Austin Neidert and Cam Wrabel
April 17, 2017
USB Serial and RS-232 Serial Sample Application.pdf 1.0

USB Serial and RS-232 Serial Sample Application with Sample Code

The purpose of this application is to demonstrate how to communicate with a Microcom Model 485 printer using either of the supported serial interfaces (USB Serial or RS-232 Serial). Specific communication features demonstrated here include communication preparation, sending information to the printer, and receiving information from the printer. This document assumes that the desired serial interface is already selected on the printer as the communication interface being used. All sample code that appears in this document is included in the Microsoft Visual Studio 485USBSerial solution.

Step 1: Preparing Communication

Before data transfer between the host and the printer can occur, the communication endpoints must be set up. More specifically, before writing to the printer, a serial port is required to know where to write data. Likewise, before reading from the printer, a serial port is required to know where to read from. In order to create a serial port, the COM port that the printer is associated with must be known. The available COM ports should be detected and the list of COM ports should be populated upon opening the application. Detecting the available COM ports can be done using `SerialPort.GetPortNames`. The COM port to use for the Microcom Model 485 printer will depend on which port it is connected to and is not known by the application at runtime.

When connecting to a serial port, parameters must be specified so that the host machine and the device can effectively communicate. These include baud rate, data parity, data bits, stop bits, and flow control. The Microcom Model 485 printer requires these parameters must be set to 115200, none, 8, 1, and CTS/RTS respectively. To connect to a serial port a `SerialPort` object must be created, after which all of the aforementioned parameters can be set within the `SerialPort` object. Once that step is complete, the serial port can be opened with `SerialPort.Open`. The sample code below demonstrates this procedure.

```
public bool Open()
{
    try
    {
        if (m_serialPort.IsOpen)
        {
            keepReading = false;
            Close();
            readThread.Join();
        }
        m_serialPort.BaudRate = m_baudRate;
        m_serialPort.DataBits = m_dataBits;
        m_serialPort.Handshake = m_handshake;
        m_serialPort.Parity = m_parity;
        m_serialPort.PortName = m_portName;
        m_serialPort.StopBits = m_stopBits;
        //m_serialPort.DataReceived += new SerialDataReceivedEventHandler(m_serialPort_DataReceived);
        m_serialPort.Open();

        readThread = new Thread(readThreadEntry);
        readThread.Start();
```

```
    }
    catch { return false; }
    return true;
}
```

Step 2: Writing to the Printer

Writing to the printer can be done any time after the serial port is open by using the `SerialPort.Write` function. This can be done in-line with code as well as resulting from a button press. The `Write` function can take in a string or a byte array of data to be sent. The write procedure for the serial port sample application is shown below.

```
public bool WriteString(string data)
{
    try
    {
        m_serialPort.Write(data);
    }
    catch(InvalidOperationException ioe)
    {
        Console.WriteLine("A Invalid Operation exception occurred\n.");
        return false;
    }
    catch(ArgumentNullException ane)
    {
        Console.WriteLine("A Argument Null exception occurred\n.");
        return false;
    }
    catch(TimeoutException toe)
    {
        Console.WriteLine("A timeout exception occurred\n.");
        return false;
    }
    return true;
}

public bool WriteBytes(byte[] byteArray)
{
    try
    {
        m_serialPort.Write(byteArray, 0, byteArray.Length);
    }
    catch (InvalidOperationException ioe)
    {
        Console.WriteLine("A Invalid Operation exception occurred\n.");
        return false;
    }
    catch (ArgumentNullException ane)
    {
        Console.WriteLine("A Argument Null exception occurred\n.");
        return false;
    }
    catch (TimeoutException toe)
    {
        Console.WriteLine("A timeout exception occurred\n.");
        return false;
    }
    return true;
}
```

The data being sent can be of any length whether it be a string or a byte array. The functions above return a boolean value indicating success or failure. This should succeed as long as the serial port is open.

Step 3: Reading from the Printer

All receiving done in the serial sample app is done asynchronously, thus a separate thread is used to continuously read. This ensures that data will be read as soon as it is available. The function used to read data from the serial port is `SerialPort.Receive`. This function returns the number of bytes read and a byte array is passed in by reference to store the data read. If there is no data to read, this call will hang until there is data to be read. This is the reason to call `SerialPort.Receive` in a separate thread. The read thread entry is shown below.

```
public void readThreadEntry()
{
    // initialize buffer to hold received data
    byte[] buffer = new byte[m_serialPort.ReadBufferSize];
    while (keepReading)
    {
        int bytesRead = 0;
        try
        {
            bytesRead = m_serialPort.Read(buffer, 0, buffer.Length);

            tString = Encoding.ASCII.GetString(buffer, 0, bytesRead);
            asciiString = tString;
            asciiToHex(ref hexString, asciiString);

            // UNUSED in this sample application, but included for the purpose of example
            //check if string contains terminating character
            if (tString.IndexOf((char)m_terminator) > -1)
            {
                //if tString does contain terminator, we cannot assume that it is the last char received
                string workingString = tString.Substring(0, tString.IndexOf((char)m_terminator));

                //remove the data up to the terminator from tString
                tString = tString.Substring(tString.IndexOf((char)m_terminator));

                Console.WriteLine(workingString);
            }
            if (bytesRead > 0)
                form.threadSafeAddConsoleText(tString);
        }
        catch (Exception ex)
        {
            if (bytesRead < 1)
                keepReading = false;
        }
    }
    keepReading = true;
}
```