What is the correct way to read a serial port using .NET framework?

up vote 7 down vote 4

I've read a lot of questions here about how to read data from serial ports using the .NET SerialPort class but none of the recommanded approaches have proven completely efficient for me.

Here is the code I am using for now:

```
SerialPort port = new SerialPort("COM1");

port.DataReceived += new
SerialDataReceivedEventHandler(MyDataReceivedHandler);
```

And the event handler:

```
void MyDataReceivedHandler(object sender, SerialDataReceivedEventArgs
e)

{

    int count = port.BytesToRead;

    byte[] ByteArray = new byte[count];

    port.Read(ByteArray, 0, count);

}
```

But I am still missing some data sometimes. I've tried different way of reading the data in the event handler but with no luck.

As the .NET 4.5 brings new possibilities to do some asynchronous tasks, like with the ReadAsync method that seems to be useable on a SerialPort stream, I'm curious to see what would be the recommended approach to handle those cases.

c# serial-port .net-4.5

share improve this question

asked Dec 6 '12 at 23:45

Yannick Blondeau
5,47952047

3 Your question is really vague. "I am still missing some data sometimes" is like telling your mechanic "my car makes a funny noise sometimes" and asking how much it will cost to fix. Perhaps this answer to a related question might help. However, I think you need to be a lot more specific if you want help; as it is, there's not a lot that can't be answered by looking at other questions on this topic. – Ken White Dec 6 '12 at 23:59

2

You're right, I will put a real example together to better illustrate what is happening... – Yannick Blondeau Dec 7 '12 at 0:13

Implementing a serial communication protocol over RS232 or 485 is a really, really hard task. It requires a LOT of experience. I've been implementing this since good old DOS days and still get stuck in some common pitfalls. Why don't you try to find a reliable third-party offering for dealing with serial communications? They will have probably ironed out all the many, many, many bugs you'll fall into yourself :-) If this is an exercise, then go ahead, it giver great satisfaction to finally master serial communications! :-) – Loudenvier Dec 7 '12 at 0:39

add a comment |

2 Answers

up vote 6 down vote

Could you try something like this for example I think what you are wanting to utilize is the **port.ReadExisting()** Method

```
class SerialPortProgram

{

 // Create the serial port with basic settings

   private SerialPort port = new   SerialPort("COM1",

     9600, Parity.None, 8, StopBits.One);

   [STAThread]

   static void Main(string[] args)

   {

     // Instatiate this

     class new SerialPortProgram();

   }



   private SerialPortProgram()

   {

       Console.WriteLine("Incoming Data:");

        // Attach a method to be called when there

        // is data waiting in the port's buffer

       port.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);

       // Begin communications
```

```
        port.Open();

        // Enter an application loop to keep this thread alive

        Application.Run();

     }


    private void port_DataReceived(object sender,
SerialDataReceivedEventArgs e)

    {

       // Show all the incoming data in the port's buffer

       Console.WriteLine(port.ReadExisting());

    }

}
```

Or is you want to do it based on what you were trying to do , you can try this

```
public class MySerialReader : IDisposable

{

  private SerialPort serialPort;

  private Queue<byte> recievedData = new Queue<byte>();


  public MySerialReader()

  {

    serialPort = new SerialPort();

    serialPort.Open();

    serialPort.DataReceived += serialPort_DataReceived;

  }


  void serialPort_DataReceived(object s, SerialDataReceivedEventArgs
e)

  {

    byte[] data = new byte[serialPort.BytesToRead];
```

```
    serialPort.Read(data, 0, data.Length);

    data.ToList().ForEach(b => recievedData.Enqueue(b));

    processData();

  }


  void processData()

  {

    // Determine if we have a "packet" in the queue

    if (recievedData.Count > 50)

    {

        var packet = Enumerable.Range(0, 50).Select(i =>
recievedData.Dequeue());

    }

  }


  public void Dispose()

  {

        if (serialPort != null)

        {

            serialPort.Dispose();

        }

  }
```

shareimprove this answer edited Dec 7 '12 at 0:38

answered Dec 7 '12 at 0:27

MethodMan
12k31538

1 Thanks for your help, I finally went with `ReadExisting` used with a blocking collection. I'm still planning to try the new `ReadAsync` function though... – Yannick Blondeau Dec 11 '12 at 13:14

don't the processData() method requires a mutex? I mean, the datareceived event could trigger again when the processData() is still running? isn't it? So we could end up with two (or even more?) threads running inside the processData() method.... – Gianluca Ghettini Sep 10 '14 at 8:22

I have not tried it to that extent and based on what Yannick mentioned in his comment above I am thinking that it would not make a difference if he were to proceed with reading Asynchronously – MethodMan Sep 10 '14 at 14:21

add a comment |

up vote 1 down vote

I used similar code to @MethodMan but I had to keep track of the data the serial port was sending and look for a terminating character to know when the serial port was done sending data.

```
private string buffer { get; set; }

private SerialPort _port { get; set; }



public Port()

{

    _port = new SerialPort();

    _port.DataReceived += new
SerialDataReceivedEventHandler(dataReceived);

    buffer = string.Empty;

}



private void dataReceived(object sender, SerialDataReceivedEventArgs
e)

{

    buffer += _port.ReadExisting();



    //test for termination character in buffer

    if (buffer.Contains("\r\n"))

    {

        //run code on data received from serial port

    }

}
```

shareimprove this answer