**ECE 323 Lab 10 Serial Communication Using C# GUI**

**Developed by Manish Ale Magar, and edited by Dr. Zheng in fall 2019**

1. **Objectives**

This is a tutorial for developing a graphic user interface (GUI) program using C# to establish serial connection between a PC and a microcontroller using a Bluetooth module. You will learn to use various tools/components to create a GUI, the architecture of object-oriented programming, the creation of various events, and event handling in C#. Your final GUI should be similar to the following picture:

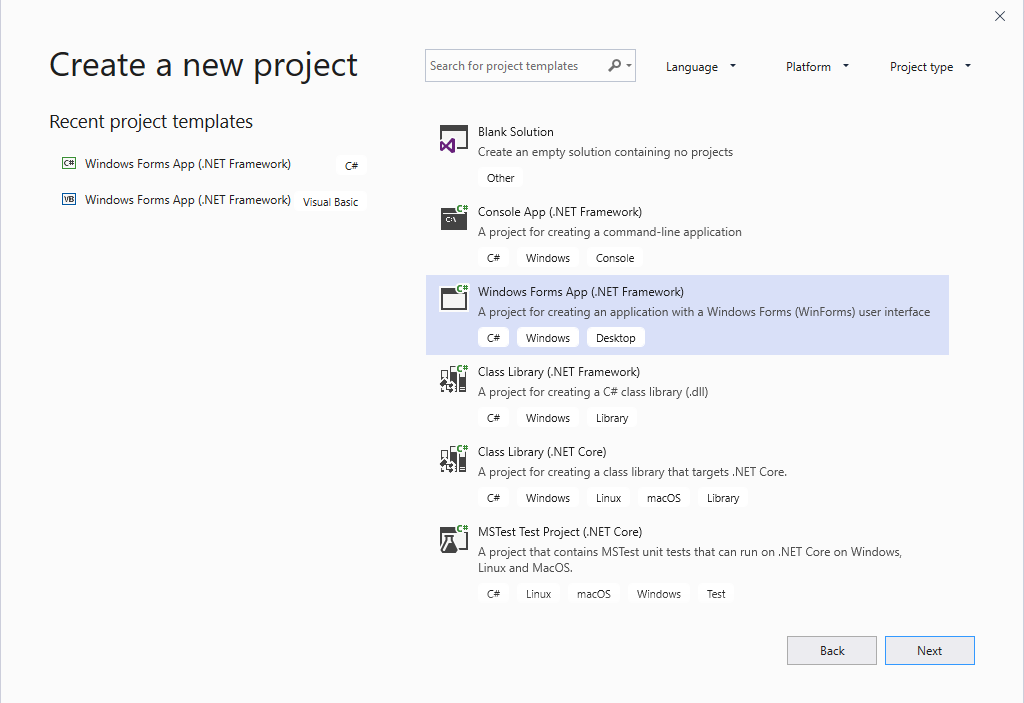
|  |  |
| --- | --- |
|  |  |

* There will be a rich text box used to send and receive characters just like in Tera Terminal.
* There is a drop-down menu to change our Bluetooth connection settings.
* It will send a character immediately after it is typed in the text box.
* For this tutorial, **Visual Studio 2019** is used which can be downloaded from the following website: <https://visualstudio.microsoft.com/downloads/>

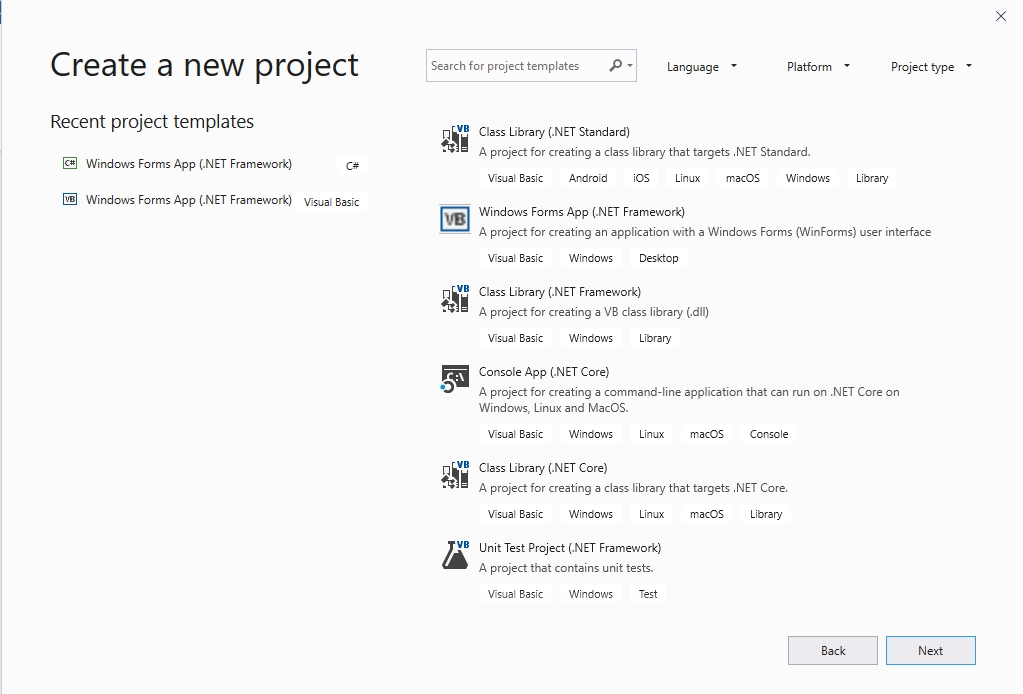
Choose **Community** version as it is free to use.

1. **Start the project with a basic GUI design:**

After successfully installing the software on your computer, **open Visual Studio and create a new Project.** We will choose **Windows Forms App** for this tutorial.

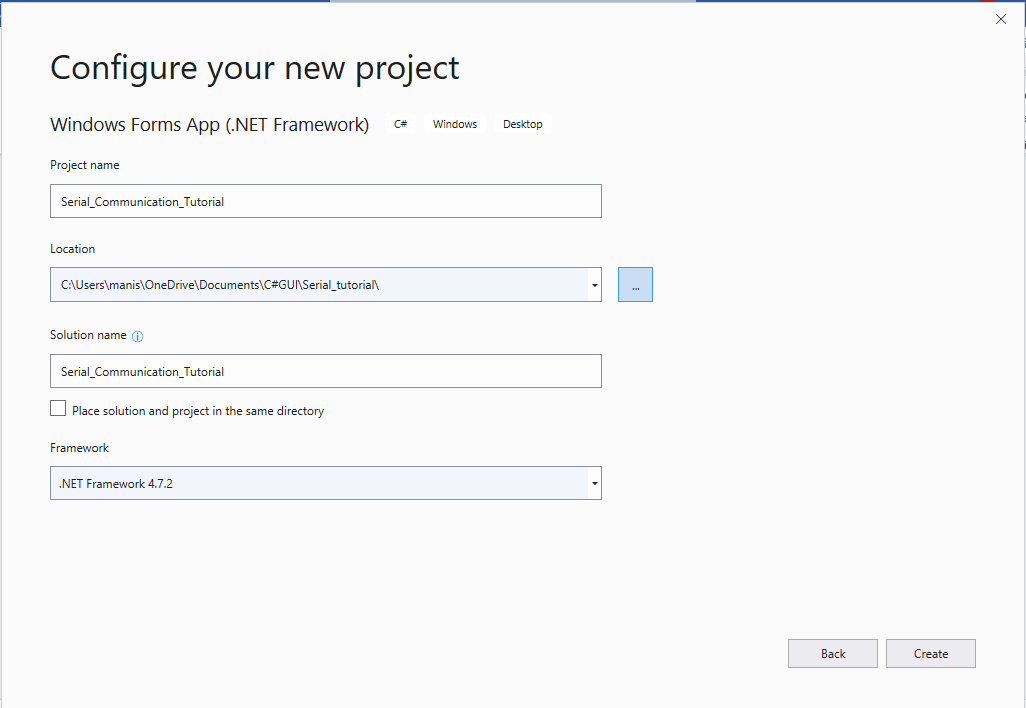


However, we need to be careful about this part because there is another version of Windows Forms Application which looks like as follows:



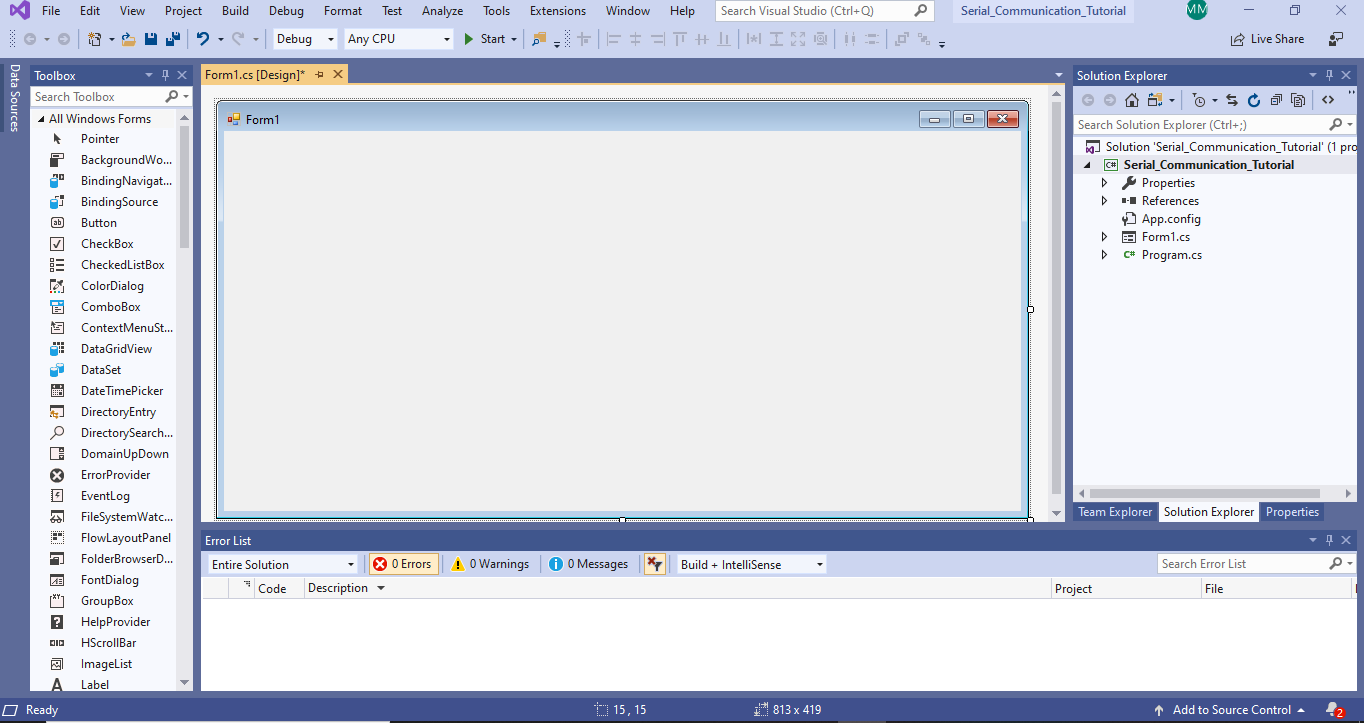
You don’t want to choose this as your project. It is for Visual basic and contains .vb extension. We want our project to be in .cs extension.

Give your project a file name and choose a directory where you want to store your project.



Toolbox

After you create the project, a new window will open which looks as follows:



Run your Program

Project Hierarchy

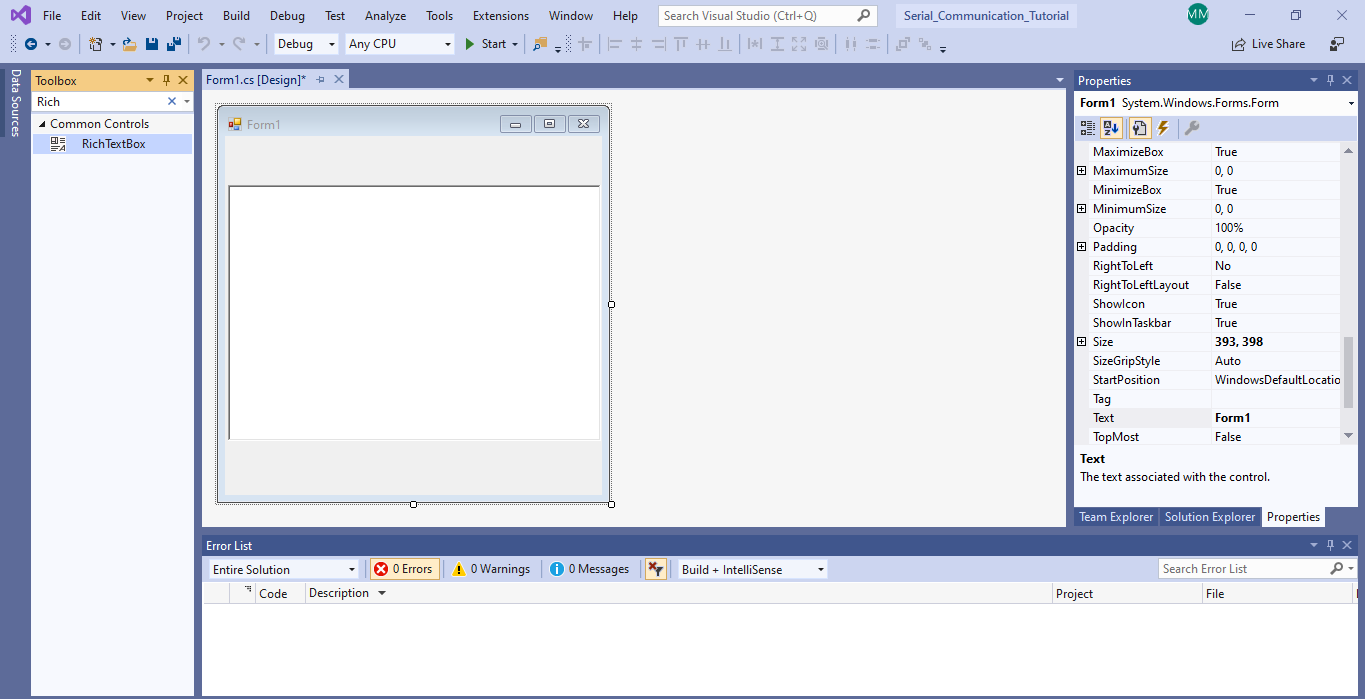
Before we add any components to our GUI, let’s get familiar with the environment:

On the left hand side, you will see the solution explorer.

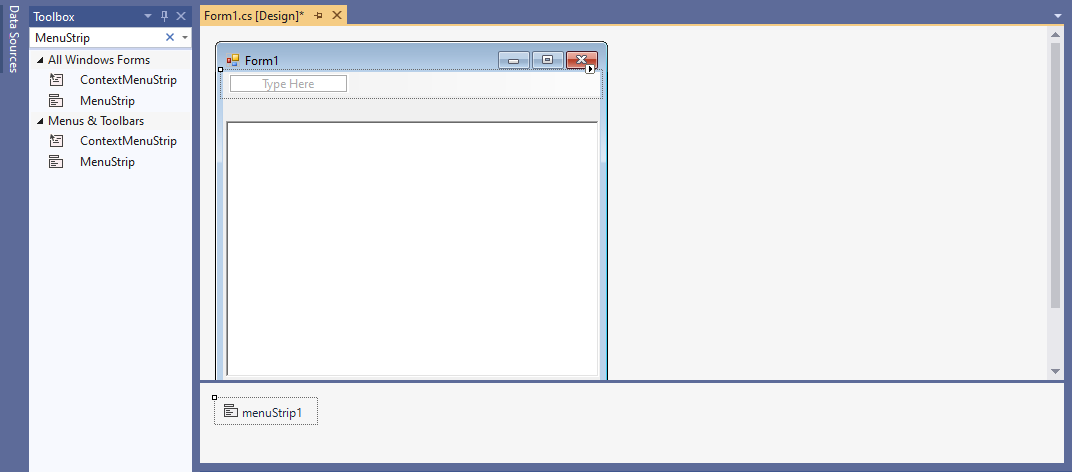
On the right hand side, it shows the properties box by click the bottom row in the solution box.

|  |  |
| --- | --- |
|  | Event Creator |
| **Solution Explorer** lets you view, navigate, and manage your code files. **Solution Explorer** can help organize your code by grouping the files into solutions and projects. It also allows us to add/import references if needed. | **Properties tab** allows us to change various properties of our component including its name. Moreover, we can create various events for each component using this tab. |

Now, let’s start by adding a rich text box into our main Form. You can navigate to toolbox and search for **RichTextBox**. The Toolbox contains various components that can be **dragged and dropped** onto the form. Hover you mouse over each one to gain information about that component.



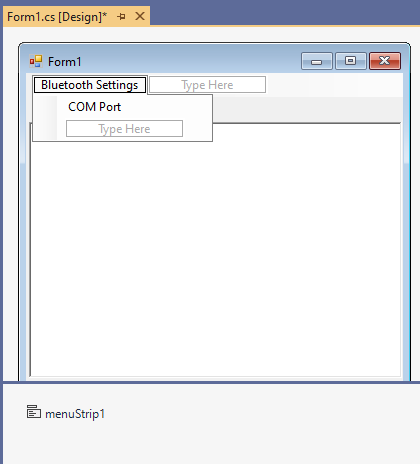
Now, select **MenuStrip** from toolbox and add it onto the top of our form as shown below:



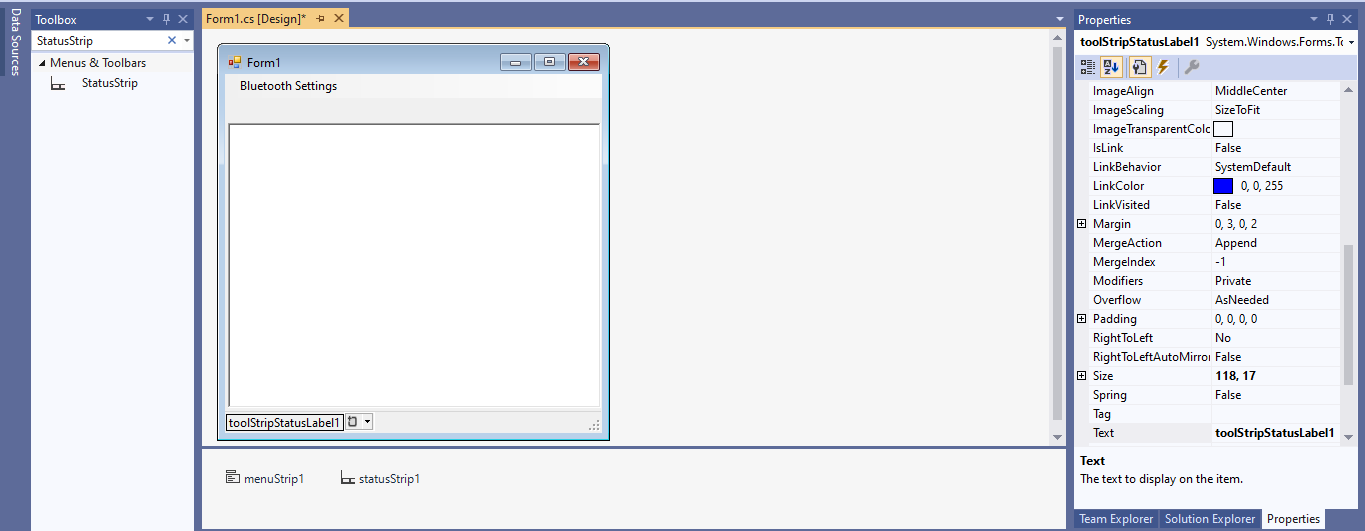
You can type the name for the menu in this box

Enter **Bluetooth Settings**.

A sub menu will be shown, enter **COM Port** in the sub menu as shown below:



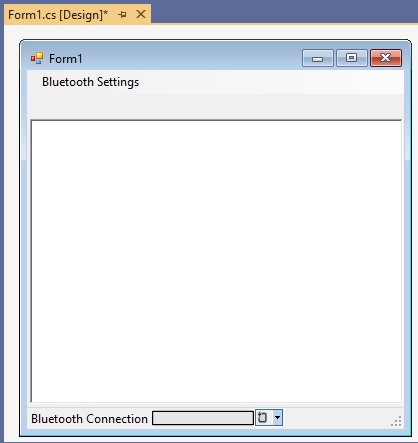
Now, select a **StatusStrip** from the toolbox and add it onto the bottom of our Form as shown below.



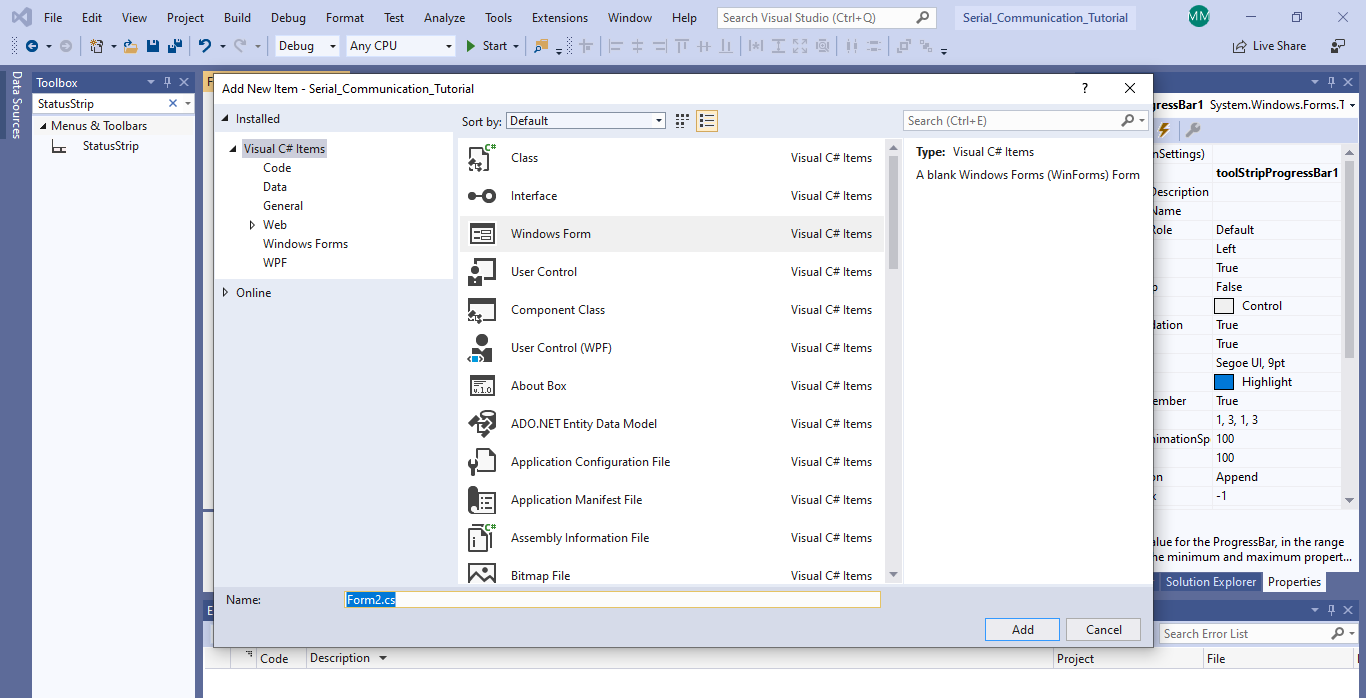
We can change the appeared text from this property tab

click on this to add a label. There are other options to choose such as a progress bar, etc.

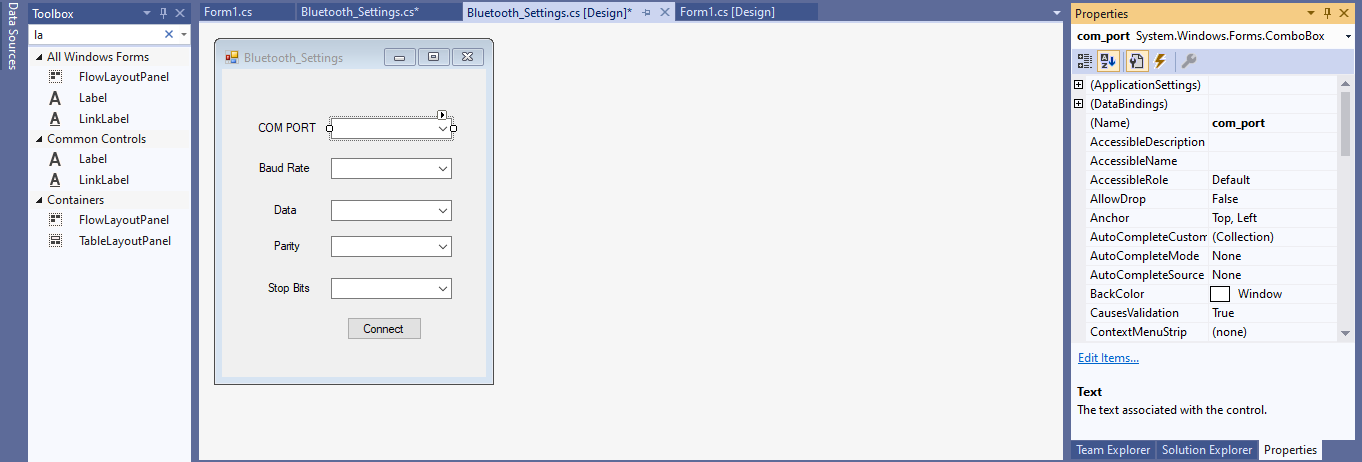
Now, quickly add a progress bar by selecting **progress bar** from drop down menu as shown below.



We are done adding all the components. Now, we will create a new form that will contain all the settings for Bluetooth Connection. Click on **Project->Add Windows Form**. The following window will pop-up.



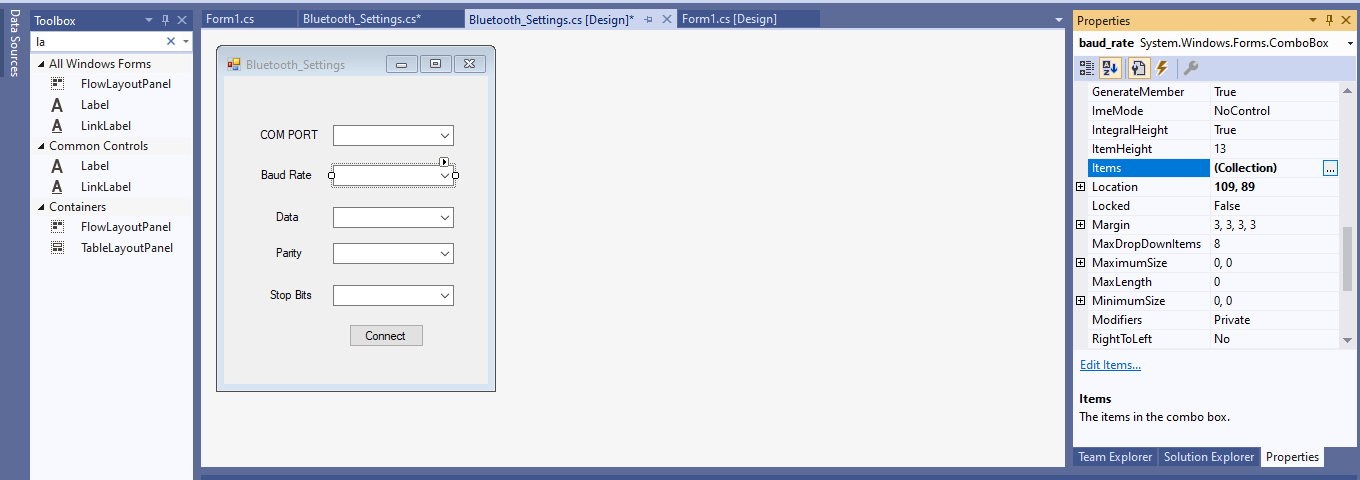
Let’s change the form name to **Bluetooth\_Settings**. Then click on **Add** to create a new form. Now **add five ComboBox, five labels and a button** as shown below:



You should change the designator using this property to label each box

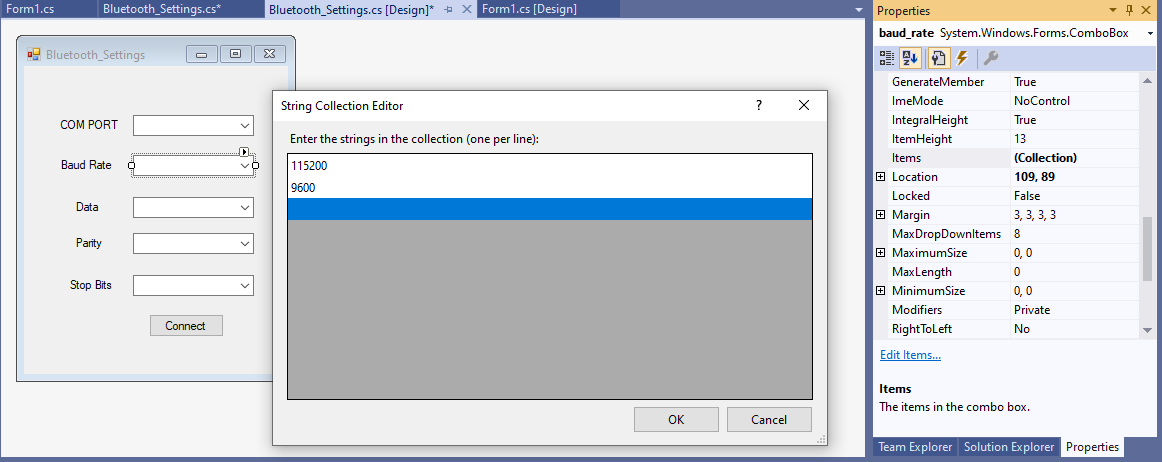
You can see that five boxes have been labeled by changing the **text** in the property tab. Also, change the **names** of ComboBox for COM PORT to **com\_port**, Baud Rate to **baud\_rate,** Data to **data**, Parity to **parity**, and Stop Bits to **stop\_bits**. It is important to have meaningful designator for each component for effective programming, as names will be used in the code. Also change the name of Connect button to **connect\_button**.

While selecting ComboBox for **Baud Rate**, navigate to the property tab and click on Items as shown below:



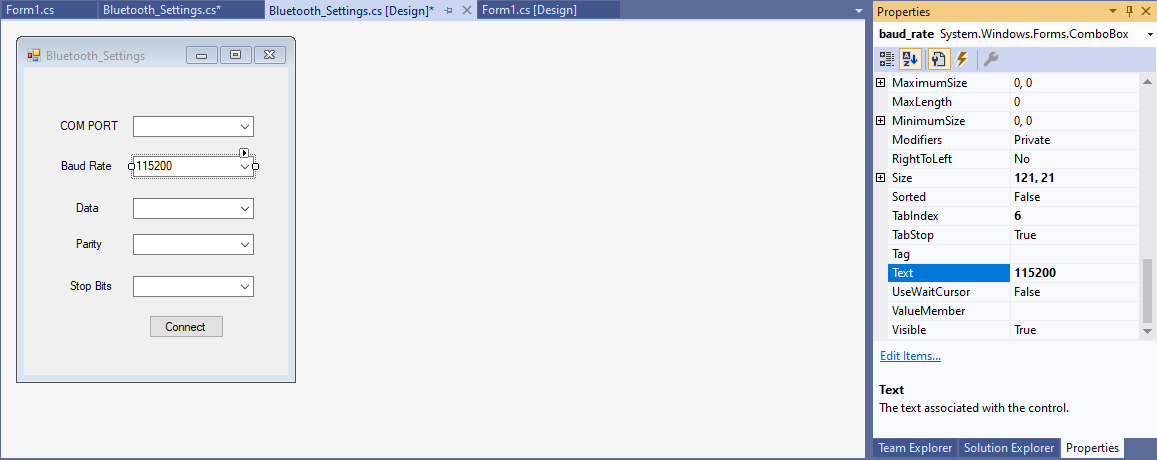
Click Here

Then add few **items** for various baud rates to the list and click on **OK** button as shown below:

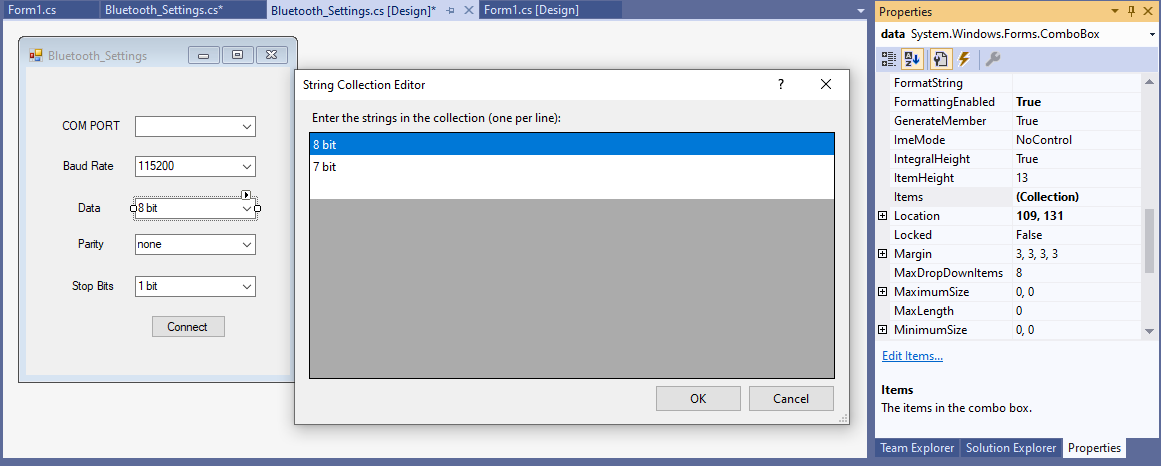


When you are done adding all the items in the list, click OK

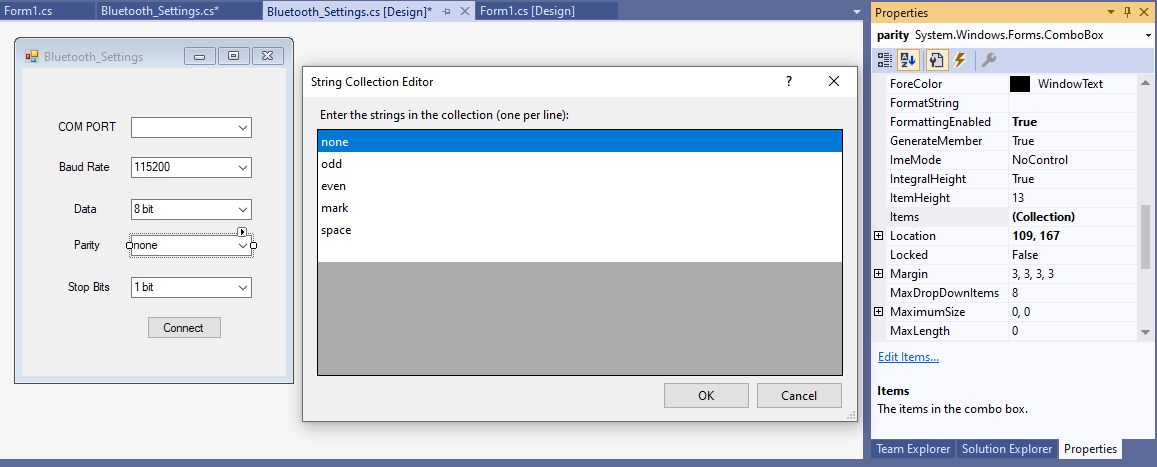
Also, while selecting the same ComboBox, go to Text Property and change it to 115200 so that it becomes the **default baud rate** for our serial connection. When you are done editing text property, your GUI should look similar to the following image.



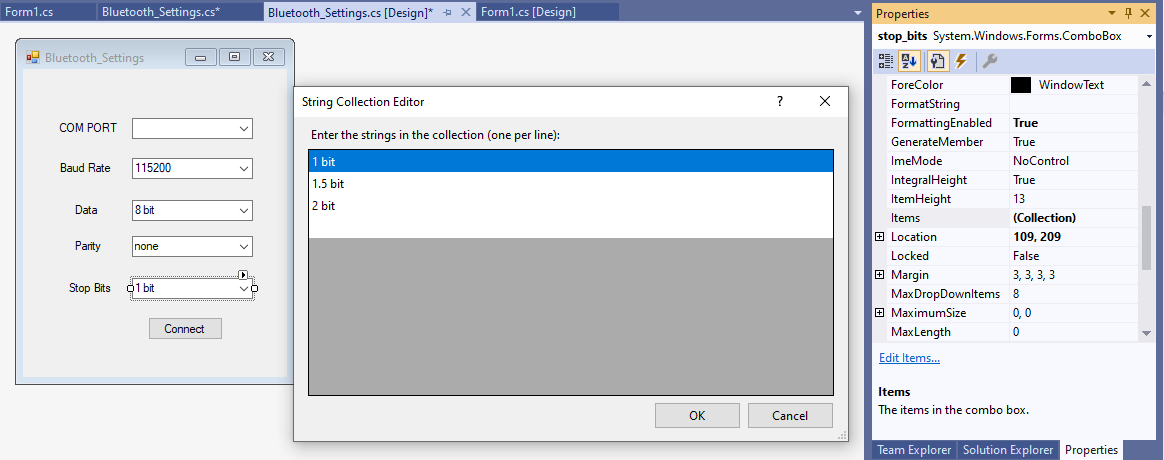
Similarly, add following items for Data as shown in image below:



Also, add following items for Parity as shown in image below:



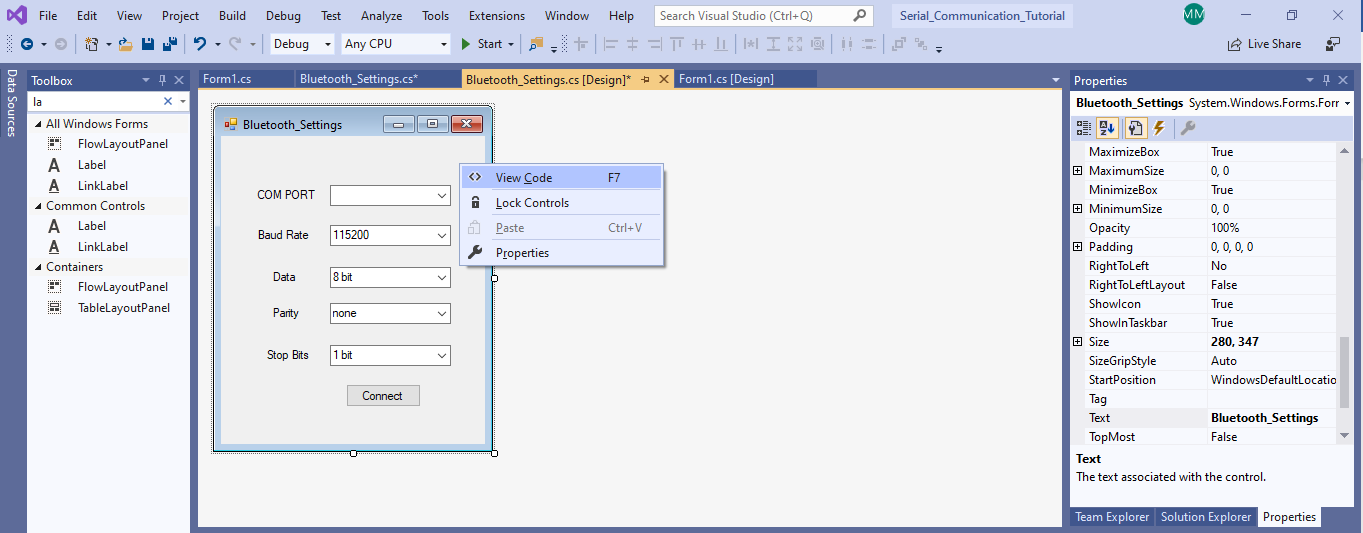
Lastly, add following items for Stop bits as shown below:



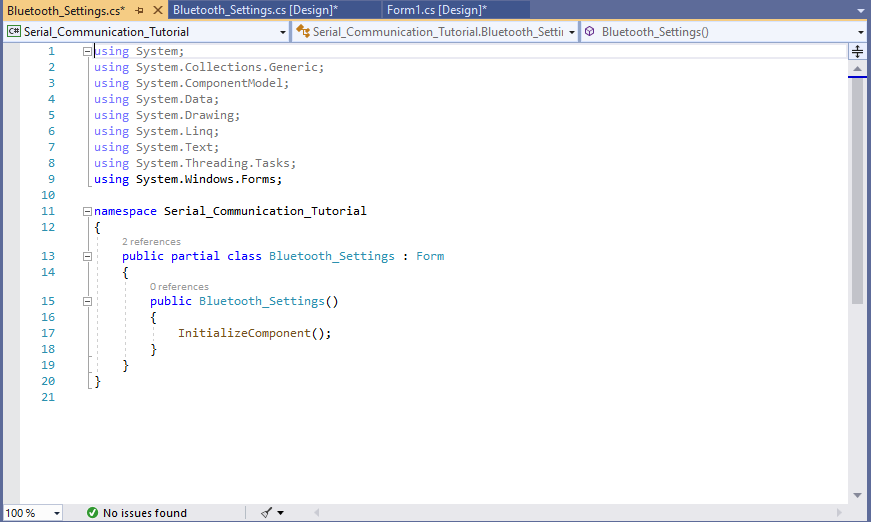
Also, change the **text property** for **each combo box** to have a **default value** for serial port connection.

**III: Code for Serial Port Connection**

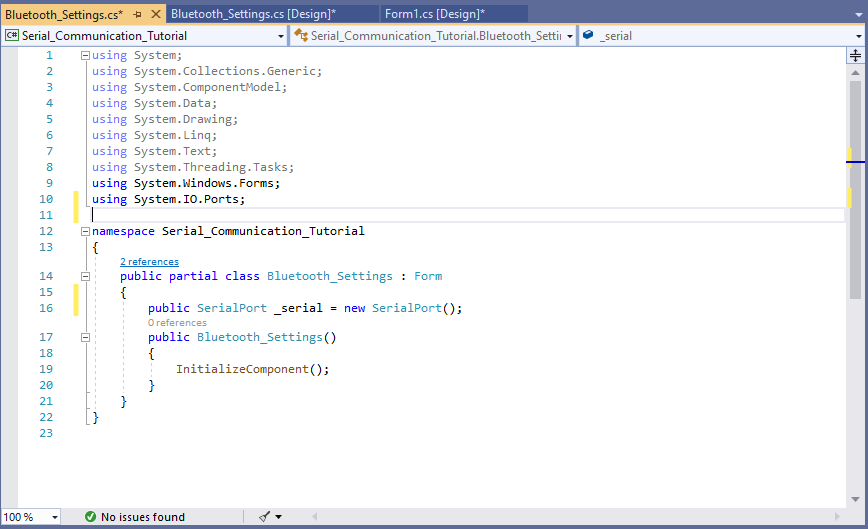
Now, we will start writing codes for our GUI. Let’s begin with the secondary form which is **Bluetooth\_Settings** (Form1 is the first form). Right click on the on the GUI and select View code as shown below:



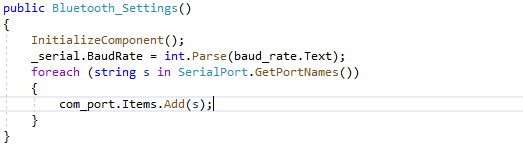
It will automatically open a new window with the initialization of our form as shown below:



Now, we need to import serial port into our form by adding (using System.IO.Ports;) at line 10, see below. This will allow us to use **Serial Port** in our program. Also, initialize an object for **SerialPort** class as shown below:

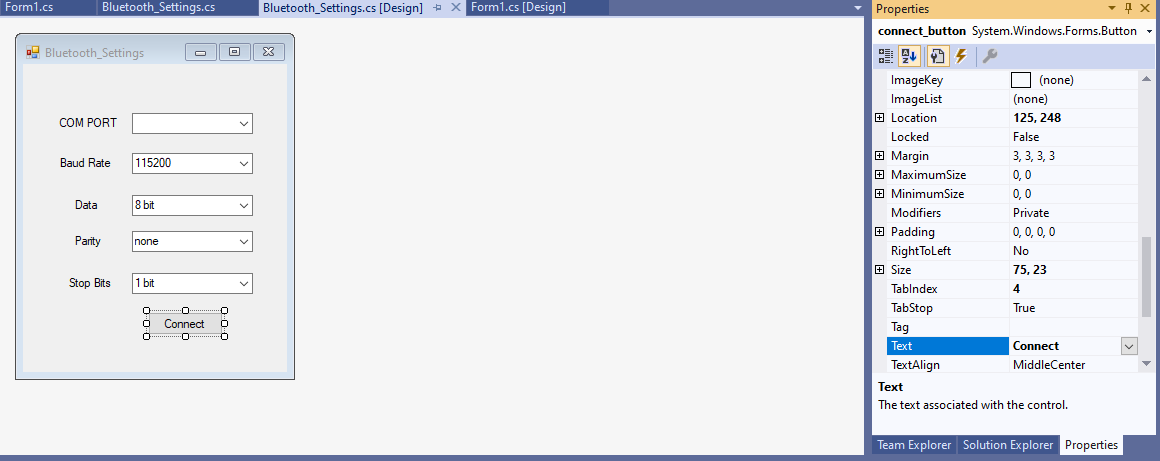


Now, change the Bluetooth\_Settings() as shown below:

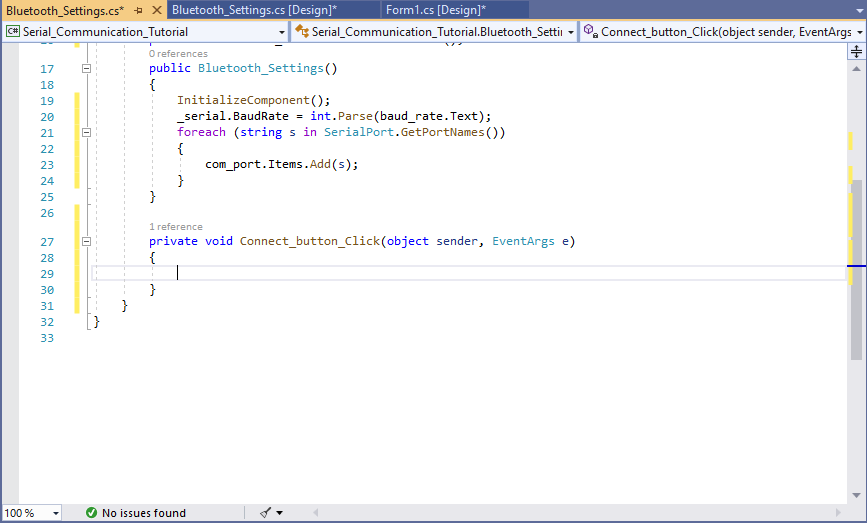


This will select the Baud Rate for Serial Port as our default value which we set in the text property for baud\_rate and display all the available COM Ports in our computer in com\_port combo box, by using a built in function, **serialPort.GetPortNames().**

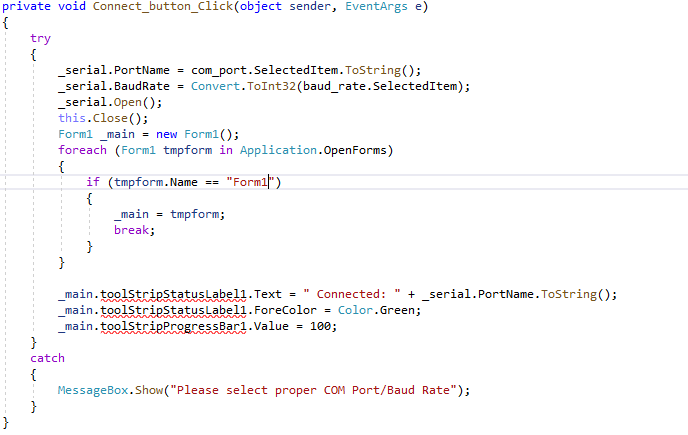
Now, let’s go back to Bluetooth\_Settings Design which is our secondary form and double click on Connect button.



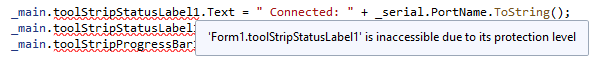
Doing so, we will create an event for Button for clicking on it meaning that while running the GUI, if we click on this button, it will create an event and call its event handler.



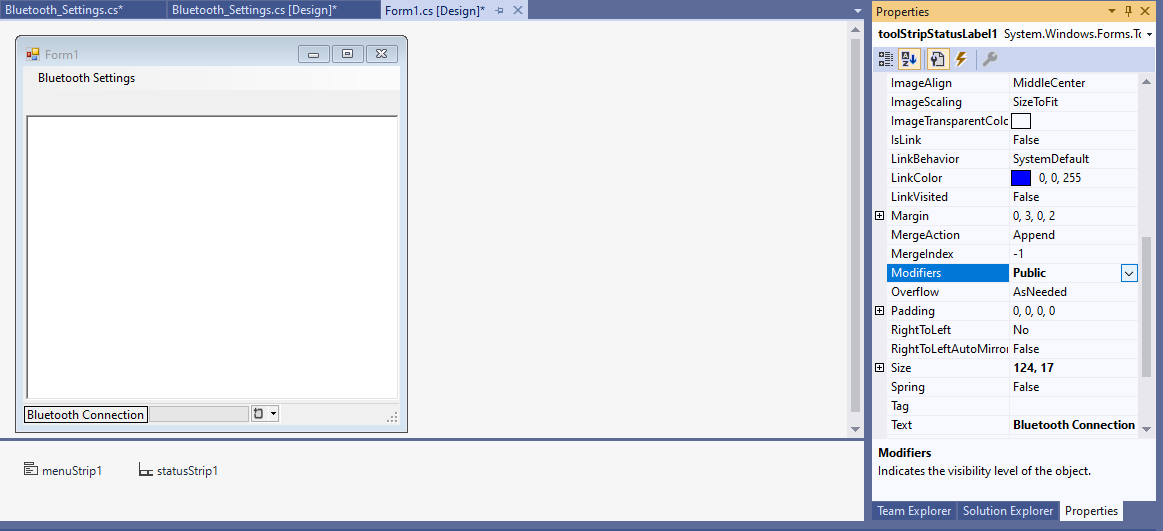
As you can see, it created a new function named Connect\_button\_Click(). This is an event handler for click event for our connect\_button. Edit this function as shown below:



Here, we can see three errors shown in red line. If you hover your mouse to that line, then it shows something like this.



We are getting this error because **toolStripStatusLabel1** and **toolStripProgressBar1** are set as private. Let’s change their modifier property from Property tab in the main form, Form1 to **public** as shown below:



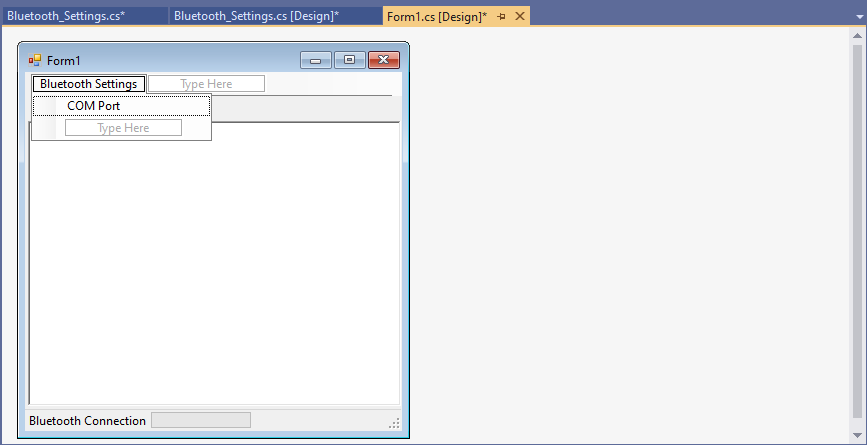
Make sure that you have changed the modifier property for **toolStripProgressBar1** as well.

After you have changed their property you can see we have no errors:



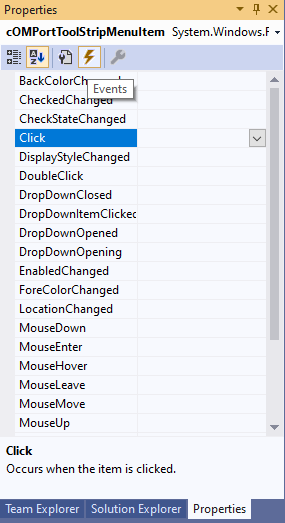
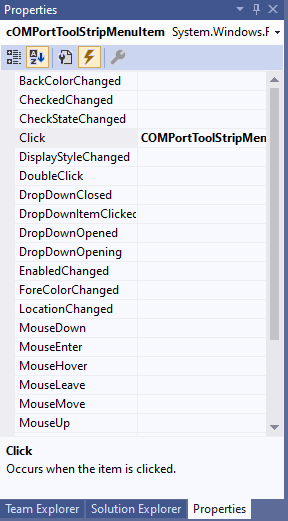
We are actually done with this secondary form.

Now, open (click) the design for Form1, click on Bluetooth Settings Menu and double click on COM Port.

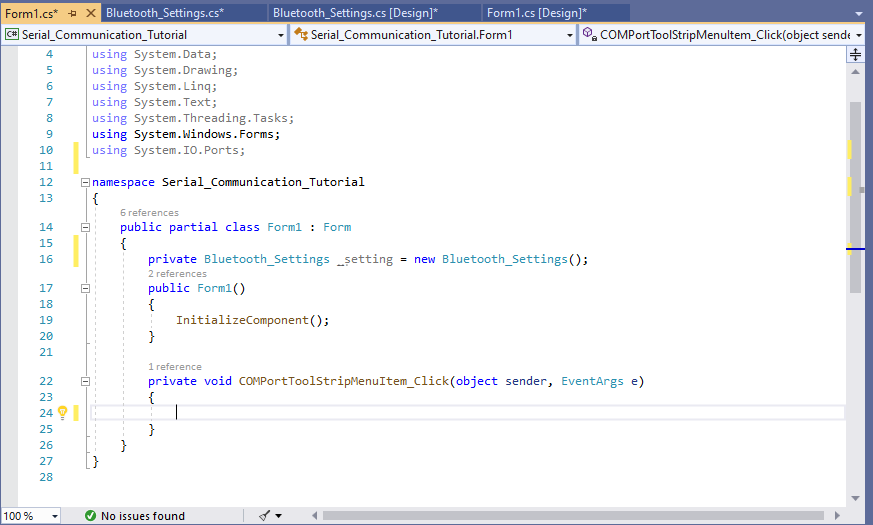


Double click here to create an event handler for click event.

Alternatively, we can create an event for any components by selecting the component, navigating to **property tab->click** on events which will be the right-most button on Property tab with lightning symbol and double click on the event that you want to create.

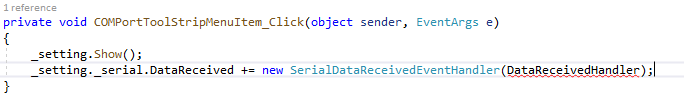
It will open the Form1.cs file and create the event handler for click event. Again, we must import Serial Port into our main form in order to send and receive data using our GUI. Also, we will instantiate an object for our secondary form i.e. **Bluetooth\_Settings** in order to open it and use serial port which will be opened from our secondary form. Change the code as shown below:



Add these two lines

**IV: Code for Receiving Data and Updating Rich Text Box**

Now, we will learn third way to create an event handler for an event. This time we will create a Data Received Handler for our serial port. Also, we will add a line of code that will open our secondary form when we click on COM Port from our menu. Change the code as shown below:

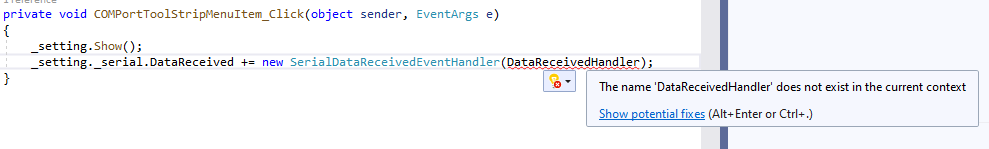


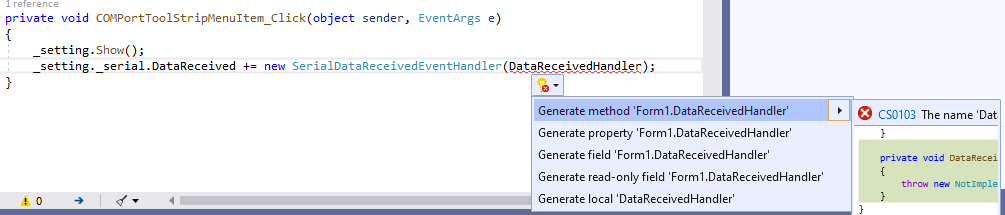
Name of Event Handler

Event

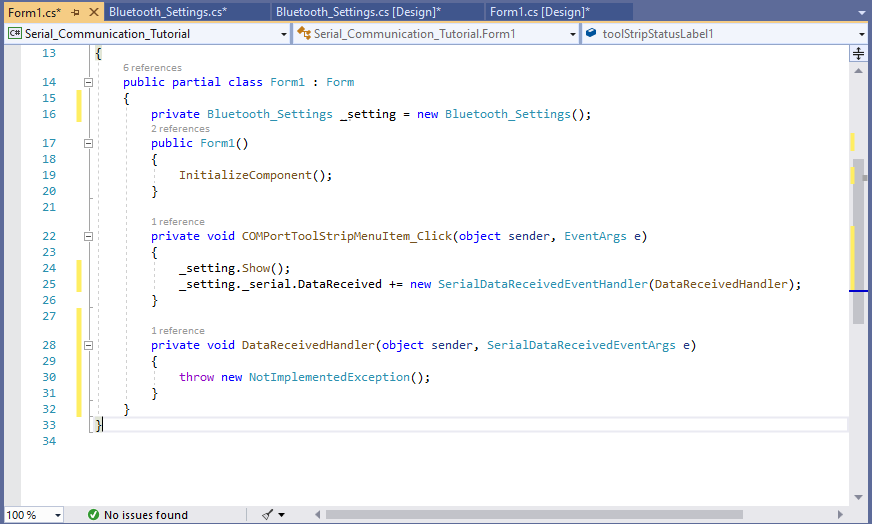
As you can see, we have an error for our name for Event Handler because we haven’t created any method having name of **DataReceivedHandler**.

In order to fix the error, hover your mouse to it and click on show potential fixes and click on Generate method (Show potential fixes).

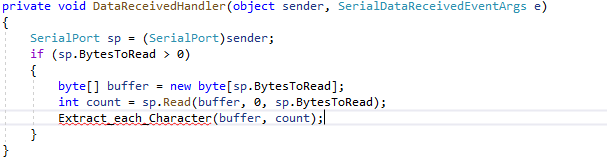




Now you have successfully created a method for your Data Received Event handler. This can be seen in the following image.



Now, let’s start with initializing an object of System.IO.Ports.SerialPort class. We need an object of SerialPort in order to use read and write functions of it so that we can send and receive data. Change your data receive handler as shown below:

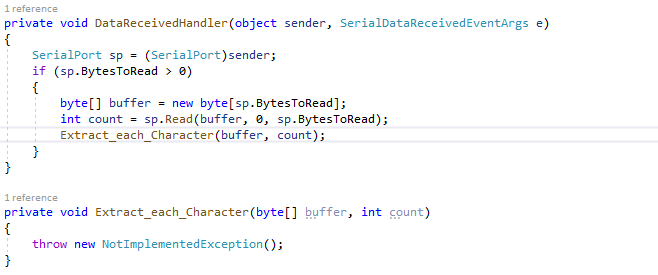


Creating a buffer to store all the incoming bytes

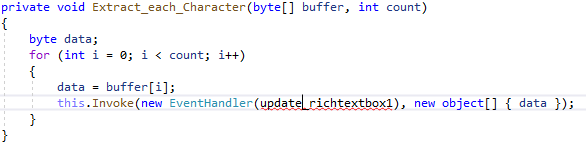
Here we are using Read() function from SerialPort class. To learn more about methods go to the following link.

<https://docs.microsoft.com/en-us/dotnet/api/system.io.ports.serialport.read?view=netframework-4.8>

As you can see, we have an error in Extract\_each\_Character(buffer, count) because we haven’t created any method of that name. You can simply create a method by hovering your mouse to it and click on show potential fixes and click on generate method. It will automatically create the method that we want to use which can be seen as below:



Now change the method that we just created as shown below:

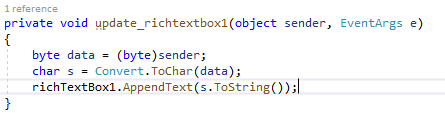


Again, do the same for this to generate new method

You might be wondering why are we using this.Invoke() method to call update\_richtextbox1(). It is because serial port runs on a different thread than our main GUI. Also, serial port is defined in our secondary form. Thus, when we try to update our main form by using data from different thread, it will lead to cross-thread call. Therefore, we must use either Delegate method or Invoke method to solve the issue of cross-thread call. To learn more about safe cross-thread call go to the following link.

<https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-make-thread-safe-calls-to-windows-forms-controls>

Now change the update\_richtextbox1() method as shown below:



Here, richTextBox1.AppendText() will append any string passed into its parameter to our RichTextBox1.

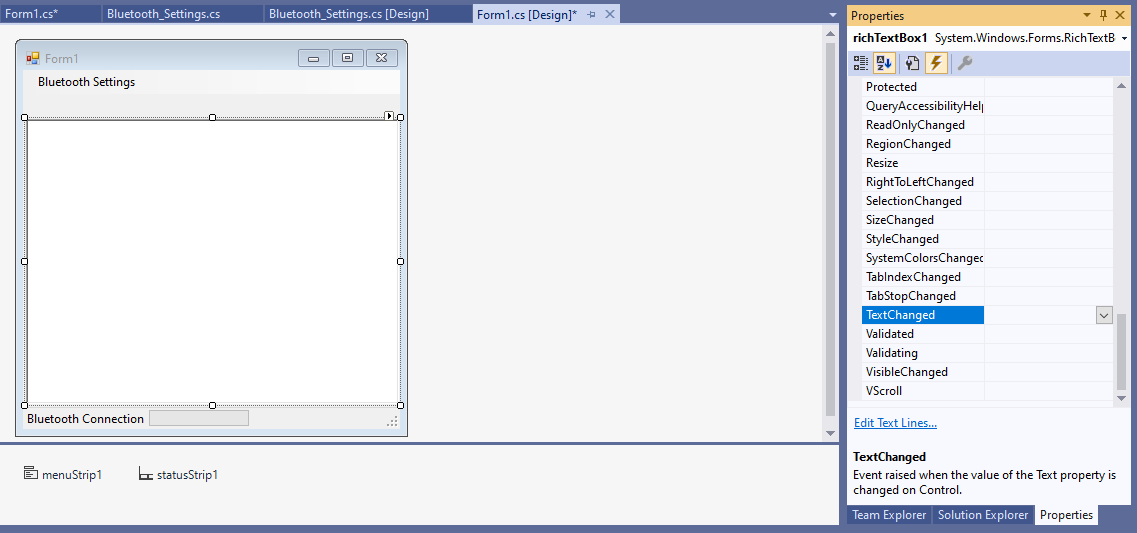
Alternatively, we can also use richTextBox1.Text += s.ToString();

Now, if you run the GUI, we should be able to receive data from the microcontroller and display it on the RichTextBox after successfully connecting our program to the Bluetooth module.

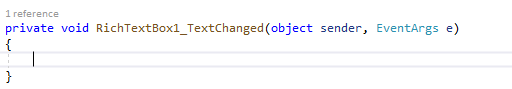
**V: Code for Sending Data**

As I mentioned earlier, we want to send data as soon as we type a character in our rich text box so that we don’t need any extra button and separate text box to send the text to the microcontroller.

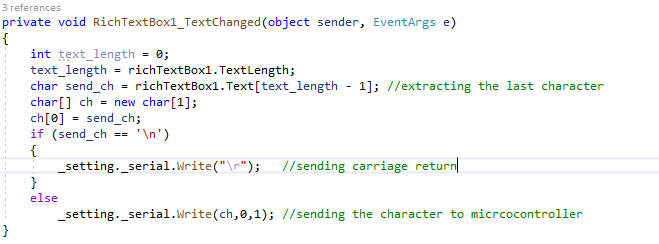
Now, go to Form1 design and select richTextBox1, navigate to Properties, click on events, and double click on Text Changed as shown below:



You might have already guessed that we are trying to create an event for our rich text box which will generate an event when we add a new character into it. Double clicking on it will generate the method for event handling as shown below:

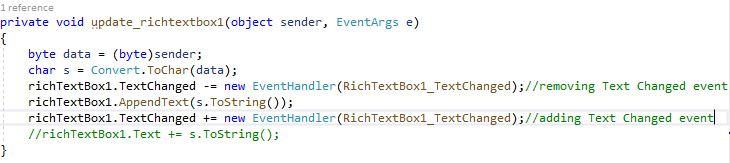


Now, let’s change it so that we can send the latest character that we just typed in.



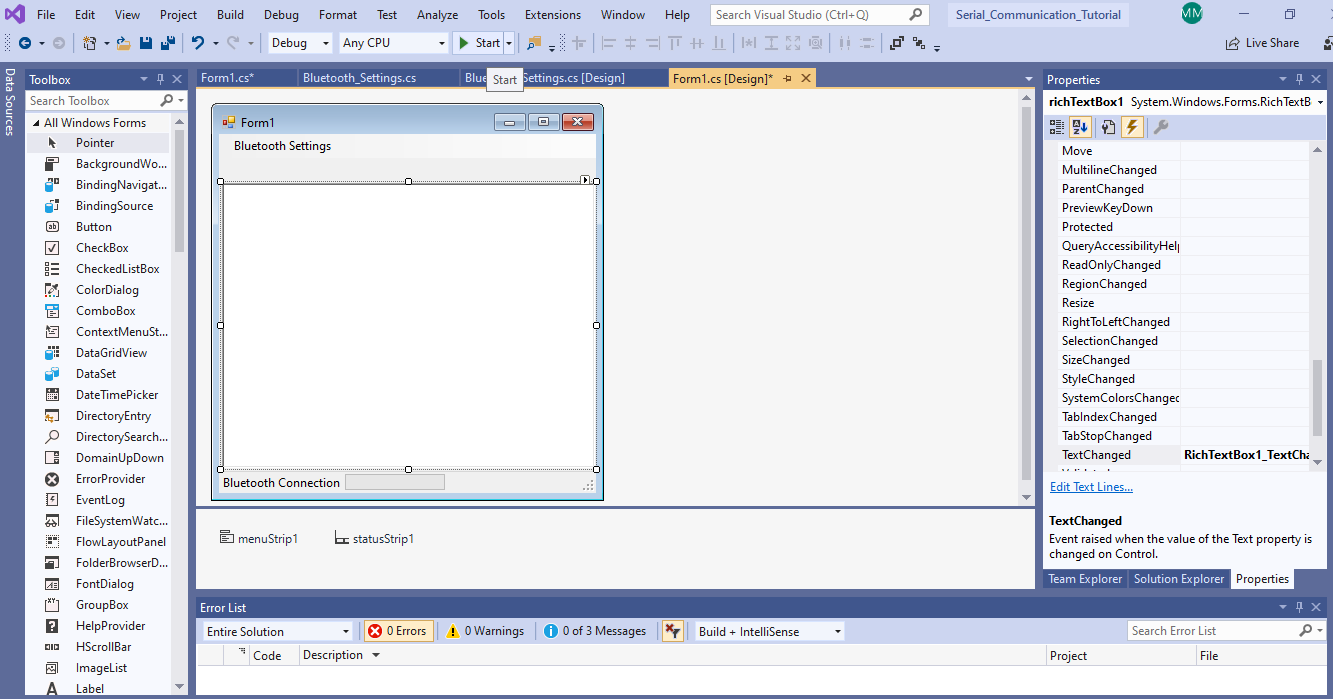
Now, we are able to send any character that we type into our **RichTextBox1**.

However, there will be an issue with using such an event. Remember that we are receiving data from our serial port and displaying it on to our RichTextBox1. Appending received data on to RichTextBox1 will also trigger the Text Changed event. But we don’t want it to trigger that event because it will unnecessarily send that character back to microcontroller. So, we will modify our update\_richTextBox1() method so that it doesn’t trigger Text Changed event when displaying characters received from Serial Port.

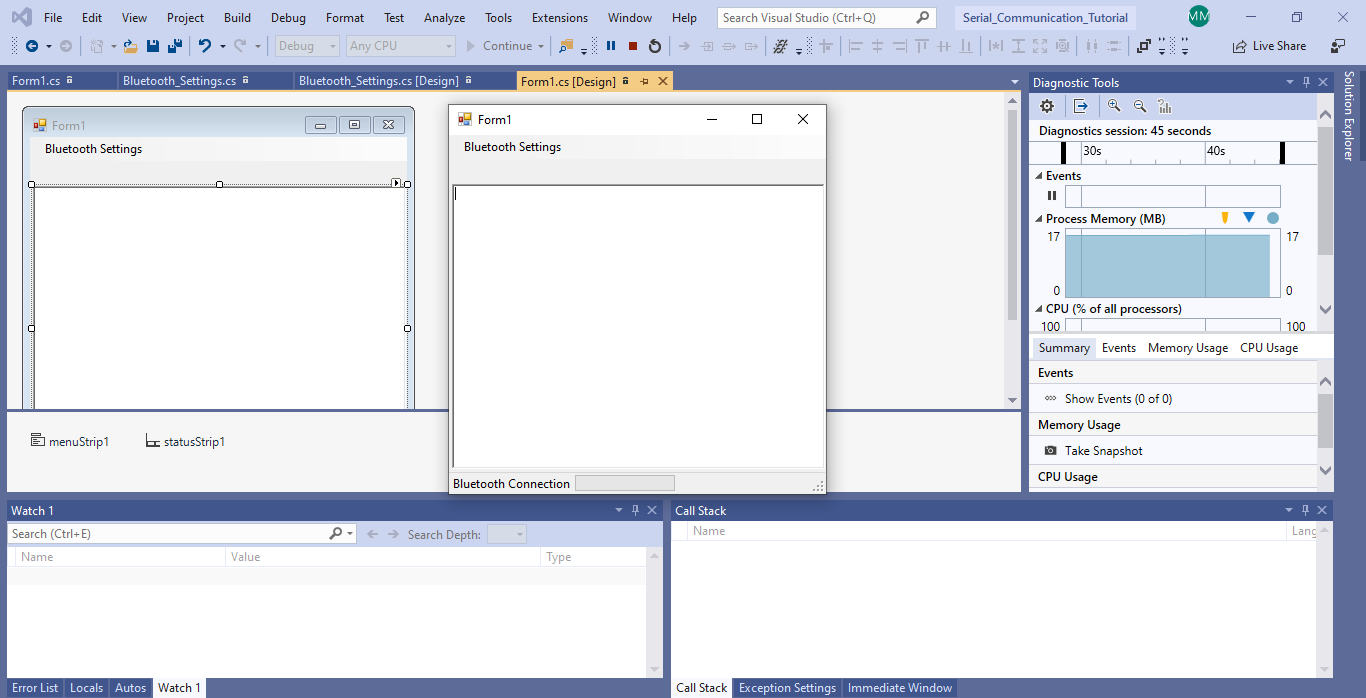


**VI: Debugging/Running your GUI**

Finally, we have successfully built a GUI program that can send and receive data through serial port. You can run your program by clicking on Start as shown below:



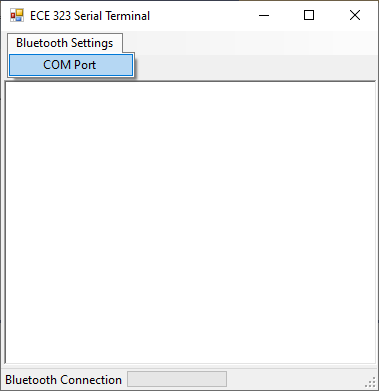
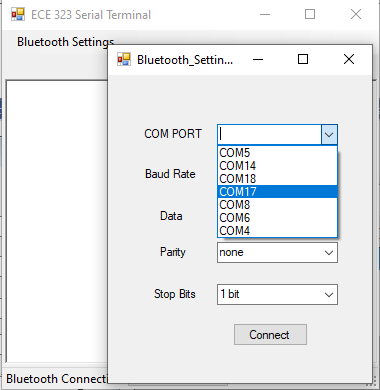
After you start running your GUI, the following window will show:



Stop

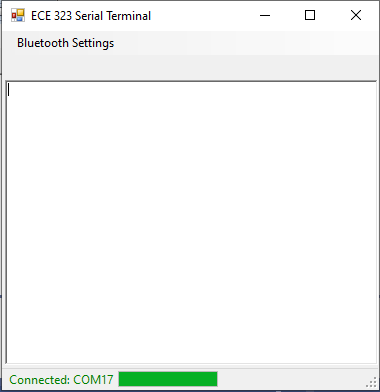
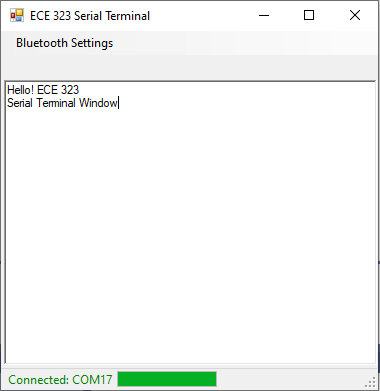
Watch Window

Now, Click on Bluetooth Settings->COM Port to open our secondary form which has settings for our Serial Port.

Then select the correct COM Port and buad rate for your Bluetooth Module and Click on Connect.

Go to Device Manager and look at your ports to determine the correct COM Port. If you have successful connection then toolStripStatusLabel1 and toolStripProgressBar1 will be changed as follows:

Now, you can just type in any character inside rich text box to send it to your microcontroller.

**Check off:** repeat Lab 9 using your serial communicaiton GUI developed in this lab. (80%)

(95%) if you can plot it.