

Implementing POS for .NET MSR for the TabletKiosk® eo a7330D

By Sean D. Liming
Managing Director
SJJ Embedded Micro Solutions

January 2010

The TabletKiosk® eo a7330D Ultra-Mobile PC features the new Intel® ATOM™ Z530P processor, and packs a wealth of features such as Dual-Band WiFi® 802.a/b/g/n, GPS, Bluetooth®, USB, option 3G/3.5G WWAN module, auto-switching dual mode digitizer / touch screen, and a 7-inch display. It's most unique feature is a new module expansion system to support Point Of Service (POS) devices like a magnetic strip reader (MSR) and a barcode scanner.



Figure 1 - TabletKiosk eo a730D with MSR Module

The eo a7330D with an MSR or barcode reader is ideal for different mobile point of service applications such inventory management, rental car return, hospitably management, and security. Developers can take advantage of Microsoft's POS for .NET to manage the POS devices.

POS for .NET is a .NET Framework implementation of the UPOS specification. UPOS is designed to separate the application from the hardware using an abstraction layer. The abstraction layer in POS for .NET is either a service object or a legacy OPOS driver. The goal of the abstraction layer allows application developers to write POS applications once, and then have them support many devices. In this paper, we will look at how to create a POS for .NET application that takes advantage of the Magtek MSR expansion module interfaced to the TabletKiosk eo a730D.

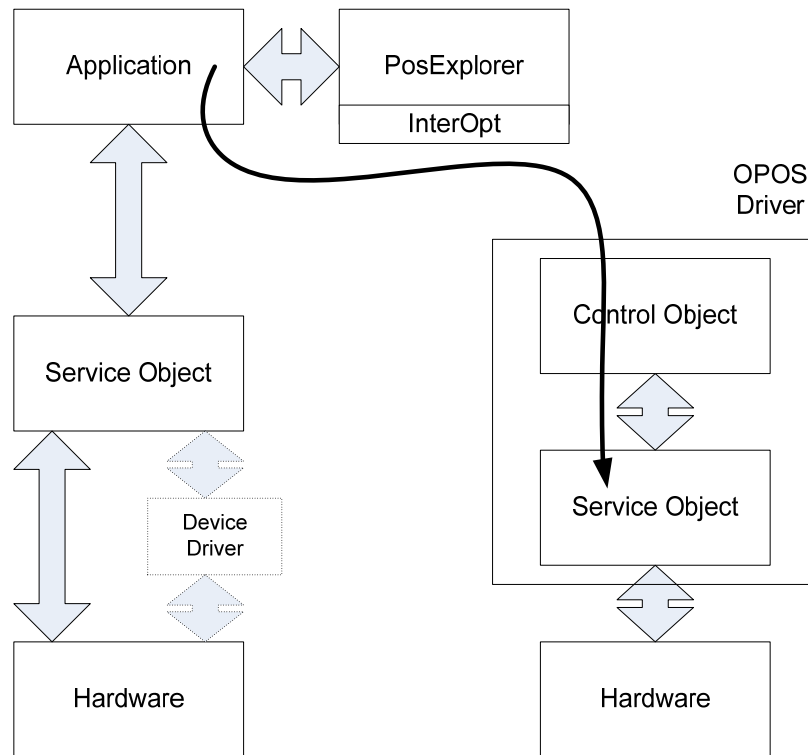


Figure 2 - POS for .NET Architecture

For more information on POS for .NET, please see Microsoft's website: <http://www.microsoft.com/windowsembedded/en-us/products/readyproducts/posready/overview.msp>

To develop the application in this paper, the follow software is required:

- Visual Studio 2005 or Higher
- POS for .NET 1.12 SDK
- .NET Framework 3.5 or higher
- Option: Magtek UPOS / Service Object for POS for .NET 1.12 downloadable from the Magtek website (<http://www.magtek.com/>).

1.1 Magtek MSR Expansion Module

The Magtek MSR expansion module for the eo a7330D is implemented as a pure USB HID device. USB HID implementation is ideal to take advantage of the POS for .NET device and data events. The alternative is a keyboard wedge that cannot take advantage of the POS for .NET features or capability, and the user has to pay attention to control focus to capture data.

When the MSR module is plugged into the eo a7330D, two different devices show up: USB Human Interface Device and a COM port. The interface hardware to the Magtek OEM MSR adds the COM port. The extra COM port has no affect on the USB-HID MSR and can be ignored.

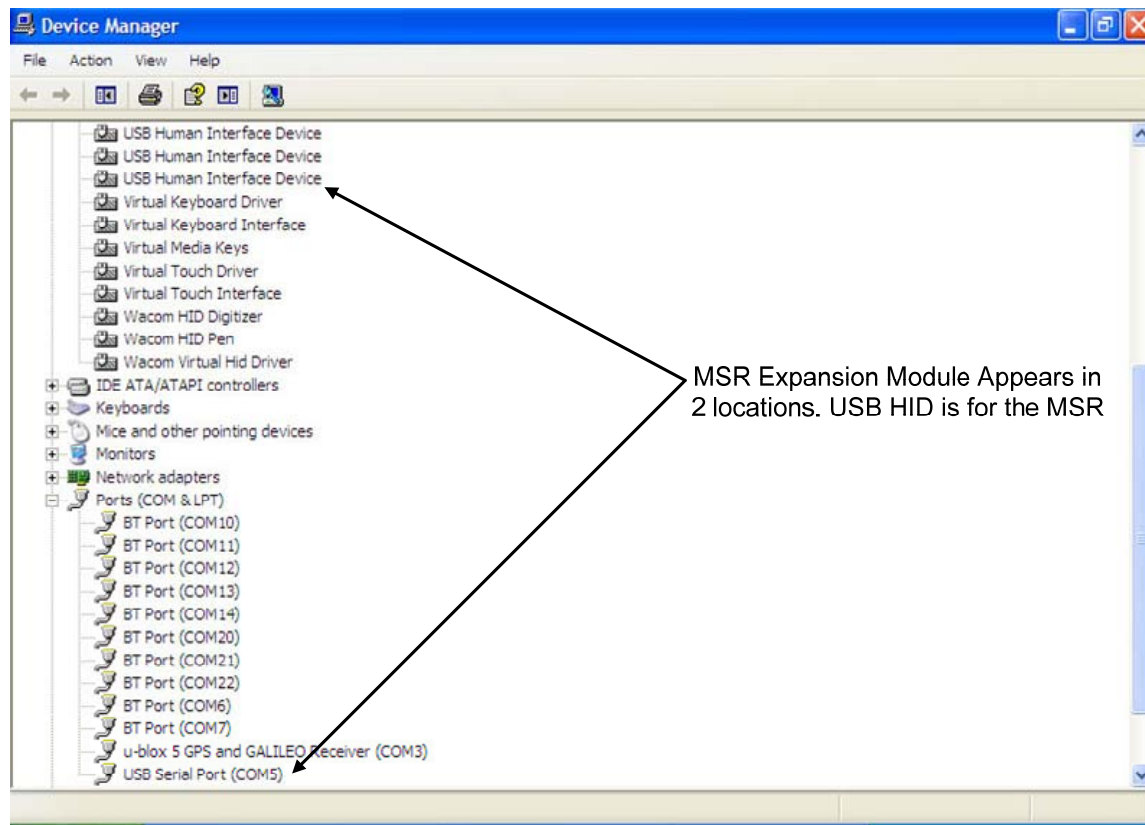


Figure 3 - MSR Expansion Module Shows up as two Different Devices

1.2 POS for .NET Service Object Setup

To take advantage of the POS for .NET capability, a service object or OPOS driver is needed for each POS device to be accessed by the POS application. The POS for .NET SDK comes with a sample service object for MSR and barcode scanners, and the sample MSR service object supports the Magtek MSR expansion module. There is also a service object available from Magtek that also works with the MSR expansion module. The following sections show how to setup both service objects.

1.2.1 Sample MSR Service Object Setup

To use the SDK's sample service object, an XML configuration file is needed to link the USB HID hardware ID to the example service object.

1. Make sure POS for .NET SDK has been installed on the eo a7330D.
2. Unplug the MSR USB device from the system.
3. Open Control Panel.
4. Open the System control panel applet.
5. Click on the Hardware tab.
6. Click on the Device Manager button.
7. Plug the MSR USB device into an open USB port. Device Manager should update with two new devices listed under Human Interface Devices.
8. Open the Properties of the USB Human Interface Device for the MSR.
9. Click on the Details tab.
10. In the drop-down, select Hardware Ids.

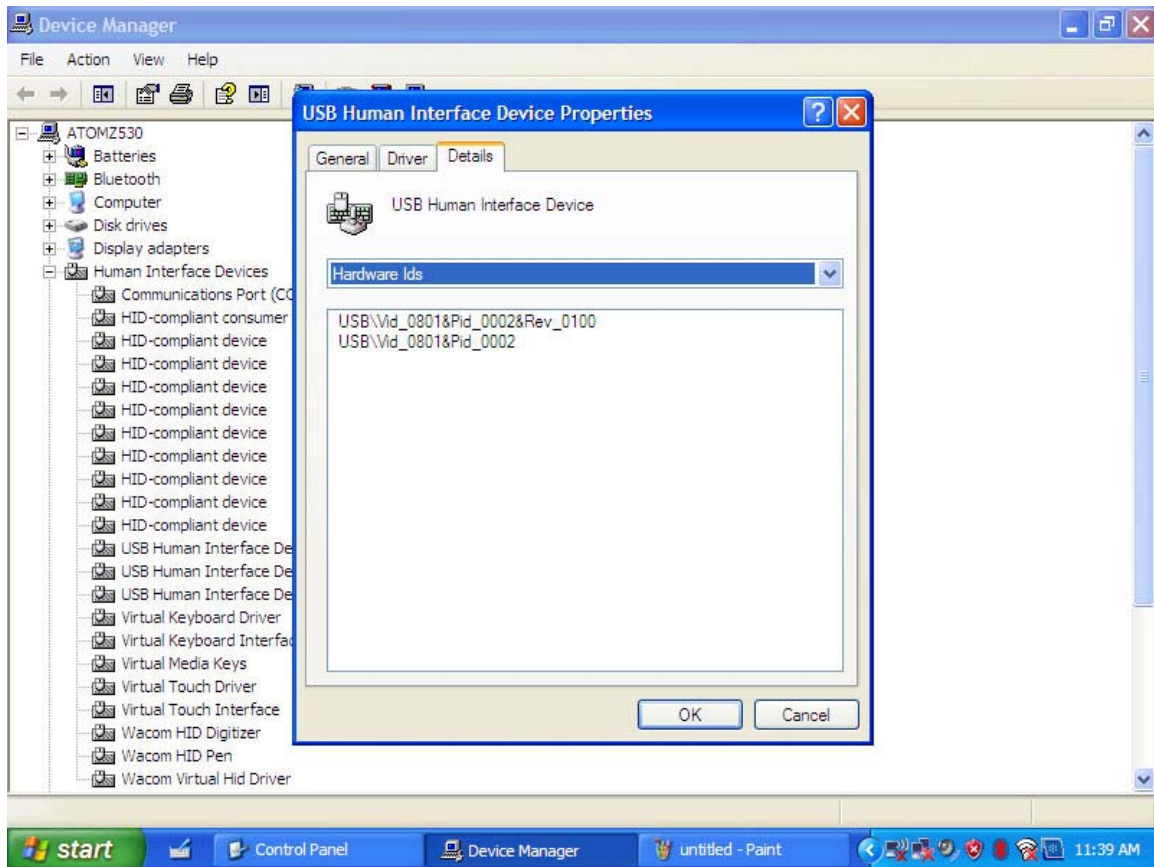


Figure 4 - Obtaining the Hardware IDs

The hardware IDs for this example are:

- HID\vid_0801&pid_0002
- HID\vid_0801&pid_0002&rev_0100

Now that we have the hardware IDs, let's create the XML file.

11. Open Notepad.
12. Enter the following:

```
<PointOfServiceConfig Version="1.0">
  <ServiceObject Type="Msr" Name="ExampleMsr">
    <HardwareId From="HID\vid_0801&pid_0002"
      To="HID\vid_0801&pid_0002" />
    <HardwareId From="HID\vid_0801&pid_0002&rev_0100"
      To="HID\vid_0801&pid_0002&rev_0100" />
  </ServiceObject>
</PointOfServiceConfig>
```

13. Save the file to the "C:\Program Files\Common Files\Microsoft Shared\Point Of Service\Control Configurations" folder, and name the file MAGTEK_msr.xml.
14. Use POSDM or SOManager (<http://www.seanliming.com/WEPOS.html>) to create a logical name, MGTK_MSR, for the MSR.

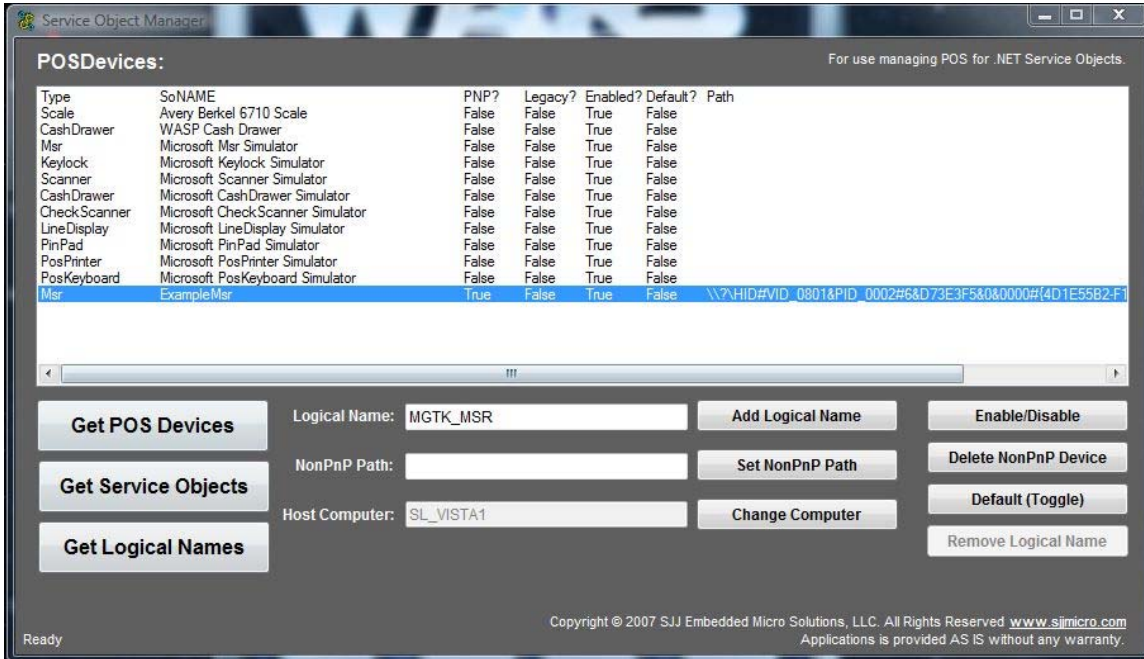


Figure 5 - Logical Name Setup for the Example Service Object

1.2.2 Magtek Service Object Setup

The Magtek service object is available from the Magtek website. Their service object has the PnP information already build into the assembly file.

1. Make sure POS for .NET SDK has been installed on the eo a7330D.
2. Download the service object from the Magtek website.
3. Run the installer to extract the contents of the download. The files will be placed in the "C:\Program Files\MagTek\MagTek UPOS SO" folder.
4. Per the readme file instructions, copy the MagTekServiceObject.dll file to the "C:\Program Files\Common Files\microsoft shared\Point Of Service\Control Assemblies" folder.
5. Use POSDM or SOManager (<http://www.seanliming.com/WEPOS.html>) to create a logical name, MTK_MSR, for the MSR.

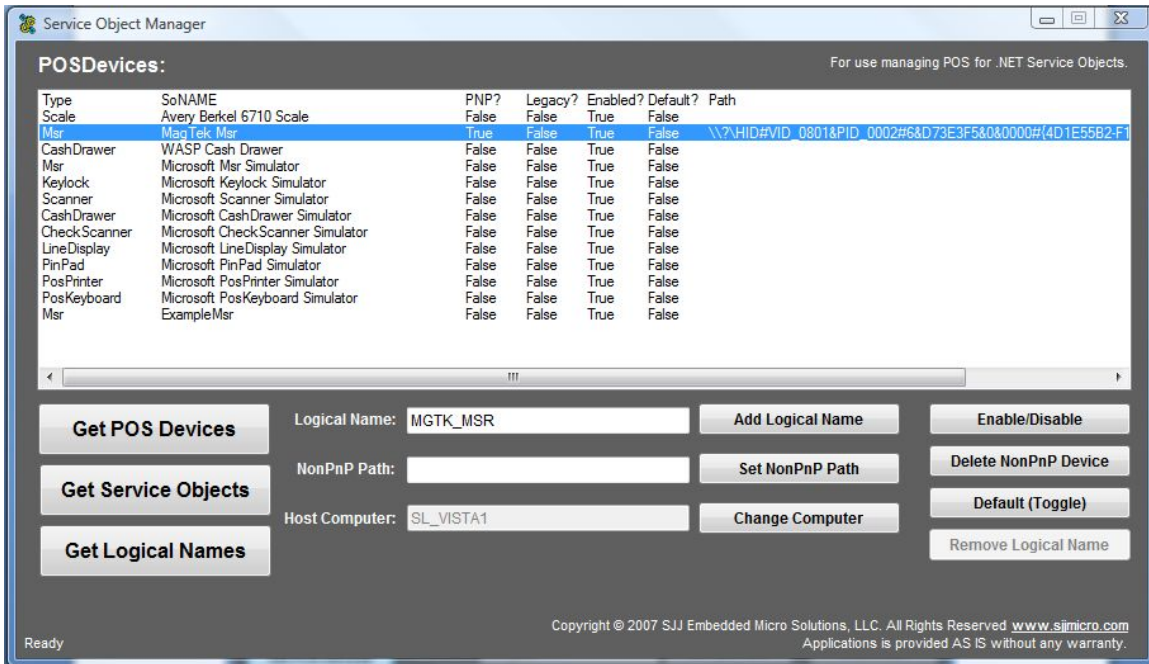


Figure 6 - Logical Name Setup for the Magtek Service Object

1.3 Create an MSR Application in VB.NET

There can be multiple tracks of data on a magnetic strip card. The MSR application will read data from 3 tracks and display all 3 tracks in three text boxes.

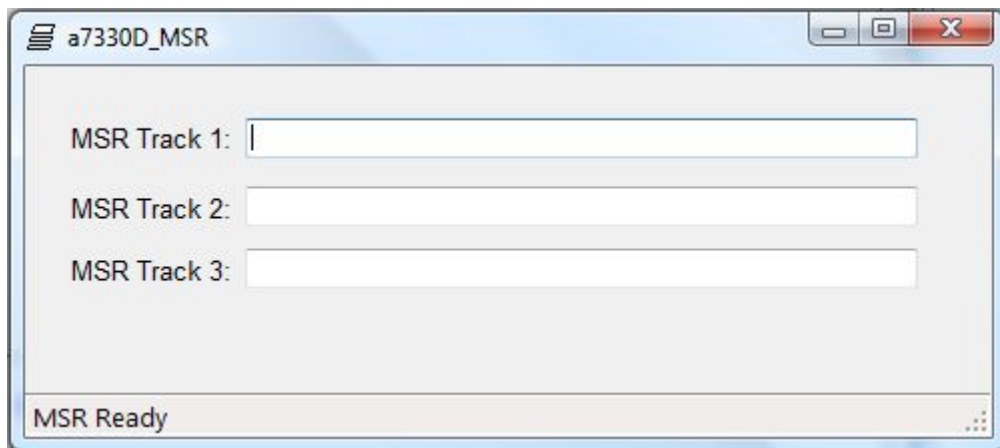


Figure 7 - What the MSR application will look like

1.3.1 Create the Application

You can develop the application on the eo a7330D, if Visual Studio is installed; or you can develop the application on a separate development workstation and copy the application over.

1. Open Visual Studio 2005 or higher (screen shots here are for Visual Studio 2008).
2. From the menu select File->New->Project. The New Project dialog appears.

3. We want to create a new Visual Basic Application. From the Project Types select Visual Basic->Windows.
4. From the Templates select Windows Application.
5. Name the project a7330D_MSR.

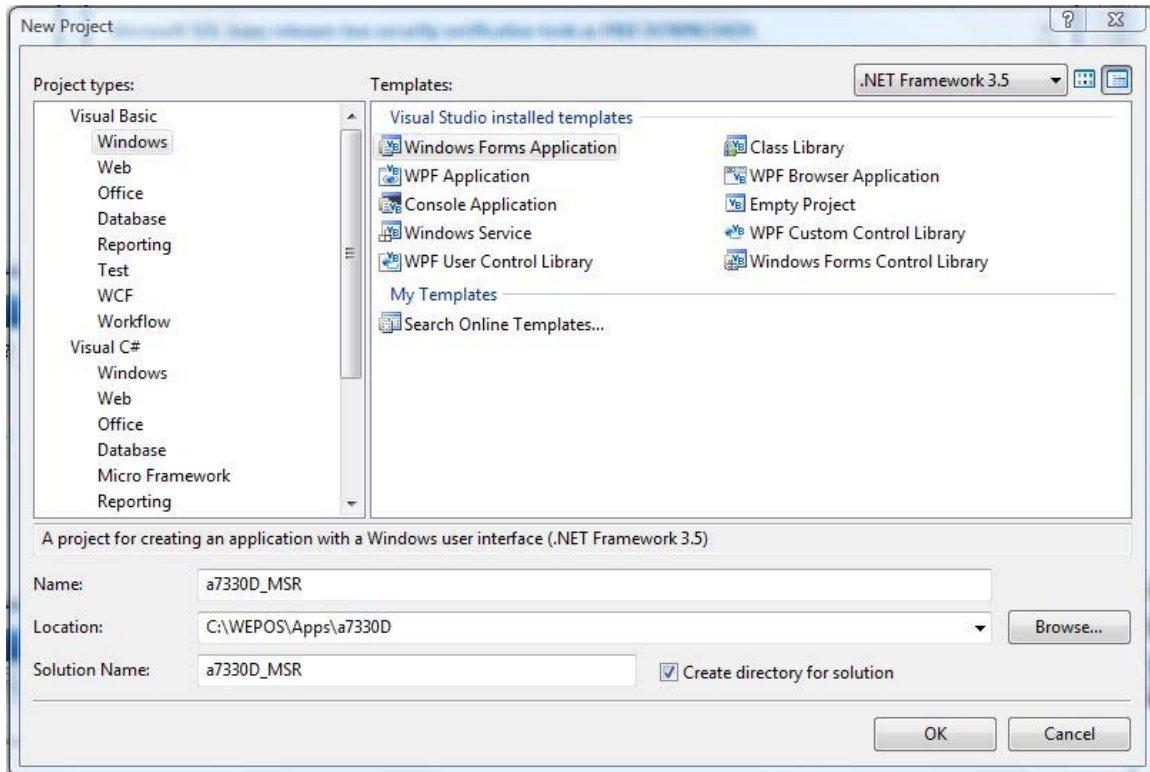


Figure 8 - Creating the VB.NET Application

6. Click OK.
7. Adjust the size of Form1 to accommodate long numbers.
8. On the form, add the following controls and properties:

TextBox1:
Name: txtTrack1
Text:

TextBox2:
Name: txtTrack2
Text:

TextBox3:
Name: txtTrack3
Text:

Label1
Name: lblMSRTrack1
Text: MSR Track1
Font: Arial, 12pt

```
Label2
  Name: lblMSRTrack2
  Text: MSR Track2:
  Font: Arial, 12pt
```

```
Label3
  Name: lblMSRTrack3
  Text: MSR Track3:
  Font: Arial, 12pt
```

```
StatusStrip
  Name: lblStatus
  Text: Ready
  Font: Arial, 12pt
```

9. Save the project.

1.3.2 Adding the POS for .NET Libraries and Code

In this section, we will add the Microsoft.PointOfService.dll reference and the setup code for the MSR.

1. From the menu, select Project->Add Reference. This will open the Add Reference dialog.
2. Click on the Browse tab, and locate the Microsoft.PointOfService.dll found under "C:\Program Files\Microsoft point of Service\SDK".
3. Click on the OK button.
4. Open Form1.VB in code view.
5. At the top of the code before the Form1 class, add the imports:

```
Imports Microsoft.PointOfService
Imports System.Text
Imports System.Threading
```

6. After the Public Class Form1, add the following to trigger events:

```
WithEvents myExplorer As PosExplorer
WithEvents myMSR As Msr
```

7. Using the Method drop-down, add the "New" method to the Form1 class.
8. In the Sub New method add the following code after the InitializeComponent() call:

```
myExplorer = New PosExplorer(Me)
Dim MsrDevice As DeviceInfo = myExplorer.GetDevice("Msr", "MGTK_MSR")
```

```
Try
  myMSR = myExplorer.CreateInstance(MsrDevice)
  myMSR.Open()
  myMSR.Claim(1000)
  myMSR.DeviceEnabled = True
  myMSR.DataEventEnabled = True
  myMSR.DecodeData = True
  lblStatus.Text = "MSR Ready"
Catch ex As Exception
  lblStatus.Text = "MSR Not Found"
End Try
Update()
```

9. Save the project.

The code gets an instance of PosExplorer, the engine that interacts with the POS devices. The next step is to get an instance of the MSR expansion module. A logical name is used to get the device information. We have to use the logical name, since the POS for .NET SDK includes an MSR simulator service object. If we removed this simulator service object, we could have opened the MSR expansion module without having to use a logical name. The Try-Catch is implemented if the MSR expansion module fails for any reason. Status will indicate that the MSR was not set up.

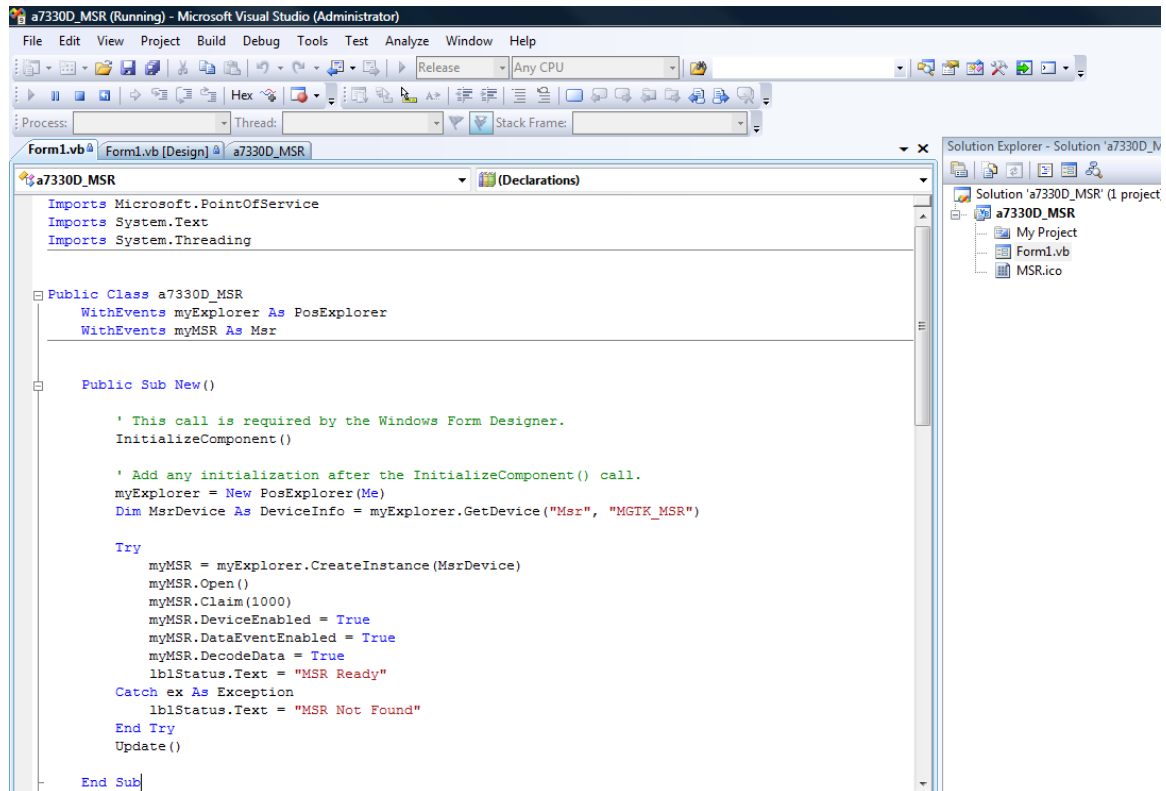


Figure 9 - Basic POS for .NET Application Setup

Now let's setup the data event for the MSR.

10. From the Class Name drop-down, select **myMSR**.
11. From the Method Name drop-down, select **DataEvent**. A new Sub myMSR_DataEvent will be added to the code.
12. Add the following code to the myMSR_DataEvent subroutine

```
Dim myEncoding As New ASCIIEncoding

Try
    txtTrack1.Text = myEncoding.GetString(myMSR.Track1Data)
    txtTrack2.Text = myEncoding.GetString(myMSR.Track2Data)
    txtTrack3.Text = myEncoding.GetString(myMSR.Track3Data)
    lblStatus.Text = "Data Event"
Catch ex As Exception
    lblStatus.Text = "Data Event Failed"
End Try
```

```

Update()

myMSR.DataEventEnabled = True
Thread.Sleep(1000)
lblStatus.Text = "Ready"
Update()

```

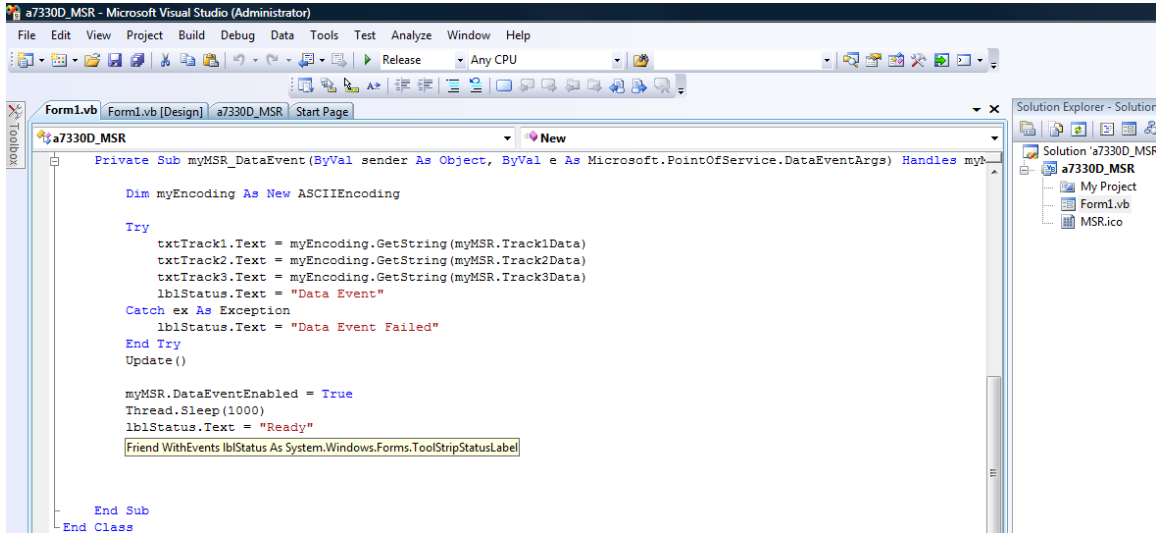


Figure 10 - DataEvent for the MSR

When a magnetic strip card is slid through the reader, a data event is triggered by service object and PosExplorer. The data is stored in the service object waiting to be retrieved by the application. ASCII Encoding is used to decode the data. Each track of data is read independently and displayed in the specific text box. The MSR base class disables further events just in case of a double read. The last steps are to re-enable data events so other scans can be made, and set the status to "Ready".

13. Save the project.

1.3.3 Build and Test

1. Build the application.
2. If you are building the application on the a7330D then start the application by hitting F5 or selecting Debug->Start Debugging from the menu. Otherwise build the application on the development PC by hitting F5 in Visual Studio, and when it has successfully built without error, copy the a7330D_MSR.EXE to the a7330D platform.
3. The status label should display that the MSR is attached. Swipe a card with a magnetic stripe. The data should be shown in the text box.

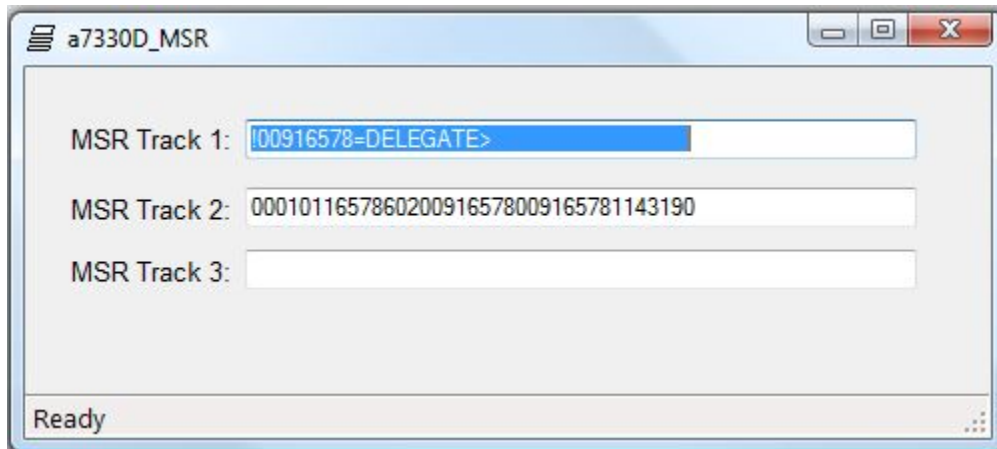


Figure 11 - Testing the MSR

1.4 C# Implementation

The same application can be written in C#. Below is the C# code implementation for the same application. The only trick here is setting up the data event. Unlike VB.NET, C# requires the data event to be setup via code. If you type in the following in Visual Studio, and hit tab key twice:

```
myMSR.DataEvent += <tab> <tab>
```

IntelliSense will generate the myMSR_DataEvent event handler.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.PointOfService;
using System.Threading;

namespace a7330D_MSR_CS
{
    public partial class Form1 : Form
    {

        private PosExplorer myExplorer;
        private Msr myMSR;

        public Form1()
        {
            InitializeComponent();
            myExplorer = new PosExplorer(this);

            DeviceInfo device = myExplorer.GetDevice("Msr", "MGTK_MSR");
```

```
        try
        {
            myMSR = (Msr)myExplorer.CreateInstance(device);
            myMSR.Open();
            myMSR.Claim(1000);
            myMSR.DeviceEnabled = true;
            myMSR.DataEventEnabled = true;
            myMSR.DecodeData = true;
            lblStatus.Text = "MSR Ready";
        }
        catch
        {
            lblStatus.Text = "MSR Not Found!";
        }
        Update();
        myMSR.DataEvent += new DataEventHandler(myMSR_DataEvent);
    }

    void myMSR_DataEvent(object sender, DataEventArgs e)
    {
        ASCIIEncoding myEncoding = new ASCIIEncoding();

        try
        {
            txtMSRTrack1.Text = myEncoding.GetString(myMSR.Track1Data);
            txtMSRTrack2.Text = myEncoding.GetString(myMSR.Track2Data);
            txtMSRTrack3.Text = myEncoding.GetString(myMSR.Track3Data);
            lblStatus.Text = "Data Event";
        }
        catch
        {
            lblStatus.Text = "Data Event Failed";
        }

        Update();
        myMSR.DataEventEnabled = true;
        Thread.Sleep(100);
        lblStatus.Text = "Ready";
        Update();
    }
}
}
```

1.5 Summary

The TabletKiosk® eo a7330D Ultra-Mobile PC with the additional magnetic strip reader is an ideal platform for a mobile POS device. Writing POS application for the a7330D is simple with the POS for .NET SDK integration into Visual Studio. You can easily develop POS applications in VB.NET or C#.

1.6 Bibliography

The following book is the primary reference for this paper:

Windows® Embedded for Point of Service / POS for .NET Step-By-Step, Sean D. Liming and John R. Malin, Cedar Hill Publishing, 2006. ISBN-13: 978-1-933324-54-8

*Windows and Visual Studio are registered trademark of Microsoft Corporation.
All other registered trademarks are owned by their respective companies.*