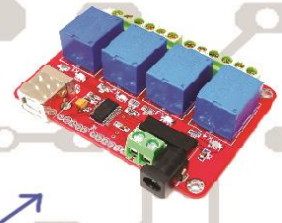
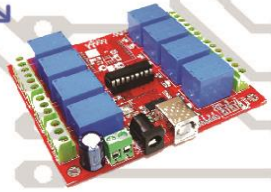
Integrating C# .Net with Embedded System





C#

.net

Build your own HMI...

- Introduction to c#.net
- Creating HMI(GUI)
- Working with controls
- Interfacing RFID reader,GSM and Relays

Research Design Lab

www.researchdesignlab.com

Email: sales@researchdesignlab.com I www.researchdesignlab.com

An ISO 9001- 2008 Certified Company

Table of Contents

OVERVIEW

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by Ecma and ISO.C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages to be used on different computer platforms and architectures.
The following reasons make C# a widely used professional language:

- Modern, general-purpose programming language.
- Object oriented.
- Component oriented.
- Easy to learn.
- Structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- Part of .Net Framework.

Getting started

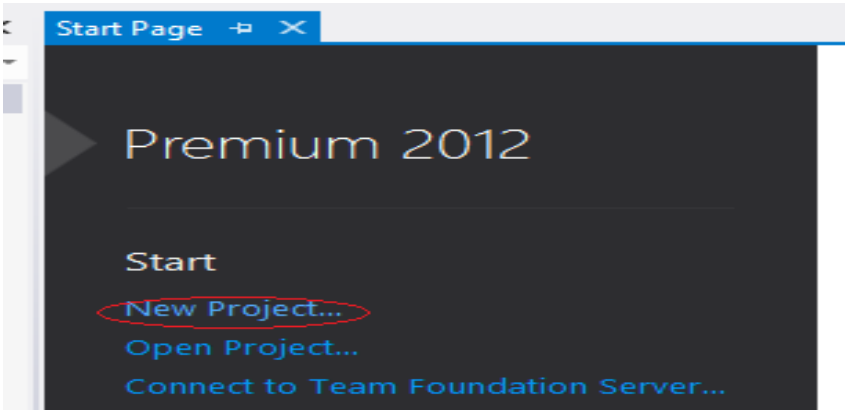
Creating a new visual studio C# project:

Once Visual Studio is running the first step is to create a new project. Do this by selecting *New Project* from the *File* menu. This will cause the *New Project* window to appear containing a range of different types of project. For the purposes of this tutorial we will be developing a *Windows Forms Application* so make sure that this option is selected.

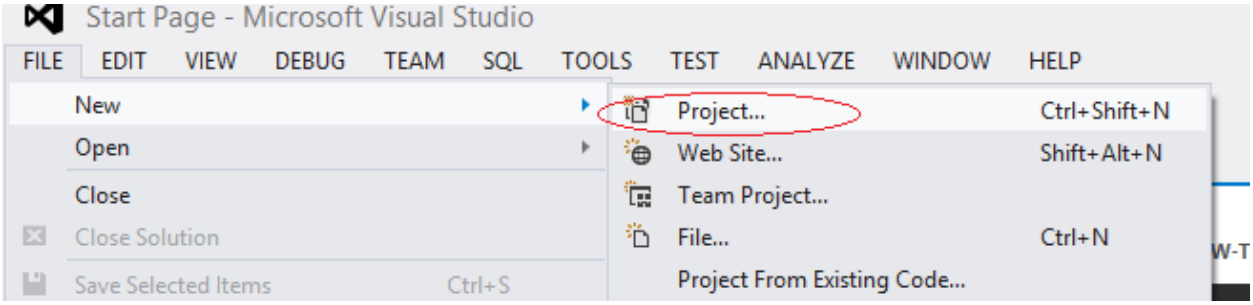
CREATING NEW PROJECT

The first thing you do when you want to create a new application is to create a NEW PROJECT.

This can be done from start page.



New project created from the NEW PROJECT window:



Then the "NEW PROJECT" window will appear.

In this window you will select an appropriate template based on what kind of application you want to create, and a name and location for your project and solution.

The most common applications are:

Windows form application.

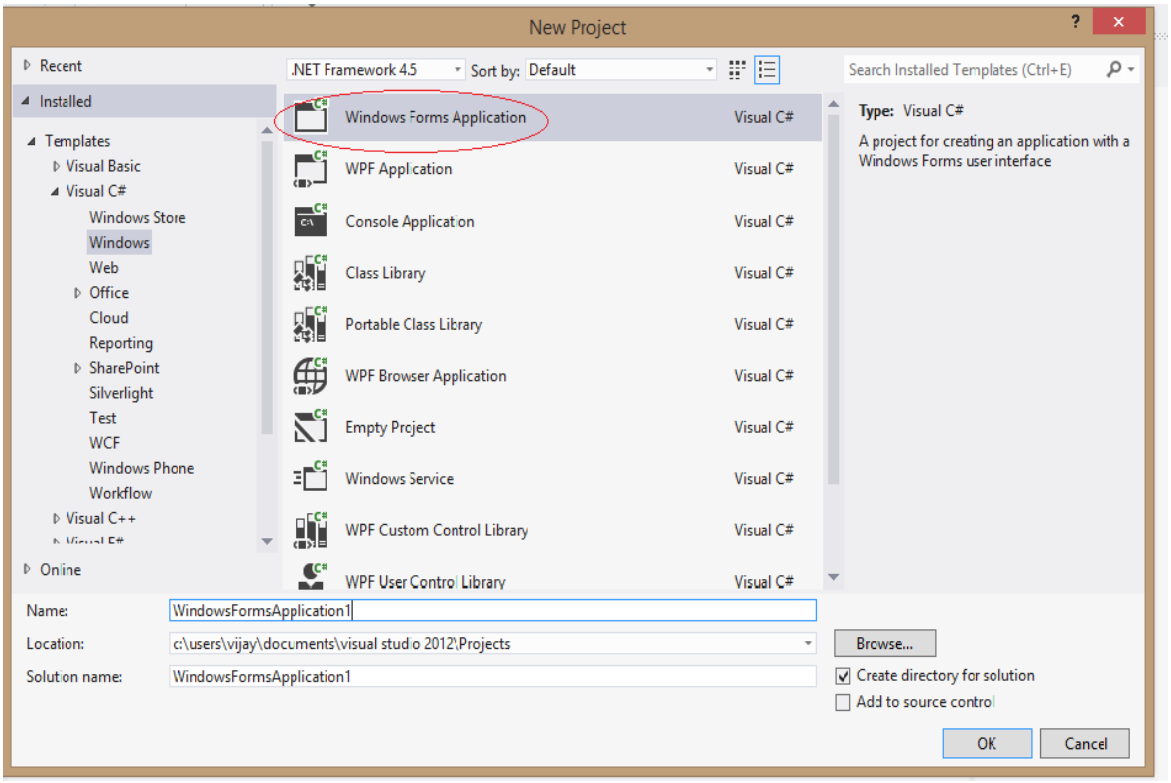
Console application.

WPF application

ASP.NET web application.

Silverlight application.

Select WINDOWS FORMS APPLICATION.



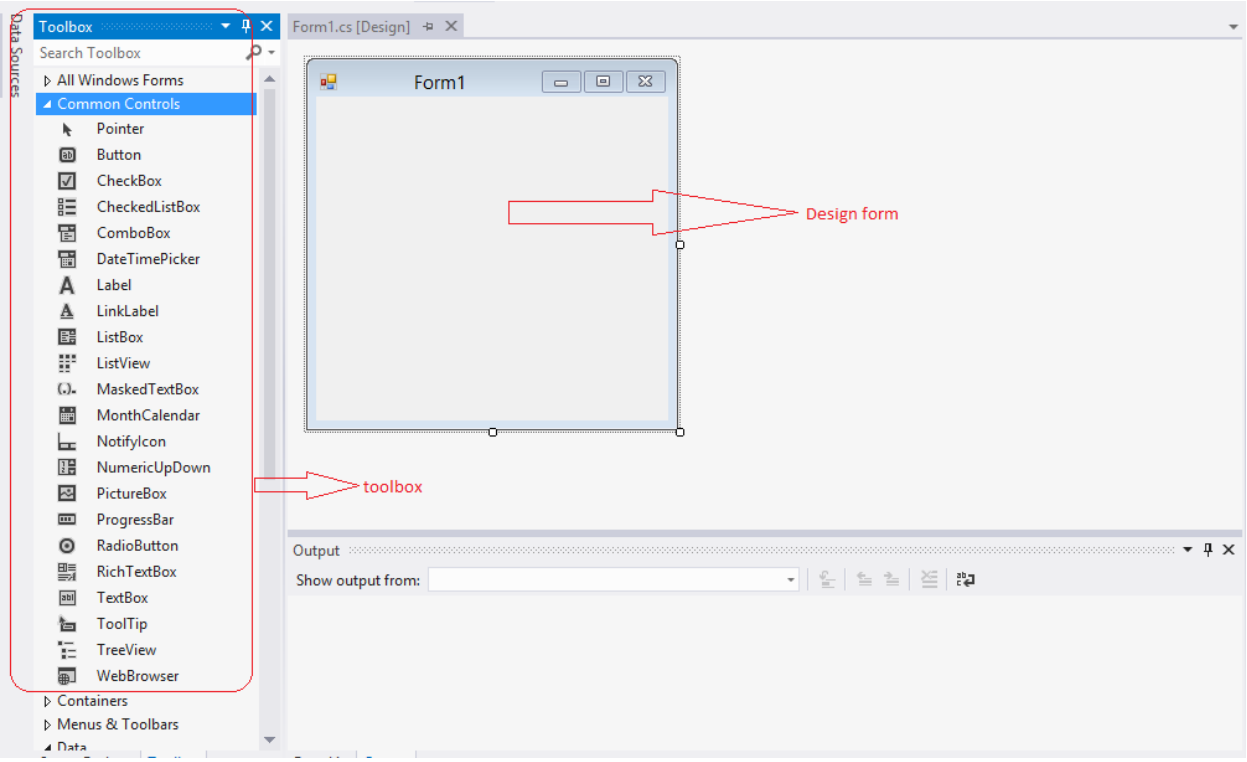
TOOLBOX

When you select WINDOWS FORM APPLICATION, you will get FORM DESIGN WINDOW,it is used to design USER interface by making use of TOOLBOX on the left side of window,

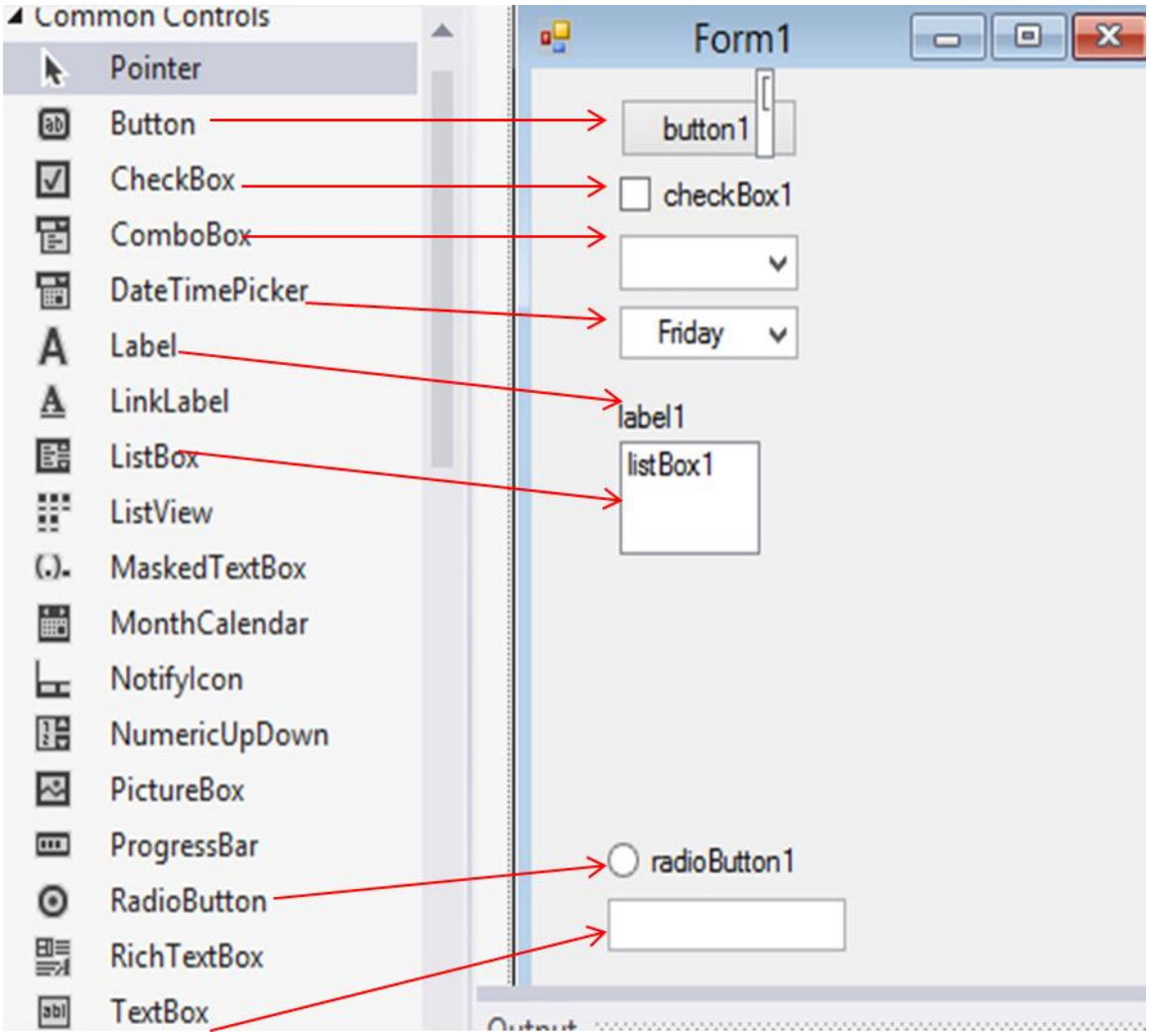
The TOOLBOX contains all the necessary controls, etc. You need to create a user interface by making use of these controls as shown in figure below.

In order to use these controls, just drag and drop it on to your Design forms, as shown in figure.

Figure shows TOOLBOX and DESIGN FORM:

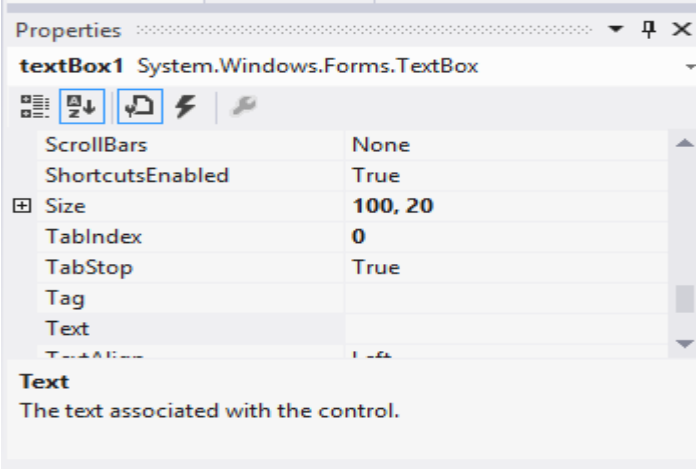


The following screenshot shows, making use of these toolbox controls for designing the user interface on DESIGN FORM.



PROPERTIES WINDOW

Each TOOLBOX we have used on our form has many properties that we can set. This is done by using Properties window. We can find the property window on the right bottom side of your project

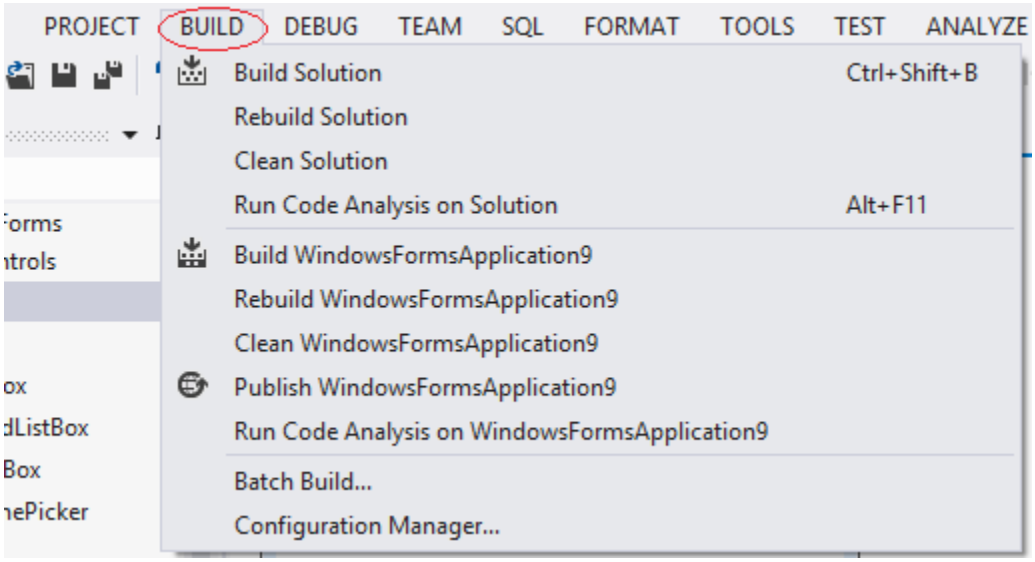


BUILD AND DEBUGGING TOOL

The visual studio has lots of Build and Debugging Tools,

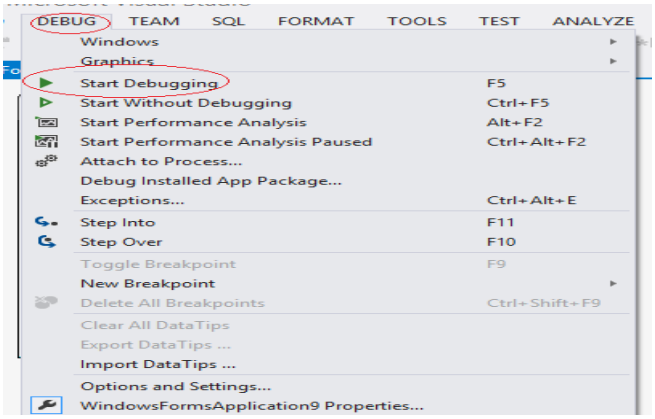
BUILD MENU:

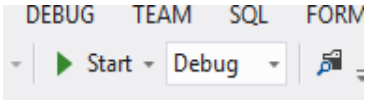
Below we see the Build menu. The most used Build tool is **BUILD SOLUTIONS**.



DEBUG MENU:

In order to RUN or DEBUG your windows form we make use of DEBUG TOOLs. The most used debug tool is START DEBUGGING.it can be find the shortcut for this on the top of your visual studio windows.





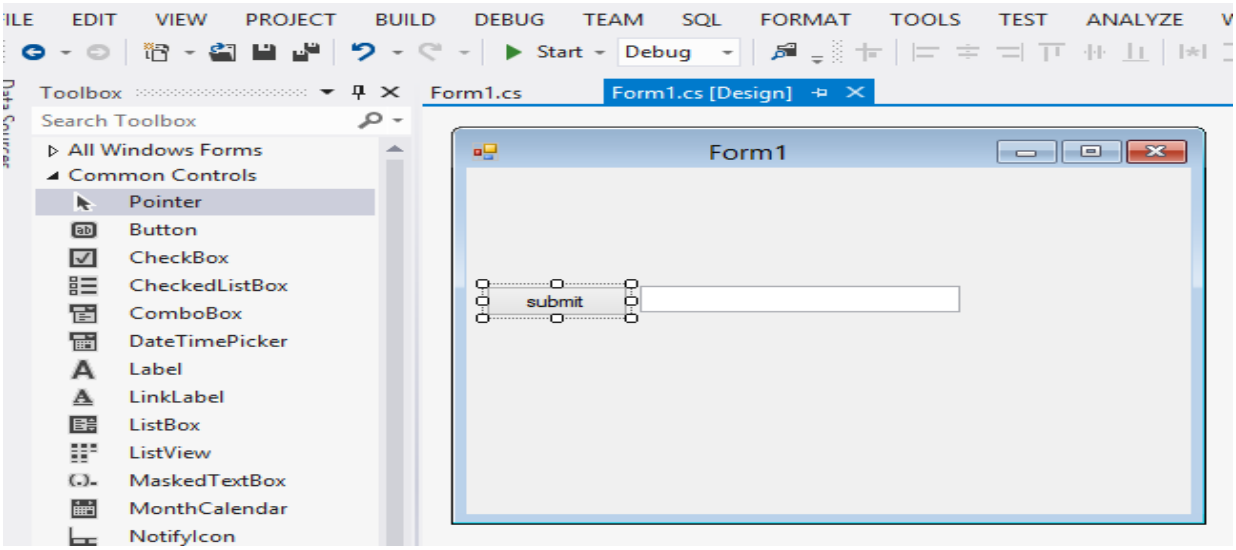
WINDOWS PROGRAMMING

When creating ordinary windows form application, we can select between the following:

- Windows form Application
- WPF application

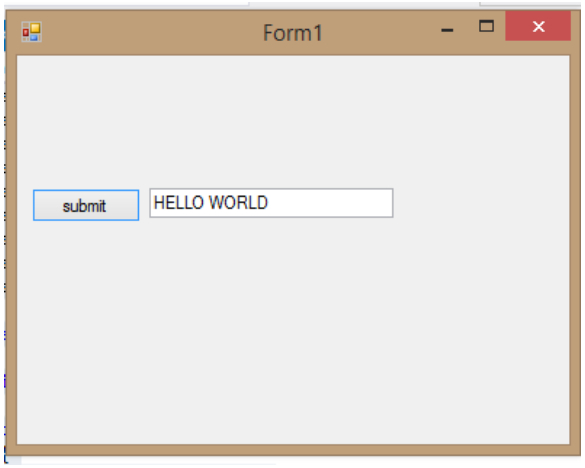
HELLO WORLD

We start by creating traditional “HELLO WORLD” application using Windows Form Application is shown below. The visual studio UI shown below.



In this application we make use of simple textbox and Button(Button name is changed to Submit in the properties) when we click on submit the “HELLO WORLD ”massage will be displayed in the Textbox.

The OUTPUT of this form as shown below:



The code is as follow:

```
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 WindowsFormsApplication9
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = "HELLO WORLD";  //displaying hello world in the textbox.
        }
    }
}
```

DATA TYPES AND VERIABLES

"Variables" are simply storage locations for data. You can place data into them and retrieve their contents as part of a C# expression. The interpretation of the data in a variable is controlled through "Types".

The C# simple types consist of:

- Boolean type
- Numeric types: Integrals, Floating Point, Decimal
- String type

BOOLEAN TYPES

Boolean types are declared using the keyword "bool". They have two values: "true" or "false". In other languages, such as C and C++, boolean conditions can be satisfied where 0 means false and anything else means true. However, in C# the only values that satisfy a boolean condition is true and false, which are official keywords.

Example:

bool content = true; bool

noContent=false

Numeric types: Integrals, Floating Point, Decimal

Example:

```
int i=35;
long y=654654;
float x;
double y;
decimal z;
```

String type

Example:

```
string myString=”Hei på deg”;
```

Special characters that may be used in strings:

Escape Sequence	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null, not the same as the C# *null* value
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

Arrays

Example:

```
int[] myInts = { 5, 10, 15 };
```

CONTROL FLOW

To be able to control the flow in your program is important in every programming language.

The two most important techniques are:

- The **if** Statement
- The **switch** Statement

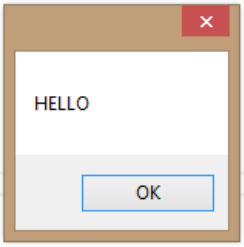
The if Statement

The if statement is probably the most used mechanism to control the flow in your application. An if statement allows you to take different paths of logic, depending on a given condition. When the condition evaluates to a Boolean true, a block of code for that true condition will execute. You have the option of a single if statement, multiple else if statements, and an optional else statement.

Example:

```
myTest=false;
if (myTest==false)
{
MessageBox.Show("Hello”);
}
```

output:



For more complex logic we use the **if … else** statement.

Example:

```
bool myTest;
myTest=true;

if (myTest == false)
{
      MessageBox.Show("Hello1");
}
else
{
      MessageBox.Show("Hello2");

}
```

Or you can use **nested if… else** if sentences.

Example:

```
int myTest;
myTest=2;
if (myTest == 1)
{
      MessageBox.Show("Hello1");
}
else if (myTest == 2)
{
      MessageBox.Show("Hello2");
}
else
{
      MessageBox.Show("Hello3");
}
```

The switch Statement

Another form of selection statement is the **switch** statement, which executes a set of logic depending on the value of a given parameter. The types of the values a switch statement operates on can be booleans, enums, integral types, and strings.

Example:

```
 switch (myTest)
{
      case 1:
      MessageBox.Show("Hello1
      "); break;
      case 2:
      MessageBox.Show("Hello2
      "); break;
      default:
      MessageBox.Show("Hello”
      ); break;
}
```

LOOPS

In C# we have different kind of loops:

- The while loop
- The do loop
- The for loop
- The foreach loop

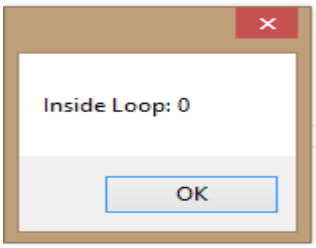
The while Loop

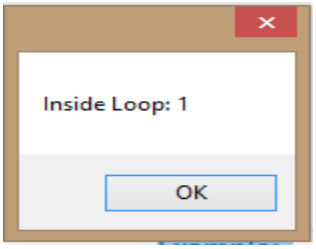
A while loop will check a condition and then continues to execute a block of code as long as the condition evaluates to a boolean value of true.

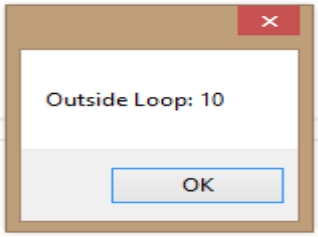
Example:

```
int myInt = 0;

while (myInt < 10)
{
      MessageBox.Show("Inside Loop: " +
      myInt.ToString()); myInt++;
}
MessageBox.Show("Outside Loop: " + myInt.ToString());
```

OUTPUT:







The do Loop

A do loop is similar to the while loop, except that it checks its condition at the end of the loop. This means that the do loop is guaranteed to execute at least one time. On the other hand, a while loop evaluates its boolean expression at the beginning and there is generally no guarantee that the statements inside the loop will be executed, unless you program the code to explicitly do so.

Example:

```
int myInt = 0;

do

{

MessageBox.Show("Inside Loop: " + myInt.ToString());

myInt++;

} while (myInt < 10);

MessageBox.Show("Outside Loop: " + myInt.ToString());
```

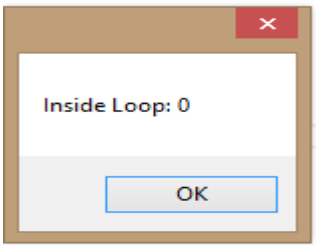
The for Loop

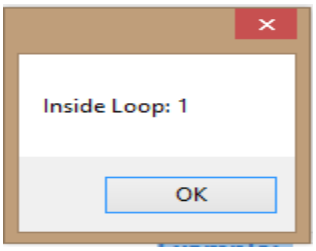
A for loop works like a while loop, except that the syntax of the for loop includes initialization and condition modification. for loops are appropriate when you know exactly how many times you want to perform the statements within the loop.

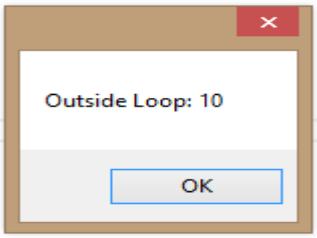
Example:

```
for (int i = 0; i < 10; i++)
{
      MessageBox.Show("Inside Loop: " +
      myInt.ToString()); myInt++;
}
MessageBox.Show("Outside Loop: " + myInt.ToString());
```

OUTPUT:





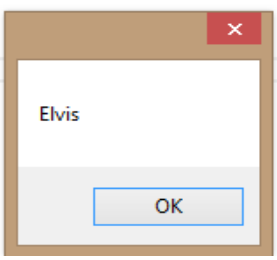


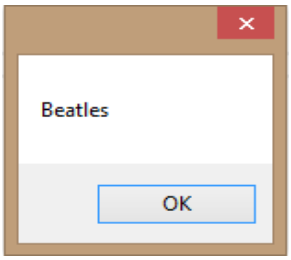
The foreach Loop

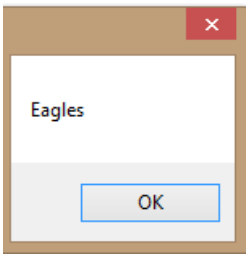
A foreach loop is used to iterate through the items in a list. It operates on arrays or collections.

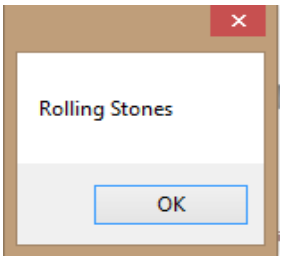
Example:

```
string[] names = { "Elvis", "Beatles", "Eagles", "Rolling Stones" };

foreach (string person in names)
{
      MessageBox.Show(person);
}
```









SERIAL COMMUNICATION

In telecommunication and computer science, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels.

Setting Up

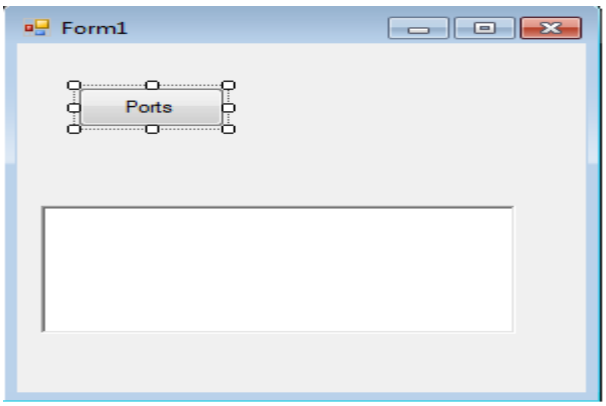
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.IO.Ports;

using System.Windows.Forms;
```

System.IO.Ports is the class to use without resulting to low level hacking. This covers all the serial ports that appear on the machine.

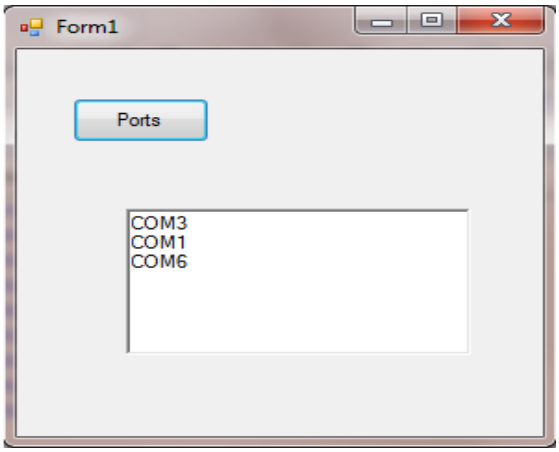
```
SerialPort ComPort = new SerialPort();
```

This will create an object called ComPort. This will create a serial port object with the following parameters as default 9600bps, no parity, one stop bit and no flow control.

Shown below is the form:



This is standard Windows Forms Application via File menu. To this add the button (name Ports) and a Rich Text Box.The button is called btnGetSerialPorts and the Rich Text called as rtbIncomingData (the name will become apparent later).The rich text box is used as it is more flexible than the ordinary text box. Its uses for sorting and aligning text are considerably more than the straight textbox.



This shows all the devices that appear as com ports, a mistake to make is thinking that a device if plugged into the USB will appear as a COM Port.

The baud rate is the amount of possible events that can happen in a second. It is displays usually as a number of bit per second, the possible number that can be used are 300, 600, 1200, 2400, 9600, 14400, 19200, 38400, 57600, and 115200 (these come from the UAR 8250 chip is used, if a 16650 the additional rates of 230400, 460800 and 921600) .

The next box is the number of Data bits, these represent the total number of transitions of the data transmission (or Tx line) 8 is the standard (8 is useful for reading certain embedded application as it gives two nibbles (4 bit sequences).

The Handshaking property is used when a full set of connections are used (such as the grey 9 way D-types that litter my desk). It was used originally to ensure both ends lined up with each other and the data was sent and received properly. A common handshake was required between both sender and receiver. Below is the code for the combo box:

Here is the complete code for serial communication between transmitter and receiver.

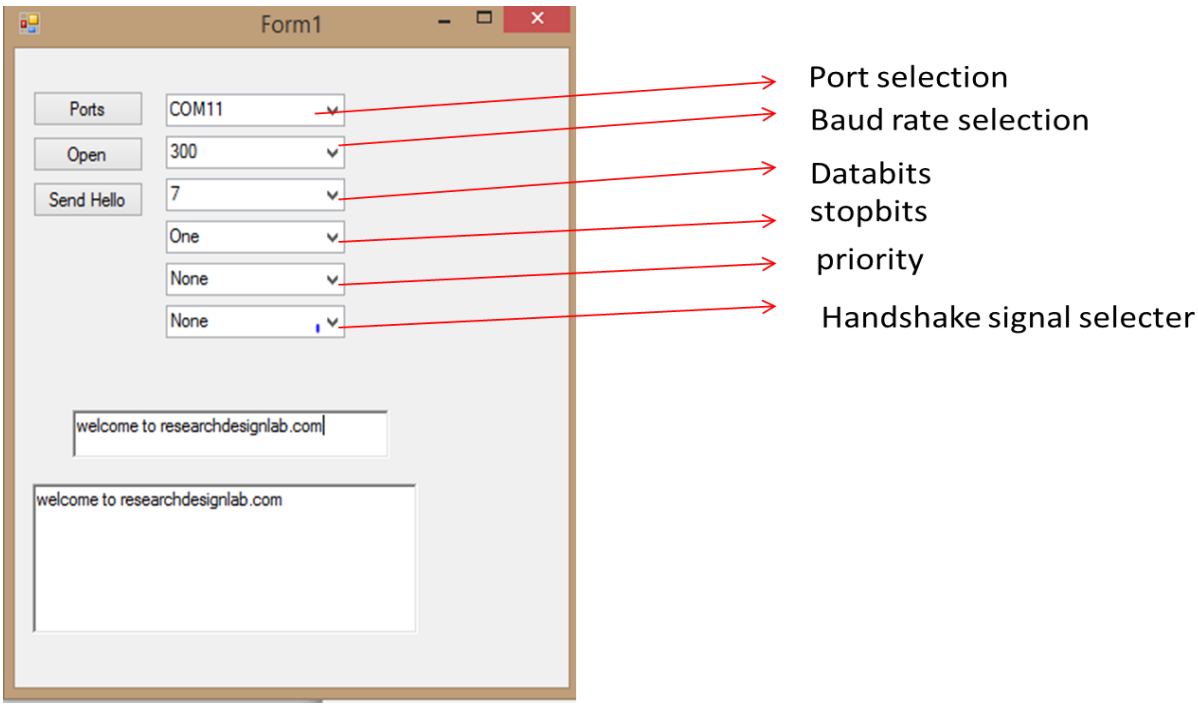
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.Windows.Forms;


namespace CodeProjectSerialComms
{
    public partial class Form1 : Form
    {
        SerialPort ComPort = new SerialPort();

        internal delegate void SerialDataReceivedEventHandlerDelegate(object sender,
SerialDataReceivedEventArgs e);
        internal delegate void SerialPinChangedEventHandlerDelegate(object sender,
SerialPinChangedEventArgs e);
        private SerialPinChangedEventHandler SerialPinChangedEventHandler1;
        delegate void SetTextCallback(string text);
        string InputData = String.Empty;

        public Form1()
        {
            InitializeComponent();
            SerialPinChangedEventHandler1 = new SerialPinChangedEventHandler(PinChanged);
            ComPort.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(port_DataReceived_1);
        }

        private void btnGetSerialPorts_Click(object sender, EventArgs e)
        {
            string[] ArrayComPortsNames = null;
            int index = -1;
            string ComPortName = null;

//Com Ports
            ArrayComPortsNames = SerialPort.GetPortNames();
            do
            {
                index += 1;
                cboPorts.Items.Add(ArrayComPortsNames[index]);


            } while (!((ArrayComPortsNames[index] == ComPortName) || (index ==
ArrayComPortsNames.GetUpperBound(0))));
            Array.Sort(ArrayComPortsNames);
```

```
            if (index == ArrayComPortsNames.GetUpperBound(0))
            {
                ComPortName = ArrayComPortsNames[0];
            }
            //get first item print in text
            cboPorts.Text = ArrayComPortsNames[0];
//Baud Rate
            cboBaudRate.Items.Add(300);
            cboBaudRate.Items.Add(600);
            cboBaudRate.Items.Add(1200);
            cboBaudRate.Items.Add(2400);
            cboBaudRate.Items.Add(9600);
            cboBaudRate.Items.Add(14400);
            cboBaudRate.Items.Add(19200);
            cboBaudRate.Items.Add(38400);
            cboBaudRate.Items.Add(57600);
            cboBaudRate.Items.Add(115200);
            cboBaudRate.Items.ToString();
            //get first item print in text
            cboBaudRate.Text = cboBaudRate.Items[0].ToString();
//Data Bits
            cboDataBits.Items.Add(7);
            cboDataBits.Items.Add(8);
            //get the first item print it in the text
            cboDataBits.Text = cboDataBits.Items[0].ToString();

//Stop Bits
            cboStopBits.Items.Add("One");
            cboStopBits.Items.Add("OnePointFive");
            cboStopBits.Items.Add("Two");
            //get the first item print in the text
            cboStopBits.Text = cboStopBits.Items[0].ToString();
//Parity
            cboParity.Items.Add("None");
            cboParity.Items.Add("Even");
            cboParity.Items.Add("Mark");
            cboParity.Items.Add("Odd");
            cboParity.Items.Add("Space");
            //get the first item print in the text
            cboParity.Text = cboParity.Items[0].ToString();
//Handshake
            cboHandShaking.Items.Add("None");
            cboHandShaking.Items.Add("XOnXOff");
            cboHandShaking.Items.Add("RequestToSend");
            cboHandShaking.Items.Add("RequestToSendXOnXOff");
            //get the first item print it in the text
            cboHandShaking.Text = cboHandShaking.Items[0].ToString();

        }
```

```
        private void port_DataReceived_1(object sender, SerialDataReceivedEventArgs e)
        {
            InputData = ComPort.ReadExisting();
            if (InputData != String.Empty)
            {
                this.BeginInvoke(new SetTextCallback(SetText), new object[] { InputData
});
            }
        }
        private void SetText(string text)
        {
            this.rtbIncoming.Text += text;
        }



        private void btnPortState_Click(object sender, EventArgs e)
        {

            if (btnPortState.Text == "Closed")
            {
                btnPortState.Text = "Open";
                ComPort.PortName = Convert.ToString(cboPorts.Text);
                ComPort.BaudRate = Convert.ToInt32(cboBaudRate.Text);
                ComPort.DataBits = Convert.ToInt16(cboDataBits.Text);
                ComPort.StopBits = (StopBits)Enum.Parse(typeof(StopBits),
cboStopBits.Text);
                ComPort.Handshake = (Handshake)Enum.Parse(typeof(Handshake),
cboHandShaking.Text);
                ComPort.Parity = (Parity)Enum.Parse(typeof(Parity), cboParity.Text);
                ComPort.Open();
            }
            else if (btnPortState.Text == "Open")
            {
                btnPortState.Text = "Closed";
                ComPort.Close();

            }
        }
        private void rtbOutgoing_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)13) // enter key
            {
                ComPort.Write("\r\n");
                rtbOutgoing.Text = "";
            }
            else if (e.KeyChar < 32 || e.KeyChar > 126)
            {
                e.Handled = true; // ignores anything else outside printable ASCII range
            }
            else
            {
                ComPort.Write(e.KeyChar.ToString());
```

```
            }
        }
        private void btnHello_Click(object sender, EventArgs e)
        {
            ComPort.Write("Hello World!");
        }
    }
```

OUTPUT:



USB RFID INTERFACE WITH C#

Visit http://researchdesignlab.com/rfid-reader-usb.html to buy this product.

Radio-frequency identification (**RFID**) is the wireless use of electromagnetic fields to transfer data, for the purposes of automatically identifying and tracking tags attached to objects. The tags contain electronically stored information. Some tags are powered byelectromagnetic induction from magnetic fields produced near the reader. Some types collect energy from the interrogating radio waves and act as a passive transponder. Other types have a local power source such as a battery and may operate at hundreds of meters from the reader. Unlike a barcode, the tag does not necessarily need to be within line of sight of the reader, and may be embedded in the tracked object. Radio frequency identification (RFID) is one method for Automatic Identification and Data Capture (AIDC).

Reading USB RFID data from serial port

We can use Serial port for Reading the Tag values from the RF-ID Reader. For this we need to connect the RF-ID Reader using the Serial cable to the port and use the relevant COM Port No# to the Serial port Object of C# to Connect.

Normally the System.Net Contains the Serial Port Class and also available in the Toolbox as Serial port component for your Win Forms App

The following code is for reading RFID data from serial port.

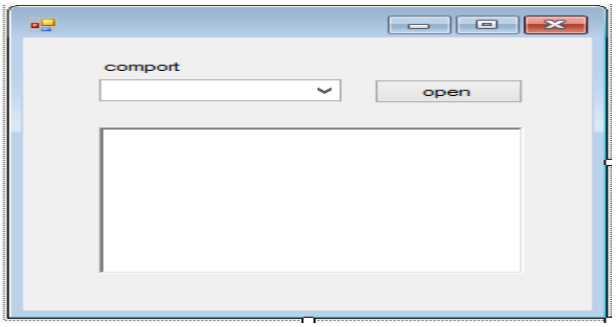
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace com
{
    public partial class Form1 : Form

    {
        CommManager com = new CommManager();
        string RxString;
        public Form1()
        {
            InitializeComponent();

        }


           }

private void Form1_Load_1(object sender, EventArgs e) //when form loads setting a values.
        {
        com .SetPortNameValues(comboBox2 );
            com.Parity="None";
            com.BaudRate = "9600";
            com.StopBits="One";
            com.DataBits = "8";
            com.DisplayWindow=richTextBox1;



        }

        private void button1_Click(object sender, EventArgs e)
        {
            com.PortName = comboBox2.Items[comboBox2.SelectedIndex].ToString();
            com.OpenPort();
```

This snippet of code reads all the comport values into combobox.and opens the port for reading.

```
        }

    }

}
```

The DESIGN of FORM is shown.



Here we made use of combobox,richtextbox for displaying RFID DATA after reading and OPEN button to open the selected comport.for above project you have to create a class for comport management, the code for this class as given below and it is common for all the serial communication interface.

```
using Microsoft.VisualBasic;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;

using System.Text;
using System.Drawing;
using System.IO.Ports;
using System.Windows.Forms;
public class CommManager
{

    public bool retriewResonse;
    public int signalStrength;
    public bool readRfid;
    // Public rxModeBin As Boolean
    //Public messag2 As String

    #region "Manager Enums"
    /// <summary>
    /// enumeration to hold our transmission types
    /// </summary>
    public enum TransmissionType
    {
        Text,
        Hex
    }

    public enum RxMode
    {
        ASCII,
        Dec,
```

```
        Hex,
        Bin
    }


    /// <summary>
    /// enumeration to hold our message types
    /// </summary>
    public enum MessageType
    {
        Incoming,
        Outgoing,
        Normal,
        Warning,
        Error
    }
    #endregion

    #region "Manager Variables"
    //property variables
    private string _baudRate = string.Empty;
    private string _parity = string.Empty;
    private string _stopBits = string.Empty;
    private string _dataBits = string.Empty;
    private string _portName = string.Empty;
    private TransmissionType _transType;
    private RxMode rx_Mode;
    private RichTextBox _displayWindow;
    private ProgressBar strngthbar;

    private TextBox rfidText;
    private Label sgnStrLbl;
    private string _msg = string.Empty;
    private MessageType _type;
    //global manager variables
    private Color[] MessageColor = {
              Color.Blue,
              Color.Green,
              Color.Black,
              Color.Orange,
              Color.Red
       };
    private SerialPort comPort = new SerialPort();
    #endregion
    private bool write = true;

    #region "Manager Properties"
    /// <summary>
    /// Property to hold the BaudRate
    /// of our manager class
    /// </summary>
    public string BaudRate
    {
```

```
        get { return _baudRate; }
        set { _baudRate = value; }
    }

    /// <summary>
    /// property to hold the Parity
    /// of our manager class
    /// </summary>
    public string Parity
    {
        get { return _parity; }
        set { _parity = value; }
    }

    /// <summary>
    /// property to hold the StopBits
    /// of our manager class
    /// </summary>
    public string StopBits
    {
        get { return _stopBits; }
        set { _stopBits = value; }
    }

    /// <summary>
    /// property to hold the DataBits
    /// of our manager class
    /// </summary>
    public string DataBits
    {
        get { return _dataBits; }
        set { _dataBits = value; }
    }

    /// <summary>
    /// property to hold the PortName
    /// of our manager class
    /// </summary>
    public string PortName
    {
        get { return _portName; }
        set { _portName = value; }
    }

    /// <summary>
    /// property to hold our TransmissionType
    /// of our manager class
    /// </summary>
    public TransmissionType CurrentTransmissionType
    {
        get { return _transType; }
        set { _transType = value; }
    }
```

```
    /// <summary>
    /// property to hold our display window
    /// value
    /// </summary>
    public RichTextBox DisplayWindow
    {
        get { return _displayWindow; }
        set { _displayWindow = value; }
    }


    public RxMode Current_rxMode
    {
        get { return rx_Mode; }
        set { rx_Mode = value; }
    }

    public TextBox rfidDisplay
    {

        get { return rfidText; }
        set { rfidText = value; }
    }

    /// Property to hold the message being sent
    /// through the serial port
>
    public string Message
    {
        get { return _msg; }
        set { _msg = value; }
    }

    public MessageType Type
    {
        get { return _type; }
        set { _type = value; }
    }

    public ProgressBar SignalStrengthBar
    {

        get { return strngthbar; }
        set { strngthbar = value; }
    }
    #endregion
    public Label SignalStrengthLbl
    {


        get { return sgnStrLbl; }
        set { sgnStrLbl = value; }
```

```
    }

    #region "Manager Constructors"
    /// <summary>
    /// Constructor to set the properties of our Manager Class
    /// </summary>
    /// <param name="baud">Desired BaudRate</param>
    /// <param name="par">Desired Parity</param>
    /// <param name="sBits">Desired StopBits</param>
    /// <param name="dBits">Desired DataBits</param>
    /// <param name="name">Desired PortName</param>
    public CommManager(string baud, string par, string sBits, string dBits, string name,
RichTextBox rtb)
    {
        _baudRate = baud;
        _parity = par;
        _stopBits = sBits;
        _dataBits = dBits;
        _portName = name;
        _displayWindow = rtb;
        //now add an event handler
        comPort.DataReceived += comPort_DataReceived;
    }

    /// <summary>
    /// Comstructor to set the properties of our
    /// serial port communicator to nothing
    /// </summary>
    public CommManager()
    {
        _baudRate = string.Empty;
        _parity = string.Empty;
        _stopBits = string.Empty;
        _dataBits = string.Empty;
        _portName = "COM1";
        _displayWindow = null;
        //add event handler
        comPort.DataReceived += comPort_DataReceived;
    }
    public CommManager(ref ProgressBar strngthBar)
    {
        _baudRate = string.Empty;
        _parity = string.Empty;
        _stopBits = string.Empty;
        _dataBits = string.Empty;
        _portName = "COM1";
        _displayWindow = null;
        //add event handler
        comPort.DataReceived += comPort_DataReceived;
    }
    #endregion

    #region "WriteData"
```

```
public void WriteData(string msg)
{

    try
    {
        switch (CurrentTransmissionType)
        {
            case TransmissionType.Text:
                //first make sure the port is open
                //if its not open then open it
                if (!(comPort.IsOpen == true))
                {
                    comPort.Open();
                }
                //send the message to the port
                comPort.Write(msg);

                //display the message
                _type = MessageType.Outgoing;
                //+ "" + Environment.NewLine + ""
                _msg = msg;
                DisplayData(_type, _msg);
                break; // TODO: might not be correct. Was : Exit Select

                break;
            case TransmissionType.Hex:
                try
                {
                    //convert the message to byte array
                    byte[] newMsg = HexToByte(msg);
                    if (!write)
                    {
                        DisplayData(_type, _msg);
                        return;
                    }
                    //send the message to the port
                    comPort.Write(newMsg, 0, newMsg.Length);
                    //convert back to hex and display
                    _type = MessageType.Outgoing;
                    //  + "" + Environment.NewLine + ""
                    _msg = ByteToHex(newMsg);
                    DisplayData(_type, _msg);
                }
                catch (FormatException ex)
                {
                    //display error message
                    _type = MessageType.Error;
                    _msg = ex.Message + "" + Environment.NewLine + "";
                    DisplayData(_type, _msg);
                }
                finally
                {
```

```
                    _displayWindow.SelectAll();
                }

                break; // TODO: might not be correct. Was : Exit Select

                break;
            default:
                //first make sure the port is open
                //if its not open then open it
                if (!(comPort.IsOpen == true))
                {
                    comPort.Open();
                }
                //send the message to the port
                comPort.Write(msg);

                //display the message
                _type = MessageType.Outgoing;
                //+ "" + Environment.NewLine + ""
                _msg = msg;
                DisplayData(MessageType.Outgoing, _msg);
                break; // TODO: might not be correct. Was : Exit Select

                break;
        }

    }
    catch (Exception ex)
    {
    }

}
#endregion

#region "HexToByte"
/// <summary>
/// method to convert hex string into a byte array
/// </summary>
/// <param name="msg">string to convert</param>
/// <returns>a byte array</returns>
private byte[] HexToByte(string msg)
{
    if (msg.Length % 2 == 0)
    {
        //remove any spaces from the string
        _msg = msg;
        _msg = msg.Replace(" ", "");
        //create a byte array the length of the
        //divided by 2 (Hex is 2 characters in length)
        byte[] comBuffer = new byte[_msg.Length / 2];
        for (int i = 0; i <= _msg.Length - 1; i += 2)
        {
```

```
                comBuffer[i / 2] = Convert.ToByte(Convert.ToByte(_msg.Substring(i, 2),
16));
            }
            write = true;
            //loop through the length of the provided string
            //convert each set of 2 characters to a byte
            //and add to the array
            //return the array
            return comBuffer;
        }
        else
        {
            _msg = "Invalid format";
            _type = MessageType.Error;
            // DisplayData(_Type, _msg)
            write = false;
            return null;
        }
    }
    #endregion

    #region "ByteToHex"
    /// <summary>
    /// method to convert a byte array into a hex string
    /// </summary>
    /// <param name="comByte">byte array to convert</param>
    /// <returns>a hex string</returns>
    private string ByteToHex(byte[] comByte)
    {
        //create a new StringBuilder object
        StringBuilder builder = new StringBuilder(comByte.Length * 3);
        //loop through each byte in the array
        foreach (byte data in comByte)
        {
            builder.Append(Convert.ToString(data, 16).PadLeft(2, '0').PadRight(3, ' '));
            //convert the byte to a string and add to the stringbuilder
        }
        //return the converted value
        return builder.ToString().ToUpper();
    }
    #endregion

    #region "DisplayData"
    /// <summary>
    /// Method to display the data to and
    /// from the port on the screen
    /// </summary>
    /// <remarks></remarks>
    [STAThread()]
    private void DisplayData(MessageType type, string msg)
    {
        _displayWindow.Invoke(new EventHandler(DoDisplay));
    }
```

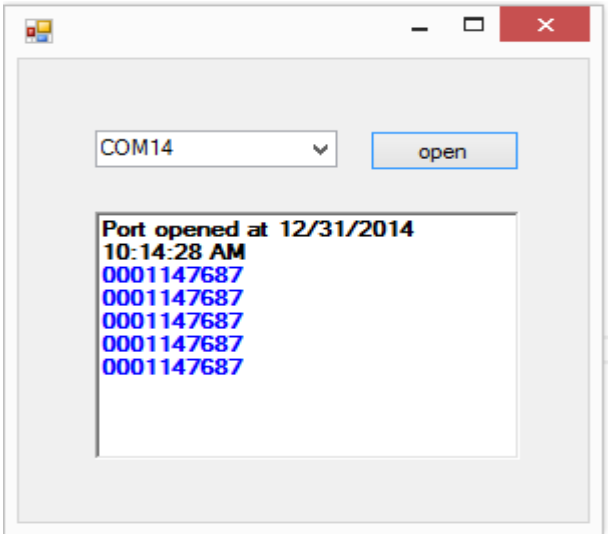
```
#endregion

#region "OpenPort"
public bool OpenPort()
{
    try
    {
        //first check if the port is already open
        //if its open then close it
        if (comPort.IsOpen == true)
        {
            comPort.Close();
        }

        //set the properties of our SerialPort Object
        comPort.BaudRate = int.Parse(_baudRate);
        //BaudRate
        comPort.DataBits = int.Parse(_dataBits);
        //DataBits
        comPort.StopBits = (StopBits)Enum.Parse(typeof(StopBits), _stopBits);
        //StopBits
        comPort.Parity = (Parity)Enum.Parse(typeof(Parity), _parity);
        //Parity
        comPort.PortName = _portName;
        //PortName
        //now open the port
        comPort.Open();
        //display message
        _type = MessageType.Normal;
        _msg = "Port opened at " + DateTime.Now + "" + Environment.NewLine + "";
        //MessageBox.Show("opened");
        DisplayData(_type, _msg);
        //return true
        return true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        DisplayData(MessageType.Error, ex.Message);
        return false;
    }
}
#endregion

#region " ClosePort "
public void ClosePort()
{
    if (comPort.IsOpen)
    {
        _msg = "Port closed at " + DateTime.Now + "" + Environment.NewLine + "";
        _type = MessageType.Normal;
        DisplayData(_type, _msg);
        comPort.Close();
```

```
        }
    }
    #endregion

    #region "SetParityValues"
    public void SetParityValues(object obj)
    {
        foreach (string str in Enum.GetNames(typeof(Parity)))
        {
            ((ComboBox)obj).Items.Add(str);
        }
    }
    #endregion

    #region "SetStopBitValues"
    public void SetStopBitValues(object obj)
    {
        foreach (string str in Enum.GetNames(typeof(StopBits)))
        {
            ((ComboBox)obj).Items.Add(str);
        }
    }
    #endregion

    #region "SetPortNameValues"

    public void SetPortNameValues(object obj)
    {
        foreach (string str in SerialPort.GetPortNames())
        {
            ((ComboBox)obj).Items.Add(str);
        }
    }
    #endregion

    #region "comPort_DataReceived"
    /// <summary>
    /// method that will be called when theres data waiting in the buffer
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void comPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        //determine the mode the user selected (binary/string)
        switch (CurrentTransmissionType)
        {
            case TransmissionType.Text:
                //user chose string
                //read data waiting in the buffer
                string msg = comPort.ReadExisting();


                //MessageBox.Show(msg)
```

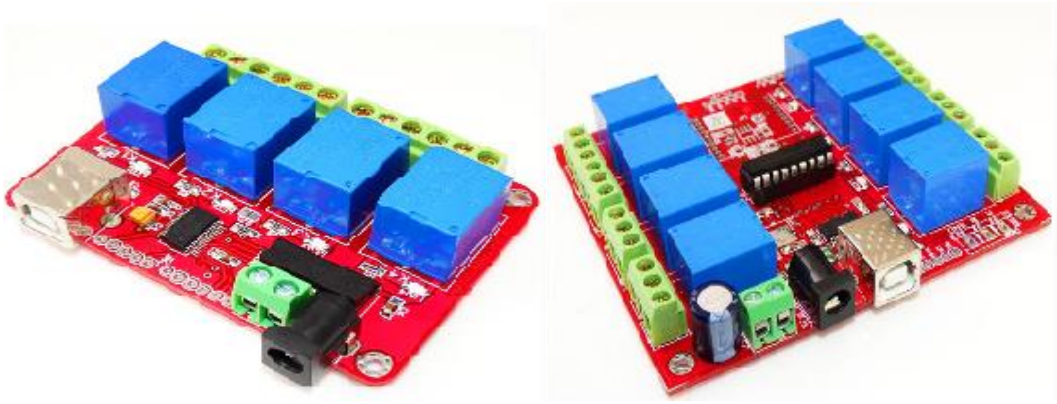
```
                try
                {

                }
                catch (Exception ex)
                {
                }

                //display the data to the user
                _type = MessageType.Incoming;
                _msg = msg;
                DisplayData(MessageType.Incoming, msg + "" + Environment.NewLine + "");
                break; // TODO: might not be correct. Was : Exit Select

                break;
            case TransmissionType.Hex:
                //user chose binary
                //retrieve number of bytes in the buffer
                int bytes = comPort.BytesToRead;
                //create a byte array to hold the awaiting data
                byte[] comBuffer = new byte[bytes];
                //read the data and store it
                comPort.Read(comBuffer, 0, bytes);
                //display the data to the user
                _type = MessageType.Incoming;
                _msg = ByteToHex(comBuffer) + "" + Environment.NewLine + "";
                DisplayData(MessageType.Incoming, ByteToHex(comBuffer) + "" +
Environment.NewLine + "");
                break; // TODO: might not be correct. Was : Exit Select

                break;
            default:
                //read data waiting in the buffer
                string str = comPort.ReadExisting();

                try
                {


                }
                catch (Exception ex)
                {
                }

                //display the data to the user
                _type = MessageType.Incoming;

                _msg = str + "" + Environment.NewLine + "";
                DisplayData(MessageType.Incoming, str + "" + Environment.NewLine + "");
                break; // TODO: might not be correct. Was : Exit Select

                break;
        }
```

```
    }
    #endregion

    #region "DoDisplay"
    private void DoDisplay(object sender, EventArgs e)
    {
        _displayWindow.SelectedText = string.Empty;
        _displayWindow.SelectionFont = new Font(_displayWindow.SelectionFont,
FontStyle.Bold);
        _displayWindow.SelectionColor = MessageColor[Convert.ToInt32(_type)];

        _displayWindow.AppendText(_msg);

        _displayWindow.ScrollToCaret();
    }
    #endregion



    public string Remove(string value, string rmv)
    {
        int pos = value.IndexOf(rmv);

        if (pos >= 0)
        {
            return value.Remove(pos, rmv.Length);
        }

        return value;
    }

}
```

Output:



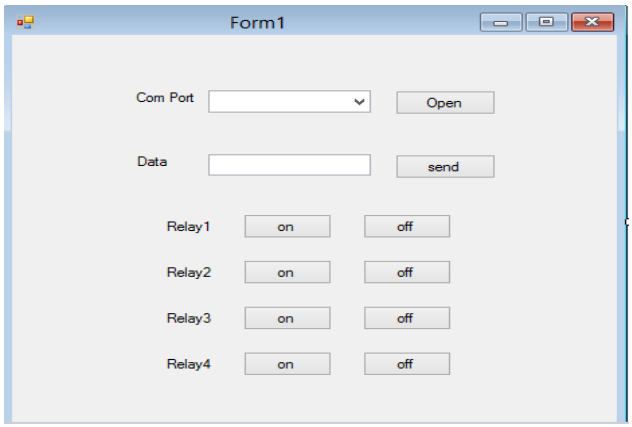
FT245 RELAY CONTROLLER



Visit http://researchdesignlab.com/usb-4-channel-relay-board.html to buy 4-channel relay and visit http://researchdesignlab.com/usb-8-channel-relay-board.html to buy 8-channel relay.

A **relay** is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relays are used where it is necessary to control a circuit by a low-power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits as amplifiers: they repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

Here we are making use of 4 channel relay to controlling it, The following picture show the design part of it, in this we have used one combo box for reading com port and open button to open the selected port, and DATA text box, this is for entering manually which relay should turn on suppose if you enter 'ff' it will turn on relay1.



The complete source code for controlling 4/8 channel relay.

```
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 ft245rlAPP
{
    public partial class Form1 : Form
    {
CommManager comMangr=new CommManager();
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            try {
                    comMangr.SetPortNameValues(comPortCmb);


                    comMangr.BaudRate = "9600";
                    comMangr.DataBits = "8";
                    comMangr.Parity = "None";
                    //comMangr.DisplayWindow = terminalRichTextBox1
                    comPortCmb.SelectedIndex = 0;
                    comMangr.PortName =
comPortCmb.Items[comPortCmb.SelectedIndex].ToString();
                    comMangr.StopBits = "One";
                    //comMangr.SignalStrengthBar = SigStrProgressBar1
```

```
                // comMangr.SignalStrengthLbl = signaStrDb
                comMangr.CurrentTransmissionType = CommManager.TransmissionType.Hex;
                System.Diagnostics.Process.Start("C:\\\\ft245RL_Init.exe");

                System.Threading.Thread.Sleep(5000);


        } catch (Exception ex) {
                MessageBox.Show(ex.Message);

        }
        Button2.Enabled = false;
        Button3.Enabled = false;
        Button4.Enabled = false;
        Button5.Enabled = false;
        Button6.Enabled = false;
        Button7.Enabled = false;
        Button8.Enabled = false;
        Button9.Enabled = false;
        Button10.Enabled = false;


}

private void Button1_Click(System.Object sender, System.EventArgs e)
{
        comMangr.OpenPort();
        Button2.Enabled = true;
        Button3.Enabled = true;
        Button4.Enabled = true;
        Button5.Enabled = true;
        Button6.Enabled = true;
        Button7.Enabled = true;
        Button8.Enabled = true;
        Button9.Enabled = true;
        Button10.Enabled = true;
}

 private void Button1_Click_1(object sender, EventArgs e)
 {
     comMangr.OpenPort();
       Button2.Enabled = true;
       Button3.Enabled = true;
       Button4.Enabled = true;
       Button5.Enabled = true;
       Button6.Enabled = true;
       Button7.Enabled = true;
       Button8.Enabled = true;
       Button9.Enabled = true;
       Button10.Enabled = true;
 }

 private void Button2_Click(object sender, EventArgs e)
```

```
        {
            comMangr.WriteData(TextBox1.Text);
        }

        private void Button3_Click(object sender, EventArgs e)
        {
            comMangr.WriteData("02");
        }

        private void Button4_Click(object sender, EventArgs e)
        {
            comMangr.WriteData("00");
        }

        private void Button6_Click(object sender, EventArgs e)
        {
            comMangr.WriteData("08");
        }

        private void Button5_Click(object sender, EventArgs e)
        {
            comMangr.WriteData("00");
        }

        private void Button10_Click(object sender, EventArgs e)
        {
            comMangr.WriteData("20");
        }

        private void Button9_Click(object sender, EventArgs e)
        {
        comMangr.WriteData("00");
        }

        private void Button8_Click(object sender, EventArgs e)
        {
            comMangr.WriteData("80");
        }

        private void Button7_Click(object sender, EventArgs e)
        {
        comMangr.WriteData("00");
        }

        }
    }
```

And create same commanger class as we discussed in RFID DATA READ FROM SERIAL PORT SECTION.

GSM INERFACE

Visit http://researchdesignlab.com/gsm-sim-900.html to buy this product.

There are many different kinds of applications SMS applications in the market today, and many others are being developed. Applications in which SMS messaging can be utilized are virtually unlimited. Some common examples of these are given below:

- Person-to-person text messaging is the most commonly used SMS application, and it is what the SMS technology was originally designed for.
- Many content providers make use of SMS text messages to send information such as news, weather report, and financial data to their subscribers.
- SMS messages can carry binary data, and so SMS can be used as the transport medium of wireless downloads. Objects such as ringtones, wallpapers, pictures, and operator logos can be encoded in SMS messages.
- SMS is a very suitable technology for delivering alerts and notifications of important events.
- SMS messaging can be used as a marketing tool.

In general, there are two ways to send SMS messages from a computer / PC to a mobile phone:

1. Connect a mobile phone or GSM/GPRS modem to a computer / PC. Then use the computer / PC and AT commands to instruct the mobile phone or GSM/GPRS modem to send SMS messages.

2. Connect the computer / PC to the SMS center (SMSC) or SMS gateway of a wireless carrier or SMS service provider. Then send SMS messages using a protocol / interface supported by the SMSC or SMS gateway

AT Commands

AT commands are instructions used to control a modem. AT is the abbreviation of ATtention. Every command line starts with "AT" or "at". That's why modem commands are called AT commands. There are two types of AT commands:

1. Basic commands are AT commands that do not start with a "+". For example, D (Dial), A (Answer), H (Hook control), and O (Return to online data state) are the basic commands.
2. Extended commands are AT commands that start with a "+". All GSM AT commands are extended commands. For example, +CMGS (Send SMS message), +CMGL (List SMS messages), and +CMGR (Read SMS messages) are extended commands.

The FORM DESIGN as show below, Here we using combo box for port selection and textbox for entering mobile number to send sms,and message field to type message and send button.

The complete code as given below, Here we have to create two class 1)sms ,2)program
The class sms will set all pre-requirements in order to send sms,and port values and program class will load the forms and this will initiate the application.

Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

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

        private void btnSend_Click(object sender, EventArgs e)
        {
            SMS sm = new SMS(cboPorts.Text);
            sm.Opens();
            sm.sendSMS(txtPhone.Text, txtMessage.Text);
            sm.Closes();
            MessageBox.Show("Message Sent!");
        }
        private void loadPorts()
        {
            string[] ports = SerialPort.GetPortNames();
            foreach (string port in ports)
            {
                cboPorts.Items.Add(port);
            }


        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
```

```
namespace SendSMS
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Sms.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.IO.Ports;
using System.Windows.Forms;
namespace SendSMS
{
    class SMS
    {
        SerialPort serialPort;
        public SMS(string comPort)
        {
            this.serialPort = new SerialPort();
            this.serialPort.PortName = comPort;
            this.serialPort.BaudRate = 9600;
            this.serialPort.Parity = Parity.None;
            this.serialPort.DataBits = 8;
            this.serialPort.StopBits = StopBits.One;
            this.serialPort.Handshake = Handshake.RequestToSend;
            this.serialPort.DtrEnable = true;
            this.serialPort.RtsEnable = true;
            this.serialPort.NewLine = System.Environment.NewLine;
        }
        public bool sendSMS(string cellNo, string sms)
        {
            string messages = null;
            messages = sms;
```

```
            if (this.serialPort.IsOpen == true)
            {
                try
                {
                    this.serialPort.WriteLine("AT" + (char)(13));
                    Thread.Sleep(4);
                    this.serialPort.WriteLine("AT+CMGF=1" + (char)(13));
                    Thread.Sleep(5);
                    this.serialPort.WriteLine("AT+CMGS=\"" + cellNo + "\"");
                    Thread.Sleep(10);
                    this.serialPort.WriteLine(">" + messages + (char)(26));
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Source);
                }
                return true;
            }
            else
            return false;
        }


        public void Opens()
        {
            if (this.serialPort.IsOpen == false)
            {
                this.serialPort.Open();
            }
        }

        public void Closes()
        {
            if (this.serialPort.IsOpen == true)
            {
                this.serialPort.Close();
            }
        }
    }
}
```