COM Port Changes in .NET Micro Framework 4.0

Author:
Mr. John R. Malin

Microsoft Embedded MVP
Principal Embedded Developer, SJJ Embedded Micro Solutions, LLC

Date:
April 14, 2010

Applies to:
Microsoft® .NET Micro Frame 4.0

Summary:
This white paper discusses the differences that were made to the COM port's Classes in the .NET Micro Framework 4.0 release. Changes were made to the COM port API in .NET Micro Framework 4.0 to bring the managed code programming experience more in line with the bigger Windows® .NET Framework. These changes require some modifications to the references required and the using declarations in the source modules.

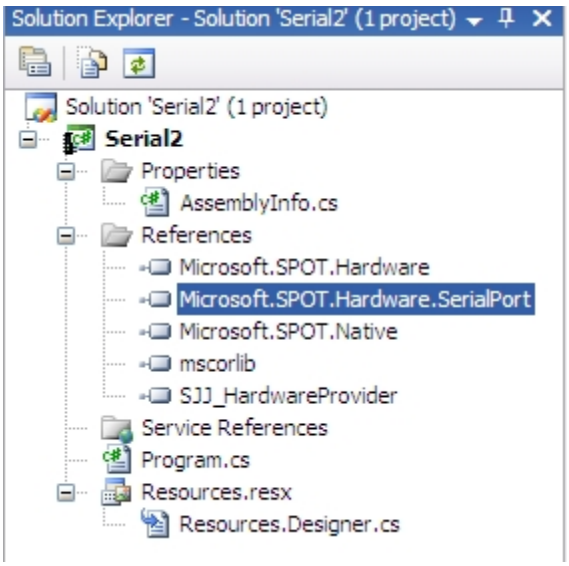
Contents

Introduction

One of the design goals of the .NET Micro Framework was to provide a subset of .NET Framework API's that would allow managed code application that were written for the Windows desktop to be easily port to the .NET Micro Framework environment with minimal modification of the code. In order to bring the .NET Micro Framework COM port programming experience more in line with .NET Framework, there were modifications made to the COM port API's in the .NET Micro Framework 4.0 Release. These modifications are clarified in the following topics.

COM Port References

The COM Port interface is now contained in a new assembly, Microsoft.SPOT.Hardware.SerialPort, which must be added to your managed code solution in Visual Studio or Visual Studio Express.



COM Port Using directives

The namespace for the COM port API's has also changed and the naming conventions between the assembly and the namespace may be confusing. The old COM port namespace was Microsoft.SPOT.Hardware. The new COM port namespace is System.IO.Ports. To easily access the COM port API's add the "using System.IO.Ports" statement to the top or your source module:

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.Hardware.SJJ;
using System.Threading;
using System.IO.Ports;
using System.Text;
```

Using the New Classes

The serial configuration class is no longer used, so if you are porting over a managed code application from an earlier version of the .NET Micro Framework, comment or delete those statements. Here is the old serial object creation:

```
static SerialPort.Configuration mySerial2Config = new
SerialPort.Configuration(SerialPort.Serial.COM2, SerialPort.BaudRate.Baud115200,
false);
```

public SerialPort mySerial2 = new SerialPort(mySerial2Config);

The SerialPort class itself now has overloaded constructors that accept the configuration arguments directly. You must at least specify the COM port that you are referencing, but baud rate, parity, data bits, and stop bits can be omitted and the overloaded versions of the class with assign default values. Check with the manufacturer of the specific .NET Micro Framework port that you are using for the default values that you can expect.

Here are some examples of using the new SerialPort class and overloaded constructors:

SerialPort mySerialPort = new SerialPort("COM2");

SerialPort mySerialPort = new SerialPort("COM2", BaudRate.Baudrate115200;

SerialPort mySerialPort = new SerialPort("COM2", BaudRate.Baudrate115200, Parity.None, 8, StopBits.None);

There are defined parameters for the allowed baud rates, the parity values, and the stop bits. The hardware manufacturer may also provide some replacement definitions that are specific to the hardware and the .NET Micro Framework port that you are using. Note that the integer values for the baud rate, parity, etc, can always be used as long as they are a supported value. The serial port identifier is now the string that names the serial port, i.e. "COM1", "COM2", etc. Again, check with your manufacturer for the number of COM ports that are supported on your hardware platform.

The handshake mode cannot be set up in the constructor. You can specify the handshake mode via SerialPort Handshake property:

mySerialPort.Handshake = Handshake.None;

Before writing to or reading from the serial port, it now must be explicitly opened:

mySerialPort.Open();

The Write & Read method now has 3 parameters: pointer to the byte buffer, the offset into the buffer to start at, and the number of bytes to write or read, i.e.:

mySerialPort.Write(WriteBuffer, 0, 4);
mySerialPort.Read(ReadBuffer, 0, ReadBuffer.Length);

When you are through with the serial port, do an explicit close:

```
mySerialPort.Close();
```

Conclusion

SerialPort objects can be created and used in the .NET Micro Framework in much the same way that they are created in used in Window® .NET Framework, once the right assembly is added to the Visual Studio solution and the appropriate using directive is added to the code. The SerialPort class constructor provides initialization of most of the key serial port parameters, and once created, the class provides methods to open, read, write, and close.