How to Build Card Reader

# Overview:

Card Reader is an application that demonstrates how to capture Media Elements from a live camera feed in order to create a new contact. This extracts the text and integrates it into the Windows People App using OCR.

This application tutorial is broken up into the following exercises:

1. Setup a Windows 10 Universal Application using Visual Studio Community edition 2015.
2. Use an embedded/external camera in order to capture Media using MediaCapture API.

* Xaml elements and Code Behind Events in order to find cameras
* Connect to selected camera
* Display the camera feed.

1. Extract text from an image using the Optical Character Recognition (OCR) library.
2. Integrate with the Windows People App.

The application allows the user to detect and select the connected Media devices. In order to begin the Media Capture exercise, there needs to be at least one (1) camera device connected or embedded.

**Note: In order to get correct text recognition, you will need to use a HD camera, embedded or external.**

The application utilises the MediaCapture API and OCR Library in order to capture pictures from a live camera feed and extract text from the images. It also uses the ContactManager API to integrate with the People App. This document will explain how to setup all the components required for you to build this yourself

# Getting Started:

In order to build and test this project you will need some experience with C# and XAML. You will need to use a Windows 10 machine running Visual Studio 2015 (Community Edition or greater) when building this app.

Create a new Universal Application

1. Open Visual Studio 2015 and select **File | New | Project** (CTRL + SHIFT + N)
2. This will open a “New Project” window. Navigate to Templates | Visual C# and
3. Choose Blank App (Universal Apps) and Name your application CardReader

Machine generated alternative text:
New Project 
Installed 
Templates 
visual 
Windows 
Wlndows 8 
Classic Desktop 
Android 
Cloud 
Extensibility 
Silver-light 
Workflow 
.NET Framework 45.2 
Sort by. Default 
Blank App (Universal Windows) 
Class Library (Universal Windows) 
Windows Runtime Component (Universal Windows) 
unit Test App (Universal Windows) 
Coded UI Test Project (Windows Phone) 
Coded UI Test Project (Windows) 
Click here to go online and find templates. 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Search Installed Templates (Ctrl*E) 
Type: Visual 
A project for a single-page Universal 
Windows Platform app that has no 
predefined controls or layout. 
Show telemetry in the Windows Dev 
Enable richer analytics with 
Application Insights 
Learn more 
Privacy sta tement 
Create directory for solution 
Add to source control 
t' Visual Basic 
Visual 
Visual C++ 
t' Online 
Name: 
Solution name: 
CardReade 
c:XGitxcardReader 
Card Reader 

1. Build and Run the new Application (F5), you should see a blank window on the screen. Close the Application.
2. Before you begin to write any code you need to setup some capabilities of the application. To change capabilities in a Windows 10 universal application open the Package.appxmanifest file which you can find in the Solution Explorer of Visual Studio. Go to the "Capabilities" tab and select Microphone, Webcam and Pictures Library.

Machine generated alternative text:
CardReader - Microsoft Visual Studio 
Edit View Project Build Debug Team 
Debug 
Tools 
Test 
Analyze Window Help 
Local Machine • E" 
Package.appxmanifest -E X 
Quick Launch (Ctrl+Q 
p 
Sign in 
The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties. 
Application 
Visual Assets 
Capabilities 
Declarations 
Content URIS 
Packaging 
Use this page to specify system features or devices that your app can use. 
Capabilities: 
All Joyn 
Blocked Chat Messages 
Bluetooth 
Chat Message Access 
Code Generation 
Enterprise Authentication 
Internet (Client) 
Internet (Client & Server) 
Location 
Microphone 
Music Library 
Objects 3D 
Phone Call 
Pictures Library 
Private Networks (Client & Server) 
Proximity 
Removable Storage 
Shared User Certificates 
User Account Information 
Videos Library 
VOIP Calling 
Webcam 
Output 
Descript ion: 
Provides the capability to add, change, or delete files in the Pictures Library for the local PC and 
HomeGroup PCs. 
More information 

# Exercise 1 - Basic Structure and using an Embedded/External Camera:

In this exercise you will be setting up a list of all attached and currently running cameras. You will then display a camera preview for a chosen camera.

1. Start by setting up a basic interface in MainPage.xaml. In the Solution Explorer double click on MainPage.xaml. Inside the Grid add the following XAML.  
   You will be adding to this XAML throughout the hands on lab.

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<StackPanel Orientation="Vertical" HorizontalAlignment="Center">

<TextBlock Text="Card Reader" Margin="4" FontSize="28" HorizontalAlignment="Center"/>

<!-- Camera Selection -->

<TextBlock Text="Select a Camera" Margin="4" HorizontalAlignment="Center"/>

<ListView x:Name="CameraSelectionList" Margin="4" Width="240" MaxHeight="200"

SelectionChanged="CameraSelectionList\_SelectionChanged"

HorizontalAlignment="Center"/>

</StackPanel>

</Grid>

1. Press F7 or hit the little arrow next to MainPage.xaml in the Solution Explorer to navigate to MainPage.xaml.cs. In here create a method handler for the camera selection changed event

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

}

private void CameraSelectionList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

}

}

1. Next implement the camera device list. Create a new private variable at the top of MainPage.xaml.cs to hold the list of camera devices. In the constructor register a handler for the Loaded event.

using Windows.Devices.Enumeration;

public sealed partial class MainPage : Page

{

private DeviceInformationCollection \_allVideoDevices;

public MainPage()

{

this.InitializeComponent();

this.Loaded += MainPage\_Loaded;

}

private void MainPage\_Loaded(object sender, RoutedEventArgs e)

{

}

private void CameraSelectionList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

}

}

Inside the Loaded event handler add the following code to populate the camera device list. You will need to make the Loaded method async as the code uses await on other async method calls.

using System.Linq;

private async void MainPage\_Loaded(object sender, RoutedEventArgs e)

{

// Get available devices for capturing media and list them

\_allVideoDevices = await DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);

if (\_allVideoDevices == null || !\_allVideoDevices.Any())

{

Debug.WriteLine("No devices found.");

return;

}

//add to device list

foreach (DeviceInformation camera in \_allVideoDevices)

{

if (CameraSelectionList.Items != null)

{

CameraSelectionList.Items.Add(camera.Name);

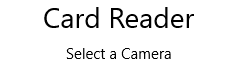
}

}

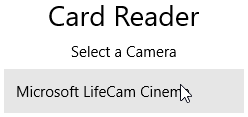
}

1. Build and Run the App. At the top under the text "Select a Camera" you will now see the available devices.

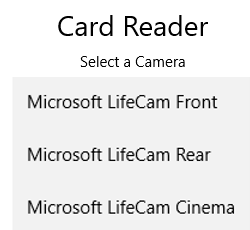
If you do not have a camera available, this list will be empty. Please refer to the images below



No camera device found



One camera device found



Multiple camera devices found

1. Implement the Camera Selection Changed event. This will have a trigger to output the camera feed to the "previewElement". This event is called "SelectionChanged".

You will need a new variable to store the selected camera. Add this at the top of MainPage.xaml.cs class

public sealed partial class MainPage : Page

{

private DeviceInformationCollection \_allVideoDevices;

private DeviceInformation \_desiredDevice;

In the Camera Selection Changed method you created earlier add the following code. This will get the selected device from the list and set it as the desired device. You may have noticed the commented out line that calls a method “StartDeviceAsync”. This does not exist but is the next thing you will need to implement.

private void CameraSelectionList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

string selectedCameraItem = e.AddedItems.FirstOrDefault().ToString();

foreach (DeviceInformation item in \_allVideoDevices)

{

if (string.Equals(item.Name, selectedCameraItem))

{

\_desiredDevice = item;

//await StartDeviceAsync();

}

}

}

1. To connect to the selected camera and start that cameras video feed you need to create a new asynchronous method called StartDeviceAsync. Create a new private variable at the top of the class that will contain the instance of a media capture device.

using Windows.Media.Capture;

public sealed partial class MainPage : Page

{

private DeviceInformationCollection \_allVideoDevices;

private DeviceInformation \_desiredDevice;

private MediaCapture \_mediaCapture;

At the bottom of the MainPage.xaml.cs class add a new region called Camera\_Methods. This will allow you to maintain cleaner code.

#region Camera\_Methods

#endregion

Inside the new region add the new StartDeviceAsync method. You will notice there are a few lines that are currently commented out. You will come back to these later on this lab.

private async Task StartDeviceAsync()

{

if (\_desiredDevice != null)

{

try

{

Debug.WriteLine("Starting device");

\_mediaCapture = new MediaCapture();

//initialize the selected device

await \_mediaCapture.InitializeAsync(

new MediaCaptureInitializationSettings

{

VideoDeviceId = \_desiredDevice.Id

});

// if you have a valid camera then enable the photo button and start the preview window display

if (\_mediaCapture.MediaCaptureSettings.VideoDeviceId != string.Empty

&& \_mediaCapture.MediaCaptureSettings.AudioDeviceId != string.Empty)

{

//TakePhotoButton.IsEnabled = true;

Debug.WriteLine("Device initialized successful");

//await StartPreviewAsync();

}

else

{

//TakePhotoButton.IsEnabled = false;

Debug.WriteLine("Error - No VideoDevice/AudioDevice Found");

}

}

catch (Exception exception)

{

Debug.WriteLine(exception);

}

}

}

1. The StartDeviceAsync method needs to be called inside the SelectionChanged method, this is where a camera is selected. In the SeletionChanged method uncomment the “await StartDeviceAsync()” line. As you are now awaiting the method you will need to make this method async.

private async void CameraSelectionList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

string selectedCameraItem = e.AddedItems.FirstOrDefault().ToString();

foreach (DeviceInformation item in \_allVideoDevices)

{

if (string.Equals(item.Name, selectedCameraItem))

{

\_desiredDevice = item;

await StartDeviceAsync();

}

}

}

1. To display the selected camera feed in a CaptureElement add the following XAML to MainPage.xaml underneath the CameraSelection

<!-- Camera Preview and Take Photo Button -->

<TextBlock Text="Camera Preview" Margin="4" HorizontalAlignment="Center"/>

<Border Background="LightSlateGray">

<CaptureElement x:Name="PreviewElement" Width="240" Height="240" />

</Border>

In the MainPage.xaml.cs file create a new method called StartPreviewAsync inside the Camera\_Methods region.

private async Task StartPreviewAsync()

{

try

{

Debug.WriteLine("Starting preview");

//set the source to the camera feed

PreviewElement.Source = \_mediaCapture;

await \_mediaCapture.StartPreviewAsync();

Debug.WriteLine("Start preview successful");

}

catch (Exception exception)

{

PreviewElement.Source = null;

Debug.WriteLine(exception);

}

}

1. Call the StartPreviewAsync method from the StartDeviceAsync method, uncomment the “await StartPreviewAsync()” line.

private async Task StartDeviceAsync()

{

if (\_desiredDevice != null)

{

try

{

Debug.WriteLine("Starting device");

\_mediaCapture = new MediaCapture();

//initialize the selected device

await \_mediaCapture.InitializeAsync(

new MediaCaptureInitializationSettings

{

VideoDeviceId = \_desiredDevice.Id

});

// if you have a valid camera then enable the photo button and start the preview window display

if (\_mediaCapture.MediaCaptureSettings.VideoDeviceId != string.Empty

&& \_mediaCapture.MediaCaptureSettings.AudioDeviceId != string.Empty)

{

//TakePhotoButton.IsEnabled = true;

Debug.WriteLine("Device initialized successful");

await StartPreviewAsync();

}

else

{

//TakePhotoButton.IsEnabled = false;

Debug.WriteLine("Error - No VideoDevice/AudioDevice Found");

}

}

catch (Exception exception)

{

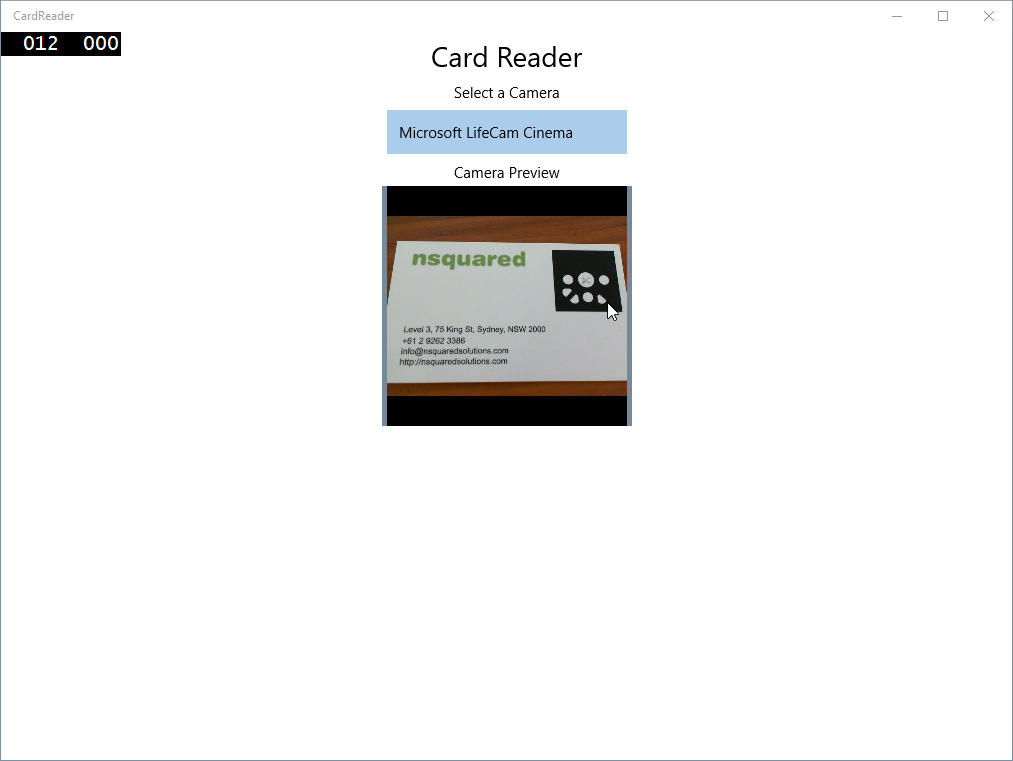
Debug.WriteLine(exception);

}

}

}

1. Build and run the application, select a camera and place a business card in front of the camera the output will show on the screen and should be similar to this:



1. To capture an image from the feed when your business card is in focus create a Button in MainPage.xaml named “Take Photo”. Also add an Image element to display the image. Put these underneath the camera preview area

<Button x:Name="TakePhotoButton" Click="TakePhotoButton\_ClickAsync"

IsEnabled="False" Margin="5" Width="240"

Content="Take Photo" />

<!-- Captured Image from Camera Feed -->

<TextBlock Text="Captured Images" HorizontalAlignment="Center"/>

<Border Background="LightSlateGray">

<Image x:Name="ImageElement" Width="240" Height="240"/>

</Border>

In MainPage.xaml.cs create the click event handler for the TakePhoto button. Add this underneath the camera selection method.

private async void TakePhotoButton\_ClickAsync(object sender, RoutedEventArgs e)

{

}

1. The Take Photo button click event handler should capture an image from the camera feed and place the image in the “ImageElement” you just created. To get an image from the camera feed you will also need to create a method called OpenImageAsBitmap.  
   Create a new region called Helper\_Methods, and inside add the new method called OpenImageAsBitmap which will get the image from the camera feed.

using Windows.Storage;

using Windows.Storage.Streams;

using Windows.UI.Xaml.Media.Imaging;

#region Helper\_Methods

//open an image file as a bitmapimage object

private async Task<BitmapImage> OpenImageAsBitmapAsync(StorageFile file)

{

IRandomAccessStreamWithContentType stream = await file.OpenReadAsync();

BitmapImage bmpImg = new BitmapImage();

bmpImg.SetSource(stream);

return bmpImg;

}

#endregion

1. To implement the take photo method add namespace references and a new private variable to store the image.

using System.Threading.Tasks;

using Windows.Media.MediaProperties;

public sealed partial class MainPage : Page

{

private DeviceInformationCollection \_allVideoDevices;

private DeviceInformation \_desiredDevice;

private MediaCapture \_mediaCapture;

private StorageFile \_photoFile;

Update the TakePhotoButton\_Click method to be async to enable it to create an image from the camera feed and save it in the photoFile using awaited async calls. You will call the OpenImageAsBitmap method in this step. This method will open the photo file and read it as a BitmapImage, then set it as the source of the image element in XAML.

private async void TakePhotoButton\_ClickAsync(object sender, RoutedEventArgs e)

{

try

{

Debug.WriteLine("Taking photo");

TakePhotoButton.IsEnabled = false;

//store the captured image

\_photoFile = await KnownFolders.PicturesLibrary.CreateFileAsync("capturedImage", CreationCollisionOption.ReplaceExisting);

Debug.WriteLine("Create photo file successful");

//create the properties to write

ImageEncodingProperties imageProperties = ImageEncodingProperties.CreateJpeg();

await \_mediaCapture.CapturePhotoToStorageFileAsync(imageProperties, \_photoFile);

TakePhotoButton.IsEnabled = true;

Debug.WriteLine("Photo taken");

//map the captured image as Bitmap image to the right column

ImageElement.Source = await OpenImageAsBitmapAsync(\_photoFile);

}

catch (Exception exception)

{

Debug.WriteLine(exception);

TakePhotoButton.IsEnabled = true;

}

}

1. You may have noticed that the Take Photo button is disabled by default. The Button will need to be switched between enabled and disabled to allow the processing of an image, and prevent the user from interrupting the start-up of the device feed.

In the StartDeviceAsync method uncomment the two lines that are setting the TakePhotoButton IsEnabled property.

private async Task StartDeviceAsync()

{

if (\_desiredDevice != null)

{

try

{

Debug.WriteLine("Starting device");

\_mediaCapture = new MediaCapture();

//initialize the selected device

await \_mediaCapture.InitializeAsync(

new MediaCaptureInitializationSettings

{

VideoDeviceId = \_desiredDevice.Id

});

// if you have a valid camera then enable the photo button and start the preview window display

if (\_mediaCapture.MediaCaptureSettings.VideoDeviceId != string.Empty

&& \_mediaCapture.MediaCaptureSettings.AudioDeviceId != string.Empty)

{

TakePhotoButton.IsEnabled = true;

Debug.WriteLine("Device initialized successful");

await StartPreviewAsync();

}

else

{

TakePhotoButton.IsEnabled = false;

Debug.WriteLine("Error - No VideoDevice/AudioDevice Found");

}

}

catch (Exception exception)

{

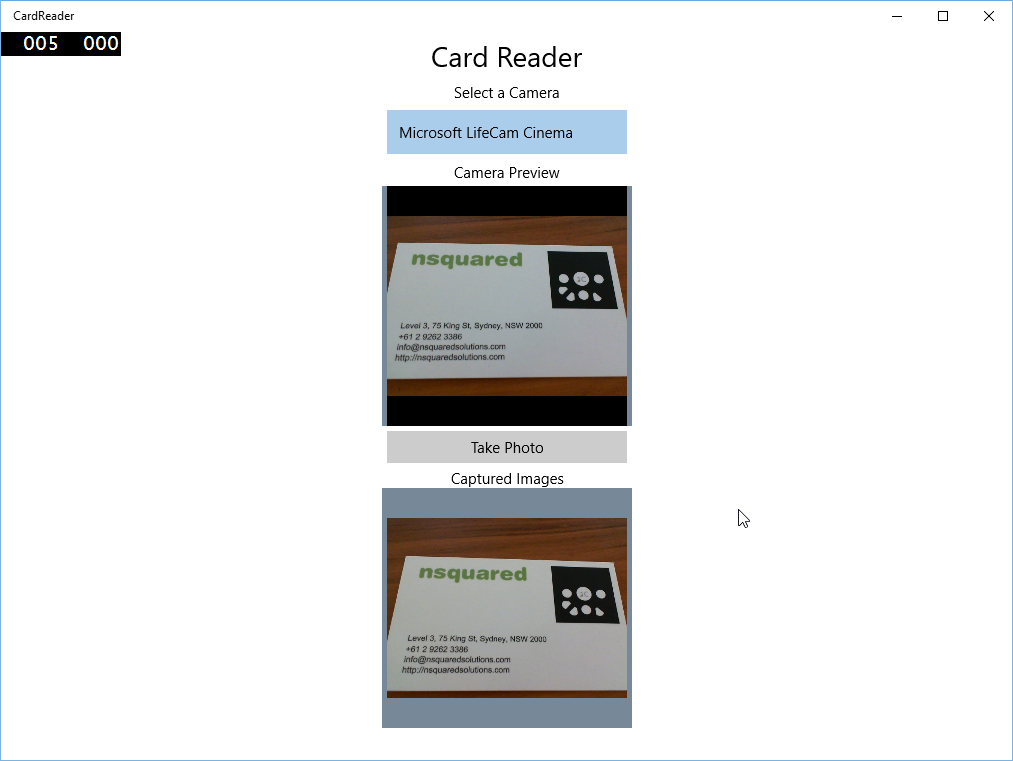
Debug.WriteLine(exception);

}

}

}

1. Build and run the application (F5), select a camera and then click “Take Photo” and this should be the output.



# Exercise 2 – Extract Text from an Image using OCR Library:

In this exercise you will be implementing text recognition. You will start by configuring an OCR engine, which will be used to retrieve an OCR result from an image.

**NOTE**: For OCR to work you will need a good camera, with the ability to auto focus. Many laptop webcams do not have short range focus that will work for OCR. An external camera such as the Microsoft LifeCam Cinema will work well.

1. In MainPage.xaml.cs add a reference to new namespaces and create a new private variable for the OCR engine.

using Windows.Globalization;

using Windows.Media.Ocr;

public sealed partial class MainPage : Page

{

private DeviceInformationCollection \_allVideoDevices;

private DeviceInformation \_desiredDevice;

private MediaCapture \_mediaCapture;

private StorageFile \_photoFile;

private OcrEngine \_ocrEngine;

public MainPage()

{

this.InitializeComponent();

this.Loaded += MainPage\_Loaded;

// Init OCR engine with English language.

\_ocrEngine = OcrEngine.TryCreateFromLanguage(new Language("en"));

}

1. In MainPage.xaml create a button to trigger getting an OCR result, and a TextBlock to display an error. Open the MainPage.xaml and add the following XAML underneath the ImageElement.

<!-- OCR Get Details Button and Results TextBlock -->

<Button x:Name="GetDetailsButton" Click="GetDetailsButton\_ClickAsync"

Margin="4" Width="240" Content="Get Details" />

<TextBlock x:Name="GetDetailsErrorTextBlock" Foreground="Red" HorizontalAlignment="Center"/>

In MainPage.xaml.cs at the bottom of the class add the following method to handle the button click event of the GetDetailsButton.

private void GetDetailsButton\_ClickAsync(object sender, RoutedEventArgs e)

{

}

In MainPage.xaml.cs you need to implement the code for the GetDetailsButton click event. The GetDetails method will need to be changed to be async as it will be awaiting the OCR result. This event creates a BitmapDecoder which returns a string. You will also create a method called ApplyPatternMatching which you will put into a new region that contains OCR specific methods.

Add a new region called “OCR\_Methods”. Inside this new region add the ApplyPatternMatching method.

#region OCR\_Methods

private void ApplyPatternMatching(OcrResult ocrResult)

{

}

#endregion

1. Implement the code in the GetDetailsButton\_ClickAsync method using the ApplyPatternMatching method. An error will be displayed to the user if the text is not recognizable.

using Windows.Graphics.Imaging;

private async void GetDetailsButton\_ClickAsync(object sender, RoutedEventArgs e)

{

GetDetailsErrorTextBlock.Text = string.Empty;

if (\_photoFile != null)

{

using (IRandomAccessStream stream = await \_photoFile.OpenAsync(FileAccessMode.Read))

{

// Create image decoder.

BitmapDecoder decoder = await BitmapDecoder.CreateAsync(stream);

// Load bitmap.

SoftwareBitmap bitmap = await decoder.GetSoftwareBitmapAsync();

// Extract text from image.

OcrResult result = await \_ocrEngine.RecognizeAsync(bitmap);

if (string.IsNullOrEmpty(result.Text))

{

GetDetailsErrorTextBlock.Text = "Text not Recognizable try again!";

Debug.WriteLine("The Text is not recognizable.");

}

else

{

Debug.WriteLine(result.Text);

//extract the details

ApplyPatternMatching(result);

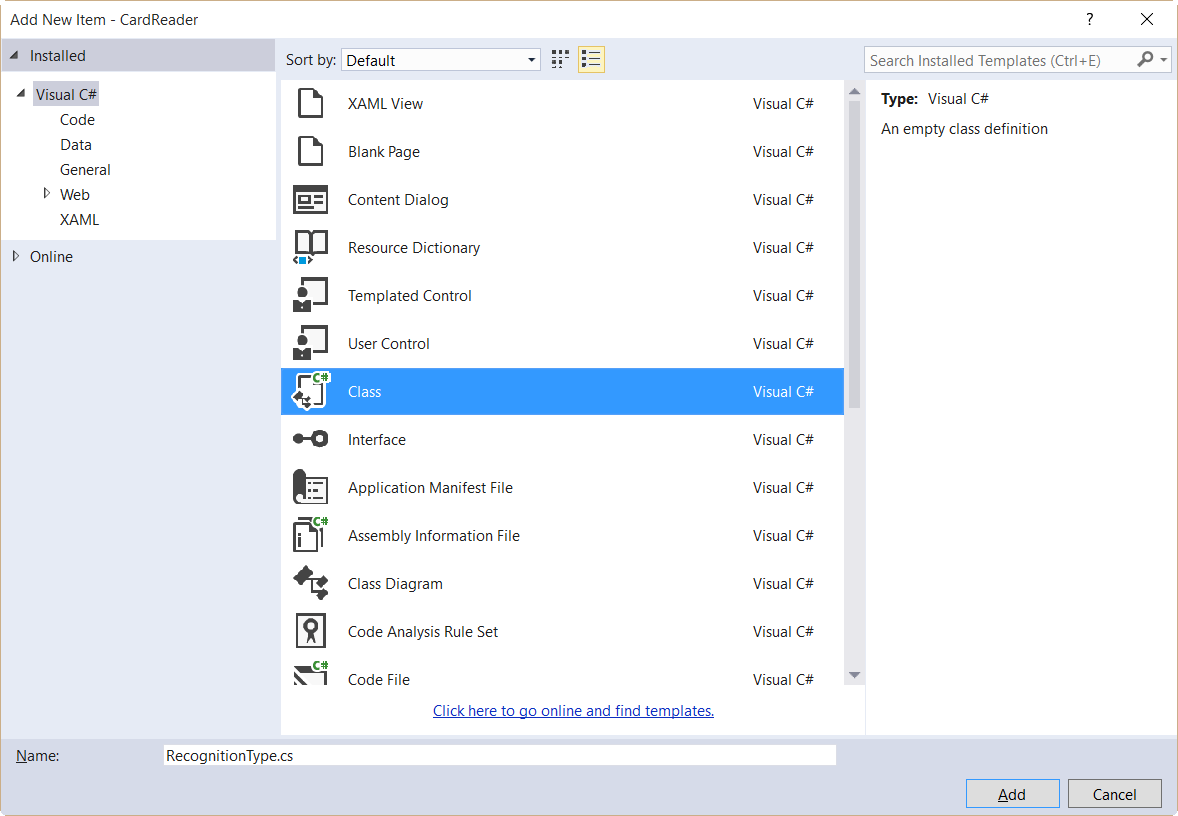
}

}

}

}

1. Create a new class that will contain an enum of Recognition types for the OCR engine. In Solution Explorer, right click on the project, select **Add** | **Class** from the popup menu. Name the class RecognitionType.cs and click Add.



1. Change the class definition to be public enum and add in the enum types.

namespace CardReader

{

public enum RecognitionType

{

Other,

Email,

WebPage,

Name,

Number

}

}

1. Create another class with methods to find patterns in the words extracted by the OCR decoder. In Solution Explorer, right click on the project, select **Add** | **Class** from the popup menu. Name the class CardRecognizer.cs and click Add.

Machine generated alternative text:
Add New Item - CardReader 
Installed 
Visual C' 
General 
Web 
D Online 
Card Recogniser.cs 
Name: 
Sort by: Default 
XAML View 
Blank Page 
Content Dialog 
Resource Dictionary 
Temp la ted Control 
user Control 
Class 
Interface 
Application Manifest File 
Assembly Information File 
Class Diagram 
Code Analysis Rule Set 
Click here to qo online and find templates. 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Visual C# 
Search Installed Templates (Ctrl*E) 
Type: Visual 
An empty class definition 
Cancel 

1. Change the CardRecognizer class definition so that it is a public static class

namespace CardReader

{

public static class CardRecognizer

{

}

}

* 1. In the Card Recognizer class add a new private static Dictionary variable named expressions, this dictionary will map the types with the corresponding Regex expressions.

Regex represents immutable regular expressions which you can browse or create your own if you follow the link included.

When you create this new Dictionary you will need to assign a regex value to the RecognitionType of Email, WebPage and Name and Number.

*Note that you are using Patterns here, such as email, webpage, name and numbers. You are able to create your own patterns or follow the already existing ones.*

public static class CardRecognizer

{

public static Dictionary<RecognitionType, string> expressions = new Dictionary<RecognitionType, string>()

{

// regex taken from MSDN: http://msdn.microsoft.com/en-us/library/01escwtf(v=vs.110).aspx

{

RecognitionType.Email,

@"^(?("")("".+?(?<!\\)""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\\*\+/=\?\^`\{\}\|~\w])\*)(?<=[0-9a-z])@))"+

@"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]\*[0-9a-z]\*\.)+[a-z0-9][\-a-z0-9]{0,22}[a-z0-9]))$"

},

// regex taken from regex lib: http://regexlib.com/REDetails.aspx?regexp\_id=296

{

RecognitionType.WebPage,

@"^(https?:\/\/)?([\w\d-\_]+)\.([\w\d-\_\.]+)\/?\??([^#\n\r]\*)?#?([^\n\r]\*)"

},

// regex taken from regex lib: http://regexlib.com/REDetails.aspx?regexp\_id=247

{

RecognitionType.Name,

@"^([ \u00c0-\u01ffa-zA-Z'])+$"

},

{

RecognitionType.Number,

@"^\+?(\d[\d-. ]+)?(\([\d-. ]+\))?[\d-. ]+\d$"

},

};

}

1. In the CardRecognizer class add a new static method called Recognize. This method will match the RecognitionType to the supplied string and returns the type.

using System.Text.RegularExpressions;

public static RecognitionType Recognize(string businessCardText)

{

RecognitionType type = RecognitionType.Other;

//iterate through each type to try and find a match.

// once a match is found stop and return the type

foreach (KeyValuePair<RecognitionType, string> expression in expressions)

{

if (Regex.IsMatch(businessCardText, expression.Value))

{

type = expression.Key;

break;

}

}

return (type);

}

1. Build and run the application. Select a camera and take a photo of a business card or something that has recognizable text and click “Get Details”. The OCR result will be output to the output window. The result depends on the quality of image. No pattern matching is being used at this point.

In the next exercise you will use pattern matching to try and determine the components of a business card and create a new contact card for a person in the Windows People application.

# Exercise 3 – How to Implement and use Windows People App:

With the Recognizer Class, you can apply search patterns to the OCR-Decoded string. This exercise will create a new contact instance for the Windows People Application.

1. Return to the "ApplyPatternMatching" method in MainPage.xaml.cs and add the below code. This code will create a new Contact and set the contact picture to the image that you took from the camera feed.

*Note: Here you are using the OCR string result as input from previous step.*

using Windows.ApplicationModel.Contacts;

private void ApplyPatternMatching(OcrResult ocrResult)

{

Contact contact = new Contact();

//set the picture

contact.SourceDisplayPicture = \_photoFile;

//TODO: Loop through OCR Result lines and words

//TODO: open the selection

}

1. You need to pass the OCR result to a method that will iterate through all the lines and words in that result. This method will take two parameters. The first parameter will be the OCR Result, and the second will be an action that will be called when a word is found.

Add this method in the OCR\_Methods region below the "ApplyPatternMatching" method.

private void RepeatForOcrWords(OcrResult ocrResult, Action<OcrResult, OcrWord> repeater)

{

if (ocrResult.Lines != null)

{

foreach (var line in ocrResult.Lines)

{

foreach (var word in line.Words)

{

repeater(ocrResult, word);

}

}

}

}

1. You now have a way to loop through the OCR Result. You will need to call this method from the "ApplyPatternMatching" method and pass through the correct parameters. Inside the "ApplyPatternMatching" method replace the //TODO: Loop through… with a call to RepeatForOcrWords method. For simplicity you will make this a delegate call so you can implement the action here at a later date.

private void ApplyPatternMatching(OcrResult ocrResult)

{

Contact contact = new Contact();

//set the picture

contact.SourceDisplayPicture = photoFile;

// this method uses an action that will run as a 'callback' for the method

// more info here https://msdn.microsoft.com/en-us/library/018hxwa8(v=vs.110).aspx

this.RepeatForOcrWords(ocrResult, (result, word) =>

{

//TODO: RepeatForOcrWords actions goes here

});

//TODO: open the selection

}

You still need to implement an action for when a word is found. You are going to do this with a switch statement. The cases in your switch statement will reflect the RecognitionTypes created in the CardRecogniser class.

Replace the //TODO: RepeatForOcrWords actions goes here with the below snippet.

private void ApplyPatternMatching(OcrResult ocrResult)

{

Contact contact = new Contact();

//set the picture

contact.SourceDisplayPicture = \_photoFile;

// this method uses an action that will run as a 'callback' for the method

// more info here https://msdn.microsoft.com/en-us/library/018hxwa8(v=vs.110).aspx

this.RepeatForOcrWords(ocrResult, (result, word) =>

{

switch (CardRecognizer.Recognize(word.Text))

{

case RecognitionType.Other:

break;

case RecognitionType.Email:

contact.Emails.Add(new ContactEmail() { Address = word.Text });

break;

case RecognitionType.Name:

contact.FirstName = word.Text;

break;

case RecognitionType.Number:

contact.Phones.Add(new ContactPhone() { Number = word.Text });

break;

case RecognitionType.WebPage:

try

{

contact.Websites.Add(new ContactWebsite() { Uri = new Uri(word.Text) });

}

catch (Exception)

{

Debug.WriteLine("OCR Result cannot be converted to a URI");

}

break;

default:

break;

}

});

//TODO: open the selection

}

With the switch statement the words found in the OCR Result are sorted and then added to the contact appropriately.

1. After you have iterated over the recognized text you need to add one more validation check. For the Windows ContactManager.ShowContactCard method to work, the contact must have at least a phone or email. Add this code next to check these properties

private void ApplyPatternMatching(OcrResult ocrResult)

{

Contact contact = new Contact();

//set the picture

contact.SourceDisplayPicture = \_photoFile;

this.RepeatForOcrWords(ocrResult, (result, word) =>

{

switch (CardRecogniser.Recognise(word.Text))

{

case RecognitionType.Other:

break;

case RecognitionType.Email:

contact.Emails.Add(new ContactEmail() { Address = word.Text });

break;

case RecognitionType.Name:

contact.FirstName = word.Text;

break;

case RecognitionType.Number:

contact.Phones.Add(new ContactPhone() { Number = word.Text });

break;

case RecognitionType.WebPage:

contact.Websites.Add(new ContactWebsite() { Uri = new Uri(word.Text) });

break;

default:

break;

}

});

if (!contact.Phones.Any()) //contact must have either a phone or email when calling ContactManager.ShowContactCard.

{

if (!contact.Emails.Any())

{

Debug.WriteLine("Contact must have phone or email info.");

return;

}

}

//TODO: open the selection

}

1. Finally you need to open the contact in the Windows People Application. Create a new method called GetElementRect in the HelpersMethods region. This method helps to define the point where to open the Contact Manager

//get the bounding rect of an element relative to 0,0

private static Rect GetElementRect(FrameworkElement element)

{

//get the element point to open the window at the correct point

GeneralTransform transform = element.TransformToVisual(null);

Point point = transform.TransformPoint(new Point());

return new Rect(point, new Size(element.ActualWidth, element.ActualHeight));

}

In the "ApplyPatternMatching" method, replace the //TODO: open the selection with the below code snippet.

This code will call the GetElementRect to get the correct location to open the contact manager and then call ShowContactCard to display the contact.

private void ApplyPatternMatching(OcrResult ocrResult)

{

Contact contact = new Contact();

//set the picture

contact.SourceDisplayPicture = \_photoFile;

// this method uses an action that will run as a 'callback' for the method

// more info here https://msdn.microsoft.com/en-us/library/018hxwa8(v=vs.110).aspx

this.RepeatForOcrWords(ocrResult, (result, word) =>

{

switch (CardRecogniser.Recognise(word.Text))

{

case RecognitionType.Other:

break;

case RecognitionType.Email:

contact.Emails.Add(new ContactEmail() { Address = word.Text });

break;

case RecognitionType.Name:

contact.FirstName = word.Text;

break;

case RecognitionType.Number:

contact.Phones.Add(new ContactPhone() { Number = word.Text });

break;

case RecognitionType.WebPage:

contact.Websites.Add(new ContactWebsite() { Uri = new Uri(word.Text) });

break;

default:

break;

}

});

if (!contact.Phones.Any()) //contact must have either a phone or email when calling ContactManager.ShowContactCard.

{

if (!contact.Emails.Any())

{

Debug.WriteLine("Contact must have phone or email info.");

return;

}

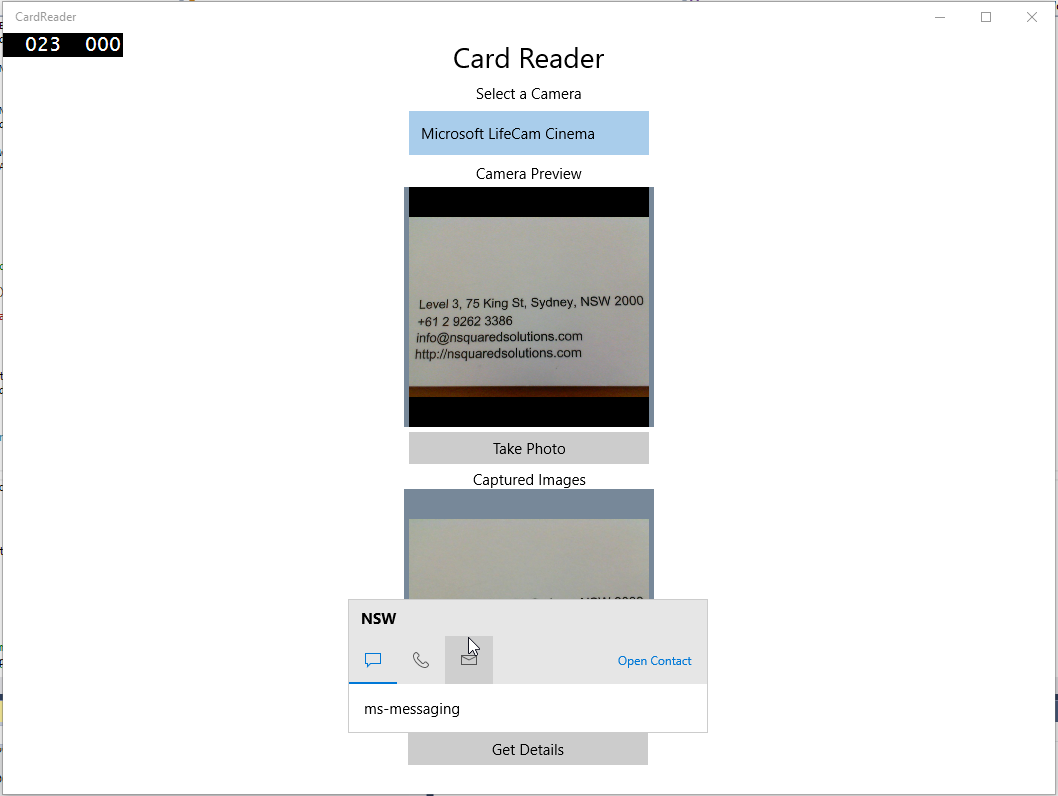
}

Rect rect = GetElementRect(GetDetailsButton);

ContactManager.ShowContactCard(contact, rect, Windows.UI.Popups.Placement.Default);

}

1. Build and Run the application (F5). Select a camera and take a photo of a business card. Click get details. You will see a popup window showing the contact card. You can click on "Open Contact", here you will have the ability to save the contact, or edit details then save the contact.



If you find that the app crashes in the ApplyPatternMatching method, it is likely due to the regex failing to correctly identify a component of the business card for the Contact.

In the next exercise you will explore how to extend the regex expressions to cover additional scenarios.

# Exercise 4 - Additional Exercises:

1. While the "ApplyPatternMatching" method may work well for very simple data, using OCR can require some complicated regex to validate words. With OCR a phone number that is separated by spaces will be returned in the OCR result as separate strings. This will break your current code for Phone numbers.

In order to get the correct phone number into the contact information more code needs to be implemented.

In the CardRecognizer class and in RecognitionType you may have noticed PhoneNumber. However you did not implement any Regex code for this, let's do that now.

**Please note:** The PhoneNumber must be listed in the enum before the Number type. A number will always validate as a number but not always as a phone number, you need to check if it is a phone number first, and then if it is a number.

public enum RecognitionType

{

Other,

Email,

WebPage,

Name,

PhoneNumber,

Number,

}

static Dictionary<RecognitionType, string> expressions = new Dictionary<RecognitionType, string>()

{

// regex taken from MSDN: http://msdn.microsoft.com/en-us/library/01escwtf(v=vs.110).aspx

{

RecognitionType.Email,

@"^(?("")("".+?(?<!\\)""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\\*\+/=\?\^`\{\}\|~\w])\*)(?<=[0-9a-z])@))"+

@"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]\*[0-9a-z]\*\.)+[a-z0-9][\-a-z0-9]{0,22}[a-z0-9]))$"

},

// regex taken from regex lib: http://regexlib.com/REDetails.aspx?regexp\_id=296

{

RecognitionType.WebPage,

@"^(https?:\/\/)?([\w\d-\_]+)\.([\w\d-\_\.]+)\/?\??([^#\n\r]\*)?#?([^\n\r]\*)"

},

// regex taken from regex lib: http://regexlib.com/REDetails.aspx?regexp\_id=247

{

RecognitionType.Name,

@"^([ \u00c0-\u01ffa-zA-Z'])+$"

},

{

RecognitionType.PhoneNumber,

@"(((\+[0-9]{1,2}|00[0-9]{1,2})[-\ .]?)?)(\d[-\ .]?){5,15}"

},

{

RecognitionType.Number,

@"^\+?(\d[\d-. ]+)?(\([\d-. ]+\))?[\d-. ]+\d$"

},

};

1. Update the switch statement inside the "ApplyPatternMatching" method in MainPage.xaml.cs. As OCR splits phone numbers that contain spaces you will need to put these separate string values back together. You will do this by declaring a private string variable phoneNumber at the top of MainPage class.

When the OCR result returns and is of type Number, that number should be appended to the phoneNumber variable. As you append numbers you can check if the phoneNumber value validates as a phone number. If it does, then add the phone number to the contact details.

 public sealed partial class MainPage : Page

{

private DeviceInformationCollection \_allVideoDevices;

private DeviceInformation \_desiredDevice;

private MediaCapture \_mediaCapture;

private StorageFile \_photoFile;

private OcrEngine \_ocrEngine;

private string \_phoneNumber = string.Empty;

this.RepeatForOcrWords(ocrResult, (result, word) =>

{

switch (CardRecogniser.Recognise(word.Text))

{

case RecognitionType.Other:

break;

case RecognitionType.Email:

contact.Emails.Add(new ContactEmail() { Address = word.Text });

break;

case RecognitionType.Name:

contact.FirstName = word.Text;

break;

case RecognitionType.Number:

//NOTE: Phone numbers are not as easy to validate because OCR results splits the numbers if they contain spaces.

\_phoneNumber += word.Text;

RecognitionType type = CardRecogniser.Recognise(\_phoneNumber);

if (type == RecognitionType.PhoneNumber)

{

contact.Phones.Add(new ContactPhone() { Number = \_phoneNumber });

}

break;

case RecognitionType.WebPage:

contact.Websites.Add(new ContactWebsite() { Uri = new Uri(word.Text) });

break;

default:

break;

}

});

1. More logic is needed to determine if numbers are concurrent, or if they are split by words. To achieve this simply use a boolean to determine if the OCR result is a number or not.

By default the boolean will be false each time the switch statement is run. If a number is found set the value to true and check if it is a phone number. At the end of the switch statement if the boolean is still false then a number has not been found, so you should clear the phoneNumber value and start again.

this.RepeatForOcrWords(ocrResult, (result, word) =>

{

bool isNumber = false;

switch (CardRecogniser.Recognise(word.Text))

{

case RecognitionType.Other:

break;

case RecognitionType.Email:

contact.Emails.Add(new ContactEmail() { Address = word.Text });

break;

case RecognitionType.Name:

contact.FirstName = word.Text;

break;

case RecognitionType.Number:

isNumber = true;

//NOTE: Phone numbers are not as easy to validate because OCR results splits the numbers if they contain spaces.

\_phoneNumber += word.Text;

var type = CardRecogniser.Recognise(\_phoneNumber);

if (type == RecognitionType.PhoneNumber)

{

contact.Phones.Add(new ContactPhone() { Number = \_phoneNumber });

}

break;

case RecognitionType.WebPage:

contact.Websites.Add(new ContactWebsite() { Uri = new Uri(word.Text) });

break;

default:

break;

}

//Encounted a word or a value other than a number.

//If we havent validated as a phone number at this stage it is clearly not a phone number so clear the string

if (!isNumber)

{

\_phoneNumber = string.Empty;

}

});

* 1. This is a quick solution to finding Phone numbers in OCR Results, see if you can come up with a better solution or try and resolve a street address from separate OCR results.

In this exercise you learned how to use a webcam to capture an image, recognize text in the image, and then use the text to create a contact in the Windows People application.

In order to make this into a commercial application you would need to work on the text recognition categorization for a contact.