E-SEEK's Technical Note

Note #: 1910-01

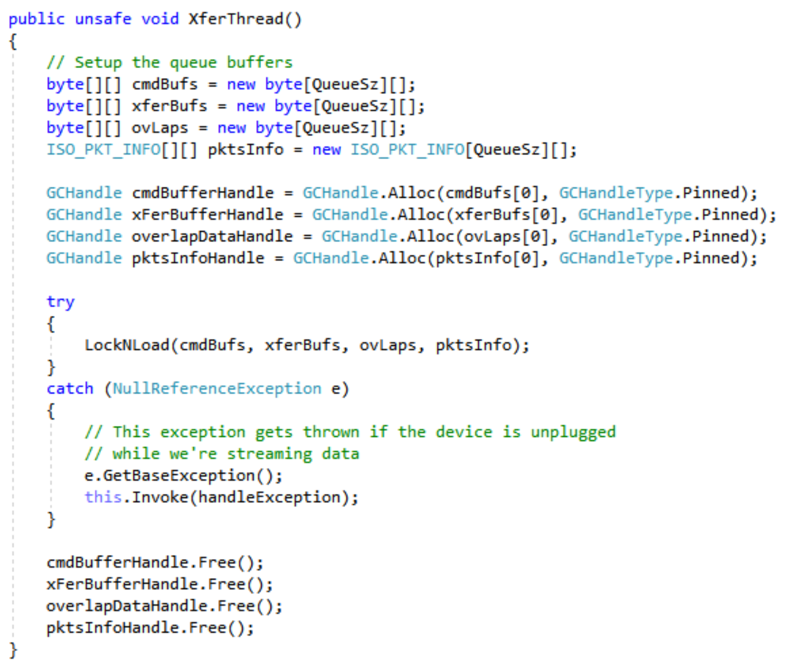
Date: 10/23/2019

Title: Issues about .NET framework.

Issue: Application is updated due to the issue with .NET framework 3.5 or higher on 64bit.

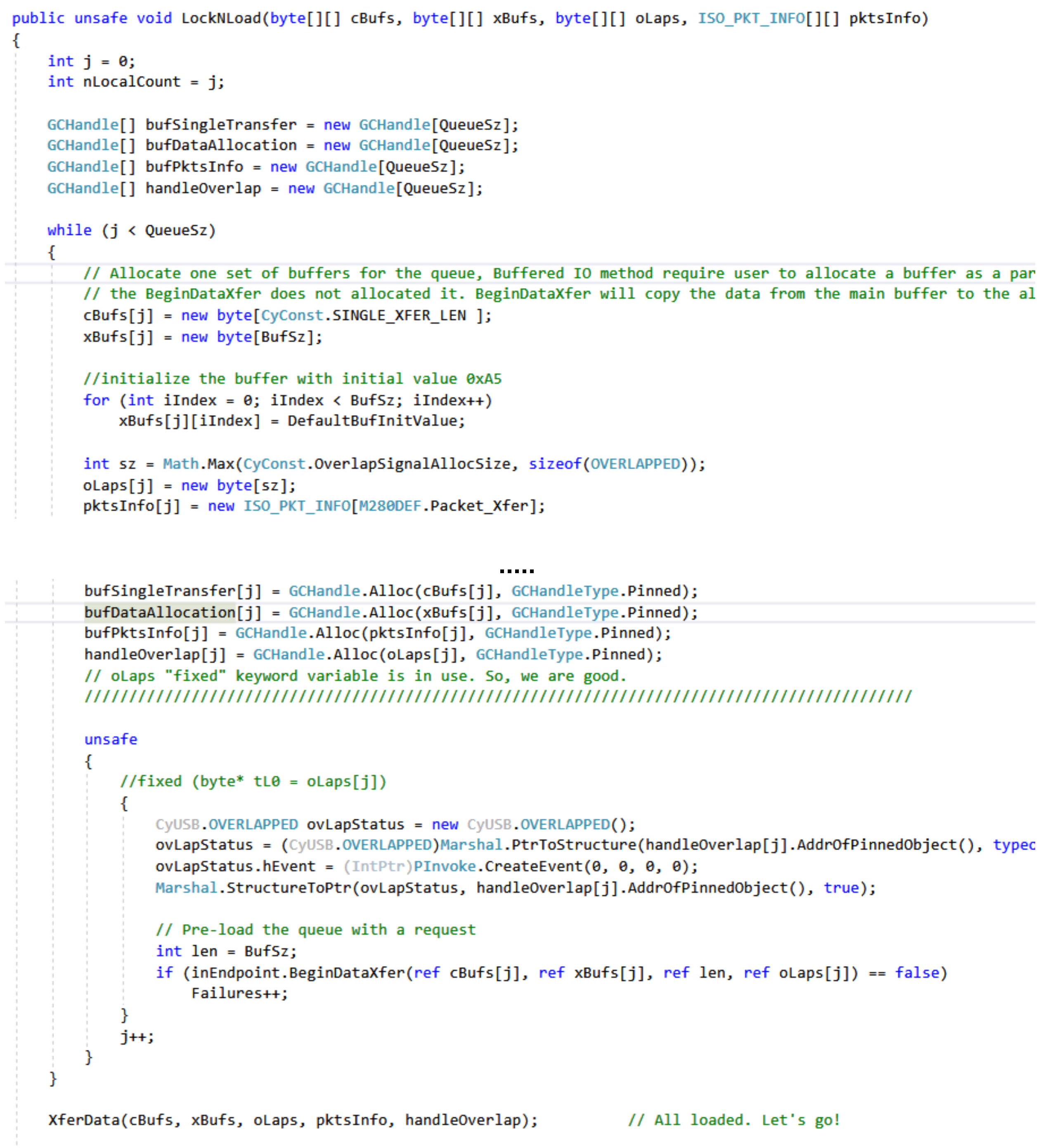
Solution: Please modify the source code in the corresponding functions below in the application software (from the V3.30.00-beta Sample App).

Form1.cs (line 212~241), XferThread()

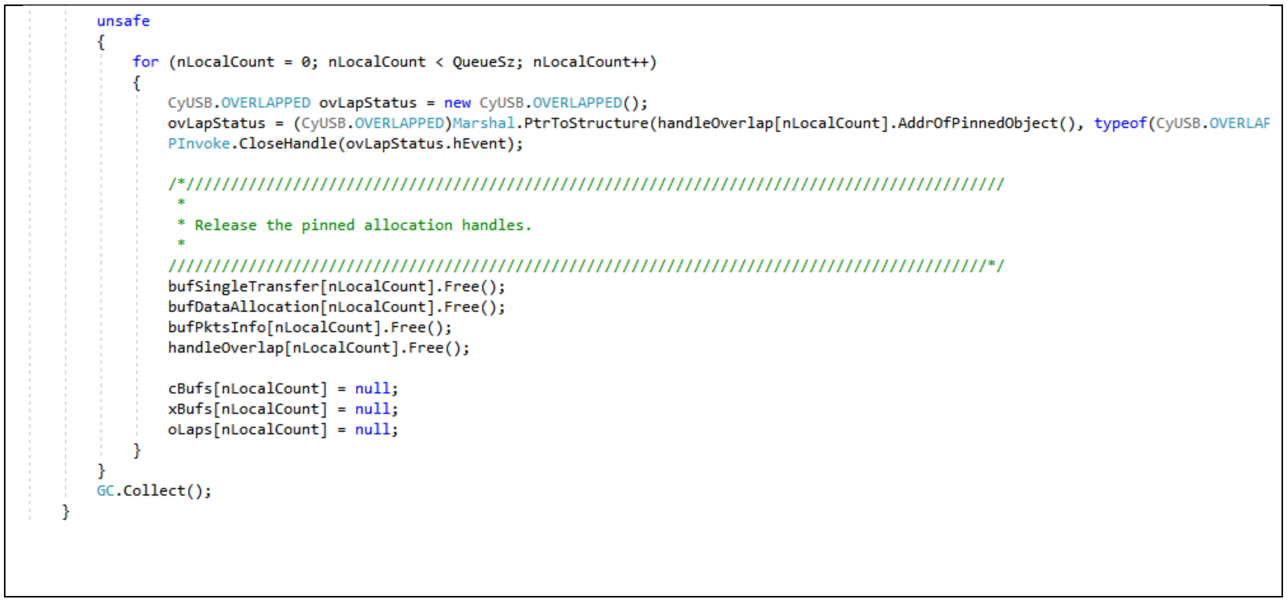


```
public unsafe void XferThread()
{
    // Setup the queue buffers
    byte[][] cmdBufs = new byte[QueueSz][];
    byte[][] xferBufs = new byte[QueueSz][];
    byte[][] ovLaps = new byte[QueueSz][];
    ISO_PKT_INFO[][] pktsInfo = new ISO_PKT_INFO[QueueSz][];

    GCHandle cmdBufferHandle = GCHandle.Alloc(cmdBufs[0], GCHandleType.Pinned);
    GCHandle xFerBufferHandle = GCHandle.Alloc(xferBufs[0], GCHandleType.Pinned);
    GCHandle overlapDataHandle = GCHandle.Alloc(ovLaps[0], GCHandleType.Pinned);
    GCHandle pktsInfoHandle = GCHandle.Alloc(pktsInfo[0], GCHandleType.Pinned);

    try
    {
        LockNLoad(cmdBufs, xferBufs, ovLaps, pktsInfo);
    }
    catch (NullReferenceException e)
    {
        // This exception gets thrown if the device is unplugged
        // while we're streaming data
        e.GetBaseException();
        this.Invoke(handleException);
    }

    cmdBufferHandle.Free();
    xFerBufferHandle.Free();
    overlapDataHandle.Free();
    pktsInfoHandle.Free();
}
```

Form1.cs (line 248~343), LockNLoad()

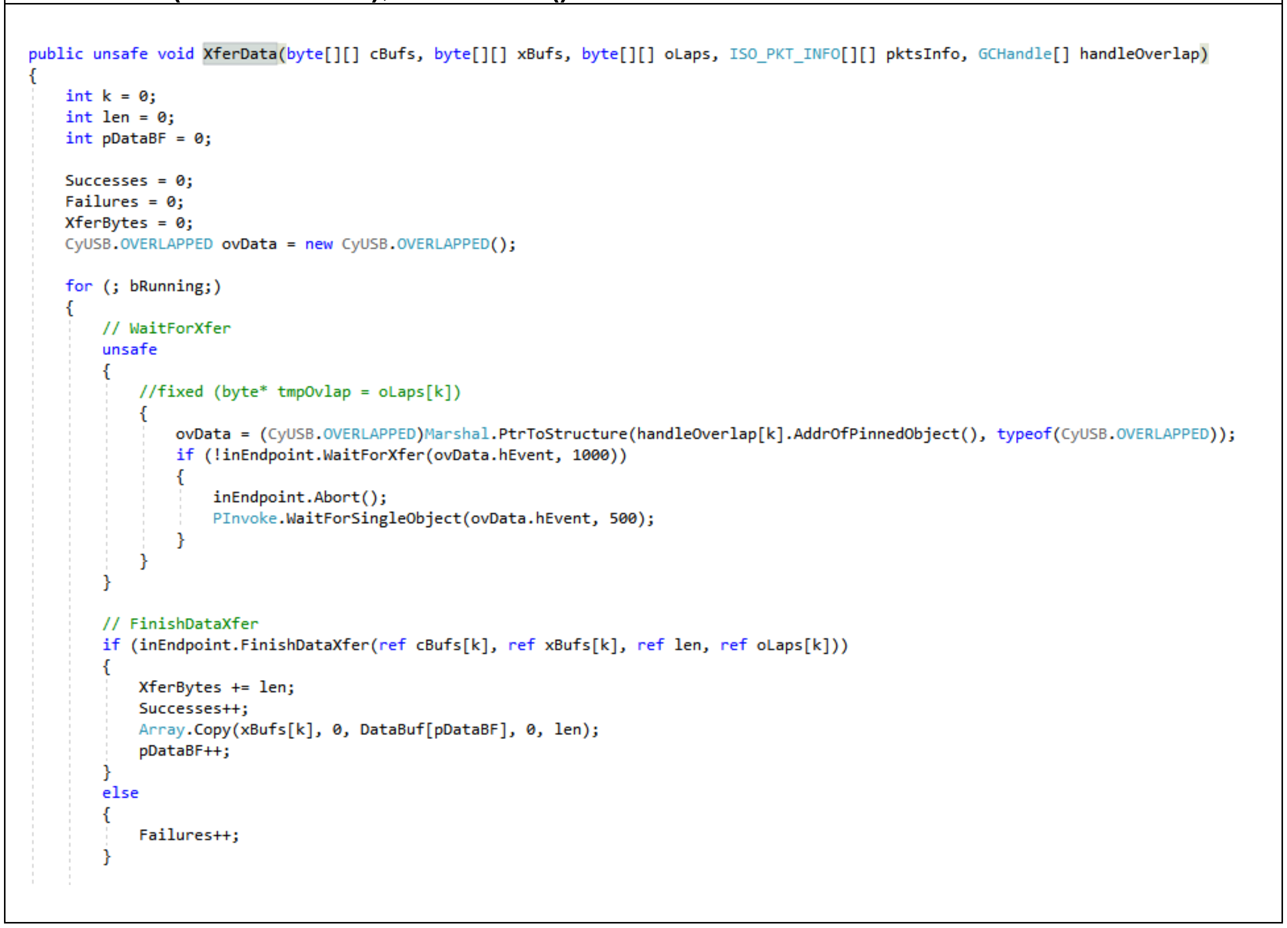


```
public unsafe void LockNLoad(byte[][] cBufs, byte[][] xBufs, byte[][] oLaps, ISO_PKT_INFO[][] pktsInfo)
{
    int j = 0;
    int nLocalCount = j;

    GCHandle[] bufSingleTransfer = new GCHandle[QueueSz];
    GCHandle[] bufDataAllocation = new GCHandle[QueueSz];
    GCHandle[] bufPktsInfo = new GCHandle[QueueSz];
    GCHandle[] handleOverlap = new GCHandle[QueueSz];

    while (j < QueueSz)
    {
        // Allocate one set of buffers for the queue, Buffered IO method require user to allocate a buffer as a par
        // the BeginDataXfer does not allocated it. BeginDataXfer will copy the data from the main buffer to the al
        cBufs[j] = new byte[CyConst.SINGLE_XFER_LEN ];
        xBufs[j] = new byte[BufSz];

        //initialize the buffer with initial value 0xA5
        for (int iIndex = 0; iIndex < BufSz; iIndex++)
            xBufs[j][iIndex] = DefaultBufInitValue;

        int sz = Math.Max(CyConst.OverlapSignalAllocSize, sizeof(OVERLAPPED));
        oLaps[j] = new byte[sz];
        pktsInfo[j] = new ISO_PKT_INFO[M280DEF.Packet_Xfer];


                                      .....
        bufSingleTransfer[j] = GCHandle.Alloc(cBufs[j], GCHandleType.Pinned);
        bufDataAllocation[j] = GCHandle.Alloc(xBufs[j], GCHandleType.Pinned);
        bufPktsInfo[j] = GCHandle.Alloc(pktsInfo[j], GCHandleType.Pinned);
        handleOverlap[j] = GCHandle.Alloc(oLaps[j], GCHandleType.Pinned);
        // oLaps "fixed" keyword variable is in use. So, we are good.
        /////////////////////////////////////////////////////////////////////////////////////////////

        unsafe
        {
            //fixed (byte* tL0 = oLaps[j])
            {
                CyUSB.OVERLAPPED ovLapStatus = new CyUSB.OVERLAPPED();
                ovLapStatus = (CyUSB.OVERLAPPED)Marshal.PtrToStructure(handleOverlap[j].AddrOfPinnedObject(), typec
                ovLapStatus.hEvent = (IntPtr)PInvoke.CreateEvent(0, 0, 0, 0);
                Marshal.StructureToPtr(ovLapStatus, handleOverlap[j].AddrOfPinnedObject(), true);

                // Pre-load the queue with a request
                int len = BufSz;
                if (inEndpoint.BeginDataXfer(ref cBufs[j], ref xBufs[j], ref len, ref oLaps[j]) == false)
                    Failures++;
            }
            j++;
        }
    }

    XferData(cBufs, xBufs, oLaps, pktsInfo, handleOverlap);          // All loaded. Let's go!
```

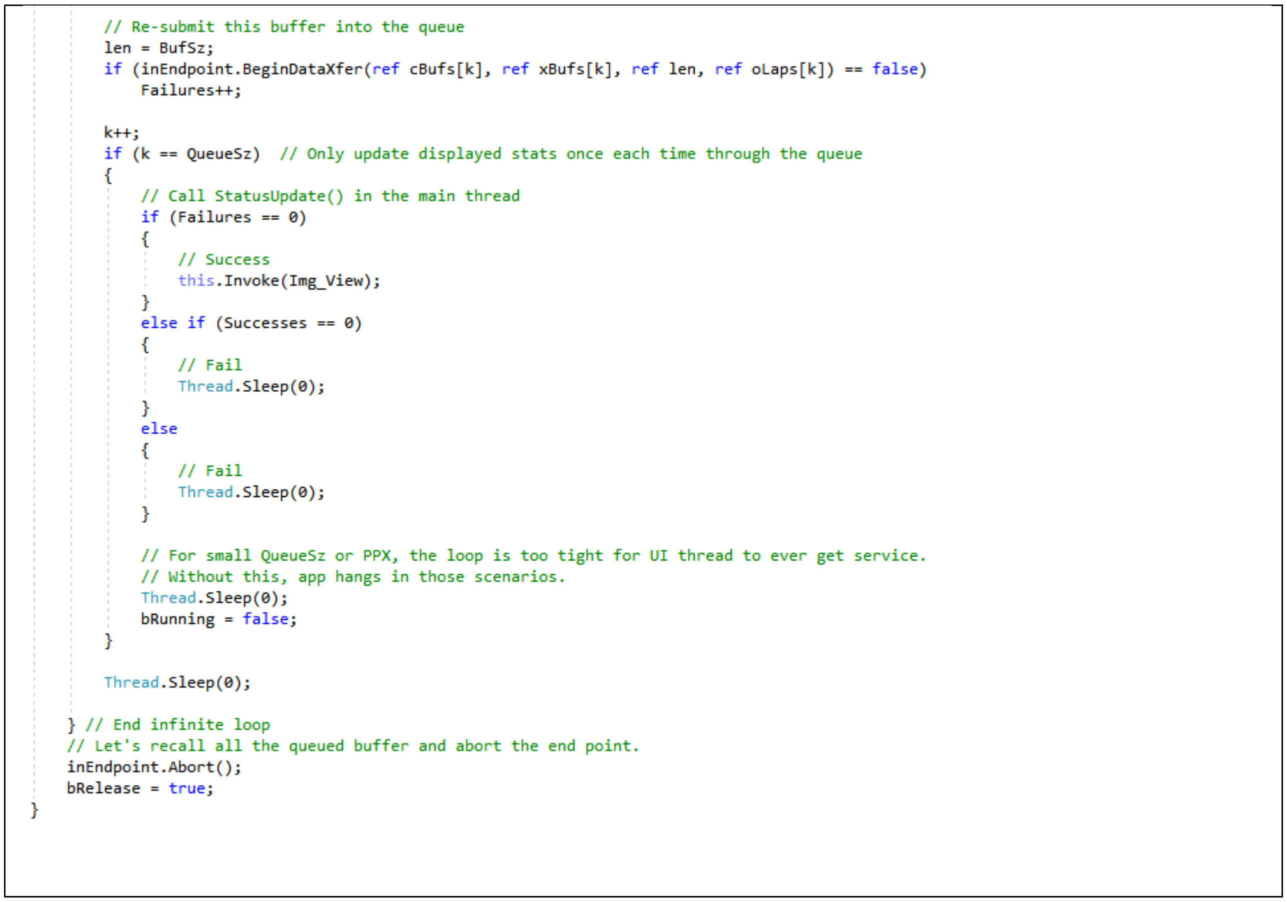


```
        unsafe
        {
            for (nLocalCount = 0; nLocalCount < QueueSz; nLocalCount++)
            {
                CyUSB.OVERLAPPED ovLapStatus = new CyUSB.OVERLAPPED();
                ovLapStatus = (CyUSB.OVERLAPPED)Marshal.PtrToStructure(handleOverlap[nLocalCount].AddrOfPinnedObject(), typeof(CyUSB.OVERLAP
                PInvoke.CloseHandle(ovLapStatus.hEvent);

                /*/////////////////////////////////////////////////////////////////////////////////////
                 *
                 * Release the pinned allocation handles.
                 *
                ///////////////////////////////////////////////////////////////////////////////////*/
                bufSingleTransfer[nLocalCount].Free();
                bufDataAllocation[nLocalCount].Free();
                bufPktsInfo[nLocalCount].Free();
                handleOverlap[nLocalCount].Free();

                cBufs[nLocalCount] = null;
                xBufs[nLocalCount] = null;
                oLaps[nLocalCount] = null;
            }
        }
        GC.Collect();
    }
```

Form1.cs (line 348~425), XFerData()



```
public unsafe void XferData(byte[][] cBufs, byte[][] xBufs, byte[][] oLaps, ISO_PKT_INFO[][] pktsInfo, GCHandle[] handleOverlap)
{
    int k = 0;
    int len = 0;
    int pDataBF = 0;

    Successes = 0;
    Failures = 0;
    XferBytes = 0;
    CyUSB.OVERLAPPED ovData = new CyUSB.OVERLAPPED();

    for (; bRunning;)
    {
        // WaitForXfer
        unsafe
        {
            //fixed (byte* tmpOvlap = oLaps[k])
            {
                ovData = (CyUSB.OVERLAPPED)Marshal.PtrToStructure(handleOverlap[k].AddrOfPinnedObject(), typeof(CyUSB.OVERLAPPED));
                if (!inEndpoint.WaitForXfer(ovData.hEvent, 1000))
                {
                    inEndpoint.Abort();
                    PInvoke.WaitForSingleObject(ovData.hEvent, 500);
                }
            }
        }

        // FinishDataXfer
        if (inEndpoint.FinishDataXfer(ref cBufs[k], ref xBufs[k], ref len, ref oLaps[k]))
        {
            XferBytes += len;
            Successes++;
            Array.Copy(xBufs[k], 0, DataBuf[pDataBF], 0, len);
            pDataBF++;
        }
        else
        {
            Failures++;
        }
```



```
        // Re-submit this buffer into the queue
        len = BufSz;
        if (inEndpoint.BeginDataXfer(ref cBufs[k], ref xBufs[k], ref len, ref oLaps[k]) == false)
            Failures++;

        k++;
        if (k == QueueSz)  // Only update displayed stats once each time through the queue
        {
            // Call StatusUpdate() in the main thread
            if (Failures == 0)
            {
                // Success
                this.Invoke(Img_View);
            }
            else if (Successes == 0)
            {
                // Fail
                Thread.Sleep(0);
            }
            else
            {
                // Fail
                Thread.Sleep(0);
            }

            // For small QueueSz or PPX, the loop is too tight for UI thread to ever get service.
            // Without this, app hangs in those scenarios.
            Thread.Sleep(0);
            bRunning = false;
        }

        Thread.Sleep(0);

    } // End infinite loop
    // Let's recall all the queued buffer and abort the end point.
    inEndpoint.Abort();
    bRelease = true;
}
```