Nokia Imaging SDK From Bottom To Top

In this article we are going to deep dive into the recently released Nokia Imaging SDK, we we'll cover some theoretical aspects and analyse some code examples crafted to show you how to achieve basic operation with the SDK, also we will build together a simple Windows Phone 8 application that uses the Nokia Imaging SDK. By the end of this article you should have a practical and theoretical knowledge of the SDK.

What is the Nokia Imaging SDK

The Nokia Imaging SDK, which by the time of writing this wiki it's in beta version, is a set of API that Nokia has made available to developers. It uses part of Nokia's technogloty used in it' own imaging application and it's an efficient library design to handle data captured by mobile device cameras. The SDK come out of the nox with 50 image filters that can be combined between them as many time as you want to let you achieve what you really want. Main features of the SDK are: decoding and encoding JPEG images, applying filters and effects, cropping, rotating and resizing. The logic behind those filter has been driven by an important need of speed+efficiency even while working with Hi-Res images. The only limitation of the SDK are two:

- The SDK is optimised for the CPUs that are powering the Windows phones. Consequently, the library will not work in the emulator.
- The Nokia Imaging SDK can only be used in Windows Phone 8 projects, it is not compatible with Windows Phone 7.

The SDK from a developer prospective

The SDK is provided as a Windows Runtime Library allowing developers from different backgrounds such as: C#, Vb and even C++ to use the SDK witout problems. It completly support the async/await pattern. Diving a bit more deeply inside the SDK we can notice that every activity of image manipulation is carried by one object called "EditingSession" which is the core of the library, this object comes with three constructor and it needs to be instantiated every time we change the image source, it implements the `IDisposable` interface which makes it suitable for the using statement. If we summarize the steps needed to use the Imaging SDK we can find a path that starts with the creation of the `EditingSession` object, by passing proper image data as parameter, than calling a method to apply an image filter or other manipulation and then render in some way the output store in the output buffer.

Prerequisites

In order to follow up with this wiki you need to have a machine running Windows 8 o Windows 8 Pro, a copy of Visual Studio 2012 + the Windows Phone 8 SDK or a copy of the recently released Visual Studio 2013 Preview that come out of the box with the Windows Phone 8 SDK installed. Also you need to be familiar with the common C# & XAML syntax since all the code examples provided in this wiki are written that way.

Note: If you are using Visual Studio 2012 you can download the Windows Phone 8 SDK from the Microsoft Download Center

Tip: Visual Studio 2013 is still in preview and need Windows 8.1 to run, so I suggest you to use the 2012 version

Notes while installing the SDK

The Nokia Imaging SDK can be installed/reference in our project in two different but equivalent ways. One way, the traditional one, is to download the SDK locally and then follow the "Add reference" procedure inside Visual Studio; The second and let me say more modern way, is through the NuGet package manager console. Installation through the NuGet console can be easily achieved by opening the console and typing "Package-Install NokiaImagingSDK", some sort of wizard will guide you to the bottom of the process.

Note: For a more deeply explained procedure of how adding the SDK to your project read Download and add the libraries to your project

Sometimes during compilation VS2012 shows error that most of the times are about a wrong platform configuration or related to the installation of the SDK by NuGet. More in detail the platform configuration error is caused by the presence of the unsupported option "All CPU" in the VS2012 configuration manager (obviously the one related to our project), in order to fix this issue you had to edit your platform configuration by removing the "All CPU" option. While for what concerns the NuGet error it may be caused by an outdated version of NuGet (it must be at least version 2.5) the solution in this case are two: one is to close and restart visual

studio, forcing it to reload all the reference of the project, while the second (which is the one that I raccomend) is to update your current version of NuGet Package manager, un-install the previously installed Imaging SDK NuGet package and then install it again.

Note: Read this for more information about compilation errors Typical Compilation Error

Let's get to work

Note: Before reading further make sure you have completely understand the Quick Start section on the Nokia Wiki

In this example we will create a basic application that given a photo, choose by the user in his camera roll, at the type of the next or previous button in the application bar will render the image on which has been automatically applied one of the 50 filters available in the Nokia Imaging SDK, the if the user wants, he can save that "new image" in his camera roll. So let's start by creating an empty Windows Phone 8 Application project.

1. Open Visual Studio
2. Go to File -> New -> Project
3. Choose Windows Phone
4. Select the first project template (Windows Phone App)
5. Give a name to your project (in my case NokiaImaginSDKDemo)

Tip: Before dealing with code make sure you have the correct configuration platform, and have installed on your PC the Nokia Imaging SDK

Note: If you don't know how to reference and external component or install a package read Download and add the libraries to your project

Defining the UI

Open in XAML mode the file **MainPage.xaml** and replace the default UI code with this one:

```
    <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="NOKIA IMAGING SDK DEMO" Style="{StaticResource
PhoneTextNormalStyle}" Margin="12,0"/>
            <TextBlock Text="home" Margin="9,-7,0,0" Style="{StaticResource
PhoneTextTitle1Style}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
         <Image x:Name="imageBox" Stretch="UniformToFill" Width="400" Height="550"
VerticalAlignment="Top"/>
        </Grid>
    </Grid>
```

Adding an ApplicationBar for a simple user interaction

Now we need to provide a way for the user to interact with our application, to do so we simply instantiate an Applicationbar with 4 buttons that will allow the user to pick an image from its Camera Roll and back and forth between our effects list (we are not going to use all the filters available in the Nokia Imaging SDK, but just a couple). You can add an application bar to your **MainPage.xaml** UI by adding this code to your code behind file (MainPage.xaml.cs)

```
        private void BuildAppBar()
        {
            // Instantiate the system ApplicationBar
            ApplicationBar = new ApplicationBar();

            // Add using System.Windows.Media;
            // We set the background color of the app bar to black
            ApplicationBar.BackgroundColor = Color.FromArgb(255, 0, 0, 0);
            // And the foreground to white
            ApplicationBar.ForegroundColor = Color.FromArgb(255, 255, 255, 255);

            // Creation of all the ApplicationBarButton with a specified icon, text and
callback
            ApplicationBarIconButton pickPhotoButton = new ApplicationBarIconButton(new
Uri("Assets/Folder-Open.png", UriKind.Relative));
            pickPhotoButton.Text = "browse";
            pickPhotoButton.Click += pickPhotoButton_Click;

            ApplicationBarIconButton savePhotoButton = new ApplicationBarIconButton(new
Uri("Assets/Save.png", UriKind.Relative));
            savePhotoButton.Text = "save";
            savePhotoButton.Click += savePhotoButton_Click;

            ApplicationBarIconButton prevPhotoButton = new ApplicationBarIconButton(new
Uri("Assets/Previous.png", UriKind.Relative));
            prevPhotoButton.Text = "prev";
            prevPhotoButton.Click += prevPhotoButton_Click;

            ApplicationBarIconButton nextPhotoButton = new ApplicationBarIconButton(new
Uri("Assets/Next.png", UriKind.Relative));
            nextPhotoButton.Text = "next";
            nextPhotoButton.Click += nextPhotoButton_Click;

            // Add the button to the AppBar
            ApplicationBar.Buttons.Add(pickPhotoButton);
            ApplicationBar.Buttons.Add(savePhotoButton);
            ApplicationBar.Buttons.Add(prevPhotoButton);
            ApplicationBar.Buttons.Add(nextPhotoButton);
        }

        // This is the method that handles the callback from a Tap on the
AppBarIconButton, it simply launches the PhotoChoosertask
        private void pickPhotoButton_Click(object sender, EventArgs e)
        {
           // Create a PhotoChooserObject
            PhotoChooserTask picker = new PhotoChooserTask();

            // Define a callback and show it
            picker.Completed += picker_Completed;
            picker.Show();
        }

       // Sets the source of our Image control (it will display our edited/choosed
image)
        private void SetImageBoxSource(Stream _imgStream)
```

```
        {
            BitmapImage tmpImage = new BitmapImage();
            tmpImage.SetSource(_imgStream);
            imageBox.Source = tmpImage;
        }

       // event callback for the PhotoChooser
        private void picker_Completed(object sender, PhotoResult e)
        {
            if (e.TaskResult != TaskResult.OK)
                return;

            // This method set the source of our Image control, so it displays the
choosed/edited picture
            SetImageBoxSource(e.ChosenPhoto);

            // Save the choosed photo stream, "photoStream" is a global variable
            photoStream = e.ChosenPhoto;
        }
```

Ok after adding this piece of code we have the logic to instatiate the system ApplicationBar with a Button that allows the user to pick an image from his camera roll, the last thing to do before dealing with the fun stuff, the Nokia Imaging SDK, is to update the constructor of our MainPage with a call to our BuildAppBar method. At the end the MainPage method should look like this.

```
        // Constructor
        public MainPage()
        {
            InitializeComponent();
            BuildAppBar();
        }
```

Ok now we are ready to play with Images.

Finally playing with the SDK

As we have said before the goal of this demo app is to apply sequentially some effects to an image, and if the users want save those images or one of those in his camera roll. But the problem is: How do we achieve this? How we can call multiple methods in a complete automatic way without running into reflections?...The answer is simple, we will use enumerations. Enumeration are often underestimated but in this case the logic that drives them is our key to success. As you may know enumerations provides a mnemonic way to define particular data, and they can be also indexed with integer numbers, starting from 0 to the number of elements. The example below will clarify a bit your ideas:

```
      private enum NokiaFilters // we declare our enumaration, with fields called like
the filter available in the SDK
      {
             Antique,
             Lomo,
             ...
       }
       // In this case I've considerated only 2 filter, so the lenght of our enumeration
will be 1

       // Let's declare a variable of type NokiaFilters
      NokiaFilters myFilter = NokiaFilters.Lomo;
      // Nothing weird so far, but as you may know we can do also this
      NokiaFilters myFilter2 = (NokiaFilter) 1;
```

```
      // The cast will convert our integer value of 1 into its corresponding value as
NokiaFilter, in this case Lomo
```

Ok now we have figured out how to call, without explicitly specified its name, a method that will apply a filter to our image. And her's the code that gets executed when the user press the next or the previous button, the code of our enumeration and the save button:

```
        private enum NokiaFilterList
        {
            Antique,
            AutoLevels,
            Cartoon,
            ColorBoost,
            Greyscale,
            HueSaturation,
            Lomo
        }
        private async void nextPhotoButton_Click(object sender, EventArgs e)
        {
           // We check if the variable "indexFilter" which is global and sore the last
effect we have used is greater that the lenght of our NokiaFilterEnumeration
            if (filterIndex >= Enum.GetValues(typeof(NokiaFilterList)).Length)
                return;

            // Here we use the power of the using statement to create an EditingSession
with the stream of the original image (the one choosed by the user)
           // and then we call the AddFilterAndCreateImageSource to apply a filter
(specified by the filterIndex variable) and render a stream that is goning to be used
           // as source for our Image control. You can finde the code of
AddFilterAndCreateImageSource below.
            using (EditingSession sess = await
EditingSessionFactory.CreateEditingSessionAsync(photoStream))
            {
                Stream newImage = await AddFilterAndCreateImageSource(sess,
(NokiaFilterList)filterIndex);

               // We save the stream of this new image just in case user wants to save
it.
                savePhotoStream = newImage;
                SetImageBoxSource(newImage);
                filterIndex++;
            }
        }

        // With the previous button the logic is the same, but we check if the
"indexfilter" variable is minor than zero, if so we set it to zero
        private async void prevPhotoButton_Click(object sender, EventArgs e)
        {
            if (filterIndex < 0)
            {
                filterIndex = 0;
                return;
            }

            // We use an editing session and our crafted AddFilterAndCreateImageSource
method once more
```

```
            using (EditingSession sess = await
EditingSessionFactory.CreateEditingSessionAsync(photoStream))
            {
                filterIndex--;
                Stream newImage = await AddFilterAndCreateImageSource(sess,
(NokiaFilterList)filterIndex);
                savePhotoStream = newImage;

                 // We render our image into the Image control
                 SetImageBoxSource(newImage);
            }
        }

        private void savePhotoButton_Click(object sender, EventArgs e)
        {
            MediaLibrary lib = new MediaLibrary();

            // The media library class does not seek automatically the begin of the
stream, so we must do it
            savePhotoStream.Seek(0, SeekOrigin.Begin);
            Picture pic = lib.SavePictureToCameraRoll("NokiaImagingSDK_DemoImg",
savePhotoStream);
        }
```

Here's the code of `AddFilterAndCreateImageSource` which returns a Task<Stream> and takes as parameter an `EditingSession` and a `NokiaFilterList` value:

```
        // As you can see it is a async task method, which means it can be awaited :)
        private async Task<Stream> AddFilterAndCreateImageSource(EditingSession
_session, NokiaFilterList _filter)
        {
            switch (_filter)
            {
                case NokiaFilterList.Antique:
                    _session.AddFilter(FilterFactory.CreateAntiqueFilter());
                    break;
                case NokiaFilterList.AutoLevels:
                    _session.AddFilter(FilterFactory.CreateAutoLevelsFilter());
                    break;
                case NokiaFilterList.Cartoon:
                    // We use true because we want to distint image edges
                    _session.AddFilter(FilterFactory.CreateCartoonFilter(true));
                    break;
                case NokiaFilterList.ColorBoost:
                    // We use a default value of one, the range of suggested values goes
from -1.0 to 1.0
                    // 1.0 rappresent the color gain of the image
                    _session.AddFilter(FilterFactory.CreateColorBoostFilter(1.0));
                    break;
                case NokiaFilterList.Greyscale:
                    _session.AddFilter(FilterFactory.CreateGrayscaleFilter());
                    break;
                case NokiaFilterList.HueSaturation:
                    // I've 180 and 160 randomly you should use more appropriate
parameters
                    // or let the user decide.
```

```
                    _session.AddFilter(FilterFactory.CreateHueSaturationFilter(180,
160));
                    break;
                case NokiaFilterList.Lomo:
                    // Random parameters here as well, I repeat, you should use more
appropriate values
                    // or let the user decide.
                    _session.AddFilter(FilterFactory.CreateLomoFilter(0.5, 0.9,
LomoVignetting.Medium, LomoStyle.Green));
                    break;
                default:
                    break;
            }

            IBuffer myBuf = await _session.RenderToJpegAsync();

            // AsStream is an extension method provided by the
System.Runtime.InteropServices.WindowsRuntime namespace.
            return myBuf.AsStream();
        }
```

Conclusion

Well guys that's all, if you compile and run this application it should do what we expect, you can download the entire project by the link provided in this wiki metadata. if you have any question, suggestion or you want to point out a problem (there may be some, I've worked on this wiki at night after work :))in this project/code fell free to comment. Thanks!!