Developing POS .NET Core 3.1 Applications

By Sean D. Liming and John R. Malin
Annabooks – www.annabooks.com

August 2021

Our new book, *Professional's Guide to POS for Windows Runtime*, covers creating UWP applications that access POS devices. In the book, we mentioned that Visual Studio application development is evolving. .NET Core and WinUI are being developed, and some of these projects are at different stages. .NET Core 3.1 is available as Long-Term Support so developing applications built with .NET Core 3.1 will be supported for some time. As it turns out, support for the Windows Runtime accessible POS devices is available for .NET Core 3.1 applications using the Microsoft.Windows.SDK.Contracts NuGet package. This paper serves as a follow-up to *Professional's Guide to POS for Windows Runtime* and shows how to create a WPF .NET Core 3.1 POS application.

Requirements:

- Visual Studio 2019 or higher.
- .NET Core 3.1 Runtime installed on the development machine.
- POS devices that are supported by Windows Runtime, see: Supported Point of Service Peripherals.

Note: the same barcode scanner, POS Printer Ethernet connection, and Cash drawer called out in the book were used to develop these examples.

Barcode Scanner / Device Watcher Example

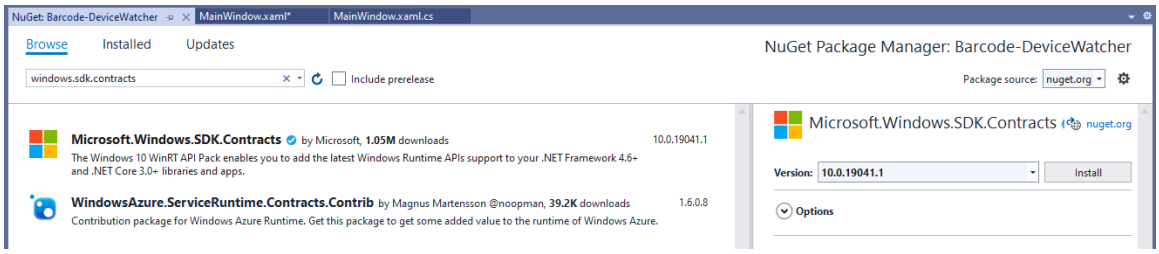
The first example will be a Barcode scanner. Device Watcher will be used for enumeration. The steps will be abbreviated.

1. Open Visual Studio.
2. Create a new WPF Application (choose the WPF project that supports .NET Core).
3. Name the project: *Barcode-DeviceWatcher*.
4. Select *.NET Core 3.1 Long-term support*.
5. In MainWindows.XAML, add a Label, a ListBox, and a Text Box control. The following XAML code shows the details assigned for each control

```
<Window x:Class="Barcode_DeviceWatcher.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Barcode_DeviceWatcher"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Label Content="Barcode Scanned Items" HorizontalAlignment="Left"
Height="27" Margin="34,32,0,0" VerticalAlignment="Top" Width="190"/>
        <ListBox x:Name="lstItems" HorizontalAlignment="Left" Height="189"
Margin="34,85,0,0" VerticalAlignment="Top" Width="740"/>
        <TextBox x:Name="txtStatus" HorizontalAlignment="Center" Height="28"
Margin="0,336,0,0" Text="Ready" TextWrapping="Wrap" VerticalAlignment="Top"
Width="796"/>
    </Grid>
</Window>
```

6. Save the project.

7. Open MainWindows.xaml.cs.
8. In Solution Explorer, right-click on *Dependencies* under the Barcode-DeviceWatcher project.
9. Select *Manage NuGet Packages…* from the context menu.
10. Click on *Browse* in the NuGet Window and type: *Windows.SDK.Contracts*.



11. Select *Microsoft.Windows.SDK.Contracts* and click *Install*. This adds the namespaces to access the Windows Runtime namespaces, which includes the PointOfService namespace.
12. Accept the next two dialogs that appear.
13. Save the project and close the NuGet window.
14. In MainWindow.xaml.cs, add the following code:

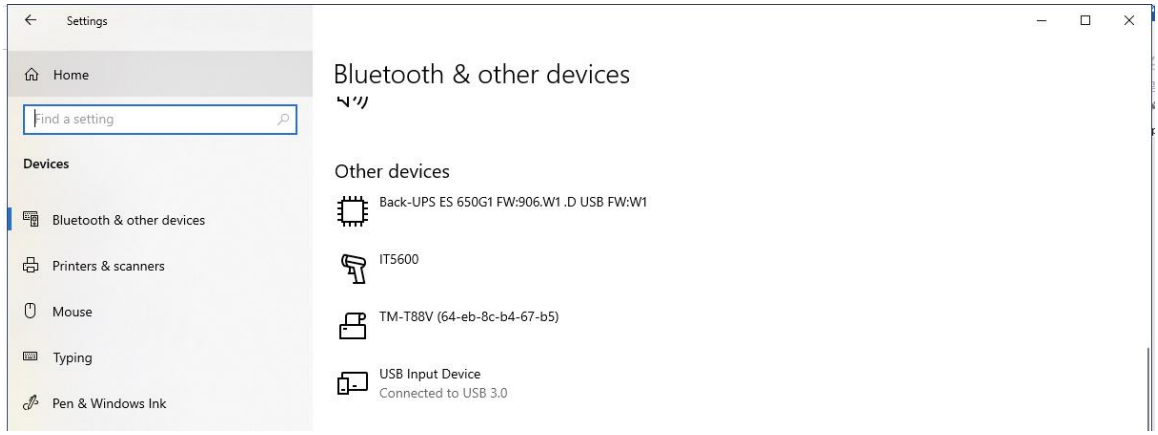
```
using System;
using System.Windows;
using Windows.Devices.PointOfService;
using Windows.Devices.Enumeration;
using Windows.Storage.Streams;


namespace Barcode_DeviceWatcher
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    ///
    public partial class MainWindow : Window
    {

        BarcodeScanner scanner = null;
        ClaimedBarcodeScanner claimedScanner = null;
        DeviceWatcher posDevWatcher =
DeviceInformation.CreateWatcher(BarcodeScanner.GetDeviceSelector(PosConnectionTy
pes.All));
        public MainWindow()
        {
            InitializeComponent();
            POSSetup();
        }

        private void POSSetup()
        {

            posDevWatcher.Added += PosDevWatcher_Added;
            posDevWatcher.Removed += PosDevWatcher_Removed;
            posDevWatcher.Start();
        }
```

```
        private void PosDevWatcher_Removed(DeviceWatcher sender,
DeviceInformationUpdate args)
        {
            claimedScanner = null;
            scanner = null;
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Barcode Scanner Removed"));

        }

        private async void PosDevWatcher_Added(DeviceWatcher sender,
DeviceInformation args)
        {
            scanner = await BarcodeScanner.FromIdAsync(args.Id);
            if(scanner != null)
            {

                claimedScanner = await scanner.ClaimScannerAsync();
                if(claimedScanner != null)
                {
                    scanner.StatusUpdated += Scanner_StatusUpdated;
                    claimedScanner.ReleaseDeviceRequested +=
ClaimedScanner_ReleaseDeviceRequested;
                    claimedScanner.DataReceived += ClaimedScanner_DataReceived;
                    claimedScanner.IsDecodeDataEnabled = true;
                    await claimedScanner.EnableAsync();

                    Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = "Barcode Scanner Claimed: " + args.Id));

                }
                else
                {
                    Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = "Barcode Scanner Not Claimed"));
                }

            }
            else
            {
                Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = "Barcode Scanner Not Found"));
            }
        }

        private void ClaimedScanner_DataReceived(ClaimedBarcodeScanner sender,
BarcodeScannerDataReceivedEventArgs args)
        {
            //Read the data from the buffer and convert to a string.
            var scanDataLabelReader =
DataReader.FromBuffer(args.Report.ScanDataLabel);
            string stringValue =
scanDataLabelReader.ReadString(args.Report.ScanDataLabel.Length);
            Application.Current.Dispatcher.Invoke((Action)(() =>
lstItems.Items.Add(stringValue)));
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Ready"));
        }
```

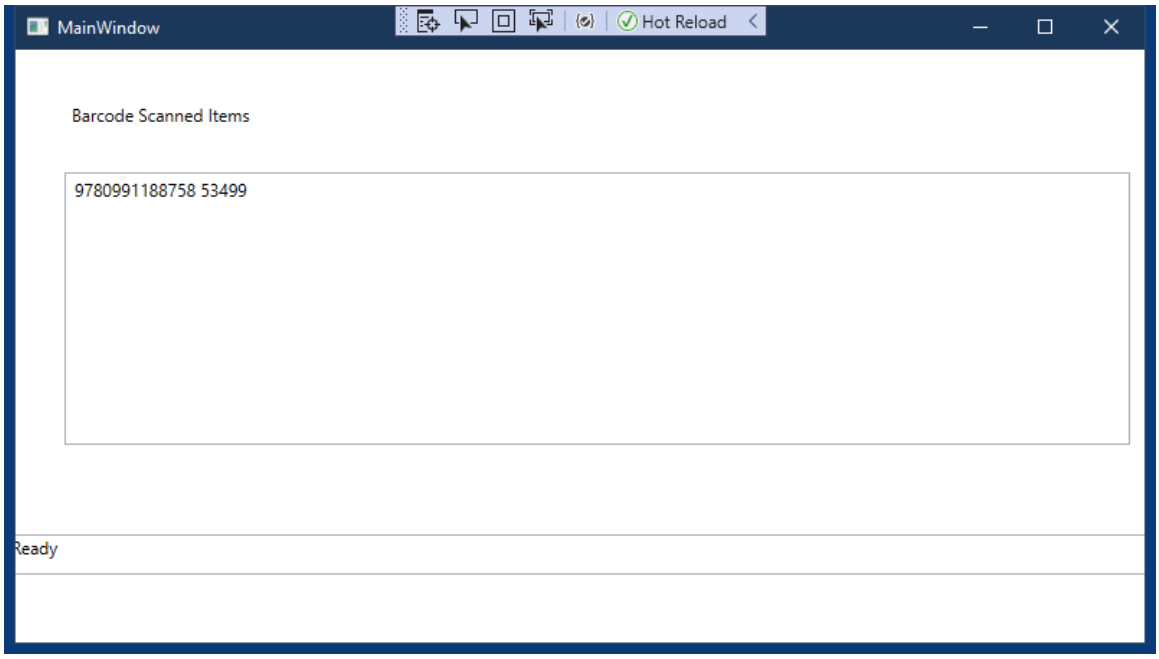
```
        private void ClaimedScanner_ReleaseDeviceRequested(object sender,
ClaimedBarcodeScanner e)
        {
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Barcode being released"));
        }

        private void Scanner_StatusUpdated(BarcodeScanner sender,
BarcodeScannerStatusUpdatedEventArgs args)
        {
            uint bcEXStatus = args.ExtendedStatus;
            BarcodeScannerStatus bcStatus = args.Status;
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Barcode status update: " + bcStatus.ToString()));
        }
    }
}
```

The project sets up the Device Watcher enumeration similar to Exercise 2.7 in the book.

15. Save the project.
16. Build the project and correct any errors.
17. Plug in the USB Barcode scanner. The barcode scanner should appear under *Settings->Devices->Bluetooth & other devices*. In the picture below, the HHP IT5600 is shown.



18. Run the application. The application should be able to enumerate the barcode scanner.
19. Scan a few barcodes.
20. Close the application when finished.



POSPrinter / Cash Drawer Example

We will create a POSPrinter Cash Drawer application that uses Device Watcher. The example is similar to Exercise 3.3 from the book.

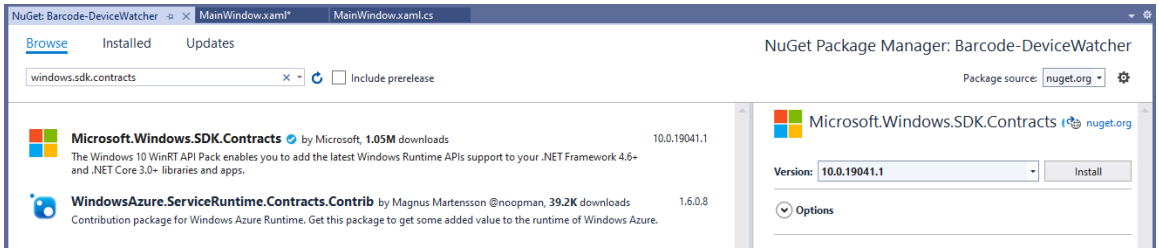
Note: The POS Printer must be connected via Ethernet.

1. Open Visual Studio.
2. Create a new WPF Application (choose the WPF project that supports .NET Core).
3. Name the project: *PrinterCD-DeviceWatcher*.
4. Select *.NET Core 3.1 Long-term support*.
5. In MainWindows.xaml add a Label, a TextBox, two Button controls, a status strip, and a text box in the status strip. The following XAML code shows the details assigned for each control.

```
<Window x:Class="PrinterCD_DeviceWatcher.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:PrinterCD_DeviceWatcher"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Label Content="Printer and Cash Drawer Example"
HorizontalAlignment="Left" Height="35" Margin="31,46,0,0"
VerticalAlignment="Top" Width="226"/>
        <TextBox x:Name="txtPrinterMSG" HorizontalAlignment="Left" Height="34"
Margin="31,102,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="543"/>
        <Button x:Name="btnPrint" Content="Send to Printer"
HorizontalAlignment="Left" Height="56" Margin="31,161,0,0"
VerticalAlignment="Top" Width="256" Click="btnPrint_Click"/>
        <Button x:Name="btnCD" Content="Open Cash Drawer"
HorizontalAlignment="Left" Height="54" Margin="31,244,0,0"
VerticalAlignment="Top" Width="256" Click="btnCD_Click"/>
```

```
        <StatusBar HorizontalAlignment="Center" Height="28" Margin="0,406,0,0"
VerticalAlignment="Top" Width="800">
            <TextBlock x:Name="txtStatus" Height="24" Text="Ready"
TextWrapping="Wrap" Width="798"/>
        </StatusBar>
    </Grid>
</Window>
```

6. Save the project.
7. Open MainWindows.xaml.cs.
8. In Solution Explorer, right-click on *Dependencies* under the Barcode-DeviceWatcher project.
9. Select *Manage NuGet Packages…* from the context menu.
10. Click on *Browse* in the NuGet Window and type *Windows.SDK.Contracts*.



11. Select *Microsoft.Windows.SDK.Contracts* and click *Install*. This adds the namespaces to access the Windows Runtime namespaces, which includes the PointOfService namespace.
12. Accept the next two dialogs that appear.
13. Save the project and close the NuGet window.
14. In MainWindow.xaml.cs add the following code:

```
using System;
using System.Windows;
using Windows.Devices.PointOfService;
using Windows.Devices.Enumeration;

namespace PrinterCD_DeviceWatcher
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        PosPrinter myPrinter = null;
        ClaimedPosPrinter myClaimedPrinter = null;
        CashDrawer myCashDrawer = null;
        ClaimedCashDrawer myClaimedCD = null;
        DeviceWatcher printerDevWatcher = null;
        DeviceWatcher cashdrawerDevWatcher = null;

        public MainWindow()
        {
            InitializeComponent();
```

```
            POSSetup();
        }

        private void POSSetup()
        {

            printerDevWatcher =
DeviceInformation.CreateWatcher(PosPrinter.GetDeviceSelector(PosConnectionTypes.
All));
            printerDevWatcher.Added += PrinterDevWatcher_Added;
            printerDevWatcher.Removed += PrinterDevWatcher_Removed;
            printerDevWatcher.Start();

            cashdrawerDevWatcher =
DeviceInformation.CreateWatcher(CashDrawer.GetDeviceSelector(PosConnectionTypes.
All));
            cashdrawerDevWatcher.Added += CashdrawerDevWatcher_Added;
            cashdrawerDevWatcher.Removed += CashdrawerDevWatcher_Removed;
            cashdrawerDevWatcher.Start();
        }

        private void CashdrawerDevWatcher_Removed(DeviceWatcher sender,
DeviceInformationUpdate args)
        {
            myClaimedCD.ReleaseDeviceRequested -=
MyClaimedCD_ReleaseDeviceRequested;
            myClaimedCD.Dispose();
            myClaimedCD = null;
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Cash Drawer Removed: " + args.Id));

        }

        private async void CashdrawerDevWatcher_Added(DeviceWatcher sender,
DeviceInformation args)
        {
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Found cash drawer " + args.Name + " | " + args.Id));

            myCashDrawer = await CashDrawer.FromIdAsync(args.Id);

            if (myCashDrawer != null)
            {

                myClaimedCD = await myCashDrawer.ClaimDrawerAsync();
                if (myClaimedCD != null)
                {
                    myClaimedCD.ReleaseDeviceRequested +=
MyClaimedCD_ReleaseDeviceRequested;
                    if (await myClaimedCD.EnableAsync())
                    {
                        Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = " Cash Drawer is claimed and enabled"));
                    }
                    else
                    {
                        Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = " Cash Drawer is NOT claimed and enabled"));
                    }
```

```
                }
            }
            else
            {
                Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = " Cash Drawer not found"));

            }
        }

        private async void MyClaimedCD_ReleaseDeviceRequested(ClaimedCashDrawer
sender, object args)
        {
            await sender.RetainDeviceAsync();
        }

        private void PrinterDevWatcher_Removed(DeviceWatcher sender,
DeviceInformationUpdate args)
        {
            myClaimedPrinter.ReleaseDeviceRequested -=
MyClaimedPrinter_ReleaseDeviceRequested;
            myClaimedPrinter.Dispose();
            myClaimedPrinter = null;
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Printer Removed: " + args.Id));
        }

        private async void PrinterDevWatcher_Added(DeviceWatcher sender,
DeviceInformation args)
        {
            Application.Current.Dispatcher.Invoke((Action)(() => txtStatus.Text
= "Found printer " + args.Name + " | " + args.Id));

            myPrinter = await PosPrinter.FromIdAsync(args.Id);
            if (myPrinter != null)
            {

                myClaimedPrinter = await myPrinter.ClaimPrinterAsync();

                if (myClaimedPrinter != null)
                {
                    myClaimedPrinter.ReleaseDeviceRequested +=
MyClaimedPrinter_ReleaseDeviceRequested;
                    if (await myClaimedPrinter.EnableAsync())
                    {

                        Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = " Printer is claimed and enabled"));

                    }
                    else
                    {
                        Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = " Printer is NOT claimed and enabled"));
                    }
                }
                else
                {
```

```
                    Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = "Claim Printer failed"));
                }
            }
            else
            {
                Application.Current.Dispatcher.Invoke((Action)(() =>
txtStatus.Text = "Printer Not Found"));
            }
        }

        private async void
MyClaimedPrinter_ReleaseDeviceRequested(ClaimedPosPrinter sender,
PosPrinterReleaseDeviceRequestedEventArgs args)
        {
            await sender.RetainDeviceAsync();
        }

        private async void btnPrint_Click(object sender, RoutedEventArgs e)
        {
            ReceiptPrintJob job = myClaimedPrinter.Receipt.CreateJob();
            job.PrintLine(txtPrinterMSG.Text);
            job.PrintLine("");
            job.PrintLine("");
            job.PrintLine("");
            job.PrintLine("");
            job.PrintLine("");
            job.PrintLine("");
            job.CutPaper();
            if (!await job.ExecuteAsync())
            {
                txtStatus.Text = "Print failed";
            }
        }

        private async void btnCD_Click(object sender, RoutedEventArgs e)
        {
            await myClaimedCD.OpenDrawerAsync();
        }
    }
}
```

15. Save the project.
16. Build the project and correct any errors.
17. Now comes the trick. Since the POS Printer is connected via Ethernet, you have to manually pair the POS Printer with the PC. Open *Settings->Devices->Bluetooth & other devices*.
18. Select *Add Bluetooth or other device*, and then select *Everything else* from the picker list.
19. The POS Printer should appear. Click *add*. The example below shows the TM-T88V printer paired with the PC.

20. Run the project. The printer and cash drawer should get enumerated.
21. Type something in the text box and click the print button. The text should be printed out.
22. Click on the open cash drawer button to open the cash drawer.
23. Close the application when finished.

Project Downloads

The two Visual Studio projects for the two examples show here are available on the *Professional's Guide to POS for Windows Runtime* page. There is an additional example for POS Printer and cash drawer using Device Picker Enumeration.

Summary: .NET Core is the first step in the new direction.

The Microsoft.Windows.SDK.Contract NuGet package provides .NET Core applications the access to POS Windows Runtime namespaces. UWP applications have their limits. .NET Core brings back the traditional application development that most developers have become accustomed to for over two decades. Expect more changes to come in the next year or so.