Optical Reader Library for Windows Phone 8

This article shows how to use the Optical Reader Library. It explains both how to integrate the Optical Reader Task into a Windows Phone 8 app in order to read QR and other barcodes, and how to extend the library to support other processors for image enhancement and decoding.

09 Feb 2014

Introduction

Creating a functional 1D or 2D barcode reader (or other optical reader) may consume a considerable amount of development time. Partly, this is because the camera systems (sensors and lenses) on different devices have different capabilities; for example, minimum focus distance and view angle. The time taken may also be impacted by constraints on the types of codes that need to be read, the licensing of decoder libraries that can be used, or the amount of "quality improvement" that needs to be performed on an image before it can be decoded.

The *Optical Reader Library* attempts to address these issues: it provides an optical code reading *Task* that is easy to integrate and simple to use, that is already optimized to use the best camera parameters for different devices, and which can be extended to support different image preprocessing and optical code decoders if needed.

By default, the Optical Reader Library enhances image quality (for example, by increasing contrast) using the Nokia Imaging SDK (v1.0.272.0) and decodes the image with ZXing.NET v0.12.0.0 (Apache License 2.0).

This article creates a very simple demo app to show how to use the *Optical Reader Task*. It also provides information on how to extend the Optical Reader Task's processing capabilities.

Optical reader demo app (after decoding a QR code).

Downloads

You can download relevant files here:

- OpticalReaderLib.0.1.0.0.nupkg.zip, a locally installable version of the library, includes API documentation
- OpticalReaderLib.0.1.0.0.zip, library source code, includes API documentation and architecture class diagram
- OpticalReaderLib.0.1.0.0.chm.zip, library API documentation (if .chm does not show any content, right-click on the .chm, select Properties and click Unblock)
- OpticalReaderApp.xap.zip, a simple demo application for quickly testing how the reader works

The Optical Reader Library source project uses Nuget Automatic Package Restore to retrieve the Nokia Imaging SDK and ZXing.NET libraries from Nuget.org. If you choose to build the library from scratch, make sure that you have *Nuget Package*

(C) Copyright Nokia 2014. All rights reserved.

Manager 2.7 or later installed. To do this, open Microsoft Visual Studio Express for Windows Phone and navigate to **Tools - Extensions and Updates - Updates - Visual Studio Gallery**, and check if there is a Nuget update available.

Using the Optical Reader Task in a Windows Phone 8 app

Using the Microsoft Visual Studio Express 2012 for Windows Phone, create a new Visual C# Windows Phone App project. In this document, we'll call our app "OpticalReaderDemo".

Installing the library from local Nuget package

The easiest way to add the Optical Reader Library to the project is to install it from a locally saved Nuget package. Therefore, download the library Nuget package and place it somewhere on your hard drive, for example **C:\Temp** , and click **Tools - Library Package Manager - Package Manager Settings - Package Manager - Package Sources**, and add **C:\Temp** as a new package source by clicking on the plus icon. Make sure that the newly added package source is ticked as **active**. Close the dialog by clicking **OK**.

Next, right-click on the *OpticalReaderDemo* project in the Solution Explorer panel, and select **Manage Nuget Packages**. On the left of the opening dialog, select **Online**, and under that, select **C:\Temp**. You should now see Optical Reader Library and an **Install** button. Click on **Install**. This installs the library and its dependencies (Nokia Imaging SDK v1.0.272.0 and ZXing.NET v0.12.0.0) to the project.

Adding the required capabilities

As the optical reader task uses the `PhotoCaptureDevice`, you need to add the `ID_CAP_ISV_CAMERA` capability to the project. In the Solution Explorer panel, open **OpticalReaderDemo/Properties/WMAppManifest.xml**, and on the *Capabilities* tab check `ID_CAP_ISV_CAMERA`. Although there may be other capabilities checked too by default, no other capabilities are required by this demo, nor by the library.

Invoking the optical reader task from code

Next, open the automatically generated **MainPage.xaml** and add an `Image` element for showing detected optical code thumbnail, a `TextBlock` element for displaying some information about the optical code, and a `Button` element for launching the optical reader task.

```
1. <phone:PhoneApplicationPage
2.     x:Class="OpticalReaderDemo.MainPage"
3.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5.     xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6.     xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7.     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8.     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```


(C) Copyright Nokia 2014. All rights reserved.

```
    mc:Ignorable="d"

    FontFamily="{StaticResource PhoneFontFamilyNormal}"

    FontSize="{StaticResource PhoneFontSizeNormal}"

    Foreground="{StaticResource PhoneForegroundBrush}"

    SupportedOrientations="Portrait" Orientation="Portrait"

    shell:SystemTray.IsVisible="False">

    <!--LayoutRoot is the root grid where all page content is placed-->

    <Grid x:Name="LayoutRoot" Background="Transparent">

        <Grid.RowDefinitions>

            <RowDefinition Height="Auto"/>

            <RowDefinition Height="*"/>

        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->

        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">

            <TextBlock Text="OPTICAL READER DEMO" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>

            <TextBlock Text="main page" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
```


(C) Copyright Nokia 2014. All rights reserved.

```
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>

            <!-- Image to show a thumbnail of the detected optical code -->
            <Image Grid.Row="0" x:Name="ResultImage"/>

            <!-- TextBlock to show information about the detected optical code -->
            <TextBlock Grid.Row="1" x:Name="ResultTextBlock" TextAlignment="Center" Margin="24"/>

            <!-- Button to launch the OpticalReaderTask -->
```


(C) Copyright Nokia 2014. All rights reserved.

```
            <Button Grid.Row="2" Margin="24" Content="launch optical reader task" Click="LaunchTaskButton_Click"/>
        </Grid>
    </Grid>

</phone:PhoneApplicationPage>
```

Next, switch to the code-behind-file **MainPage.xaml.cs** and add an instance of the optical reader task as a member, and also add a member for detected optical reader results. In the constructor page add an event handler for the task's `Completed` event, and catch the optical reader result in it to the member. Implement the handler for the task launch button `Click` event, simply by calling the task's Show method. This navigates the application to the optical reader viewfinder. Finally, as the Completed event handler function is called before the application navigates back from the viewfinder to this page, the result (if anything was found) is available for the page to use in the `OnNavigatedTo` method.

```
using Microsoft.Phone.Controls;
using System;
using System.Windows;
using System.Windows.Navigation;

namespace OpticalReaderDemo
{
    public partial class MainPage : PhoneApplicationPage
    {
        private OpticalReaderLib.OpticalReaderTask _task = new OpticalReaderLib.OpticalReaderTask();
        private OpticalReaderLib.OpticalReaderResult _result = null;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        public MainPage()
        {
            InitializeComponent();

            _task.Completed += OpticalReaderTask_Completed;
        }

        private void OpticalReaderTask_Completed(object sender, OpticalReaderLib.OpticalReaderResult e)
        {
            _result = e;
        }

        private void LaunchTaskButton_Click(object sender, RoutedEventArgs e)
        {
            _task.Show();
        }
```


(C) Copyright Nokia 2014. All rights reserved.

```

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            base.OnNavigatedTo(e);

            if (_result != null && _result.TaskResult == Microsoft.Phone.Tasks.TaskResult.OK)
            {
                ResultImage.Source = _result.Thumbnail;
                ResultTextBlock.Text = String.Format("{0} ({1} bytes)\n\n{2}",
                    _result.Format,
                    _result.Data != null ? _result.Data.Length : 0,
                    _result.Text);
            }

            _result = null;
        }
    }
}
```


(C) Copyright Nokia 2014. All rights reserved.

Running the application

Reading optical codes mostly only makes sense on a device, not on an emulator. Therefore, in order to run the application, connect a Windows Phone 8 device to your computer, select **Device** and **ARM** as the build target, and hit F5 to start running the application on the device.

Configuring the optical reader task

The *Optical Reader Task API* contains properties for configuring the reader. Probably the most important and useful configuration property is the `ObjectSize`, which you can use to set the target optical code size in millimeters. The library reader viewfinder will automatically zoom the camera preview so that a given size optical code fits the reader target rectangle when the device is at an optimal reading distance (minimum focus distance) from the code. This is very useful since it means that the UI implicitly guides the user of the application to position the device on the optimal reading distance where the device can still focus, but at the same time the maximum amount of sensor resolution is used to capture the code. How much to zoom on different devices is based on compiled-in information (camera focal length, sensor size) about a range of Nokia Lumia devices.

```
namespace OpticalReaderLib
{
    // Summary:
    //     Optical reader task is a Windows Phone chooser task implementation that allows
    //     easy and quick integration of 1D and 2D optical code reading functionality.
    public class OpticalReaderTask : ChooserBase<OpticalReaderResult>, IDisposable
```


(C) Copyright Nokia 2014. All rights reserved.

```
    {
        public OpticalReaderTask();

        // Summary:
        //     Reader camera focus interval, meaning the time that needs to pass without
        //     the reader finding anything before it attempts to re-focus the lens.  Default
        //     is 2500 milliseconds.
        public TimeSpan FocusInterval { get; set; }
        //
        // Summary:
        //     Target object real-life millimeter size. This affects the zoom factor used
        //     in the reader viewfinder.  Default is no zoom.
        public Size ObjectSize { get; set; }
        //
        // Summary:
        //     Processor to use for processing the frames.  Zxing processor is used if no
```


(C) Copyright Nokia 2014. All rights reserved.

```
        //      processor is set explicitly.

        public IProcessor Processor { get; set; }

        //

        // Summary:

        //      Set to true to require the user to confirm a found result by tapping on a

        //      result preview. If false, the first result found will be used automatically.

        //       Default is false.

        public bool RequireConfirmation { get; set; }

        //

        // Summary:

        //      Set to true to see debug frames in the reader viewfinder. Default is false,

        //      meaning that debug frames are not displayed.  Debug frames are meant to be

        //      used while developing normalizers, enhancers and processor, in order to get

        //      visual feedback on how the frame is modified during processing. Debug frames

        //      are not meant to be displayed in final consumer applications.

        public bool ShowDebugInformation { get; set; }

```


(C) Copyright Nokia 2014. All rights reserved.

```
        public void Dispose();

        //

        // Summary:

        //     Show the optical reader viewfinder.  Application is navigated to an optical

        //     reader viewfinder page and the Completed event is fired when user either

        //     navigates away from the viewfinder or if an optical code is detected.

        public override void Show();

    }

}
```

Extending the Optical Reader processing capabilities

Processors take in camera frames and return information about any optical codes they detected in the frame. In addition to the optical reader task, the *Optical Reader Library* contains interfaces that can be implemented to create custom processors.

```
using System;

using System.Threading.Tasks;

using Windows.Foundation;

namespace OpticalReaderLib

{
```


(C) Copyright Nokia 2014. All rights reserved.

```
    // Summary:
    //     Frame processor implementation interface.  Processors are all-in-one frame
    //     decoding systems. Depending on the implementation, they may use normalizers,
    //     enhancers and decoders to help in the job.
    public interface IProcessor
    {
        // Summary:
        //     Fired when there is a debugging frame available.  Debug frames are meant
        //     to be used while developing normalizers, enhancers and processor, in order
        //     to get visual feedback on how the frame is modified during processing.  Debug
        //     frames are not meant to be displayed in final consumer applications.
        event EventHandler<DebugFrameEventArgs> DebugFrameAvailable;

        // Summary:
        //     Attempts to decode an optically encoded code from the frame by processing
        //     it.
```


(C) Copyright Nokia 2014. All rights reserved.

```
        //
        // Parameters:
        //   frame:
        //     Frame to process.
        //
        //   area:
        //     Interesting frame area.
        //
        //   rotation:
        //     Frame rotation, how many degrees it should be rotated clockwise.
        //
        // Returns:
        //     Processing result or null if no code was found.
        Task<ProcessResult> ProcessAsync(Frame frame, Rect area, double rotation);
    }
}
```

Processors may further utilize other interfaces and constructs from the library, for example:

- `OpticalReaderLib.INormalizer`: Normalizers crop areas from frames and rotate them.
- `OpticalReaderLib.IEnhancer`: Enhancers attempt to improve frame image quality so that decoding is easier.
- `OpticalReaderLib.IDecoder`: Decoders attempt to find and decode optically encoded information from frames.

(C) Copyright Nokia 2014. All rights reserved.

The library contains built-in basic implementations for all the interfaces. So, for example, if you want to implement a simple custom processor that attempts to improve the camera frame image quality before feeding the frames to a ZXing.NET decoder, you can use the built-in `BasicNormalizer` and `ZxingDecoder`, and add a custom `IEnhancer` implementation between them.

```
using System.Threading.Tasks;

namespace OpticalReaderDemo
{
    public class CustomProcessor : OpticalReaderLib.BasicProcessor
    {
        public CustomProcessor() : base(new OpticalReaderLib.ZxingDecoder())
        {
            Normalizer = new OpticalReaderLib.BasicNormalizer();
            Enhancer = new CustomEnhancer();
        }
    }
}
```

```
using System;
using System.Threading.Tasks;

namespace OpticalReaderDemo
```


(C) Copyright Nokia 2014. All rights reserved.

```
{
    /// <summary>
    /// Dumb little custom enhancer that turns all frame pixels to either white or black
    /// depending on whether the grayscale value of the pixel exceeds a threshold or not,
    /// rotating the threshold for adjacent frames.
    /// </summary>
    class CustomEnhancer : OpticalReaderLib.IEnhancer
    {
        private uint _threshold = 0;
        private uint _jump = 16;
        private uint _min = 48;
        private uint _max = 160;

        public async Task<OpticalReaderLib.EnhanceResult> EnhanceAsync(OpticalReaderLib.Frame frame)
        {
            if (frame.Format == OpticalReaderLib.FrameFormat.Gray8)
            {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                _threshold = _threshold + _jump;

                if (_threshold > _max)
                {
                    _threshold = _min;
                }

                var f = await Task.Run<OpticalReaderLib.Frame>(() =>
                {
                    for (int i = 0; i < frame.Buffer.Length; i++)
                    {
                        frame.Buffer[i] = (byte)(frame.Buffer[i] < _threshold ? 0x00 : 0xff);
                    }

                    return frame;
                });

                return new OpticalReaderLib.EnhanceResult()
```


(C) Copyright Nokia 2014. All rights reserved.

```
                {
                    Frame = f
                };
            }
            else
            {
                throw new Exception("Dumb little custom enhancer only supports Gray8 encoded frames");
            }
        }
    }
}
```

Further reading

For detailed description on the library public APIs, see the library documentation (included in the downloads).

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.