What is new for POS for .NET 1.14

By Sean D. Liming & John R. Malin
Annabooks

Published in the United States by

Annabooks, LLC
6432 Glendale Dr.
Yorba Linda, CA 92886 USA

www.annabooks.com

Table of Contents

1 Finally! POS for .NET 1.14

The long overdue update to POS for .NET SDK has finally arrived. POS for .NET v1.14 release was a surprise since Microsoft never announced that they were working on an update. Unfortunately, the beta cycle was very short giving very little time to do significant testing, but the SDK delivers the basic elements that developers have been asking for:

- Support for .NET Framework 4.x and 64-bit applications. The .NET Framework 2.0 requirement is no longer needed.
- Installation consistent with POS for .NET 1.12. The SDK location doesn't change.
- Aligned with UnifiedPOS v1.14.
- Builds on the OPOS service object support with 8 new devices.

Best of all, the development process has not changed. The development processes discussed in the book, *Professional's Guide to POS for .NET,* are still the same. This paper services as an addendum to the book to cover POS for .NET v1.14 SDK.

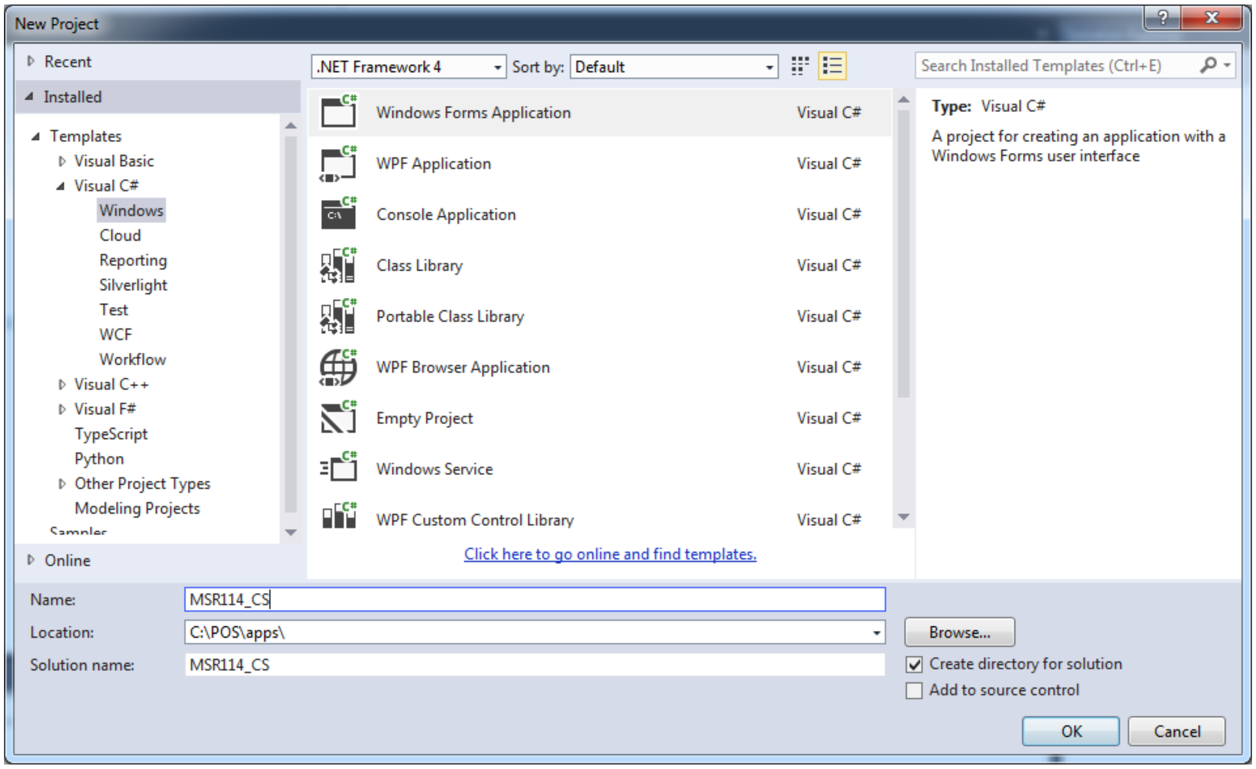
2 Creating an Application

Although the process to create an application using the new SDK is the same, you need to use Visual Studio 2012 or higher. Visual Studio 2013 will be used here to create a simple application that uses an MSR (Magnetic Strip Reader). A Magtek MSR will be the MSR POS device. The MagTek's POS for .NET 1.12 service object will be used for testing the application. In the future, the 1.14 service object should be used for a real product when available, but until then MagTek's 1.12 version will have to do. The first steps are to make sure that the Service Objects or OPOS drivers are installed and successfully work with the SDK's TestApp.exe. Once the Service Object or OPOS driver is in place, you can create the application.

2.1 Part 1 – Create the Application

First step is to create the project.

1. Open Visual Studio 2013.
2. From the menu, select File->New->Project. The New Project dialog appears.
3. From the templates, select Visual C#->Windows.
4. The click on Windows Form Application.
5. Name the project MSR114_CS.
6. Click OK.


New Project

Name: MSR114_CS
Location: C:\POS\apps\
Solution name: MSR114_CS
☑ Create directory for solution
☐ Add to source control

Type: Visual C#
A project for creating an application with a Windows Forms user interface

Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
TypeScript
Python
Other Project Types
Modeling Projects
Online
.NET Framework 4
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

7. Adjust the size of Form1 to accommodate long numbers.
8. On the form, add a TextBox control and two Labels with the following properties:

   TextBox:
   Name: txtInput

   Label1:
   Name: lblMSRData
   Text: MSR data:
   Font: Arial, Regular, 12pt

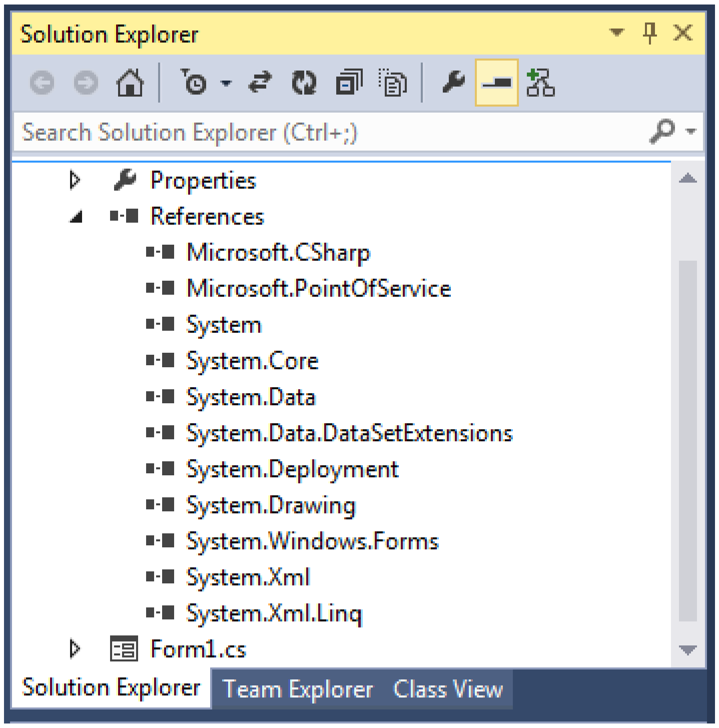
   StatusStrip->Label:
   Name: TTStatus
   Text: Ready
   Font: Arial, Regular, 12pt

9. Save the project.

2.2 Part 2: Adding the POS for .NET Libraries and Code

Next, we add the Microsoft.PointOfService.dll and the 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(x86)\Microsoft point of Service\SDK.
3. Click on the OK button.



Properties
References
Microsoft.CSharp
Microsoft.PointOfService
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Deployment
System.Drawing
System.Windows.Forms
System.Xml
System.Xml.Linq
Form1.cs

4. Open Form1.CS in code view.
5. At the top of the code before the From1 class, add the imports:

```
using Microsoft.PointOfService;
```

6. After the Public Partial Class Form1, add the following:

```
private PosExplorer myExplorer;
private Msr myMsr;
```

1. In the Form1() method, add the following code after the InitializeComponent() call:

```
myExplorer = new PosExplorer(this);
myExplorer.DeviceAddedEvent += new DeviceChangedEventHandler(myExplorer_DeviceAddedEvent);
myExplorer.DeviceRemovedEvent += new DeviceChangedEventHandler(myExplorer_DeviceRemovedEvent);

DeviceInfo device = myExplorer.GetDevice ("Msr");

if (device == null)
{
        TTStatus.Text = "Msr Not Found";
}
else
{
        myMsr = (Msr)myExplorer.CreateInstance(device);
        myMsr.Open();
        myMsr.Claim(1000);
        myMsr.DataEvent += new DataEventHandler(myMsr_DataEvent);
        myMsr.DeviceEnabled = true;
        myMsr.DataEventEnabled = true;
        myMsr.DecodeData = true;
        TTStatus.Text = "Found Msr - Ready";
}
```

Notice that we are getting the Msr device by default and not using a logical name. Therefore, we will have to remove any other Msr service objects that may be in the system.

After you enter += for each event, hit Tab twice so Visual Studio can automatically generate the event callback.

2. Save the project.
3. Add the following code to the myExplorer_DeviceAddedEvent subroutine:

```
if (e.Device.Type == "Msr")
{
        myMsr = (Msr)myExplorer.CreateInstance(e.Device);
        TTStatus.Text = "Found Msr - Ready";
        myMsr.Open();
        myMsr.Claim(1000);
        myMsr.DataEvent += new DataEventHandler(myMsr_DataEvent);
        myMsr.DeviceEnabled = true;
        myMsr.DataEventEnabled = true;
        myMsr.DecodeData = true;
}
```

4. Save the project.
5. Add the following code to the myExplorer_DeviceRemovedEvent:

```
if (e.Device.Type == "Msr")
{
        myMsr.DataEventEnabled = false;
        myMsr.DeviceEnabled = false;
        myMsr.Release();
        myMsr.Close();
        TTStatus.Text = "Found Msr - removed";
}
```

6. Save the project
7. Add the following code to the myMSR_DataEvent subroutine

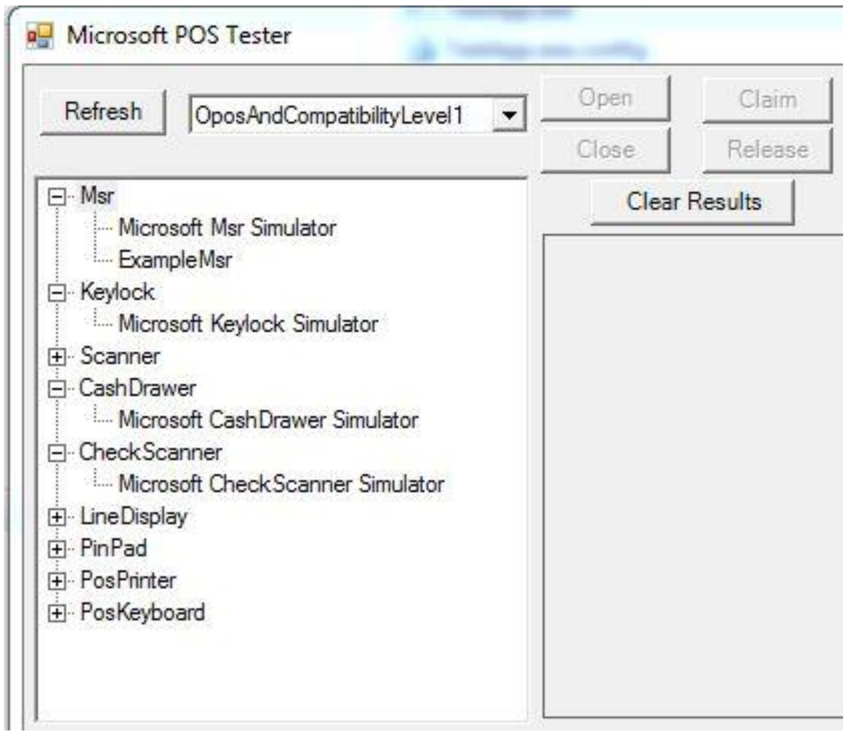
```
ASCIIEncoding myEncoding = new ASCIIEncoding();
txtInput.Text = myEncoding.GetString(myMsr.Track2Data);
myMsr.DataEventEnabled = true;
```

The last step is to re-enable data events so other Msr swipes can be made.

8. Save the project.

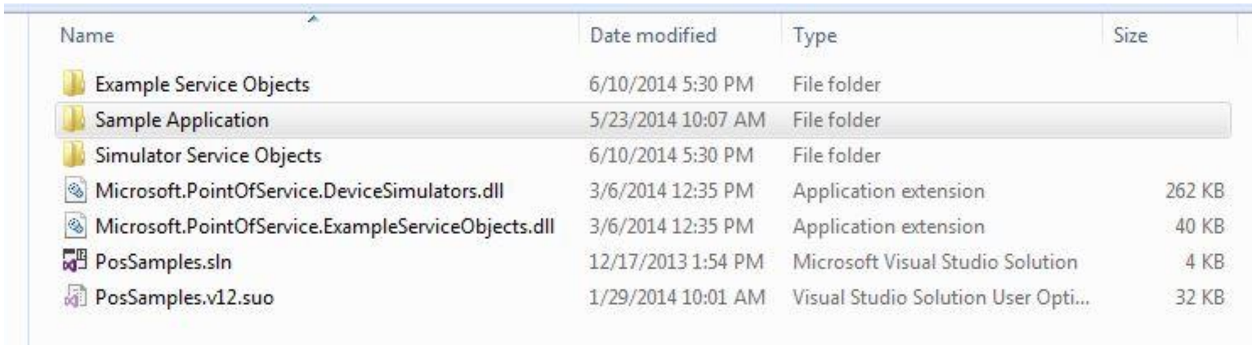
2.3 Part 3: Build and Test

The project can now be tested, but first we need to isolate the MSR being used. The first steps are to move the example service object and the simulator service object to another location. We can then test the application.



- Msr
  - Microsoft Msr Simulator
  - ExampleMsr
- Keylock
  - Microsoft Keylock Simulator
- Scanner
- CashDrawer
  - Microsoft CashDrawer Simulator
- CheckScanner
  - Microsoft CheckScanner Simulator
- LineDisplay
- PinPad
- PosPrinter
- PosKeyboard

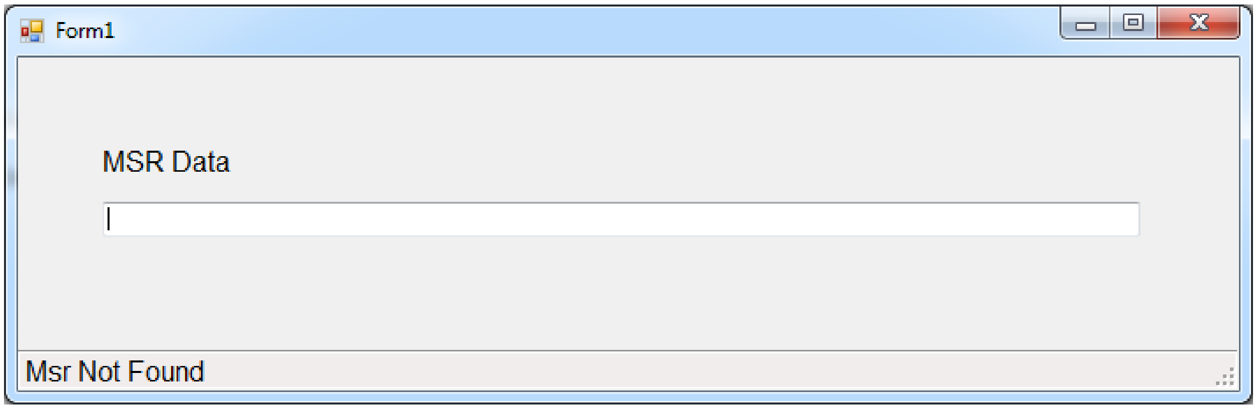
1. Open File Explorer.
2. Go to the C:\Program Files (x86)\Microsoft Point Of Service\SDK\Samples\Simulator Service Objects directory.
3. Cut and paste Microsoft.PointOfService.DeviceSimulators.dll to C:\Program Files (x86)\Microsoft Point Of Service\SDK\Samples
4. If you are using the Example service Object from the SDK, skip to step 6. Go to C:\Program Files (x86)\Microsoft Point Of Service\SDK\Samples\Example Service Objects.
5. Cut and paste Microsoft.PointOfService.ExampleServiceObjects.dll to C:\Program Files (x86)\Microsoft Point Of Service\SDK\Samples.



Name	Date modified	Type	Size
Example Service Objects	6/10/2014 5:30 PM	File folder	
Sample Application	5/23/2014 10:07 AM	File folder	
Simulator Service Objects	6/10/2014 5:30 PM	File folder	
Microsoft.PointOfService.DeviceSimulators.dll	3/6/2014 12:35 PM	Application extension	262 KB
Microsoft.PointOfService.ExampleServiceObjects.dll	3/6/2014 12:35 PM	Application extension	40 KB
PosSamples.sln	12/17/2013 1:54 PM	Microsoft Visual Studio Solution	4 KB
PosSamples.v12.suo	1/29/2014 10:01 AM	Visual Studio Solution User Opti...	32 KB

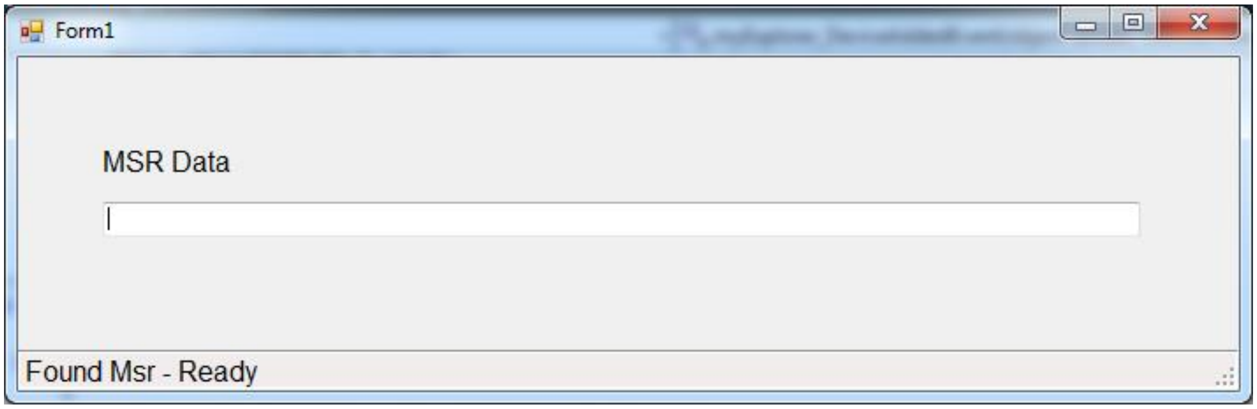
The above steps remove the extra virtual Msr’s that are not needed for the application. The program will select the first Msr as the default, and we want it to be the actual physical Msr, rather than the simulator service object.

6. Make sure the MSR is not attached to the system.
7. In Visual Studio, run the application.



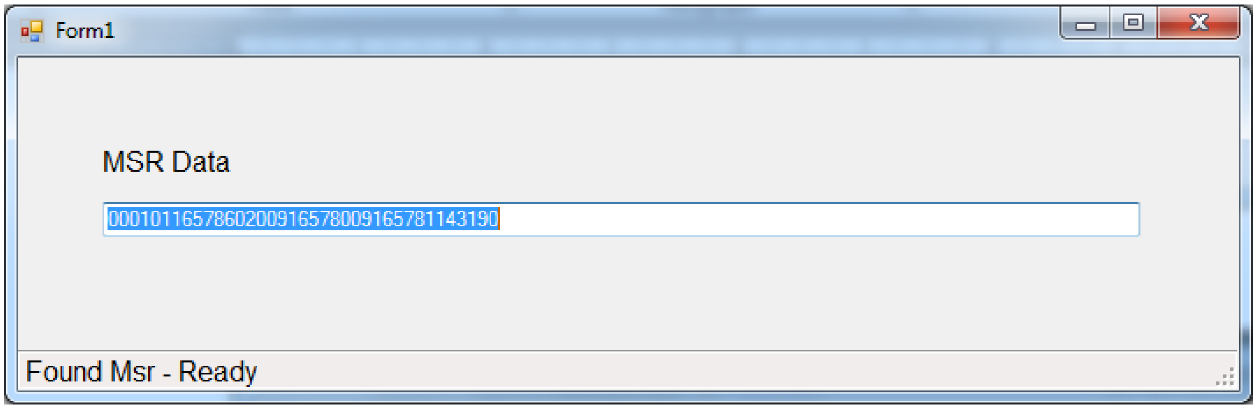
Msr Not Found

8. Connect the MSR.



Found Msr - Ready

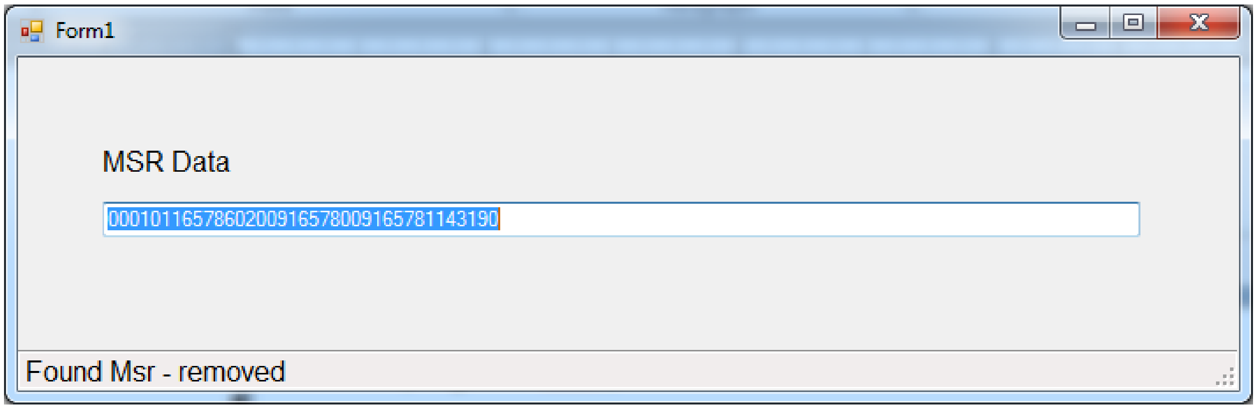
9. Swipe a card to read the data.



0001011657860200916578009165781143190

Found Msr - Ready

10. Unplug the MSR.



MSR Data

0001011657860200916578009165781143190

Found Msr - removed

11. Close the application.

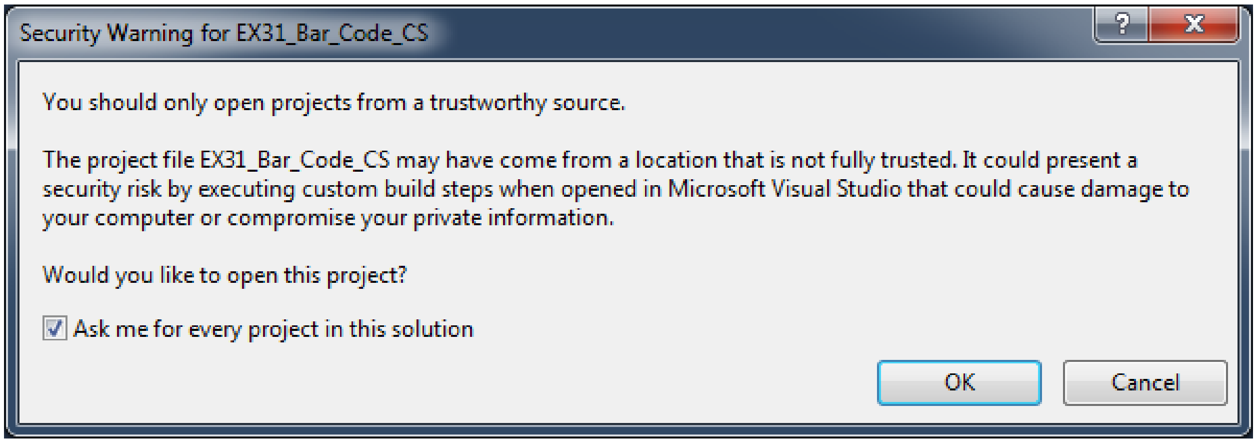
3 Upgrade Applications to .NET Framework 4.x

For those of you who have built applications with an older Visual Studio version, updating the POS for .NET portion of the project requires a few changes. These steps are only for POS for .NET. If you are using other libraries in your project, you might have to upgrade those as well.

3.1 Updating A C# Application

Using an example from the book, EX31_Bar_Code_CS, here are the steps to update a POS for .NET v1.12 project to POS for .NET v1.14.

1. Download and extract the Book Exercises for *Professional's Guide to POS for .NET* *http://www.annabooks.com/Book_PGPOS.html*.
2. Open Visual Studio 2013.
3. From the menu Select File->Open->Project/Solution.
4. Locate and open EX31_bar_Code_CS.
5. A dialog appears about trustworthy source, click OK to continue.


Security Warning for EX31_Bar_Code_CS

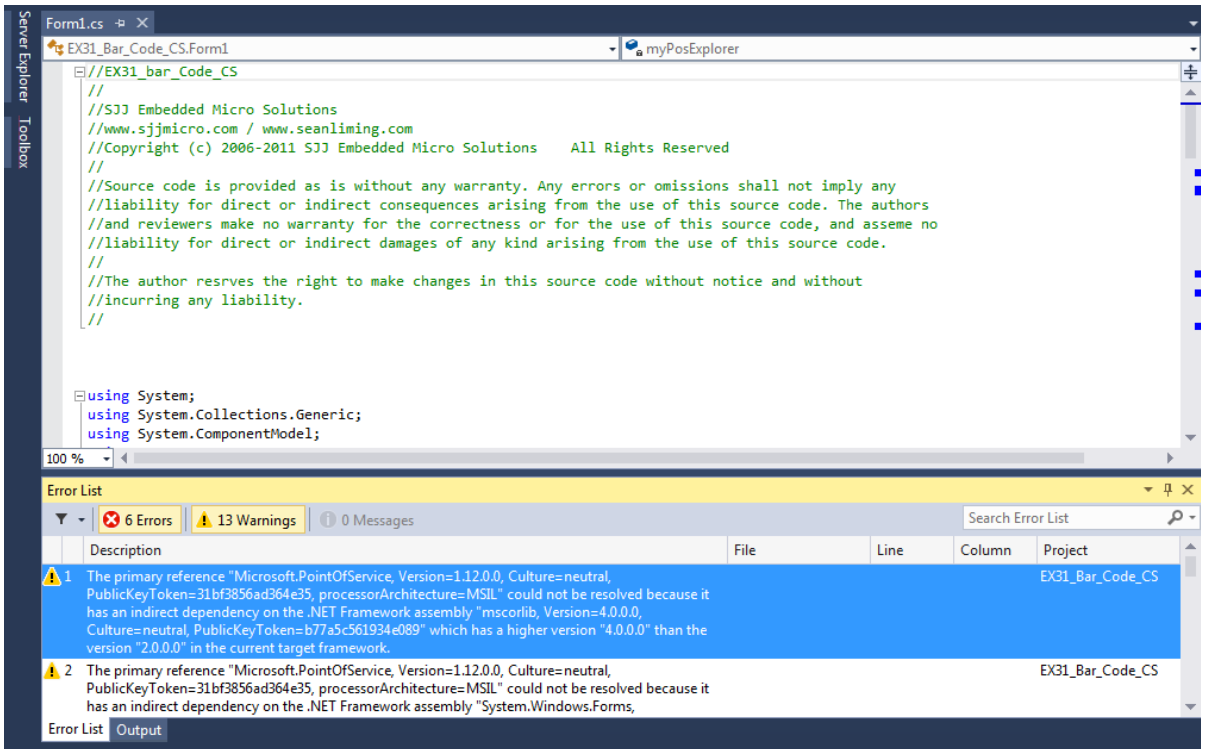
You should only open projects from a trustworthy source.

The project file EX31_Bar_Code_CS may have come from a location that is not fully trusted. It could present a security risk by executing custom build steps when opened in Microsoft Visual Studio that could cause damage to your computer or compromise your private information.

Would you like to open this project?

☑ Ask me for every project in this solution

6. Visual Studio 2013 will open the C# project without errors. Try building the application and you will see errors referencing the old assembly.



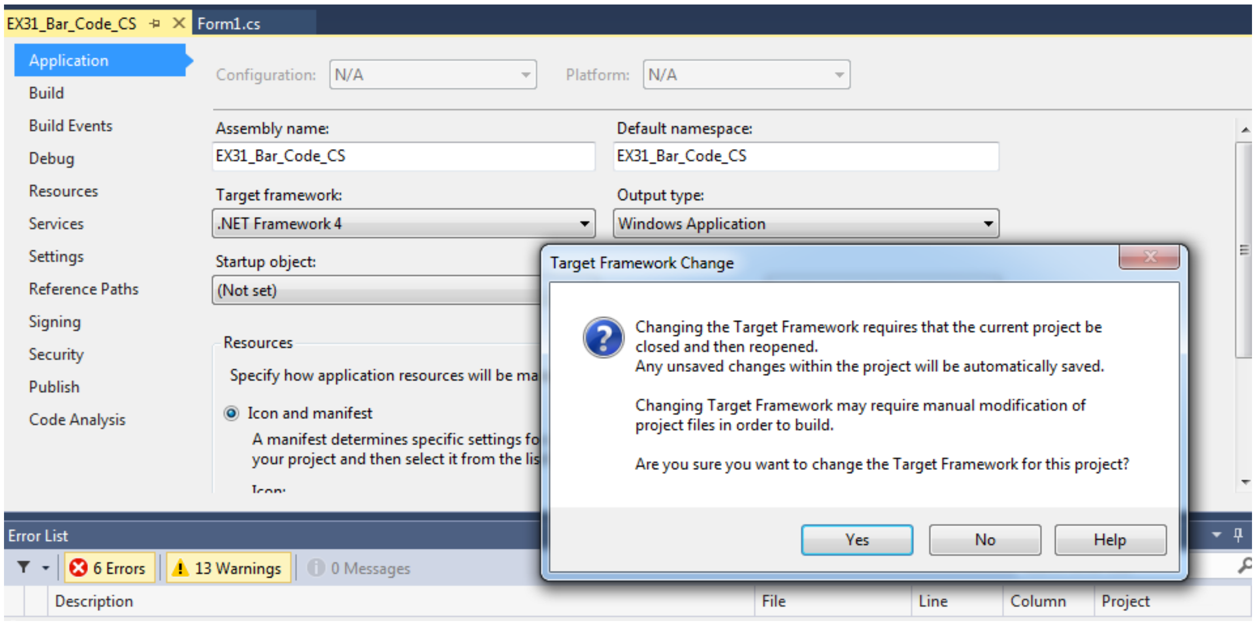
```
//EX31_bar_Code_CS
//
//SJJ Embedded Micro Solutions
//www.sjjmicro.com / www.seanliming.com
//Copyright (c) 2006-2011 SJJ Embedded Micro Solutions    All Rights Reserved
//
//Source code is provided as is without any warranty. Any errors or omissions shall not imply any
//liability for direct or indirect consequences arising from the use of this source code. The authors
//and reviewers make no warranty for the correctness or for the use of this source code, and asseme no
//liability for direct or indirect damages of any kind arising from the use of this source code.
//
//The author resrves the right to make changes in this source code without notice and without
//incurring any liability.
//


using System;
using System.Collections.Generic;
using System.ComponentModel;
```

Error List: 6 Errors, 13 Warnings, 0 Messages

	Description	File	Line	Column	Project
1	The primary reference "Microsoft.PointOfService, Version=1.12.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL" could not be resolved because it has an indirect dependency on the .NET Framework assembly "mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" which has a higher version "4.0.0.0" than the version "2.0.0.0" in the current target framework.				EX31_Bar_Code_CS
2	The primary reference "Microsoft.PointOfService, Version=1.12.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL" could not be resolved because it has an indirect dependency on the .NET Framework assembly "System.Windows.Forms,				EX31_Bar_Code_CS

7. From the menu, select Project->EX31_Bar_Code_CS Properties.
8. Change the Target Framework from 3.5 to 4.0.
9. A dialog will appear asking if you really want to do this, click Yes. This is a simple application. If there were any APIs specific to .NET Framework 3.5, you will have to use the new ones for 4.0.



Assembly name: EX31_Bar_Code_CS
Default namespace: EX31_Bar_Code_CS
Target framework: .NET Framework 4
Output type: Windows Application
Startup object: (Not set)

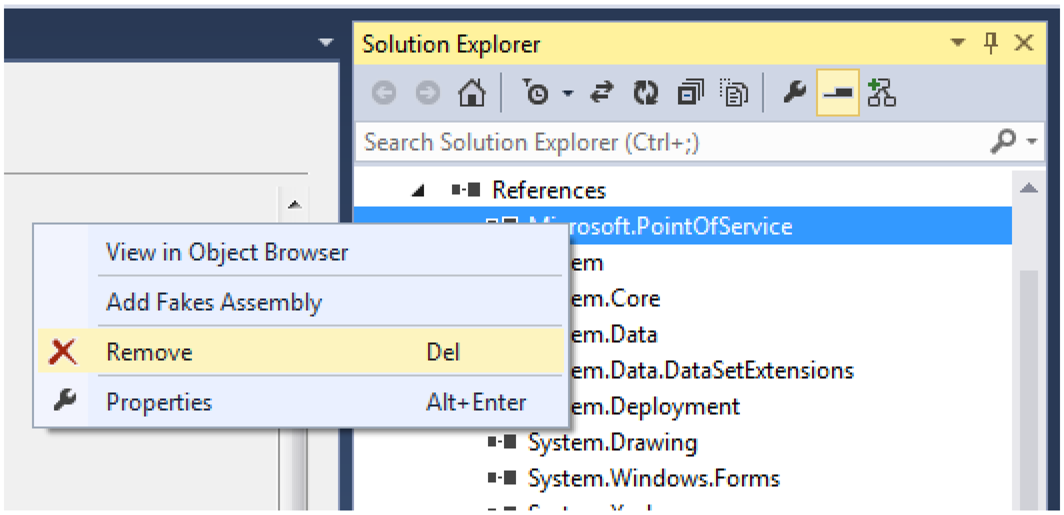
Target Framework Change

Changing the Target Framework requires that the current project be closed and then reopened.
Any unsaved changes within the project will be automatically saved.

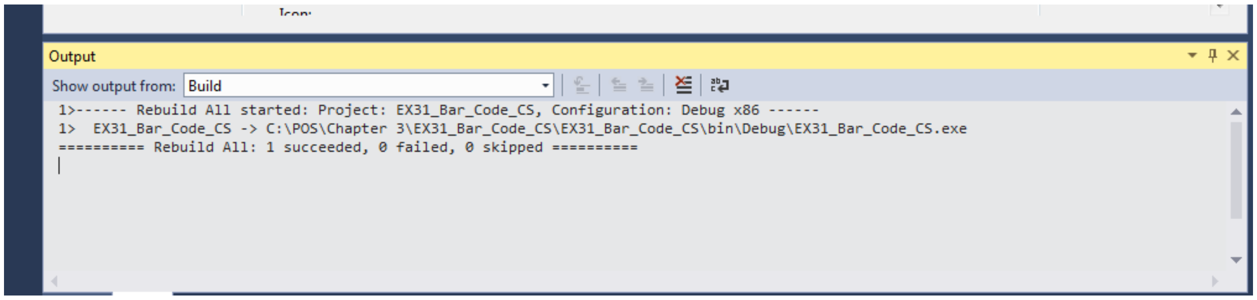
Changing Target Framework may require manual modification of project files in order to build.

Are you sure you want to change the Target Framework for this project?

10. In Solution Explorer, delete the reference for the old POS for .NET 1.12.



11. From the menu, select Project->Add Reference. This will open the Add Reference dialog.
12. Click on the Browse tab, and locate the Microsoft.PointOfService.dll found under c:\Program Files(x86)\Microsoft point of Service\SDK.
13. Click on the OK button.
14. Save the project
15. Try rebuilding the project, and this time it should succeed.

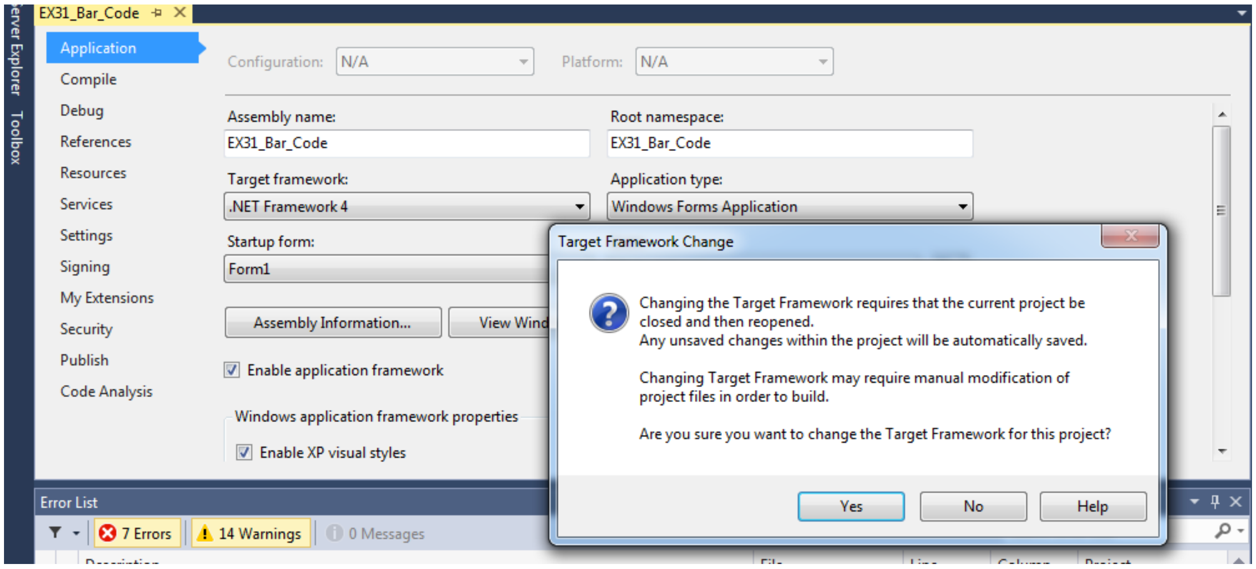


```
1>------ Rebuild All started: Project: EX31_Bar_Code_CS, Configuration: Debug x86 ------
1>  EX31_Bar_Code_CS -> C:\POS\Chapter 3\EX31_Bar_Code_CS\EX31_Bar_Code_CS\bin\Debug\EX31_Bar_Code_CS.exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

3.2 Updating A VB.NET Application

VB.NET conversion is a little different from C#. We will use the VB.NET version of the application above: EX31_Bar_Code.

1. Open Visual Studio 2013.
2. From the menu Select File->Open->Project/Solution.
3. Locate and open EX31_bar_Code.
4. If asked about a trusted source, click Ok.
5. If you run the application, you will get an assembly error.
6. From the menu, select Project->EX31_Bar_Code Properties.
7. Change the Target Framework from 3.5 to 4.0.
8. A dialog will appear asking if you really want to do this, click Yes. This is a simple application. If there were any APIs specific to .NET Framework 3.5, you will have to use the new ones for 4.0.



Configuration: N/A Platform: N/A

Assembly name: EX31_Bar_Code
Root namespace: EX31_Bar_Code
Target framework: .NET Framework 4
Application type: Windows Forms Application
Startup form: Form1

☑ Enable application framework
☑ Enable XP visual styles

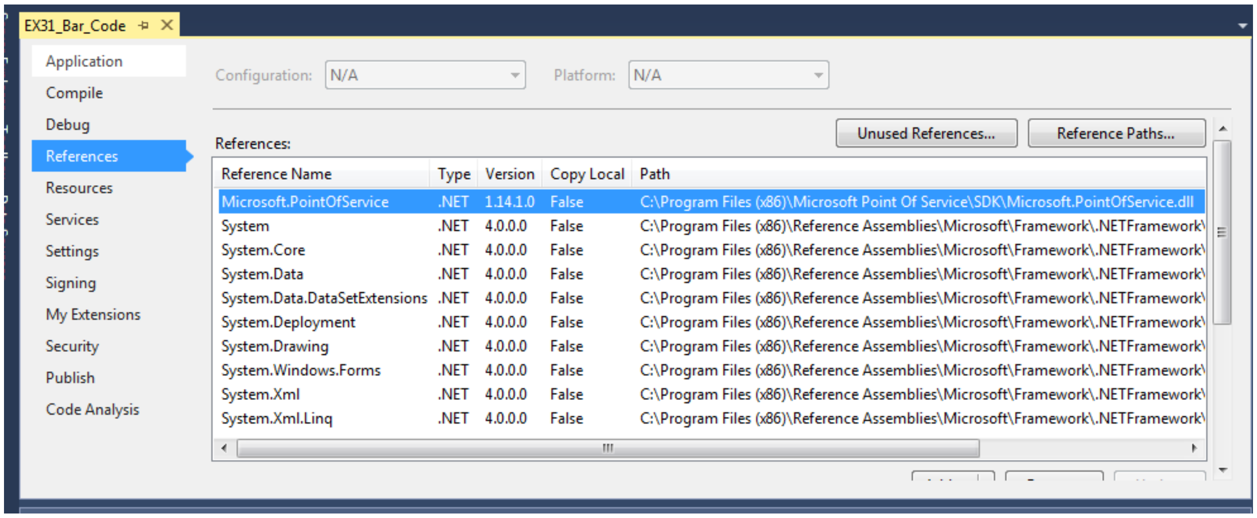
Target Framework Change

Changing the Target Framework requires that the current project be closed and then reopened.
Any unsaved changes within the project will be automatically saved.

Changing Target Framework may require manual modification of project files in order to build.

Are you sure you want to change the Target Framework for this project?

9. Click on the References tab, you will notice that Microsoft.PointOfService 1.14 is already referenced.

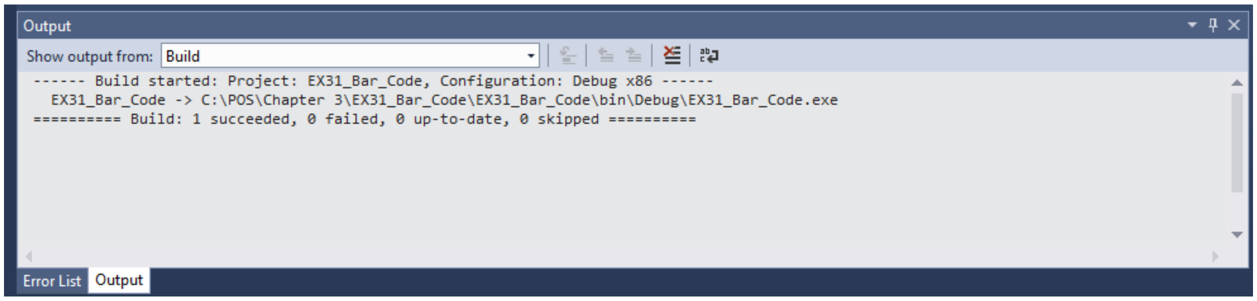


Configuration: N/A Platform: N/A

References:

Reference Name	Type	Version	Copy Local	Path
Microsoft.PointOfService	.NET	1.14.1.0	False	C:\Program Files (x86)\Microsoft Point Of Service\SDK\Microsoft.PointOfService.dll
System	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Core	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Data	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Data.DataSetExtensions	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Deployment	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Drawing	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Windows.Forms	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Xml	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\
System.Xml.Linq	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\

10. Build the application again, and it should build successfully.



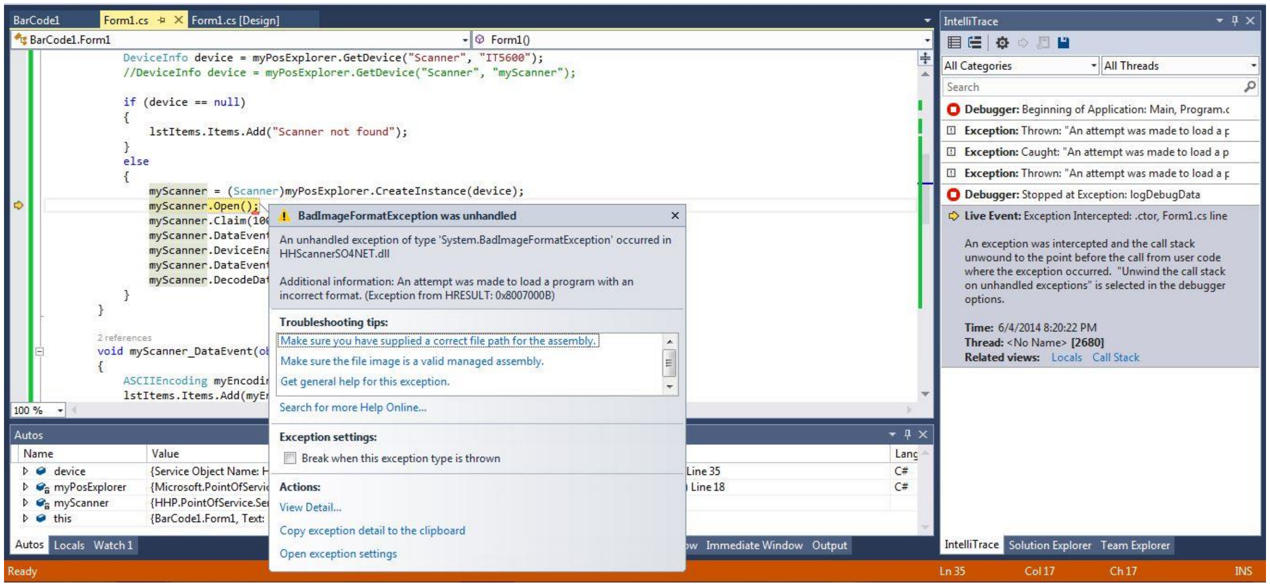
Show output from: Build

```
------ Build started: Project: EX31_Bar_Code, Configuration: Debug x86 ------
  EX31_Bar_Code -> C:\POS\Chapter 3\EX31_Bar_Code\EX31_Bar_Code\bin\Debug\EX31_Bar_Code.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

4 32bit versus 64bit Support

POS for .NET 1.12 only supports 32-bit applications. POS for .NET 1.14 adds 64-bit support, but 64-bit support is a little tricky. All OPOS and Service Objects created with POS for .NET 1.12 are 32-bit. A 64-bit application cannot access a 32-bit OPOS or Service Object.

For example, I recreated the book's EX31_Bar_Code project from scratch using Visual Studio 2013 and POS for .NET 1.14. The bar code scanner is the Honeywell IT5600. The IT5600 POS for .NET 1.12 Service Object was the latest from Honeywell. Building the application was clean, but when I ran the application, a run-time error appeared.



```
            DeviceInfo device = myPosExplorer.GetDevice("Scanner", "IT5600");
            //DeviceInfo device = myPosExplorer.GetDevice("Scanner", "myScanner");

            if (device == null)
            {
                lstItems.Items.Add("Scanner not found");
            }
            else
            {
                myScanner = (Scanner)myPosExplorer.CreateInstance(device);
                myScanner.Open();
                myScanner.Claim(10
                myScanner.DataEvent
                myScanner.DeviceEna
                myScanner.DataEvent
                myScanner.DecodeDat
            }
        }

        void myScanner_DataEvent(ob
        {
            ASCIIEncoding myEncodin
            lstItems.Items.Add(myEn
```

BadImageFormatException was unhandled

An unhandled exception of type 'System.BadImageFormatException' occurred in HHScannerSO4NET.dll

Additional information: An attempt was made to load a program with an incorrect format. (Exception from HRESULT: 0x8007000B)

Troubleshooting tips:

Make sure you have supplied a correct file path for the assembly.
Make sure the file image is a valid managed assembly.
Get general help for this exception.

Search for more Help Online...

Exception settings:

☐ Break when this exception type is thrown

Actions:

View Detail...
Copy exception detail to the clipboard
Open exception settings

Name	Value
device	{Service Object Name: H
myPosExplorer	{Microsoft.PointOfServic
myScanner	{HHP.PointOfService.Se
this	{BarCode1.Form1, Text:

IntelliTrace

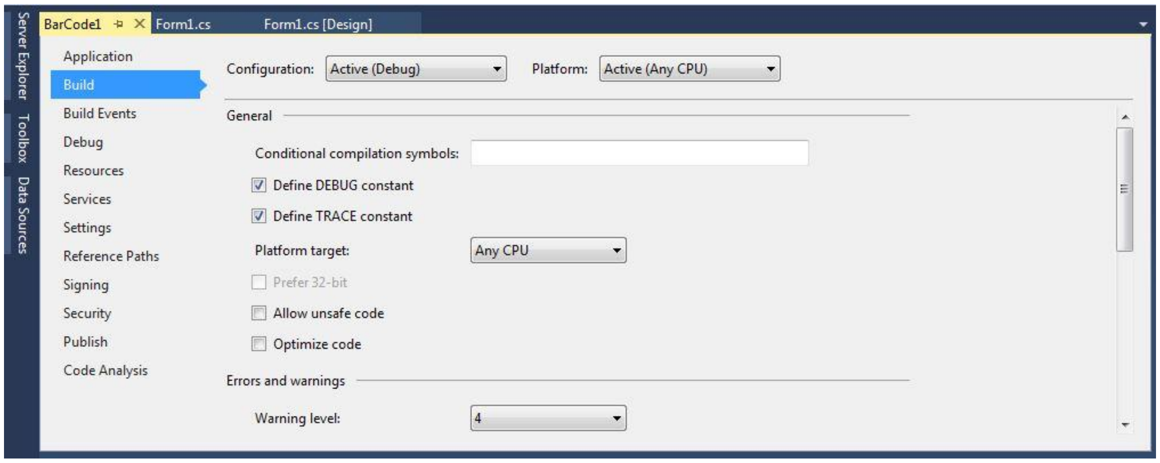
Debugger: Beginning of Application: Main, Program.c
Exception: Thrown: "An attempt was made to load a p
Exception: Caught: "An attempt was made to load a p
Exception: Thrown: "An attempt was made to load a p
Debugger: Stopped at Exception: logDebugData
Live Event: Exception Intercepted: .ctor, Form1.cs line

An exception was intercepted and the call stack unwound to the point before the call from user code where the exception occurred. "Unwind the call stack on unhandled exceptions" is selected in the debugger options.

Time: 6/4/2014 8:20:22 PM
Thread: <No Name> **[2680]**
Related views: Locals Call Stack

Ready Ln 35 Col 17 Ch 17 INS

The problem was that the project was set to run as "Any CPU" by default. On a 64-bit system the application runs in 64-bit mode.



Configuration: Active (Debug) Platform: Active (Any CPU)

General

Conditional compilation symbols:

☑ Define DEBUG constant
☑ Define TRACE constant

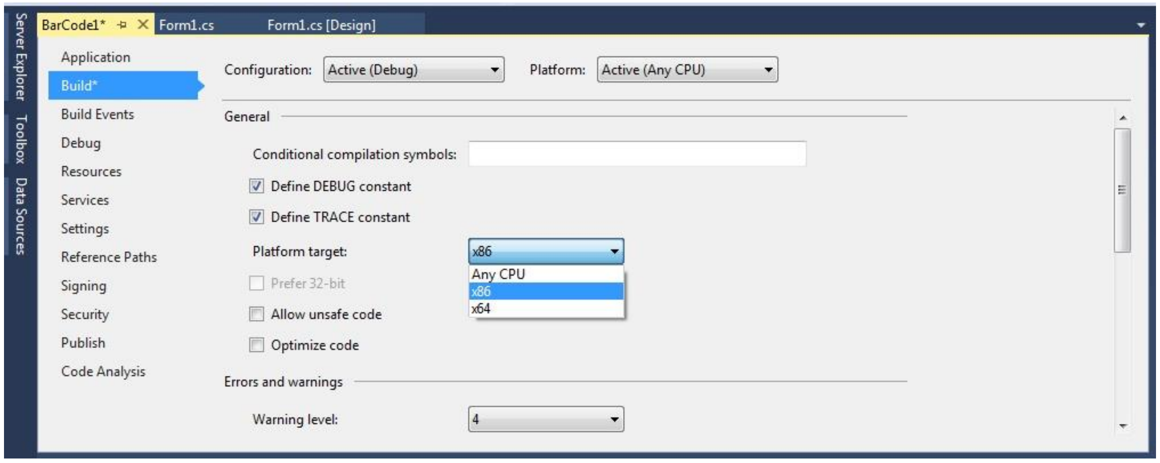
Platform target: Any CPU

☐ Prefer 32-bit
☐ Allow unsafe code
☐ Optimize code

Errors and warnings

Warning level: 4

Since the Service Object is 32-bit, the application should be set to 32-bit.



Configuration: Active (Debug)
Platform: Active (Any CPU)
Conditional compilation symbols:
☑ Define DEBUG constant
☑ Define TRACE constant
Platform target: x86
☐ Prefer 32-bit
☐ Allow unsafe code
☐ Optimize code
Warning level: 4

After rebuilding, the application runs successfully. If you really want a 64-bit application, you're going to be stuck in this case. The POS for .NET online documentation discusses the 32-bit vs. 64-bit issue and how to modify the registry for 32-bit OPOS drivers so that 64-bit applications can interact with a 32-bit OPOS driver via IPC and marshaling. There is nothing obvious or discussed on how to re-register Service Objects in this manner. You will have to wait until the POS device manufacturer releases a 64-bit Service Object or continue with 32-bit development, which brings us to the last topic on Service Objects.

5 Service Objects and More 32-bit versus 64-bit

To explore the 32-bit and 64-bit dynamic a little further and review service object development, I recreated the Avery Berkel 6710 scale service object from the book using POS for .NET 1.14 and Visual Studio 2013. I also updated the ScaleSOTest application to test the service object. The following sections report what was found:

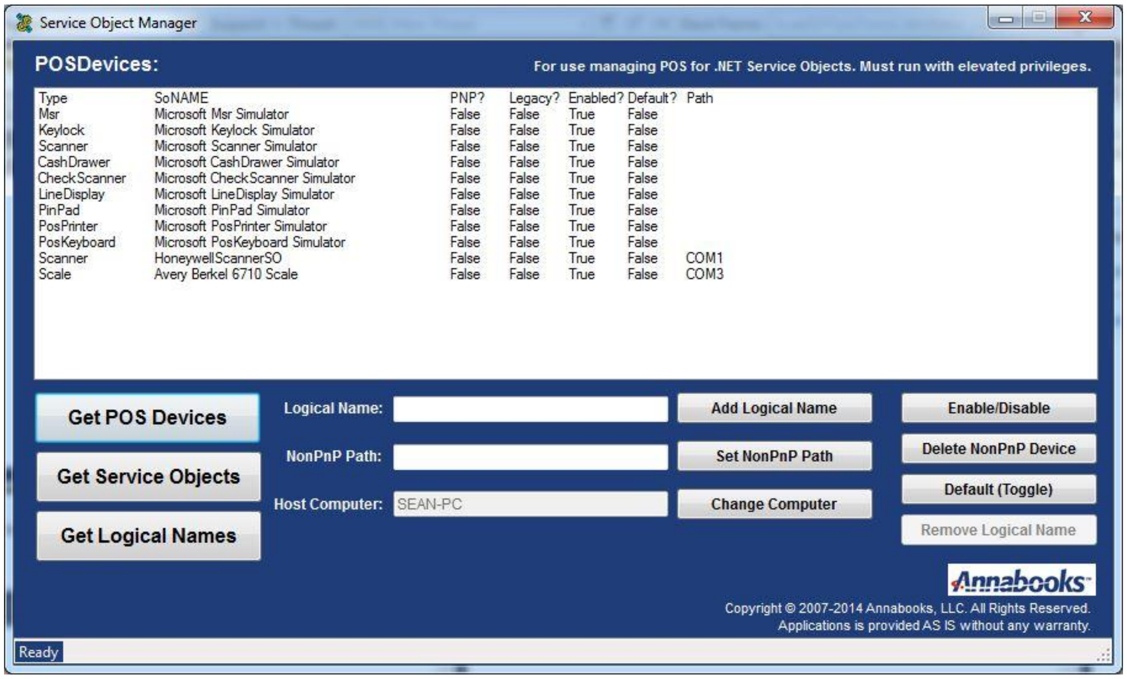
5.1 Information on Creating Service Objects in POS for .NET 1.14

The steps to create service objects have not changed, but there are a few changes in filling out the code. When you enter "Inherits ScaleBasic" class, there are some addition capabilities and features added to meet the changes in the UnifiedPOS 1.14 specification. For the scale service object, there were some capabilities and subroutines added for pricing. Each service object is different. If you are a service object developer, you will have to be aware of these changes. Also, the SecurityAction items in AssembyInfo.vb file are no longer needed and can be removed:

```
<Assembly: PermissionSet(SecurityAction.RequestMinimum, Name:="FullTrust")>
<Assembly: SecurityPermission(SecurityAction.RequestMinimum, Execution:=True,
ControlAppDomain:=True)>
<Assembly: ReflectionPermission(SecurityAction.RequestMinimum)>
```

5.2 Managing Service Objects in POS for .NET 1.14

Managing service objects has not changed. POSDM.EXE and the WMI capabilities operate as before. It was a little concerning that POSDM and the WMI was not available in the beta, but it appears to be working as before. Any custom application used to manage service objects will have to be updated. We are working on an updated SOManager utility.



Service Object Manager

POSDevices:

For use managing POS for .NET Service Objects. Must run with elevated privileges.

```
Type          SoNAME                          PNP?   Legacy?  Enabled?  Default?  Path
Msr           Microsoft Msr Simulator         False  False    True      False
Keylock       Microsoft Keylock Simulator     False  False    True      False
Scanner       Microsoft Scanner Simulator     False  False    True      False
CashDrawer    Microsoft CashDrawer Simulator  False  False    True      False
CheckScanner  Microsoft CheckScanner Simulator False False    True      False
LineDisplay   Microsoft LineDisplay Simulator False  False    True      False
PinPad        Microsoft PinPad Simulator      False  False    True      False
PosPrinter    Microsoft PosPrinter Simulator  False  False    True      False
PosKeyboard   Microsoft PosKeyboard Simulator False  False    True      False
Scanner       HoneywellScannerSO              False  False    True      False     COM1
Scale         Avery Berkel 6710 Scale         False  False    True      False     COM3
```

Logical Name:

NonPnP Path:

Host Computer: SEAN-PC

Copyright © 2007-2014 Annabooks, LLC. All Rights Reserved.
Applications is provided AS IS without any warranty.

5.3 *32-bit versus 64-bit Service Object Investigation*

With the test application and service object updated to POS for .NET 1.14 and .NET Framework 4.0, I tested to see what happens when the test application and service object are compiled to AnyCPU, x86, and x64. The test computer was running Windows 7 64-bit operating system. The table below shows the results:

		Service Object		
Application		**AnyCPU**	**x86**	**x64**
	AnyCPU	Yes	No	Yes
	x86	Yes	Yes	No
	x64	Yes	No	Yes

When the service object is compiled as AnyCPU, any compiled version of the application will run. If the service object is compiled with x86, only the x86 application will run. The inverse happens when the service object is compiled to x64. The next logical test is to run the tests on a Windows 7 32-bit operating system, but I will leave this test to the reader.