VoiceMaging for Windows Phone 8

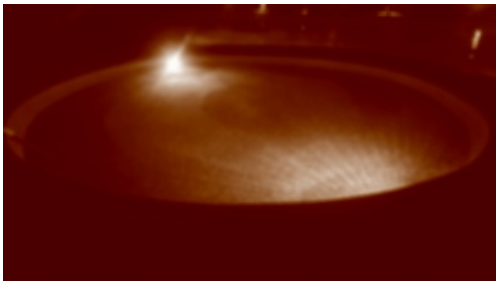
This article explains how to apply effects to images using voice commands.

Introduction

Nokia recently launched the Nokia Imaging SDK which includes filters to make image editing easier for developers. This article provides a walkthrough on how to use imaging API's along with Speech API's for Windows Phone 8 to create a cool innovative application for Windows Phone 8 powered devices.

The application we are going to create *VoiceMaging* derives it's name from the words *Voice* and *Imaging*. It allows users to apply filters to images using normal speech.

Preview of a sample image before and after applying Blur and Sepia filters by voice.



Prerequisites

Make sure you have Windows Phone 8 SDK and Nokia Imaging SDK installed.

You also need to have 'en-US' language pack installed on your Windows Phone device to use speech recognition functionality.

Once you are done with installing above stuff, let us begin with application implementation.

Note: See adding libraries to the project for details on downloading and adding imaging libraries to your project

Required Capabilities

The application requires some capabilities for using the Speech and Imaging APIs. Enable them in **WPAppManifest.xml** file under the Capabilities tab.

- `ID_CAP_MEDIALIB_PHOTO`
- `ID_CAP_SPEECH_RECOGNITION`
- `ID_CAP_MICROPHONE`

User Interface

Let us start with creating user interface (UI). The UI should be simple and clean, and yet still provide all the required functionality.

Things you need in the application UI are:

1. A 'pick image' button to let user select an image for editing.
2. Two `ImageView`'s, one to display selected image and one for displaying edited image.
3. A 'mic' button to start speech recognition session.
4. A 'save' button to save edited image to images gallery.
5. An 'undo' button to undo last applied image filter.

Code implementation of the UI fulfilling above needs will look something like:

```
<Canvas x:Name="LayoutRoot" Background="Transparent">
        <Canvas.Resources>
            <Storyboard x:Name="hide">
                <DoubleAnimation
                    Storyboard.TargetName="buttonPanel"
                    Storyboard.TargetProperty="(Canvas.Top)"
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    From="10" To="480" Duration="0:0:0.5" />
                <DoubleAnimation
                    Storyboard.TargetName="buttonPanel"
                    Storyboard.TargetProperty="Opacity"
                    From="1.0" To="0.5" Duration="0:0:0.5" />
                <DoubleAnimation
                    Storyboard.TargetName="rotateRoot"

Storyboard.TargetProperty="(UIElement.RenderTransform).(RotateTransform.Angle)"
                    From="0" To="90" Duration="0:0:0.5" />
            </Storyboard>
            <Storyboard x:Name="show">
                <DoubleAnimation
                    Storyboard.TargetName="buttonPanel"
                    Storyboard.TargetProperty="(Canvas.Top)"
                    From="480" To="10" Duration="0:0:0.5" />
                <DoubleAnimation
                    Storyboard.TargetName="buttonPanel"
                    Storyboard.TargetProperty="Opacity"
                    From="0.5" To="1.0" Duration="0:0:0.5" />
                <DoubleAnimation
                    Storyboard.TargetName="rotateRoot"

Storyboard.TargetProperty="(UIElement.RenderTransform).(RotateTransform.Angle)"
                    From="90" To="0" Duration="0:0:0.5" />
            </Storyboard>
        </Canvas.Resources>
        <Image x:Name="EditedImage"  Height="480" Width="800" Stretch="UniformToFill"
Canvas.Top="10" Canvas.ZIndex="0"/>
        <Image x:Name="OriginalImage" Height="480" Width="800" Stretch="UniformToFill"
Canvas.Top="10" Canvas.ZIndex="0" />
        <Slider Height="84" x:Name="slider1" Width="369" Visibility="Collapsed"/>
        <StackPanel x:Name="buttonPanel" Canvas.Left="720" Canvas.Top="10"
Orientation="Vertical" Width="80" Height="410">
            <StackPanel.Background>
                <ImageBrush ImageSource="/Assets/Images/panel_background.png" />
            </StackPanel.Background>
            <Image x:Name="microphoneImage" Source="/Assets/Images/microphone.png"
Height="68" Width="68" MouseLeftButtonUp="MicrophoneClicked"/>
            <Image x:Name="resetImage" Source="/Assets/Images/reset.png" Height="68"
Width="68" MouseLeftButtonUp="ResetClicked"/>
            <Image x:Name="undoImage" Source="/Assets/Images/undo.png" Height="68"
Width="68"  MouseLeftButtonUp="UndoClicked"/>
            <Image x:Name="openImage" Source="/Assets/Images/open.png" Height="68"
Width="68" MouseLeftButtonUp="PickImage_Click"/>
            <Image x:Name="OriginalImageIcon" Height="68" Width="68"
Stretch="UniformToFill" MouseLeftButtonUp="OriginalImageIcon_Click"/>
            <Image x:Name="saveImage" Source="/Assets/Images/save.png" Height="68"
Width="68" MouseLeftButtonUp="SaveImage_Click" />
        </StackPanel>
        <Image x:Name="rotateRoot" Source="/Assets/Images/settings.png" Height="60"
Width="60" Stretch="UniformToFill" Canvas.Left="730" Canvas.Top="420"
MouseLeftButtonUp="RotateRoot_Clicked">
            <Image.RenderTransform>
                <RotateTransform CenterX="34" CenterY="34" />
            </Image.RenderTransform>
        </Image>
```


(C) Copyright Nokia 2014. All rights reserved.

```
    </Canvas>
```

All the buttons are placed inside a vertical `StackPanel` with a foggy background to give it a nice style and to make sure buttons are clearly visible. The animations in `Canvas` resources are associated with `StackPanel` to hide and show it on user demand.

I have used a `Canvas` layout since it allows buttons to be placed over the image. This way you will have the whole screen to display image without wasting any dedicated space for buttons.

Speech Recognition

Since the basic feature of this application is to allow user edit images using speech, you need to implement speech recognition. I will not be going into depth about speech recognition - see my previous article Speech Enabled Calculator For Windows Phone 8 and MSDN atricle for details.

For using speech recognition you need to:

1. Instantiate an object of `SpeechRecognizer`.
2. Define speech grammar containing speech recognition rules.
3. Provide `SpeechRecognizer` with the speech grammar to let it know what speech commands to look for.
4. Start a speech recognition session.

Instantiating SpeechRecognizer

you have option of using SpeechRecognizer with UI (SpeechRecognizerUI) or without UI (SpeechRecognizer). The later one is more suitable for the application since a user will want to look just at the image during the editing session, displaying a pop up every now and then will block the image and is not a good design practice.

Use following line of code to instantiate an object of `SpeechRecognizer` class.

```
SpeechRecognizer recoInstance = new SpeechRecognizer();
```

Defining speech grammar

you again have two options to define speech rules. Inline or using a SRGS file. Using a SRGS file for defining speech grammar is always a good practice since it keeps a separation between your C# code and xml speech rules. Yes, a SRGS file is defined using xml. Look at SRGS Grammar for more details.

Now what exactly you want to recognize is *image editing speech commands* from the user. Like 'Sepia', 'Blur', 'Fog', 'Sketch' etc. For doing this you first need to create a SRGS file following below steps

- Right click the windows phone project in visual studio.
- Click on Add New Item.
- Select "SRGS Grammar" from list.
- Name it as **SpeechRules.xml**.
- Save it.

Now open the created SRGS file and replace its contents by below code.

```
<?xml version="1.0" encoding="utf-8" ?>
```


(C) Copyright Nokia 2014. All rights reserved.

```
<grammar version="1.0" xml:lang="en-IN" root="lens" tag-format="semantics-ms/1.0"
         xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:sapi="http://schemas.microsoft.com/Speech/2002/06/SRGSExtensions">
  <rule id="lens" scope="public">
    <one-of>
      <item>cartoon</item>
      <item>antique</item>
      <item>negative</item>
      <item>sepia</item>
      <item>blur</item>
      <item>brightness</item>
      <item>fog</item>
      <item>gray scale</item>
      <item>magic pen</item>
      <item>milky</item>
      <item>mirror</item>
      <item>oily</item>
      <item>sketch</item>
      <item>undo</item>
      <item>reset</item>
    </one-of>
  </rule>
</grammar>
```

Here you have a rule with `id="lens"`. Inside which you have a `<one-of>` tag containing lots of `<item>` elements. This SRGS file tells the `SpeechRecognizer`, *Let me know if user speaks out any of these phrases* (each `<item>` corresponds to a phrase).

Assigning SRGS file to Speech Recognizer

Now you need to direct `SpeechRecogizer` to use the grammar file for detecting user spoken speech commands. Do it using below code snippet.

```
Uri calculatorRulesUri = new Uri("file://" +
Windows.ApplicationModel.Package.Current.InstalledLocation.Path + @"/SpeechRules.xml",
UriKind.RelativeOrAbsolute);

// Add the grammar to the grammar set.
recoInstance.Grammars.AddGrammarFromUri("calculatorRulesUri", calculatorRulesUri);
```

Starting speech recognition session

Now that everything is set up, start a speech recognition session

```
SpeechRecognitionResult recoResult = await recoInstance.RecognizeAsync();
```

Since no speech recognition UI will be displayed, to let user know speech recognition is happening in background, you can change the microphone button image to glow or something similar.

```
microphoneImage.Source = (ImageSource)new
ImageSourceConverter().ConvertFromString("/Assets/Images/microphone_glow.png");
```

Now once user speak out any of the phrases mentioned in the SRGS file it will be reported to you in the `SpeechRecognitionResult` instance.

(C) Copyright Nokia 2014. All rights reserved.

```
String recognizedText = recoResult.Text;
```

Overall speech recognition implementation will look like

```
//Instantiate speech recognizer object
SpeechRecognizer recoInstance = new SpeechRecognizer();

// Initialize a URI with a path to the SRGS-compliant XML file.
Uri calculatorRulesUri = new Uri("file://" +
Windows.ApplicationModel.Package.Current.InstalledLocation.Path + @"/SpeechRules.xml",
UriKind.RelativeOrAbsolute);

// Add the grammar to the grammar set.
recoInstance.Grammars.AddGrammarFromUri("calculatorRulesUri", calculatorRulesUri);

SpeechRecognitionResult recoResult = await recoInstance.RecognizeAsync();
microphoneImage.Source = (ImageSource)new
ImageSourceConverter().ConvertFromString("/Assets/Images/microphone.png");
String recognizedText = recoResult.Text;
```

Once you have the recognised text you can do a string comparison on it and take corresponding actions on the image using Nokia Imaging SDK libraries.

Image Editing

Before the launch of Nokia Imaging SDK, image editing was a really difficult task. But Nokia Imaging SDK API's take out all the pain and makes editing images much easier than ever. Nokia Imaging SDK include decoding and encoding JPEG images, applying filters and effects, cropping, rotating and resizing. It provides more than 50 pre-made filters and effects that have been specifically developed for mobile imaging, with speed and memory performance as key drivers. The SDK is super-fast, thanks to meticulous memory and code optimisation. The patented JPEG technology, RAJPEG, contributes to making this possible, as it allows access to any image data without decoding the whole image.

Basics

Below mentioned are the foundation classes in imaging library:

- `EditingSession`
- `IBuffer`
- `Bitmap`
- `WriteableBitmap`

Go through Nokia Imaging Core Concepts for a descent understanding of these classes.

Note: I recommend going throught Nokia Imaging Core Concepts before proceeding further.

Picking up image from gallery

To start off with image editing part, you first need to have functionality for selecting an image from the gallery in place. Windows Phone Choosers serve exactly the same purpose, Photo Chooser more specifically.

Once you are done with photo chooser implementation, it should look something like

```
private Stream imageStream;

private void PickImage_Click(object sender, System.Windows.Input.MouseButtonEventArgs e)
{
    PhotoChooserTask chooser = new PhotoChooserTask();
    chooser.Completed += PickImageCallback;
    chooser.Show();
```


(C) Copyright Nokia 2014. All rights reserved.

```
}

private async void PickImageCallback(Object sender, PhotoResult e)
{
    if (e.TaskResult == TaskResult.OK)
    {
        //selected image is available in e.ChosenPhoto as imageStream
        imageStream = e.ChosenPhoto;
    }
}
```

At this point your application can pick an image from gallery and recognize user spoken words. What remains is applying effects/filters on the image and save edited image back to gallery on user's demand. To do that we need

- 2 `WriteableBitmap`. 1 for the picked image and another one for the edited image.
- A `Stream` object to store picked image stream.
- An `EditingSession` instance to handle image editing.
- An `IBuffer` instance to store edited image jpeg data for saving it to image gallery.

Declare these as your class variables

```
private WriteableBitmap editedBitmap;
private WriteableBitmap originalImageBitmap;
private IBuffer jpegOut;
private Stream imageStream;
private EditingSession session;
```

After an image is picked from gallery, it should be displayed on the screen. Best place to do that is *Photo Chooser* callback method. It is also a good place to instantiate the `EditingSession`.

```
private async void PickImageCallback(Object sender, PhotoResult e)
{
    if (e.TaskResult == TaskResult.OK)
    {
        imageStream = e.ChosenPhoto;
        session = await EditingSessionFactory.CreateEditingSessionAsync(imageStream);
        try
            {
            //Decode the jpeg for showing the original image
            await session.RenderToBitmapAsync(originalImageBitmap.AsBitmap());
            //Force the framework to redraw the Image.
            originalImageBitmap.Invalidate();
            }
        catch (Exception exception)
            {
            MessageBox.Show("Exception:" + exception.Message);
            }
    }
}
```

Applying effects to picked image

Now you need to apply effects on the `originalImageBitmap` by using `EditingSession`'s `AddFilter()` method. `AddFilter()` method takes one parameter of type `IFilter`, which is the the type of filter you want to apply on the current session's image (ex. Sepia, Milky, Blur etc). Below code snippet will apply Sepia effect to the current sessions image and display it.

(C) Copyright Nokia 2014. All rights reserved.

```
//Add sepia filter to the session
session.AddFilter(FilterFactory.CreateSepiaFilter());

//Execute the filtering and render to a bitmap
await session.RenderToBitmapAsync(editedBitmap.AsBitmap());

//Force the framework to redraw the Image.
editedBitmap.Invalidate();
```

The filter to apply will depend on the commands recognized using Speech Recognition. So create a method named `ApplyFilters` containing all the logic to apply filters based on the input parameter (user spoken command). `ApplyFilters` method will be called after every successful speech recognition.

```
. . .
SpeechRecognitionResult recoResult = await recoInstance.RecognizeAsync();
microphoneImage.Source = (ImageSource)new
ImageSourceConverter().ConvertFromString("/Assets/Images/microphone.png");
String recognizedText = recoResult.Text;

//call method to apply user spoken filter to the image
ApplyFilters(recognizedText);
```

The logic inside `ApplyFilters` method will make a string comparison on `recognizedText` and apply corresponding filter.

```
if (recognizedText.Equals("sepia", StringComparison.CurrentCultureIgnoreCase))
{
    session.AddFilter(FilterFactory.CreateSepiaFilter());
    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());
    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();
}
else if (recognizedText.Equals("negative", StringComparison.CurrentCultureIgnoreCase))
{
    session.AddFilter(FilterFactory.CreateNegativeFilter());
    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());
    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();
}
else if (recognizedText.Equals("cartoon", StringComparison.CurrentCultureIgnoreCase))
{
    session.AddFilter(FilterFactory.CreateCartoonFilter(true));
    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());
    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();
}
. . .
```

Handling user input

Some *Filters* don't require any input parameters or contains those can be hard coded ex Sepia, Milky etc. But some Filters requires inputs to come from user, for example *Blur Filter*. The amount of blurriness should come from user so you need to have some way of taking this input. The best way is to provide user with a Slider Control.

You can set `Maximum` and `Minimum` values for the slider depending on the filter's input parameter. And register for Slider's

(C) Copyright Nokia 2014. All rights reserved.

`MouseLeftButtonUp` event for getting change in slider's value. You might imagine why `MouseLeftButtonUp` instead of `ValueChanged`. It's because if you register for `ValueChanged` event and user slides his finger on the slider, 100's of `ValueChanged` events are generated, enough to break your code if not properly handled. But if you register for `MouseLeftButtonUp` then the event will fired only once(when user will take his finger off the `Slider`). Add a `Slider` to your xaml file

```
<Slider Height="84" Canvas.Top="20" Canvas.Left="10" Name="slider1" Width="369"
Visibility="Collapsed" />
```

The slider should be visible only for filter's needing some user input hence, `Visibility="Collapsed"`. The slider must be Visible for Filters needing user input say Blur, Brightness etc. Best place to do this is in the `ApplyFilters` method.

```
else if (recognizedText.Equals("blur", StringComparison.CurrentCultureIgnoreCase))
{
    //Apply default blur
    session.AddFilter(FilterFactory.CreateBlurFilter(BlurLevel.Blur1));

    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());

    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();

    //Enable the slider
    slider1.Visibility = System.Windows.Visibility.Visible;
    slider1.Minimum = 0.0;
    slider1.Maximum = 1.0;

    //This if for safety purpose because WP allows same events to be registered multiple
    //times firing an event for each registration.
    slider1.MouseLeftButtonUp -= BlurSliderChanged;

    //Register for MouseLeftButtonUp event
    slider1.MouseLeftButtonUp += BlurSliderChanged;
}
```

Now implement `BlurSliderChanged` event handler

```
private async void BlurSliderChanged(object sender,
System.Windows.Input.MouseButtonEventArgs e)
{
    //Undo last blur to avoid applying blur recursively
    session.Undo();

    //cast double value from slider to int because CreateBlurFilter
    //requires parameter of type BlurLevel which is enum and
    //int can be cast to enum
    int blurValue = (int)slider1.Value;

    //set sliders value to rounded int value
    slider1.Value = blurValue;

    //cast int blurValue to BlurLevel enum, since that's what BlurFilter requires
    session.AddFilter(FilterFactory.CreateBlurFilter((BlurLevel)blurValue));

    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());
```


(C) Copyright Nokia 2014. All rights reserved.

```
    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();
}
```

Similarly for `BrighnessFilter` a double parameter is needed ranging from -1.0 to 1.0 (0.0 indicates no adjustment). So here you can have a slider with `Minimum` = 0, `Maximum` = 2 and default value set to 1 (indicating no effect)

```
else if (recognizedText.Equals("brightness", StringComparison.CurrentCultureIgnoreCase))
{
    //Apply default brigtness of 0.0
    session.AddFilter(FilterFactory.CreateBrightnessFilter(0.0));

    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());

    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();

    //Enable the slider
    slider1.Visibility = System.Windows.Visibility.Visible;
    slider1.Minimum = 0.0;
    slider1.Maximum = 2.0;
    slider1.Value = 1.0;

    slider1.MouseLeftButtonUp -= BrightnessSliderChanged;

    //Register for MouseLeftButtonUp event
    slider1.MouseLeftButtonUp += BrightnessSliderChanged;
}
```

Implementation of `BrightnessSliderChanged` event handler will look like

```
private async void BrightnessSliderChanged(object sender,
System.Windows.Input.MouseButtonEventArgs e)
{
    //Undo last brightness to avoid applying brigthness recursively
    session.Undo();

    //subtract 1.0 from slider value to get brightness level
    //since our slider returns values ranging from 1.0 to 2.0
    double brightnessValue = slider1.Value - 1.0;

    session.AddFilter(FilterFactory.CreateBrightnessFilter(brightnessValue));

    await session.RenderToBitmapAsync(editedBitmap.AsBitmap());

    //Force the framework to redraw the Image.
    editedBitmap.Invalidate();
}
```

Implement Undo and Reset

Last applied effect to the image should be undone when user press Undo button. This can be very easily achieved by using `EditingSession` `Undo()` method. Make sure to check if undo is allowed by calling `CanUndo()` on session's instance which returns a boolean.

(C) Copyright Nokia 2014. All rights reserved.

```
private async void undo()
{
    if (session.CanUndo())
        {
        session.Undo();
        await session.RenderToBitmapAsync(editedBitmap.AsBitmap());
        //Force the framework to redraw the Image.
        editedBitmap.Invalidate();
    }
}
```

Same way there is a `UndoAll()` method present in `EditingSession` class to undo all applied effects.

```
private async void reset()
{
    if (session.CanUndo())
    {
        session.UndoAll();
        await session.RenderToBitmapAsync(editedBitmap.AsBitmap());
        //Force the framework to redraw the Image.
        editedBitmap.Invalidate();
    }
}
```

Saving edited image to gallery

After user is happy with the added effects he would like to save the image in gallery as a JPEG file. Which can be achieved using `EditingSession's RenderToJpegAsync()` and `MediaLibrary's SavePictureToCameraRoll()` methods. `RenderToJpegAsync()` returns **JPEG Buffer** data for the image being edited, which can be saved in an `IBuffer` instance. The method also allocates the memory required for the buffer and `SavePictureToCameraRoll()` writes JPEG Buffer into a JPEG file and saves it to camera roll (image gallery).

```
//get JPEG Buffer from the session
jpegOut = await editsession.RenderToJpegAsync();

MediaLibrary library = new MediaLibrary();
string filename = "SavedPicture_" + DateTime.Now.ToString("G");

// Save the image as a jpeg to the camera roll
Picture pic = library.SavePictureToCameraRoll(filename, jpegOut.ToArray());
```

Complete SaveImage method looks like

```
private async void SaveImage_Click(object sender,
System.Windows.Input.MouseButtonEventArgs e)
{
    if (session == null)
    {
        return;
    }

    jpegOut = await session.RenderToJpegAsync();
    // Save the image as a jpeg to the camera roll
```


(C) Copyright Nokia 2014. All rights reserved.

```
    MediaLibrary library = new MediaLibrary();
    string filename = "SavedPicture_" + DateTime.Now.ToString("G");

    // Save the image as a jpeg to the camera roll
    Picture pic = library.SavePictureToCameraRoll(filename, jpegOut.ToArray());
    MessageBox.Show("Image saved to the Camera Roll");
}
```

Summary

To wrap up, here is what this article covered,

1. Using speech recognition API's to take voice input from user.
2. Using PhotoPickerTask to pick image from gallary.
3. Using Nokia Imaging SDK APIs for applying various filters to images.
4. Saving edited image to camera roll/gallary.

TODO

I have implemented cartoon, antique, negative, sepia, blur, brightness, fog, gray scale, magic pen, milky, mirror, oily and sketch effects in the attached source code. You can add the remaining effects Nokia Imaging SDK provides and also your custom effects by mixing the default ones.

References

- Nokia Imaging SDK (Lumia Developers' Library)

(C) Copyright Nokia 2014. All rights reserved.