using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Text;

using System.IO.Ports;

using System.Threading;

using FTD2XX\_NET;

namespace RUSA.Tools

{

/// <summary>

/// An FTDI device wrapper to handle communications between PC and RUSA dongle

/// </summary>

public class RUSAPort : IDisposable

{

public enum SerialMode

{

FTDI,

COM

}

public enum PortName

{

COM1,

COM2,

COM3,

COM4

}

private SerialMode \_serialMode;

private SerialPort \_serialPort;

#region myVariables

private FTDI \_device = null;

private Thread \_rXThread = null;

private Queue<byte> \_rXQueue = new Queue<byte>();

private bool \_getHeader = true;

private bool \_getRest = false;

private byte \_getExpectedBytes = 0;

private bool \_debug = false;

private string \_serial;

public string Serial

{

get { return \_serial; }

set { \_serial = value; }

}

#endregion

/// <summary>

/// Default constructor. Don't forget to assign ReceiveMessage eventhandler

/// </summary>

public RUSAPort(string serialOverride = null)

{

\_device = new FTDI();

\_serialMode = SerialMode.FTDI;

}

public RUSAPort(SerialMode serialMode, string serialOverride = null)

{

\_device = new FTDI();

\_serialMode = serialMode;

}

/// <summary>

/// Just in case we can catch out the unended thread..

/// </summary>

~RUSAPort()

{

this.Close(); //just in case this wasn't done properly

}

/// <summary>

/// Returns true if the device is opened already, false otherwise

/// </summary>

public bool IsOpen

{

get { return \_device != null && \_device.IsOpen; }

}

/// <summary>

/// Enable this property to get console output of raw incoming bytes from the Dongle

/// </summary>

public bool ShowDebugInformationOnConsole

{

get { return \_debug; }

set { \_debug = value; }

}

/// <summary>

/// Exception handler

/// </summary>

/// <param name="error"></param>

public delegate void ExceptionHandler(Exception error);

/// <summary>

/// Fires when an exception is raised.

/// </summary>

public static event ExceptionHandler OnError;

private static void HandleException(Exception error)

{

if (OnError != null)

{

OnError(error);

}

}

/// <summary>

/// Discover all FTDI devices attached to the computer

/// </summary>

/// <returns>Array of device descriptions</returns>

public static FTDIDeviceInfoNode[] ListDevices()

{

FTDI myFtdiDevice = new FTDI();

FTDI.FT\_STATUS ftStatus;

UInt32 ftdiDeviceCount = 0;

List<FTDIDeviceInfoNode> lstDevices = new List<FTDIDeviceInfoNode>();

// Determine the number of FTDI devices connected to the machine

ftStatus = myFtdiDevice.GetNumberOfDevices(ref ftdiDeviceCount);

// Check status

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

throw new Exception("Error occured in RUSAPort: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

if (ftdiDeviceCount > 0)

{

FTDI.FT\_DEVICE\_INFO\_NODE[] ftdiDeviceList = new FTDI.FT\_DEVICE\_INFO\_NODE[ftdiDeviceCount];

ftStatus = myFtdiDevice.GetDeviceList(ftdiDeviceList);

if (ftStatus == FTDI.FT\_STATUS.FT\_OK)

{

for (UInt32 i = 0; i < ftdiDeviceCount; i++)

{

FTDIDeviceInfoNode aNode = new FTDIDeviceInfoNode();

aNode.Flags = ftdiDeviceList[i].Flags;

aNode.Type = ftdiDeviceList[i].Type.ToString();

aNode.ID = ftdiDeviceList[i].ID;

aNode.LocID = ftdiDeviceList[i].LocId;

aNode.SerialNumber = ftdiDeviceList[i].SerialNumber;

aNode.Description = ftdiDeviceList[i].Description;

lstDevices.Add(aNode);

}

}

}

return lstDevices.ToArray();

}

public void SelectChannel(int channelId)

{

byte[] toSend = new byte[2];

toSend[0] = 0xFF; //header

toSend[1] = (byte)(100 + channelId);

\_serialPort.Write(toSend, 0, 2);

}

/// <summary>

/// Send a message to the processor board with payload

/// </summary>

/// <param name="CommandByte">Command byte</param>

/// <param name="Payload">Array of payload bytes</param>

/// <param name="numBytesToSendFromPayload">Number of items to send from the payload array</param>

public void SendData(byte CommandByte, byte[] Payload, byte numBytesToSendFromPayload)

{

byte[] toSend;

switch (\_serialMode)

{

case SerialMode.FTDI:

try

{

FTDI.FT\_STATUS ftStatus = FTDI.FT\_STATUS.FT\_OK;

UInt32 numBytesWritten = 0;

if ((3 + numBytesToSendFromPayload) > 99) throw new Exception("Error occured in RUSAPort: Cannot send a message over 99 bytes in length.");

toSend = new byte[numBytesToSendFromPayload + 3];

toSend[0] = 0xFF; //header

toSend[1] = (byte)(numBytesToSendFromPayload + 1); //payload length

toSend[2] = CommandByte; //command

for (int i = 0; i < numBytesToSendFromPayload; i++) toSend[3 + i] = Payload[i];

// Note that the Write method is overloaded, so can write string or byte array data

ftStatus = \_device.Write(toSend, numBytesToSendFromPayload + 3, ref numBytesWritten);

if ((ftStatus != FTDI.FT\_STATUS.FT\_OK) || (numBytesWritten != (numBytesToSendFromPayload + 3)))

{

throw new Exception("Error occured in URSAport: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

}

catch (Exception error)

{

HandleException(error);

}

break;

case SerialMode.COM:

if ((3 + numBytesToSendFromPayload) > 99) throw new Exception("Error occured in RUSAPort: Cannot send a message over 99 bytes in length.");

toSend = new byte[numBytesToSendFromPayload + 3];

toSend[0] = 0xFF; //header

toSend[1] = (byte)(numBytesToSendFromPayload + 1); //payload length

toSend[2] = CommandByte; //command

for (int i = 0; i < numBytesToSendFromPayload; i++) toSend[3 + i] = Payload[i];

// Note that the Write method is overloaded, so can write string or byte array data

try

{

\_serialPort.Write(toSend, 0, numBytesToSendFromPayload + 3);

}

catch (Exception error)

{

throw new Exception("Error occured in RUSAport: " + error.Message);

}

break;

}

}

/// <summary>

/// Send a message to the processor board with exact payload array

/// </summary>

/// <param name="CommandByte"></param>

/// <param name="Payload"></param>

public void SendData(byte CommandByte, byte[] Payload)

{

try

{

if (Payload.Length > 254) throw new Exception("Error occured in RUSAPort: the Payload argument is too large");

SendData(CommandByte, Payload, (byte)Payload.Length);

}

catch (Exception error)

{

HandleException(error);

}

}

/// <summary>

/// Send a message to the processor board without a payload

/// </summary>

/// <param name="CommandByte">Command byte</param>

public void SendData(byte CommandByte)

{

try

{

SendData(CommandByte, new byte[] { }, 0);

}

catch (Exception error)

{

HandleException(error);

}

}

public void Open(PortName portName = PortName.COM1)

{

switch (\_serialMode)

{

case SerialMode.FTDI:

try

{

if (\_serial == string.Empty)

{

throw new Exception("No port serial provided");

}

OpenBySerialNumber(\_serial);

}

catch (Exception error)

{

HandleException(error);

}

break;

case SerialMode.COM:

try

{

\_serialPort = new SerialPort(Enum.GetName(typeof(PortName), portName), 4800);

\_serialPort.Open();

\_rXThread = new Thread(new ThreadStart(RXThread));

\_rXThread.Start();

}

catch (Exception error)

{

HandleException(error);

}

break;

}

}

/// <summary>

/// Open the communications to a specific device. Use the ListDevices function to discover available devices

/// </summary>

/// <param name="SerialNumber">The serial number of the FTDI device to connect to</param>

public void OpenBySerialNumber(string SerialNumber)

{

try

{

if (\_device != null) if (\_device.IsOpen)

throw new Exception("Error occured in RUSAPort: The device is already open");

FTDI.FT\_STATUS ftStatus = FTDI.FT\_STATUS.FT\_OK;

ftStatus = \_device.OpenBySerialNumber(SerialNumber);

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

throw new Exception("Error occured in RUSAPort: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

ftStatus = \_device.SetBaudRate(4800);

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

\_device.Close();

throw new Exception("Error occured in RUSAPort: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

// Set data characteristics - Data bits, Stop bits, Parity

ftStatus = \_device.SetDataCharacteristics(FTDI.FT\_DATA\_BITS.FT\_BITS\_8, FTDI.FT\_STOP\_BITS.FT\_STOP\_BITS\_1, FTDI.FT\_PARITY.FT\_PARITY\_NONE);

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

\_device.Close();

throw new Exception("Error occured in RUSAPort: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

// Set flow control - set RTS/CTS flow control

ftStatus = \_device.SetFlowControl(FTDI.FT\_FLOW\_CONTROL.FT\_FLOW\_NONE, 0x11, 0x13);

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

\_device.Close();

throw new Exception("Error occured in RUSAPort: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

// Set read timeout to 5 seconds, write timeout to infinite

ftStatus = \_device.SetTimeouts(1000, 0);

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

\_device.Close();

throw new Exception("Error occured in RUSAPort: " + Enum.GetName(typeof(FTDI.FT\_STATUS), ftStatus));

}

//we were successful, so start reading thread to process incoming bytes

\_device.Purge(FTDI.FT\_PURGE.FT\_PURGE\_RX); //empty the incoming bytes buffer

\_rXThread = new Thread(RXThread);

\_rXThread.Start();

}

catch (Exception error)

{

HandleException(error);

}

}

/// <summary>

/// Closes an open communication port

/// </summary>

public void Close()

{

switch (\_serialMode)

{

case SerialMode.FTDI:

try

{

if (\_rXThread != null)

{

\_rXThread.Abort(); //signal the RX thread to stop

if (\_rXThread.ThreadState != System.Threading.ThreadState.Stopped) \_rXThread.Join(); //wait for the thread to die

if (\_rXThread.ThreadState == System.Threading.ThreadState.Stopped) \_rXThread = null;

}

if (\_device != null) if (\_device.IsOpen) \_device.Close();

}

catch (Exception error)

{

HandleException(error);

}

break;

case SerialMode.COM:

try

{

if (\_rXThread != null)

{

\_rXThread.Abort(); //signal the RX thread to stop

if (\_rXThread.ThreadState != System.Threading.ThreadState.Stopped) \_rXThread.Join(); //wait for the thread to die

if (\_rXThread.ThreadState == System.Threading.ThreadState.Stopped) \_rXThread = null;

}

if (\_serialPort != null) if (\_serialPort.IsOpen) \_serialPort.Close();

}

catch (Exception error)

{

HandleException(error);

}

break;

}

}

private void RXThread()

{

UInt32 numBytesAvailable = 0;

UInt32 numBytesRead = 0;

byte[] readData = new byte[255];

switch (\_serialMode)

{

case SerialMode.FTDI:

FTDI.FT\_STATUS ftStatus = FTDI.FT\_STATUS.FT\_OK;

while (\_device.IsOpen) //until we're interrupted

{

ftStatus = \_device.GetRxBytesAvailable(ref numBytesAvailable);

if (ftStatus != FTDI.FT\_STATUS.FT\_OK)

{

// Wait for a key press

\_device.Close();

throw new Exception("RUSAPort: Failed to communicate with the device (error code: " + ftStatus.ToString() + ")");

//Try to handle things more carefully

//throw new Exception("URSAport: Failed to get number of bytes available to read (error " + ftStatus.ToString() + ")");

}

if (numBytesAvailable > 0)

{

ftStatus = \_device.Read(readData, numBytesAvailable, ref numBytesRead);

if (numBytesRead > 0)

for (int i = 0; i < numBytesRead; i++) processBytes(readData[i]);

}

Thread.Sleep(1);

}

break;

case SerialMode.COM:

while (\_serialPort.IsOpen) //until we're interrupted

{

if (!\_serialPort.IsOpen)

{

// Wait for a key press

Close();

throw new Exception("RUSAPort: Failed to communicate with the device (error code: port closed)");

//Try to handle things more carefully

//throw new Exception("URSAport: Failed to get number of bytes available to read (error " + ftStatus.ToString() + ")");

}

if (\_serialPort.BytesToRead > 0)

{

int bytesToRead = \_serialPort.Read(readData, 0, readData.Length);

if (bytesToRead > 0)

for (int i = 0; i < bytesToRead; i++) processBytes(readData[i]);

}

Thread.Sleep(1);

}

break;

}

}

private void processBytes(byte p)

{

if (\_getHeader && !\_getRest)

{

if (\_debug) Console.Write("\n");

if (p == 255) \_getRest = true;

\_rXQueue.Clear();

} else

if (\_getHeader && \_getRest)

{

\_getExpectedBytes = p;

\_getHeader = false;

} else

if (\_getRest)

{

\_rXQueue.Enqueue(p);

if (\_rXQueue.Count == \_getExpectedBytes)

{

\_getRest = false; \_getHeader = true;

//fire event

FireMessageReceived(\_rXQueue.ToArray());

} else

if (\_rXQueue.Count > \_getExpectedBytes)

{

//there was an error, so lets do header only

\_getRest = false; \_getHeader = true;

}

}

if (\_debug) Console.Write(string.Format("{0:x2}", p));

}

/// <summary>

/// EventHandler for the MessageReceived event

/// </summary>

/// <param name="sender">Reference to the instance of URSAport</param>

/// <param name="message">Payload of the message 0-RSSI, 1-Type, 2->payload</param>

public delegate void MessageReceivedHandler(object sender, byte[] message);

/// <summary>

/// This event is fired when a complete message has been received from the URSA dongle

/// </summary>

public event MessageReceivedHandler OnMessageReceived;

/// <summary>

/// Function to call in order to fire the event

/// </summary>

/// <param name="data"></param>

protected virtual void FireMessageReceived(byte[] data)

{

if (OnMessageReceived != null)

{

OnMessageReceived(this, data);

}

}

public bool ScanForFTDISerial()

{

return ScanForFTDISerial(null);

}

public bool ScanForFTDISerial(string serialOverride)

{

string selectedNodeSerial = string.Empty;

FTDIDeviceInfoNode[] nodes = RUSAPort.ListDevices();

// Check for an FTDI device

if (nodes.Length == 0)

{

throw new Exception("No FTDI nodes found, could not detect pc dongle");

}

//We can assume that only one device is connected unless serial is provided

if (serialOverride != null)

{

foreach (FTDIDeviceInfoNode node in nodes)

{

if (node.SerialNumber == serialOverride)

{

selectedNodeSerial = node.SerialNumber;

}

}

if (selectedNodeSerial == string.Empty)

{

throw new Exception("Could not find serial override specified");

}

}

else

{

//Look for first FTDI node connected to the system

selectedNodeSerial = ((FTDIDeviceInfoNode)nodes.GetValue(nodes.GetLowerBound(0))).SerialNumber;

}

\_serial = selectedNodeSerial;

return \_serial != string.Empty;

}

// Double sure we close the port and abort the thread

public void Dispose()

{

this.Close();

}

}

/// <summary>

/// Structure which holds FTDI Device relevant information

/// </summary>

public struct FTDIDeviceInfoNode

{

/// <summary>

/// Device Flags - refer to the FTDI user manual for further information

/// </summary>

public uint Flags;

/// <summary>

/// Device type - the Name of the FTDI device

/// </summary>

public string Type;

/// <summary>

/// Device ID

/// </summary>

public uint ID;

/// <summary>

/// Device's USB LocationID

/// </summary>

public uint LocID;

/// <summary>

/// Device's SerialNumber, unique to each device

/// </summary>

public string SerialNumber;

/// <summary>

/// Device Description

/// </summary>

public string Description;

}

}