User interface design

for Window Phone 8, using C#

# Lab 2 – Asynchronously connecting to data

## Functional Goals

Take a functional Windows Phone 8 app that contains the necessary controls to display bandwidth usage data from Rose-Hulman’s network usage tool and add asynchronous data connections.

## Learning Goals

* Understand Windows Phone UI paradigms, including the Panorama control and the use of “Metro” design to create a straightforward, attractive Windows Phone application
* Understand the basics of the User Control extension paradigm of C# and Windows Phone

## Prerequisites

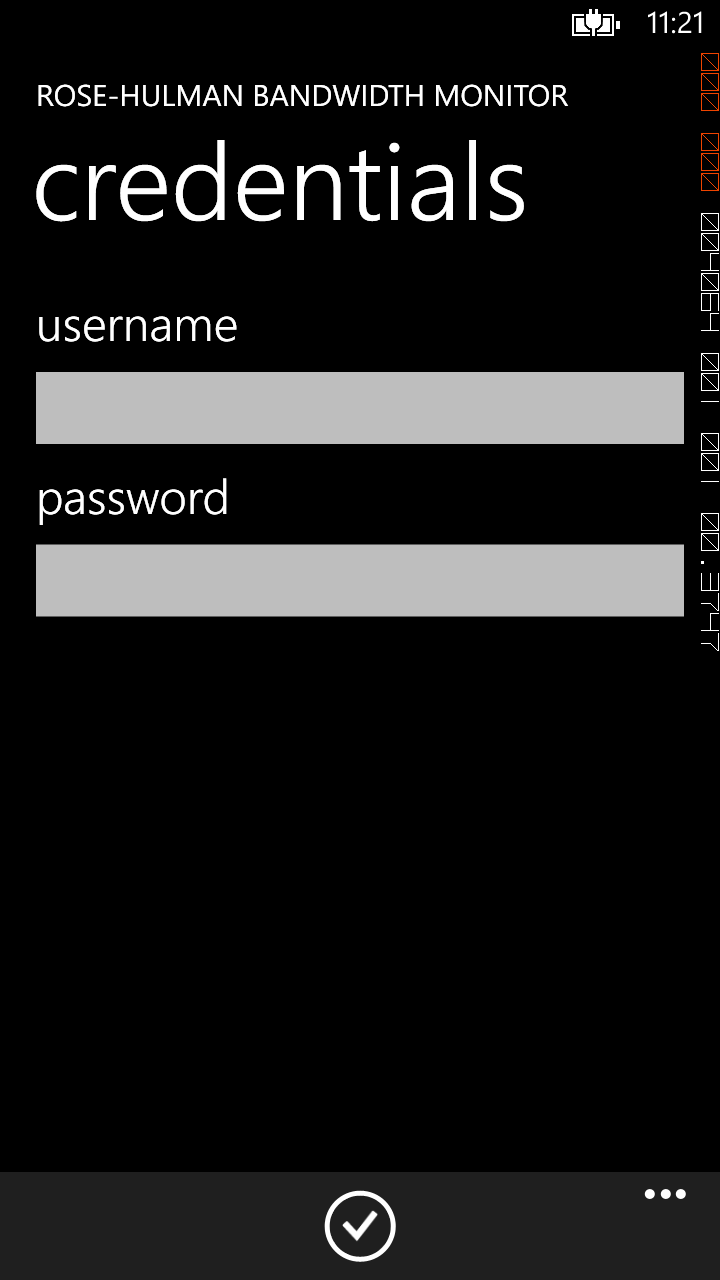
* You’ll need to install Visual Studio (2012 Ultimate was used to create this lab) from the MSDN/DreamSpark service on ANGEL’s RosePortal.
* You’ll also need to download and install the Windows Phone SDK (8.0 was used to create this lab) from https://dev.windowsphone.com/en-us/downloadsdk
  + To properly 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 Lab1 (a complete version may be available from your instructor)

## Submission Instructions

Submit answers to the **3** (or **4**, with extra credit)questions in this lab as a .pdf to the appropriate Moodle submission form.

## Get started: Make a Settings page

To start with, you’ll need to collect the user’s Rose-Hulman network credentials in order to connect to the bandwidth tool. To do that, you’re going to make a new page called SettingsPage.xaml.

1. Create a new Portrait Page called SettingsPage.xaml.
2. Add controls to the page so that it looks like the screenshot on the right.
3. To create the checkmark in the bottom, you’ll need to add an ApplicationBar:

<phone:PhoneApplicationPage.ApplicationBar>

<shell:ApplicationBar>

<shell:ApplicationBarIconButton

IconUri="/Assets/AppBar/check.png"

IsEnabled="True" Text="Save" Click="SaveClick"/>

</shell:ApplicationBar>

</phone:PhoneApplicationPage.ApplicationBar>

1. Copy check.png from **C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v8.0\Icons\Light** to **Assets\AppBar** in your solution folder. Use the Add New Folder and Add Existing Item tools in Visual Studio to add the image to your solution.

## Saving information: IsolatedStorage

To use the credentials the user enters on the Settings page, you’ll need to first add them to IsolatedStorage so that you can access them across pages and app sessions.

1. In SettingsPage.xaml.cs, add/modify the following code:

public SettingsPage()

{

InitializeComponent();

var settings = IsolatedStorageSettings.ApplicationSettings;

if (settings.Contains("user"))

UsernameTextBox.Text = (string) settings["user"];

if (settings.Contains("pass"))

PasswordBox.Password = (string) settings["pass"];

}

private void SaveClick(object sender, EventArgs e)

{

var settings = IsolatedStorageSettings.ApplicationSettings;

if (settings.Contains("user"))

settings["user"] = UsernameTextBox.Text;

else

settings.Add("user", UsernameTextBox.Text);

if (settings.Contains("pass"))

settings["pass"] = PasswordBox.Password;

else settings.Add("pass", PasswordBox.Password);

NavigationService.Navigate(new Uri("/MainPage.xaml", UriKind.Relative));

}

Question 1: What is the data structure type (don’t forget to specify the inner types) that IsolatedStorageSettings.ApplicationSettings uses to store the settings you added above? (9 points)

## Making it accessible: MainPage AppBar

To get to the Settings page from MainPage, you’ll need an AppBar button and a call to NavigationService. Follow the example code in the previous two sections if you’re not sure how to do this.

Question 2: Add the AppBar and navigation logic as described above. Make sure to add an appropriate icon and subtext. Submit the XAML code and codebehind methiod. 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. (18 points)

## Rolling out the welcome mat: First-time setup

Before you get the data to fill out your bandwidth meters, consider the first-run experience. The user is taken to MainPage, but you need them to go to SettingsPage and give you credentials before you can get the bandwidth data. To do so, you’ll need to catch the user in their initial navigation and redirect them to SettingsPage. To do so:

1. In App.xaml.cs, in the constructor, add a new event handler to RootFrame.Navigating:
   1. RootFrame.Navigating += RootFrame\_Navigating;

private void RootFrame\_Navigating(object sender, NavigatingCancelEventArgs e)

{

if (!e.Uri.ToString().Contains("/MainPage.xaml"))

return;

var settings = IsolatedStorageSettings.ApplicationSettings;

if (settings.Contains("user"))

return; // no first-time setup required

e.Cancel = true; // first-time setup required

RootFrame.Dispatcher.BeginInvoke(() => RootFrame.Navigate(new Uri("/SettingsPage.xaml",

UriKind.Relative)));

}

1. Then, add the following method:

When you’re testing the application, you can also use this method to hard-code your username and password so you don’t have to type them with the emulator keyboard each time you run a new build. Just don’t forget to delete them before you check into source or turn in the lab!

Protip

## Getting the data: Scraper.cs

Now that you have the necessary UI in place to display the bandwidth results, you just need to get those results! To get started, create a new file in the root of your solution called Scraper.cs and insert the following code:

using System;

using System.IO.IsolatedStorage;

using System.Linq;

using System.Net;

using System.Text;

using System.Windows;

using HtmlAgilityPack;

namespace RoseHulmanBandwidthMonitorApp

{

public struct BandwidthResults

{

public String BandwidthClass { get; internal set; }

public String PolicyReceived { get; internal set; }

public String PolicySent { get; internal set; }

public String ActualReceived { get; internal set; }

public String ActualSent { get; internal set; }

public void SaveToIsolatedStorage()

{

var settings = IsolatedStorageSettings.ApplicationSettings;

settings["BandwidthClass"] = BandwidthClass;

settings["PolicyRecieved"] = PolicyReceived;

settings["PolicySent"] = PolicySent;

settings["ActualReceived"] = ActualReceived;

settings["ActualSent"] = ActualSent;

}

public static BandwidthResults RetrieveFromIsolatedStorage()

{

var settings = IsolatedStorageSettings.ApplicationSettings;

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;

}

}

You’ll notice that Visual Studio reports a lot of errors related to the using HtmlAgilityPack; line – since you don’t have that reference just yet. To add it, use the Library Package Manager:

public class Scraper

{

private static MainPage \_page;

public static void Scrape(object page)

{

\_page = (MainPage)page;

var web = new HtmlWeb();

web.LoadCompleted += ParseBandwidthDocument;

var settings = IsolatedStorageSettings.ApplicationSettings;

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

web.LoadAsync(siteToLoad,

new UTF8Encoding(),

(String)settings["user"],

(String)settings["pass"],

"rose-hulman");

}

private static void ParseBandwidthDocument(object sender, HtmlDocumentLoadCompleted e)

{

if (e.Error is WebException)

{

\_page.ReportCredentialsError();

return;

}

if (e.Error != null) return;

var doc = e.Document;

var summaryTable = from desc in doc.DocumentNode.Descendants()

where desc.Name == "td" &&

desc.InnerText == "Bandwidth Class"

select desc.ParentNode.ParentNode;

var resultsList = summaryTable.ElementAt(0).Elements("tr").ElementAt(1).Elements("td");

var 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

};

Deployment.Current.Dispatcher.BeginInvoke(() => \_page.UpdateUi(results, true));

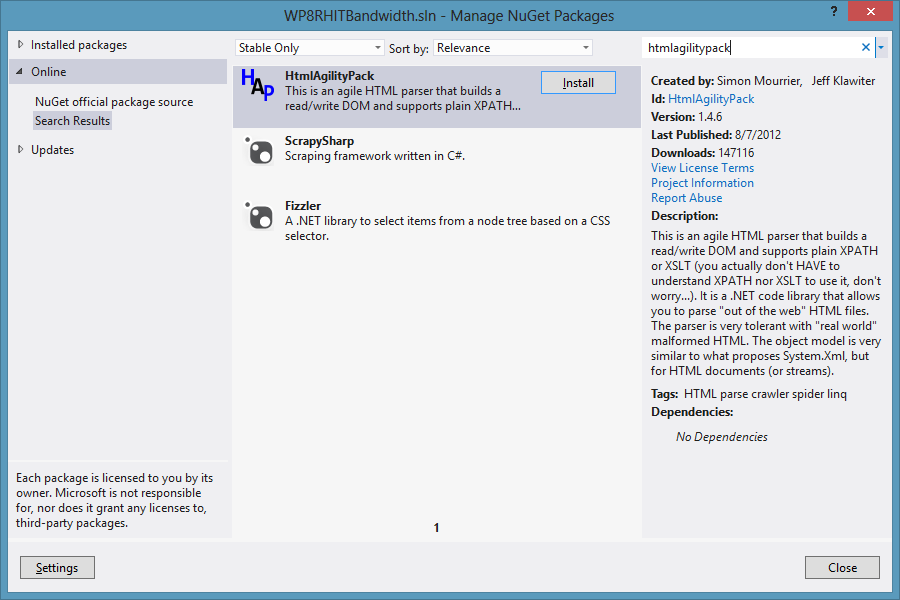
results.SaveToIsolatedStorage();

}

}

}

1. In the Tools menu, select Library Package Manager -> Manage NuGet Packages for Solution…
2. In the resulting dialog, search for HtmlAgilityPack and install the first listing.
3. Note the contents of the readme.txt that appears after you’ve installed the package. Follow the instructions to add a reference to the **Silverlight 4** Xpath dll.

Your project won’t build just yet – there’s a few method calls Scraper relies on that you haven’t yet added to the MainPage codebehind.

## Updating the UI

public MainPage()

{

InitializeComponent();

Loaded += MainPageLoaded;

}

private void MainPageLoaded(object sender, RoutedEventArgs e)

{

var settings = IsolatedStorageSettings.ApplicationSettings;

if (settings.Contains("BandwidthClass"))

UpdateUi(BandwidthResults.RetrieveFromIsolatedStorage(), false);

new Thread(Scraper.Scrape).Start(this);

}

public void UpdateUi(BandwidthResults bandwidthResults, bool fromNetwork)

{

foreach (var control in

new Dictionary<BandwidthMeter, String> {

Add the following code to MainPage.xaml.cs:

## Showing the data: Updating the bandwidth meter

{ PolicyDown, bandwidthResults.PolicyReceived },

{ PolicyUp, bandwidthResults.PolicySent },

{ ActualDown, bandwidthResults.ActualReceived },

{ ActualUp, bandwidthResults.ActualSent } })

{

control.Key.UpdateBorder(GetBandwidthNumberFromString(control.Value), PolicyDown.ActualHeight);

control.Key.UsageTextBlock.Text =

control.Value;

}

}

private static double GetBandwidthNumberFromString(String str)

{

return Double.Parse(str.Split(' ')[0]);

}

internal void ReportCredentialsError()

{

Dispatcher.BeginInvoke(() =>

{

MessageBox.Show("The credentials you entered don't seem to be working, or we can't find the bandwidth tool right now.");

NavigationService.Navigate(new Uri("/SettingsPage.xaml", UriKind.Relative));

});

}

Add the following method to BandwidthMeter.xaml.cs:

public void UpdateBorder(double value, double gridHeight)

{

var to = value / 5000 \* gridHeight;

UsageBorder.Visibility = Visibility.Visible;

UsageBorder.Height = to;

}

Question 3: Comment the Dispatcher.BeginInvoke line and run the app. It will crash; explain the exception you receive and describe how the dispatcher overcomes that exception (MSDN will be of immense help here). (12 points)

## Congratulations

You’re done; run your app and check it out! Don’t forget: submit answers to the **3** (or **4**, with extra credit)questions in this lab as a .pdf to the appropriate Moodle submission form.

## Above and beyond

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

Question 4: Examine Scraper.cs – specifically, the ParseBandwidthDocument method. What is the full name of the SQL-like syntax? (Hint: It’s a C# language feature, introduced with version 3.5 of the .NET Framework) (6 points)