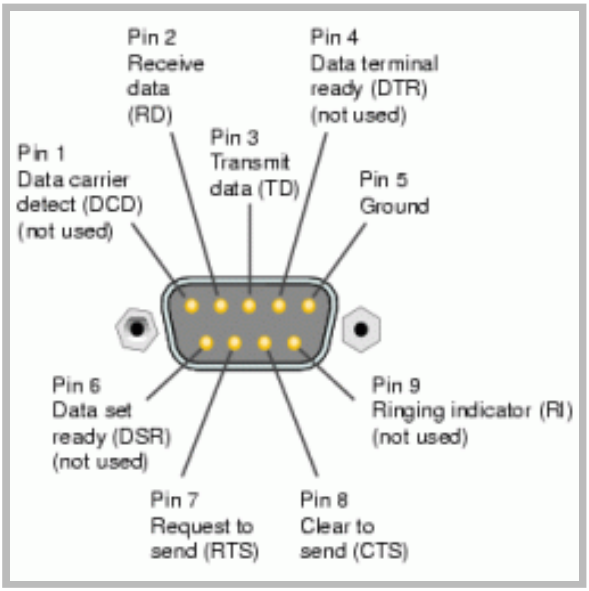
How To Work With C# Serial Port Communication

MD ALI AHSAN RANA FEBRUARY 7, 2013 37 COMMENTS



In today's programming tutorial, I am going to describe some basics about how we can perform serial port communication from our C#.NET applications. Serial communications can be done via either direct to physical serial port connected to the computer or via a USB to serial converter interface. If the device do require a serial port and your computer don't have any, you can make use of such converters easily.

This type of communication aren't as much easy as other similar tasks such as working with logic drive on computer via c# and need use of specific kind of communication protocol.

One interesting thing that you might need to remember that, when the physical serial port are being used, it doesn't have any PID or VID. But if you are using any specific type of devices which facilitate this kind of communication via USB interface, you can retrieve their PID/VID respectively and communicate accordingly. .NET has a very useful internal classes which can make this kind of communication to be very easy and efficient. Lets have a look into them.

Retrieve List Serial Ports:

OK, lets first see whether we can detect the serial ports from within our application. As a prerequisite, you need to make sure that, while the application is running, the windows user must need to have access to the ports. The following C# code examples will return a list of Serial port names connected to the computer:

```
1  public List<string> GetAllPorts()
2  {
3       List<String> allPorts = new List<String>();
4       foreach (String portName in System.IO.Ports.SerialPort.GetPortNames
5       {
6           allPorts.Add(portName);
7       }
8       return allPorts;
9  }
```

And it is enough for further processing. .NET can understand where to communicate via the port name in string like "COM1", "COM2" etc.

Using WMI query:

The following code snippet will work similarly as the one given above, but it make use of core WMI and returns a list of Management objects:

```
private List<ManagementObject> getAllComPort()
{
     List<ManagementObject> objct = new List<ManagementObject>();
     using (ManagementObjectSearcher searcher = new ManagementObjectSea
     WIN_SERIAL_OBJECT_NAME + ""))
     {
         foreach (ManagementObject serialPortObj in searcher.Get())
         {
             objct.Add(serialPortObj);
         }
     }
     return objct;
}
```

Open Or Close Serial Ports:

well, as have now been able to get the list of ports, now we can start communicating. First step to start serial port communication is to open the port, then send/receive necessary data and finally close the port(s). Lets see an example how we can open and close ports:

```
System.IO.Ports.SerialPort myPort = new System.IO.Ports.SerialPort("COM
if (myPort.IsOpen == false) //if not open, open the port
   myPort.Open();
//do your work here
myPort.Close();
```

Read/Write Data via Serial Port Communication:

OK, now we can start doing the real communication. However, it is very important that, you have prior knowledge what kind of data the connected device is expecting. and you will need to process their response as well to understand what they are saying. For this, you will need the corresponding firmware API command lists. Here, I will give a simple prototype how the send/receive data workflow will be:

```
using System;
using System.Timers;
    public class CommTimer
    {
                public  Timer tmrComm = new Timer();
        public bool timedout = false;
        public CommTimer()
        {
            timedout = false;
                        tmrComm.AutoReset = false;
            tmrComm.Enabled = false;
            tmrComm.Interval = 1000; //default to 1 second
            tmrComm.Elapsed += new ElapsedEventHandler(OnTimedCommEvent
        }

```

```
16          public void OnTimedCommEvent(object source, ElapsedEventArgs e)
17          {
18             timedout = true;
19                     tmrComm.Stop();
20          }
21
22          public void Start(double timeoutperiod)
23          {
24             tmrComm.Interval = timeoutperiod;              //time to time
25                     tmrComm.Stop();
26             timedout = false;
27                     tmrComm.Start();
28          }
29
30
31      }
32
33  public void SendReceiveData()
34  {
35        byte[] cmdByteArray = new byte[1];
36        SerialObj.DiscardInBuffer();
37        SerialObj.DiscardOutBuffer();
38
39        //send
40        cmdByteArray[0] = 0x7a;
41        SerialObj.Write(cmdByteArray, 0, 1);
42
43        CommTimer tmrComm = new CommTimer();
44        tmrComm.Start(4000);
45        while ((SerialObj.BytesToRead == 0) && (tmrComm.timedout == false
46        {
47            Application.DoEvents();
48        }
49        if (SerialObj.BytesToRead > 0)
50        {
51            byte[] inbyte = new byte[1];
52            SerialObj.Read(inbyte, 0, 1);
53            if (inbyte.Length > 0)
54            {
55                byte value = (byte)inbyte.GetValue(0);
56                //do other necessary processing you may want.
57            }
58        }
59        tmrComm.tmrComm.Dispose();
60        SerialObj.DiscardInBuffer();
61        SerialObj.DiscardOutBuffer();
62        SerialObj.Close();
63  }
```

First thing we are doing here, is discarding existing buffer, if any. Then, we will write an array of bytes to the port. This array can contain several hex values to represent a single command. Here, I have used one . After writing, and before you start reading the response, it's always good to wait for a while, thus add a slight delay, which helps to make up the time required between receiving and sending reply for the device. In this time, normally, windows do ques your work instruction and sends to devices. But, it may not happen because of CPU scheduling issue etc. So, better to check

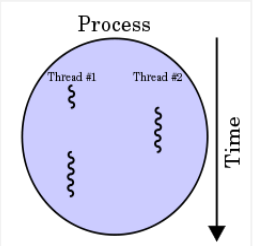
whether any response came or not. If not, force windows to perform this action now by 'Application.DoEvents()' command statement.

References:

For working more with deep communication and troubleshoot, you will need to study carefully the Microsoft's official documentation on serial object class.
Hope this small tutorial on serial port communication with c# will be helpful to you in some extent. Let me know if you want some more similar tutorials or have any questions. Happy coding :).

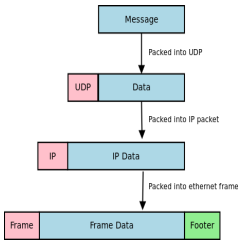
Share If Liked

Related Tutorials:

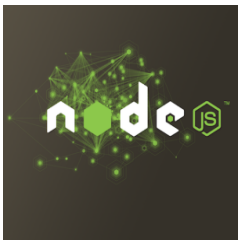


How To Work With

How To Work With LinQ C#



Getting Started With UDP



How To Work With JSON In

FILED UNDER: PROGRAMMING

TAGGED WITH: .NET, C#

About Md Ali Ahsan Rana

Rana is a passionate software engineer/Technology Enthusiast.
Twitter: @ranacseruet
Github: ranacseruet

Comments