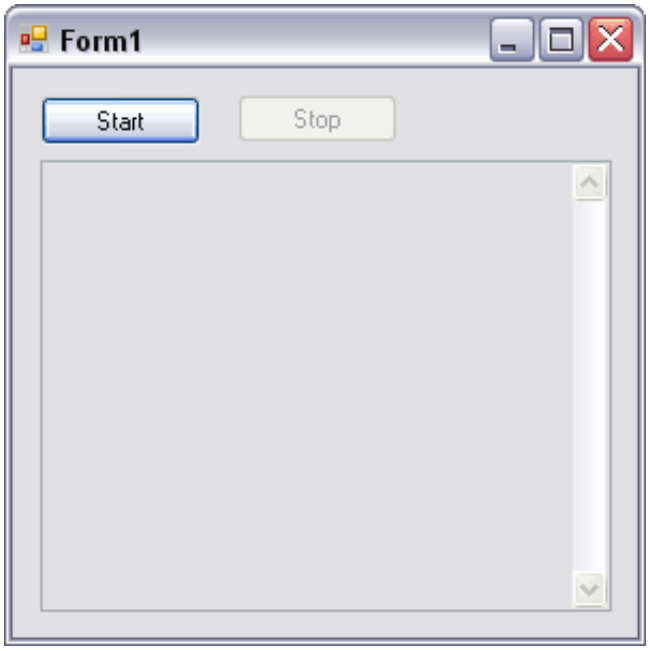
Simple Serial Communication with Microsoft Visual C# Express

Introduction

Here's a simple Windows serial communication program you can write yourself in 10 minutes. Microsoft's FREE Visual C# Express with the .Net Framework now includes a serial port class that eliminates the cumbersome setup for threads and overlapped I/O. They've made serial communication easy again. This project can readily be adapted to a variety of applications requiring serial communication.

The example has 2 buttons (Start and Stop) and a TextBox. When the application is running, click Start to open a COM port. Once the port is open, incoming serial data will appear in the TextBox. Characters typed into the TextBox will be transmitted out the serial port. Click Stop to close the COM port. That's it. The example code is explained below. You can either follow along and write your own version or simply download the Visual C# Express Simple Serial project directly. A complete code listing appears at the end of this page.

Create the Application



Open Visual C# Express and select File->New Project. Chose the "Windows Application" icon and name the new project "Simple Serial" (the Project Name edit box is at the bottom of dialog). This will create a blank Form1 and skeleton code for the application.

From the Toolbox palette, add two Buttons to Form1. Select button1. In the Properties palette change the **button1** Name to **buttonStart**. Change its its Text property to **Start**. Name the other button **buttonStop**. Change its Text property to **Stop** and set its Enable property to **False**.

Add a TextBox to Form1. Change its Multiline property to **True**, ReadOnly property to **True**, and Scrollbars property to **Vertical**. Resize the Textbox and arrange the Buttons to look like the image here.

Last, add a SerialPort class from the Toolbox palette to Form1. Since it's a non-visual class, its icon does not stay on Form1 but appears at the bottom of the Form1 design page.

Select the Start button in Form1. In the Properties palette, select the Events icon. Find the event labled "Click" and double-click it. This will create skeleton code for the Start button "Click" event-handler. Leave it empty for now. We'll add our code to it shortly. Go back to the design tab, select the Stop button and add a "Click" event-handler for it just like you did for the Start button. Using the same technique, add a "KeyPress" event-handler to the TextBox, a "DataReceived" event-handler to the SerialPort, and a "FormClosing" event-handler to Form1 (be sure to select FormClosing... not FormClosed).

Now, cut and paste code from the event handlers in the listing below into the empty event handlers in your project. Cut and paste the variable declaration for **RxString**. Be sure to place it immediately before the Form1( ) function as shown in the listing. Last, cut and paste the entire DisplayText( ) function. That's it. You're done.

Test the Application

In the Visual C# Express IDE, press F5 (Run) or the green-arrow Run button. With the Simple Serial application running, press the Start button to open the COM port. Incoming serial text data will appear in the textBox1. Anything you type in textBox1 will be transmitted. If you don't have a connected serial device suitable for testing the program, use a loopback connector. Make one by simply jumpering pins 2 and 3 of a serial cable or DB9F connector. With a loopback connector, anything you type in textBox1 will be echoed back through the port and displayed in textBox1. Press the Stop button to close the port.

If the program doesn't work, carefully review your code and serial connections. In the code for buttonStart_Click( ), be sure the COM port you specify is the one you're connected to. It must physically exist on your computer. COM1 is most common but you might be connected to COM2, COM3, etc. If the port is connected to an external serial device, the comm parameters (baud rate, etc.) of the device and your program must agree. Also note that there is no serial handshaking in our application. An external device might require it. Change parameters if needed.

How it Works

All the heavy lifting is performed behind the scenes thanks to the SerialPort component. The rest is pretty simple. The only tricky part is the Invoke method that allows our DisplayText( ) function to update textBox1. Let's look at the code for each event handler.

In buttonStart_Click( ), we begin by setting the port name and baud rate. In our example PortName is COM1 but you can set it to any other port available on your computer. Notice the PortName is a string and must be in quotes. The baud rate must agree with the baud rate of whatever is on the other end of the serial connection. We then call the Open( ) function. If the port opened OK, we disable the Start button, enable the Stop button, and allow writing in textBox1.

```
private void buttonStart_Click(object sender, EventArgs e)
{
    serialPort1.PortName = "COM1";
    serialPort1.BaudRate = 9600;

    serialPort1.Open();
    if (serialPort1.IsOpen)
    {
        buttonStart.Enabled = false;
        buttonStop.Enabled = true;
        textBox1.ReadOnly = false;
    }
}
```

Once the serialPort1 is open, any incoming serial characters will cause a DataReceived event to fire. Inside the event handler we read all existing characters from the internal serial receive buffer into string RxString. *The next part is critical and not obvious.* serialPort1 runs in it own separate thread behind the scenes. This thread cannot directly call any functions in the main thread of our application. However, a special function, Invoke( ), will allow it. So we use Invoke to call our DisplayText( ) function. RxString is the global string variable accessable by both threads.

```
 private void serialPort1_DataReceived
   (object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
 {
     RxString = serialPort1.ReadExisting();
     this.Invoke(new EventHandler(DisplayText));
 }
```

Our DisplayText( ) function is simple. We just append the text in RxString to whatever is already in textBox1.

```
private void DisplayText(object sender, EventArgs e)
{
    textBox1.AppendText(RxString);
}
```

The textbox1 KeyPress( ) function captures characters typed into textBox1 and writes them to serialPort1. Write( ) can only send characters from a char type array so we declare one with a length of [1] and assign the KeyChar value to it. With the arguments in Write( ), we tell it to send the characters in the buff, offset of 0 chars into the array, and a length of 1 char. We set the event to "Handled" to prevent the typed character from appearing in textBox1. If you want it to appear (local echo), omit the line.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    // If the port is closed, don't try to send a character.
    if(!serialPort1.IsOpen) return;

    // If the port is Open, declare a char[] array with one element.

    char[] buff = new char[1];

    // Load element 0 with the key character.
    buff[0] = e.KeyChar;

    // Send the one character buffer.
    serialPort1.Write(buff, 0, 1);

    // Set the KeyPress event as handled so the character won't
    // display locally. If you want it to display, omit the next line.

    e.Handled = true;
}
```

Clicking the Stop button calls buttonStop_Click( ). If serialPort1 is open, we close the port and set the button enables and textBox1 ReadOnly state back to their previous value.

```
private void buttonStop_Click(object sender, EventArgs e)
{
     if (serialPort1.IsOpen)
     {
         serialPort1.Close();
         buttonStart.Enabled = true;
         buttonStop.Enabled = false;
         textBox1.ReadOnly = true;
     }
}
```

Last but not least, if the application is closed while serialPort1 is open, the port must be closed first or the program will hang.

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
```

```
    if (serialPort1.IsOpen) serialPort1.Close();
}
```

Final Thoughts

For the sake of clarity and simplicity, our example application omits a few things.

- textBox1 cannot properly display a CR without a LF. TextBoxes have the quirky need for a CR-LF pair and they must be in that order. If a CR arrives by itself, you'll need to add the LF.
- textBox1 holds a lot of characters but will eventually run out of memory and just stop displaying anything new that arrives. You can limit the number of characters it will display by setting the textBox1 MaxLength property but then it will just run out of space sooner. A better approach is to limit the number of lines without affecting the display.
- Our program has no handshaking. Some devices require it.

You are free to use all the information presented here. There are no restrictions. There's no support available but the program is simple enough that you probably won't need it. Send comments to csharpcomments@simpleserial.com

Links

SimpleSerialCS.zip Download Simple Serial Project for Visual C# Express
Microsoft MSDN Serial Communication Resources WIN32 API serial communication reference.
www.microsoft.com/express/vcsharp/ Microsoft's FREE Visual C# Express

Donations via PayPal are optional. Thank you.

Full Code Listing

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

namespace SimpleSerial
{
    public partial class Form1 : Form
    {
        // Add this variable

        string RxString;

        public Form1()
        {
            InitializeComponent();
        }

        private void buttonStart_Click(object sender, EventArgs e)
        {
```

```
            serialPort1.PortName = "COM1";
            serialPort1.BaudRate = 9600;

            serialPort1.Open();
            if (serialPort1.IsOpen)
            {
                buttonStart.Enabled = false;
                buttonStop.Enabled = true;
                textBox1.ReadOnly = false;
            }
        }

        private void buttonStop_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Close();
                buttonStart.Enabled = true;
                buttonStop.Enabled = false;
                textBox1.ReadOnly = true;
            }

        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (serialPort1.IsOpen) serialPort1.Close();
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            // If the port is closed, don't try to send a character.

            if(!serialPort1.IsOpen) return;

            // If the port is Open, declare a char[] array with one element.
            char[] buff = new char[1];

            // Load element 0 with the key character.

            buff[0] = e.KeyChar;

            // Send the one character buffer.
            serialPort1.Write(buff, 0, 1);

            // Set the KeyPress event as handled so the character won't
            // display locally. If you want it to display, omit the next line.
            e.Handled = true;
        }

        private void DisplayText(object sender, EventArgs e)
        {
            textBox1.AppendText(RxString);
        }

        private void serialPort1_DataReceived
          (object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            RxString = serialPort1.ReadExisting();
            this.Invoke(new EventHandler(DisplayText));
        }
    }
}
```