

Build for multiple devices

Lab version: 1.2.0

Last updated: 2/5/2014

# Overview

Phones and tablets are now so common in the workplace that developers simply can’t afford to ignore them. Modern line-of-business systems have to be able to reach users on their terms. Fortunately, Microsoft’s developer platform provides everything developers need to quickly and efficiently bring their line-of-business services to devices of all shapes and sizes.

If you have never built a Windows Store or Windows Phone app, Exercise 1 of this lab will serve as an introduction. You will build the same app twice, once as a Windows Store app and then again as a Windows Phone app. You will then move common code into a portable class library.

In our scenario, our expense reporting environment leverages a WCF service that acts as a hub for all clients. While we only have a WPF application today, it was architected in such a way that we can easily reuse most of the business logic and data access when building applications for other targets, such as Windows Phone and Windows Store. In Exercise 2 of this lab, you will build Windows Store and Windows Phone apps that help you manage expenses.

## Objectives

In this hands-on lab, you will learn how to:

* Build Windows Store and Windows Phone apps
* Create and use portable class libraries

## Prerequisites

The following is required to complete this hands-on lab:

* Microsoft Visual Studio 2013
* Windows 8.1
* Windows Phone 8 SDK

## Setup

This lab picks up where module 3 left off (the environment was not changed in module 4).

## Exercises

This hands-on lab includes the following exercises:

1. Get Started Building Windows Store and Windows Phone Apps
2. Build a Windows Store and Windows Phone Expense App
3. Build a Windows Phone Expense App

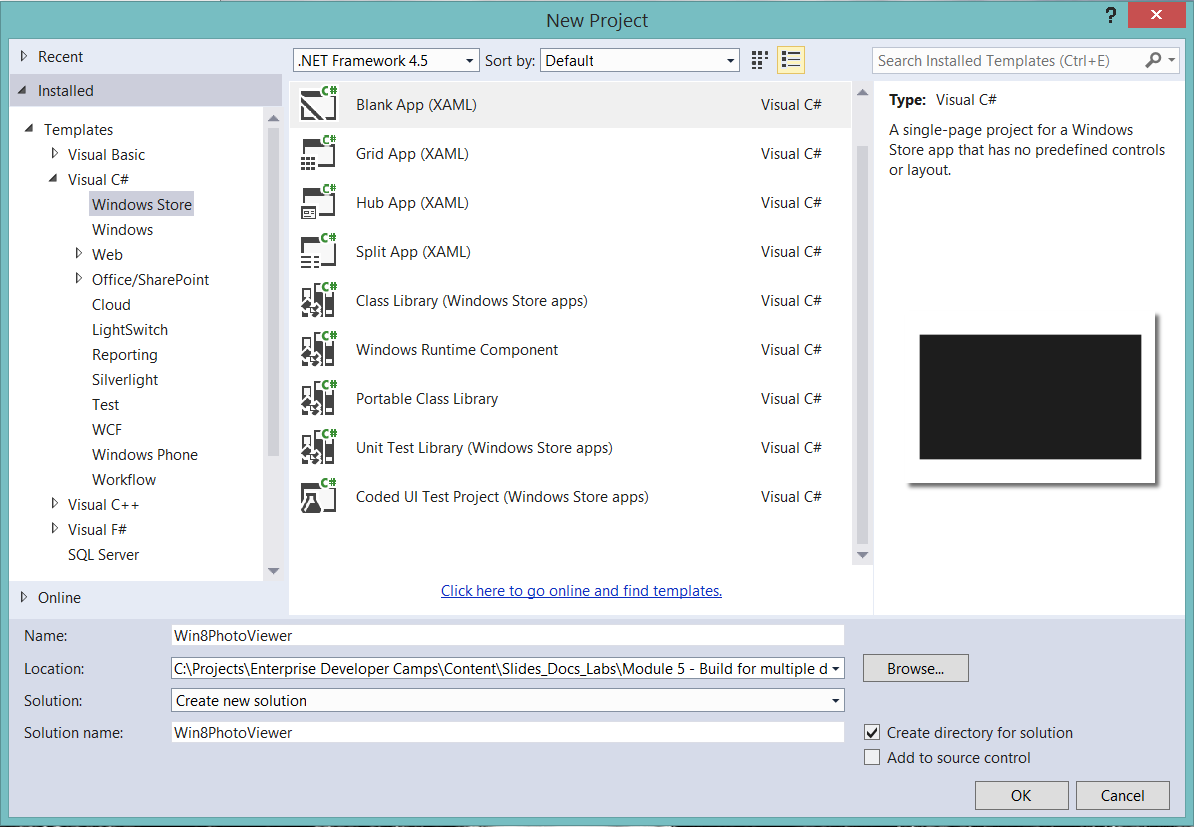
# Exercise 1: Get Started Building Windows Store and Windows Phone Apps

In this exercise, we’ll build Windows Store and Windows Phone versions of a photo viewer app. The app will display a list of photos. If you select a photo the app will display the photo with additional details.

## Task 1: Build a Windows Store app

In this exercise, we’ll build a Windows Store photo viewer app.

1. Open Visual Studio 2013.
2. Select **File | New | Project…** from the main menu.
3. From the **Templates** list on the left side, select **Visual C# | Windows Store**. Select the **Blank App (XAML)** template and use the **Name** **“Win8PhotoViewer”**. Click **OK** to create the project.



1. Right-click the **Win8PhotoViewer** project and select **Add | New Folder**. Name the folder **Models**.
2. Right-click the **Models** folder and select **Add | Class**... Name the class **Snapshot** and click **Add** to add it to the project.
3. Make the Snapshot class public and then add the following code:

public int ID { get; set; }

public string Image { get; set; }

public string Location { get; set; }

public string Comment { get; set; }

public DateTime DateTaken { get; set; }

1. Right-click the **Win8PhotoViewer** project and select **Add | New Folder**. Name the folder **Photos**.
2. Right-click the **Photos** folder and select **Add | Existing Item**... Navigate to the **Lab Files\Photo Viewer\Photos** folder for this module and select the images. Click **Add** to add them to the project.
3. Right-click the **Win8PhotoViewer** project and select **Add | New Folder**. Name the folder **Services**.
4. Right-click the **Services** folder and select **Add | Class**... Name the class **SnapshotService** and click **Add** to add it to the project.
5. Make the Snapshot class public and then add the following using statement:

using System.Collections.ObjectModel;

using Win8PhotoViewer.Models;

1. Add the following code:

static ObservableCollection<Snapshot> listOfSnapshots;

public SnapshotService() { }

public ObservableCollection<Snapshot> GetSnapshots()

{

listOfSnapshots = new ObservableCollection<Snapshot>();

// Add sample data at first, then use to connect

// to real data later.

listOfSnapshots.Add(new Snapshot()

{

ID = 0,

Image = "/Photos/Image1.jpg",

Comment = "Still hungry!",

DateTaken = DateTime.Now,

Location = "Marcy Mountain"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 1,

Image = "/Photos/Image2.jpg",

Comment = "It took a lot of stings to get this one.",

DateTaken = DateTime.Now,

Location = "Algonquin Peak"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 2,

Image = "/Photos/Image3.jpg",

Comment = "Beautiful lake view! What a great day.",

DateTaken = DateTime.Now,

Location = "Whitaker Lake"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 3,

Image = "/Photos/Image4.jpg",

Comment = "A burning flower of power",

DateTaken = DateTime.Now,

Location = "Haystack"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 4,

Image = "/Photos/Image5.jpg",

Comment = "Berrrrup... I can't do the voice right, you had to be there.",

DateTaken = DateTime.Now,

Location = "Skylight Molehill"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 5,

Image = "/Photos/Image6.jpg",

Comment = "Lily pad with the rare Marshall flower",

DateTaken = DateTime.Now,

Location = "Whiteface Pond"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 6,

Image = "/Photos/Image7.jpg",

Comment = "Never found out what kind of flower this is.",

DateTaken = DateTime.Now,

Location = "Dix Valley"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 7,

Image = "/Photos/Image8.jpg",

Comment = "I totally found him this way and did not in any way pose this photo.",

DateTaken = DateTime.Now,

Location = "Gray Skies Ranch"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 8,

Image = "/Photos/Image9.jpg",

Comment = "Then some sort of flying rat in a blue hat attacked me.",

DateTaken = DateTime.Now,

Location = "Iroquois Peak"

});

return listOfSnapshots;

}

public Snapshot GetSnapshot (int snapshotId)

{

Snapshot snapshot = (from \_snapshot in listOfSnapshots

where \_snapshot.ID == snapshotId

select \_snapshot).First();

return snapshot;

}

1. Right-click the **Win8PhotoViewer** project and select **Add | New Folder**. Name the folder **ViewModels**.
2. Right-click the **ViewModels** folder and select **Add | Existing Item**... Navigate to the **Lab Files\Photo Viewer\Windows Store** folder for this module and select **ViewModelBase.cs**. Click **Add** to add it to the project.
3. Right-click the **ViewModels** folder and select **Add | Class**... Name the class **MainViewModel** and click **Add** to add it to the project.
4. Add the following using statements:

using System.Collections.ObjectModel;

using Win8PhotoViewer.Models;

using Win8PhotoViewer.Services;

1. Modify the class code as follows:

public class MainViewModel: ViewModelBase

{

public ObservableCollection<Snapshot> Snapshots { get; private set; }

public MainViewModel()

{

var snapShotService = new SnapshotService();

this.Snapshots = snapShotService.GetSnapshots();

}

}

1. Right-click the **ViewModels** folder and select **Add | Class**... Name the class **SnapshotViewModel** and click **Add** to add it to the project.
2. Add the following using statements:

using Win8PhotoViewer.Models;

using Win8PhotoViewer.Services;

1. Modify the class code as follows:

public class SnapshotViewModel : ViewModelBase

{

public Snapshot Snapshot { get; private set; }

public SnapshotViewModel()

{

}

public void GetSnapshot(int snapshotId)

{

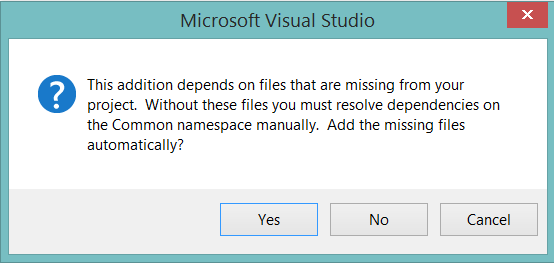
var snapShotService = new SnapshotService();

this.Snapshot = snapShotService.GetSnapshot(snapshotId);

}

}

1. Delete MainPage.xaml.
2. Right-click the **Win8PhotoViewer** project and select **Add | New Item**.
3. Select the **Basic Page** template in the Windows Store section and use the **Name** **“MainPage”**. Click **Add** to create the page. Click **Yes** when you are prompted to add files. Visual Studio adds a number of files in the Common folder.



1. In the MainPage.xaml file, make the following change in bold:

<Page

x:Name="pageRoot"

x:Class="W8PhotoFeed.MainPage"

DataContext="{Binding **MainViewModel**, RelativeSource={RelativeSource Self}}"

1. Make the following change in bold:

<Page.Resources>

<x:String x:Key="AppName">**Windows 8 Snapshot Viewer**</x:String>

</Page.Resources>

1. Add the following code between the two </Grid> elements:

<ListView x:Name="SnapshotsListView"

ItemsSource="{Binding Snapshots}"

Grid.Row="1"

Margin="120,0,0,0">

<ListView.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,17">

<Image Source="{Binding Image}"

Height="250"

Margin="12,12,0,0"

Stretch="Uniform"

HorizontalAlignment="Left"/>

<TextBlock Text="{Binding Location}"

TextWrapping="Wrap"

Margin="12,0,0,0"/>

</StackPanel>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

1. Right-click and select **View Code**.
2. Add the following using statement:

Using Win8PhotoViewer.ViewModels;

1. Remove the following code:

private ObservableDictionary defaultViewModel = new ObservableDictionary();

/// <summary>

/// This can be changed to a strongly typed view model.

/// </summary>

public ObservableDictionary DefaultViewModel

{

get { return this.defaultViewModel; }

}

1. Replace the above code with the following:

private MainViewModel mainViewModel = new MainViewModel();

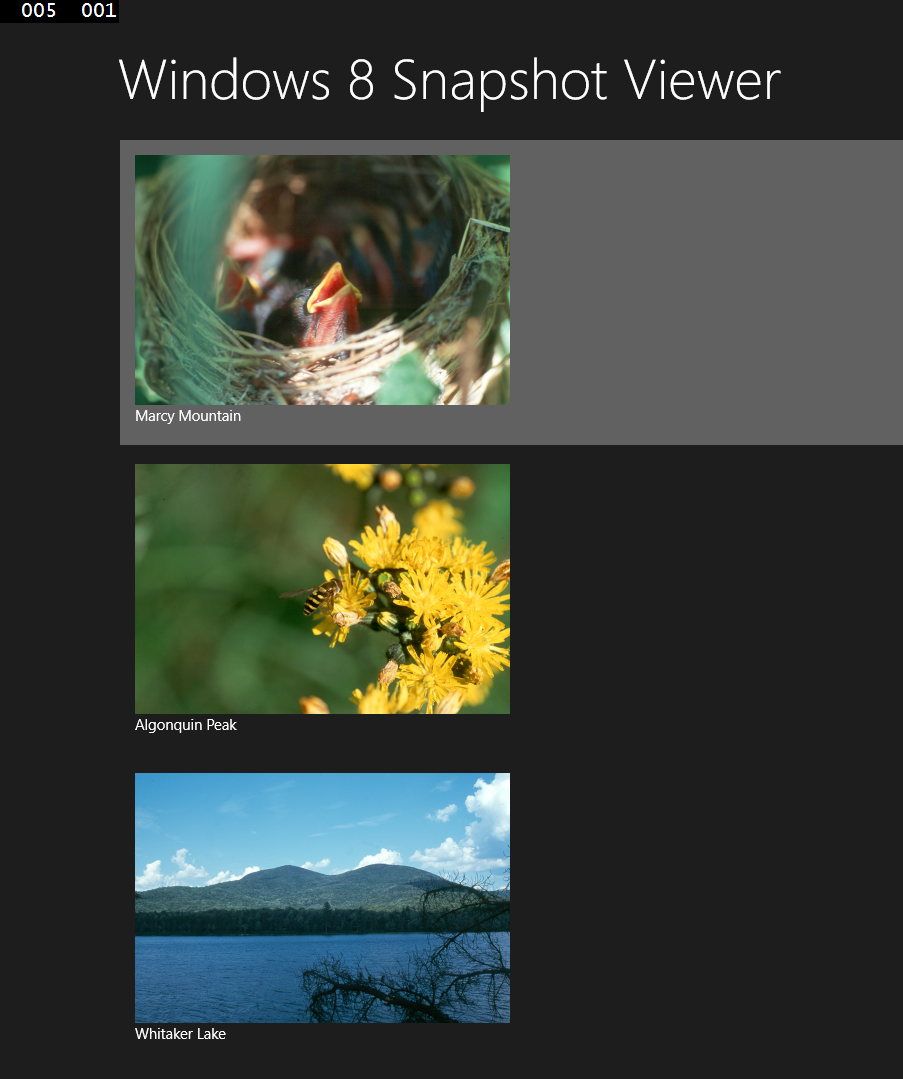
public MainViewModel MainViewModel

{

get { return this.mainViewModel; }

}

1. Press **F5** to build and run the application. You should see images. In addition, you should see counters in the upper left and right of the screen.



1. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.
2. Open App.xaml.cs and comment out the following code in the **OnLaunched** method:

if (System.Diagnostics.Debugger.IsAttached)

{

this.DebugSettings.EnableFrameRateCounter = true;

}

1. Press **F5** to build and run the application. You should not see counters in the upper left and right of the screen.
2. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.
3. Right-click the **Win8PhotoViewer** project and select **Add | New Item**.
4. Select the **Basic Page** template and use the **Name** **“SnapshotPage”**. Click **Add** to create the page.
5. In the SnapshotPage.xaml file, make the following change in bold:

<Page

x:Name="pageRoot"

x:Class="W8PhotoFeed.MainPage"

DataContext="{Binding **SnapshotViewModel**, RelativeSource={RelativeSource Self}}"

1. Add the following code in bold:

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"

**DataContext="{Binding Snapshot}"**>

1. Make the following change in bold:

<TextBlock x:Name="pageTitle" Text="**{Binding Location}**"

1. Add the following code between the two </Grid> elements:

<StackPanel Grid.Row="1"

Margin="120,0,0,0"

Orientation="Vertical">

<Image Source="{Binding Image}"

HorizontalAlignment="Left"

Height="500"

Width="750"

Stretch="Fill"/>

<TextBlock Text="{Binding Comment}"

Style="{StaticResource HeaderTextBlockStyle}"

TextWrapping="Wrap"

Margin="0,20,0,0"/>

<TextBlock Text="taken on:"

Style="{StaticResource SubtitleTextBlockStyle}"

Margin="0,10,0,0"/>

<TextBlock Text="{Binding DateTaken}"

Style="{StaticResource SubtitleTextBlockStyle}"/>

</StackPanel>

1. Right-click and select **View Code**.
2. Add the following using statement:

Using Win8PhotoViewer.ViewModels;

1. Remove the following code:

private ObservableDictionary defaultViewModel = new ObservableDictionary();

/// <summary>

/// This can be changed to a strongly typed view model.

/// </summary>

public ObservableDictionary DefaultViewModel

{

get { return this.defaultViewModel; }

}

1. Replace the above code with the following:

private SnapshotViewModel snapshotViewModel = new SnapshotViewModel ();

public SnapshotViewModel SnapshotViewModel

{

get { return this. snapshotViewModel; }

}

1. In the MainPage.xaml, add the following code to handle the SelectionChanged event when the user selects a photo to view:

<ListView x:Name="SnapshotsListView"

ItemsSource="{Binding Snapshots}"

Grid.Row="1"

Margin="120,0,0,0"

**SelectionChanged="SnapshotsListView\_SelectionChanged"**>

1. Right-click on **SnapshotsListView\_SelectionChanged** and select **Go To Definition**.
2. Add the following code to navigate to the Snapshot page and pass the id of the selected photo:

if (this.SnapshotsListView.SelectedItem == null)

return;

this.Frame.Navigate(typeof(SnapshotPage),

((Snapshot)SnapshotsListView.SelectedItem).ID);

1. Add the following using statement:

Using Win8PhotoViewer.Models;

1. Return to the SnapshotPage.xaml.cs file.
2. In the LoadState method, add the following code to call the GetSnapshot method of the SnapshotViewModel and pass the selected photo’s id:

snapshotViewModel.GetSnapshot(Convert.ToInt32(e.NavigationParameter));

1. Press **F5** to build and run the application. Select an image. You should see the image on a new page with additional information.

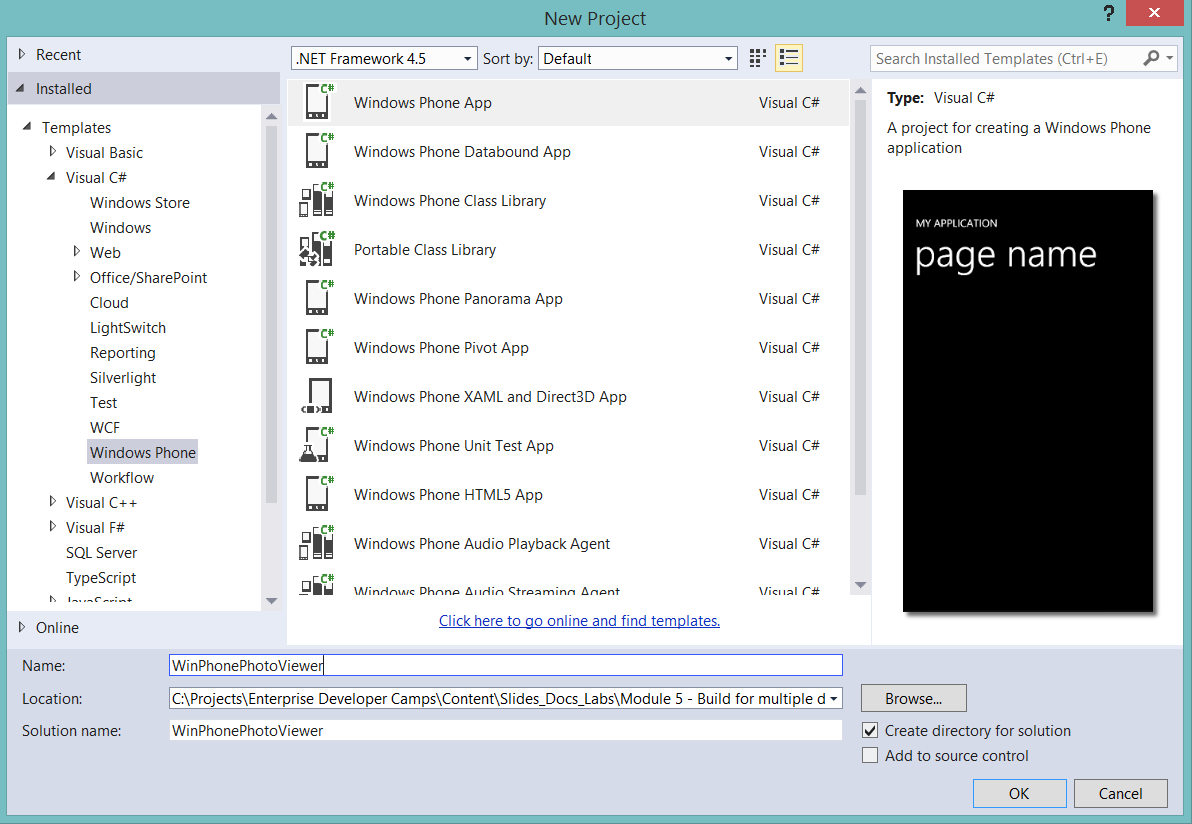


1. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.

## Task 2: Build a Windows Phone app

In this task, we’ll build a Windows Phone version of the photo viewer app.

1. Open Visual Studio 2013.
2. Select **File | New | Project…** from the main menu.
3. From the **Templates** list on the left side, select **Visual C# | Windows Phone**. Select the W**indows Phone App** template and use the **Name** **“WinPhonePhotoViewer”**. Click **OK** to create the project.



1. Right-click the **WinPhonePhotoViewer** project and select **Add | New Folder**. Name the folder **Models**.
2. Right-click the **Models** folder and select **Add | Class**... Name the class **Snapshot** and click **Add** to add it to the project.
3. Make the Snapshot class public and then add the following code:

public int ID { get; set; }

public string Image { get; set; }

public string Location { get; set; }

public string Comment { get; set; }

public DateTime DateTaken { get; set; }

1. Right-click the **WinPhonePhotoViewer** project and select **Add | New Folder**. Name the folder **Photos**.
2. Right-click the **Photos** folder and select **Add | Existing Item**... Navigate to the **Lab Files\Photo Viewer\Photos** folder for this module and select the images. Click **Add** to add them to the project.
3. Right-click the **WinPhonePhotoViewer** project and select **Add | New Folder**. Name the folder **Services**.
4. Right-click the **Services** folder and select **Add | Class**... Name the class **SnapshotService** and click **Add** to add it to the project.
5. Make the Snapshot class public and then add the following using statement:

using System.Collections.ObjectModel;

using WinPhonePhotoViewer.Models;

1. Add the following code:

static ObservableCollection<Snapshot> listOfSnapshots;

public SnapshotService() { }

public ObservableCollection<Snapshot> GetSnapshots()

{

listOfSnapshots = new ObservableCollection<Snapshot>();

// Add sample data at first, then use to connect

// to real data later.

listOfSnapshots.Add(new Snapshot()

{

ID = 0,

Image = "/Photos/Image1.jpg",

Comment = "Still hungry!",

DateTaken = DateTime.Now,

Location = "Marcy Mountain"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 1,

Image = "/Photos/Image2.jpg",

Comment = "It took a lot of stings to get this one.",

DateTaken = DateTime.Now,

Location = "Algonquin Peak"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 2,

Image = "/Photos/Image3.jpg",

Comment = "Beautiful lake view! What a great day.",

DateTaken = DateTime.Now,

Location = "Whitaker Lake"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 3,

Image = "/Photos/Image4.jpg",

Comment = "A burning flower of power",

DateTaken = DateTime.Now,

Location = "Haystack"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 4,

Image = "/Photos/Image5.jpg",

Comment = "Berrrrup... I can't do the voice right, you had to be there.",

DateTaken = DateTime.Now,

Location = "Skylight Molehill"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 5,

Image = "/Photos/Image6.jpg",

Comment = "Lily pad with the rare Marshall flower",

DateTaken = DateTime.Now,

Location = "Whiteface Pond"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 6,

Image = "/Photos/Image7.jpg",

Comment = "Never found out what kind of flower this is.",

DateTaken = DateTime.Now,

Location = "Dix Valley"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 7,

Image = "/Photos/Image8.jpg",

Comment = "I totally found him this way and did not in any way pose this photo.",

DateTaken = DateTime.Now,

Location = "Gray Skies Ranch"

});

listOfSnapshots.Add(new Snapshot()

{

ID = 8,

Image = "/Photos/Image9.jpg",

Comment = "Then some sort of flying rat in a blue hat attacked me.",

DateTaken = DateTime.Now,

Location = "Iroquois Peak"

});

return listOfSnapshots;

}

public Snapshot GetSnapshot (int snapshotId)

{

Snapshot snapshot = (from \_snapshot in listOfSnapshots

where \_snapshot.ID == snapshotId

select \_snapshot).First();

return snapshot;

}

1. Right-click the **WinPhonePhotoViewer** project and select **Add | New Folder**. Name the folder **ViewModels**.
2. Right-click the **ViewModels** folder and select **Add | Existing Item**... Navigate to the **Lab Files\Photo Viewer\Windows Phone** folder for this module and select **ViewModelBase.cs**. Click **Add** to add it to the project.
3. Right-click the **ViewModels** folder and select **Add | Class**... Name the class **MainViewModel** and click **Add** to add it to the project.
4. Add the following using statements:

using System.Collections.ObjectModel;

using WinPhonePhotoViewer.Models;

using WinPhonePhotoViewer.Services;

1. Modify the class code as follows:

public class MainViewModel: ViewModelBase

{

public ObservableCollection<Snapshot> Snapshots { get; private set; }

public MainViewModel()

{

var snapShotService = new SnapshotService();

this.Snapshots = snapShotService.GetSnapshots();

}

}

1. Right-click the **ViewModels** folder and select **Add | Class**... Name the class **SnapshotViewModel** and click **Add** to add it to the project.
2. Add the following using statements:

using WinPhonePhotoViewer.Models;

using WinPhonePhotoViewer.Services;

1. Modify the class code as follows:

public class SnapshotViewModel : ViewModelBase

{

public Snapshot Snapshot { get; private set; }

public SnapshotViewModel()

{

}

public void GetSnapshot(int snapshotId)

{

var snapShotService = new SnapshotService();

this.Snapshot = snapShotService.GetSnapshot(snapshotId);

}

}

1. Delete MainPage.xaml.
2. Right-click the **WinPhonePhotoViewer** project and select **Add | New Item**.
3. Select the **Windows Phone Portrait Page** template in the Windows Phone section and use the **Name** **“MainPage”**. Click **Add** to create the page.
4. In the MainPage.xaml file, add the following code in bold:

<phone:PhoneApplicationPage

x:Class="WPPhotoFeed.MainPage"

**DataContext="{Binding MainViewModel, RelativeSource={RelativeSource Self}}"**

1. Make the following changes in bold:

<StackPanel Grid.Row="0" Margin="12,17,0,28">

<TextBlock Text="**Windows Phone Photo Feed**"

Style="{StaticResource PhoneTextNormalStyle}">

<TextBlock Text="**Snapshots**" Margin="9,-7,0,0"

Style="{StaticResource PhoneTextTitle1Style}"/>

</StackPanel>

1. Add the following code to the ContentPanel Grid:

<phone:LongListSelector x:Name="SnapshotLongListSelector"

Margin="0,0,-12,0"

ItemsSource="{Binding Snapshots}">

<phone:LongListSelector.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,17">

<Image Source="{Binding Image}"

Height="250"

Margin="12,12,0,0"

Stretch="Uniform"

HorizontalAlignment="Left"/>

<TextBlock Text="{Binding Location}"

TextWrapping="Wrap"

Style="{StaticResource PhoneTextExtraLargeStyle}"/>

</StackPanel>

</DataTemplate>

</phone:LongListSelector.ItemTemplate>

</phone:LongListSelector>

1. Right-click and select **View Code**.
2. Add the following using statement:

Using WinPhonePhotoViewer.ViewModels;

1. Add the following code to the MainPage class:

private MainViewModel mainViewModel = new MainViewModel();

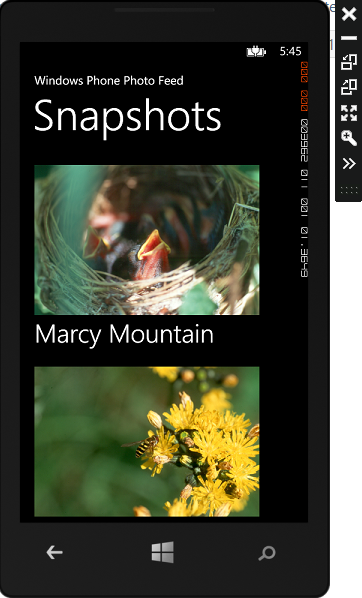
public MainViewModel MainViewModel

{

get { return this.mainViewModel; }

}

1. Press **F5** to build and start the Widows Phone emulator and run the application. You should see images and you should able to scroll down to see more images.



1. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.
2. Right-click the **WinPhonePhotoViewer** project and select **Add | New Item**.
3. Select the **Windows Phone Portrait Page** template and use the **Name** **“SnapshotPage”**. Click **Add** to create the page.
4. In the SnapshotPage.xaml file, make the following change in bold:

<phone:PhoneApplicationPage

x:Class="WPPhotoFeed.SnapshotPage"

**DataContext="{Binding SnapshotViewModel, RelativeSource={RelativeSource Self}}"**

1. Make the following change in bold:

<Grid x:Name="LayoutRoot"

Background="Transparent"

**DataContext="{Binding Snapshot}"**>

1. Make the following changes in bold:

<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">

<TextBlock Text="**Windows Phone Photo Feed**"

Style="{StaticResource PhoneTextNormalStyle}"/>

<TextBlock Text="**{Binding Location}**"

Margin="9,-7,0,0"

Style="{StaticResource PhoneTextTitle1Style}"/></StackPanel>

1. Replace the ContentPanel Grid with the following code:

<ScrollViewer Grid.Row="1"

Margin="12,0,12,0">

<StackPanel >

<Image Source="{Binding Image}"

Margin="12,0" />

<TextBlock Text="{Binding Comment}"

Style="{StaticResource PhoneTextLargeStyle}"

TextWrapping="Wrap"/>

<TextBlock Text="taken on:"

Style="{StaticResource PhoneTextNormalStyle}"/>

<TextBlock Text="{Binding DateTaken, StringFormat=\{0:f\}}"

Style="{StaticResource PhoneTextNormalStyle}"/>

</StackPanel>

</ScrollViewer>

1. Right-click and select **View Code**.
2. Add the following using statement:

Using WinPhonePhotoViewer.ViewModels;

1. Add the following code to the SnapshotPage class:

private SnapshotViewModel snapshotViewModel = new SnapshotViewModel ();

public SnapshotViewModel SnapshotViewModel

{

get { return this. snapshotViewModel; }

}

1. In the MainPage.xaml, add the following code to handle the SelectionChanged event when the user selects a photo to view:

<phone:LongListSelector x:Name="SnapshotLongListSelector"

Margin="0,0,-12,0"

ItemsSource="{Binding Snapshots}"

**SelectionChanged=**

**"SnapshotLongListSelector\_SelectionChanged"**>

1. Right-click on **SnapshotsLongListSelector\_SelectionChanged** and select **Go To Definition**.
2. Add the following code to navigate to the Snapshot page and pass the id of the selected photo:

NavigationService.Navigate(new Uri("/SnapshotPage.xaml?snapshotId=" +

(SnapshotLongListSelector.SelectedItem as Snapshot).ID,

UriKind.Relative));

1. Add the following using statement:

Using WinPhonePhotoViewer.Models;

1. Return to the SnapshotPage.xaml.cs file.
2. Add the following code to call the GetSnapshot method of the SnapshotViewModel and pass the selected photo’s id:

protected override void OnNavigatedTo(NavigationEventArgs e)

{

string selectedIndex = "";

if (NavigationContext.QueryString.TryGetValue("snapshotId", out selectedIndex))

{

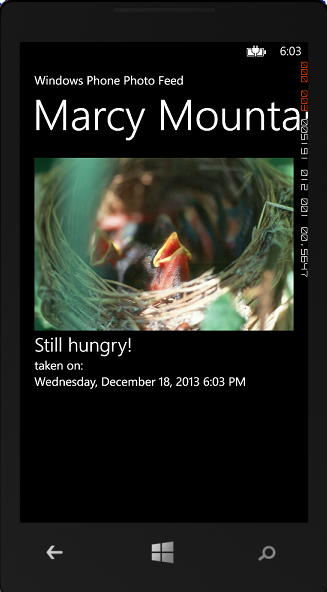
int snapshotId = int.Parse(selectedIndex);

snapshotViewModel.GetSnapshot(Convert.ToInt32(snapshotId));

}

}

1. Press **F5** to build and run the application. Select an image. You should see the image on a new page with additional information.

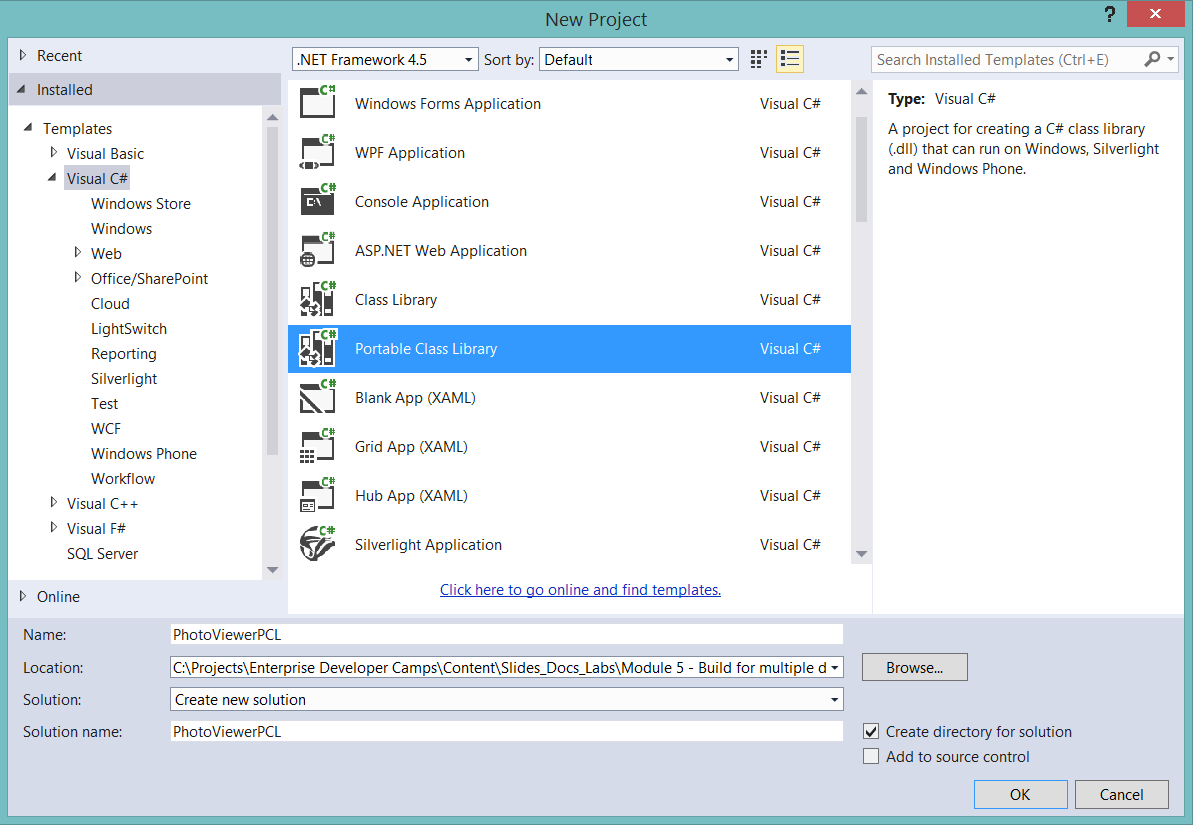


1. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.

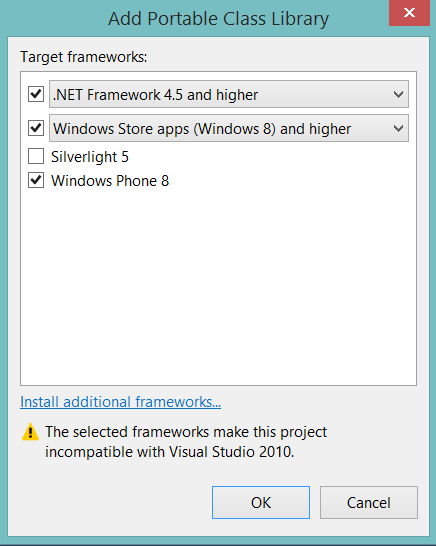
## Task 3: Use a Portable Class Library

As you built each of the two apps, you noticed that a lot of the code is the same in both projects. In this task, we’ll create a portable class library that will contain that shared code. We’ll then modify both the Windows Store and Windows Phone apps to use the portable class library.

1. Open Visual Studio 2013.
2. Select **File | New | Project…** from the main menu.
3. From the **Templates** list on the left side, select **Visual C#**. Select the **Portable Class Library** template and use the **Name** **“PhotoViewerPCL”**. Click **OK** to create the project.



1. Accept the default target frameworks and click **OK**.



1. Delete **Class1.cs**.
2. Right-click the **PhotoViewerPCL** project and select **Add | New Folder**. Name the folder **Models**.
3. Right-click the **Models** folder and select **Add | Existing Item**... Navigate to the Models folder of either the Windows Store or Windows Phone photo viewer project. Select **Snapshot.cs** and click **Add** to add it to the project.
4. Right-click the **PhotoViewerPCL** project and select **Add | New Folder**. Name the folder **Services**.
5. Right-click the **Services** folder and select **Add | Existing Item**... Navigate to the Services folder of either the Windows Store or Windows Phone photo viewer project. Select **SnapshotService.cs** and click **Add** to add it to the project.
6. Right-click the **PhotoViewerPCL** project and select **Add | New Folder**. Name the folder **ViewModels**.
7. Right-click the **ViewModels** folder and select **Add | Existing Item**... Navigate to the ViewModels folder of either the Windows Store or Windows Phone photo viewer project. Select **MainViewModel.cs**, **SnapshotViewModel.cs** and **ViewModelBase.cs** and click **Add** to add them to the project.
8. Open any of the files you just added. Right-click on **Win8PhotoViewer** or **WinPhonePhotoViewer** and select **Refactor | Rename**. Set the **New name** to **PhotoViewerPCL** and click **OK**.
9. Build the solution to confirm you changed the namespaces.
10. Open the **Win8PhotoViewer** solution.
11. Right-click the **References** node in the **Solution Explorer** and select **Add Reference…**.
12. Select **Browse** from the left list and click the **Browse…** button in the bottom right corner to select assemblies to add.
13. Navigate to the **bin\Debug** folder of the **PhotoViewerPCL** project. Select **PhotoViewerPCL.dll**. Click **Add** to add the reference to the project.
14. Click **OK** to close the **Reference Manager** dialog.
15. Delete the **Models**, **Services** and **ViewModels** folders. The code in these files is contained in the portable class library.
16. Open **MainPage.xaml.cs**.
17. Make the following changes in bold:

using **PhotoViewerPCL**.ViewModels;

using **PhotoViewerPCL**.Models;

1. Open **SnapshotPage.xaml.cs**.
2. Make the following change in bold:

using **PhotoViewerPCL**.ViewModels;

1. Press **F5** to build and run the application. The app should behave as it did before. You should be able to view images on the main page and then select an image to view it in more detail.
2. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.
3. Open the **WinPhonePhotoViewer** solution.
4. Right-click the **References** node in the **Solution Explorer** and select **Add Reference…**.
5. Select **Browse** from the left list and click the **Browse…** button in the bottom right corner to select assemblies to add.
6. Navigate to the **bin\Debug** folder of the **PhotoViewerPCL** project. Select **PhotoViewerPCL.dll**. Click **Add** to add the reference to the project.
7. Click **OK** to close the **Reference Manager** dialog.
8. Delete the **Models**, **Services** and **ViewModels** folders. The code in these files is contained in the portable class library.
9. Open **MainPage.xaml.cs**.
10. Make the following changes in bold:

using **PhotoViewerPCL**.ViewModels;

using **PhotoViewerPCL**.Models;

1. Open **SnapshotPage.xaml.cs**.
2. Make the following change in bold:

using **PhotoViewerPCL**.ViewModels;

1. Press **F5** to build and run the application. The app should behave as it did before. You should be able to view images on the main page and then select an image to view it in more detail.
2. In Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.

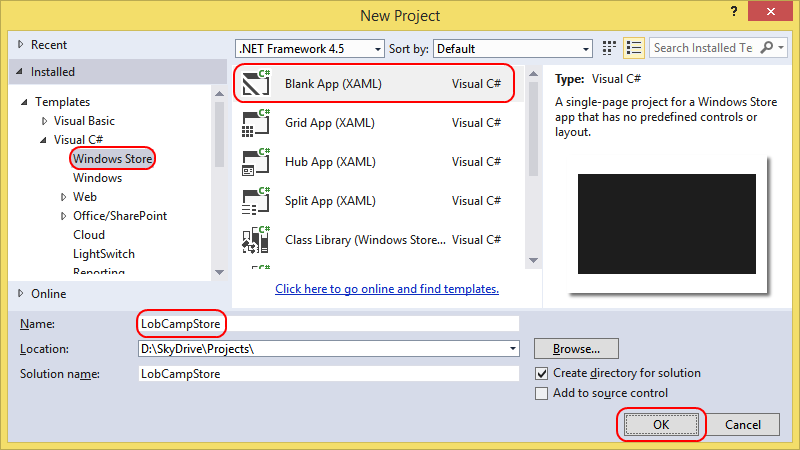
# Exercise 2: Build a Windows Store Expense App

In this exercise, we’ll build a Windows Store app that help users manage expenses. The app will authenticate with Windows Azure Active Directory and display a list of outstanding charges, which the user will be able to edit.

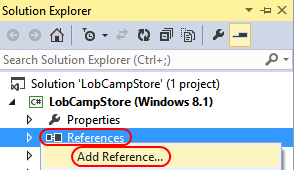
## Task 1: Setting up our Windows Store application

In this task, we’ll create our Windows Store application and set up some of the application plumbing.

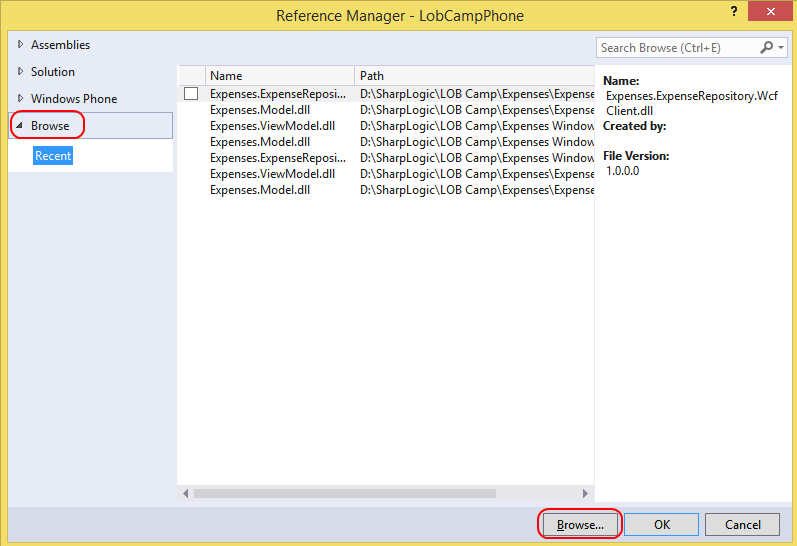
1. Open Visual Studio 2013.
2. Select **File | New | Project…** from the main menu.
3. From the **Templates** list on the left side, select **Visual C# | Windows Store**. Select the **Blank App (XAML)** template and use the **Name** **“LobCampStore”**. Click **OK** to create the project.



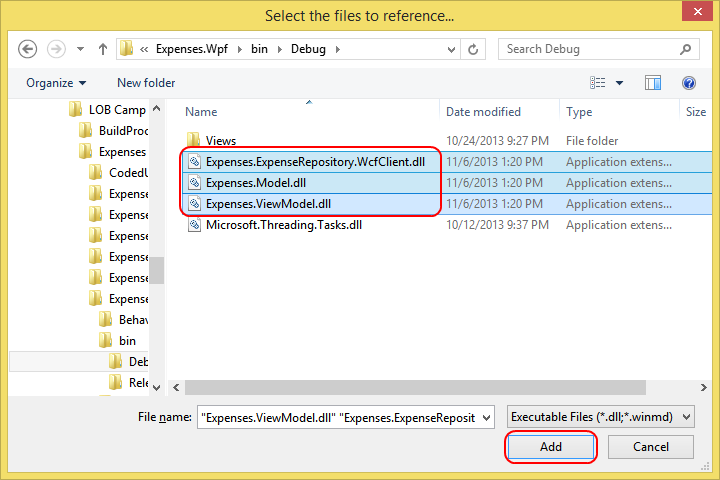
1. Right-click the **References** node in the **Solution Explorer** and select **Add Reference…**.



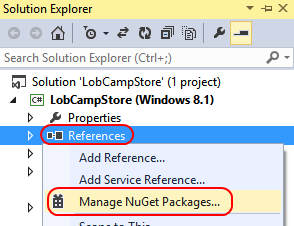
1. Select **Browse** from the left list and click the **Browse…** button in the bottom right corner to select assemblies to add.



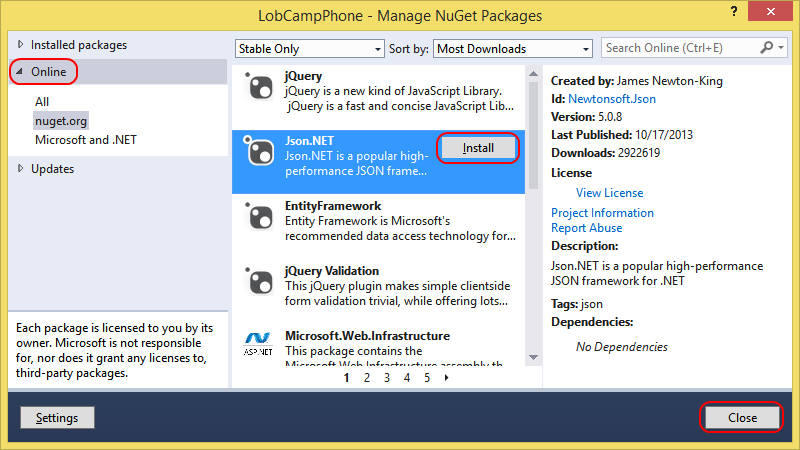
1. Navigate to the **C:\EnterpriseDevCamp\Module 3\Code\End\Expenses WPF\Expenses.Wpf\bin\Debug** folder. We’re going to use the binaries as-is since they’re portable class libraries and were built within the Windows Store parameters. Multi-select **Expenses.ExpenseRepository.WcfClient.dll**, **Expenses.Model.dll**, and **Expenses.ViewModel.dll**. Click **Add** to add references to the current project.



1. Click **OK** to close the **Reference Manager** dialog.
2. In addition to the project assemblies we’ve referenced, we’ll also need to add a reference to **Json.NET**. This library will help us parse the JSON we receive during the Windows Azure Active Directory authentication process. Right-click **References** and select **Manage NuGet Packages**.



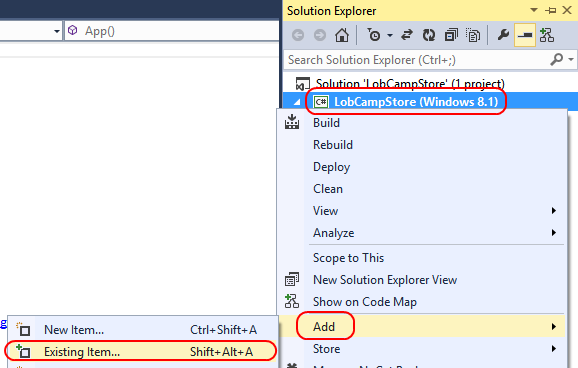
1. Make sure you have the **Online** option selected in the left tree view. **Json.NET** is very popular, so it should be one of the first results you see without searching. If not, search for it using the search box in the top right corner. Click **Install** to install it and click **Close** to dismiss the dialog.



## Task 2: Integrate with Windows Azure Active Directory

In this task, we’ll integrate our application with Windows Azure Active Directory.

1. Now that we’ve set up all the references we need, it’s time to start adding code. We’ll write most of the code ourselves, but there is one file we’ll add to save time. Right-click the project node in **Solution Explorer** and select **Add | Existing Item…**.

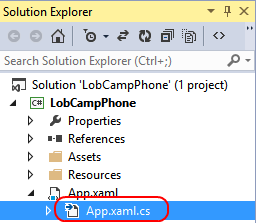


1. Navigate to the **Lab Files** folder for this module and select **WinRtServiceAuthorizer.cs**. Click **Add** to add it to our current project.
2. Open **WinRtServiceAuthorizer.cs**. This class is an **IServiceAuthorizer** that will help us get the authorization token we need to call our secured WCF service. Since we don’t have the same NuGet library we had for WPF (yet), we need to do a little work. In a moment we’ll add a **WebBrowser** control to the **MainPage** so we can use it to log in via Windows Azure Active Directory.

Locate the **LoginUri** property. This is a URI crafted with our Active Directory parameters to provide the login page, hosted at **login.windows.net**.

Every time this browser navigates, we’ll pass the URL to this **WinRtServiceAuthorizer** class so that it can look for the marker token and complete the process to acquire an access token for use with WCF calls. The **ProcessUri** method is well-commented, but the main idea is that once our user signs in, we can use the same HTTP session to request an access code, which is done in an **HttpWebRequest** outside the bounds of the **WebBrowser** itself.

1. Now it’s time to configure the various components in our application. Open **App.xaml.cs**.



1. Add the following **using** declarations at the top of the file.

using Expenses.ExpenseRepository.WcfClient;

using Expenses.Model;

using Expenses.ViewModel;

1. Add the following five constants directly inside the **App** class definition. It’s important to note that Windows Store apps do not have **App.config** files, so you’ll typically need to hardcode certain settings directly in code, especially if they’re not user-configurable. Each line requires an edit.
   1. **\*\*YOUR SITE\*\*** is the Windows Azure Web Site name you deployed your WCF service to. Since we reused that URL as the Active Directory redirect URL and resource ID, this value should be the same for each.
   2. **\*\*YOUR AD CLIENT ID\*\*** is the GUID assigned to the client application we created in your Active Directory instance. You can copy & paste this from the WPF app if needed.
   3. **\*\*YOUR AD NAME\*\*** is the Active Directory name you used when creating your tenant. The final name will be **<something>.onmicrosoft.com**.

const string ExpensesServiceUrl =

"http://\*\*YOUR SITE\*\*.azurewebsites.net/ExpenseService.svc";

const string AzureActiveDirectoryClientID =

"\*\*YOUR AD CLIENT ID\*\*";

const string AzureActiveDirectoryDomainName =

"\*\*YOUR AD NAME\*\*.onmicrosoft.com";

const string AzureActiveDirectoryRedirectUri =

"http://\*\*YOUR SITE\*\*.azurewebsites.net";

const string AzureActiveDirectoryResource =

"http://\*\*YOUR SITE\*\*.azurewebsites.net";

1. The final thing we’ll do in this class is to register some services with the **ServiceLocator**. Add the following code at the end of the **App()** constructor. This code creates an instance of the **WinRtServiceAuthorizer** class with the settings we defined in the previous step. It also sets up a **WcfExpenseRepository** that points to our cloud service, and uses the **WinRtServiceAuthorizer** as its source for authorization tokens.

WinRtServiceAuthorizer winRtServiceAuthorizer =

new WinRtServiceAuthorizer()

{

ClientID = App.AzureActiveDirectoryClientID,

DomainName = App.AzureActiveDirectoryDomainName,

RedirectUri = App.AzureActiveDirectoryRedirectUri,

Resource = App.AzureActiveDirectoryResource,

};

ServiceLocator.Current.SetService<WinRtServiceAuthorizer>(

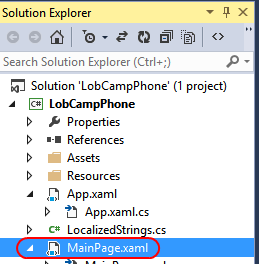
winRtServiceAuthorizer);

ServiceLocator.Current.SetService<IExpenseRepository>(

new WcfClientExpenseRepository(

App.ExpensesServiceUrl, winRtServiceAuthorizer));

1. Open **MainPage.xaml** and switch to XAML view, if necessary.



1. Add the following markup inside the **Grid**. It contains two UI elements.
   1. The **WebBrowser** will present the Windows Azure Active Directory Web site UI that allows the user to log in. It’s defined last, so it will appear in front because we are using a **Grid**. Once the user logs in, we’ll hide the **WebBrowser** so the UI behind it becomes visible.
   2. The **ListBox** will become visible after the user has logged in. It will show a list of outstanding charges, and we’ve designed the item template to be simple. It will provide a **Button** that the user can click to edit the item and bind a **TextBlock** to the item’s **Description**.

<ListBox x:Name="\_chargesListBox">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<Button Content="Edit"

CommandParameter="{Binding ChargeId}"

Click="Button\_Click" />

<TextBlock Text="{Binding Description}" />

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<WebView x:Name="\_webBrowser" />

1. Right-click select **View Code** to bring up **MainPage.xaml.cs**.
2. At the top of **MainPage.xaml.cs**, add the following **using** declarations.

using Expenses.Model;

using Expenses.ViewModel;

1. Add the following private member to the **MainPage** class.

private List<Charge> \_charges;

1. In the **MainPage** constructor, after the call to **InitializeComponent**, add the following code. The code comments explain what we’re doing in each section.

// 1. Get our service authorizer from the ServiceLocator.

WinRtServiceAuthorizer winRtServiceAuthorizer =

ServiceLocator.Current.GetService<WinRtServiceAuthorizer>();

this.Loaded +=

(\_, \_\_) =>

{

// 2. Once the page loads, navigate the WebBrowser to the login.

this.\_webBrowser.Navigate(winRtServiceAuthorizer.LoginUri);

};

this.\_webBrowser.NavigationStarting +=

(\_, e) =>

{

// 3. As the user navigates through the login process, we'll have

// the WinRtServiceAuthorizer look for the code needed to complete

// the process.

winRtServiceAuthorizer.ProcessUri(e.Uri);

};

winRtServiceAuthorizer.TokenReceived +=

async (\_, \_\_) =>

{

// 4. When we complete the token process, hide the WebBrowser.

this.\_webBrowser.Visibility = Visibility.Collapsed;

// 5. Next, we'll get our repository.

IExpenseRepository repository =

ServiceLocator.Current.GetService<IExpenseRepository>();

// 6. Make a request for the "rogreen" user.

Employee employee = await repository.GetEmployeeAsync("rogreen");

// 7. Get the outstanding charges and save them in the app state.

this.\_charges =

await repository.GetOutstandingChargesAsync(employee.EmployeeId);

// 8. Bind the list to our UI.

this.\_chargesListBox.ItemsSource = this.\_charges;

};

## Task 3: Add support for viewing and editing charges

In this task, we’ll add support for viewing and editing details for a charge.

1. When we added the XAML for the **ListBox**, it included an **ItemTemplate** with a **Button** that had a **Click** handler wired up to a method named **Button\_Click**. Add the following method to the **MainPage** class to implement it. It navigates to an **EditChargePage** we’ll create next with the **Charge ID** as the parameter.

private void Button\_Click(object sender, RoutedEventArgs e)

{

string chargeId = (sender as Button).CommandParameter.ToString();

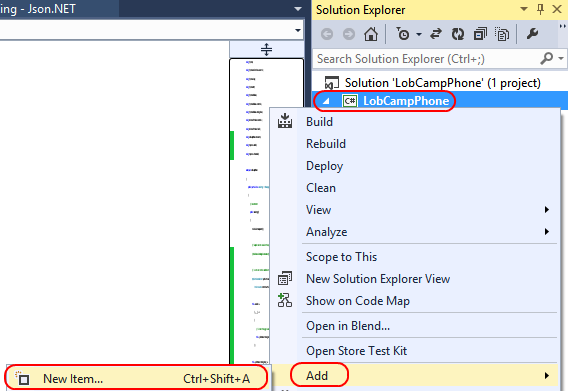
Charge charge =

this.\_charges.First(item => item.ChargeId.ToString() == chargeId);

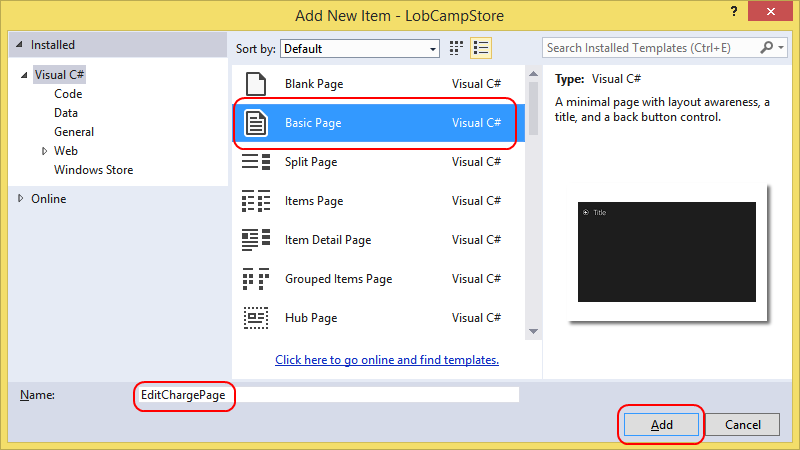
this.Frame.Navigate(typeof(EditChargePage), charge);

}

1. Right-click the project node in **Solution Explorer** and select **Add | New Item…**.



1. In the **Add New Item** dialog, select the **Basic Page** template, set the **Name** to **“EditChargePage”**, and click **Add**. If Visual Studio offers to add dependencies to the project, allow it to.



1. Locate the final **</Grid>** tag near the bottom of the page. Add the following markup right before the closing tag. It contains some labels and fields that we’ll databind to the requested **Charge**. There’s also a **Save** button that we’ll wire up later.

<StackPanel Grid.Row="1">

<TextBlock Text="Description" />

<TextBox Text="{Binding Description, Mode=TwoWay}" />

<TextBlock Text="Merchant" />

<TextBox Text="{Binding Merchant, Mode=TwoWay}" />

<TextBlock Text="Location" />

<TextBox Text="{Binding Location, Mode=TwoWay}" />

<TextBlock Text="TransactionAmount" />

<TextBlock Text="{Binding TransactionAmount}" />

<Button Content="Save" Click="Button\_Click" />

</StackPanel>

1. Right-click in the XAML editor and select **View Code**.
2. Add the following **using** declarations at the top of the file.

using Expenses.Model;

using Expenses.ViewModel;

1. Add the following code to the end of the **OnNavigatedTo** method (you’ll need to expand the **NavigationHelper registration** region. This gets called whenever this page is navigated to, giving us a chance to load the parameters and required data. In this case, we know it’ll be a **Charge**.

this.DataContext = e.Parameter;

1. Finally, we’ll add in code to handle the **Save** button click. This code will save the charge using our **IExpenseRepository** and then navigate back to the list page. Note that since we’re using the **await** keyword on the asynchronous **SaveChargeAsync** call, we need to add the **async** keyword to the method definition.

async private void Button\_Click(object sender, RoutedEventArgs e)

{

await ServiceLocator.Current.GetService<IExpenseRepository>()

.SaveChargeAsync(this.DataContext as Charge);

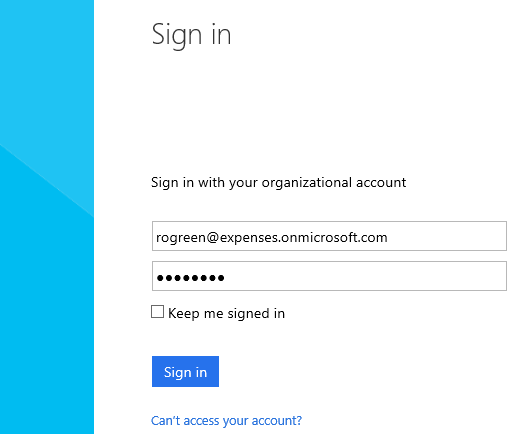
this.Frame.GoBack();

}

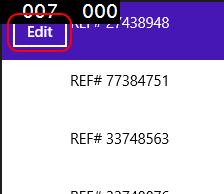
## Task 4: Testing the Windows Store application

In this task, we’ll test out the Windows Store app. If you’d like to reset the data, you can do so at any time from the WPF application’s **Settings** tab by selecting **Reset Data**.

1. Press **F5** to build and run the application.
2. When the app loads, it will present our login page. Log in using the credentials for your **rogreen** account. If you’d like to run the app multiple times without having to keep logging in, you can also check the **Keep me signed in** box. It will remember your login session across app sessions.



1. After logging in, the list of outstanding charges loads. Click the **Edit** button next to the first one.



**NOTE**: The black boxes with numbers are displaying frame rate and other counters. To turn these off, go into App.xaml.cs and comment out the following lines of code:

#if DEBUG

If (System.Diagnostics.Debugger.IsAttached)

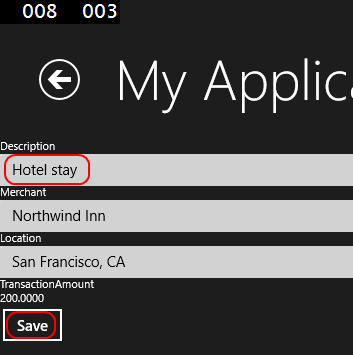
{

this.DebugSettings.EnableFrameRateCounter = true;

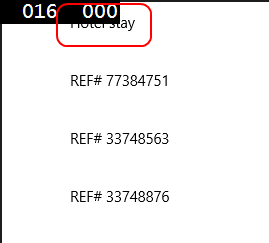
}

#endif

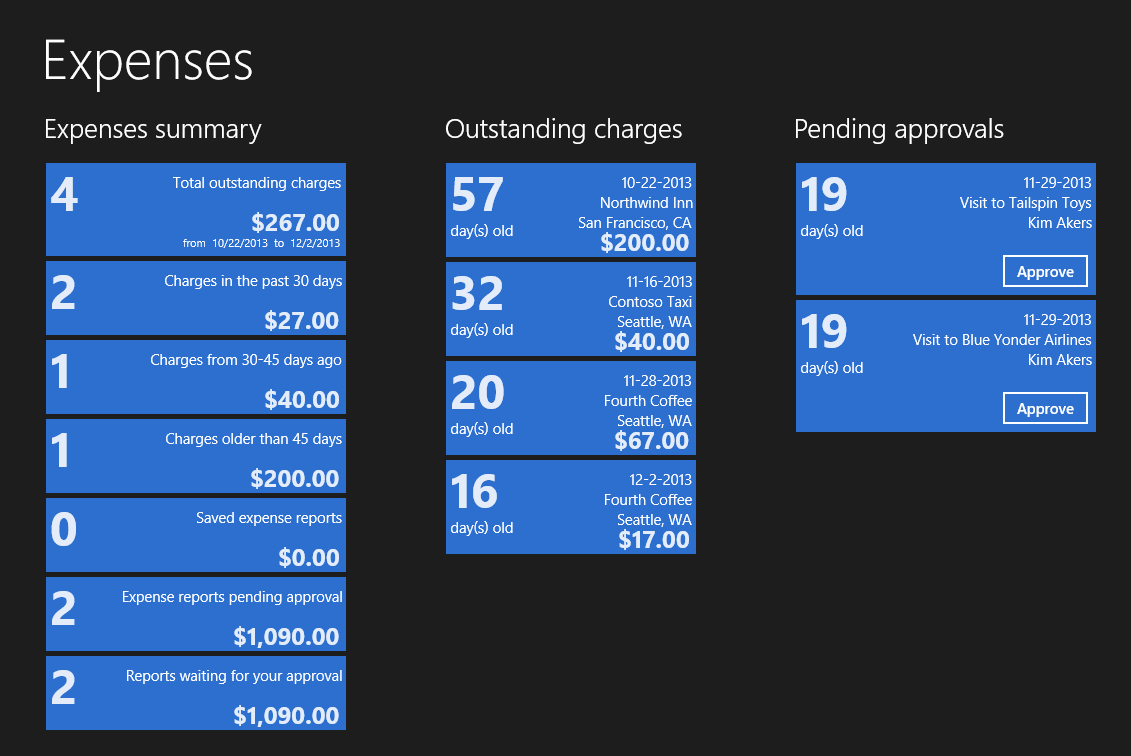
1. On the edit page, change the **Description** to **“Hotel stay”** and click **Save**. This will save our change and navigate back to the list page.



1. Back on the list page, we can see that the description of the first item has updated.



1. In the Visual Studio 2013 instance of your Windows Store application, select **Debug | Stop Debugging** from the main menu.
2. The project you just created is a first pass proof of concept. For another version, open **C:\EnterpriseDevCamp\Module 5\Code\End\Expenses Windows Store\Expenses Windows Store.sln** in Visual Studio 2013.
3. Open **App.xaml.cs** and replace the five configuration settings near the top with the configuration settings from your earlier Windows Store application.
4. Press **F5** to build and run the app. Log in using the same credentials from earlier, if asked. This application has the same functionality as the Windows Phone app you just saw.



1. Back in Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.

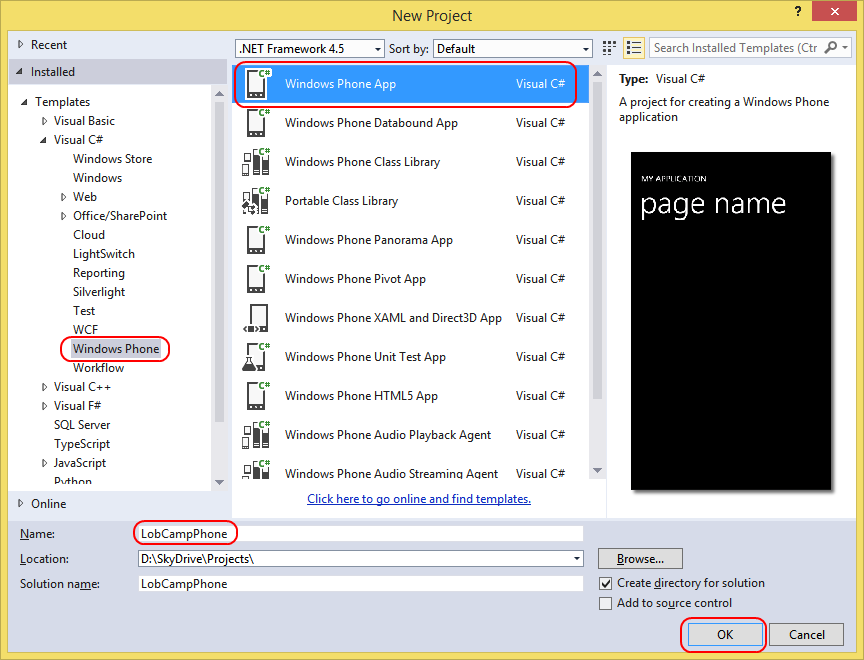
# Exercise 3: Build a Windows Phone Expense App

In this exercise, we’ll build a Windows Phone app that help users manage expenses. The app will authenticate with Windows Azure Active Directory and display a list of outstanding charges, which the user will be able to edit. The entire process is very similar to what we did for Windows Phone, with some subtle differences.

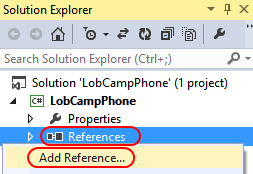
## Task 1: Setting up our Windows Phone application

In this task, we’ll create our Windows Phone application and set up some of the application plumbing.

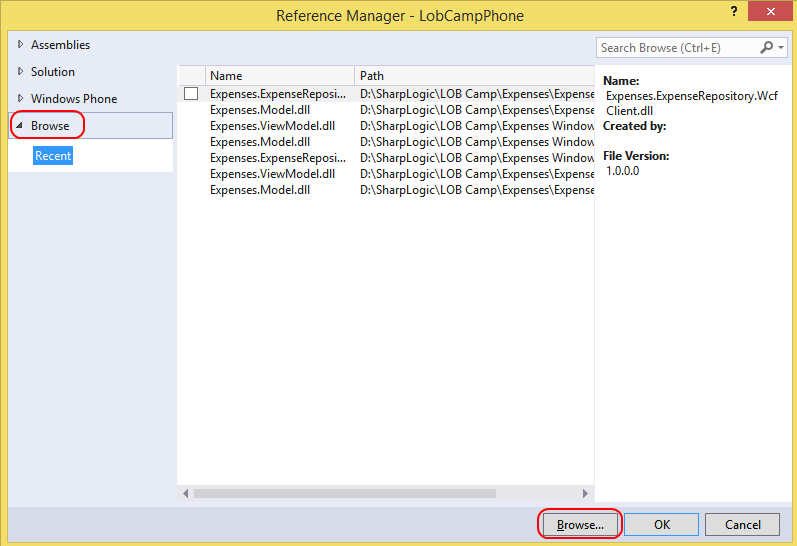
1. Open Visual Studio 2013.
2. Select **File | New | Project…** from the main menu.
3. From the **Templates** list on the left side, select **Visual C# | Windows Phone**. Select the **Windows Phone App** template and use the **Name** **“LobCampPhone”**. Click **OK** to create the project.



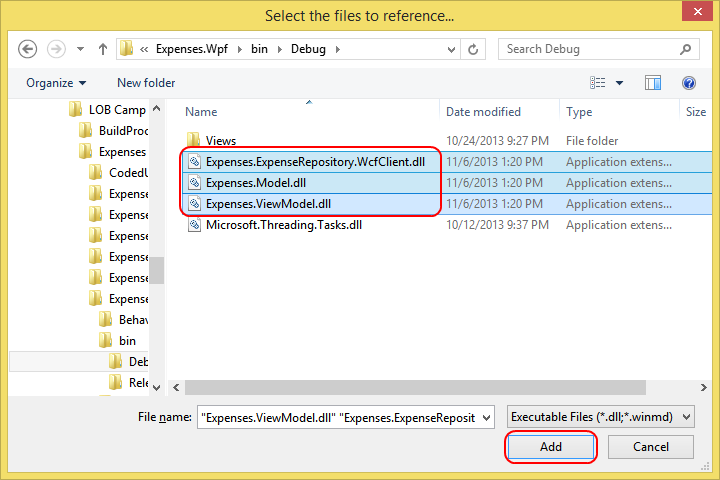
1. Right-click the **References** node in the **Solution Explorer** and select **Add Reference…**.



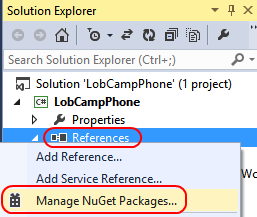
1. Select **Browse** from the left list and click the **Browse…** button in the bottom right corner to select assemblies to add.



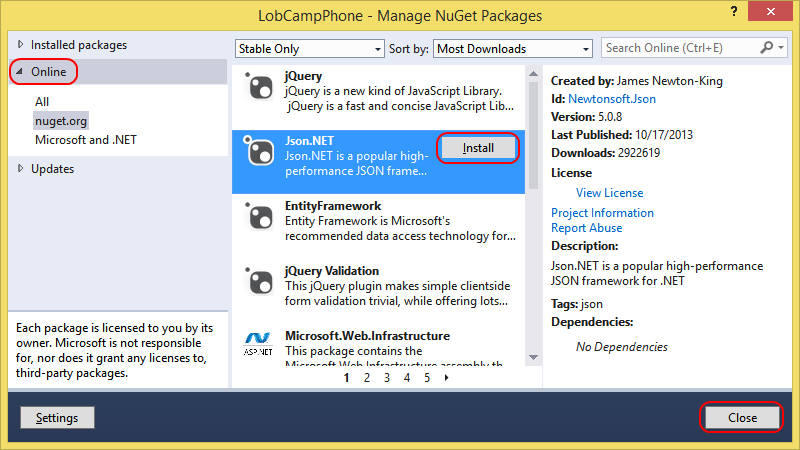
1. Navigate to the **C:\EnterpriseDevCamp\Module 3\Code\End\Expenses WPF\Expenses.Wpf\bin\Debug** folder. We’re going to use the binaries as-is since they’re portable class libraries and were built within the Windows Phone parameters. Multi-select **Expenses.ExpenseRepository.WcfClient.dll**, **Expenses.Model.dll**, and **Expenses.ViewModel.dll**. Click **Add** to add references to the current project.



1. Click **OK** to close the **Reference Manager** dialog.
2. In addition to the project assemblies we’ve referenced, we’ll also need to add a reference to **Json.NET**. This library will help us parse the JSON we receive during the Windows Azure Active Directory authentication process. Right-click on the **References** folder and select **Manage NuGet Packages**…



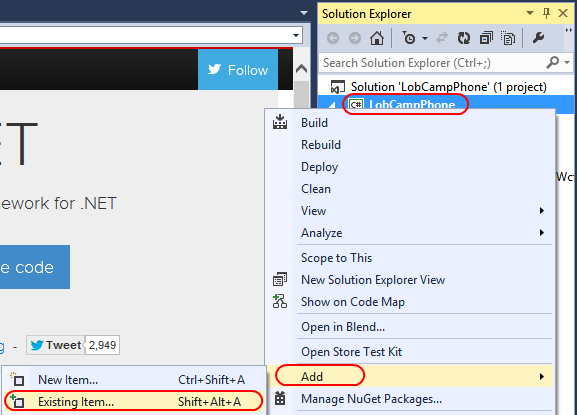
1. Make sure you have the **Online** option selected in the left tree view. **Json.NET** is very popular, so it should be one of the first results you see without searching. If not, search for it using the search box in the top right corner. Click **Install** to install it and click **Close** to dismiss the dialog.



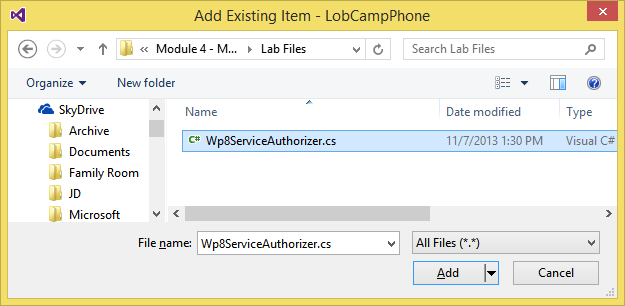
## Task 2: Integrate with Windows Azure Active Directory

In this task, we’ll integrate our application with Windows Azure Active Directory.

1. Now that we’ve set up all the references we need, it’s time to start adding code. We’ll write most of the code ourselves, but there is one file we’ll add to save time. Right-click the project node in **Solution Explorer** and select **Add | Existing Item…**.



1. Navigate to the **Lab Files** folder for this module and select **Wp8ServiceAuthorizer.cs**. Click **Add** to add it to our current project.

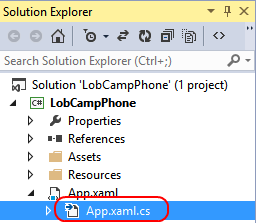


1. Open **Wp8ServiceAuthorizer.cs**. This class is an **IServiceAuthorizer** that will help us get the authorization token we need to call our secured WCF service. Since we don’t have the same NuGet library we had for WPF (yet), we need to do a little work. In a moment we’ll add a **WebBrowser** control to the **MainPage** so we can use it to log in via Windows Azure Active Directory.

Locate the **LoginUri** property. This is a URI crafted with our Active Directory parameters to provide the login page, hosted at **login.windows.net**.

Every time this browser navigates, we’ll pass the URL to this **Wp8ServiceAuthorizer** class so that it can look for the marker token and complete the process to acquire an access token for use with WCF calls. The **ProcessUri** method is well-commented, but the main idea is that once our user signs in, we can use the same HTTP session to request an access code, which is done in an **HttpWebRequest** outside the bounds of the **WebBrowser** itself.

1. Now it’s time to configure the various components in our application. Open **App.xaml.cs**.



1. Add the following **using** declarations at the top of the file.

using Expenses.ExpenseRepository.WcfClient;

using Expenses.Model;

using Expenses.ViewModel;

1. Add the following five constants directly inside the **App** class definition (App.xaml.cs). It’s important to note that Windows Phone apps do not have **App.config** files, so you’ll typically need to hardcode certain settings directly in code, especially if they’re not user-configurable. Each line requires an edit. You can copy this code from the previous solution if you prefer.
   1. **\*\*YOUR SITE\*\*** is the Windows Azure Web Site name you deployed your WCF service to. Since we reused that URL as the Active Directory redirect URL and resource ID, this value should be the same for each.
   2. **\*\*YOUR AD CLIENT ID\*\*** is the GUID assigned to the client application we created in your Active Directory instance. You can copy & paste this from the WPF app if needed.
   3. **\*\*YOUR AD NAME\*\*** is the Active Directory name you used when creating your tenant. The final name will be **<something>.onmicrosoft.com**.

const string ExpensesServiceUrl =

"http://\*\*YOUR SITE\*\*.azurewebsites.net/ExpenseService.svc";

const string AzureActiveDirectoryClientID =

"\*\*YOUR AD CLIENT ID\*\*";

const string AzureActiveDirectoryDomainName =

"\*\*YOUR AD NAME\*\*.onmicrosoft.com";

const string AzureActiveDirectoryRedirectUri =

"http://\*\*YOUR SITE\*\*.azurewebsites.net";

const string AzureActiveDirectoryResource =

"http://\*\*YOUR SITE\*\*.azurewebsites.net";

1. The final thing we’ll do in this class is to register some services with the **ServiceLocator**. Add the following code at the end of the **App()** constructor. This code creates an instance of the **Wp8ServiceAuthorizer** class with the settings we defined in the previous step. It also sets up a **WcfExpenseRepository** that points to our cloud service, and uses the **Wp8ServiceAuthorizer** as its source for authorization tokens.

Wp8ServiceAuthorizer wp8ServiceAuthorizer =

new Wp8ServiceAuthorizer()

{

ClientID = App.AzureActiveDirectoryClientID,

DomainName = App.AzureActiveDirectoryDomainName,

RedirectUri = App.AzureActiveDirectoryRedirectUri,

Resource = App.AzureActiveDirectoryResource,

};

ServiceLocator.Current.SetService<Wp8ServiceAuthorizer>(

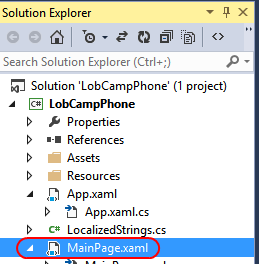
wp8ServiceAuthorizer);

ServiceLocator.Current.SetService<IExpenseRepository>(

new WcfClientExpenseRepository(

App.ExpensesServiceUrl, wp8ServiceAuthorizer));

1. Open **MainPage.xaml** and switch to XAML view, if necessary.



1. Locate the **<Grid>** named “**ContentPanel”**. Add the following markup inside the **Grid**. It contains two UI elements.
   1. The **WebBrowser** will present the Windows Azure Active Directory Web site UI that allows the user to log in. It’s defined last, so it will appear in front because we are using a **Grid**. Once the user logs in, we’ll hide the **WebBrowser** so the UI behind it becomes visible.
   2. The **ListBox** will become visible after the user has logged in. It will show a list of outstanding charges, and we’ve designed the item template to be simple. It will provide a **Button** that the user can click to edit the item and bind a **TextBlock** to the item’s **Description**.

<ListBox x:Name="\_chargesListBox">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<Button Content="Edit"

CommandParameter="{Binding ChargeId}"

Click="Button\_Click" />

<TextBlock Text="{Binding Description}" />

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<phone:WebBrowser x:Name="\_webBrowser" IsScriptEnabled="True" />

1. Right-click somewhere in the XAML and select **View Code** to bring up **MainPage.xaml.cs**.
2. At the top of **MainPage.xaml.cs**, add the following **using** declarations.

using Expenses.Model;

using Expenses.ViewModel;

1. In the **MainPage** constructor, after the call to **InitializeComponent**, add the following code. The code comments explain what we’re doing in each section.

// 1. Get our service authorizer from the ServiceLocator.

Wp8ServiceAuthorizer wp8ServiceAuthorizer =

ServiceLocator.Current.GetService<Wp8ServiceAuthorizer>();

this.Loaded +=

(\_, \_\_) =>

{

// 2. Once the page loads, navigate the WebBrowser to the login.

this.\_webBrowser.Navigate(wp8ServiceAuthorizer.LoginUri);

};

this.\_webBrowser.Navigating +=

(\_, e) =>

{

// 3. As the user navigates through the login process, we'll have

// the Wp8ServiceAuthorizer look for the code needed to complete

// the process.

wp8ServiceAuthorizer.ProcessUri(e.Uri);

};

wp8ServiceAuthorizer.TokenReceived +=

async (\_, \_\_) =>

{

// 4. When we complete the token process, hide the WebBrowser.

this.\_webBrowser.Visibility = Visibility.Collapsed;

// 5. Next, we'll get our repository.

IExpenseRepository repository =

ServiceLocator.Current.GetService<IExpenseRepository>();

// 6. Make a request for the "rogreen" user.

Employee employee = await repository.GetEmployeeAsync("rogreen");

// 7. Get the outstanding charges and save them in the app state.

PhoneApplicationService.Current.State["charges"] =

await repository.GetOutstandingChargesAsync(employee.EmployeeId);

// 8. Bind the list to our UI.

this.\_chargesListBox.ItemsSource =

PhoneApplicationService.Current.State["charges"] as List<Charge>;

};

### Task 3: Add support for viewing and editing charges

In this task, we’ll add support for viewing and editing details for a charge.

1. When we added the XAML for the **ListBox**, it included an **ItemTemplate** with a **Button** that had a **Click** handler wired up to a method named **Button\_Click**. Add the following method to the **MainPage** class to implement it. It navigates to an **EditChargePage** we’ll create next with the **Charge ID** as the parameter.

private void Button\_Click(object sender, RoutedEventArgs e)

{

string chargeId = (sender as Button).CommandParameter.ToString();

this.NavigationService.Navigate(

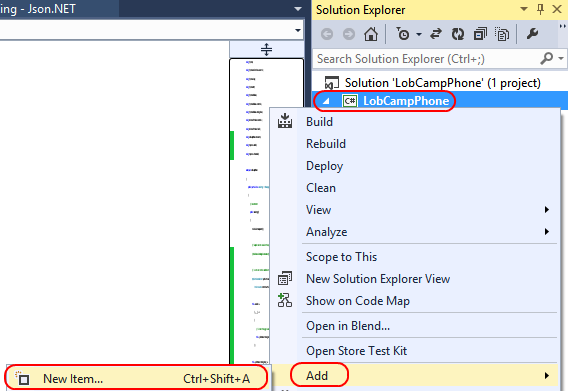
new Uri(

string.Format("/EditChargePage.xaml?chargeId={0}", chargeId),

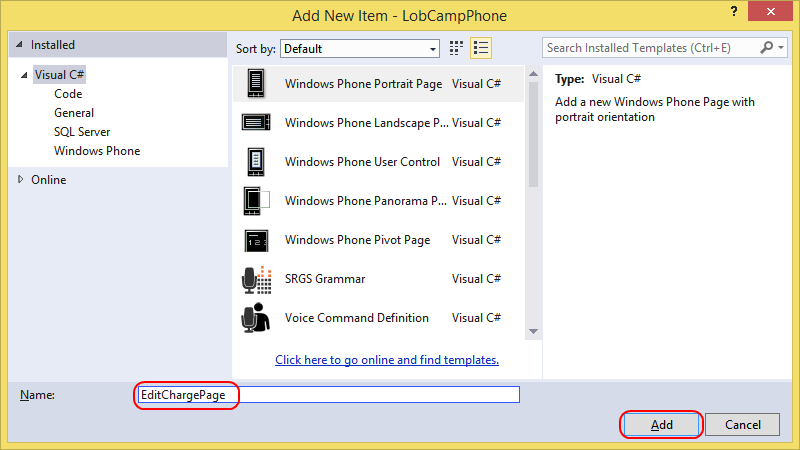
UriKind.Relative));

}

1. Right-click the project node in **Solution Explorer** and select **Add | New Item…**.



1. In the **Add New Item** dialog, select the Windows Phone Portrait Page template, set the **Name** to **“EditChargePage”** and click **Add**.



1. Locate the **<Grid>** named “**ContentPanel”**. Add the following markup inside the **Grid**. It contains some labels and fields that we’ll databind to the requested **Charge**. There’s also a **Save** button that we’ll wire up later.

<StackPanel>

<TextBlock Text="Description" />

<TextBox Text="{Binding Description, Mode=TwoWay}" />

<TextBlock Text="Merchant" />

<TextBox Text="{Binding Merchant, Mode=TwoWay}" />

<TextBlock Text="Location" />

<TextBox Text="{Binding Location, Mode=TwoWay}" />

<TextBlock Text="TransactionAmount" />

<TextBlock Text="{Binding TransactionAmount, StringFormat='C'}" />

<Button Content="Save" Click="Button\_Click" />

</StackPanel>

1. Right-click in the XAML editor and select **View Code**.
2. Add the following **using** declarations at the top of the file.

using Expenses.Model;

using Expenses.ViewModel;

1. Add the following method to override **OnNavigatedTo**. This gets called whenever this page is navigated to, giving us a chance to load the parameters and required data. Here we’ll simply grab the list we stored on the previous page and find the instance that matches the specified ID.

protected override void OnNavigatedTo(NavigationEventArgs e)

{

base.OnNavigatedTo(e);

string chargeId = this.NavigationContext.QueryString["chargeId"];

List<Charge> charges =

PhoneApplicationService.Current.State["charges"] as List<Charge>;

this.DataContext =

charges.First(item => item.ChargeId.ToString() == chargeId);

}

1. Finally, we’ll add in code to handle the **Save** button click. This code will save the charge using our **IExpenseRepository** and then navigate back to the list page. Note that since we’re using the **await** keyword on the asynchronous **SaveChargeAsync** call, we need to add the **async** keyword to the method definition.

async private void Button\_Click(object sender, RoutedEventArgs e)

{

await ServiceLocator.Current.GetService<IExpenseRepository>()

.SaveChargeAsync(this.DataContext as Charge);

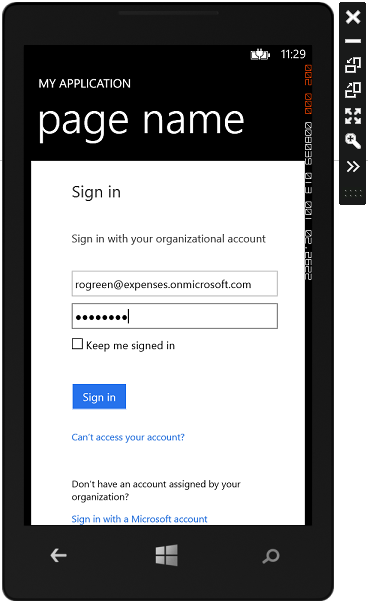
this.NavigationService.GoBack();

}

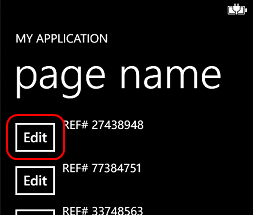
## Task 4: Testing the phone application

In this task, we’ll test out the phone app. If you’d like to reset the data, you can do so at any time from the WPF application’s **Settings** tab by selecting **Reset Data**.

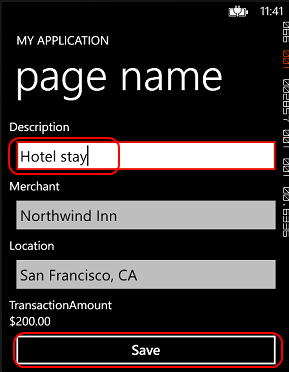
1. Press **F5** to build and run the application.
2. When the app loads, it will present our login page. Log in using the credentials for your **rogreen** account. Note that you can use your system’s keyboard by clicking the emulator and pressing the **Page Down** key. If you’d like to run the app multiple times without having to keep logging in, you can also check the **Keep me signed in** box. It will remember your login session across app sessions.



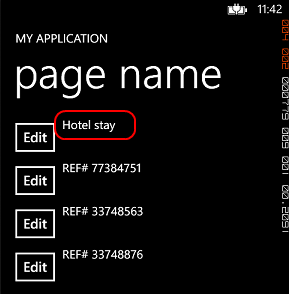
1. After logging in, the list of outstanding charges loads. Click the **Edit** button next to the first one.



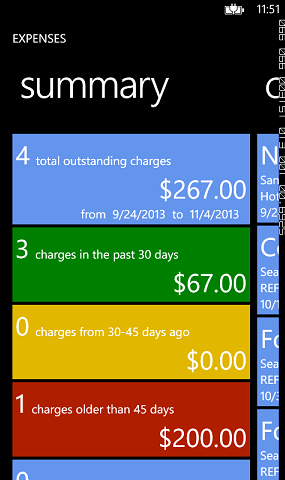
1. On the edit page, change the **Description** to **“Hotel stay”** and click **Save**. This will save our change and navigate back to the list page.



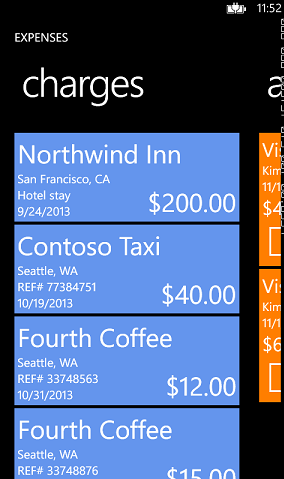
1. Back on the list page, we can see that the description of the first item has updated.



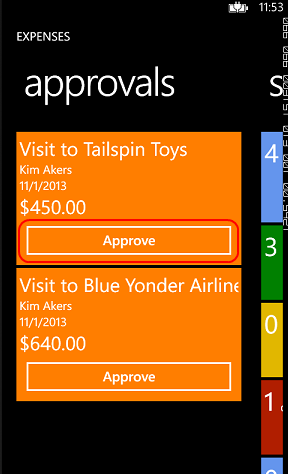
1. In the Visual Studio 2013 instance of your phone application, select **Debug | Stop Debugging** from the main menu.
2. The project you just created is a first pass proof of concept. For another version, open **C:\EnterpriseDevCamp\Module 5\Code\End\Expenses Windows Phone\Expenses Windows Phone.sln** in Visual Studio 2013.
3. Open **App.xaml.cs** and replace the five configuration settings near the top with the configuration settings from your earlier phone application.
4. Press **F5** to build and run the app. Log in using the same credentials from earlier, if asked. This application represents an extension of the codebase we began earlier. It makes more use of the **ViewModel** PCL and provides more functionality.



1. In Visual Studio 2013, open **SummaryView.xaml** from the **Views** folder of the project. This application takes advantage of the **Panorama** control to provide users with a horizontal layout of different views in the same page.
2. Switch back to the application and swipe from right to left. This will bring up the outstanding charges.



1. Swipe again from right to left. This brings us to our approvals view. Note that if this view is empty, you can reset the data using the WPF application and then reload this application. Click to approve the first expense report.



1. Back in Visual Studio 2013, select **Debug | Stop Debugging** from the main menu.