**Contoso Jobs**

**Lab 6. Add NFC Messaging**

In this Lab you will add support for NFC. NFC can be used for a number of scenarios on Surface Hub. You could use NFC to identify a user, send messages from a personal device, or pair a personal device to an application.

Universal Windows Apps on the Surface Hub support the Windows.Network.Proximity classes for access to the NFC device embedded in the Surface Hub.

In order to complete this lab you will need at least two devices that support NFC with the Proximity classes. Surface Hub can be one of these devices, the other device could be a Windows Phone or Windows tablet that supports NFC. You can also purchase USB NFC devices, such as the Sony RC-S380.

In the folder that you find this document should also be a folder named NfcConnect. The NfcConnect folder contains a simple sample application that sends out a message using NFC. This is a UWP app that can be run on a Windows Phone or a Windows tablet.

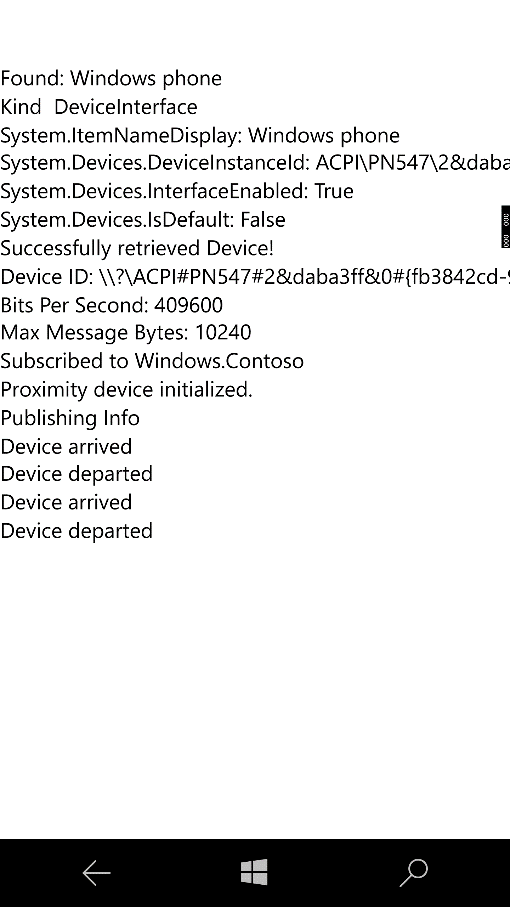


Figure 1The NfcConnect app running on a Windows Phone

This Lab continues from the previous Lab, or you can start from the start point in the folder named *Start Contoso Jobs NFC*

1. Open the existing **Contoso Jobs.sln** solution file in **Visual Studio 2015**
2. In order to support the proximity sensor (NFC) in a UWP App, the App Manifest file must request access to the proximity sensors from the user.
3. In Microsoft Visual Studio, in **Solution Explorer**, open the designer for the application manifest by double-clicking the **package.appxmanifest** item.
4. Select the **Capabilities** tab.
5. Check the box for **Proximity**.
6. In **Solution Explorer** right click on the **Common** folder and from the context menu select **Add | Class…**
7. Name the new class NFC
8. Change the NFC class to be public and add a local private member variable to the class to hold an instance of a **ProximityDevice** class

using Windows.Networking.Proximity;

...

public class NFC

{

ProximityDevice proximityDevice;

}

1. Add a method to the NFC class to retrieve the NFC device. This method uses the same DeviceInformation class that you used in the Camera tutorial. Instead of getting Camera devices you are now getting Proximity devices. This method also outputs debug strings to provide information about the Proximity device found.   
   NOTE: you need to be cautious here as not all Proximity devices support NFC, for example the Surface Pro 3, Surface Pro 4 and Surface Book will return a ProximityDevice for the Pen sensor. This makes sense as the screen uses proximity to determine when the Surface pen is near.

using Windows.Devices.Enumeration;  
using System.Diagnostics;  
...

private async Task<ProximityDevice> GetNfcDevice()

{

ProximityDevice device = null;

string selectorString = ProximityDevice.GetDeviceSelector();

DeviceInformationCollection deviceInfoCollection =

await DeviceInformation.FindAllAsync(selectorString, new List<string>()

{ "{FB3842CD-9E2A-4F83-8FCC-4B0761139AE9} 2" });

if (deviceInfoCollection.Count > 0)

{

foreach (DeviceInformation info in deviceInfoCollection)

{

Debug.WriteLine("Found: " + info.Name);

foreach (string prop in info.Properties.Keys)

{

object value = info.Properties[prop];

if (null != value)

{

Debug.WriteLine(prop + ": " + value.ToString());

}

}

device = ProximityDevice.FromId(info.Id);

}

}

return device;  
 }

1. Add an Init method to the NFC class. This method will initialize the NFC device and set it up to receive messages published from another NFC device.

public async Task Init()

{

proximityDevice = await GetNfcDevice();

if (proximityDevice != null)

{

string deviceInfo = "Successfully retrieved Device!\r\n" +

"Device ID: " + proximityDevice.DeviceId + "\r\n" +

"Bits Per Second: " + proximityDevice.BitsPerSecond + "\r\n" +

"Max Message Bytes: " + proximityDevice.MaxMessageBytes + "\r\n";

Log(deviceInfo);

// Reference: https://msdn.microsoft.com/en-us/library/windows/apps/hh701129.aspx

TryToSubscribe("Windows.Contoso", messageReceivedHandler);

proximityDevice.DeviceArrived += ProximityDeviceArrived;

proximityDevice.DeviceDeparted += ProximityDeviceDeparted;

Log("Proximity device initialized.\n");

}

else

{

Log("Failed to initialized proximity device.");

}

}

1. This requires some other methods be added to the NFC class to compile. Add the Log method

private void Log(string msg)

{

Debug.WriteLine(msg);

}

1. Add the TryToSubscribe method. This method attempts to subscribe the ProximityDevice to a certain message type. Some devices only support some of the message types, this will protect your code from crashing the app if the type of message you wish to subscribe to is not available on the device.

private void TryToSubscribe(string type,

MessageReceivedHandler messageReceivedHandler)

{

try

{

proximityDevice.SubscribeForMessage(type, messageReceivedHandler);

Log($"Subscribed to {type}");

}

catch (Exception)

{

Log($"Cannot subscribe for {type}");

}

}

1. Add event handlers for a Proximity Device arriving and a Proximity Device departing. These will simply log the events for now. You may want to use these to trigger a behavior in your application.

private void ProximityDeviceDeparted(ProximityDevice sender)

{

Log("Device departed");

}

private void ProximityDeviceArrived(ProximityDevice sender)

{

Log($"Device arrived: {sender.DeviceId}");

}

1. Add the event handler for a message received from an NFC publisher. Here you will log the message again. You can use this to receive messages from other devices and those could be used to trigger different behaviors.

private void messageReceivedHandler(ProximityDevice sender, ProximityMessage message)

{

Log($"Recieved {message.MessageType}");

if ("Windows.Contoso" == message.MessageType)

{

Log($"Message is: {message.DataAsString}");

}

}

1. You need to initialize the NFC class when the app starts, for now lets do this in the Jobs page. Open the Jobs.xaml.cs file and add a new class member variable of type NFC.

private NFC nfc;

1. Add the Jobs class constructor to initialize the NFC object

public Jobs()

{

InitializeComponent();

DataContext = jobsViewModel;

nfc = new NFC();

nfc.Init();

}

1. Build and Run the application (F5). If you watch the Output debug Window you should see the ProximityDevice be initialized. Then run the NfcConnect app on another device and place the two device together. The device running the NfcConnect app will publish a message to the device running Contoso Jobs. You should the following in your Output debug Window

Recieved Windows.Contoso  
Message is: I love Surface Hub

In this tutorial you have learned how to initialize the NFC device using the Proximity APIs. You have learned how to subscribe to a message from another NFC device that is publishing data.

Think how you can use this to add features to your software to help streamline some actiovities.