed the coding of the form. Run the windows form by pressing F5. Also run your Raspberry Pi program created earlier. Try cover the light sensor with your hand to simulate a dark room light condition. Then remove your hand to simulate a bright room light condition. You should see data send over to your windows form.

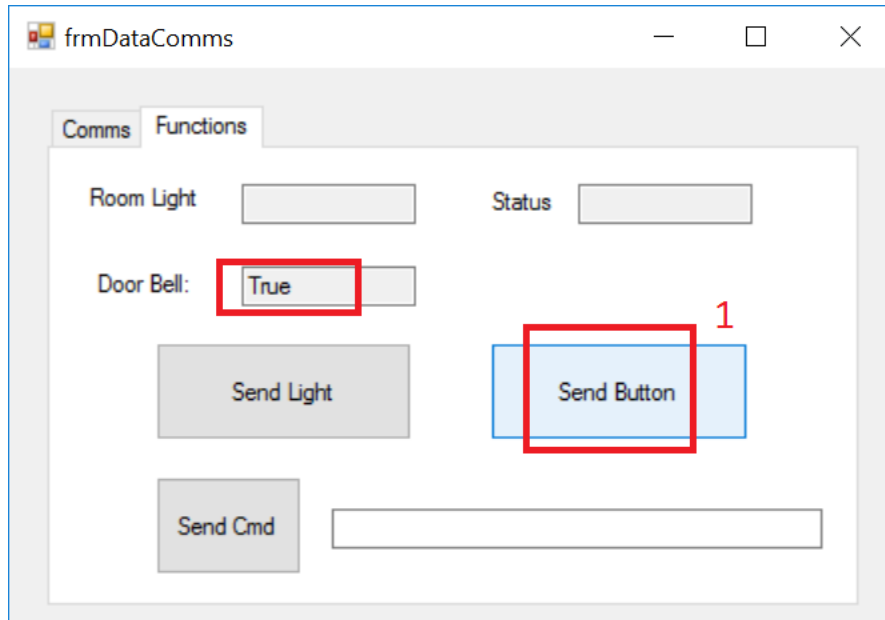


These data are also saved onto the C:\Temp\DataCommsDB.mdf. Use the Server Explorer to verify that data has been saved into the database.



| Id | TimeOccurred          | SensorValue | SensorStatus |
|----|-----------------------|-------------|--------------|
| 1  | 11/10/2017 6:41:20 PM | 49          | Dark         |
| 2  | 11/10/2017 6:41:28 PM | 735         | Bright       |
| 3  | 11/10/2017 6:42:36 PM | 90          | Dark         |
| 4  | 11/10/2017 6:42:40 PM | 735         | Bright       |
| *  | NULL                  | NULL        | NULL         |

25. You should also try out the Send Button command and then press the push button at the Raspberry Pi and verify the button status is sent and display in the Windows Form.
- (**Note:** You won't find the button status in database as no code is added in this lab for that but you could always refer to the codes for the Light Sensor value to add it on your own)



In this Lab, you've learnt how to

- Send data from Raspberry Pi to Windows Form
- At raspberry Pi, receive data from Windows form and process it
- Send data from Windows Form to Raspberry Pi
- At Windows form, receive data from Raspberry Pi and save it onto database

The things you learn in this practical is very important as all project will definitely need it. It is thus very important for you to understand this lab and know where and how to edit the codes to suit your purpose.

**==End of Practical==**