NRC·CNRC

NRC Publications Archive
Archives des publications du CNRC

Inter-computer communications

Senior, D.

For the publisher's version, please access the DOI link below./ Pour consulter la version de l'éditeur, utilisez le lien DOI ci-dessous.

Publisher's version / Version de l'éditeur:
https://doi.org/10.4224/8895746
Student Report (National Research Council of Canada. Institute for Ocean Technology); no. SR-2005-15, 2005

NRC Publications Archive Record / Notice des Archives des publications du CNRC :
https://nrc-publications.canada.ca/eng/view/object/?id=ed9f1e21-d913-43f2-966e-4b529ee23f0b
https://publications-cnrc.canada.ca/fra/voir/objet/?id=ed9f1e21-d913-43f2-966e-4b529ee23f0b

Access and use of this website and the material on it are subject to the Terms and Conditions set forth at
https://nrc-publications.canada.ca/eng/copyright
READ THESE TERMS AND CONDITIONS CAREFULLY BEFORE USING THIS WEBSITE.

L'accès à ce site Web et l'utilisation de son contenu sont assujettis aux conditions présentées dans le site
https://publications-cnrc.canada.ca/fra/droits
LISEZ CES CONDITIONS ATTENTIVEMENT AVANT D'UTILISER CE SITE WEB.

Questions? Contact the NRC Publications Archive team at PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca. If you wish to email the authors directly, please see the first page of the publication for their contact information.

Vous avez des questions? Nous pouvons vous aider. Pour communiquer directement avec un auteur, consultez la première page de la revue dans laquelle son article a été publié afin de trouver ses coordonnées. Si vous n'arrivez pas à les repérer, communiquez avec nous à PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca.

National Research Council Canada | Conseil national de recherches Canada

Canada

DOCUMENTATION PAGE

REPORT NUMBER	NRC REPORT NUMBER	DATE
SR-2005-15		August 2005
REPORT SECURITY CLASSIFICATION		**DISTRIBUTION**
Unclassified		Unlimited

TITLE

INTER-COMPUTER COMMUNICATIONS

AUTHOR(S)

Dena Senior

CORPORATE AUTHOR(S)/PERFORMING AGENCY(S)

Institute for Ocean Technology, National Research Council, St. John's, NL

PUBLICATION

SPONSORING AGENCY(S)

Institute for Ocean Technology, National Research Council, St. John's, NL

IOT PROJECT NUMBER	NRC FILE NUMBER
42_10_18	

KEY WORDS	PAGES	FIGS.	TABLES
Bluetooth, spread spectrum, modems	7, App. A-E		

SUMMARY

This report is intended as a guide through testing wireless communications using Bluetooth and Spread Spectrum modems. Included is a description of software applications developed, testing and analysis of data, and recommendations.

ADDRESS National Research Council
Institute for Ocean Technology
Arctic Avenue, P. O. Box 12093
St. John's, Newfoundland, Canada
A1B 3T5
Tel.: (709) 772-5185, Fax: (709) 772-2462

National Research Council Canada
Institute for Ocean Technology

Conseil national de recherches Canada
Institut des technologies océaniques

INTER-COMPUTER COMMUNICATIONS

SR-2005-15

Dena Senior

August 2005

TABLE OF CONTENTS

1.0 INTRODUCTION

Communication problems were encountered during testing in the OEB. The model did not respond properly when Bluetooth modems were in use. The problem appeared to come from timing issues with the modems.

Thus, investigation into the cause of the delays and methods to reduce time delays was necessary. This involved testing Bluetooth and Spread Spectrum modems using different software applications.

2.0 SOFTWARE APPLICATIONS

Two software applications were used:

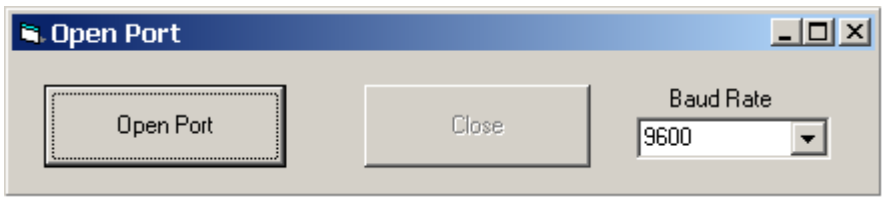
-Visual Basic .NET using WIN32 API

-Visual Basic 6 using MSComm Control

Both applications use similar timing routines and program structures. Two programs were produced for each application. One to act as a loop, “Open Port”, and the other, “Send Data” to send, receive, and process data.

The Visual Basic .NET application uses a serial port library written in C#. The library is available at www.openNetcf.org.

2.1 Open Port Program



Open Port program allows the user to open, close, and change the baud rate of a communication port.

The program is currently set to communicate with COM1. To change this setting, open the VB project or VB .NET solution, whichever application is applicable, and change the comm port number in the form_load sub.

NOTE: The combo box changes the baud rate of the comm port only. It does not change the baud rate of the modems. The comm port and modem baud rates must match.

To change the baud rate of the Spread Spectrum modem:

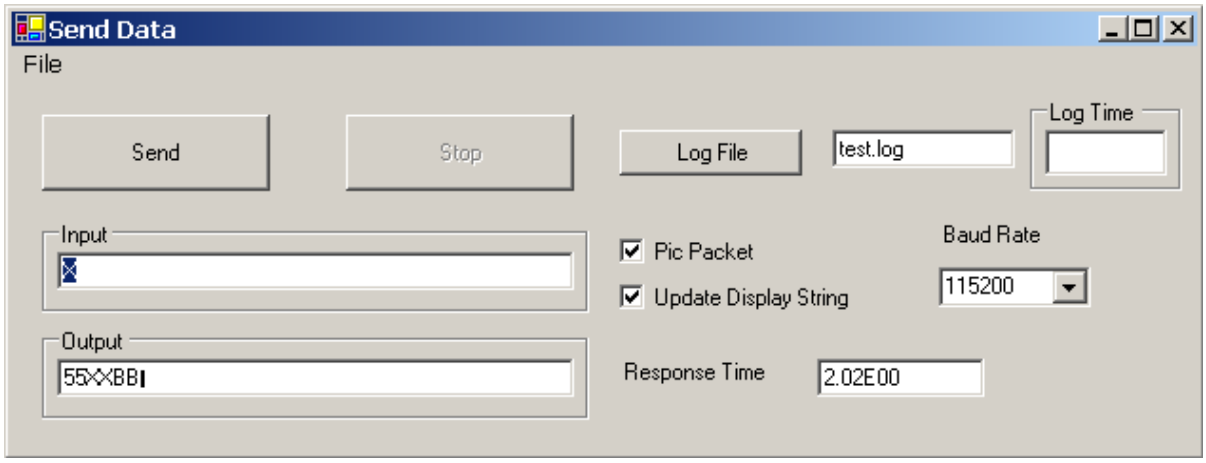
- ⇒ Open HyperTerminal (or a similar terminal application)
- ⇒ Set the comm port baud rate to 9600
- ⇒ Push the white reset button on the modem
- ⇒ While the 3 LED's are flashing, type "mhx"
- ⇒ Type "ATS102=" number code for baud rate "&WA". This sets the modem baud rate and returns the modem to data mode

To change the baud rate of the Bluetooth modem:

- ⇒ Install PromiWIN Software
- ⇒ Choose PromiWin→PromiWin Configuration, set the properties as necessary
- ⇒ Choose the *Device Setting* icon and change the baud rate

For more information, refer to user manuals.

2.2 Send Data Program



Send Data program allows the user to complete a number of tasks:

1. Send either a packet of data or text in the input text window
2. View the data received in the output window
3. Response time is visible and can be logged to a file
4. Baud rate of the comm port can be changed

NOTE: As stated in section 2.1 the combo box, "Baud Rate", does not change the modem baud rate. The modem and comm port baud rates must match.

Response time is calculated by:

Response time = time - (10 * 1.5 / baudrate)

time = (testFinish – testStart)/curFreq

Time starts (**testStart**) as soon as data is sent and stops (**testFinish**) when data is received.

curFreq = current frequency obtained using **QueryPerformanceFrequency**

baudrate = current baud rate setting for communication

(10*1.5/baudrate) = approximate required time to send 1 byte at a specified baud rate.

These calculations occur in the **cmdSend** routine.
A flag (Boolean variable) **m_dataReceived** is used to determine when data is received.

For the MSComm Control (VB 6.0) application, **MSComm_OnComm** routine handles comm events appropriately and handles comm port input.

In the WIN32 (VB .NET) application, **m_port2_DataReceived** handles data.

Refer to appendices for complete Visual Basic and Visual Basic .NET source code.

3.0 TESTING

Testing included:

- ⇒ Null modem (straight cable)
- ⇒ Spread Spectrum modem
- ⇒ Bluetooth modem
- ⇒ Two computers were used. They were both setup in a cubicle approximately 3 feet apart. One computer acted as a loop while the other sent, received, and processed data.
- ⇒ Both Bluetooth and Spread Spectrum modems were set to their default settings. The only setting that changed was their baud rate.
- ⇒ Each modem was tested at different baud rates using both software applications.
- ⇒ A packet of data containing 380 characters was streamed across each connection for 10 minutes.
- ⇒ The response times were logged for each run and analyzed. The average response times for each run was calculated and graphed, see figures.

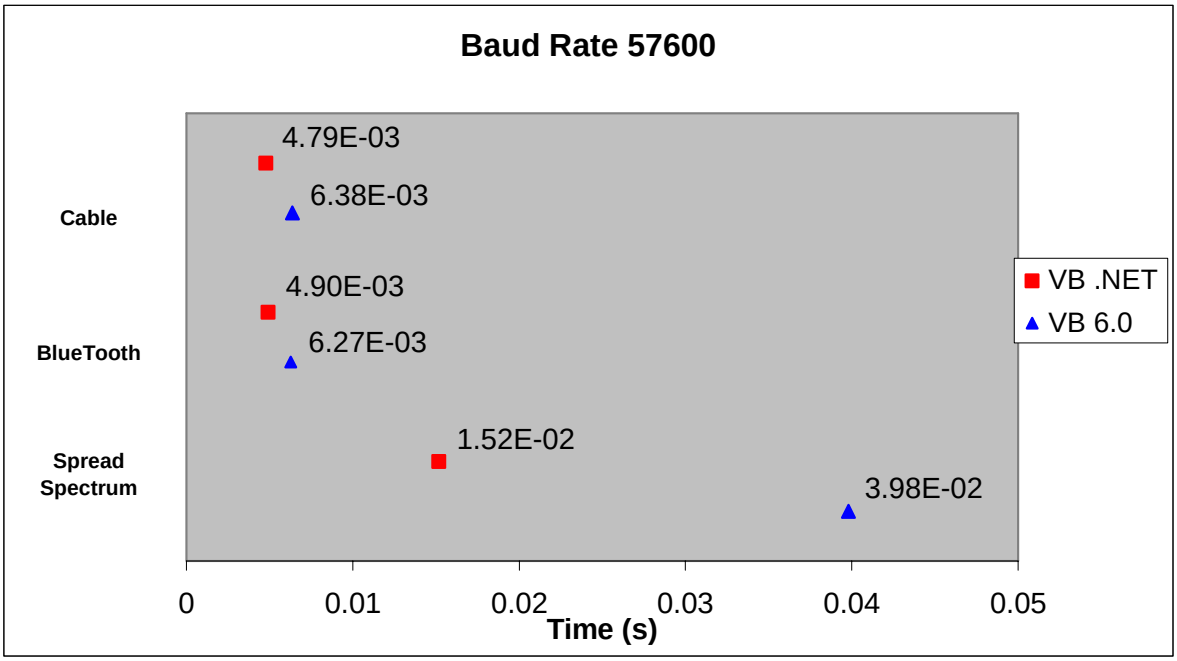


Figure 1

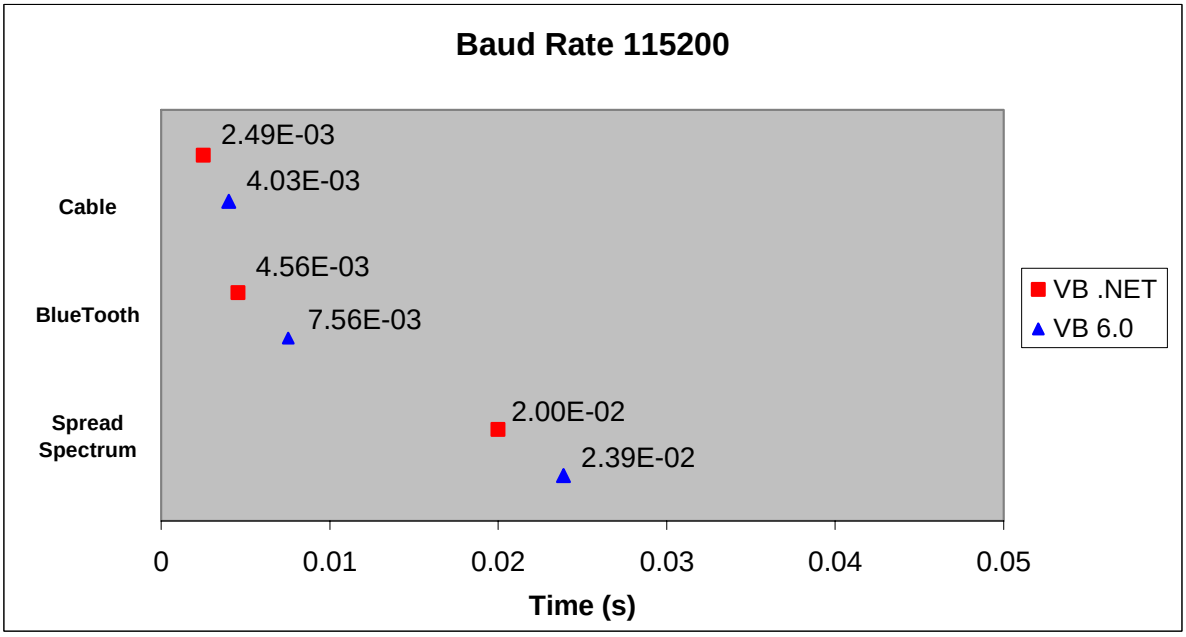


Figure 2

VB .NET: Average response time of logged data for the specified modem using Visual Basic .NET WIN32 application.

VB 6.0: Average response time of logged data for the specified modem using Visual Basic 6.0 MSComm Control application.

It is obvious from both figures that the Visual Basic .NET WIN32 application produced a slightly faster response time for each modem than the MSComm Control application. Bluetooth modems have a much faster response time when compared to Spread Spectrum modems at 115200 baud rate and 57600 baud rate. Bluetooth response times were comparable to the cable (null modem) response times.

It must be noted, however, these are initial results. More testing is needed to justify these findings.

4.0 CONCLUSIONS

Initial results indicate Bluetooth modems have a fast response time over a small area. The Spread Spectrum modems did not perform as well over a small area.

Further investigation into the delay problems encountered in the OEB with the Bluetooth modems is necessary.

5.0 RECOMMENDATIONS

Testing of Bluetooth and Spread Spectrum modems must be expanded including:

- ⇒ Placing the modems further apart. Spread Spectrum modems are designed for use over larger areas, from 200-500ft indoors. This may have been a factor in their poor performance over such a small area.

⇒ Finding optimal settings for the modems in various scenarios. This can potentially help reduce the number of re-transmissions of data and interference from other objects.

⇒ Implementing other software applications such as Visual C and C#. These may reduce the amount of overhead produced from software processing.

⇒ Testing in “noisy” areas to determine how well each type of modem reacts to noise and interference.

Appendix A

Visual Basic .NET Open Port Source Code

```
Imports OpenNETCF.IO.Serial
Imports System.Text

Public Class FrmSerialPortTesting
    Inherits System.Windows.Forms.Form

    Private Declare Function PostMessage Lib "user32" Alias "PostMessageA" _
                                (ByVal hwnd As Int32, _
                                 ByVal wMsg As Int32, _
                                 ByVal wParam As Int32, _
                                 ByVal lParam As Int32) As Int32

    Private Const WM_CLOSE As Int32 = &H10

    Private Const EPSILON As Double = 1.0E-30

    Private WithEvents m_port1 As Port

    Private m_stopSending As Boolean

    Private m_exiting As Boolean


 Windows Form Designer generated code
    Private Sub FrmSerialPortTesting_Load(ByVal sender As System.Object, ByVal e As System. ↙
    EventArgs) Handles MyBase.Load
        m_port1 = New Port("COM1:", CType(New HandshakeCtsRts, DetailedPortSettings))

        Dim baudRate As Integer = Convert.ToInt32(cboSpeed.SelectedItem.ToString())

        m_port1.Settings.BaudRate = baudRate

        m_port1.Settings.ByteSize = 8

        m_port1.Settings.Parity = Parity.none

        m_port1.Settings.StopBits = StopBits.one

        m_port1.SThreshold = 1

        m_port1.RThreshold = 1

        m_port1.InputLen = 0

        m_stopSending = False

        m_exiting = False
    End Sub

    Private Sub FrmSerialPortTesting_Closing(ByVal sender As Object, ByVal e As System. ↙
    ComponentModel.CancelEventArgs) Handles MyBase.Closing

        m_exiting = True

        m_port1.Close()

        m_port1.Dispose()

        m_port1 = Nothing
    End Sub

    Private Sub cmdopen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
    Handles cmdOpen.Click

        cmdOpen.Enabled = False

        cmdStop.Enabled = True

        cboSpeed.Enabled = False
```

```
        m_port1.Open()
    End Sub

    Private Sub cmdStop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles cmdStop.Click
        m_stopSending = True

        m_port1.Close()

        cboSpeed.Enabled = True

        cmdStop.Enabled = False
        cmdOpen.Enabled = True
    End Sub

    Private Sub Reopen()
        m_port1.Close()

        Dim baudRate As Integer = Convert.ToInt32(cboSpeed.SelectedItem.ToString())

        m_port1.Settings.BaudRate = baudRate

        m_port1.Open()
    End Sub

    Private Sub cboSpeed_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles cboSpeed.SelectedIndexChanged
        If (Not IsNothing(m_port1)) Then
            Reopen()
        End If
    End Sub

    Private Sub m_port1_DataReceived() Handles m_port1.DataReceived
        Dim bytesReceived As Long = m_port1.InBufferCount

        Dim bytes As Byte() = m_port1.Input
        m_port1.Output = bytes

    End Sub

    Private Sub mainMenuFileExit_Click(ByVal sender As System.Object, ByVal e As System.
    EventArgs) Handles mainMenuFileExit.Click
        Close()
    End Sub

End Class
```

Appendix B

Visual Basic .NET Send Data Source Code

```
Imports OpenNETCF.IO.Serial
Imports System.Text

Public Class FrmSerialPortTesting
    Inherits System.Windows.Forms.Form

    Private Declare Function PostMessage Lib "user32" Alias "PostMessageA" _
                                (ByVal hwnd As Int32, _
                                 ByVal wMsg As Int32, _
                                 ByVal wParam As Int32, _
                                 ByVal lParam As Int32) As Int32

    Private Const WM_CLOSE As Int32 = &H10

    Private Const EPSILON As Double = 1.0E-30

    Private Delegate Sub UpdateGUIInvoker(ByVal text As String, ByVal bytesReceived As Long)

    Dim m_highPerformanceTimer As New Win32.HiPerfTimer
    Dim m_highPerformanceLog As New Win32.HiPerfTimer

    Private WithEvents m_port2 As Port

    Private m_bytesReceived As Long

    Private m_UTF8Encoding As UTF8Encoding

    Private m_stopSending As Boolean
    Private m_dataReceived As Boolean

    Private m_exiting As Boolean
    Dim baudRate As Integer
    Public iMode As Integer
    Public dwRet As Long

    Dim dwPic As Integer

    Dim sBuf As String

    Dim bStop As Boolean

    Dim PicBuf As String
    Dim respTime As Double

    Private openedFilename As String                    'filename
    Private fileObj As New Scripting.FileSystemObject
    Private fileStream As Scripting.TextStream
    Private fileOpen As Boolean  'status of log file


 Windows Form Designer generated code
    Private Sub FrmSerialPortTesting_Load(ByVal sender As System.Object, ByVal e As System. ↙
    EventArgs) Handles MyBase.Load
        dwPic = -1
        optPicPacket.Enabled = True

        PicBuf = ""
        m_port2 = New Port("COM2:", CType(New HandshakeCtsRts, DetailedPortSettings))

        baudRate = Convert.ToInt32(cboSpeed.SelectedItem.ToString())

        m_port2.Settings.BaudRate = baudRate

        m_port2.Settings.ByteSize = 8

        m_port2.Settings.Parity = Parity.none

        m_port2.Settings.StopBits = StopBits.one
```

```
        m_port2.SThreshold = 1

        m_port2.RThreshold = 1

        m_port2.InputLen = 0

        Text1.Text = "X"

        m_bytesReceived = 0

        m_UTF8Encoding = New UTF8Encoding

        m_stopSending = False

        m_exiting = False
    End Sub

    Private Sub FrmSerialPortTesting_Closing(ByVal sender As Object, ByVal e As System.
    ComponentModel.CancelEventArgs) Handles MyBase.Closing
        'If data was being sent over the serial port when the user exited then
        'the Application.DoEvents in the Send Data Loop is currently allowing the
        'FrmSerialPortTesting_Closing event rotuine to execute. Setting m_exiting to True
    will
        'cause the Send Data Loop to exit after FrmSerialPortTesting_Closing completes (the
    Send Data Loop
        'will return from its Application.DoEvents, check the looping conditions, and
        'exit the loop when it finds that m_exiting is True
        m_exiting = True
        m_highPerformanceLog.Stop()
        m_highPerformanceTimer.Stop()

        m_port2.Close()

        m_port2.Dispose()

        m_port2 = Nothing
    End Sub

    Private Sub cmdSend_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles cmdSend.Click
        Dim text As String

        m_dataReceived = True
        cmdSend.Enabled = False
        cmdLog.Text = "StartLog"
        cmdLog.Enabled = True
        cmdStop.Enabled = True

        cboSpeed.Enabled = False

        m_stopSending = False

        m_port2.Open()

        m_bytesReceived = 0

        While (Not (m_stopSending Or m_exiting))
            If m_dataReceived Then
                m_highPerformanceTimer.Start()

                If (optPicPacket.Checked) Then
                    dwPic = dwPic + 1

                    Select Case dwPic
                        Case 0
                            PicBuf = "AA000000000000000000000000000000XXBB"
                        Case 1
                            PicBuf = "AA111111111111111111111111111111XXBB"
                        Case 2
```

```
                        PicBuf = "AA22222222222222222222222222222222XXBB"
                    Case 3
                        PicBuf = "AA33333333333333333333333333333333XXBB"
                    Case 4
                        PicBuf = "AA44444444444444444444444444444444XXBB"
                    Case 5
                        PicBuf = "AA55555555555555555555555555555555XXBB"
                    Case 6
                        PicBuf = "AA66666666666666666666666666666666XXBB"
                    Case 7
                        PicBuf = "AA77777777777777777777777777777777XXBB"
                    Case 8
                        PicBuf = "AA88888888888888888888888888888888XXBB"
                    Case 9
                        PicBuf = "AA99999999999999999999999999999999XXBB"
                        dwPic = -1
                End Select
                text = PicBuf & vbCr
            Else
                text = Text1.Text
            End If

            If (text.Length > 0) Then
                m_port2.Output = m_UTF8Encoding.GetBytes(text)
            End If
        End If

        m_dataReceived = False

        System.Windows.Forms.Application.DoEvents()

        If m_dataReceived Then
            m_highPerformanceTimer.Stop()
            respTime = m_highPerformanceTimer.Duration
            TextBox1.Text = Format((respTime - (10 * 1.5 / baudRate)), "0.00E-00")
            If fileOpen Then
                txtLogTime.Text = Format$(m_highPerformanceLog.ElpasedTime, "0000")
                fileStream.Write("  " & TextBox1.Text & vbNewLine)
                If txtLogTime.Text = 600 Then
                    fileStream.Close()
                    fileStream = Nothing
                    cmdLog.Text = "Start Log"
                    fileOpen = False
                End If

            End If
        End If

    End While

    m_stopSending = False
End Sub

Private Sub cmdStop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles cmdStop.Click
    m_stopSending = True
    m_highPerformanceLog.Stop()
    m_highPerformanceTimer.Stop()
    cmdLog.Text = "Log File"

    If fileOpen = True Then
        fileStream.Close()
        fileStream = Nothing
        fileOpen = False
    End If

    Dim timerWait As New Win32.HiPerfTimer
    timerWait.Start()
```

```
        While timerWait.ElpasedTime < 2
            System.Windows.Forms.Application.DoEvents()
        End While
        m_port2.Close()

        cboSpeed.Enabled = True

        cmdStop.Enabled = False
        cmdSend.Enabled = True
    End Sub

    Private Sub Reopen()
        m_port2.Close()

        Dim baudRate As Integer = Convert.ToInt32(cboSpeed.SelectedItem.ToString())

        m_port2.Settings.BaudRate = baudRate

        m_port2.Open()
    End Sub

    Private Sub cboSpeed_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles cboSpeed.SelectedIndexChanged
        If (Not IsNothing(m_port2)) Then
            Reopen()
        End If
    End Sub

    Private Sub m_port2_DataReceived() Handles m_port2.DataReceived
        m_dataReceived = True

        Dim bytesReceived As Long = m_port2.InBufferCount

        Dim bytes As Byte() = m_port2.Input

        'The GUI must be updated on the thread that created it
        UpdateGUI(m_UTF8Encoding.GetString(bytes), bytesReceived)
    End Sub

    Private Sub UpdateGUI(ByVal text As String, ByVal bytesReceived As Long)
        If (Not Me.InvokeRequired()) Then
            m_bytesReceived += bytesReceived

            If (chkUpdateDisplayString.Checked) Then
                If (Not (text Is Nothing)) Then
                    txtOutput.Text = text
                Else
                    txtOutput.Text = ""
                End If
            End If
            Else
                Dim pList() As Object = {text, bytesReceived}

                Me.Invoke(New UpdateGUIInvoker(AddressOf Me.UpdateGUI), pList)
            End If
    End Sub

    Private Sub mainMenuFileExit_Click(ByVal sender As System.Object, ByVal e As System.
    EventArgs) Handles mainMenuFileExit.Click
        Close()
    End Sub
    Private Sub cmdLog_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles cmdLog.Click
        m_highPerformanceLog.Start()

        If Not fileOpen Then
            fileStream = fileObj.CreateTextFile(txtLog.Text, True, False)
            fileStream.WriteLine("# Log file created at " & Now & vbNewLine & vbNewLine & _
                                 "  Time")
```

```
            fileOpen = True
            cmdLog.Text = "Stop Log"
        Else
            fileStream.Close()
            fileStream = Nothing
            cmdLog.Text = "Start Log"
            fileOpen = False
        End If
    End Sub
End Class
```

```
using System;
using System.Runtime.InteropServices;
using System.ComponentModel;
using System.Threading;

namespace Win32
{
    public class HiPerfTimer
    {
        [DllImport("Kernel32.dll")]
        private static extern bool QueryPerformanceCounter(
            out long lpPerformanceCount);

        [DllImport("Kernel32.dll")]
        private static extern bool QueryPerformanceFrequency(
            out long lpFrequency);

        private long startTime, stopTime;
        private long freq;

        // Constructor
        public HiPerfTimer()
        {
            startTime = 0;
            stopTime  = 0;

            if (QueryPerformanceFrequency(out freq) == false)
            {
                // high-performance counter not supported
                throw new Win32Exception();
            }
        }

        // Start the timer
        public void Start()
        {
            // lets do the waiting threads there work
            Thread.Sleep(0);

            QueryPerformanceCounter(out startTime);
        }

        // Stop the timer
        public void Stop()
        {
            QueryPerformanceCounter(out stopTime);
        }

        // Returns the duration of the timer (in seconds)
        public double Duration
        {
            get
            {
                return (double)(stopTime - startTime) / (double) freq;
            }
        }

        public double ElpasedTime
        {
            get
            {
                long currentTime;


                QueryPerformanceCounter(out currentTime);

                return (double)(currentTime - startTime) / (double) freq;
            }
        }
```

```
    }
}
```

Appendix C

Visual Basic 6.0 Open Port Source Code

```
Form1 - 1

Option Explicit

Public Sub Form_Load()

         Form1.Caption = "Open Port"
         With MSComm1
            .CommPort = 1
            .Handshaking = comRTS
            .RThreshold = 1
            .RTSEnable = True
            '.Settings = "19200,n,8,1"
            .SThreshold = 1
            '.PortOpen = True
            ' Leave all other settings as default values.
         End With

         cboSpeed.AddItem "9600"
         cboSpeed.AddItem "19200"
         cboSpeed.AddItem "38400"
         cboSpeed.AddItem "57600"
         cboSpeed.AddItem "115200"

         cmdOpen.Enabled = True
         cmdStop.Enabled = False

End Sub

Private Sub cmdOpen_Click()

        cmdStop.Enabled = True
        cmdOpen.Enabled = False
        cboSpeed.Enabled = False

        Dim baudRate As String

        baudRate = cboSpeed.Text

        MSComm1.Settings = baudRate

        MSComm1.PortOpen = True

End Sub

Private Sub form_unload(cancel As Integer)
        End
End Sub

Private Sub MSComm1_OnComm()

         Select Case MSComm1.CommEvent
         ' Handle each event or error by placing
         ' code below each case statement.

         ' This template is found in the Example
         ' section of the OnComm event Help topic
         ' in VB Help.

         ' Errors
            Case comEventBreak    ' A Break was received.
            Case comEventCDTO     ' CD (RLSD) Timeout.
            Case comEventCTSTO    ' CTS Timeout.
            Case comEventDSRTO    ' DSR Timeout.
            Case comEventFrame    ' Framing Error.
            Case comEventOverrun  ' Data Lost.
            Case comEventRxOver   ' Receive buffer overflow.
            Case comEventRxParity   ' Parity Error.
            Case comEventTxFull   ' Transmit buffer full.
            Case comEventDCB      ' Unexpected error retrieving DCB]

         ' Events
            Case comEvCD    ' Change in the CD line.
```

```
            Case comEvCTS   ' Change in the CTS line.
            Case comEvDSR   ' Change in the DSR line.
            Case comEvRing  ' Change in the Ring Indicator.
            Case comEvReceive ' Received RThreshold # of chars.

               MSComm1.Output = MSComm1.Input

            Case comEvSend  ' There are SThreshold number of
                            ' characters in the transmit buffer.
            Case comEvEOF   ' An EOF character was found in the
                            ' input stream.

        End Select

      End Sub


Private Sub cmdStop_Click()

        cmdStop.Enabled = False
        cmdOpen.Enabled = True
        cboSpeed.Enabled = True

        MSComm1.PortOpen = False
End Sub
```

Appendix D

Visual Basic 6.0 Send Data Source Code

```
Form1 - 1

Option Explicit

Private Declare Function QueryPerformanceCounter Lib "Kernel32.dll" (lpPerformanceCount As Currenc
y) As Boolean
Private Declare Function QueryPerformanceFrequency Lib "Kernel32.dll" (lpPerformanceFreq As Curren
cy) As Boolean

Dim curfreq As Currency
Dim startCount As Currency
Dim finishCount As Currency

Dim startLogTime As Currency
Dim endLogTime As Currency

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Dim testStart As Currency
Dim testFinish As Currency

Dim m_dataReceived As Boolean

Private openedFilename As String                    'filename
Private fileObj As New FileSystemObject
Private fileStream As Scripting.TextStream
Private fileOpen As Boolean                  'status of log file

Private m_bytesReceived As Long
Dim m_bytesSent As Long

Private m_dataEventsReceived As Long

Public iMode As Integer

Public dwRet As Long

Dim dwPic As Integer

Dim sBuf As String

Dim bStop As Boolean

Dim PicBuf As String

Dim time As Double

Public baudrate As String

Dim firstTime As Boolean
Dim newTimer As New HighPerfTimer

Public Sub InitModem()
    Me.MousePointer = vbHourglass

    sBar.SimpleText = "Putting modem into command mode."
    dwRet = InitializeModem(MSComm1)

    If (dwRet = SUCCESS) Then
        sBar.SimpleText = "Modem is in command mode."
        iMode = COMMAND_MODE
    Else
        Me.MousePointer = vbDefault
        MsgBox "Unable to put modem into command mode.", vbCritical + vbOKOnly
        iMode = UNKNOWN_MODE
        Exit Sub
    End If

    ' Now the modem should be in command mode, or it already was in
    ' commmand mode.

    cmdSend.Enabled = True
    cmdStop.Enabled = False
```

```
    Me.MousePointer = vbDefault

End Sub

Public Sub Form_Load()

        dwPic = -1
        optPicpacket.Enabled = True

        PicBuf = ""

        Form1.Caption = "MSComm App"
        With MSComm1
           .CommPort = 1
           .Handshaking = comRTS
           .RThreshold = 1
           .RTSEnable = True
          ' .Settings = "115200"
           .SThreshold = 1
           .PortOpen = True
           ' Leave all other settings as default values.
        End With


        cboPort.AddItem "9600"
        cboPort.AddItem "19200"
        cboPort.AddItem "38400"
        cboPort.AddItem "57600"
        cboPort.AddItem "115200"

        QueryPerformanceFrequency curfreq

        Text1.Text = "T"

        m_bytesReceived = 0
        m_dataEventsReceived = 0
        m_bytesSent = 0

        cmdSend.Enabled = True
        cmdStop.Enabled = False

        Me.Visible = True
End Sub

Private Sub cmdSend_Click()

        Dim outString As String

        cmdStop.Enabled = True
        cmdSend.Enabled = False
        cmdLog.Enabled = True
        cmdLog.Caption = "Start Log"
        cboPort.Enabled = False

        m_dataReceived = True
        firstTime = True
        m_bytesReceived = 0
        m_bytesSent = 0
        m_dataEventsReceived = 0

        bStop = False

        If MSComm1.PortOpen = False Then
            MSComm1.PortOpen = True
        End If

        baudrate = cboPort.Text
        MSComm1.Settings = baudrate

        QueryPerformanceCounter startCount
```

```
Do
    If m_dataReceived Then
        QueryPerformanceCounter testStart

        If (optPicpacket.Value = vbChecked) Then
             dwPic = dwPic + 1

            Select Case dwPic
                Case 0
                    PicBuf = "AA00000000000000000000000000000000XXBB"
                Case 1
                    PicBuf = "AA11111111111111111111111111111111XXBB"
                Case 2
                    PicBuf = "AA22222222222222222222222222222222XXBB"
                Case 3
                    PicBuf = "AA33333333333333333333333333333333XXBB"
                Case 4
                    PicBuf = "AA44444444444444444444444444444444XXBB"
                Case 5
                    PicBuf = "AA55555555555555555555555555555555XXBB"
                Case 6
                    PicBuf = "AA66666666666666666666666666666666XXBB"
                Case 7
                    PicBuf = "AA77777777777777777777777777777777XXBB"
                Case 8
                    PicBuf = "AA88888888888888888888888888888888XXBB"
                Case 9
                    PicBuf = "AA99999999999999999999999999999999XXBB"
                dwPic = -1
            End Select
            outString = PicBuf & vbCr

        Else
            outString = Text1.Text
        End If
            MSComm1.Output = outString
           'm_bytesSent = Len(outString) + m_bytesSent
           ' Debug.Print m_bytesSent & "  Bytes Sent"
    End If

    QueryPerformanceCounter endLogTime

    m_dataReceived = False

    DoEvents

    If m_dataReceived Then
        QueryPerformanceCounter testFinish

        time = (testFinish - testStart) / curfreq

       ' Debug.Print time

        Text4.Text = Format((time) - (10 * 1.5 / baudrate), "0.00E-00")
        Debug.Print Text4.Text
        If fileOpen Then
            txtLogTime.Text = Format$((endLogTime - startLogTime) / curfreq, "0000")
            fileStream.Write ("   " & Text4.Text & vbNewLine)

            'If txtLogTime.Text > 600 Then
'               fileStream.Close
'               Set fileStream = Nothing
'               cmdLog.Caption = "Start Log"
'               fileOpen = False
            'End If

        End If
    End If
Loop Until bStop = True
```

```
End Sub

Private Sub Form_Unload(Cancel As Integer)
    bStop = True

    MSComm1.PortOpen = False

    End
End Sub

Private Sub MSComm1_OnComm()
        Dim InBuff As String
        Dim bytesReceived As String

        Select Case MSComm1.CommEvent
        ' Handle each event or error by placing
        ' code below each case statement.

        ' This template is found in the Example
        ' section of the OnComm event Help topic
        ' in VB Help.

        ' Errors
           Case comEventBreak    ' A Break was received.
           Case comEventCDTO     ' CD (RLSD) Timeout.
           Case comEventCTSTO    ' CTS Timeout.
           Case comEventDSRTO    ' DSR Timeout.
           Case comEventFrame    ' Framing Error.
           Case comEventOverrun  ' Data Lost.
           Case comEventRxOver   ' Receive buffer overflow.
           Case comEventRxParity   ' Parity Error.
           Case comEventTxFull   ' Transmit buffer full.
           Case comEventDCB      ' Unexpected error retrieving DCB]

        ' Events
           Case comEvCD    ' Change in the CD line.
           Case comEvCTS   ' Change in the CTS line.
           Case comEvDSR   ' Change in the DSR line.
           Case comEvRing  ' Change in the Ring Indicator.
           Case comEvReceive ' Received RThreshold # of chars.
               InBuff = MSComm1.Input
'               m_bytesReceived = Len(InBuff) + m_bytesReceived
'               Debug.Print m_bytesReceived & "  bytes received"

               m_dataReceived = True

               Call HandleInput(InBuff)
           Case comEvSend  ' There are SThreshold number of
                           ' characters in the transmit buffer.
           Case comEvEOF   ' An EOF character was found in the
                           ' input stream.

       End Select

     End Sub

   Sub HandleInput(InBuff As String)
        ' This is where you will process your input. This
        ' includes trapping characters, parsing strings,
        ' separating data fields, etc. For this case, you
        ' are simply going to display the data in the TextBox.
        If (chkUpdate.Value = vbChecked) Then
               If (Not (InBuff = "")) Then
                   Text2.Text = InBuff
               Else
                   Text2.Text = ""
               End If
        End If
   End Sub

Private Sub cmdLog_Click()
```

```
    QueryPerformanceCounter startLogTime

    If Not fileOpen Then
        Set fileStream = fileObj.CreateTextFile(txtlog.Text, True, False)
        fileStream.WriteLine ("# Log file created at " & Now & vbNewLine & vbNewLine & _
                              "  Time")
        fileOpen = True
        cmdLog.Caption = "Stop Log"
    Else
        fileStream.Close
        Set fileStream = Nothing
        cmdLog.Caption = "Start Log"
        fileOpen = False
    End If

End Sub

Private Sub cmdStop_Click()
        QueryPerformanceCounter finishCount
        QueryPerformanceCounter endLogTime

        cmdLog.Caption = "Log File"

        If fileOpen = True Then
            fileStream.Close
            Set fileStream = Nothing
            fileOpen = False
        End If

        Dim timerWait As New HighPerfTimer
        timerWait.StartTimer (True)

        While timerWait.ElapsedTime < 2
            DoEvents
        Wend

        cmdStop.Enabled = False
        cmdSend.Enabled = True
        cboPort.Enabled = True

        bStop = True

        MSComm1.PortOpen = False
End Sub
```

```
'-----------------------------------------
' PerformanceTimer class module
'-----------------------------------------

' Use this class to profile your code and any other operation
' typically with a precision greater than 1 millionth of a second
'
' As soon as you create an object, the timer starts
' but you can also start it explicitly with StartTimer
' Stop the timer and retrieve timing with StopTimer, or
' get the timing without stopping the timer with ElapsedTime
'
' The TotalTime property returns the number of seconds the
' timer has been active, so you can use it to sum up partial
' timings, after swithing the timer on and off
' The FormatTime is similar to elapsed time, but returns
' the time as a formatted string with desired precision
'
' Example:
'         Dim pc As New PerformanceCounter
'         pc.StartTimer
'         ' ...
'         ' put here the code you want to benchmark
'         ' ...
'         ' print elapsed time, but don't stop the timer
'         Debug.Print pc.ElapsedTime
'         ' ...
'         ' so something else here
'         ' ...
'         ' print elapsed time and stop the timer
'         Debug.Print pc.StopTimer
'         ' ...
'         ' prepare another benchmark here
'         ' ...
'         ' start the benchmark, without resetting total time
'         pc.StartTimer
'         ' ...
'         ' put here the code you want to benchmark
'         ' ...
'         ' print elapsed as a formatted string
'         Debug.Print pc.FormatTime("Second benchmark ### secs.", 4)
'         ' print total time
'         Debug.Print pc.TotalTime
'

Option Explicit

Private Declare Function QueryPerformanceFrequencyAny Lib "kernel32" Alias _
    "QueryPerformanceFrequency" (lpFrequency As Any) As Long
Private Declare Function QueryPerformanceCounterAny Lib "kernel32" Alias _
    "QueryPerformanceCounter" (lpPerformanceCount As Any) As Long

' the frequency for this computer
Dim frequency As Currency
Dim startTime As Currency
Dim endTime As Currency
Dim totTime As Currency

' Start the timer
'
' if argument is True, it also resets the
' internal total time counter

Sub StartTimer(Optional ByVal ResetTotalTime As Boolean)
    ' get the current value of the counter
    QueryPerformanceCounterAny startTime
    ' reset total time counter if requested
    If ResetTotalTime Then totTime = 0
End Sub

' stop the timer
```

```
HighPerfTimer - 2

'
' returns the time elapsed since StartTimer

Function StopTimer() As Double
    ' get the elapsed time
    StopTimer = ElapsedTime
    ' update the total time counter
    totTime = totTime + (endTime - startTime)
    ' reset starting time
    startTime = 0
End Function

' return the elapsed time in seconds since StartTimer
' without stopping the timer

Property Get ElapsedTime() As Double
    ' exit if StartTimer hasn't been called since
    ' the previous call to StopTimer
    If startTime = 0 Then Exit Property

    ' get the current value of the counter
    QueryPerformanceCounterAny endTime
    ' return the elapsed time in seconds
    ElapsedTime = (endTime - startTime) / frequency
End Property

' return the total time in seconds

Property Get TotalTime() As Double
    If startTime = 0 Then
        ' StopTimer has been called
        ' so totTime is correctly updated
        TotalTime = totTime / frequency
    Else
        TotalTime = (totTime + (endTime - startTime)) / frequency
    End If
End Property

' return a time value as a formatted string
' if second argument is omitted, it uses ElapsedTime
'
' return it as a formatted string with
' specified number of decimal - use ### in the string
' as a placeholder for the elapsed time
'   e.g.  Print GetTimeMsg("Elapsed ### secs.", , 4)
'
' NOTE: this function is slightly less precise than
'       GetTime, because arguments are passed

Property Get FormatTime(msg As String, Optional seconds As Double = -1, _
    Optional ByVal decDigits As Integer = 7) As String
    ' get the elapsed time if not passed as an argument
    If seconds < 0 Then seconds = ElapsedTime()
    ' build the result string
    FormatTime = Replace(msg, "###", CStr(Round(seconds, decDigits)))
End Property

' return the timer precision in seconds

Property Get Precision() As Double
    ' frequency must be scaled up by 10E4
    Precision = 1 / (frequency * 10000#)
End Property

' evaluate the frequency once and for all
' when this object is created

Private Sub Class_Initialize()
    ' raise error if API functions aren't supported
    If QueryPerformanceFrequencyAny(frequency) = 0 Then
        Err.Raise 1001, , "This system doesn't support high-res timing"
```

```
    End If
    ' get start time as well
    StartTimer
End Sub
```

Appendix E

Serial Port Library

```
//=====================================================================================================
//
//      namespace OpenNETCF.IO.Ports.SerialEventsAndArgs
//      Copyright (c) 2005, OpenNETCF.org
//
//      This library is free software; you can redistribute it and/or modify it under
//      the terms of the OpenNETCF.org Shared Source License.
//
//      This library is distributed in the hope that it will be useful, but
//      WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//      FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//      for more details.
//
//      You should have received a copy of the OpenNETCF.org Shared Source License
//      along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//      If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//      email licensing@opennetcf.org.
//
//      For general enquiries, email enquiries@opennetcf.org or visit our website at:
//      http://www.opennetcf.org
//
//=====================================================================================================

using System;

namespace OpenNETCF.IO.Ports {
    Delegates

    EventArgs
}
```

```
//=============================================================================================
//
//      OpenNETCF.IO.Serial.PortCapabilities
//      Copyright (c) 2004, OpenNETCF.org
//
//      This library is free software; you can redistribute it and/or modify it under
//      the terms of the OpenNETCF.org Shared Source License.
//
//      This library is distributed in the hope that it will be useful, but
//      WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//      FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//      for more details.
//
//      You should have received a copy of the OpenNETCF.org Shared Source License
//      along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//      If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//      email licensing@opennetcf.org.
//
//      For general enquiries, email enquiries@opennetcf.org or visit our website at:
//      http://www.opennetcf.org
//
//=============================================================================================
using System;
using System.Runtime.InteropServices;
using System.Collections.Specialized;
namespace OpenNETCF.IO.Serial
{
    //
    // Serial provider type.
    //
/// <summary>
/// SEP enumerates known serial provider types. Currently SERIALCOMM is the only
/// provider in this enumeration.
/// </summary>
    [Flags]
    public enum SEP
    {
        /// <summary>
        /// SERIALCOMM is the only service provider supported by serial APIs.
        /// </summary>
        SEP_SERIALCOMM   = 0x00000001
    };
    //
    // Provider SubTypes
    //
    /// <summary>
    /// PST enumerates the provider subtypes supported by the WIN32 serial APIs. PST
    indicates which
    /// Port is used for serial communication. Ports can either be physical or logical
    devices.
    /// </summary>
    public enum PST
    {
        /// <summary>
        /// no provider subtype specified
        /// </summary>
        PST_UNSPECIFIED     = 0x00000000,
        /// <summary>
        /// RS232 Port
        /// </summary>
        PST_RS232           = 0x00000001,
        /// <summary>
        /// parallel port
        /// </summary>
        PST_PARALLELPORT    = 0x00000002,
        /// <summary>
        /// RS422 Port
        /// </summary>
```

```
        PST_RS422           = 0x00000003,
        /// <summary>
        /// RS423 Port
        /// </summary>
        PST_RS423           = 0x00000004,
        /// <summary>
        /// RS449 Port
        /// </summary>
        PST_RS449           = 0x00000005,
        /// <summary>
        /// Modem
        /// </summary>
        PST_MODEM           = 0x00000006,
        /// <summary>
        /// Fax
        /// </summary>
        PST_FAX             = 0x00000021,
        /// <summary>
        /// Scanner
        /// </summary>
        PST_SCANNER         = 0x00000022,
        /// <summary>
        /// unspecified network bridge
        /// </summary>
        PST_NETWORK_BRIDGE  = 0x00000100,
        /// <summary>
        /// DEC's LAT Port
        /// </summary>
        PST_LAT             = 0x00000101,
        /// <summary>
        /// Telnet connection
        /// </summary>
        PST_TCPIP_TELNET    = 0x00000102,
        /// <summary>
        /// X.25 standard
        /// </summary>
        PST_X25             = 0x00000103
    };

    //
    // Provider capabilities flags.
    //
/// <summary>
/// PCF enumerates the provider capabilites supported by the specified COMx: Port. This enumeration
/// is used internaly only. Access to this bitfield information is provided through attributes of the
/// CommProp class.
/// </summary>
    [Flags]
    internal enum PCF
    {
        PCF_DTRDSR       = 0x0001,
        PCF_RTSCTS       = 0x0002,
        PCF_RLSD         = 0x0004,
        PCF_PARITY_CHECK = 0x0008,
        PCF_XONXOFF      = 0x0010,
        PCF_SETXCHAR     = 0x0020,
        PCF_TOTALTIMEOUTS= 0x0040,
        PCF_INTTIMEOUTS  = 0x0080,
        PCF_SPECIALCHARS = 0x0100,
        PCF_16BITMODE    = 0x0200
    };

    //
    // Comm provider settable parameters.
    //
/// <summary>
/// SP
```

```
/// </summary>
    [Flags]
    internal enum SP
    {
        SP_PARITY        = 0x0001,
        SP_BAUD          = 0x0002,
        SP_DATABITS      = 0x0004,
        SP_STOPBITS      = 0x0008,
        SP_HANDSHAKING   = 0x0010,
        SP_PARITY_CHECK  = 0x0020,
        SP_RLSD          = 0x0040
    };

    //
    // Settable baud rates in the provider.
    //
    /// <summary>
    /// baud rates settable by Comm API
    /// </summary>
    [Flags]
    public enum BAUD
    {
        /// <summary>
        /// 75 bits per second
        /// </summary>
        BAUD_075         = 0x00000001,
        /// <summary>
        /// 110 bits per second
        /// </summary>
        BAUD_110         = 0x00000002,
        /// <summary>
        /// 134.5 bits per second
        /// </summary>
        BAUD_134_5       = 0x00000004,
        /// <summary>
        /// 150 bits per second
        /// </summary>
        BAUD_150         = 0x00000008,
        /// <summary>
        /// 300 bits per second
        /// </summary>
        BAUD_300         = 0x00000010,
        /// <summary>
        /// 600 bits per second
        /// </summary>
        BAUD_600         = 0x00000020,
        /// <summary>
        /// 1,200 bits per second
        /// </summary>
        BAUD_1200        = 0x00000040,
        /// <summary>
        /// 1,800 bits per second
        /// </summary>
        BAUD_1800        = 0x00000080,
        /// <summary>
        /// 2,400 bits per second
        /// </summary>
        BAUD_2400        = 0x00000100,
        /// <summary>
        /// 4,800 bits per second
        /// </summary>
        BAUD_4800        = 0x00000200,
        /// <summary>
        /// 7,200 bits per second
        /// </summary>
        BAUD_7200        = 0x00000400,
        /// <summary>
        /// 9,600 bits per second
        /// </summary>
```

```
        BAUD_9600       = 0x00000800,
        /// <summary>
        /// 14,400 bits per second
        /// </summary>
        BAUD_14400      = 0x00001000,
        /// <summary>
        /// 19,200 bits per second
        /// </summary>
        BAUD_19200      = 0x00002000,
        /// <summary>
        /// 38,400 bits per second
        /// </summary>
        BAUD_38400      = 0x00004000,
        /// <summary>
        /// 56 Kbits per second
        /// </summary>
        BAUD_56K        = 0x00008000,
        /// <summary>
        /// 129 Kbits per second
        /// </summary>
        BAUD_128K       = 0x00010000,
        /// <summary>
        /// 115,200 bits per second
        /// </summary>
        BAUD_115200     = 0x00020000,
        /// <summary>
        /// 57,600 bits per second
        /// </summary>
        BAUD_57600      = 0x00040000,
        /// <summary>
        /// User defined bitrates
        /// </summary>
        BAUD_USER       = 0x10000000
    };
    //
    // Settable Data Bits
    //

    [Flags]
    internal enum DB
    {
        DATABITS_5      = 0x0001,
        DATABITS_6      = 0x0002,
        DATABITS_7      = 0x0004,
        DATABITS_8      = 0x0008,
        DATABITS_16     = 0x0010,
        DATABITS_16X    = 0x0020
    };
    //
    // Settable Stop and Parity bits.
    //
    [Flags]
    internal enum SB
    {
        STOPBITS_10     = 0x00010000,
        STOPBITS_15     = 0x00020000,
        STOPBITS_20     = 0x00040000,
        PARITY_NONE     = 0x01000000,
        PARITY_ODD      = 0x02000000,
        PARITY_EVEN     = 0x04000000,
        PARITY_MARK     = 0x08000000,
        PARITY_SPACE    = 0x10000000
    };
    //
    // Set dwProvSpec1 to COMMPROP_INITIALIZED to indicate that wPacketLength
    // is valid when calling GetCommProperties().
    //
    [Flags]
    internal enum CPS:uint
```

```
    {
        COMMPROP_INITIALIZED= 0xE73CF52E
    };
    /// <summary>
    /// Container for all available information on port's capabilties
    /// </summary>
    [StructLayout(LayoutKind.Sequential)]
    public class CommCapabilities
    {
        private UInt16 wPacketLength;
        private UInt16 wPacketVersion;
        /// <summary>
        /// Indicates which services are supported by the port. SP_SERIALCOMM is specified for communication
        /// providers, including modem providers.
        /// </summary>
        public IO.Serial.SEP dwServiceMask;
        private UInt32 dwReserved1;
        /// <summary>
        /// Specifies the maximum size, in bytes, of the driver's internal output buffer. A value of zero
        /// indicates that no maximum value is imposed by the driver.
        /// </summary>
        [CLSCompliant(false)]
        public UInt32 dwMaxTxQueue;
        /// <summary>
        /// Specifies the maximum size, in bytes, of the driver's internal input buffer. A value of zero
        /// indicates that no maximum value is imposed by the driver.
        /// </summary>
        [CLSCompliant(false)]
        public UInt32 dwMaxRxQueue;
        /// <summary>
        /// Specifies the maximum baud rate, in bits per second (bps).
        /// </summary>
        public IO.Serial.BAUD dwMaxBaud;
        /// <summary>
        /// Specifies the communication provider type.
        /// </summary>
        public IO.Serial.PST dwProvSubType;
        private BitVector32 dwProvCapabilities;
        private BitVector32 dwSettableParams;
        private BitVector32 dwSettableBaud;
        private BitVector32 dwSettableStopParityData;
        /// <summary>
        /// Specifies the size, in bytes, of the driver's internal output buffer. A value of zero indicates
        /// that the value is unavailable.
        /// </summary>
        [CLSCompliant(false)]
        public UInt32 dwCurrentTxQueue;
        /// <summary>
        /// Specifies the size, in bytes, of the driver's internal input buffer. A value of zero indicates
        /// that the value is unavailable.
        /// </summary>
        [CLSCompliant(false)]
        public UInt32 dwCurrentRxQueue;
        private IO.Serial.CPS dwProvSpec1;
        private UInt32 dwProvSpec2;
        private UInt16 wcProvChar;

        internal CommCapabilities()
        {
            this.wPacketLength=(ushort)Marshal.SizeOf(this);
            this.dwProvSpec1=CPS.COMMPROP_INITIALIZED;

            dwProvCapabilities=new BitVector32(0);
            dwSettableParams=new BitVector32(0);
```

```
            dwSettableBaud=new BitVector32(0);
            dwSettableStopParityData=new BitVector32(0);
        }

        //
        // We need to have to define reserved fields in the CommCapabilties class definition
        // to preserve the size of the
        // underlying structure to match the Win32 structure when it is
        // marshaled. Use these fields to suppress compiler warnings.
        //
        internal void _SuppressCompilerWarnings()
        {
            wPacketVersion +=0;
            dwReserved1 +=0;
            dwProvSpec1 +=0;
            dwProvSpec2 +=0;
            wcProvChar +=0;
        }

        // Provider Capabilties
        /// <summary>
        /// Port supports special 16-bit mode
        /// </summary>
        public bool Supports16BitMode
        {
            get { return dwProvCapabilities[(int)PCF.PCF_16BITMODE]; }
        }

        /// <summary>
        /// Port supports DTR (Data Terminal ready) and DSR (Data Set Ready) flow control
        /// </summary>
        public bool SupportsDtrDts
        {
            get { return dwProvCapabilities[(int)PCF.PCF_DTRDSR]; }
        }

        /// <summary>
        /// Port supports interval timeouts
        /// </summary>
        public bool SupportsIntTimeouts
        {
            get { return dwProvCapabilities[(int)PCF.PCF_INTTIMEOUTS]; }
        }

        /// <summary>
        /// Port supports parity checking
        /// </summary>
        public bool SupportsParityCheck
        {
            get { return dwProvCapabilities[(int)PCF.PCF_PARITY_CHECK]; }
        }

        /// <summary>
        /// Port supports RLSD (Receive Line Signal Detect)
        /// </summary>
        public bool SupportsRlsd
        {
            get { return dwProvCapabilities[(int)PCF.PCF_RLSD]; }
        }

        /// <summary>
        /// Port supports RTS (Request To Send) and CTS (Clear To Send) flowcontrol
        /// </summary>
        public bool SupportsRtsCts
        {
            get { return dwProvCapabilities[(int)PCF.PCF_RTSCTS]; }
        }

        /// <summary>
```

```
        /// Port supports user definded characters for XON and XOFF
        /// </summary>
        public bool SupportsSetXChar
        {
            get { return dwProvCapabilities[(int)PCF.PCF_SETXCHAR]; }
        }

        /// <summary>
        /// Port supports special characters
        /// </summary>
        public bool SupportsSpecialChars
        {
            get { return dwProvCapabilities[(int)PCF.PCF_SPECIALCHARS]; }
        }

        /// <summary>
        /// Port supports total and elapsed time-outs
        /// </summary>
        public bool SupportsTotalTimeouts
        {
            get { return dwProvCapabilities[(int)PCF.PCF_TOTALTIMEOUTS]; }
        }

        /// <summary>
        /// Port supports XON/XOFF flow control
        /// </summary>
        public bool SupportsXonXoff
        {
            get { return dwProvCapabilities[(int)PCF.PCF_XONXOFF]; }
        }

        // Settable Params
        /// <summary>
        /// Baud rate can be set
        /// </summary>
        public bool SettableBaud
        {
            get { return dwSettableParams[(int)SP.SP_BAUD]; }
        }

        /// <summary>
        /// Number of data bits can be set
        /// </summary>
        public bool SettableDataBits
        {
            get { return dwSettableParams[(int)SP.SP_DATABITS]; }
        }

        /// <summary>
        /// Handshake protocol can be set
        /// </summary>
        public bool SettableHandShaking
        {
            get { return dwSettableParams[(int)SP.SP_HANDSHAKING]; }
        }

        /// <summary>
        /// Number of parity bits can be set
        /// </summary>
        public bool SettableParity
        {
            get { return dwSettableParams[(int)SP.SP_PARITY]; }
        }

        /// <summary>
        /// Parity check can be enabled/disabled
        /// </summary>
        public bool SettableParityCheck
        {
```

```
        get { return dwSettableParams[(int)SP.SP_PARITY_CHECK]; }
    }
    /// <summary>
    /// Receive Line Signal detect can be enabled/disabled
    /// </summary>
    public bool SettableRlsd
    {
        get { return dwSettableParams[(int)SP.SP_RLSD]; }
    }
    /// <summary>
    /// Number of stop bits can be set
    /// </summary>
    public bool SettableStopBits
    {
        get { return dwSettableParams[(int)SP.SP_STOPBITS]; }
    }

    // Settable Databits
    /// <summary>
    /// Port supports 5 data bits
    /// </summary>
    public bool Supports5DataBits
    {
        get { return dwSettableStopParityData[(int)DB.DATABITS_5]; }
    }

    /// <summary>
    /// Port supports 6 data bits
    /// </summary>
    public bool Supports6DataBits
    {
        get { return dwSettableStopParityData[(int)DB.DATABITS_6]; }
    }

    /// <summary>
    /// Port supports 7 data bits
    /// </summary>
    public bool Supports7DataBits
    {
        get { return dwSettableStopParityData[(int)DB.DATABITS_7]; }
    }

    /// <summary>
    /// Port supports 8 data bits
    /// </summary>
    public bool Supports8DataBits
    {
        get { return dwSettableStopParityData[(int)DB.DATABITS_8]; }
    }

    /// <summary>
    /// Port supports 16 data bits
    /// </summary>
    public bool Supports16DataBits
    {
        get { return dwSettableStopParityData[(int)DB.DATABITS_16]; }
    }

    /// <summary>
    /// Port supports special wide data path through serial hardware lines
    /// </summary>
    public bool Supports16XDataBits
    {
        get { return dwSettableStopParityData[(int)DB.DATABITS_16X]; }
    }

    // Settable Stop

    /// <summary>
```

```
        /// Port supports even parity
        /// </summary>
        public bool SupportsParityEven
        {
            get { return dwSettableStopParityData[(int)SB.PARITY_EVEN]; }
        }

        /// <summary>
        /// Port supports mark parity
        /// </summary>
        public bool SupportsParityMark
        {
            get { return dwSettableStopParityData[(int)SB.PARITY_MARK]; }
        }

        /// <summary>
        /// Port supports none parity
        /// </summary>
        public bool SupportsParityNone
        {
            get { return dwSettableStopParityData[(int)SB.PARITY_NONE]; }
        }

        /// <summary>
        /// Port supports odd parity
        /// </summary>
        public bool SupportsParityOdd
        {
            get { return dwSettableStopParityData[(int)SB.PARITY_ODD]; }
        }

        /// <summary>
        /// Port supports space parity
        /// </summary>
        public bool SupportsParitySpace
        {
            get { return dwSettableStopParityData[(int)SB.PARITY_SPACE]; }
        }

        /// <summary>
        /// Port supports 1 stop bit
        /// </summary>
        public bool SupportsStopBits10
        {
            get { return dwSettableStopParityData[(int)SB.STOPBITS_10]; }
        }

        /// <summary>
        /// Port supports 1.5 stop bits
        /// </summary>
        public bool SupportsStopBits15
        {
            get { return dwSettableStopParityData[(int)SB.STOPBITS_15]; }
        }

        /// <summary>
        /// Port supports 2 stop bits
        /// </summary>
        public bool SupportsStopBits20
        {
            get { return dwSettableStopParityData[(int)SB.STOPBITS_20]; }
        }

        // settable Baud Rates
        /// <summary>
        /// Port can be set to 75 bits per second
        /// </summary>
        public bool HasBaud75
        {
```

```
        get { return dwSettableBaud[(int)BAUD.BAUD_075];}
    }
    /// <summary>
    /// Port can be set to 110 bits per second
    /// </summary>
    public bool HasBaud110
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_110];}
    }
    /// <summary>
    /// Port can be set to 134.5 bits per second
    /// </summary>
    public bool HasBaud134_5
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_134_5];}
    }
    /// <summary>
    /// Port can be set to 150 bits per second
    /// </summary>
    public bool HasBaud150
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_150];}
    }

    /// <summary>
    /// Port can be set to 300 bits per second
    /// </summary>
    public bool HasBaud300
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_300];}
    }

    /// <summary>
    /// Port can be set to 600 bits per second
    /// </summary>
    public bool HasBaud600
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_600];}
    }

    /// <summary>
    /// Port can be set to 1,200 bits per second
    /// </summary>
    public bool HasBaud1200
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_1200];}
    }

    /// <summary>
    /// Port can be set to 2,400 bits per second
    /// </summary>
    public bool HasBaud2400
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_2400];}
    }

    /// <summary>
    /// Port can be set to 4,800 bits per second
    /// </summary>
    public bool HasBaud4800
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_4800];}
    }

    /// <summary>
    /// Port can be set to 7,200 bits per second
    /// </summary>
    public bool HasBaud7200
    {
```

```
        get { return dwSettableBaud[(int)BAUD.BAUD_7200];}
    }

    /// <summary>
    /// Port can be set to 9,600 bits per second
    /// </summary>
    public bool HasBaud9600
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_9600];}
    }

    /// <summary>
    /// Port can be set to 14,400 bits per second
    /// </summary>
    public bool HasBaud14400
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_14400];}
    }

    /// <summary>
    /// Port can be set to 19,200 bits per second
    /// </summary>
    public bool HasBaud19200
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_19200];}
    }

    /// <summary>
    /// Port can be set to 38,400 bits per second
    /// </summary>
    public bool HasBaud38400
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_38400];}
    }

    /// <summary>
    /// Port can be set to 56 Kbits per second
    /// </summary>
    public bool HasBaud56K
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_56K];}
    }

    /// <summary>
    /// Port can be set to 128 Kbits per second
    /// </summary>
    public bool HasBaud128K
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_128K];}
    }

    /// <summary>
    /// Port can be set to 115,200 bits per second
    /// </summary>
    public bool HasBaud115200
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_115200];}
    }

    /// <summary>
    /// Port can be set to 57,600 bits per second
    /// </summary>
    public bool HasBaud57600
    {
        get { return dwSettableBaud[(int)BAUD.BAUD_57600];}
    }

    /// <summary>
    /// Port can be set to user defined bit rate
```

```
        /// </summary>
        public bool HasBaudUser
        {
            get { return dwSettableBaud[(int)BAUD.BAUD_USER];}
        }

    };


}
```

```
//=============================================================================================
//
//      namespace OpenNETCF.IO.Serial.PortSettings
//      Copyright (c) 2003, OpenNETCF.org
//
//      This library is free software; you can redistribute it and/or modify it under
//      the terms of the OpenNETCF.org Shared Source License.
//
//      This library is distributed in the hope that it will be useful, but
//      WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//      FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//      for more details.
//
//      You should have received a copy of the OpenNETCF.org Shared Source License
//      along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//      If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//      email licensing@opennetcf.org.
//
//      For general enquiries, email enquiries@opennetcf.org or visit our website at:
//      http://www.opennetcf.org
//
//=============================================================================================

using System;
using System.Runtime.InteropServices;

namespace OpenNETCF.IO.Serial
{
    namespace enumerations

    /// <summary>
    /// Used for manipulating several basic Port settings of a Port class
    /// </summary>
    [StructLayout(LayoutKind.Sequential)]
    public class BasicPortSettings
    {
        /// <summary>
        /// Baud rate (default = 19200bps)
        /// </summary>
        public BaudRates    BaudRate    = BaudRates.CBR_19200;
        /// <summary>
        /// Byte Size of data (default = 8)
        /// </summary>
        public byte         ByteSize    = 8;
        /// <summary>
        /// Data Parity (default = none)
        /// </summary>
        public Parity       Parity      = Parity.none;
        /// <summary>
        /// Number of stop bits (default = 1)
        /// </summary>
        public StopBits     StopBits    = StopBits.one;
    }

    /// <summary>
    /// Used for manipulating all settings of a Port class
    /// </summary>
    [StructLayout(LayoutKind.Sequential)]
    public class DetailedPortSettings
    {
        /// <summary>
        /// Create a DetailedPortSettings class
        /// </summary>
        public DetailedPortSettings()
        {
            BasicSettings = new BasicPortSettings();
            Init();
        }
```

```
    /// <summary>
    /// These are the default port settings
    /// override Init() to create new defaults (i.e. common handshaking)
    /// </summary>
    protected virtual void Init()
    {
        BasicSettings.BaudRate  = BaudRates.CBR_19200;
        BasicSettings.ByteSize  = 8;
        BasicSettings.Parity    = Parity.none;
        BasicSettings.StopBits  = StopBits.one;

        OutCTS              = false;
        OutDSR              = false;
        DTRControl          = DTRControlFlows.disable;
        DSRSensitive        = false;
        TxContinueOnXOff    = true;
        OutX                = false;
        InX                 = false;
        ReplaceErrorChar    = false;
        RTSControl          = RTSControlFlows.disable;
        DiscardNulls        = false;
        AbortOnError        = false;
        XonChar             = (char)ASCII.DC1;
        XoffChar            = (char)ASCII.DC3;
        ErrorChar           = (char)ASCII.NAK;
        EOFChar             = (char)ASCII.EOT;
        EVTChar             = (char)ASCII.NULL;
    }

    /// <summary>
    /// Basic port settings
    /// </summary>
    public BasicPortSettings    BasicSettings;
    /// <summary>
    /// Specifies if the CTS (clear-to-send) signal is monitored for output flow control
. If this member is TRUE and CTS is turned off, output is suspended until CTS is sent
again.
    /// </summary>
    public bool                 OutCTS              = false;
    /// <summary>
    /// Specifies if the DSR (data-set-ready) signal is monitored for output flow
control. If this member is TRUE and DSR is turned off, output is suspended until DSR is
sent again.
    /// </summary>
    public bool                 OutDSR              = false;
    /// <summary>
    /// Specifies the DTR (data-terminal-ready) flow control.
    /// </summary>
    public DTRControlFlows      DTRControl          = DTRControlFlows.disable;
    /// <summary>
    /// Specifies if the communications driver is sensitive to the state of the DSR
signal. If this member is TRUE, the driver ignores any bytes received, unless the DSR
modem input line is high.
    /// </summary>
    public bool                 DSRSensitive        = false;
    /// <summary>
    /// Specifies if transmission stops when the input buffer is full and the driver has
 transmitted the XoffChar character. If this member is TRUE, transmission continues
after the input buffer has come within XoffLim bytes of being full and the driver has
transmitted the XoffChar character to stop receiving bytes. If this member is FALSE,
transmission does not continue until the input buffer is within XonLim bytes of being
empty and the driver has transmitted the XonChar character to resume reception.
    /// </summary>
    public bool                 TxContinueOnXOff    = true;
    /// <summary>
    /// Specifies if XON/XOFF flow control is used during transmission. If this member
is TRUE, transmission stops when the XoffChar character is received and starts again
when the XonChar character is received.
```

```
        /// </summary>
        public bool                 OutX                = false;
        /// <summary>
        /// Specifies if XON/XOFF flow control is used during reception. If this member is
    TRUE, the XoffChar character is sent when the input buffer comes within XoffLim bytes of
     being full, and the XonChar character is sent when the input buffer comes within XonLim
     bytes of being empty
        /// </summary>
        public bool                 InX                 = false;
        /// <summary>
        /// Specifies if bytes received with parity errors are replaced with the character
    specified by the ErrorChar member. If this member is TRUE and the fParity member is TRUE
    , replacement occurs.
        /// </summary>
        public bool                 ReplaceErrorChar    = false;
        /// <summary>
        /// Specifies the RTS (request-to-send) flow control. If this value is zero, the
    default is RTS_CONTROL_HANDSHAKE. The following table shows possible values for this
    member.
        /// </summary>
        public RTSControlFlows      RTSControl          = RTSControlFlows.disable;
        /// <summary>
        /// Specifies if null bytes are discarded. If this member is TRUE, null bytes are
    discarded when received.
        /// </summary>
        public bool                 DiscardNulls        = false;
        /// <summary>
        /// Specifies if read and write operations are terminated if an error occurs. If
    this member is TRUE, the driver terminates all read and write operations with an error
    status if an error occurs. The driver will not accept any further communications
    operations until the application has acknowledged the error by calling the ClearError
    function.
        /// </summary>
        public bool                 AbortOnError        = false;
        /// <summary>
        /// Specifies the value of the XON character for both transmission and reception
        /// </summary>
        public char                 XonChar             = (char)ASCII.DC1;
        /// <summary>
        /// Specifies the value of the XOFF character for both transmission and reception.
        /// </summary>
        public char                 XoffChar            = (char)ASCII.DC3;
        /// <summary>
        /// Specifies the value of the character used to replace bytes received with a
    parity error.
        /// </summary>
        public char                 ErrorChar           = (char)ASCII.NAK;
        /// <summary>
        /// Specifies the value of the character used to signal the end of data.
        /// </summary>
        public char                 EOFChar             = (char)ASCII.EOT;
        /// <summary>
        /// Specifies the value of the character used to signal an event.
        /// </summary>
        public char                 EVTChar             = (char)ASCII.NULL;
    }

    /// <summary>
    /// A common implementation of DetailedPortSettings for non handshaking
    /// </summary>
    public class HandshakeNone : DetailedPortSettings
    {
        /// <summary>
        /// Initialize the port
        /// </summary>
        protected override void Init()
        {
            base.Init ();
```

```
            OutCTS = false;
            OutDSR = false;
            OutX = false;
            InX = false;
            RTSControl = RTSControlFlows.enable;
            DTRControl = DTRControlFlows.enable;
            TxContinueOnXOff = true;
            DSRSensitive = false;
        }
    }

    /// <summary>
    /// A common implementation of DetailedPortSettings for XON/XOFF handshaking
    /// </summary>
    public class HandshakeXonXoff : DetailedPortSettings
    {
        /// <summary>
        /// Initialize the port
        /// </summary>
        protected override void Init()
        {
            base.Init ();

            OutCTS = false;
            OutDSR = false;
            OutX = true;
            InX = true;
            RTSControl = RTSControlFlows.enable;
            DTRControl = DTRControlFlows.enable;
            TxContinueOnXOff = true;
            DSRSensitive = false;
            XonChar = (char)ASCII.DC1;
            XoffChar = (char)ASCII.DC3;
        }
    }

    /// <summary>
    /// A common implementation of DetailedPortSettings for CTS/RTS handshaking
    /// </summary>
    public class HandshakeCtsRts : DetailedPortSettings
    {
        /// <summary>
        /// Initialize the port
        /// </summary>
        protected override void Init()
        {
            base.Init ();

            OutCTS = true;
            OutDSR = false;
            OutX = false;
            InX = false;
            RTSControl = RTSControlFlows.handshake;
            DTRControl = DTRControlFlows.enable;
            TxContinueOnXOff = true;
            DSRSensitive = false;
        }
    }

    /// <summary>
    /// A common implementation of DetailedPortSettings for DSR/DTR handshaking
    /// </summary>
    public class HandshakeDsrDtr : DetailedPortSettings
    {
        /// <summary>
        /// Initialize the port
        /// </summary>
        protected override void Init()
        {
```

```
            base.Init ();

            OutCTS = false;
            OutDSR = true;
            OutX = false;
            InX = false;
            RTSControl = RTSControlFlows.enable;
            DTRControl = DTRControlFlows.handshake;
            TxContinueOnXOff = true;
            DSRSensitive = false;
        }
    }
}
```

//===
//
//   OpenNETCF.IO.Serial.Port
//   Copyright (c) 2003, OpenNETCF.org
//
//   This library is free software; you can redistribute it and/or modify it under
//   the terms of the OpenNETCF.org Shared Source License.
//
//   This library is distributed in the hope that it will be useful, but
//   WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//   FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//   for more details.
//
//   You should have received a copy of the OpenNETCF.org Shared Source License
//   along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//   If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//   email licensing@opennetcf.org.
//
//   For general enquiries, email enquiries@opennetcf.org or visit our website at:
//   http://www.opennetcf.org
//
//===

```
using System;
using System.Runtime.InteropServices;
using System.Threading;
using System.Text;
using System.Collections;

namespace OpenNETCF.IO.Serial
{
    /// <summary>
    /// Exceptions throw by the OpenNETCF.IO.Serial class
    /// </summary>
    public class CommPortException : Exception
    {
        /// <summary>
        /// Default CommPortException
        /// </summary>
        /// <param name="desc"></param>
        public CommPortException(string desc) : base(desc) {}
    }

    /// <summary>
    /// A class wrapper for serial port communications
    /// </summary>
    public class Port : IDisposable
    {
        [DllImport("kernel32", EntryPoint="LocalAlloc", SetLastError=true)]
        internal static extern IntPtr LocalAlloc(int uFlags, int uBytes);

        [DllImport("kernel32", EntryPoint="LocalFree", SetLastError=true)]
        internal static extern IntPtr LocalFree(IntPtr hMem);

        delegates and events

        ##### variable declarations #####

        private void Init()
        {
            // create the API class based on the target
            if (System.Environment.OSVersion.Platform != PlatformID.WinCE)
                m_CommAPI=new IO.Serial.WinCommAPI();
            else
                m_CommAPI=new IO.Serial.CECommAPI();

            // create a system event for synchronizing Closing
            closeEvent = m_CommAPI.CreateEvent(true, false, closeEventName);
```

```
        rxFIFO = new Queue(rxBufferSize);
        txBuffer = new byte[txBufferSize];
        portSettings = new DetailedPortSettings();
    }

    constructors

    // since the event thread blocks until the port handle is closed
    // implement both a Dispose and destrucor to make sure that we
    // clean up as soon as possible
    /// <summary>
    /// Dispose the object's resources
    /// </summary>

    // Implement IDisposable.
    // Do not make this method virtual.
    // A derived class should not be able to override this method.
    public void Dispose()
    {
        Dispose(true);

        // This object will be cleaned up by the Dispose method.
        // Therefore, you should call GC.SupressFinalize to
        // take this object off the finalization queue
        // and prevent finalization code for this object
        // from executing a second time.
        GC.SuppressFinalize(this);
    }

    // Dispose(bool disposing) executes in two distinct scenarios.
    // If disposing equals true, the method has been called directly
    // or indirectly by a user's code. Managed and unmanaged resources
    // can be disposed.
    // If disposing equals false, the method has been called by the
    // runtime from inside the finalizer and you should not reference
    // other objects. Only unmanaged resources can be disposed.
    private void Dispose(bool disposing)
    {
        if(!this.m_disposed)
        {
            if(isOpen)
            {
                this.Close();
            }
        }

        m_disposed = true;
    }

    /// <summary>
    /// Class destructor
    /// </summary>
    ~Port()
    {
        // Do not re-create Dispose clean-up code here.
        // Calling Dispose(false) is optimal in terms of
        // readability and maintainability.
        Dispose(false);
    }

    /// <summary>
    /// The name of the Port (i.e. "COM1:")
    /// </summary>
    public string PortName
    {
        get
        {
            return portName;
```

```
        }
        set
        {
            if(! CommAPI.FullFramework)
            {
                // for CE, ensure the port name is colon terminated "COMx:"
                if(! value.EndsWith(":"))
                {
                    portName = value + ":";
                    return;
                }
            }

            portName = value;
        }
    }

    /// <summary>
    /// Returns whether or not the port is currently open
    /// </summary>
    public bool IsOpen
    {
        get
        {
            return isOpen;
        }
    }

    /// <summary>
    /// Open the current port
    /// </summary>
    /// <returns>true if successful, false if it fails</returns>
    public bool Open()
    {
        if(isOpen) return false;

        if(CommAPI.FullFramework)
        {
            // set up the overlapped tx IO
            //              AutoResetEvent are = new AutoResetEvent(false);
            OVERLAPPED o = new OVERLAPPED();
            txOverlapped = LocalAlloc(0x40, Marshal.SizeOf(o));
            o.Offset = 0;
            o.OffsetHigh = 0;
            o.hEvent = txevent.Handle;
            Marshal.StructureToPtr(o, txOverlapped, true);
        }

        hPort = m_CommAPI.CreateFile(portName);

        if(hPort == (IntPtr)CommAPI.INVALID_HANDLE_VALUE)
        {
            int e = Marshal.GetLastWin32Error();

            if(e == (int)APIErrors.ERROR_ACCESS_DENIED)
            {
                // port is unavailable
                return false;
            }

            // ClearCommError failed!
            string error = String.Format("CreateFile Failed: {0}", e);
            throw new CommPortException(error);
        }


        isOpen = true;

        // set queue sizes
```

```
            m_CommAPI.SetupComm(hPort, rxBufferSize, txBufferSize);

            // transfer the port settings to a DCB structure
            dcb.BaudRate = (uint)portSettings.BasicSettings.BaudRate;
            dcb.ByteSize = portSettings.BasicSettings.ByteSize;
            dcb.EofChar = (sbyte)portSettings.EOFChar;
            dcb.ErrorChar = (sbyte)portSettings.ErrorChar;
            dcb.EvtChar = (sbyte)portSettings.EVTChar;
            dcb.fAbortOnError = portSettings.AbortOnError;
            dcb.fBinary = true;
            dcb.fDsrSensitivity = portSettings.DSRSensitive;
            dcb.fDtrControl = (byte)portSettings.DTRControl;
            dcb.fErrorChar = portSettings.ReplaceErrorChar;
            dcb.fInX = portSettings.InX;
            dcb.fNull = portSettings.DiscardNulls;
            dcb.fOutX = portSettings.OutX;
            dcb.fOutxCtsFlow = portSettings.OutCTS;
            dcb.fOutxDsrFlow = portSettings.OutDSR;
            dcb.fParity = (portSettings.BasicSettings.Parity == Parity.none) ? false : true;
            dcb.fRtsControl = (byte)portSettings.RTSControl;
            dcb.fTXContinueOnXoff = portSettings.TxContinueOnXOff;
            dcb.Parity = (byte)portSettings.BasicSettings.Parity;

            dcb.StopBits = (byte)portSettings.BasicSettings.StopBits;
            dcb.XoffChar = (sbyte)portSettings.XoffChar;
            dcb.XonChar = (sbyte)portSettings.XonChar;

            dcb.XonLim = dcb.XoffLim = (ushort)(rxBufferSize / 10);

            m_CommAPI.SetCommState(hPort, dcb);

            // store some state values
            brk = 0;
            dtr = dcb.fDtrControl == (byte)DCB.DtrControlFlags.Enable ? 1 : 0;
            rts = dcb.fRtsControl == (byte)DCB.RtsControlFlags.Enable ? 1 : 0;

            // set the Comm timeouts
            CommTimeouts ct = new CommTimeouts();

            // reading we'll return immediately
            // this doesn't seem to work as documented
            ct.ReadIntervalTimeout = uint.MaxValue; // this = 0xffffffff
            ct.ReadTotalTimeoutConstant = 0;
            ct.ReadTotalTimeoutMultiplier = 0;

            // writing we'll give 5 seconds
            ct.WriteTotalTimeoutConstant = 5000;
            ct.WriteTotalTimeoutMultiplier = 0;

            m_CommAPI.SetCommTimeouts(hPort, ct);

            // read the ports capabilities
            bool status=GetPortProperties();

            // start the receive thread
            eventThread = new Thread(new ThreadStart(CommEventThread));
            eventThread.Priority = ThreadPriority.Highest;
            eventThread.Start();

            // wait for the thread to actually get spun up
            threadStarted.WaitOne();

            return true;
        }

        /// <summary>
        /// Query the current port's capabilities without accessing it. You can only call ↙
    the Close()
        /// method after reading the capabilities. This method does neither initialize nor ↙
```

```
    Open() the
        /// port.
        /// </summary>
        ///
        /// <example>
        ///
        /// </example>
        public bool Query()
        {
            if(isOpen) return false;

            hPort = m_CommAPI.QueryFile(portName);

            if(hPort == (IntPtr)CommAPI.INVALID_HANDLE_VALUE)
            {
                int e = Marshal.GetLastWin32Error();

                if(e == (int)APIErrors.ERROR_ACCESS_DENIED)
                {
                    // port is unavailable
                    return false;
                }

                // ClearCommError failed!
                string error = String.Format("CreateFile Failed: {0}", e);
                throw new CommPortException(error);
            }


            // read the port's capabilities
            bool status=GetPortProperties();

            return true;
        }

        // parameters without closing and reopening the port
        /// <summary>
        /// Updates communication settings of the port
        /// </summary>
        /// <returns>true if successful, false if it fails</returns>
        private bool UpdateSettings()
        {
            if(!isOpen) return false;

            // transfer the port settings to a DCB structure
            dcb.BaudRate = (uint)portSettings.BasicSettings.BaudRate;
            dcb.ByteSize = portSettings.BasicSettings.ByteSize;
            dcb.EofChar = (sbyte)portSettings.EOFChar;
            dcb.ErrorChar = (sbyte)portSettings.ErrorChar;
            dcb.EvtChar = (sbyte)portSettings.EVTChar;
            dcb.fAbortOnError = portSettings.AbortOnError;
            dcb.fBinary = true;
            dcb.fDsrSensitivity = portSettings.DSRSensitive;
            dcb.fDtrControl = (byte)portSettings.DTRControl;
            dcb.fErrorChar = portSettings.ReplaceErrorChar;
            dcb.fInX = portSettings.InX;
            dcb.fNull = portSettings.DiscardNulls;
            dcb.fOutX = portSettings.OutX;
            dcb.fOutxCtsFlow = portSettings.OutCTS;
            dcb.fOutxDsrFlow = portSettings.OutDSR;
            dcb.fParity = (portSettings.BasicSettings.Parity == Parity.none) ? false : true;
            dcb.fRtsControl = (byte)portSettings.RTSControl;
            dcb.fTXContinueOnXoff = portSettings.TxContinueOnXOff;
            dcb.Parity = (byte)portSettings.BasicSettings.Parity;
            dcb.StopBits = (byte)portSettings.BasicSettings.StopBits;
            dcb.XoffChar = (sbyte)portSettings.XoffChar;
            dcb.XonChar = (sbyte)portSettings.XonChar;

            dcb.XonLim = dcb.XoffLim = (ushort)(rxBufferSize / 10);
```

```
            return m_CommAPI.SetCommState(hPort, dcb);

        }

        /// <summary>
        /// Close the current serial port
        /// </summary>
        /// <returns>true indicates success, false indicated failure</returns>
        public bool Close()
        {

            if(txOverlapped != IntPtr.Zero)
            {
                LocalFree(txOverlapped);
                txOverlapped = IntPtr.Zero;
            }

            if(!isOpen) return false;

            isOpen = false; // to help catch intentional close

            if(m_CommAPI.CloseHandle(hPort))
            {
                m_CommAPI.SetEvent(closeEvent);

                isOpen = false;

                hPort = (IntPtr)CommAPI.INVALID_HANDLE_VALUE;

                m_CommAPI.SetEvent(closeEvent);

                return true;
            }

            return false;
        }

        /// <summary>
        /// The Port's output buffer.  Set this property to send data.
        /// </summary>
        public byte[] Output
        {
            set
            {
                if(!isOpen)
                    throw new CommPortException("Port not open");

                int written = 0;

                // more than threshold amount so send without buffering
                if(value.GetLength(0) > sthreshold)
                {
                    // first send anything already in the buffer
                    if(ptxBuffer > 0)
                    {
                        if (!m_CommAPI.WriteFile(hPort, txBuffer, ptxBuffer, ref written,
txOverlapped))
                        {
                            if (Marshal.GetLastWin32Error() == CommAPI.ERROR_IO_PENDING)
                            {
                                // See Geoff McIlraith's modification to the ReadFile call
in CommEventThread:
                                // Write operation was not completed on WriteFile call...
block until
                                // complete, and grab result.
                                if (!m_CommAPI.GetOverlappedResult(hPort, txOverlapped, out
written, true))
                                {
```

```
                                string errString = String.Format("GetOverlappedResult Failed: {0}", Marshal.GetLastWin32Error());
                                if(OnError != null)
                                    OnError(errString);

                                return;
                            }
                        }
                        else
                        {
                            string errString = String.Format("WriteFile Failed: {0}", Marshal.GetLastWin32Error());
                            if(OnError != null)
                                OnError(errString);

                            return;
                        }
                    }

                    ptxBuffer = 0;
                }

                if (!m_CommAPI.WriteFile(hPort, value, (int)value.GetLength(0), ref written, txOverlapped))
                {
                    if (Marshal.GetLastWin32Error() == CommAPI.ERROR_IO_PENDING)
                    {
                        // See Geoff McIlraith's modification to the ReadFile call in CommEventThread:
                        // Write operation was not completed on WriteFile call...block until
                        // complete, and grab result.
                        if (!m_CommAPI.GetOverlappedResult(hPort, txOverlapped, out written, true))
                        {
                            string errString = String.Format("GetOverlappedResult Failed: {0}", Marshal.GetLastWin32Error());
                            if(OnError != null)
                                OnError(errString);

                            return;
                        }
                    }
                    else
                    {
                        string errString = String.Format("WriteFile Failed: {0}", Marshal.GetLastWin32Error());
                        if(OnError != null)
                            OnError(errString);

                        return;
                    }
                }
            }
            else
            {
                // copy it to the tx buffer
                value.CopyTo(txBuffer, (int)ptxBuffer);
                ptxBuffer += (int)value.Length;

                // now if the buffer is above sthreshold, send it
                if(ptxBuffer >= sthreshold)
                {
                    if (!m_CommAPI.WriteFile(hPort, txBuffer, ptxBuffer, ref written, txOverlapped))
                    {
                        if (Marshal.GetLastWin32Error() == CommAPI.ERROR_IO_PENDING)
                        {
                            // See Geoff McIlraith's modification to the ReadFile call
```

```
in CommEventThread:
                            // Write operation was not completed on WriteFile call...  ↙
block until
                            // complete, and grab result.
                            if (!m_CommAPI.GetOverlappedResult(hPort, txOverlapped, out  ↙
written, true))
                            {
                                string errString = String.Format("GetOverlappedResult  ↙
Failed: {0}", Marshal.GetLastWin32Error());
                                if(OnError != null)
                                    OnError(errString);

                                return;
                            }
                        }
                        else
                        {
                            string errString = String.Format("WriteFile Failed: {0}",  ↙
Marshal.GetLastWin32Error());
                            if(OnError != null)
                                OnError(errString);

                            return;
                        }
                    }

                    ptxBuffer = 0;
                }
            }
        }
    }

    /// <summary>
    /// The Port's input buffer.  Incoming data is read from here and a read will pull  ↙
InputLen bytes from the buffer
    /// <seealso cref="InputLen"/>
    /// </summary>
    public byte[] Input
    {
        get
        {
            if(!isOpen) return null;

            int dequeueLength = 0;

            // lock the rx FIFO while reading
            rxBufferBusy.WaitOne();

            // how much data are we *actually* going to return from the call?
            if(inputLength == 0)
                dequeueLength = rxFIFO.Count;  // pull the entire buffer
            else
                dequeueLength = (inputLength < rxFIFO.Count) ? inputLength : rxFIFO.  ↙
Count;

            byte[] data = new byte[dequeueLength];

            // dequeue the data
            for(int p = 0 ; p < dequeueLength ; p++)
                data[p] = (byte)rxFIFO.Dequeue();

            // release the mutex so the Rx thread can continue
            rxBufferBusy.ReleaseMutex();

            return data;
        }
    }

    /// <summary>
```

```
        /// The length of the input buffer
        /// </summary>
        public int InputLen
        {
            get
            {
                return inputLength;
            }
            set
            {
                inputLength = value;
            }
        }

        /// <summary>
        /// The actual amount of data in the input buffer
        /// </summary>
        public int InBufferCount
        {
            get
            {
                if(!isOpen) return 0;

                return rxFIFO.Count;
            }
        }

        /// <summary>
        /// The actual amount of data in the output buffer
        /// </summary>
        public int OutBufferCount
        {
            get
            {
                if(!isOpen) return 0;

                return ptxBuffer;
            }
        }

        /// <summary>
        /// The number of bytes that the receive buffer must exceed to trigger a Receive ↙
    event
        /// </summary>
        public int RThreshold
        {
            get
            {
                return rthreshold;
            }
            set
            {
                rthreshold = value;
            }
        }

        /// <summary>
        /// The number of bytes that the transmit buffer must exceed to trigger a Transmit ↙
    event
        /// </summary>
        public int SThreshold
        {
            get
            {
                return sthreshold;
            }
            set
            {
                sthreshold = value;
```

```
        }
    }

    /// <summary>
    /// Send or check for a communications BREAK event
    /// </summary>
    public bool Break
    {
        get
        {
            if(!isOpen) return false;

            return (brk == 1);
        }
        set
        {
            if(!isOpen) return;
            if(brk < 0) return;
            if(hPort == (IntPtr)CommAPI.INVALID_HANDLE_VALUE) return;

            if (value)
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.SETBREAK))
                    brk = 1;
                else
                    throw new CommPortException("Failed to set break!");
            }
            else
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.CLRBREAK))
                    brk = 0;
                else
                    throw new CommPortException("Failed to clear break!");
            }
        }
    }

    /// <summary>
    /// Returns whether or not the current port support a DTR signal
    /// </summary>
    public bool DTRAvailable
    {
        get
        {
            return dtravail;
        }
    }

    /// <summary>
    /// Gets or sets the current DTR line state (true = 1, false = 0)
    /// </summary>
    public bool DTREnable
    {
        get
        {
            return (dtr == 1);
        }
        set
        {
            if(dtr < 0) return;
            if(hPort == (IntPtr)CommAPI.INVALID_HANDLE_VALUE) return;

            if (value)
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.SETDTR))
                    dtr = 1;
                else
                    throw new CommPortException("Failed to set DTR!");
            }
```

```
            else
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.CLRDTR))
                    dtr = 0;
                else
                    throw new CommPortException("Failed to clear DTR!");
            }
        }
    }

    /// <summary>
    /// Returns whether or not the current port support an RTS signal
    /// </summary>
    public bool RTSAvailable
    {
        get
        {
            return rtsavail;
        }
    }

    /// <summary>
    /// Gets or sets the current RTS line state (true = 1, false = 0)
    /// </summary>
    public bool RTSEnable
    {
        get
        {
            return (rts == 1);
        }
        set
        {
            if(rts < 0) return;
            if(hPort == (IntPtr)CommAPI.INVALID_HANDLE_VALUE) return;

            if (value)
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.SETRTS))
                    rts = 1;
                else
                    throw new CommPortException("Failed to set RTS!");
            }
            else
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.CLRRTS))
                    rts = 0;
                else
                    throw new CommPortException("Failed to clear RTS!");
            }
        }
    }
    /// <summary>
    /// Gets or sets the com port for IR use (true = 1, false = 0)
    /// </summary>

    public bool IREnable
    {
        get
        {
            return (setir == 1);
        }
        set
        {
            if(setir < 0) return;
            if(hPort == (IntPtr)CommAPI.INVALID_HANDLE_VALUE) return;

            if (value)
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.SETIR))
```

```
                    setir = 1;
                else
                    throw new CommPortException("Failed to set IR!");
            }
            else
            {
                if (m_CommAPI.EscapeCommFunction(hPort, CommEscapes.CLRIR))
                    setir = 0;
                else
                    throw new CommPortException("Failed to clear IR!");
            }
        }
    }

    /// <summary>
    /// Get or Set the Port's DetailedPortSettings
    /// </summary>
    public DetailedPortSettings DetailedSettings
    {
        get
        {
            return portSettings;
        }
        set
        {
            portSettings = value;
            UpdateSettings();
        }
    }

    /// <summary>
    /// Get or Set the Port's BasicPortSettings
    /// </summary>
    public BasicPortSettings Settings
    {
        get
        {
            return portSettings.BasicSettings;
        }
        set
        {
            portSettings.BasicSettings = value;
            UpdateSettings();
        }
    }

    /// <summary>
    /// <code>GetPortProperties initializes the commprop member of the port object</
code>
    /// </summary>
    /// <returns></returns>
    private bool GetPortProperties()
    {
        bool success;

        success=m_CommAPI.GetCommProperties(hPort,Capabilities);

        return (success);
    }

    private void CommEventThread()
    {
        CommEventFlags  eventFlags  = new CommEventFlags();
        byte[]          readbuffer  = new Byte[rxBufferSize];
        int             bytesread   = 0;
        AutoResetEvent  rxevent     = new AutoResetEvent(false);

        // specify the set of events to be monitored for the port.
        if(CommAPI.FullFramework)
```

```
            {
                m_CommAPI.SetCommMask(hPort, CommEventFlags.ALLPC);

                // set up the overlapped IO
                OVERLAPPED o = new OVERLAPPED();
                rxOverlapped = LocalAlloc(0x40, Marshal.SizeOf(o));
                o.Offset = 0;
                o.OffsetHigh = 0;
                o.hEvent = rxevent.Handle;
                Marshal.StructureToPtr(o, rxOverlapped, true);
            }
            else
            {
                m_CommAPI.SetCommMask(hPort, CommEventFlags.ALLCE);
            }


            try
            {
                // let Open() know we're started
                threadStarted.Set();

                >>>> thread loop <<<<
            } // try
            catch(Exception e)
            {
                if(rxOverlapped != IntPtr.Zero)
                    LocalFree(rxOverlapped);

                if(OnError != null)
                    OnError(e.Message);

                return;
            }
        }
    }
}
```

```
//=====================================================================================================
//
//      namespace OpenNETCF.IO.Ports.Enumerations
//      Copyright (c) 2005, OpenNETCF.org
//
//      This library is free software; you can redistribute it and/or modify it under
//      the terms of the OpenNETCF.org Shared Source License.
//
//      This library is distributed in the hope that it will be useful, but
//      WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//      FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//      for more details.
//
//      You should have received a copy of the OpenNETCF.org Shared Source License
//      along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//      If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//      email licensing@opennetcf.org.
//
//      For general enquiries, email enquiries@opennetcf.org or visit our website at:
//      http://www.opennetcf.org
//
//=====================================================================================================

using System;

namespace OpenNETCF.IO.Ports {
    /// <summary>
    /// Specifies the number of stop bits used on the <see cref="SerialPort"/> object.
    /// <para><b>New in v1.3</b></para>
    /// </summary>
    public enum StopBits {
        /// <summary>
        /// One stop bit is used
        /// </summary>
        One = 1,

        /// <summary>
        /// Three stop bits are used.
        /// </summary>
        OnePointFive = 3,

        /// <summary>
        /// Two stop bits are used.
        /// </summary>
        Two = 2
    }

    /// <summary>
    /// Specifies the control protocol used in establishing a serial port communication for ↙
    a <see cref="SerialPort"/> object.
    /// <para><b>New in v1.3</b></para>
    /// </summary>
    public enum Handshake {
        /// <summary>
        /// No control is used for the handshake.
        /// </summary>
        None = 0,

        /// <summary>
        /// Request-to-Send (RTS) hardware flow control is used. RTS is used to signal that ↙
    data is available for transmission.
        /// </summary>
        RequestToSend = 2,

        /// <summary>
        /// Both the Request-to-Send (RTS) hardware control and the XON/XOFF software ↙
    controls are used.
        /// </summary>
```

```
        RequestToSendXOnXOff = 3,

        /// <summary>
        /// The XON/XOFF software control protocol is used. XOFF is a software control sent ↙
    to stop the transmission of data and the XON control is sent to resume the transmission. ↙
     These controls are used instead of Request to Send (RTS) and Clear to Send (CTS)       ↙
    hardware controls.
        /// </summary>
        XOnXOff = 1
    }

    /// <summary>
    /// Specifies the parity bit for a <see cref="SerialPort"/> object.
    /// <para><b>New in v1.3</b></para>
    /// </summary>
    public enum Parity {
        /// <summary>
        /// Sets the parity bit so that the count of bits set is an even number.
        /// </summary>
        Even = 2,

        /// <summary>
        /// Leaves the parity bit set to 1.
        /// </summary>
        Mark = 3,

        /// <summary>
        /// No parity check occurs.
        /// </summary>
        None = 0,

        /// <summary>
        /// Sets the parity bit so that the count of bits set is an odd number.
        /// </summary>
        Odd = 1,

        /// <summary>
        /// Leaves the parity bit set to 0.
        /// </summary>
        Space = 4
    }

    /// <summary>
    /// <para><b>New in v1.3</b></para>
    /// </summary>
    public enum SerialData {
        Chars = 1,
        Eof = 2
    }

    /// <summary>
    /// Specifies errors that occur on the <see cref="SerialPort"/> object.
    /// <para><b>New in v1.3</b></para>
    /// </summary>
    /// <remarks>This enumeration is used with the <see cref="SerialPort.ErrorEvent"/> event ↙
    .</remarks>
    public enum SerialError {
        /// <summary>
        /// The hardware detected a framing error.
        /// </summary>
        Frame = 8,
        /// <summary>
        /// A character-buffer overrun has occurred.
        /// The next character is lost.
        /// </summary>
        Overrun = 2,
        /// <summary>
        /// An input buffer overflow has occurred.
        /// There is either no room in the input buffer, or a character was received after   ↙
```

```
    the end-of-file (EOF) character.
        /// </summary>
        RXOver = 1,
        /// <summary>
        /// The hardware detected a parity error.
        /// </summary>
        RXParity = 4,
        /// <summary>
        /// The application tried to transmit a character, but the output buffer was full.
        /// </summary>
        TXFull = 0x100
    }

    /// <summary>
    /// Specifies the type of change that occurred on the <see cref="SerialPort"/> object.
    /// <para><b>New in v1.3</b></para>
    /// </summary>
    /// <remarks>This enumeration is used with the <see cref="SerialPort.PinChangedEvent"/> ↙
    event.</remarks>
    public enum SerialPinChange {
        /// <summary>
        /// A break was detected on input.
        /// </summary>
        Break = 0x40,
        /// <summary>
        /// The Receive Line Signal Detect (RLSD) signal changed state.
        /// </summary>
        CDChanged = 0x20,
        /// <summary>
        /// The Clear to Send (CTS) signal changed state.
        /// </summary>
        CtsChanged = 8,
        /// <summary>
        /// The Data Set Ready (DSR) signal changed state.
        /// </summary>
        DsrChanged = 0x10,
        /// <summary>
        /// A ring indicator was detected.
        /// </summary>
        Ring = 0x100
    }
}
```

```
//=====================================================================================================
//
//      namespace OpenNETCF.IO.Serial.DCB
//      Copyright (c) 2003, OpenNETCF.org
//
//      This library is free software; you can redistribute it and/or modify it under
//      the terms of the OpenNETCF.org Shared Source License.
//
//      This library is distributed in the hope that it will be useful, but
//      WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//      FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//      for more details.
//
//      You should have received a copy of the OpenNETCF.org Shared Source License
//      along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//      If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//      email licensing@opennetcf.org.
//
//      For general enquiries, email enquiries@opennetcf.org or visit our website at:
//      http://www.opennetcf.org
//
//=====================================================================================================

using System;
using System.Text;
using System.IO;
using System.Runtime.InteropServices;
using System.Collections.Specialized;

namespace OpenNETCF.IO.Serial
{

    //
    // The Win32 DCB structure is implemented below in a C# class.
    //

    [StructLayout(LayoutKind.Sequential)]
    internal class DCB
    {
        private  UInt32 DCBlength;
        public   UInt32 BaudRate;
        BitVector32 Control;
        internal UInt16 wReserved;
        public   UInt16 XonLim;
        public   UInt16 XoffLim;
        public   byte   ByteSize;
        public   byte   Parity;
        public   byte   StopBits;
        public   sbyte  XonChar;
        public   sbyte  XoffChar;
        public   sbyte  ErrorChar;
        public   sbyte  EofChar;
        public   sbyte  EvtChar;
        internal UInt16 wReserved1;

        private readonly BitVector32.Section sect1;
        private readonly BitVector32.Section DTRsect;
        private readonly BitVector32.Section sect2;
        private readonly BitVector32.Section RTSsect;

        public DCB()
        {
            //
            // Initialize the length of the structure. Marshal.SizeOf returns
            // the size of the unmanaged object (basically the object that
            // gets marshalled).
            //
            this.DCBlength = (uint)Marshal.SizeOf(this);
```

```
            // initialize BitVector32
            Control=new BitVector32(0);

            // of the following 4 sections only 2 are needed
            sect1=BitVector32.CreateSection(0x0f);
            DTRsect=BitVector32.CreateSection(3,sect1); // this is where the DTR setting is ↙
    stored
            sect2=BitVector32.CreateSection(0x3f,DTRsect);
            RTSsect=BitVector32.CreateSection(3,sect2); // this is where the RTS setting is ↙
    stored
        }



        //
        // We need to have to define reserved fields in the DCB class definition
        // to preserve the size of the
        // underlying structure to match the Win32 structure when it is
        // marshaled. Use these fields to suppress compiler warnings.
        //
        internal void _SuppressCompilerWarnings()
        {
            wReserved +=0;
            wReserved1 +=0;
        }

        //
        // Enumeration for fDtrControl bit field.
        //
        public enum DtrControlFlags
        {
            Disable = 0,
            Enable =1 ,
            Handshake = 2
        }

        //
        // Enumeration for fRtsControl bit field.
        //
        public enum RtsControlFlags
        {
            Disable = 0,
            Enable = 1,
            Handshake = 2,
            Toggle = 3
        }

        // Helper constants for manipulating the bit fields.
        // these are defined as an enum in order to preserve memory
        [Flags]
        enum ctrlBit {
            fBinaryMask             = 0x001,
            fParityMask             = 0x0002,
            fOutxCtsFlowMask        = 0x0004,
            fOutxDsrFlowMask        = 0x0008,
            fDtrControlMask         = 0x0030,
            fDsrSensitivityMask     = 0x0040,
            fTXContinueOnXoffMask   = 0x0080,
            fOutXMask               = 0x0100,
            fInXMask                = 0x0200,
            fErrorCharMask          = 0x0400,
            fNullMask               = 0x0800,
            fRtsControlMask         = 0x3000,
            fAbortOnErrorMask       = 0x4000
        }

        // get and set of bool works with the underlying BitVector32
        // by using a mask for each bit field we can let the compiler
```

```
        // and JIT do the work
        //

        public bool fBinary
        {
            get { return (Control[(int)ctrlBit.fBinaryMask]); }
            set { Control[(int)ctrlBit.fBinaryMask]=value; }
        }
        public bool fParity
        {
            get { return (Control[(int)ctrlBit.fParityMask]); }
            set { Control[(int)ctrlBit.fParityMask]=value; }
        }
        public bool fOutxCtsFlow
        {
            get { return (Control[(int)ctrlBit.fOutxCtsFlowMask]); }
            set { Control[(int)ctrlBit.fOutxCtsFlowMask] = value; }
        }
        public bool fOutxDsrFlow
        {
            get { return (Control[(int)ctrlBit.fOutxDsrFlowMask]); }
            set { Control[(int)ctrlBit.fOutxDsrFlowMask]=value; }
        }

        // we have to use a segment because the width of the underlying information
        // is wider than just one bit
        public byte fDtrControl
        {
            get {return (byte)Control[DTRsect]; }
            set { Control[DTRsect]=(int)value; }
        }

        public bool fDsrSensitivity
        {
            get { return Control[(int)ctrlBit.fDsrSensitivityMask];}
            set { Control[(int)ctrlBit.fDsrSensitivityMask] = value; }
        }
        public bool fTXContinueOnXoff
        {
            get { return Control[(int)ctrlBit.fTXContinueOnXoffMask]; }
            set { Control[(int)ctrlBit.fTXContinueOnXoffMask]=value; }
        }
        public bool fOutX
        {
            get { return Control [(int)ctrlBit.fOutXMask]; }
            set { Control[(int)ctrlBit.fOutXMask]=value; }
        }
        public bool fInX
        {
            get { return Control[(int)ctrlBit.fInXMask]; }
            set { Control[(int)ctrlBit.fInXMask]=value; }
        }
        public bool fErrorChar
        {
            get { return Control[(int)ctrlBit.fErrorCharMask]; }
            set { Control[(int)ctrlBit.fErrorCharMask]=value; }
        }
        public bool fNull
        {
            get { return Control[(int)ctrlBit.fNullMask]; }
            set { Control[(int)ctrlBit.fNullMask]=value; }
        }

        // we have to use a segment because the width of the underlying information
        // is wider than just one bit
        public byte fRtsControl
        {
            get { return (byte)Control[RTSsect]; }
            set { Control[RTSsect]=(int)value; }
```

```
        }

        public bool fAbortOnError
        {
            get { return Control[(int)ctrlBit.fAbortOnErrorMask]; }
            set { Control[(int)ctrlBit.fAbortOnErrorMask]=value; }
        }

        //
        // Method to dump the DCB to take a look and help debug issues.
        //
        public override String ToString()
        {
            StringBuilder sb = new StringBuilder();

            sb.Append("DCB:\r\n");
            sb.AppendFormat(null, "  BaudRate:      {0}\r\n", BaudRate);
            sb.AppendFormat(null, "  Control:       0x{0:x}\r\n", Control.Data);
            sb.AppendFormat(null, "    fBinary:             {0}\r\n", fBinary);
            sb.AppendFormat(null, "    fParity:             {0}\r\n", fParity);
            sb.AppendFormat(null, "    fOutxCtsFlow:        {0}\r\n", fOutxCtsFlow);
            sb.AppendFormat(null, "    fOutxDsrFlow:        {0}\r\n", fOutxDsrFlow);
            sb.AppendFormat(null, "    fDtrControl:         {0}\r\n", fDtrControl);
            sb.AppendFormat(null, "    fDsrSensitivity:     {0}\r\n", fDsrSensitivity);
            sb.AppendFormat(null, "    fTXContinueOnXoff:   {0}\r\n", fTXContinueOnXoff);
            sb.AppendFormat(null, "    fOutX:               {0}\r\n", fOutX);
            sb.AppendFormat(null, "    fInX:                {0}\r\n", fInX);
            sb.AppendFormat(null, "    fNull:               {0}\r\n", fNull);
            sb.AppendFormat(null, "    fRtsControl:         {0}\r\n", fRtsControl);
            sb.AppendFormat(null, "    fAbortOnError:       {0}\r\n", fAbortOnError);
            sb.AppendFormat(null, "  XonLim:        {0}\r\n", XonLim);
            sb.AppendFormat(null, "  XoffLim:       {0}\r\n", XoffLim);
            sb.AppendFormat(null, "  ByteSize:      {0}\r\n", ByteSize);
            sb.AppendFormat(null, "  Parity:        {0}\r\n", Parity);
            sb.AppendFormat(null, "  StopBits:      {0}\r\n", StopBits);
            sb.AppendFormat(null, "  XonChar:       {0}\r\n", XonChar);
            sb.AppendFormat(null, "  XoffChar:      {0}\r\n", XoffChar);
            sb.AppendFormat(null, "  ErrorChar:     {0}\r\n", ErrorChar);
            sb.AppendFormat(null, "  EofChar:       {0}\r\n", EofChar);
            sb.AppendFormat(null, "  EvtChar:       {0}\r\n", EvtChar);

            return sb.ToString();
        }
    }
}
```

```
//=====================================================================================
//
//      OpenNETCF.IO.Serial.CommAPI
//      Copyright (c) 2003-2005, OpenNETCF.org
//
//      This library is free software; you can redistribute it and/or modify it under
//      the terms of the OpenNETCF.org Shared Source License.
//
//      This library is distributed in the hope that it will be useful, but
//      WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//      FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//      for more details.
//
//      You should have received a copy of the OpenNETCF.org Shared Source License
//      along with this library; if not, email licensing@opennetcf.org to request a copy.
//
//      If you wish to contact the OpenNETCF Advisory Board to discuss licensing, please
//      email licensing@opennetcf.org.
//
//      For general enquiries, email enquiries@opennetcf.org or visit our website at:
//      http://www.opennetcf.org
//
//=====================================================================================
using System;
using System.Runtime.InteropServices;
using System.Collections.Specialized;

namespace OpenNETCF.IO.Serial
{
    API structs and enums


    [StructLayout(LayoutKind.Sequential)]
    internal class CommStat
    {
        //
        // typedef struct _COMSTAT {
        //     DWORD fCtsHold : 1;
        //     DWORD fDsrHold : 1;
        //     DWORD fRlsdHold : 1;
        //     DWORD fXoffHold : 1;
        //     DWORD fXoffSent : 1;
        //     DWORD fEof : 1;
        //     DWORD fTxim : 1;
        //     DWORD fReserved : 25;
        //     DWORD cbInQue;
        //     DWORD cbOutQue;
        // } COMSTAT, *LPCOMSTAT;
        //
        private BitVector32 bitfield = new BitVector32(0); // UKI added for CLR bitfield
    support
        public UInt32 cbInQue    = 0;
        public UInt32 cbOutQue   = 0;

        // Helper constants for manipulating the bit fields.

        [Flags]
        private enum commFlags
        {
            fCtsHoldMask  = 0x01,
            fDsrHoldMask  = 0x02,
            fRlsdHoldMask = 0x04,
            fXoffHoldMask = 0x08,
            fXoffSentMask = 0x10,
            fEofMask      = 0x20,
            fTximMask     = 0x40

        };
```

```
        public bool fCtsHold
        {
            get { return bitfield[(int)commFlags.fCtsHoldMask]; }
            set { bitfield[(int)commFlags.fCtsHoldMask]=value; }
        }
        public bool fDsrHold
        {
            get { return bitfield[(int)commFlags.fDsrHoldMask]; }
            set { bitfield[(int)commFlags.fDsrHoldMask] = value; }
        }
        public bool fRlsdHold
        {
            get { return bitfield[(int)commFlags.fRlsdHoldMask]; }
            set { bitfield[(int)commFlags.fRlsdHoldMask]= value; }
        }
        public bool fXoffHold
        {
            get { return bitfield[(int)commFlags.fXoffHoldMask]; }
            set { bitfield[(int)commFlags.fXoffHoldMask]=value; }
        }
        public bool fXoffSent
        {
            get { return bitfield[(int)commFlags.fXoffSentMask]; }
            set { bitfield[(int)commFlags.fXoffSentMask]= value; }
        }
        public bool fEof
        {
            get { return bitfield[(int)commFlags.fEofMask]; }
            set { bitfield[(int)commFlags.fEofMask] = value; }
        }
        public bool fTxim
        {
            get { return bitfield[(int)commFlags.fTximMask]; }
            set { bitfield[(int)commFlags.fTximMask] = value; }
        }
    }

    CommAPI base class

    CE CompactFramework (cf) implementation for CommAPI

    Full Framework (aka Win) implementation for CommAPI
}
```