User interface design

for Windows (Phone) 8, using C#

# Lab 5 – Convert your app to a Windows Store app

## Functional Goals

Migrate the code in a Windows Phone app to a Windows Store app for Windows 8/RT with the same functionality.

## Learning Goals

* Understand the differences and similarities between the Windows Phone SDK and the Windows Store (WinRT) SDK

## Prerequisites

* You’ll need to install Visual Studio (2012 Ultimate was used to create this lab) from the MSDN/DreamSpark service on ANGEL’s RosePortal.
* You’ll also need to download and install the Windows Phone SDK (8.0 was used to create this lab) from https://dev.windowsphone.com/en-us/downloadsdk
  + To properly run the Windows Phone emulator, you’ll need to ensure that **second-level address translation (SLAT)** and **hardware Data Execution Prevention** (Execute Disable on Intel systems; No Execute on AMD) are enabled in your BIOS settings.
* A basic understanding of Visual Studio and C# development, such as that gained from the **User interface design in C#, using WPF** series in this document’s repository.
* Code from the Windows Phone RHIT Bandwidth Monitor at <https://github.com/alexmullans/RHITBandwidth/tree/master/WP8RHITBandwidth>

## Submission Instructions

Submit answers to the **1** (or **2**, with extra credit)questions in this lab as a .pdf to the appropriate Moodle submission form.

## Getting started: create a new solution and move the Scraper

Migrating a Windows Phone app to a Windows Store app is relatively straightforward. Essentially, you’re going to create a new Windows Store app, move your code to that app, and then make small changes where necessary to work with the Windows Runtime APIs.

1. In Visual Studio, create a New Project. For the project template, select Blank App (XAML) under the Visual C# -> Windows Store category.
2. In the new project, delete MainPage.xaml. Add a new Basic Page to the project and call it MainPage.xaml.
   1. When asked about missing Common dependencies, allow them to be added.
3. Create a new class called Scraper.cs and copy the code at the end of this lab into the file.

The scraper code is mostly unchanged from what you used for the Windows Phone labs; however, there are three significant changes. The first is a change in how the HTML Agility Pack is used – you don’t have to worry about this. The second is a change in the object used to access application settings; where before you used IsolatedStorageSettings.ApplicationSettings, you must now use ApplicationData.Current.LocalSettings.Values.

Question 1: What is the difference between RoamingSettings and LocalSettings? Why are Local Settings a better choice for this application? (Feel free to use MSDN) (6 points)

The third change is in the way the scraper is used and referred to. The WinRT API does not allow direct control of threads, so the Dispatcher callback you used on Windows Phone to update the UI is both not needed and not possible. To accommodate this change, add the following method to MainPage.xaml.cs:

private async Task Scrape()

{

var results = await Scraper.Scrape();

UpdateUi(results, true);

}

Notice the async keyword; this is a new feature of C# that enables asynchronous programming. You should consider reading the summary at <http://msdn.microsoft.com/en-US/library/vstudio/hh156513.aspx> to get a better idea for how it works.

Question 2: What return types can an async method have? (6 points)

## Starting off on the right foot: App.xaml.cs

In order to make migrating MainPage as easy as possible, it will help you to have a copy of the Scraper’s username and password in settings before you construct a Settings page. You’ll also want to migrate the initial limits settings from the Windows Phone app so that the meters draw correctly. To do so, add the following code just before if (!rootFrame.Navigate(typeof(MainPage), args.Arguments)):

IPropertySet settings = ApplicationData.Current.LocalSettings.Values;

if (!settings.ContainsKey("MidThreshold"))

{

AddDefaultSettings(settings);

}

And then add the following method:

private static void AddDefaultSettings(IPropertySet settings)

{

settings.Add("MidThreshold", 8000);

settings.Add("LowThreshold", 9000);

settings.Add("MidRate", 1024);

settings.Add("LowRate", 256);

settings.Add("PctDiscount", 75);

}

## Migrating the UI, part 0: BandwidthMeter

The code for BandwidthMeter can be migrated almost verbatim from the code for the Windows Phone app – just add a new UserControl called BandwidthMeter.xaml and copy-paste the XAML and the codebehind. The only changes you’ll need to make are removing Windows Phone-specific styles (Visual Studio should point out the problematic lines) and this change:

* Remove the BandwidthMeterLoaded method and the reference to it in the constructor.
* Change the two public properties to match the code below:

public int LowThresholdMb {

get { return (int)GetValue(LowThresholdMbProperty); }

set { RedTextBlock.Text = value + " MB";

SetValue(LowThresholdMbProperty, value); }

}

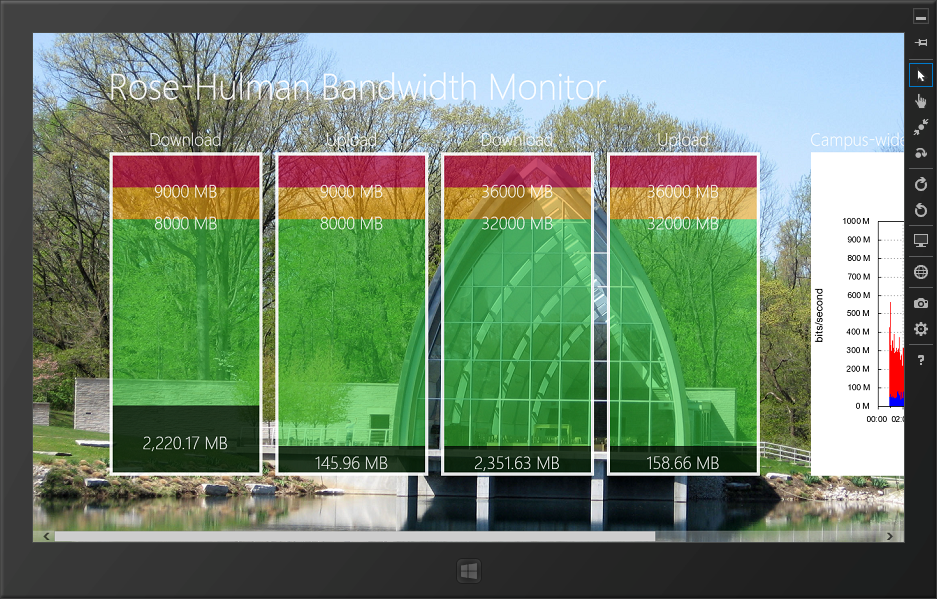
public int MidThresholdMb {

get { return (int)GetValue(MedThresholdMbProperty); }

set { YellowTextBlock.Text = value + " MB";

SetValue(MedThresholdMbProperty, value); } }

## Migrating the UI, part 1: MainPage

The major work that has to occur when you port a Windows Phone app to a Windows Store app is in preparing the UI for a larger interface (and removing dependencies on Windows Phone-only controls like the Panorama used in previous labs). You’re going to craft a UI that looks like the screenshot below.

To do, you’ll need a ScrollViewer wrapped around a standard Grid control. The basis of this layout is provided below to get you started; you’ll need to migrate the code for each bandwidth meter and the campus-wide usage image from the previous lab. This code should go directly after the back button and page title provided with the MainPage.xaml you created early.

You’ll also need to migrate the code in MainPage.xaml.cs, including the UpdateUi and MainPageLoaded methods, and make sure they’re wired up properly using the compatibility steps discussed in the last section. The proper use of the Scrape method is await Scrape();.

Question 3: Submit the XAML code you put in place of the comment in the above code block and a screenshot of the UI that the xaml renders. If you are using a word processor that doesn’t retain Visual Studio’s text formatting on copy/paste, please take a screenshot of your code so that it remains properly formatted and colored. (12 points)

<ScrollViewer Grid.Row="1" VerticalScrollMode="Disabled" VerticalScrollBarVisibility="Hidden"

HorizontalScrollMode="Enabled" HorizontalScrollBarVisibility="Auto">

<Grid Margin="120,0,0,100">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="240" />

<ColumnDefinition Width="20" />

<ColumnDefinition Width="240" />

<ColumnDefinition Width="20" />

<ColumnDefinition Width="240" />

<ColumnDefinition Width="20" />

<ColumnDefinition Width="240" />

<ColumnDefinition Width="80" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="20" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<!-- your controls here -->

</Grid>

</ScrollViewer>

## Migrating the UI, part 2: SettingsFlyout

Migrating the Settings UI is a little more complex because of a paradigm change between Windows Phone and Windows 8. In the former, app developers create a settings page with their activation mechanism of choice; in the latter, all developers are encouraged to use the Settings charm and UI guidelines.

To migrate the Settings UI for your app, use the example at http://code.msdn.microsoft.com/windowsapps/app-settings-sample-1f762f49. The various methods in the example for setting up the Settings charm can all go in MainPage.xaml.cs, since your app only has one page. When you add the OnPopupClosed method, don’t forget to trigger the Scraper with a call to await Scrape();.

Once you’ve added those methods, use the example’s SettingsFlyout.xaml as a starting point for migrating the controls from the Windows Phone SettingsPage.xaml. Your UI should eventually look like the image at right. Because the Windows 8 Settings UI paradigm dictates that apps save each setting as soon as it is entered/chosen, you’ll need to use bindings and public properties to power each field. One example is given on the next page.

One note: the Windows Phone code contains some error handling that WinRT does not support at the moment; just remove this code when you port it.

Question 4: Submit the code for SettingsFlyout.xaml and a screenshot of the UI that the xaml renders. If you are using a word processor that doesn’t retain Visual Studio’s text formatting on copy/paste, please take a screenshot of your code so that it remains properly formatted and colored. (20 points)

public int LowRate

{

get { IPropertySet settings = ApplicationData.Current.LocalSettings.Values;

return (int)settings["LowRate"]; }

set { IPropertySet settings = ApplicationData.Current.LocalSettings.Values;

settings["LowRate"] = value; }

}

<TextBox Name="LowRateTextBox" MaxLength="5" InputScope="Number"

Text="{Binding LowRate, Mode=TwoWay}" />

## Congratulations

You’re done! Don’t forget: submit answers to the **4** question(s) in this lab as a .pdf to the appropriate Moodle submission form.

## Above and beyond: Making it fluid

From here on is extra credit. It is possible to earn full credit on the lab without doing this section.

At this point, you have a fully functional app. However, it’s missing loading animations, error handling, and some other “nice” features that would make it a more fluid, beautiful app. Your instructor may allow you to add some of these things to the app for extra credit.

namespace RoseHulmanBandwidthMonitorApp

{

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using HtmlAgilityPack;

using Windows.Foundation.Collections;

using Windows.Storage;

public struct BandwidthResults

{

#region Public Properties

public string ActualReceived { get; internal set; }

public string ActualSent { get; internal set; }

public string BandwidthClass { get; internal set; }

public string PolicyReceived { get; internal set; }

public string PolicySent { get; internal set; }

#endregion

#region Public Methods and Operators

public static BandwidthResults RetrieveFromIsolatedStorage()

{

IPropertySet settings = ApplicationData.Current.LocalSettings.Values;

var toReturn = new BandwidthResults

{

BandwidthClass = (string)settings["BandwidthClass"],

PolicyReceived = (string)settings["PolicyRecieved"],

PolicySent = (string)settings["PolicySent"],

ActualReceived = (string)settings["ActualReceived"],

ActualSent = (string)settings["ActualSent"]

};

return toReturn;

}

public void SaveToIsolatedStorage()

{

IPropertySet settings = ApplicationData.Current.LocalSettings.Values;

settings["BandwidthClass"] = BandwidthClass;

settings["PolicyRecieved"] = PolicyReceived;

settings["PolicySent"] = PolicySent;

settings["ActualReceived"] = ActualReceived;

settings["ActualSent"] = ActualSent;

}

#endregion

}

public class Scraper

{

#region Public Methods and Operators

public static async Task<BandwidthResults> Scrape()

{

var web = new HtmlWeb();

IPropertySet settings = ApplicationData.Current.RoamingSettings.Values;

string siteToLoad = "http://netreg.rose-hulman.edu/tools/networkUsage.pl";

HtmlDocument doc =

await

web.LoadFromWebAsync(

siteToLoad, new UTF8Encoding(), (string)settings["user"], (string)settings["pass"], "rose-hulman");

return ParseBandwidthDocument(doc);

}

#endregion

#region Methods

private static BandwidthResults ParseBandwidthDocument(HtmlDocument doc)

{

IEnumerable<HtmlNode> summaryTable = from desc in doc.DocumentNode.Descendants()

where desc.Name == "td" && desc.InnerText == "Bandwidth Class"

select desc.ParentNode.ParentNode;

IEnumerable<HtmlNode> resultsList = summaryTable.ElementAt(0).Elements("tr").ElementAt(1).Elements("td");

HtmlNode[] htmlNodes = resultsList as HtmlNode[] ?? resultsList.ToArray();

var results = new BandwidthResults

{

BandwidthClass = htmlNodes.ElementAt(0).InnerText,

PolicyReceived = htmlNodes.ElementAt(1).InnerText,

PolicySent = htmlNodes.ElementAt(2).InnerText,

ActualReceived = htmlNodes.ElementAt(3).InnerText,

ActualSent = htmlNodes.ElementAt(4).InnerText

};

results.SaveToIsolatedStorage();

return results;

}

#endregion

}

}

public class Scraper

{

#region Public Methods and Operators

public static async Task<BandwidthResults> Scrape()

{

var web = new HtmlWeb();

IPropertySet settings = ApplicationData.Current.LocalSettings.Values;

string siteToLoad = "http://netreg.rose-hulman.edu/tools/networkUsage.pl";

HtmlDocument doc =

await

web.LoadFromWebAsync(

siteToLoad, new UTF8Encoding(), (string)settings["user"], (string)settings["pass"], "rose-hulman");

return ParseBandwidthDocument(doc);

}

#endregion

#region Methods

private static BandwidthResults ParseBandwidthDocument(HtmlDocument doc)

{

IEnumerable<HtmlNode> summaryTable = from desc in doc.DocumentNode.Descendants()

where desc.Name == "td" && desc.InnerText == "Bandwidth Class"

select desc.ParentNode.ParentNode;

IEnumerable<HtmlNode> resultsList = summaryTable.ElementAt(0).Elements("tr").ElementAt(1).Elements("td");

HtmlNode[] htmlNodes = resultsList as HtmlNode[] ?? resultsList.ToArray();

var results = new BandwidthResults

{

BandwidthClass = htmlNodes.ElementAt(0).InnerText,

PolicyReceived = htmlNodes.ElementAt(1).InnerText,

PolicySent = htmlNodes.ElementAt(2).InnerText,

ActualReceived = htmlNodes.ElementAt(3).InnerText,

ActualSent = htmlNodes.ElementAt(4).InnerText

};

results.SaveToIsolatedStorage();

return results;

}

#endregion

}

}