

Enabling wallet payment by face recognition

This article shows how you can use face recognition to authorize a wallet payment. This is the second of a series of articles based on the Digital Signal Processing and focused on face recognition.

Introduction



A new interesting feature of Windows Phone 8 is certainly the Wallet and Deals APIs which allows users to do the following:

- Collect coupons, credit cards, memberships, loyalty cards, and more in one place.
- Manage the payment instruments that they use in the app and music store.
- Link items in the Wallet to apps on their phone.
- Make contact-less transactions, using Near-Field Communication (NFC), in some markets.

The Wallet API offers full programmatic access to the Wallet. It allows you to create, read, update, and delete Wallet items to implement Deals Or Payments.


Deals are useful to manage coupons, loyalty cards, Payment instrument manage a balance on an account maintained by your back-end.

The sample application we will create first acquires the face sample to compare and store it into a file. Through `RootFrame.UriMapper` rewriting when the app is called from Wallet before to process payment is required to compare your face with the one saved into the system. If there is a match payment process can continue else is stopped.

Required capabilities

In order to use Wallet API in Windows Phone 8 you need to specify in the **WMAppManifest.xml** the following required capabilities otherwise your app might not work correctly or it might fail the process of submission to the Store.

Capability	API that requires this capability
ID_CAP_WALLET	Required for all Wallet API, which is anything in <code>Microsoft.Phone.Wallet</code> Or <code>Microsoft.Phone.SecureElement</code> .
ID_CAP_WALLET_PAYMENTINSTRUMENTS	Required for <code>PaymentInstrument</code> and <code>OnlinePaymentInstrument</code> .
ID_CAP_WALLET_SECUREELEMENT	Required for <code>SecureElementSession</code> , <code>SecureElementChannel</code> and <code>SecureElementReader</code> .

 **Warning:** To deploy or submit an app that uses `ID_CAP_WALLET_SECUREELEMENT` or `ID_CAP_WALLET_PAYMENTINSTRUMENTS`, you must request special permissions and have that permission applied to your developer account. As reported by [\[documentation\]](#) contact the [\[developer support\]](#)

Creating the UI

The following code create the object to display the camera video flow and the snap button to save the captured image, compute or save it. Besides **MainPage.xaml** the same code is needed for the secondary screen called **Wallet.xaml** we need to create to be opened when the application is called from the wallet.

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.Background>
        <VideoBrush x:Name="viewfinderBrush">
            <VideoBrush.RelativeTransform>
                <CompositeTransform CenterY="0.5" CenterX="0.5" Rotation="-90"/>
            </VideoBrush.RelativeTransform>
        </VideoBrush>
    </Grid.Background>
</Grid>
<Button Content="Snap Picture" Click="SaveImage" Margin="0,624,0,0" Grid.Row="1" />
<!--Application progress bar.-->
<ProgressBar Height="10" Name="progressBar" Width="460" Visibility="Collapsed"
    Margin="12,365,8,286" Grid.Row="1" IsIndeterminate="True" />
```

Remapping URI

In Windows Phone 8 we can use URI associations to automatically launch our app from another app launches with a specific file type or URI scheme. When launched, a deep link URI is used to send the file (a reference to the file) or URI to your app. We use URI scheme feature in order to open the `wallet.xaml` when the app is called from the Wallet, `MainPage.xaml` else. We need to choose a parameter to intercept, `PaymentInstrument` in our sample, but you can choose your own name. So we need also to create a class that inherits `UriMapperBase`. The object created from this class will be delegated to manage page switching.

Into **App.xaml.cs** add the following code:

```
class MyAppUriMapper : UriMapperBase
{
```

```

    public override Uri MapUri(Uri uri)
    {
        string tempUri = uri.ToString();
        // Check if the URI contains the parameter we choosed.
        // If yes return a new URI corresponding to the .xaml we want to open when
the App is called from another App
        // else the uri is unchanged
        if (tempUri.Contains("PaymentInstrument"))
        {
            return new Uri("/Wallet.xaml", UriKind.Relative);
        }
        else
        {
            return uri;
        }
    }
}

public partial class App : Application
{
    .....

    public App()
    {
        .....
        // Override the application UriMapper
        RootFrame.UriMapper = new MyAppUriMapper();
    }
}

```

Camera

In common with **Wallet.xaml**.

```

using Microsoft.Devices; // Needed for PhotoCamera
using Microsoft.Xna.Framework.Media;

using System.Windows.Media.Imaging;
using Microsoft.Phone; // Needed for PictureDecoder
using System.IO;

```

```

public partial class MainPage : PhoneApplicationPage
{
    PhotoCamera cam;
    MediaLibrary library = new MediaLibrary();

    public MainPage()
    {
        InitializeComponent();
        this.Loaded += Page_Loaded;
    }

    void Page_Loaded(object sender, RoutedEventArgs e)
    {
        if (PhotoCamera.IsCameraTypeSupported(CameraType.FrontFacing))
        {
            cam = new Microsoft.Devices.PhotoCamera(CameraType.FrontFacing);
            cam.CaptureImageAvailable += cam_CaptureImageAvailable;
            viewfinderBrush.SetSource(cam);
        } else
        if (PhotoCamera.IsCameraTypeSupported(CameraType.Primary))
        {
            cam = new Microsoft.Devices.PhotoCamera(CameraType.Primary);

            cam.CaptureImageAvailable += cam_CaptureImageAvailable;
            viewfinderBrush.SetSource(cam);
        }
    }
}

```

```

    }

    private void SaveImage(object sender, RoutedEventArgs e)
    {
        cam.CaptureImage();
    }
}

```

Face Recognition Support

For a deeper understanding of face recognition please refer to [Face Recognition using 2D Fast Fourier Transform](#).

- Download **DSP.cs** file from [File:FaceWallet.zip](#) and add it into your project. **DSP.cs** provides a namespace called **dsp** and a class **FourierTransform** containing a set of functions to compute the FFT.

Include the namespace DSP:

```
using DSP;
```

Add the following lines to **MainPage.xaml**, in common with **Wallet.xaml**.

```

public partial class MainPage : PhoneApplicationPage
{
    PhotoCamera cam;
    MediaLibrary library = new MediaLibrary();

    public static WriteableBitmap CapturedImage;
    private static int W = 256;
    private static int H = 256;
    private static int matchSamples = 25;

    private double[] compareSignal = new Double[matchSamples];

    private Double[] pRealIn = new Double[W * H];
    private Double[] pImagIn = new Double[W * H];
    private Double[] pRealOut = new Double[W * H];
    private Double[] pImagOut = new Double[W * H];
    private Double[] pRealOut2 = new Double[W * H];
    private Double[] pImagOut2 = new Double[W * H];

    private const string faceFileName = "face.csv";
    .....
}

```

```

void cam_CaptureImageAvailable(object sender, ContentReadyEventArgs e)
{
    //Take JPEG stream and decode into a WriteableBitmap object
    Deployment.Current.Dispatcher.BeginInvoke(delegate()
    {
        MainPage.CapturedImage = PictureDecoder.DecodeJpeg(e.ImageStream, W, H);

        //Collapse visibility on the progress bar once writeable bitmap is
visible.
        progressBar.Visibility = Visibility.Collapsed;

        int[] pixel = MainPage.CapturedImage.Pixels;

        int color = 0;
        for (int y = 0; y < MainPage.CapturedImage.PixelHeight; y++)
        {
            for (int x = 0; x < MainPage.CapturedImage.PixelWidth; x++)
            {
                color = MainPage.CapturedImage.Pixels[x + (y *
MainPage.CapturedImage.PixelWidth)];
                pRealIn[x + (y * MainPage.CapturedImage.PixelWidth)] =
DSP.Utilities.ColorToGray(color) & 0xFF;
            }
        }
    }
}

```

```

        Double[] signal;

        System.Diagnostics.Debug.WriteLine("Using Fourier");
        DSP.FourierTransform.Compute2D((uint)W, (uint)H, ref pRealIn, null, ref
pRealOut, ref pImagOut, false);
        signal = DSP.Utilities.triangularExtraction(ref pRealOut, (uint)W,
(uint)H, (uint)matchSamples, 0);

        DSP.Utilities.saveSignal(ref signal, faceFileName);

        Wallet_CreateNewPaymentInstrument();

    });
}

```

Wallet:PaymentInstruments

The following code is needed to register our application into the wallet in order to be able to call it from the Wallet. First we need to create a [PaymentInstruments](#) object whose methods are intuitive enough. Created the Item we use [AddWalletItemTask](#) allowing our application to launch the Wallet application.

The provided Wallet item is displayed to the user and the user can choose to add the item to his or her Wallet.

Include the following namespaces:

```

using System.Windows.Media.Imaging;
using Windows.System;
using Microsoft.Phone.Tasks;
using Microsoft.Phone.Wallet;
using System.Windows.Media;

```

```

private void Wallet_CreateNewPaymentInstrument()
{
    // Check if the object is already created
    if ((Microsoft.Phone.Wallet.Wallet.FindItem("facedetectionwallet") as
PaymentInstrument) == null)
    {
        // Create the PaymentInstruments object
        var item = new PaymentInstrument()
        {
            IssuerName = "sebastiano.galazzo@gmail.com",
            CustomerName = "Sebastiano Galazzo",
            ExpirationDate = DateTime.Now.AddDays(1),
            PaymentInstrumentKinds = PaymentInstrumentKinds.Debit,
            DisplayName = "Face Recognition Wallet",
            Logo99x99 = GetBitmapSource("99x99.png"),
            Logo336x336 = GetBitmapSource("336x336.png"),
            Logo159x159 = GetBitmapSource("159x159.png"),
            DisplayAvailableBalance = "50€?",
            DisplayBalance = "100€?",
            DisplayCreditLimit = "1000€",
            DisplayAvailableCredit = "1000€",
            BackgroundColor = Color.FromArgb(255, 70, 150, 250),
            Nickname = "Bino",
            Message = "Face recognition based wallet.",
            MemberSince = DateTime.Now.AddYears(-1),
            AccountNumber = "12345679",
            NavigationUri = new Uri("/mainpage.xaml?wallet=PaymentInstrument",
UriKind.Relative)
        };
        // Assign an ID we use to avoid duplication of our App into the Wallet
        item.UpdateId("facedetectionwallet");

        // Create the task able to register into the Wallet the Item created
        above
        AddWalletItemTask task = new AddWalletItemTask()
        {

```

```

        Item = item
    };

    // The code to execute when the registration task is completed
    task.Completed += (s, args) =>
    {
        MessageBox.Show(args.TaskResult.ToString());

        task.Show();
    }
    else {
        MessageBox.Show("Face updated!");
    }
    Launcher.LaunchUriAsync(new Uri("wallet://", UriKind.RelativeOrAbsolute));
}

private BitmapSource GetBitmapSource(string url)
{
    var bmp = new BitmapImage();
    bmp.SetSource(Application.GetResourceStream(new Uri(url,
        UriKind.Relative)).Stream);
    return bmp;
}

```

Wallet.xaml

This page is opened when the application is called from Wallet. Most of the code is in common with **MainPage.xaml** except `cam_CaptureImageAvailable` code and the lack of `Wallet_CreateNewPaymentInstrument` that is needed to register application into the wallet but unnecessary here.

The sample code stops after the recognition process and it's limited to display just a `MessageBox`. This is because from this point onwards you put your own implementation based on your needs. In the same way when recognition fails you can choose to simply stop the process or start another action. An idea could be for example to send an alert email after three attempts attaching the photo of the face is trying to access, assuming is an unauthorized person that wants to access to your data.

```

using DSP;

namespace FaceWallet
{
    public partial class Wallet : PhoneApplicationPage
    {
        PhotoCamera cam;
        MediaLibrary library = new MediaLibrary();

        public static WriteableBitmap CapturedImage;
        private static int W = 256;
        private static int H = 256;
        private static int matchSamples = 25;

        private double[] compareSignal = new Double[matchSamples];

        private Double[] pRealIn = new Double[W * H];
        private Double[] pImagIn = new Double[W * H];
        private Double[] pRealOut = new Double[W * H];
        private Double[] pImagOut = new Double[W * H];
        private Double[] pRealOut2 = new Double[W * H];
        private Double[] pImagOut2 = new Double[W * H];

        private const string faceFileName = "face.csv";

        public Wallet()
        {
            InitializeComponent();
            this.Loaded += Page_Loaded;
        }

        void Page_Loaded(object sender, RoutedEventArgs e)
        {
            if ((compareSignal = DSP.Utilities.loadSignal(faceFileName)) == null) {
                MessageBox.Show("Face to recognize has not been sampled.\nPlease proceed to sampling before continue."); return; }
        }
    }
}

```

```

        if (PhotoCamera.IsCameraTypeSupported(CameraType.FrontFacing))
        {
            cam = new Microsoft.Devices.PhotoCamera(CameraType.FrontFacing);
            cam.CaptureImageAvailable += cam_CaptureImageAvailable;
            viewfinderBrush.SetSource(cam);
        }
        else
        {
            if (PhotoCamera.IsCameraTypeSupported(CameraType.Primary))
            {
                cam = new Microsoft.Devices.PhotoCamera(CameraType.Primary);

                cam.CaptureImageAvailable += cam_CaptureImageAvailable;
                viewfinderBrush.SetSource(cam);
            }
        }

        void cam_CaptureImageAvailable(object sender, ContentReadyEventArgs e)
        {
            Deployment.Current.Dispatcher.BeginInvoke(delegate()
            {
                //Take JPEG stream and decode into a WriteableBitmap object
                MainPage.CapturedImage = PictureDecoder.DecodeJpeg(e.ImageStream, W, H);

                //Collapse visibility on the progress bar once writeable bitmap is
                visible.
                progressBar.Visibility = Visibility.Collapsed;

                int[] pixel = MainPage.CapturedImage.Pixels;

                int color = 0;
                for (int y = 0; y < MainPage.CapturedImage.PixelHeight; y++)
                {
                    for (int x = 0; x < MainPage.CapturedImage.PixelWidth; x++)
                    {
                        color = MainPage.CapturedImage.Pixels[x + (y *
MainPage.CapturedImage.PixelWidth)];
                        pRealIn[x + (y * MainPage.CapturedImage.PixelWidth)] =
DSP.Utilities.ColorToGray(color) & 0xFF;
                    }
                }

                Double[] match;
                double mse = 0;

                System.Diagnostics.Debug.WriteLine("Using Fourier");
                DSP.FourierTransform.Compute2D((uint)W, (uint)H, ref pRealIn, null, ref
pRealOut, ref pImagOut, false);
                match = DSP.Utilities.triangularExtraction(ref pRealOut, (uint)W,
(uint)H, (uint)matchSamples, 0);

                mse = DSP.Utilities.MSE(ref compareSignal, ref match,
(int)matchSamples);

                // Normalize Fourier result
                mse /= 1000000000;

                mse = Math.Round(mse);


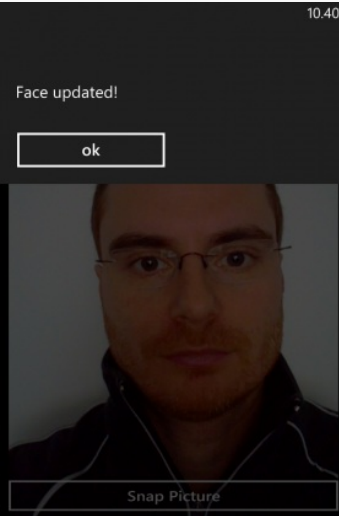

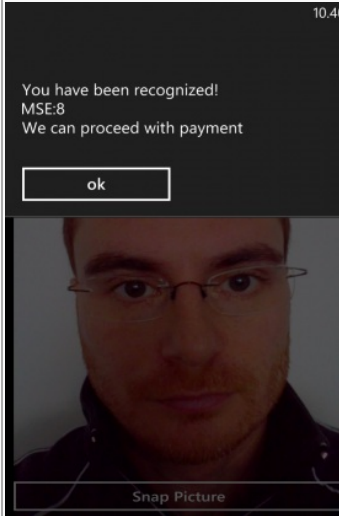
                System.Diagnostics.Debug.WriteLine("MSE:" + mse);


                if (mse < 55)
                {
                    // There was a recognition. You can proceed with your payment
                    implementation.
                    MessageBox.Show("You have been recognized!\nMSE:" + mse + "\nWe can
proceed with payment");
                }
                else
                {
                    // There wasn't a recognition. You can choose to simply stop the
                    payment or start another process based on your needs

```

```
        MessageBox.Show("You have not been recognized!\nMSE:"+mse+"\nWe  
can't proceed with payment.\nIf you are the correct people, please retry to shoot a new  
photo with different environment conditions.");  
    }  
  
    });  
}  
  
private void SaveImage(object sender, RoutedEventArgs e)  
{  
    cam.CaptureImage();  
}  
  
}  
}
```

Screenshots

MainPage.xaml Acquiring the sample image	MainPage.xaml Updated the face into file	How Application looks into Wallet	Wallet.xaml Recognition process
			

 Warning: Please be aware that recognition process is strongly influenced by the background, how the face is rotated or light conditions. When skin detection will be available we can count on better performance

Summary

Although it is still possible to greatly improve the algorithm the results already achieved are very good. Anyway it's important to understand that results are strongly influenced by the background, how the face is rotated or light conditions when compared to the master image. Provided that there is no method completely safe, the proposed solution is intended as an additional method to the normal safety systems not necessarily as a substitute. The choice depends most on the security level you want to achieve regard to the data that you want to protect.

Interesting links

- [PayPal set to launch facial-recognition for ins-store purchasing via smartphones](#)

