

# Windows Phone 8

development for absolute beginners

**Video Series:** <http://channel9.msdn.com/Series/Windows-Phone-8-Development-for-Absolute-Beginners>

**Source Code:** <http://aka.ms/absbeginnerdevwp8>

Bob Tabor, Clint Rutkas, Larry Lieberman



## Contents

<b>Part 1: Series Introduction .....</b>	8
<b>Part 2: Installing Windows Phone SDK 8.0 .....</b>	11
1. Understanding the Operating System Requirements.....	11
2. Optional ... Enabling Hyper-V.....	12
Unconventional Installs.....	15
<b>Part 3: Writing your First Windows Phone 8 App .....</b>	17
1. Create a new Windows Phone App project, name it "PetSounds" .....	17
2. Delete unnecessary comments to more easily navigate through the code .....	19
3. Add a Button control to the ContentPanel and style it .....	23
4. Add a MediaElement control .....	26
5. Add a wav file as an asset to the project .....	27
6. Add an event handler for the Button click event.....	29
7. Run the app.....	31
Recap.....	32
<b>Part 4: Introduction to XAML.....</b>	34
1. What is XAML? .....	34
2. Introducing Type Converters .....	37
3. Understanding XAML Namespace Declarations .....	38
4. Understanding the relationship between the .xaml and .xaml.cs files .....	41
5. Understanding Default Properties .....	42
6. Understanding Complex Properties and the Property Element Syntax .....	42
Recap.....	46
<b>Part 5: Basics of Layout and Events .....</b>	48
1. Understanding the Basics of Grids .....	48
2. Grid RowDefinitions and ColumnDefinitions and defining sizes .....	48
3. Grid cell Alignments and Margins .....	52
4. StackPanel basics .....	53
5. Understanding Events.....	56
Recap.....	60
<b>Part 6: Styling the App .....</b>	61
1. Change out the app's tile icons .....	61
2. Modifying the App and Page Titles .....	67

3. Understanding Binding Syntax and Static Resources.....	69
4. Discovering Theme Resources .....	71
5. Customizing a Theme Resource by creating a style based on it .....	74
Recap.....	78
<b>Part 7: Localizing the App.....</b>	80
1. Modify the AppResources.resx settings and bind to its values .....	80
2. Add support for a second language .....	82
3. Test the different language version of the app .....	86
Recap.....	90
<b>Part 8: Understanding Compilation and Deployment.....</b>	92
1. Discovering what happens during compilation and deployment.....	92
3. Deploying to a physical phone .....	98
4. Obtaining a Windows Phone Dev Center membership .....	108
Recap.....	109
<b>Part 9: Overview of the Windows Phone 8 Emulator .....</b>	110
1. What is the Windows Phone Emulator? .....	110
2. Choosing different versions of the Emulator for debugging .....	110
3. Working with the Phone Emulator's special features.....	113
Recap.....	124
<b>Part 10: Overview of the Databound App and Pivot App Project Templates .....</b>	126
1. Understanding the Windows Phone Databound App project template's functionality.....	126
2. Understanding the Windows Phone Pivot App project template's functionality.....	136
Recap.....	140
<b>Part 11: Setting up the SoundBoard App .....</b>	141
1. Sketch out the screens in a low-tech mockup .....	141
2. Create the SoundBoard project based on the Windows Phone Pivot App project template .....	143
3. Replace project assets.....	144
4. Confirm the project icons are properly referenced.....	145
5. Configure the title of main page to pull from the AppResources.resx file .....	148
Recap.....	149
<b>Part 12: Improving the View Model and Sample Data .....</b>	150
1. Analyze our mockup and design a data model.....	150
2. Create the new data model classes for our app .....	151

3. Modify the App.xaml.cs to use the new data model.....	156
4. Create sample / design-time display data .....	158
5. Use the new sample data in the MainPage.xaml.....	162
Recap.....	165
<b>Part 13: Styling Tiles in the LongListSelector.....</b>	166
1. Change the LongListSelector's LayoutMode to Grid.....	166
2. Modify the DataTemplate to create the layout we need .....	167
3. Adding the DataTemplate's design to the Page's Resources so that we can re-use it .....	171
Recap.....	176
<b>Part 14: Binding to Real Data at Runtime.....</b>	177
1. Adding real run-time data to our app.....	177
2. Fixing a data binding problem with the PivotItem Header.....	184
Recap.....	187
<b>Part 15: Playing a Sound when a ListItem is Selected.....</b>	189
1. Add a MediaElement to the MainPage.xaml .....	189
2. Handle the LongListSelector_SelectionChanged event .....	189
3. Fixing select selection problem.....	191
Recap.....	192
<b>Part 16: Working with the Application Bar .....</b>	193
1. Enable the boilerplate BuildLocalizedApplicationBar() method .....	193
2. Modifying our Application Bar Button and Text .....	194
3. Removing the old data model.....	198
4. Responding to the Click event of the App Bar's Record button .....	200
Recap.....	202
<b>Part 17: Introducing the Coding4Fun Toolkit .....</b>	203
1. Install the Coding4Fun Package .....	204
2. Employing the AboutPrompt .....	208
Recap.....	210
<b>Part 18: Navigating Between Pages.....</b>	211
1. Revisiting the Databound Project template to learn about navigation .....	211
2. Create the RecordAudio.xaml Page .....	217
3. Implement the code to navigate to the new page .....	218
Recap.....	219

<b>Part 19: Setting up the RecordAudio.xaml Page .....</b>	220
1. Perform simple branding changes .....	220
2. Perform basic layout of primary controls .....	221
3. Add an App Bar .....	224
Recap.....	226
<b>Part 20: Recording an Audio Wav File .....</b>	227
1. Modify the ToggleButton Control wiring up Event Handler Methods.....	227
2. Create a private instance of the Coding4Fun.Toolkit.Audio.MicrophoneRecord class .....	227
3. Saving the sound data collected by the MicrophoneRecorder into a file .....	228
4. Add a MediaElement to play the new temporary file .....	233
5. Handle the Play button's Click Event to Test the Sound .....	233
6. Declaring a Microphone capability for the app .....	234
7. Add defensive programming statements to guard against potential exceptions .....	236
Recap.....	240
<b>Part 21: Permanently Saving the Audio Wav File .....</b>	241
1. Add an event handler method to the "save" button and manage application bar state .....	241
2. Use the Coding4Fun Toolkit to display an InputDialog to capture the name of the new custom sound audio file.....	243
3. Save the sound file into a permanent IsolatedStorage area, serialize the data for the CustomSounds into a JSON file .....	246
4. Serialize and deserialize the CustomSounds SoundGroup into / out of Json.....	248
Recap.....	256
<b>Part 22: Animating the Reel Grid with a Storyboard .....</b>	257
1. Declaratively define the animation.....	257
2. Programmatically start and stop the animation.....	260
3. Deploy to physical Phone Device .....	262
Recap.....	263
<b>Part 23: Testing and Submitting to the Store .....</b>	264
1. Run the Store Test Kit .....	265
2. Submitting the app to the Store .....	282
Recap.....	299
<b>Part 24: Getting Started with the AroundMe Project .....</b>	300
1. Create a low-tech mockup for the AroundMe app.....	300
2. Create the AroundMe Solution and Project .....	301

3. Add a Map Control from the Toolbox to the visual XAML Editor .....	302
Recap.....	309
<b>Part 25: Working with the Geolocator and Geoposition Classes .....</b>	<b>310</b>
1. Modify UpdateMap() to retrieve it's position from the Geolocator class .....	310
2. Use the Emulator's Additional Tools to change the virtual location of the Emulator for testing ....	312
Recap.....	315
<b>Part 26: Retrieving a Photo from Flickr's API .....</b>	<b>316</b>
1. Become familiar with the flickr API site, sign up for developer access .....	316
2. Learning about Flickr's search API .....	322
3. Setting up our project to use the Flickr API .....	327
4. Programmatically calling the Flickr API via HTTP .....	328
5. Deserializing the JSON result into classes.....	335
Recap.....	344
<b>Part 27: Navigating and Passing Data to the SearchResults Page .....</b>	<b>345</b>
1. Add AroundMe assets to project .....	345
2. Make sure assets appear in the WMAppManifest.xml correctly .....	347
3. Add an Application Bar and Search Button.....	349
4. Navigate to a new page, pass data to the page .....	350
5. Show the Application Bar.....	351
6. Add a new item to the project, call it SearchResults.xaml .....	352
7. Validate that the values are being passed correctly between the MainPage and SearchResults page .....	353
8. Refactor the classes into a separate file .....	355
9. Implement the FlickrImage.cs class .....	359
8. Refactor the layout and display logic.....	363
Recap.....	369
<b>Part 28: Understanding Async and Awaitable Tasks.....</b>	<b>370</b>
Recap.....	372
<b>Part 29: Filtering the Results by Keyword .....</b>	<b>373</b>
1. Edit MainPage.xaml: clean up layout, add Search text box.....	373
2. Modify the Navigation code to pass Topic to SearchResults.xaml .....	374
3. Modify the Navigation code to RECEIVE the search topic in the SearchResults.xaml.....	375
4. Adding radius and programming defensively .....	379

Recap.....	384
<b>Part 30: Adding a Progress Indicator .....</b>	<b>385</b>
1. Understanding the Progress Indicator .....	385
2. Create a new ProgressIndicator() object and set it as the SystemTray's current ProgressIndicator. .....	386
3. Create a helper method to show / hide the ProgressIndicator .....	386
4. Modify the UpdateMap() method to use the ProgressIndicator and helper method.....	386
Recap.....	388
<b>Part 31: Multiple Selection with the LongListMultiSelector .....</b>	<b>389</b>
1. Install the Windows Phone toolkit package and review the samples .....	389
2. Install the NuGet package into the project.....	399
3. Implement the LongListMultiSelector .....	402
Recap.....	405
<b>Part 32: Animating Image Search Results.....</b>	<b>406</b>
1. Prepare the Image control for a fade-in animation.....	406
2. Implement the Image_ImageOpened Event Handler .....	407
3. Add a TextBlock for "No Photos Found", a Progress Bar and TextBlock for "Loading" .....	410
Recap.....	414
<b>Part 33: Working with the Lock Screen to Display an Image .....</b>	<b>415</b>
1. Add an App Bar .....	415
2. Write display logic in the SelectionChanged event handler of the LongListMultiSelector.....	417
3. Write code to handle the Set Application Bar Button's Click event .....	419
4. Add the LockScreenHelper.cs Class File and write the desired functionality .....	421
Recap.....	436
<b>Part 34: Creating a Background Agent for Scheduled Tasks .....</b>	<b>437</b>
1. Add a new Windows Phone Scheduled Task Agent project to our solution called AroundMe.Scheduler.....	438
2. Create a new Phone Class Library called AroundMe.Core and move class files to it .....	443
3. Add reference from AroundMe to AroundMe.Core and fix any lingering problems from the refactoring.....	447
4. Create a reference from AroundMe.Scheduler to AroundMe.Core and write code to set background lock screen image.....	451
5. Modify the AroundMe project's App.xaml.cs to introduce the background scheduled task agent to the operating system .....	454

5. Run the app in the Phone Emulator to make sure it works.....	459
6. Clean up the code removing development only code, adding other details before submitting to the Store.....	462
Recap.....	465
<b>Part 35: Where to go from here.....</b>	<b>467</b>

# Part 1: Series Introduction

Hello and welcome to this 35-lesson series on building apps for the Windows Phone 8 platform. My name is Bob Tabor, and for the past 11 years I've been creating screen cast training for Microsoft's developer-centric tools and technologies, both on Microsoft's web properties and my own web site, [www.LearnVisualStudio.NET](http://www.LearnVisualStudio.NET).

According to the title of this series, this training is for "absolute beginners", and while that is definitely true—we will begin with the very basics of building phone apps—you'll see that we quickly move into utilizing some of the new and advanced features on the Windows Phone 8 operating system.

This series is made possible due to the positive response to my original series, Windows Phone 7 Development for Absolute Beginners. We've redesigned this series completely—so if you watched that series, you'll not recognize a single thing in this new series.

**Before watching this series, you should already be familiar with C#.** If you're not, please put this series on the back-burner for a few days and instead watch [Channel9's C# Fundamentals for Absolute Beginners](#). I designed that series with you, the absolute C# novice, in mind. At a minimum, you'll need to get the basics of object oriented programming —classes, properties, methods, visibility modifiers, collections, generic collections, and the like—under your belt before attempting this series.

We approached this series of lessons as a tutorial...in other words, the series teaches how to build apps by walking through the steps required to build two full featured apps. Hopefully this approach will help you see how concepts fit together in a real app scenario. I'll also build a number of tiny apps just to illustrate some small concepts in hopes of clarifying fundamental ideas.

I'm also going to cover things like the operating system and hardware requirements, the software you'll need to install to get started, getting a developer license, designing an app, submitting to the store, and more. So, hopefully, this is a great starting point for developers new to Windows Phone 8 development.

Before I show you the apps we'll build in this series, we need to do a little house keeping. On screen right now is my desktop. There's code loaded into visual studio. The videos are recorded in a high def format, 720p, and are crystal clear. So, if you can't read the text on my screen, that could be because your internet connection can't handle streaming at a high bit rate. Your best option is to use the download link beneath each video. You can download various formats and resolutions based on the target device you want to watch these videos on.

Next, to follow along, you'll want to download the assets that are contained in a zip file. I'll make sure the link is available on this page, and every page where the videos are hosted. It will contain assets you'll need to include in your projects as well as the

finished versions of the apps so that you can compare the code I've written with the code you're working on.

Finally, for the first time on Channel9, I'm including a text and screenshot version of the videos...you'll find it posted below each video...while these are not necessarily a word-by-word transcript of what I say in the videos, they do cover the exact same material and provide the code that I type in so you can copy and paste it into your app. I'm providing this for those that have a hearing disability and for those who don't use English as their primary language. Also, it should be helpful for reference purposes so that you don't have to go back through and re-watch the videos to recall some previously covered idea or technique.

Ok, so what are we going to build in this series?

- SoundBoard app demonstration
- AroundMe app demonstration

While I'm the voice you'll be listening to for the next 11 hours, this effort was actually a collaboration between a number of parties. First and foremost, Clint Rutkas of Channel9 is the mastermind behind the two apps we'll be building in this series. I think we literally had 100 email threads running about various nuances of the code, and he was patient and very helpful and really deserves the lion-share of the credit for this series. The Windows Phone Team supported this effort and made it happen...I think that was due in large part to the warm reception the previous version of this series received from you, the loyal Channel9 viewer. So, thank you!

And finally, Nokia and their Developer Ambassadors were very helpful in helping me secure the assets I needed to put this together. Nokia has stepped up and supported the Windows Phone 8 platform and DEVELOPERS on the platform and I've been nothing but impressed with their passion for what they do. Check out their website:

<http://www.dvlup.com>

They offer one-on-one support, frequent meet-ups, contests with prizes and more to get developers like you and me more engaged and thinking about working together to build this platform. You should register with their site [www.dvlup.com](http://www.dvlup.com) to get involved.

And that brings me to this—I'm in love with my Nokia Lumia 920. It is without a doubt the coolest device I've ever owned—and trust me, I own several of the most popular devices built on other platforms available on the market today. If you're interested in Windows Phone 8 development, it's not a requirement, but I think you will really want to own one of these phones. It's not just a great developer testbed for the apps you build, but it's a great DEVICE. Let me tell you about my favorite features:

- Great camera—My wife is constantly asking me to send her the pics I take with my phone because her i...her, um, less capable phone...just doesn't even compare, especially in low light situations.
- NFC—Near Field Communications...exchange data with other NFC phones regardless of platform.
- Pinning—You can pin anything to the start page...music, websites, apps...and I seek out apps that update the tiles with new information so I don't have to open up the app, for example, to see the weather, or my calendar, or the countdown to my vacation.
- Wireless charging—Yes, you can buy a case for just about any phone that will do this, but it's built INTO THE PHONE!
- Voice commands—Great for creating OneNote TODO items, or sending a text message while I'm on the road.

The other cool thing is that the platform is growing...every time I do a demo of my phone, I convert another user. I've got my family and friends convinced that this is their next phone when their contract comes up for renewal. And I just read an article about the growth of the enterprise market for app builders, which is really exciting.

<http://www.windowsphone.com/en-US/business/for-business>

And the best feature of all? At least in my opinion? The fact that I can leverage my existing C#, .NET and Windows Runtime experience into building apps that I can carry around with me.

Yes, I suppose if I wanted to create apps for another platform, I could spend weeks and months learning a new language, a new API, etc. Or, I could build apps that try to target all the platforms but ultimately miss key new features unique to the Windows Phone 8 operating system. But this feels natural and easy and so it's fun.

So, if you're just getting started with Windows Phone 8 development, I'm sure you'll soon share my excitement. This series is one of the best ways to get up to speed quickly.

Promise me this—if you get stuck or something doesn't make sense, you'll ask a question in the comments area at the very bottom of the page. Either Clint or I...or perhaps someone else who's working through the material...will try to help get you unstuck and moving forward.

So, let's get started in the next lesson setting up our environment, and then we'll begin writing code immediately. We'll see you in the next lesson.

**Source Code For Entire Series:** <http://aka.ms/absbeginnerdevwp8>

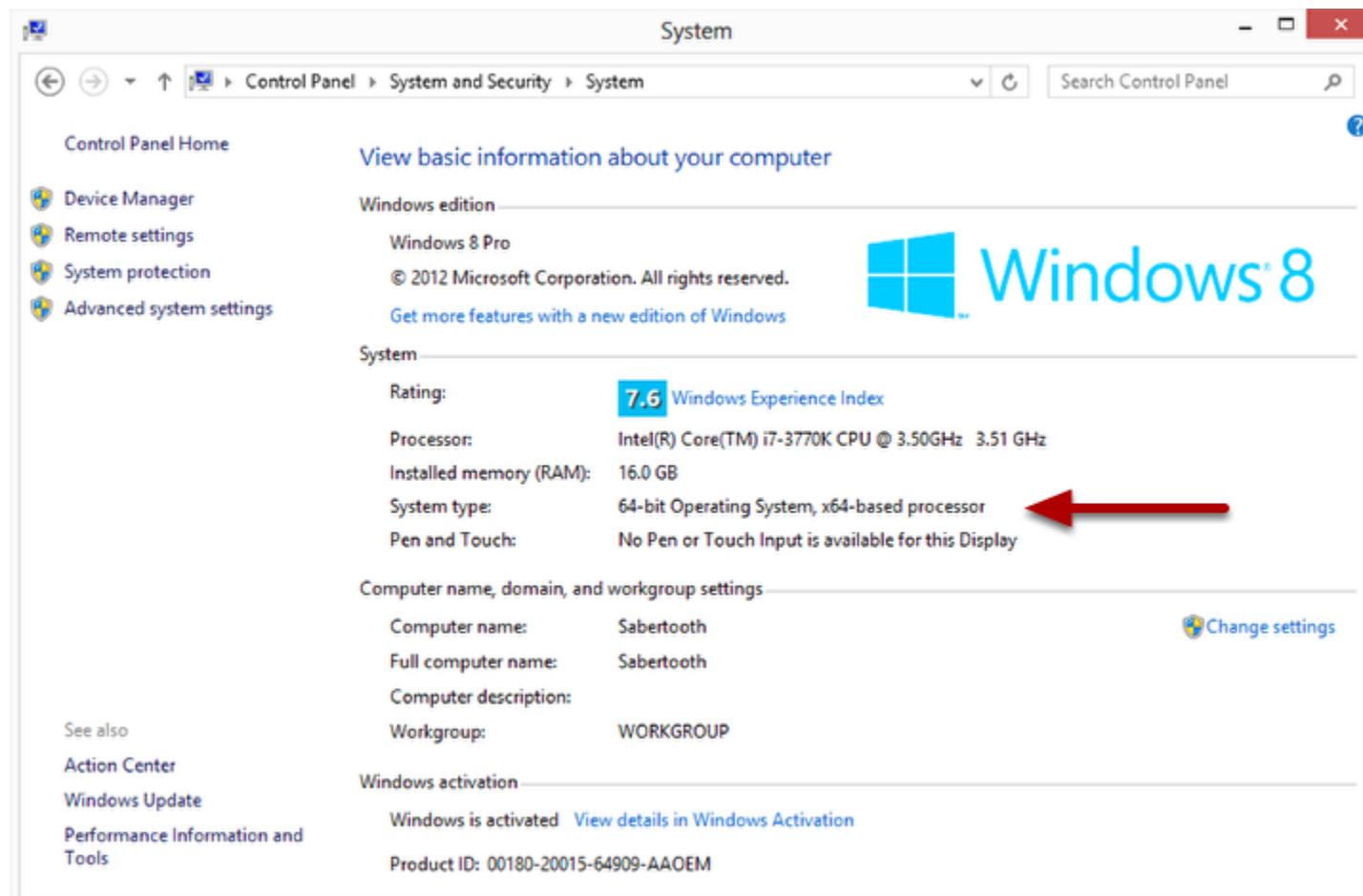
## Part 2: Installing Windows Phone SDK 8.0

Source Code: <http://aka.ms/absbeginnerdevwp8>

Before you can develop a Windows Phone 8 app, you'll need to install the Windows Phone 8 SDK (Software Development Kit) on a computer running Windows 8 64-bit edition. The reason for this requirement is the Windows Phone Emulator ... it is running as a virtual machine in Hyper-V, Microsoft's virtualization platform. So, you'll be running the Windows Phone 8 operating system in a window that looks like a real phone, on your desktop for the purpose of testing your work.

### 1. Understanding the Operating System Requirements

So to be clear, if you're not sitting at a computer with Windows 8 64-bit edition at this moment, you'll need to install that first. If you're not sure which version of Windows 8 your computer is running, you can go to the Control Panel, System and Security ... under System you'll see the System type:



TIP: I'm going to save you some time and expense. If you're running a previous version of Windows and it is a 32-bit edition, you cannot simply use Windows Upgrade Advisor

by visiting: <http://windows.microsoft.com/en-us/windows/buy> to electronically upgrade from a previous version of a Windows 32-bit operating system to Windows 8 64-bit. Instead, you'll need to purchase the Windows Pro Upgrade DVD.

For a more thorough explanation of this and every possible scenario imaginable, check out Paul Thurrott's post at this URL:

<http://winstersite.com/article/windows8/windows-8-upgrade-32bit-64bit-144649>

In my case, I purchased the OEM version of Windows 8 64-bit Pro. I was concerned that I couldn't do a clean install using the Upgrade, however that apparently is not the case and I could have saved a few bucks with the Upgrade option.

TIP: Also, Windows 8 64-bit is NOT the same as Windows 8 Pro. There's actually:

- Windows 8 64-bit Pro
- Windows 8 64-bit
- Windows 8 32-bit Pro
- Windows 8 32-bit

For the purpose of developing Windows Phone apps, you don't need the Pro option, just make sure whatever you choose is 64-bit.

Next, you download and install the Windows Phone SDK 8. If you already have Visual Studio 2012 Professional or greater, the installer will merely add the tools required for Phone development. If you DON'T have Visual Studio 2012 installed, the installer will add the Visual Studio 2012 Express for Windows Phone 8. This will provide a single-task version of Visual Studio meant specifically for Phone 8 development, so you won't get the tools to create Windows 8 Store apps, Windows Presentation Foundation apps, ASP.NET web apps, and so on. I'll be using this version for the remainder of this video series, but I assure you the experience is almost identical to using Visual Studio 2012 Professional or greater.

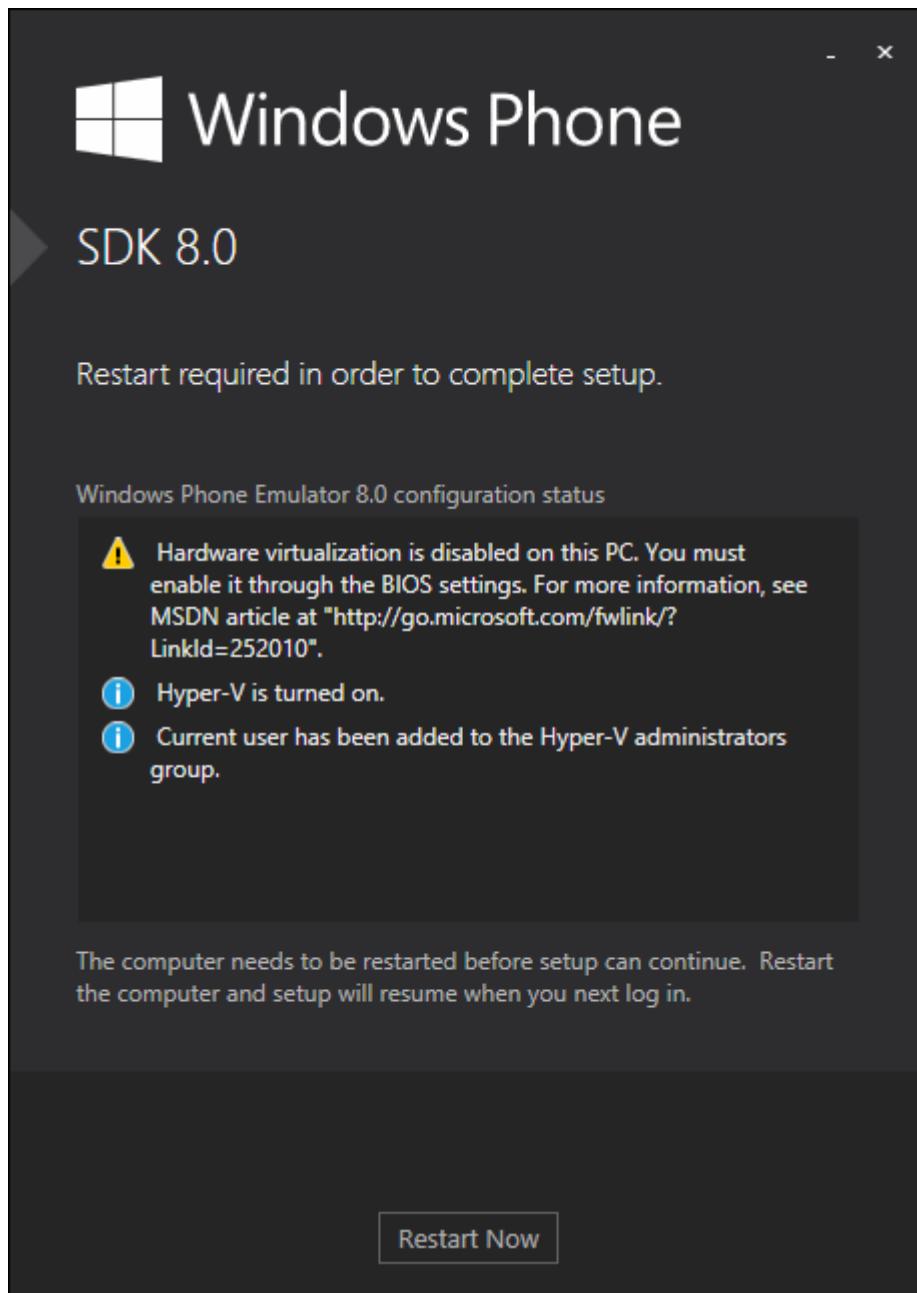
### Windows Phone SDK

<http://developer.windowsphone.com/en-us/downloadsdk>

I'm sure you're familiar with downloading and running an installer, so I won't walk through that process here.

### 2. Optional ... Enabling Hyper-V

During installation, you may see this message:



In this case you'll need to enable your motherboard to run Hyper-V.

How to enable Hyper-V for the Windows Phone Emulator

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj863509\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj863509(v=vs.105).aspx)

In my case, I recently built my own machine and used the Asus Sabertooth Z77, which is a high-end military grade motherboard. I used the latest Intel i7 chip—the i7-3770K which fits in an LGA1155 socket. Most importantly, it supports Intel's Hyper-Threading Technology ... I just have to tell the motherboard to turn that functionality ON.

In the BIOS for my motherboard, I had to enable Hyper-V by going to Advanced Settings, then the Advanced tab, then looking through the possible settings. In my case, it was something called Hyper-threading.

This may sound scary, but it's a one-time change and after I worked past the terminology and how to actually get into your BIOS, it all went smoothly:



What I would recommend is this—if you're not sure how to do this for your particular brand of computer, just run the SDK installer and follow its instructions. It's possible you'll need to do nothing special. If you DO need to do something, it's very possible someone else in the world with the same exact computer that you have has worked through this issue and blogged about it. Here's where good searching skills with a search engine like [www.bing.com](http://www.bing.com) is an invaluable skill. A few minutes—heck, even an hour of reading and researching—can save you headaches. As a last resort, contact the manufacturer of your computer to simply ask how to enable hyper-threading in the

BIOS. They should be able to point you to a knowledge base article on how to perform this operation.

### Unconventional Installs

But as the old expression goes, there's more than one way to skin a cat ... I was able to successfully install the Windows Phone SDK 8 and Visual Studio 2012 Express for Windows Phone on a Mac Pro running OSX Mountain Lion and VMWare Fusion. I just made sure when I created the VM that it was 64-bit and prior to installing the operating system, I setup the Processors and Memory in VMWare Fusion:



... I gave it plenty of cores and memory, then made sure that "Enable hypervisor applications in this virtual machine" was checked. I can't remember if I needed the second option, "Enable code profiling applications ion this virtual machine" checked ... it's been a month since I set this up ... but since it works with this turned on, I recommend this, too:

The screenshot shows the VMWare configuration interface for a virtual machine named "Windows Phone 8: Processors & Memory".

**Processors:**  
This virtual machine is configured to use:  
4 processor cores

**Memory:**  
Increasing the memory allocation of the virtual machine can improve performance but may also reduce the performance of other running applications.  
Current value: 8192 MB (with a shield icon)  
Recommended value: 2048 MB

**Advanced options:**

- Enable hypervisor applications in this virtual machine  
Enables running modern virtualization applications by providing support for Intel VT-x/EPT inside this virtual machine.
- Enable code profiling applications in this virtual machine  
Enables running modern code profiling applications by providing support for CPU performance monitoring counters inside this virtual machine.

A red box highlights the "Advanced options" section.

I only bring up the Mac with VMWare for this reason ... if there's a will, there's a way. What seems hard is usually pretty easy ... just need to know which options to configure. Again, [www.bing.com](http://www.bing.com) can be your friend in cases like these.

After you work through these requirements, you're ready to get started and follow along in this series. So, we'll begin in the next video.

# Part 3: Writing your First Windows Phone 8 App

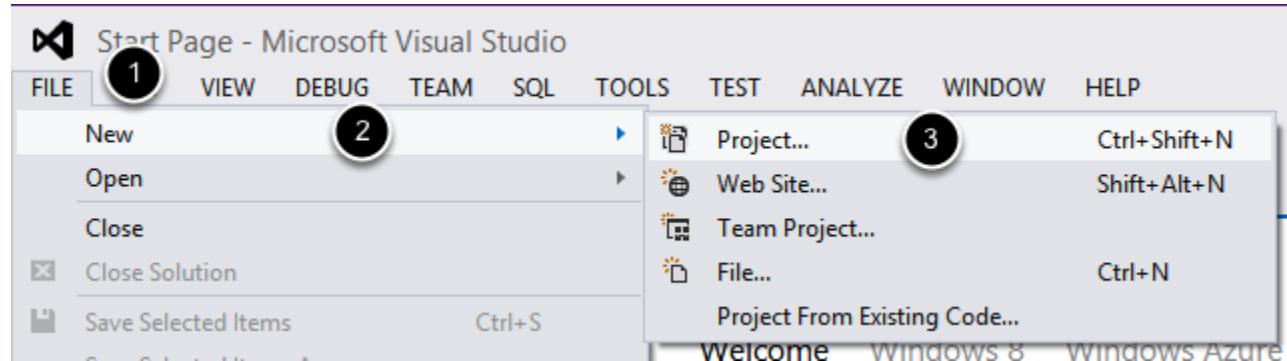
Source Code: <http://aka.ms/absbeginnerdevwp8>

Now that we have the tools we need installed, we can build our first Windows Phone 8 app.

Here's our game plan:

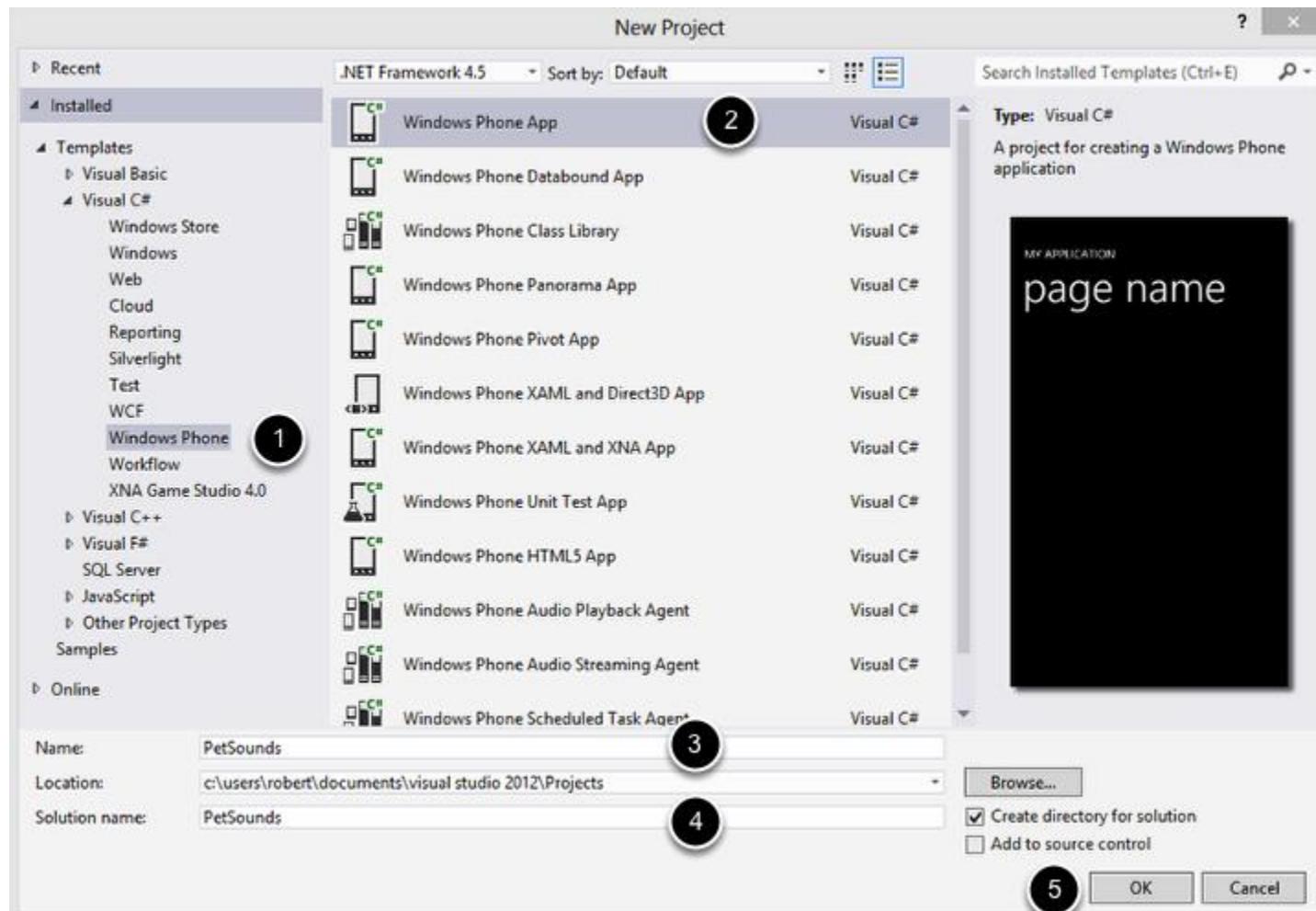
1. We'll create a new Windows Phone App project
  2. We'll make some simple edits, like removing comments and adding a MediaControl and a Button control and we'll style it up
  3. We'll write an event handler that will respond to the click event of the Button
  4. In the Button click event handler we'll play a wav sound file
1. Create a new Windows Phone App project, name it "PetSounds"

Hopefully some of the basics steps like creating new projects, adding new files and so on are already familiar to you from personal experience or from watching other Absolute Beginner's series on Channel9. I won't take much time to explain those ... if something is unfamiliar, you may want to review the C# for Absolute Beginners series:



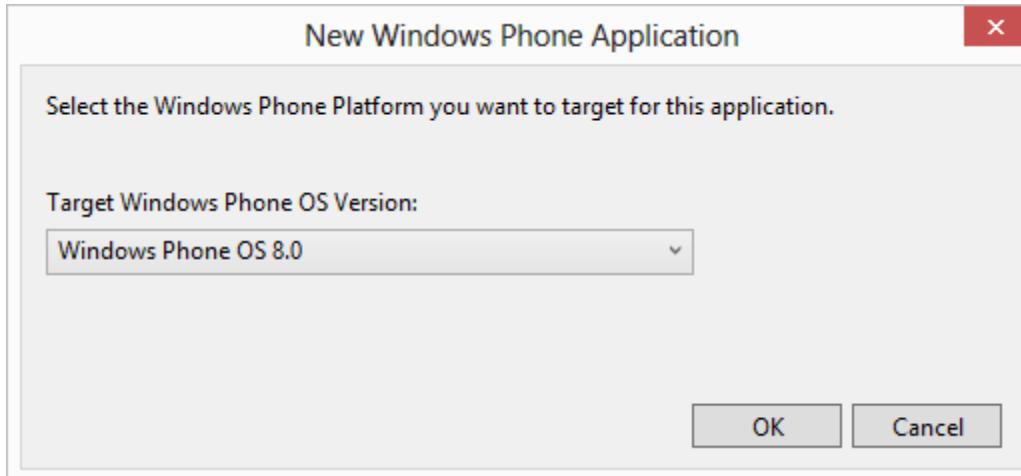
1. File
2. New
3. Project ...

In the New Project dialog:



1. Make sure you're in the Windows Phone project templates
2. Select the Windows Phone App project template
3. Rename to: PetSounds
4. Make sure the name of the Solution has changed to "PetSounds" as well
5. Click OK

You may see the following dialog:

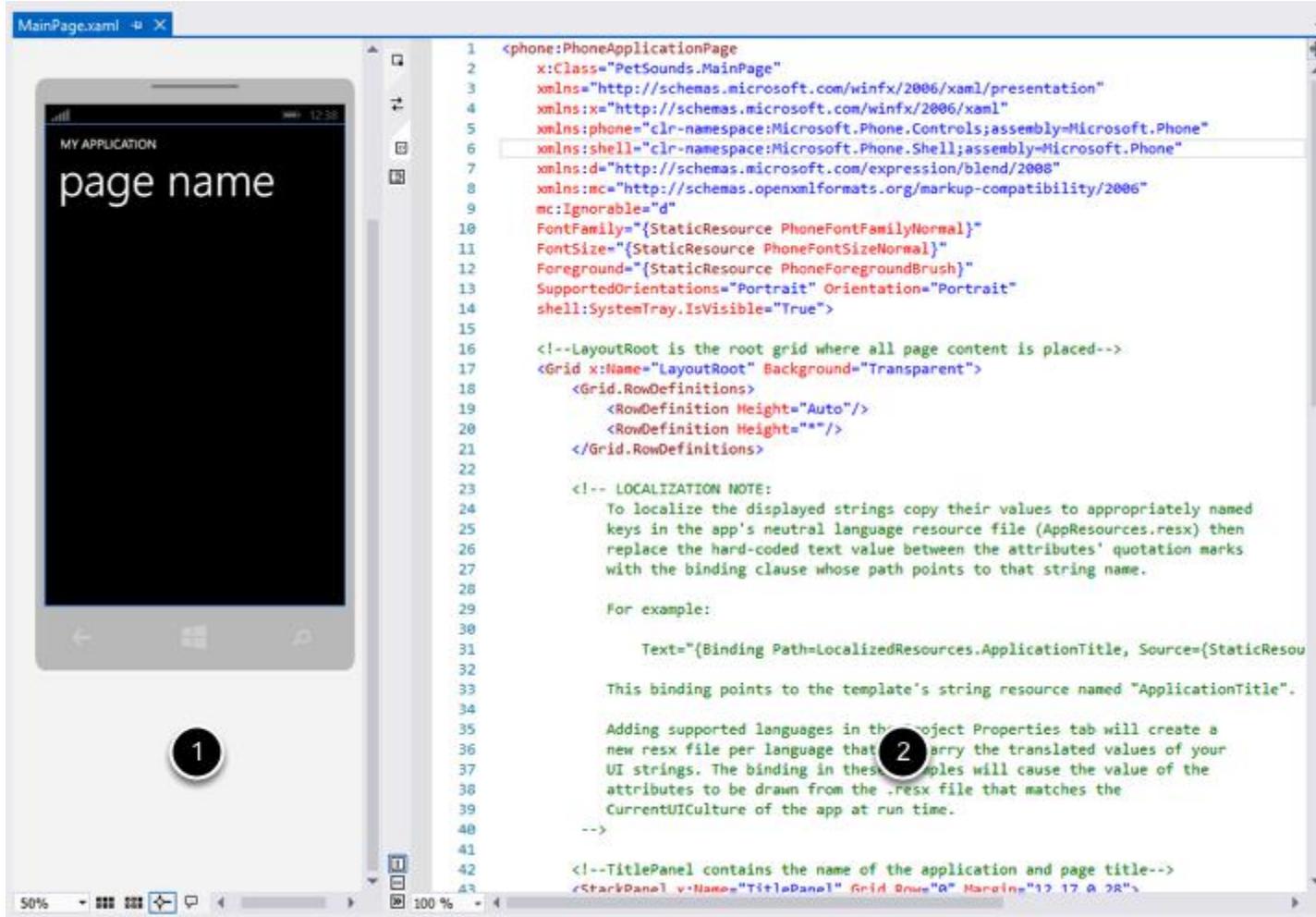


Since this series is only targeting the Windows Phone OS 8.0, select this option. Just know that you can create apps that target previous versions of the Windows Phone operating system in Visual Studio if you want to expand your apps reach to owners of older Windows Phone devices.

## 2. Delete unnecessary comments to more easily navigate through the code

After a few moments, the new Project will be created and loaded into the Solution Explorer, and the MainPage.xaml will be visible in the main area of Visual Studio. Notice that these "screens" are called "Pages", as in MainPage.xaml. In our first apps we'll only work with a single "page", but in other apps we'll add pages and navigate between them.

I'll assume you've never worked with a Windows Phone project template before. You'll see the MainPage.xaml loaded into the visual designer:



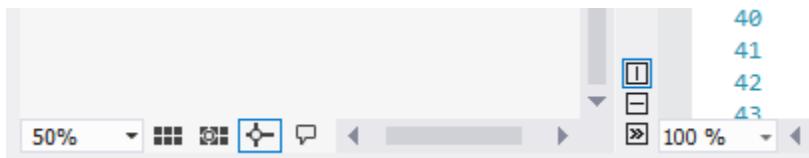
```

1 <phone:PhoneApplicationPage
2     x:Class="PetSounds.MainPage"
3     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5     xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6     xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
9     mc:Ignorable="d"
10    FontFamily="{StaticResource PhoneFontFamilyNormal}"
11    FontSize="{StaticResource PhoneFontSizeNormal}"
12    Foreground="{StaticResource PhoneForegroundBrush}"
13    SupportedOrientations="Portrait" Orientation="Portrait"
14    shell:SystemTray.IsVisible="True"
15
16    <!--LayoutRoot is the root grid where all page content is placed--&gt;
17    &lt;Grid x:Name="LayoutRoot" Background="Transparent"&gt;
18        &lt;Grid.RowDefinitions&gt;
19            &lt;RowDefinition Height="Auto"/&gt;
20            &lt;RowDefinition Height="*"/&gt;
21        &lt;/Grid.RowDefinitions&gt;
22
23        &lt;!-- LOCALIZATION NOTE:
24             To localize the displayed strings copy their values to appropriately named
25             keys in the app's neutral language resource file (AppResources.resx) then
26             replace the hard-coded text value between the attributes' quotation marks
27             with the binding clause whose path points to that string name.
28
29             For example:
30
31             Text="{Binding Path=LocalizedResources.ApplicationTitle, Source={StaticResou
32
33             This binding points to the template's string resource named "ApplicationTitle".
34
35             Adding supported languages in the project Properties tab will create a
36             new resx file per language that will carry the translated values of your
37             UI strings. The binding in these examples will cause the value of the
38             attributes to be drawn from the .resx file that matches the
39             CurrentUICulture of the app at run time.
40
41
42        &lt;!--TitlePanel contains the name of the application and page title--&gt;
43        &lt;StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28"&gt;
</pre>

```

1. On the left, you'll see the visual designer. While you can use this as your primary means of laying out and adding controls to your Windows Phone app, I use it primarily as feedback for what I do in the ...
2. XAML editor pane, on the right. I typically write all my XAML by hand. Changes I make in the XAML code will be reflected in the visual designer and vice versa. They are two perspectives, but represent the same thing.

There are a number of tools beneath the panes:



... and between the panes:



... feel free to experiment with them. They are self explanatory and you need to take a moment to play around with them and understand their usage. However, since it's not critical to our lesson, I want to move ahead. In the XAML pane, I want to remove two large comment passages so that it's easier to navigate through the XAML. I've identified these sections as follows:

```
1 <phone:PhoneApplicationPage
2     x:Class="petSounds.MainPage"
3     xmlns="http://schemas.microsoft.com/winfx/2009/xaml/presentation"
4     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
5     xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6     xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2009"
9     mc:Ignorable="d"
10    FontFamily="{StaticResource PhoneFontFamilyNormal}"
11    FontSize="{StaticResource PhoneFontSizeNormal}"
12    Foreground="{StaticResource PhoneForegroundBrush}"
13    SupportedOrientations="Portrait" Orientation="Portrait"
14    Shell:SystemTray.isVisible="True">
15
16    <!--LayoutRoot is the root grid where all page content is placed-->
17    <Grid x:Name="LayoutRoot" Background="Transparent">
18        <Grid.RowDefinitions>
19            <RowDefinition Height="Auto"/>
20            <RowDefinition Height="*"/>
21        </Grid.RowDefinitions>
22
23    <!-- LOCALIZATION NOTE:
24        To localize the displayed strings copy their values to appropriately named
25        keys in the app's neutral language resource file (AppResources.resx) then
26        replace the hard-coded text value between the attributes' quotation marks
27        with the binding clause whose path points to that string name.
28
29        For example:
30
31            Text="{binding Path=LocalizedResources.ApplicationTitle, Source={StaticResource LocalizedStrings}}"
32
33        This binding points to the template's string resource named "ApplicationTitle".
34
35        Adding supported languages in the Project Properties tab will create a
36        new resx file per language that can carry the translated values of your
37        UI strings. The binding in these examples will cause the value of the
38        attributes to be drawn from the .resx file that matches the
39        CurrentUICulture of the app at run time.
40    -->
41
42    <!--TitlePanel contains the name of the application and page title-->
43    <StackPanel x:Name="titlePanel" Grid.Row="0" Margin="12,17,0,28">
44        <Textblock Text="My APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0,0,0"/>
45        <Textblock Text="page name" Margin="0,-7,0,0" Style="{StaticResource PhoneTextTitleStyle}"/>
46    </StackPanel>
47
48    <!--ContentPanel - place additional content here-->
49    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
50
51    </Grid>
52
53    <!--Uncomment to see an alignment grid to help ensure your controls are
54        aligned on common boundaries. The image has a top margin of -32px to
55        account for the System Tray. Set this to 0 (or remove the margin altogether)
56        if the System Tray is hidden.
57
58        before shipping remove this xaml and the image itself.-->
59        <Image Source="/Assets/AlignmentGrid.png" VerticalAlignment="Top" Height="800" Width="480" Margin="0,-32,0,0" Grid.Row="2"/>
60    </Grid>
61
62 </phone:PhoneApplicationPage>
```

You can safely delete these comments. Be sure to delete everything from the starting:

```
<!--  
... to the ending ...  
-->
```

Your XAML should look like this:

```
1  <phone:PhoneApplicationPage  
2      x:Class="PetSounds.MainPage"  
3      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
5      xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"  
6      xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"  
7      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
8      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
9      mc:Ignorable="d"  
10     FontFamily="{StaticResource PhoneFontFamilyNormal}"  
11     FontSize="{StaticResource PhoneFontSizeNormal}"  
12     Foreground="{StaticResource PhoneForegroundBrush}"  
13     SupportedOrientations="Portrait" Orientation="Portrait"  
14     shell:SystemTray.IsVisible="True">  
15  
16     <!--LayoutRoot is the root grid where all page content is placed-->  
17     <Grid x:Name="LayoutRoot" Background="Transparent">  
18         <Grid.RowDefinitions>  
19             <RowDefinition Height="Auto"/>  
20             <RowDefinition Height="*"/>  
21         </Grid.RowDefinitions>  
22  
23         <!--TitlePanel contains the name of the application and page title-->  
24         <StackPanel x:Name="TitlePanel"  
25             Grid.Row="0"  
26             Margin="12,17,0,28">  
27             <TextBlock Text="MY APPLICATION"  
28                 Style="{StaticResource PhoneTextNormalStyle}"  
29                 Margin="12,0"/>  
30             <TextBlock Text="page name"  
31                 Margin="9,-7,0,0"  
32                 Style="{StaticResource PhoneTextTitle1Style}"/>  
33         </StackPanel>  
34  
35         <!--ContentPanel - place additional content here-->  
36         <Grid x:Name="ContentPanel"  
37             Grid.Row="1"  
38             Margin="12,0,12,0">  
39  
40             </Grid>  
41         </Grid>  
42  
43     </phone:PhoneApplicationPage>
```

We'll be focusing on the "ContentPanel" beginning in line 36 throughout this lesson adding new XAML code between the opening and closing `<Grid>` elements.

3. Add a Button control to the ContentPanel and style it

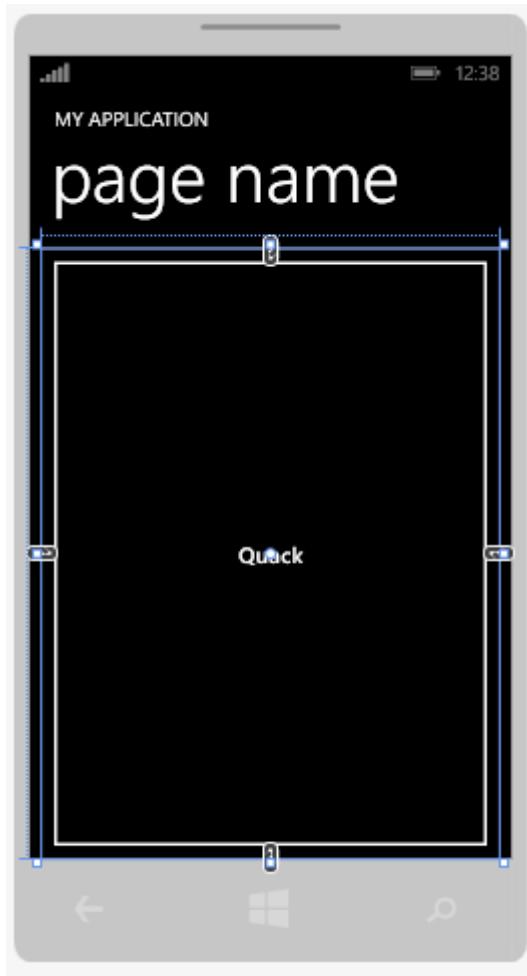
Add the following code between the opening and closing <Grid> elements:

```
35      <!--ContentPanel - place additional content here-->
36      <Grid x:Name="ContentPanel"
37          Grid.Row="1"
38          Margin="12,0,12,0">
39          <Button>Quack</Button>
40      </Grid>
41  -->
```

Once you add the:

```
<Button>Quack</Button>
```

... code, notice how the visual designer changes:



The button takes up practically the entire area of the screen. That won't do ... we need to limit the size of the Button by setting a Height and Width attribute:

```
34
35      <!--ContentPanel - place additional content here-->
36      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
37          <Button Name="_playAudio"
38              Height="200"
39              Width="200">
40                  Hello World!
41              </Button>
42          </Grid>
43
44      </Grid>
45
```

The value "200" infers "200 pixels".

The visual designer updates to a smaller button size:

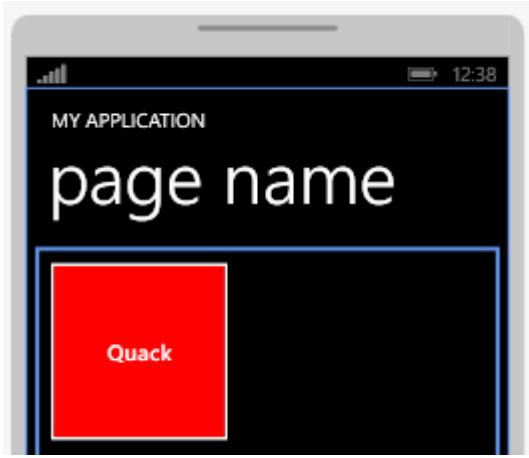


Let's move the Button control to the upper left-hand corner of the page (beneath the page title), and make its background color red:

```
39 <Button Name="PlayAudioButton"
40   Width="200"
41   Height="200"
42   HorizontalAlignment="Left"
43   VerticalAlignment="Top"
44   Background="Red">
45   Quack
46 </Button>
```

I also gave the <Button> control a name so that I can reference that control in C#. Giving controls names is optional. You only need to give things names that you want to access somehow in C#. I know I'll want to access the button later, so I give it a programmatic name now.

The visual designer updates to show the effect of our changes:



That's a great start. You can see that we can manipulate the visual qualities of objects on our Windows Phone page by setting its attributes.

#### 4. Add a MediaElement control

Next, let's add a MediaElement control to the XAML, beneath the Button control:

```
39 <Button Name="PlayAudioButton"
40     Width="200"
41     Height="200"
42     HorizontalAlignment="Left"
43     VerticalAlignment="Top"
44     Background="Red">
45     Quack
46 </Button>
47
48 <MediaElement x:Name="QuackMediaElement"
49     Source=""
50     />
51
```

Notice I can add whitespace and extra lines between my XAML code and it won't harm anything. As you'll learn later, Visual Studio will automatically indent and space the code for readability, however it will not impact how it is rendered on the page.

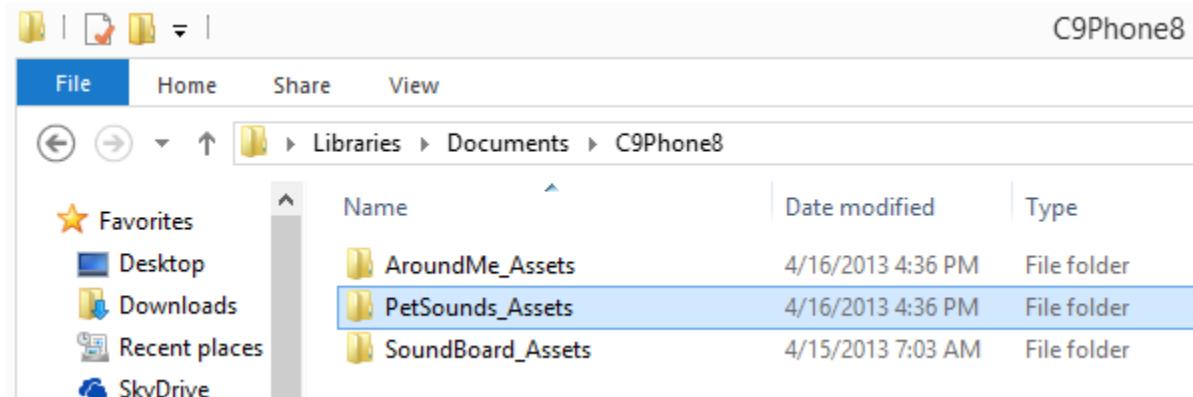
Also, I've not set the Source attribute of the MediaElement yet. That's because I do not have any sources (i.e., media elements, like sound files) in my project to choose from.

5. Add a wav file as an asset to the project

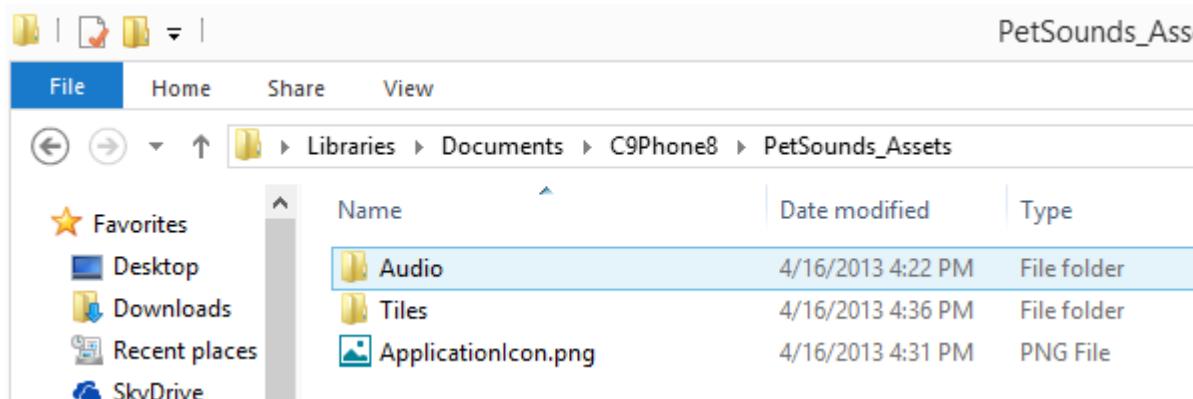
Make sure you've downloaded the assets the accompany this video. You can download them from the place where you downloaded this document, or are watching the videos.

I've unzipped that file into a folder called C9Phone8 in my Documents directory.

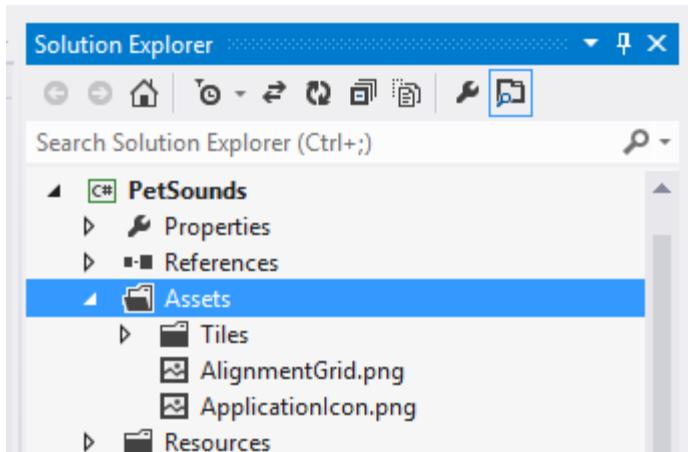
The C9Phone8 directory has 3 sub-directories:



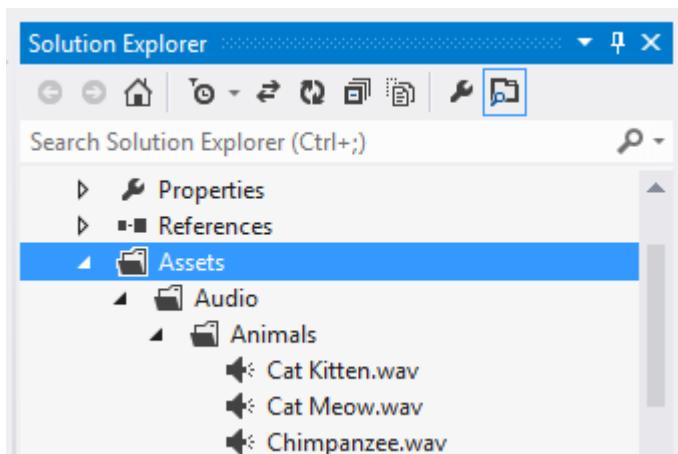
We want to use the assets contained in the PetSounds\_Assets folder. Inside that folder, there's two sub folders ... we want to copy the Audio sub-folder into our project.



The target for our Audio folder is the Assets subfolder of our project:



I drag and drop the Audio folder from Windows Explorer to the target directory in Visual Studio's Solution Explorer, specifically the Assets folder ... now my Assets folder looks like this as I expand everything out:



There are a number of .wav sound files in my app. I want to access one specific file ... a Duck.wav file. I'll add that as the value for the Source attribute in my MediaElement XAML element:

```
47
48      <MediaElement x:Name="QuackMediaElement"
49          Source="/Assets/Audio/Animals/Duck.wav"
50          Volume="1"
51          AutoPlay="False"
52
53
```

First, notice that I added the full subfolder path relative to the MainPage.xaml file, which lives in the project's root directory.

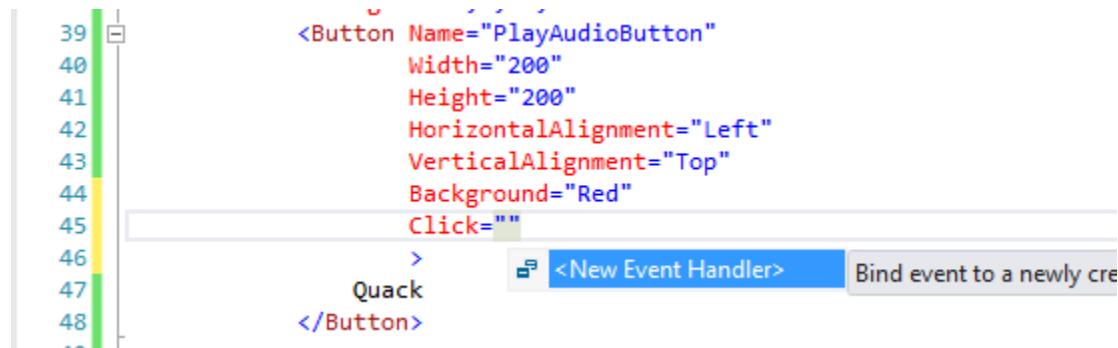
Second, notice that I also have added two more attributes ... I set Volume to "1" which means the loudest. Many settings in the Windows Phone API are 0.0 for the smallest and 1.0 for the largest value, with 0.5 somewhere in between. Most of these values are the C# data type Double.

Third, I will want to access this control programmatically to trigger the playback of the sound when someone taps the button, so I'll need to give the MediaControl a name. You may wonder what the x: prefix is for ... I'll explain that in the next lesson. Just keep that in the back of your mind for now.

Finally, I set the AutoPlay attribute to "False". If I set this to "True", the Duck.wav sound file would play immediately as my app loaded. That's not what I want. I want to trigger the wav file to play when I click the "Quack" button. We'll write code to accomplish that next.

## 6. Add an event handler for the Button click event

In the <Button> XAML element, I'll add the Click="" attribute. Visual Studio's Intellisense feature gives me the option to create a new event handler:



```
39 <Button Name="PlayAudioButton"
40     Width="200"
41     Height="200"
42     HorizontalAlignment="Left"
43     VerticalAlignment="Top"
44     Background="Red"
45     Click="">
46     Quack
47 </Button>
```

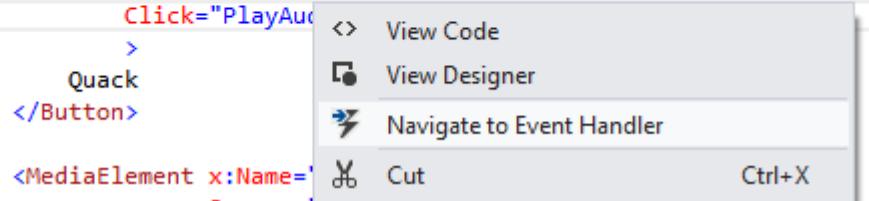
Making sure that option is highlighted, I'll hit the enter key on my keyboard to generate a name for the click event handler:

```
39 <Button Name="PlayAudioButton"
40     Width="200"
41     Height="200"
42     HorizontalAlignment="Left"
43     VerticalAlignment="Top"
44     Background="Red"
45     Click="PlayAudioButton_Click"
46     >
47     Quack
48 </Button>
```

I want to write code that will execute when someone clicks the button, so I'll want to navigate to the PlayAudioButton\_Click() method and write C# code there.

I right-click anywhere on that line of code and choose "Navigate to Event Handler" from the context menu:

```
39 <Button Name="PlayAudioButton"
40     Width="200"
41     Height="200"
42     HorizontalAlignment="Left"
43     VerticalAlignment="Top"
44     Background="Red"
45     Click="PlayAudioButton_Click"
46     >
47     Quack
48 </Button>
49
50 <MediaElement x:Name="mediaElement1" />
```



This opens up a file called MainPage.xaml.cs in the main area of Visual Studio.

If you're new to creating Windows, Web or now Phone apps in Visual Studio, you'll come to realize that there's two parts to these "pages" we're creating. The XAML and design view allows us to write declarative code (XAML). The related code view (the .cs file) allows us to define behavior in C#. These are two halves of the same concept. More on that later. In Visual Studio, your cursor should be located between the opening and closing curly braces for the PlayAudioButton\_Click() method:

```
24
25     private void PlayAudioButton_Click(object sender, RoutedEventArgs e)
26     {
27
28     }
```

Since this code block will execute whenever someone taps the Quack button on the phone, we'll want to trigger the MediaElement to play the sound we set in its Source attribute, namely, the Quack.wav file:

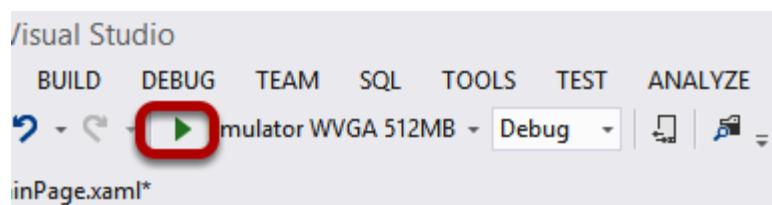
```
24
25     private void PlayAudioButton_Click(object sender, RoutedEventArgs e)
26     {
27         QuackMediaElement.Play();
28     }
```

I use the MediaElement's name, QuackMediaElement, to access it programmatically. I want to call its Play() method to kick off the playback of the Source, in other words, the Quack.wav file.

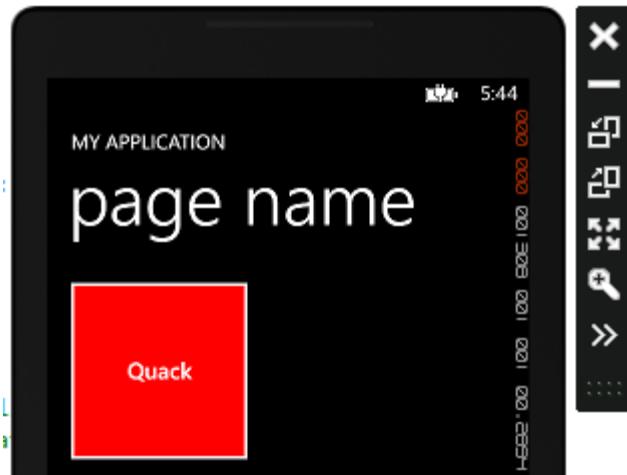
Now, let's test the app.

## 7. Run the app

Again, this should be familiar to you. You'll run the application in debug mode the same way you would run Console applications that we created in the C# Fundamentals series ... by either using the Play button in the toolbar OR using the F5 key on your keyboard.



What you see next is the Windows Phone Emulator. It's a virtual machine that is running the full Windows Phone 8.0 operating system. In other words, the operating system actually thinks it is running on a physical phone device, however it is "virtual" in the sense that Microsoft created software that mimics the phone hardware in every way. We'll be using the Phone Emulator extensively in this series as it's easier than deploying our tests to a physical phone each time we want to test the code we've written or a change we've made. You'll learn more about the features of the Phone Emulator in this series.



Use your mouse to simulate using your finger to tap the screen of the phone. You want to click on the red Quack button. If your computer is set up to hear audio, then you'll hear a duck's quack through your headset or speakers.

To stop debugging the app, click the red square button in the toolbar:



## Recap

To quickly recap, in just one lesson, we've created a simple sound board app. We learned how to create a Windows Phone project, how to modify the declarative XAML code to add and configure controls. We learned how to add assets to the project and reference them in our code, and how to add event handlers to respond to certain events that are triggered by the end user. We learned how the `MainPage.xaml` and the `MainPage.xaml.cs` are related ... we'll learn more about that in the next lesson. We learned how to trigger methods of the controls to play the sound when a user taps the button. Finally, we learned about the Windows Phone emulator as a means of testing our apps in a virtual environment.

However, there's a lot to talk about ... if you're new to XAML, it's really important for you to have a solid foundation with it, so in the next lesson I want to talk about the features of XAML and build on those throughout the rest of this series. As I said at the outset, many of the lessons you learn here will transfer over to all the APIs that utilize XAML, such as the Windows Presentation Foundation and Windows Store apps.



# Part 4: Introduction to XAML

Source Code: <http://aka.ms/absbeginnerdevwp8>

In this lesson, I want to talk about the XAML syntax we wrote in our first pass at the SoundBoard app. Hopefully you could see how the XAML we wrote impacted what we saw in the Phone preview pane. It's relatively easy to figure out the absolute basics of XAML just by looking at it, but I want to point out some of the features and functions that may not be obvious at first glance.

At a high level, here's our game plan in this lesson:

1. We'll talk about the purpose and nature of XAML, comparing it to C#
2. We'll talk about the special features of XAML ... little hidden features of the language that may not become obvious by just staring at it

My aim is by the end of this lesson you'll have enough knowledge that you can look at the XAML we write in the remainder of this series and be able to take a pretty good guess at what it's doing before I even try to explain it.

## 1. What is XAML?

In the previous lesson, I made a passing remark about XAML and how it looks similar to HTML. That's no accident. XAML is really just XML, the eXtensible Markup Language. I'll explain that relationship in a moment, but at a higher level, XML looks like HTML insomuch that they share a common ancestry. Whereas HTML is specific to structuring a web page document, XML is more generic. By "generic" I mean that you can use it for any purpose you devise and you can define the names of the elements and attributes to suit your needs. In the past, developers have used XML for things like storing application settings, or using it as a means of transferring data between two systems that were never meant to work together. To use XML, you define a schema, which declares the proper names of elements and their attributes. A schema is like a contract. Everyone agrees—both the producer of the XML and the consumer of the XML abide by that contract in order to communicate with each other. So, a schema is an important part of XML. Keep that in mind ... we'll come back to that in a moment.

XAML is a special usage of XML. Obviously, we see that, at least in this case, XAML has something to do with defining a user interface in our Phone's interface. So in that regard, it feels very much like HTML. But there's a big difference ... XAML is actually used to create instances of classes and set the values of the properties. So, for example, in the previous lesson we defined a Button control in XAML:

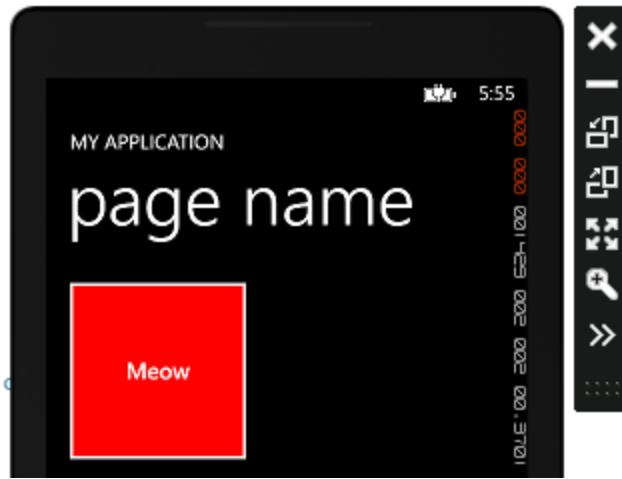
```
38           Margin="12,0,12,0"/>
39           <Button Name="PlayAudioButton"
40               Width="200"
41               Height="200"
42               HorizontalAlignment="Left"
43               VerticalAlignment="Top"
44               Background="Red"
45               Click="PlayAudioButton_Click"
46           >
47               Quack
48           </Button>
```

... that line of code is roughly equivalent to this in C#:

```
17     // Constructor
18     public MainPage()
19     {
20         InitializeComponent();
21
22         Button myButton = new Button();
23         myButton.Name = "MeowButton";
24         myButton.Height = 200;
25         myButton.Width = 200;
26         myButton.VerticalAlignment = System.Windows.VerticalAlignment.Top;
27         myButton.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
28         myButton.Background = new SolidColorBrush(Colors.Red);
29         myButton.Content = "Meow";
30         ContentPanel.Children.Add(myButton);
31     }
```

I've added this C# code in the constructor of my MainPage class. I'll talk about the relationship between the MainPage.xaml and MainPage.xaml.cs in just a moment, but we've already seen how we can define behavior by writing procedural C# code in the MainPage.xaml.cs file. Here, I'm merely writing code that will execute as soon as a new instance of the MainPage class is created by writing the code in the constructor of that class.

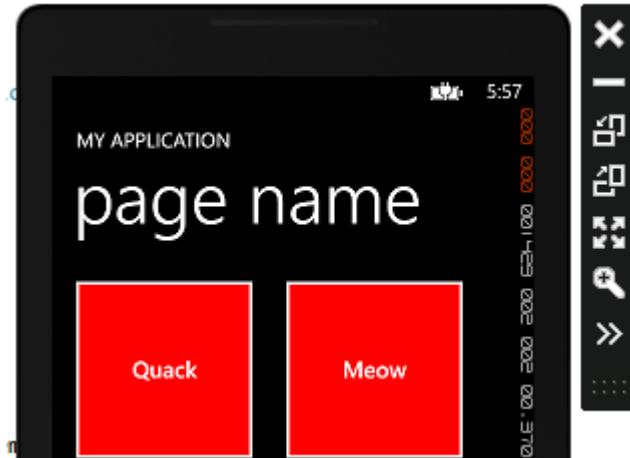
At this point, I now have two buttons ... one defined declaratively in XAML that has the content of "Hello World" and will quack when we click the button, and a new second button that has the content of "Quack". When we run the application:



we can see only one button. That's because the button we just created procedurally in C#, in the constructor of the MainPage class, is sitting on top of the first button we created in the previous lesson in XAML. Just to prove it, I'll add one more line of C# code that will set a margin on the new Quack button, moving it to the left 210 pixels:

```
18     public MainPage()
19     {
20         InitializeComponent();
21
22         Button myButton = new Button();
23         myButton.Name = "MeowButton";
24         myButton.Height = 200;
25         myButton.Width = 200;
26         myButton.VerticalAlignment = System.Windows.VerticalAlignment.Top;
27         myButton.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
28         myButton.Background = new SolidColorBrush(Colors.Red);
29         myButton.Content = "Meow";
30         ContentPanel.Children.Add(myButton);
31
32         myButton.Margin = new Thickness(210, 0, 0, 0);
33     }
```

The Margin property of the Button is of type Thickness, a general purpose class that represents 4 dimensions. In this case, we create a new Thickness class and set its first constructor argument to 210 pixels. When we run the application again:



... we can now see both buttons.

The larger issue is that we have two (almost) identical buttons ... one created declaratively in XAML and the other procedurally in C#.

When I create a new XAML element like so:

```
<Button></Button>
```

I'm basically creating a new instance of the Button class.

When I set attributes on the Button element, I'm basically setting properties on the instance of the Button class.

The important take away is this: XAML is simply a way to create instances of classes and set those objects' properties in a much more simplified, succinct syntax. What took us 10 lines of C# code we were able to accomplish in just one line of XAML (even if I did separate it on to different lines in my editor, it's still MUCH SHORTER than it would have been had I used C# to create my objects).

Furthermore, using XAML I have this immediate feedback in the Phone preview pane. I can see the impact of my changes instantly. In the case of the procedural C# code I wrote, I would have to run the app each time I wanted to see how my tweaks to the code actually worked.

## 2. Introducing Type Converters

If you have a keen eye, you might notice the difference in the XAML and C# versions when it comes to the HorizontalAlignment attribute / property ... If you tried:

```
myButton.HorizontalAlignment = "Left";
```

... you would get a compilation error. The XAML parser will perform a conversion to turn the string value "Left" into the enumeration value System.Windows.HorizontalAlignment.Left through the use of a Type Converter. A Type Converter is a class that can translate from a string value into a strong type—there are several of these built into the Windows 8 API that we'll use throughout this series. In this example, the HorizontalAlignment property, when it was developed by Microsoft's developers, was marked with a special attribute in the source code which signals to the XAML parser to run the string value through a type converter method to try and match the literal string "Left" with the enumeration value System.Windows.HorizontalAlignment.Left.

Just for fun, take a look at what happens when you attempt to misspell "Left":



```
 39 <Button Name="PlayAudioButton"
40     Width="200"
41     Height="200"
42     HorizontalAlignment="eft"
43     VerticalAlignment="Top"
44     Background="Red"
45     Click="PlayAudioButton_Click"
46     >
47     Quack
48 </Button>
```

The screenshot shows a code editor with a vertical line of numbers on the left (39 to 48) and a horizontal line of code on the right. The word 'eft' in the line 'HorizontalAlignment="eft"' is underlined with a blue wavy line, indicating a spelling error.

... you'll get a compilation error because the Type Converter can't find an exact match that it can convert into the enumeration value System.Windows.HorizontalAlignment.Left.

So, the first characteristic of XAML is that it is a succinct means of creating instances of classes. In the context of building a Windows 8 application, it is used to create instances of user interface elements, however XAML is not just a user interface technology—it could be used for other purposes in other technologies.

### 3. Understanding XAML Namespace Declarations

Next, let's talk about all that XAML code at the very top of the MainPage.xaml file ... we ignored it until now.

At the very top of the file, we see the following:

```
1 <phone:PhoneApplicationPage
2     x:Class="PetSounds.MainPage"
3     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5     xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6     xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
9     mc:Ignorable="d"
10    FontFamily="{StaticResource PhoneFontFamilyNormal}"
11    FontSize="{StaticResource PhoneFontSizeNormal}"
12    Foreground="{StaticResource PhoneForegroundBrush}"
13    SupportedOrientations="Portrait" Orientation="Portrait"
14    shell:SystemTray.IsVisible="True">
15
16    <!--LayoutRoot is the root grid where all page content is placed-->
17    <Grid x:Name="LayoutRoot" Background="Transparent">
18        <Grid.RowDefinitions>
19            <RowDefinition Height="Auto"/>
20            <RowDefinition Height="*"/>
21        </Grid.RowDefinitions>
22
```

While you're looking this over, remember what I said a moment ago—about schemas being a part of XML. If that's the case, then where does this XAML promise to adhere to a schema?

See lines 3 through 8 ... there are SIX schemas this MainPage.xaml is promising to adhere to. Each is defined with the xmlns attribute. The first xmlns defined in line 3 is the default namespace—in other words, there's no colon and no word after the colon like you see in lines 4 through 8.

The rest of the namespaces in lines 4 through 8 will use name / colon combination. So, just to be clear ... the :x or :phone is the NAMESPACE, that is associated with a SCHEMA (what we've called a contract). Each element and attribute in the rest of this MainPage.xaml MUST ADHERE TO AT LEAST ONE OF THESE SCHEMA's, otherwise the document is said to be invalid. In other words, if there's an element or attribute expressed in this XAML file that is not defined in one of these namespaces, then there's no guarantees that the compiler—the program that will parse through our source code and create an executable that will run on the Phone—the compiler will not be able to understand how to carry out that particular instruction.

So, in this example:

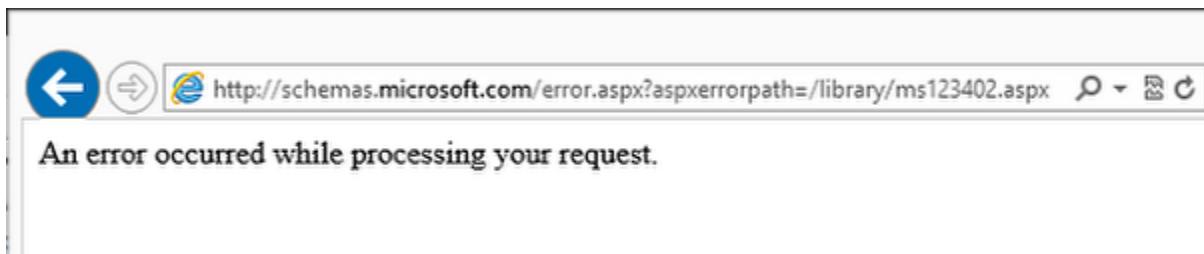
```
<Grid x:Name="LayoutRoot" Background="Transparent">
```

We would expect the element Grid and attribute Background to be part of the default schema corresponding with the default namespace defined at the location in line 3.

However, x:Name is part of the schema corresponding with the x: namespace defined at the location in line 4.

I have a bright idea ... let's try to navigate to default namespace to learn more about what makes up a namespace in the first place:

<http://schemas.microsoft.com/winfx/2006/xaml/presentation>



What?! The schema doesn't actually exist at that URL! That's because the schema is not published in the sense that you can go to that URL and view it. Instead, a schema is simply a unique name, similar to how we used Namespaces in C# to identify two classes that may have the same name—the schema (and therefore, the namespace in our XAML) keeps class names sorted out, kind of like a last name or surname. This URL, or more properly, we should refer to it as a URI (Uniform Resource IDENTIFIER ... rather than LOCATOR) is used as a namespace identifier. The XML namespaces are instructions to the various applications that will parse through the XAML ... the Windows Runtime XAML parser will be seeking to turn it into executable code, while the Visual Studio and Blend designers will be seeking to turn it into a design-time experience.

So, the second XML Namespace defines a mapping, x: as belonging to this schema:

<http://schemas.microsoft.com/winfx/2006/xaml>

Therefore, any elements or attribute names that are preceded by the x: prefix means that they adhere to this second schema.

But, what's the difference? It's subtle, but the second schema defines the intrinsic rules for XAML in general. The first schema defines the contract / rules for Windows 8 specific usage of XAML. In other words, the fact that we can work with the Grid, Button, MediaElement and the other Windows Phone 8 XAML elements without using a prefix means that they are defined in the default namespace.

Lines 5 & 6 define namespaces and schemas for the Phone and Shell, which are at a different URI, one that takes its cues from the Microsoft.Phone CLR namespace as defined in assemblies that were installed on our computers after we installed the Windows Phone 8 API. As you can see, the very first line:

```
<phone:PhoneApplicationPage
```

indicates that the PhoneApplicationPage class itself is part of this definition. The PhoneApplicationPage derives from Windows.System.Controls.Page ... that just happens to be the same class that is a parent to Windows Presentation Foundation page classes, and Windows Store app page classes. It has a lot of the base functionality shared by all three of these project types. So, the upside is that you're able to leverage what you learn in this series to building WPF desktop apps and Windows Store apps. There will be differences, but there's a lot in common, too!

Lines 7 & 8 define namespaces and schemas that are used to allow Visual Studio's Phone Preview Pane on the left to display properly. These instructions are ignored at runtime, which is what line 9 is doing—when compiling the XAML code, ignore anything prefixed with :d.

Ok, I know there are some questions left unanswered ... we could spend a lot of time talking about the specifics, but the main takeaway is that this code at the very top of each XAML file you add to your phone project does have a purpose, it defines the rules that your XAML code must follow. You'll almost never need to modify this code, but if you remove it, you could potentially break your application. So, I would encourage you to not fiddle around with it unless you have a good reason to. There are a few additional attributes in lines 10 through 14 ... we may talk about them later in this series.

#### 4. Understanding the relationship between the .xaml and .xaml.cs files

In Visual Studio's Solution Designer, you can see that the XAML files have an arrow which means that we can expand them to reveal a C# file by the same name, the only difference is that it has a .cs file name extension appended to the end. If you look at the .cs version of the file, you'll see that it defines a MainPage class, and furthermore, it defines it as a PARTIAL class:

```
12
13  namespace PetSounds
14  {
15      public partial class MainPage : PhoneApplicationPage
16      {
17          // Constructor
18          public MainPage()
19          {
20              InitializeComponent();
```

The other half of the equation is defined in lines 1 & 2 of the MainPage.xaml:

```
<phone:PhoneApplicationPage
x:Class="SoundBoard.MainPage"
...

```

While it doesn't use the term Partial like its procedural counterpart, it does indicate the relationship between the two.

Why is this important? This relationship means that the compiler will combine the output of the MainPage.xaml and the MainPage.xaml.cs files into a SINGLE CLASS. This means that they are two parts of a whole. That's an important concept ... that the XAML gets compiled into Intermediate Language just like the C# gets compiled into Intermediate Language and they are both partial implementations of a single class. This allows you to create an instance of a class in one file then use it in the other file, so to speak. This is what allows me to create an instance of the MediaElement class called \_audioPlayer in XAML and then call its methods or set its properties in C#. We'll see more examples of this later in this lesson.

## 5. Understanding Default Properties

Since XAML is essentially an XML document, we can embed elements inside of other elements. We've already seen an example of this:

```
<PhoneApplicationPage>
<Grid ...>
<Grid ... >
<MediaElement ... />
<Button ... />
</Grid>
</Grid>
</PhoneApplicationPage>
```

Here PhoneApplicationPage "contains" a Grid and the Grid "contains" a MediaElement and Button. Or, perhaps more correctly in XAML parlance, UserControl's Content property is set to Grid, and Grid's Children collection includes the MediaElement and Button. Depending on the type of control you're working with, the default property can be populated using this embedded style syntax. So, you could do this:

```
<Button Content="Hello World" ... />
```

... or this ...

```
<Button ... >
Hello World
</Button>
```

since the Content property is the default property of the Button class.

## 6. Understanding Complex Properties and the Property Element Syntax

In some cases, merely setting attribute values masks the complexity of what's going on behind the scenes. A good example of this is setting Background="Red". We've already seen this procedurally in C# -- to accomplish the same thing:

```
myButton.Background = new SolidColorBrush(Colors.Red);
```

... we have to create a new instance of a SolidColorBrush and pass in an enumerated Colors value. This is another great example of a property type converter that we learned about earlier in this lesson. But some attributes are simply too complex to be represented as attributes.

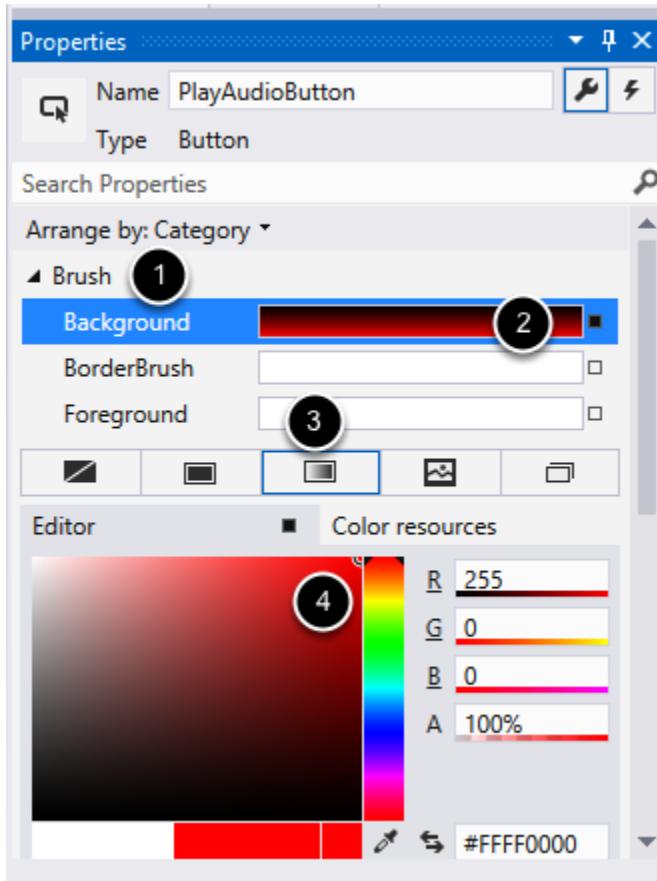
When a properties is not easily represented as a XAML attribute, it's referred to as a "complex property". To demonstrate this, first I'm going to remove the Background="Red" attribute from the Button, remove "Hello World!" as the default property, and add it back with a Content="Hello World!" attribute:



The screenshot shows a code editor with a vertical ruler on the left. The code is as follows:

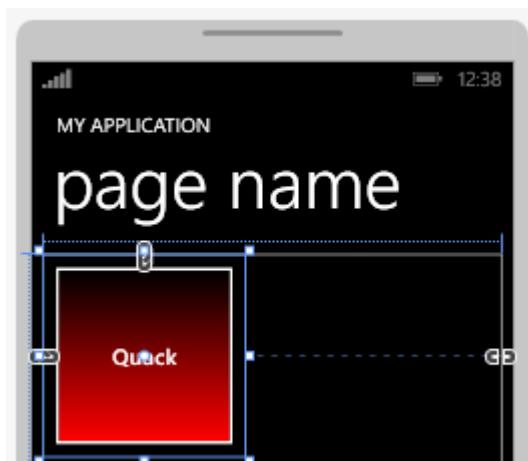
```
39     <Button Name="PlayAudioButton"
40         Width="200"
41         Height="200"
42         HorizontalAlignment="Left"
43         VerticalAlignment="Top"
44         Content="Quack"
45         Click="PlayAudioButton_Click"
46     >
47
48 </Button>
```

Next, in the Properties pane, I'm going to set the Background property to a linear gradient brush.



1. Select the Brush property to reveal the Brush editor.
2. Make sure that Background is selected.
3. Select the Linear Brush tab (middle tab).
4. Move the selector all the way to the upper-right hand side of the color editor.

I should now see the following in the Phone Preview pane:



... but more importantly, let's look at the XAML that was generated by the Brush editor:



```
39 <Button Name="PlayAudioButton"
40     Width="200"
41     Height="200"
42     HorizontalAlignment="Left"
43     VerticalAlignment="Top"
44     Content="Quack"
45     Click="PlayAudioButton_Click"
46     >
47     <Button.Background>
48         <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
49             <GradientStop Color="Black" Offset="0"/>
50             <GradientStop Color="Red" Offset="1"/>
51         </LinearGradientBrush>
52     </Button.Background>
53
54 </Button>
```

The XAML required to create that background cannot be easily set in a simple literal string like before when we simply used the word "Red". Instead, notice how the Background property is broken out into its own element:

```
<Button ... >
<Button.Background>
...
</Button.Background>
</Button>
```

This is called "property element" syntax and is in the form `<Control.Property>`.

A good example is the `LinearGradientBrush`. The term "brush" means that we're working with an object that represents a color or colors. Think of "brush" like a paint brush ... this particular paint brush will create a gradient that is linear—the color will change from top to bottom or left to right. Now, admittedly, you would NEVER want to do what I'm doing in this code example because it goes against the aesthetic of all Windows Phone 8 applications. But, let's pretend for now that we're expressing our individuality by using a gradient color as the background color for a Button.

As you can see (below), if we want to define a `LinearGradientBrush`, we have to supply a lot of information in order to render the brush correctly ... the colors, at what point that color should break into the next color, etc. The `LinearGradientBrush` has a collection of `GradientStop` objects which define the colors and their positions in the gradient (i.e., their "Offset").

However, the XAML representing the `LinearGradientBrush` in the code snippet above is actually SHORTENED automatically by Visual Studio. Here's what it should be:

```
<Button.Background>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
```

```
<LinearGradientBrush.GradientStops>
<GradientStopCollection>
<GradientStop Color="Red" Offset="1" />
<GradientStop Color="Black" Offset="0" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
</Button.Background>
```

Notice how the `<LinearGradientBrush.GradientStops>` and `<GradientStopCollection>` elements are omitted? This is done for conciseness and compactness and is made possible by an intelligent XAML parser. First of all, the `GradientStops` property is the default property for the `LinearGradientBrush`. Next, `GradientStops` is of type `GradientStopCollection` and implements `IList<T>`, the `T` in this case would be of type `GradientStop`. Given that, it is possible for the XAML parser to deduce that the only thing that could be nested inside the `<LinearGradientBrush ... />` is one or more instances of `GradientBrush`, each being implicitly `.Add()`'ed to the `GradientStopCollection`.

So the moral of the story is that XAML allows us to create instances of classes declaratively, and we have a granular fidelity of control to design user interface elements. Even so, the XAML parser is intelligent and doesn't require us to include redundant code—as long as it has enough information to create the object graph correctly.

## Recap

To recap, we've learned about the syntax of XAML. Most of XAML is pretty straightforward, but there are a few things that are not quite as obvious:

1. XAML is simply an implementation of XML, and relies heavily on schemas and namespaces to adhere to "contracts" so that different applications can create, interpret, display, or compile the XAML code.
2. The purpose of XAML is to allow for a compact, concise syntax to create instances of classes and set their properties. We compared the procedural version of a button created in C# versus one created declaratively in XAML and saw how much less code was required.
3. XAML requires less code due to its built-in features like property type converters which allows a simple string value be converted into an instance of a class.
4. For more complex property settings, XAML offers property element syntax which harnesses the intelligent XAML parser to rely on default properties and deduction to reduce the amount of XAML code required to express a design.
5. We learned about the embedded syntax style and the embedded nature of elements which suggests a relationships between visual elements. For example, the `PhoneApplicationPage` contains a `Grid` for layout, which in turn contains input or other

types of controls. These are represented with opening and closing elements representing containment or ownership.

6. We learned about default properties; each control has a default property that can be set using that same style of embedded syntax.

Next, let's learn more about the Grid for layout, we'll learn about XAML's attached property syntax and how events are wired up in the next lesson.

# Part 5: Basics of Layout and Events

Source Code: <http://aka.ms/absbeginnerdevwp8>

In this lesson I want to talk about layout, or in other words, how controls are positioned and arranged on your app's user interface.

So, my game plan:

1. We'll start by talking about the two primary elements used in layout and positioning: the Grid element and StackPanel element.
2. With regards to the Grid, I'll talk about defining rows and columns and various sizing options and techniques you can use to fully unlock the power of the Grid.
3. Next, we'll learn about the StackPanel and just for fun, we'll convert our app from a Grid layout to a StackPanel layout—this will teach us about some key differences between the Grid and StackPanel.
4. Finally, we'll talk about how event handlers are "wired up" from both XAML and in C#.

## 1. Understanding the Basics of Grids

The default Windows Phone Page template creates a <Grid> element called "ContentPanel".

```
47
48      <!--ContentPanel - place additional content here-->
49      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
50
51      </Grid>
52
```

The Grid element is used for laying out other controls ... it allows you to define rows and columns, and then each control can request which row and column they want to be placed inside of.

Now, in the default MainPage.xaml page template, between the opening and closing <Grid></Grid> tags, the Grid appears empty ... it appears that there's no rows or columns defined. However, by default, there's always one RowDefinition and one ColumnDefinition even if it's not explicitly defined, and these take up the full vertical and horizontal space available to represent one large "cell" in the Grid. Any items placed between the opening and closing <Grid></Grid> elements are understood to be inside that single implicitly defined "cell".

## 2. Grid RowDefinitions and ColumnDefinitions and defining sizes

An example of a Grid that actually defines rows is the "LayoutRoot" grid. The Grid defines two rows:

```
16      <!--LayoutRoot is the root grid where all page content is placed-->
17      <Grid x:Name="LayoutRoot" Background="Transparent">
18          <Grid.RowDefinitions>
19              <RowDefinition Height="Auto"/>
20              <RowDefinition Height="*"/>
21          </Grid.RowDefinitions>
```

For now, notice how the StackPanel with the project and page titles places itself inside the first row ... the StackPanel has an attribute called Grid.Row="0". You reference both rows and columns using a zero-based numbering scheme.

Notice how the ContentPanel Grid places itself in the second row (by setting the attribute Grid.Row="1").

The first row has its height set to "Auto". The second row (row 1) is set to "\*". There are three syntaxes that you can use to help persuade the sizing for each row and column. I used the term "persuade" intentionally. With XAML layout, heights and widths are relative and can be influenced by a number of factors. All of these factors are considered by the layout engine to determine the actual placement of items on the page.

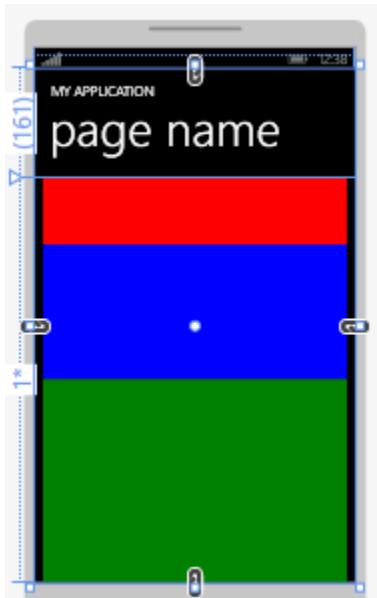
For example, "Auto" means that the height for the row should be tall enough to accommodate all of the controls that are placed inside of it. If the tallest control is 150 pixels tall, then that's the actual height of the row. If it's only 100 pixels, then THAT is the height of the row. Therefore, "Auto" means the height is relative to the controls inside of the row.

The asterisk is known as "star sizing" and means that the height of the row should take up all the rest of the height available.

Here's a quick example of another way to use "star sizing" ... I created a project that has three rows defined in the ContentPanel. Notice the heights of each one:

```
28
29      <!--ContentPanel - place additional content here-->
30      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
31          <Grid.RowDefinitions>
32              <RowDefinition Height="1*"/>
33              <RowDefinition Height="2*"/>
34              <RowDefinition Height="3*"/>
35          </Grid.RowDefinitions>
36
37          <Rectangle Fill="Red" Grid.Row="0" />
38          <Rectangle Fill="Blue" Grid.Row="1" />
39          <Rectangle Fill="Green" Grid.Row="2" />
40      </Grid>
```

Putting a number before the asterisk, I'm saying "Of all the space available, give me 1 or 2 or 3 shares of the remainder". Since the sum of all those rows adds up to six, each  $1^*$  is equivalent to 1/6th of the height available. Therefore,  $3^*$  would get half of the height available as depicted in the output of this example:



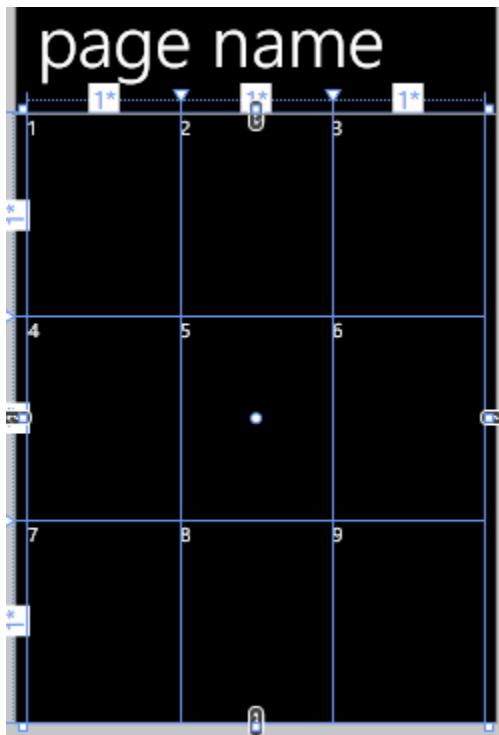
Besides Auto and star sizing, you can also specify widths and heights (as well as margins) in terms of pixels. In fact, when only numbers are present, it represents that number of pixels. Generally, it's not a good idea to use exact pixels in layouts for widths and heights because of the likelihood that screens—even Windows Phone screens—can have different dimensions. Instead, it's preferable to use relative values like Auto and star sizing for layout.

One thing to note from this example (and from our original PetSounds example) is that control widths and heights are assumed to be 100% unless otherwise specified. That's why the rectangles take up the entire "cell" size. This is why the button first occupied the entire ContentPanel grid, and why I had to specify I wanted the button to be 200 x 200 pixels instead.

I want to also point out that a Grid can have a collection of ColumnDefinitions as you can see from this example app I created called GridsRowsAndColumns. Here we have a 3 by 3 grid:

```
31      <!--ContentPanel - place additional content here-->
32      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
33          <Grid.RowDefinitions>
34              <RowDefinition Height="*" />
35              <RowDefinition Height="*" />
36              <RowDefinition Height="*" />
37          </Grid.RowDefinitions>
38          <Grid.ColumnDefinitions>
39              <ColumnDefinition Width="*" />
40              <ColumnDefinition Width="*" />
41              <ColumnDefinition Width="*" />
42          </Grid.ColumnDefinitions>
43
44          <TextBlock>1</TextBlock>
45          <TextBlock Grid.Column="1">2</TextBlock>
46          <TextBlock Grid.Column="2">3</TextBlock>
47
48          <TextBlock Grid.Row="1">4</TextBlock>
49          <TextBlock Grid.Row="1" Grid.Column="1">5</TextBlock>
50          <TextBlock Grid.Row="1" Grid.Column="2">6</TextBlock>
51
52          <TextBlock Grid.Row="2">7</TextBlock>
53          <TextBlock Grid.Row="2" Grid.Column="1">8</TextBlock>
54          <TextBlock Grid.Row="2" Grid.Column="2">9</TextBlock>
55
56      </Grid>
```

... the result is a simple grid with a number in each "cell":



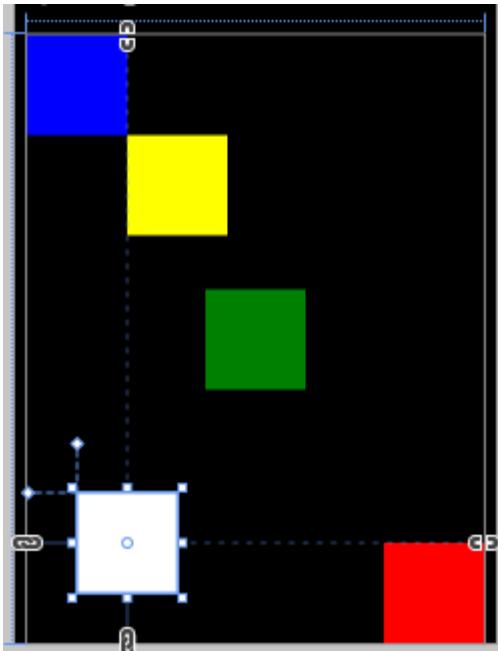
The other thing I want you to notice about this example is that when you don't specify a Grid.Row or Grid.Column, it is assumed to mean the first row (row 0) or first column (column 0). Relying on the defaults keeps your code more concise.

### 3. Grid cell Alignments and Margins

I have another example app called AlignmentAndMargins. This XAML:

```
29      <!--ContentPanel - place additional content here-->
30      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
31          <Rectangle Fill="Blue"
32              Width="100" Height="100"
33              HorizontalAlignment="Left" VerticalAlignment="Top" />
34
35          <Rectangle Fill="Red"
36              Width="100" Height="100"
37              HorizontalAlignment="Right" VerticalAlignment="Bottom" />
38
39          <Rectangle Fill="Green"
40              Width="100" Height="100"
41              HorizontalAlignment="Center" VerticalAlignment="Center" />
42
43          <Rectangle Fill="Yellow"
44              Width="100" Height="100"
45              HorizontalAlignment="Left" VerticalAlignment="Top"
46              Margin="100,100" />
47
48          <Rectangle Fill="White"
49              Width="100" Height="100"
50              HorizontalAlignment="Left" VerticalAlignment="Bottom"
51              Margin="50,0,0,50" />
52      </Grid>
```

Produces this result:



Most of this example should be obvious if you stare at it for a few moments, but there are several finer distinctions I want to make about Alignments and Margins. First, it points out how VerticalAlignment and HorizontalAlignment work, even in a given Grid cell (and the same will hold true in a StackPanel as well). The `__Alignment` attributes PULL controls TOWARDS their boundaries. By contrast, the Margin attributes PUSH controls AWAY from their boundaries. The second observation is the odd way in which Margins are defined ... as a series of numeric values separated by commas. This convention was borrowed from Cascading Style Sheets. The numbers represent the margin pixel values in a clock-wise fashion starting from the left side. So,

`Margin="10,20,30,40"`

Means, 10 pixels from the left boundary, 20 pixels from the top boundary, 30 pixels from the right boundary, 40 pixels from the bottom boundary. In this case, the boundary is a single Grid cell (since the Content panel only has not defined any additional RowDefinitions and ColumnDefinitions).

A bit earlier I said that it's generally a better idea to use relative sizes like Auto and \* to define heights and widths ... why, then, are margins are defined using pixels? Margins are expressed in exact pixels because they are usually just small values to provide spacing or padding between two relative values and can therefore be a fixed value without negatively impacting the overall layout of the page.

#### 4. StackPanel basics

The second style of layout is enabled by using the StackPanel element. It will arrange your controls in a flow from top to bottom.

Back in our PetSounds project, we have the following StackPanel defined to create the app title and page title at the top of the MainPage.xaml:

```
23      <!--TitlePanel contains the name of the application and page title-->
24      <StackPanel x:Name="TitlePanel"
25          Grid.Row="0"
26          Margin="12,17,0,28">
27          <TextBlock Text="MY APPLICATION"
28              Style="{StaticResource PhoneTextNormalStyle}"
29              Margin="12,0"/>
30          <TextBlock Text="page name"
31              Margin="9,-7,0,0"
32              Style="{StaticResource PhoneTextTitle1Style}"/>
33      </StackPanel>
```

Both TextBlock elements take up the entire horizontal width of their parent (the LayoutRoot Grid) and therefore they are stacked vertically. Let's convert our ContentPanel:

```
35      <!--ContentPanel - place additional content here-->
36      <Grid x:Name="ContentPanel"
37          Grid.Row="1"
38          Margin="12,0,12,0">
39          <Button Name="PlayAudioButton" ...>
55
56          <MediaElement x:Name="QuackMediaElement" ... />
61
62      </Grid>
```

from a Grid into a StackPanel:

```
35      <!--ContentPanel - place additional content here-->
36      <StackPanel x:Name="ContentPanel"
37          Grid.Row="1"
38          Margin="12,0,12,0">
39          <Button Name="PlayAudioButton" ...>
55
56          <MediaElement x:Name="QuackMediaElement" ... />
61
62      </StackPanel>
```

At first, this conversion only moves the Meow Button vertically downward:



However, if I were to remove the Width and HorizontalAlignment attributes from the Quack button (I also remove the Button.Background property):

```
39 <Button Name="PlayAudioButton"
40     Height="100"
41     VerticalAlignment="Top"
42     Content="Quack"
43     Click="PlayAudioButton_Click"
44     />
45 
```

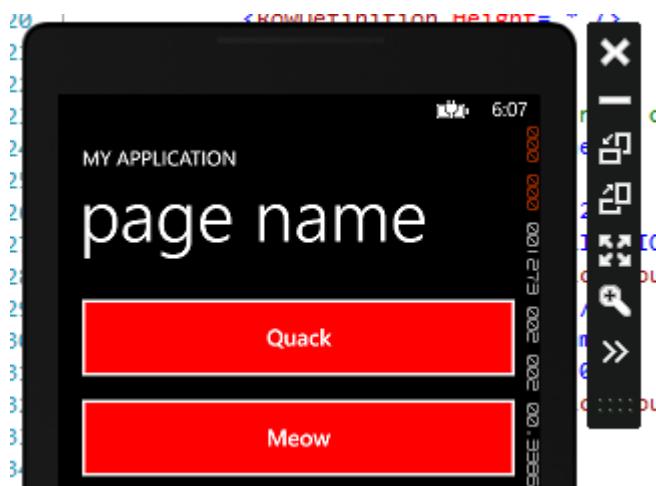
And comment out the Width and HorizontalAlignment attributes from the Meow button ...  
AND comment out the Margin in my MainPage() constructor:

```

17     // Constructor
18     public MainPage()
19     {
20         InitializeComponent();
21
22         Button myButton = new Button();
23         myButton.Name = "MeowButton";
24         //myButton.Height = 200;
25         myButton.Height = 100;
26         //myButton.Width = 200;
27         myButton.VerticalAlignment = System.Windows.VerticalAlignment.Top;
28         //myButton.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
29         myButton.Background = new SolidColorBrush(Colors.Red);
30         myButton.Content = "Meow";
31         ContentPanel.Children.Add(myButton);
32
33         //myButton.Margin = new Thickness(210, 0, 0, 0);
34     }

```

Then you could see that the two buttons are vertically stacked in the StackPanel:



So, as you can see, you can achieve just about any layout you can imagine by using a combination of four things:

- Grid layout, including RowDefinitions and ColumnDefinitions
- StackPanel layout
- Margins to move elements away from the left-side, top-side, right-side or bottom-side
- The HorizontalAlignment and VerticalAlignment attributes

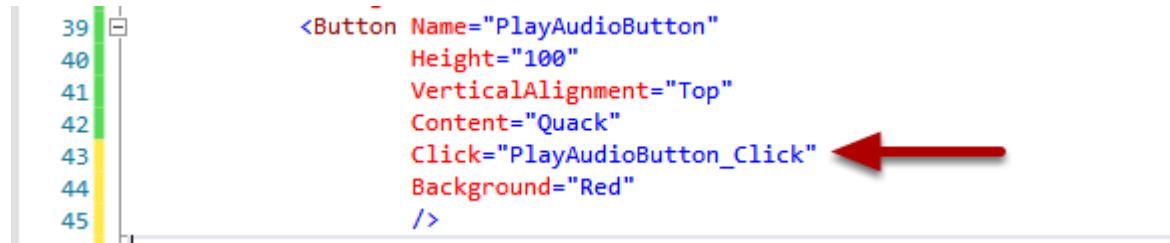
Ok, so that's XAML Layout in a nutshell. Let's move on to events.

## 5. Understanding Events

If you watched the C# Fundamentals series on Channel9, in one of the last videos we talked about events. In the Windows Phone API, Pages and Control raise events for key

moments in their lifecycle OR for interactions with the end user. In this series, we've already "wired up" the Click event of our PlayAudioButton with a method that I called an "event handler". When I use the term "event handler" I'm merely referring to a method that is associated with an event. I use the term "wired up" to mean that the event is tied to a particular event handler method. Some events can be triggered by a user, like the Click event of a Button control. Other events happen during the lifetime of an event, like a Loaded event that occurs after the given Page or Control object has been instantiated by the Phone APIs Runtime engine.

There are two ways to "wire up" an event for a Page or Control to an event handler method. The first is to set it using the attribute syntax of XAML. We've already done this in our PlayAudioButton example:



```
39 <Button Name="PlayActionButton"
40     Height="100"
41     VerticalAlignment="Top"
42     Content="Quack"
43     Click="PlayActionButton_Click" ← Red Arrow
44     Background="Red"
45 />
```

If you'll recall, we typed:

Click="

And before we typed in the closing double-quotation mark, Intellisense asked if we wanted to select or create an event handler. We told it we wanted to create a new event handler, and Visual Studio named the event handler using the naming convention:

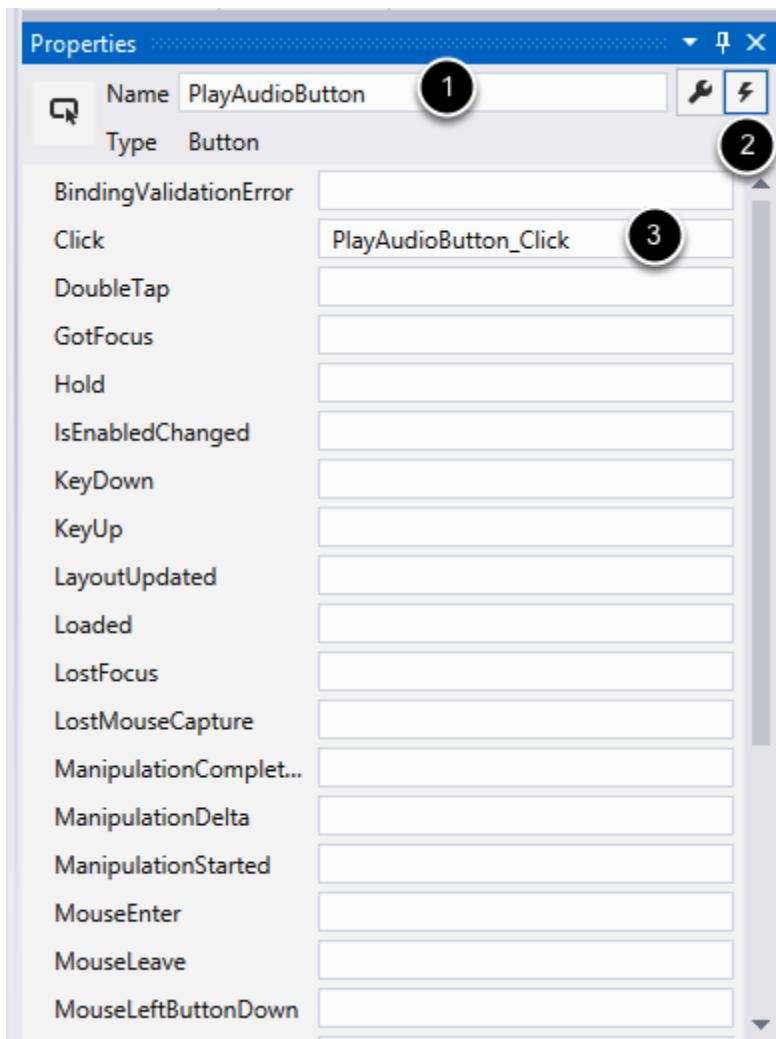
NameOfElement\_EventName

... so in our case ...

PlayActionButton\_Click

Visual Studio also created a method stub in the code-behind for the XAML page, in our case, the MainPage.xaml.cs. Keep in mind that these are Visual Studio automations. We could have hit the escape key on the keyboard when Intellisense first popped up and typed that code in by hand.

A second way to associate an event with an event handler is to use the Properties window in Visual Studio:



(1) First, you must make sure you've selected the Control you want to edit by placing your mouse cursor in that element. The name of that element will appear in the Name field at the top letting you know the context of the settings below. In other words, any settings you make will be to the PlayAudioButton as opposed to another control on the page.

(2) Click the lightening bolt icon. This will switch from a listing of Properties / attributes that can be changed in the Properties window to the Events that can be handled for this Control.

(3) Double-click on a particular textbox to create an association between that Control's event and an event handler. Double-clicking will automatically name the event and create a method stub for you in the code behind.

The third technique (that we'll use several times in this series) is to wire up the event handler in C# code. See line 35 below for an example:



```
17     // Constructor
18     public MainPage()
19     {
20         InitializeComponent();
21
22         Button myButton = new Button();
23         myButton.Name = "MeowButton";
24         //myButton.Height = 200;
25         myButton.Height = 100;
26         //myButton.Width = 200;
27         myButton.VerticalAlignment = System.Windows.VerticalAlignment.Top;
28         //myButton.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
29         myButton.Background = new SolidColorBrush(Colors.Red);
30         myButton.Content = "Meow";
31         ContentPanel.Children.Add(myButton);
32
33         //myButton.Margin = new Thickness(210, 0, 0, 0);
34
35         myButton.Click += PlayAudioButton_Click; ← Red arrow here
36     }
37 }
```

The `+=` operator can be used in other contexts to mean "add and set" ... so:

`x += 1;`

... is the same as ...

`x = x + 1;`

The same is true here (in a sense) ... we want to "add and set" the Click event to one more event handler method. Yes, the Click event of `myButton` can be set to trigger the execution of MULTIPLE event handlers! Here we're saying "When the Click event is triggered, add the `PlayAudioButton_Click` event handler to the list of event handlers that should be executed." One Click event could trigger on or more methods executing. In our app, I would anticipate that only this one event handler, `PlayAudioButton_Click`, would execute.

You might wonder: why doesn't that line of code look like this:

`myButton.Click += PlayAudioButton_Click();`

Notice the open and closed parenthesis on the end of the `_Click`. Recall from the C# Fundamentals series ... the `()` is the method invocation operator. In other words, when we use `()` we are telling the Runtime to EXECUTE the method on that line of code

IMMEDIATELY. That's NOT what we want in this instance. We only want to associate or point to the PlayAudioButton\_Click method.

One other interesting note about event handler methods. With my addition of line 35 in the code above, I now have two events both pointing to the same event handler method PlayAudioButton\_Click. That's perfectly legitimate. How, then, could we determine which button actually triggered the execution of the method?

If you look at the definition of the PlayAudioButton\_Click method:

```
private void PlayAudioButton_Click(object sender, RoutedEventArgs e)
{
}
```

The Windows Phone Runtime will pass along the sender as an input parameter. Since we could associate this event with any Control, the sender is of type Object (the type that virtually all data types in the .NET Framework ultimately derive from), and so one of the first things we'll need to do is do a few checks to determine the ACTUAL data type (Are you a Button control? Are you a Rectangle control?) and then cast the Object to that specific data type. Once we cast the Object to a Button (for example) then we can access the Button's properties, etc.

The method signature in the code snippet above is typical of methods in the Windows Phone API. In addition to the sender input parameter, there's a RoutedEventArgs type used for those events that pass along extra information. You'll see how these are used in advanced scenarios, but I don't believe we'll see them used in this series.

### Recap

To recap, the big takeaways in this lesson are the ways in which we can influence the layout of Pages and Controls. With regards to Grid layout we learned about the different ways to define the heights and widths of Rows and Columns (respectively) using Auto sizing, star sizing and pixel sizing. We talked about how VerticalAlignment and HorizontalAlignment pull controls towards boundaries while Margins push Controls away from boundaries. Then we talked about the different ways to wire up events to event handler methods, and the anatomy of an event handler method's input parameters.

# Part 6: Styling the App

Source Code: <http://aka.ms/absbeginnerdevwp8>

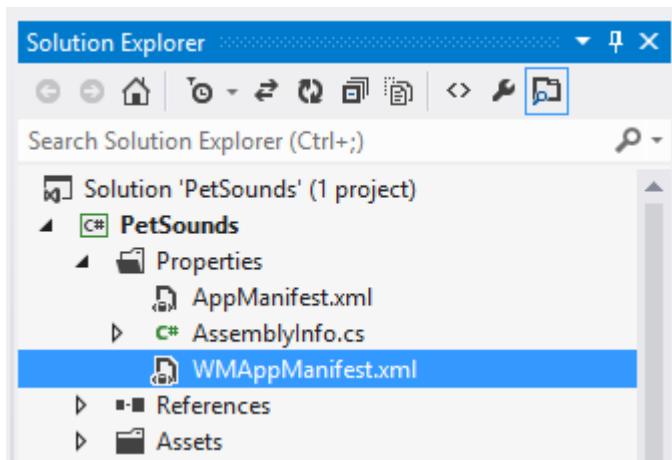
With the basics of XAML, layout and events in place, let's do something fun. We'll give some unique character to the app by styling it. Obviously we'll want to follow Microsoft's guidelines so that our app looks like it belongs as part of the Windows Phone 8 ecosystem, however we still have a lot of latitude on how we can personalize our app.

Here's the game plan for this lesson:

1. We'll change the tile icons used on the Phone's Application List and Start Page.
2. We'll access the user's Phone theme color selections and incorporate them into the colors in our app, and in so doing we'll learn some extremely important XAML syntax features called binding expressions.
3. Finally, we'll talk about creating re-usable styles in our app.

## 1. Change out the app's tile icons

Our first task in this lesson is to change the tiles for our app that the user will see in the alphabetical list of apps as well as the Start page if they should want to pin our app to their Start page. To begin, we'll open the WMAppManifest.xml file in the Properties directory of our Project:



When you double-click this file in the Solution Explorer it will be opened in a special designer window providing a number of options that affect how our application is introduced to the Windows Phone 8 operating system. For example, on the first tab, "Application UI" we can change the display name, the app icon and more:

WMAppManifest.xml + X MainPage.xaml.cs MainPage.xaml

Use this designer to set or modify some of the properties in the Windows Phone app manifest file.

Application UI Capabilities Requirements Packaging

Use this page to set the UI details that identify and describe your application.

Display Name: PetSounds

Description: Sample description

Navigation Page: MainPage.xaml

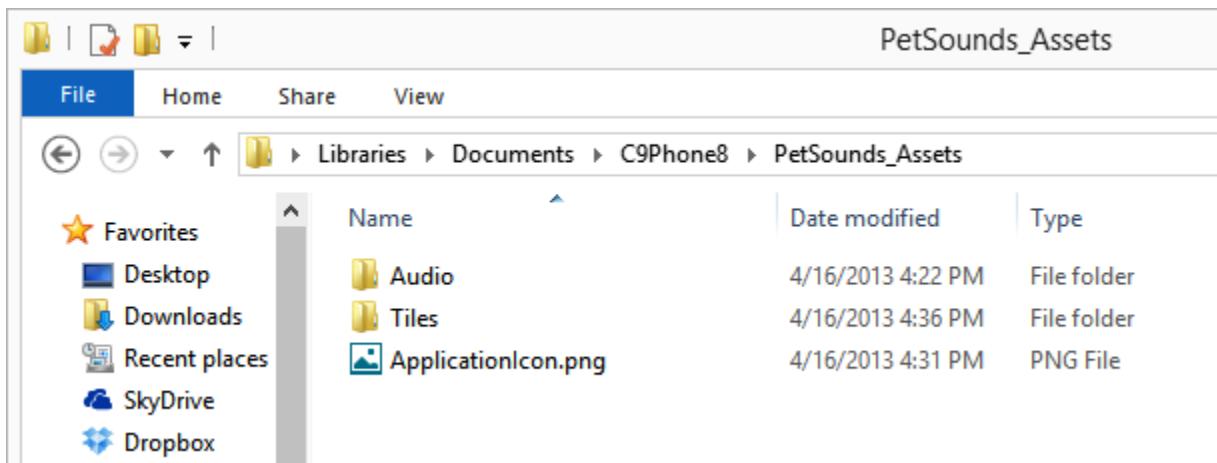
App Icon:

A placeholder icon for the application. It features a white sunburst-like shape on a grey square background, with a small 'X' in the top right corner and three dots at the bottom left indicating it's a placeholder.

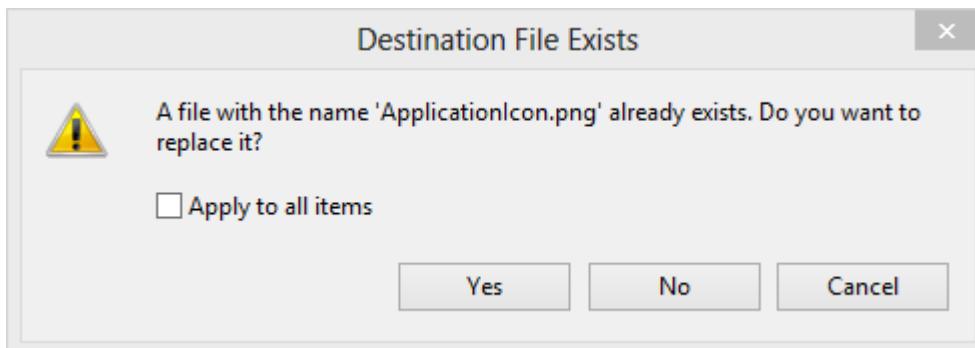
...

We want to change from the default icon to one more suited for our app. I've created such an icon and it's available in the C9Phone8\PetSounds\_Assets folder. These assets for this series are available from wherever you originally downloaded this document, or watched the videos that accompany it.

Inside that folder is the ApplicationIcon.png:

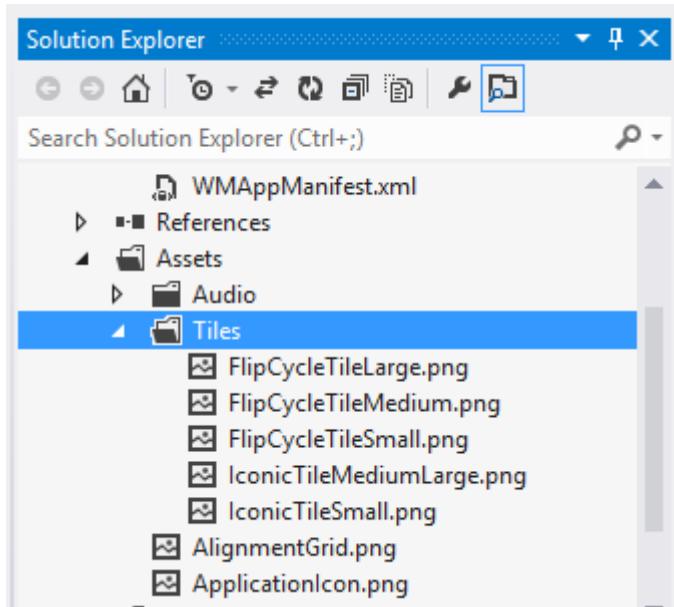


I'll drag and drop that file from Windows Explorer into the Assets folder of my Project. When I do that, I see a dialog notifying me that a file by that name already exists:

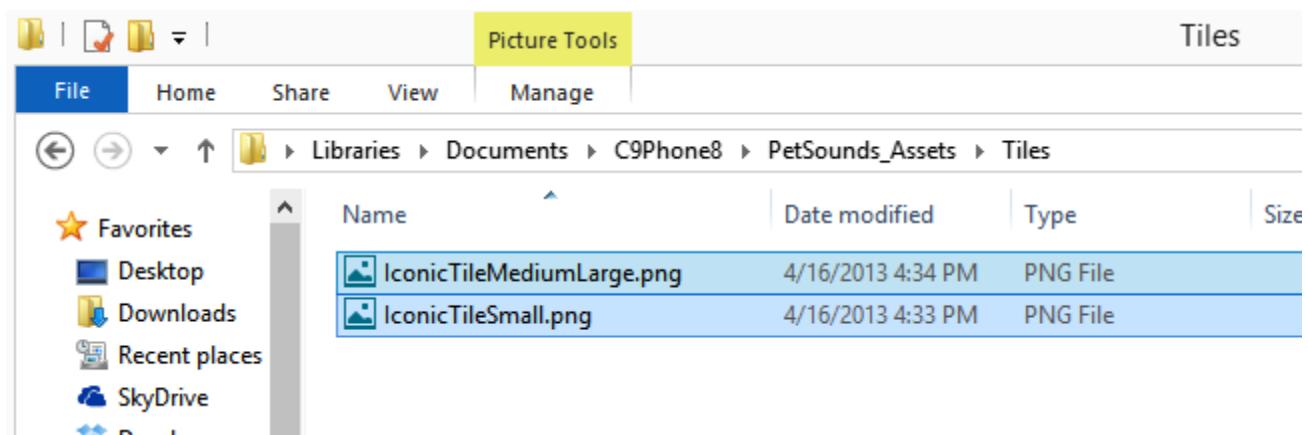


We do want to replace the old file with our new image file.

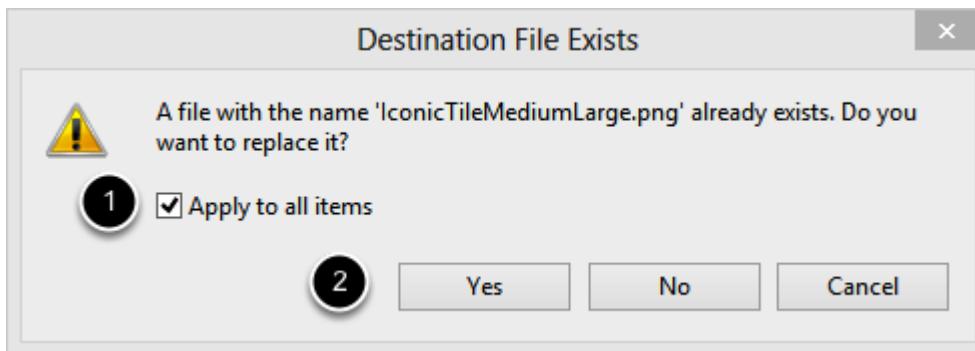
Next, I want to replace two images in the Assets\Tiles subfolder. I'll ready the target in the Solution Explorer by expanding the Tiles subfolder.



In Windows Explorer, I'll open the C9Phone8\PetSounds\_Assets\Tiles subfolder:



... I'll highlight the two image files and drag and drop them into the target Tiles folder in the Solution Explorer. I'll see the dialog again:



1. I want to apply my response to both files
2. I'll respond by clicking "Yes"

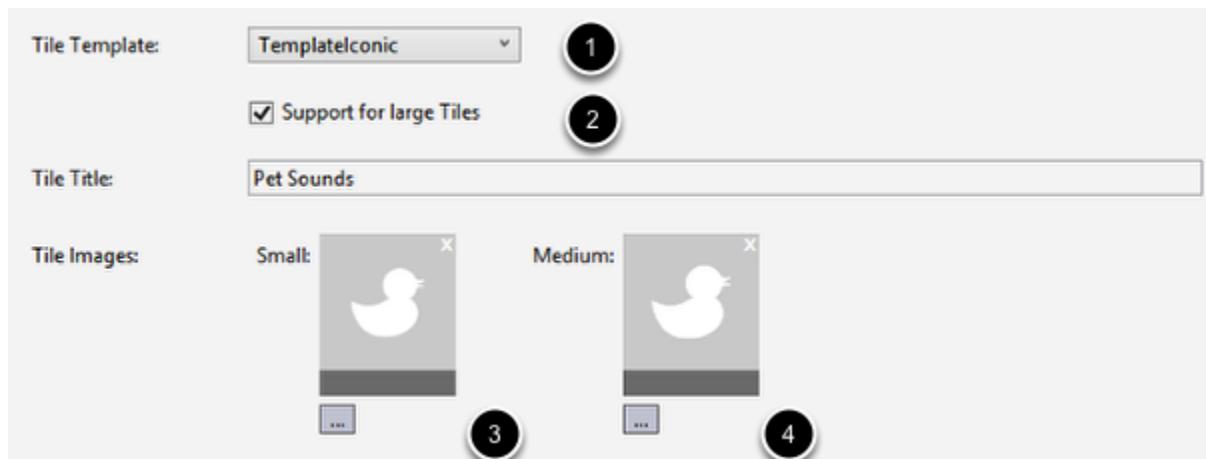
Now, I've replaced the necessary tile files in my project. Back in the WMAppManifest.xml file, since I replaced the old image file with a new one, I may need to close this file and re-open it to see the new App Icon reflected there:

WMAppManifest.xml\* X MainPage.xaml.cs MainPage.xaml

Use this designer to set or modify some of the properties in the Windows Phone app manifest file.

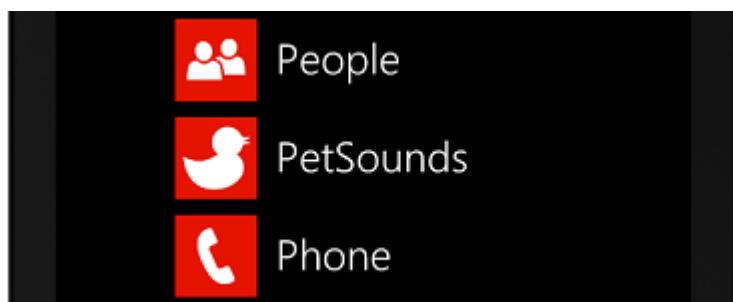
Application UI	Capabilities	Requirements	Packaging
Display Name: PERSONALIS			
Description: Sample description			
Navigation Page: MainPage.xaml			
App Icon: 			

Further down on that settings page, I want to make sure the following settings are chosen:



1. Tile Template set to TemplateIconic
2. Support for large Tiles should be checked
3. Small Tile Images should be set
4. Medium Tile Images should be set

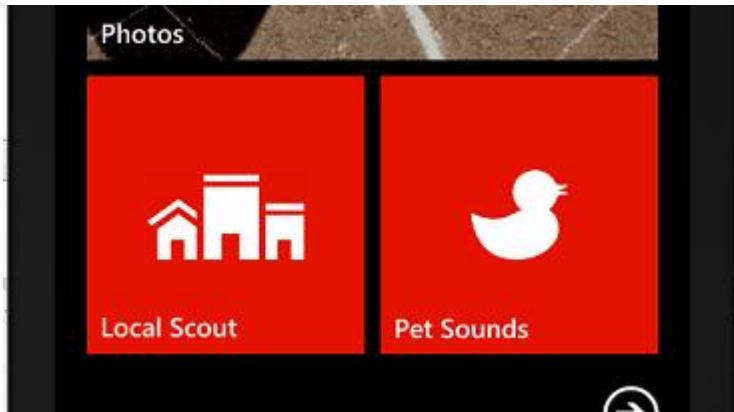
To test these settings, I'll run (F5) the app. Once it's running, on the Phone Emulator, click the Start button (the Windows icon), then swipe (click and hold down the mouse button while dragging the mouse cursor) from right-to-left to see the alphabetical list of apps and locate our PetSounds app:



Great! Now, let's click and hold down until a context menu appears displaying the options to "pin to start" and "uninstall":

```
30             <TextBlock Text="animals"
31                 Margin="9,-7,0,0">
32                 <TextBlock.Style>
33                     <Style BasedOn="{StaticResource PhoneTextBlockBase}"
34                         TargetType="TextBlock">
35                         <Setter Property="FontFamily"
36                             Value="{StaticResource PhoneFontFamilySemiLight}"/>
37                         <Setter Property="FontSize"
38                             Value="{StaticResource PhoneFontSizeExtraExtraLarge}"/>
39                     </Style>
40                 </TextBlock.Style>
41             </TextBlock>
```

Clicking "pin to start" will add the app to the Start page. Click the Start button on the Phone Emulator and scroll down to see the pinned tile:



It's a small detail, but it already feels like a more legitimate app just with that small change.

## 2. Modifying the App and Page Titles

Next, we'll change the app's title text and page's title text. In the MainPage.xaml, locate the TitlePanel, the StackPanel added by default by the Page template:

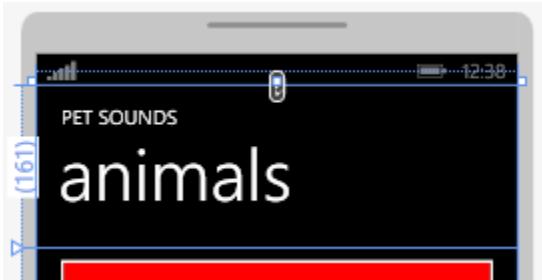
```
23     <!--TitlePanel contains the name of the application and page title-->
24     <StackPanel x:Name="TitlePanel"
25         Grid.Row="0"
26         Margin="12,17,0,28">
27         <TextBlock Text="MY APPLICATION"
28             Style="{StaticResource PhoneTextNormalStyle}"
29             Margin="12,0"/>
30         <TextBlock Text="page name"
31             Margin="9,-7,0,0"
32             Style="{StaticResource PhoneTextTitle1Style}"/>
33     </StackPanel>
34
```

... and we'll make the following changes:

```
22
23
24     <!--TitlePanel contains the name of the application and page title-->
25     <StackPanel x:Name="TitlePanel"
26         Grid.Row="0"
27         Margin="12,17,0,28">
28         <TextBlock Text="PET SOUNDS" ①
29             Style="{StaticResource PhoneTextNormalStyle}"
30             Margin="12,0"/>
31         <TextBlock Text="animals" ②
32             Margin="9,-7,0,0"
33             Style="{StaticResource PhoneTextTitle1Style}"/>
34
```

1. Change the text property representing the app's title to "PET SOUNDS". We're using all caps since that seems to be the convention used in Windows Phone apps.
2. Change the text property representing the page's title to "animals". We're using all lower-cased letters since that seems to be the convention as well.

The result:



Another small styling step, but again it makes the app feel more legitimate.

### 3. Understanding Binding Syntax and Static Resources

By default, the Style attribute of the second textbox is set to:

```
Style="{StaticResource PhoneTextTitle1Style}"
```

This will require a bit of explanation. First, whenever you see open and closed curly braces in XAML, it is referred to as "binding syntax". There are two types of binding syntaxes:

{StaticResource } - Let me start with the term "resource". A resource is an object that can be reused in different places in your application. Examples of resources include brushes and styles.

<http://msdn.microsoft.com/en-us/library/ms750613.aspx>

I created a simple Phone project called XAMLResources with the most simple {StaticResource} example I could think of:

```

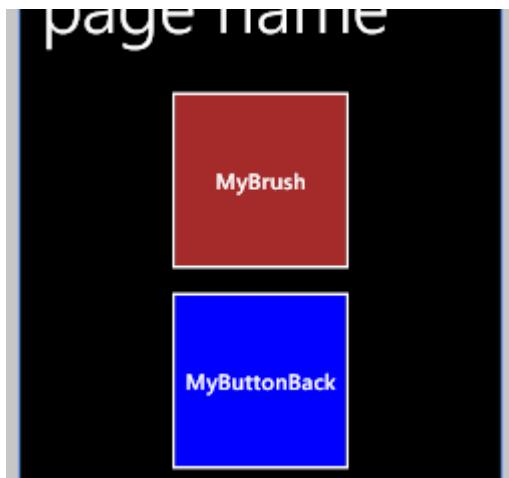
15  <!-- ----- -->
16  <phone:PhoneApplicationPage.Resources>
17      <SolidColorBrush x:Key="MyBrush" Color="Brown"/>
18      <Style TargetType="Button" x:Key="MyButtonBackground">
19          <Setter Property="Background" Value="Blue"/>
20      </Style>
21  </phone:PhoneApplicationPage.Resources>
22
23  <!-- LayoutRoot is the root grid where all page content is placed-->
24
25
26
27
28
29
30
31
32
33
34
35
36  <!--ContentPanel - place additional content here-->
37  <StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
38      <Button
39          Background="{StaticResource MyBrush}"
40          Content="MyBrush"
41          Height="200"
42          Width="200"
43          VerticalAlignment="top"/>
44      <Button
45          Style="{StaticResource MyButtonBackground}"
46          Content="MyButtonBackground"
47          Height="200"
48          Width="200" />
49  </StackPanel>
50

```

1. I add a child element of <phone:PhoneApplicationPage> to hold Local (or rather, page-level) Resources. Any brushes or styles created here will only be available on this page, not other pages.
2. I create a SolidColorBrush that I can reference with the key "MyBrush".
3. I create a Style that I can reference with the key "MyButtonBackground". Notice that the TargetType is set to Button ... this style only applies to Button controls. Inside of this style, I can set individual attributes of the Button. I have one attribute set, the Background attribute, which I set to the SolidColorBrush "Blue".
4. Here I bind my Button's Background attribute to the value bound to MyBrush.
5. Here I bind the style of my Button to MyButtonBackground.

In this overly simplistic example, it may not be readily apparent the value of this approach. As your application grows large and you want to keep a consistent appearance between the controls on the page, you may find this quite handy. It keeps the XAML concise and compact. And, if you need to change the style or color for any reason, you can change it once and have the change propagate to everywhere it has been applied.

Here's the result:



I created these Local Resources on the page, meaning they are scoped to just the MainPage.xaml. What if I wanted to share these Resources across the entire application? In that case, I would have defined them in the App.xaml's `<Application.Resources>` section as a System Resource:

```
App.xaml*  MainPage.xaml
1 <Application
2     x:Class="XAMLResources.App"
3     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5     xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6     xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone">
7
8     <!--Application Resources-->
9     <Application.Resources>
10    <local:LocalizedStrings xmlns:local="clr-namespace:XAMLResources" x:Key="LocalizedSt
11    |   |
12    |   </Application.Resources>           ← Red arrow points here
13
14    <Application.ApplicationLifetimeObjects>
15        <!--Required object that handles lifetime events for the application-->
16        <shell:PhoneApplicationService>
```

#### 4. Discovering Theme Resources

So, back in the PetSounds project, you may be wondering where the `PhoneTextTitle1Style` is defined. Actually, it's a "built-in style" as part of the Windows Phone Operating System's Theme Resources:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769552\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769552(v=vs.105).aspx)

The screenshot shows the Windows Phone Dev Center website. At the top, there's a navigation bar with links for Design, Develop, Publish, Community, and Dashboard. On the left, there are three buttons: 'SUBMIT APP', 'GET SDK', and 'VIEW SAMPLES'. The main content area features a large heading 'Theme resources for Windows Phone'. Below the heading, it says '5 out of 8 rated this helpful - Rate this topic' and 'April 08, 2013'.

Windows Phone | Dev Center

Search Dev Center with

Design Develop Publish Community Dashboard

SUBMIT APP  
GET SDK  
VIEW SAMPLES

## Theme resources for Windows Phone

5 out of 8 rated this helpful - [Rate this topic](#)

April 08, 2013

If you scroll down that page, you can see the Text styles available to Windows Phone apps:

## Text styles

The following table offers a selection of text styles that you can apply to a **TextBlock** control. Attribute = Style.

 Tip:

You can use Expression Blend for Windows Phone to preview text styles before applying it to an object.  
For more information, see [Expression Blend for Windows Phone Overview](#).

Name	Applies to:	Type	Description
PhoneTextBlockBase	TextBlock	Style	<b>FontFamily:</b> PhoneFontFamilyNormal <b>FontSize:</b> PhoneFontSizeNormal <b>Foreground:</b> PhoneForegroundBrush <b>Margin:</b> PhoneHorizontalMargin
PhoneTextNormalStyle	TextBlock	Style	<b>BasedOn:</b> PhoneTextBlockBase
PhoneTextSubtleStyle	TextBlock	Style	<b>BasedOn:</b> PhoneTextBlockBase <b>Foreground:</b> PhoneSubtleBrush
PhoneTextTitle1Style	TextBlock	Style	<b>BasedOn:</b> PhoneTextBlockBase <b>FontFamily:</b> PhoneFontFamilySemiLight <b>FontSize:</b> PhoneFontSizeExtraExtraLarge
PhoneTextTitle2Style	TextBlock	Style	<b>BasedOn:</b> PhoneTextBlockBase <b>FontFamily:</b> PhoneFontFamilySemiLight

These Themed Resources should be leveraged to keep your apps looking like they below on the Windows Phone. You should resist the urge to use custom colors, fonts and the like unless you have a good reason to (such as to match your company's established branding elements, etc.).

It's also worth noting that many of the styles are based on styles which are based on yet other styles. This visual inheritance allows developers to avoid repeating the attribute settings that will be common across variations of the styles, much like how Cascading Style Sheets work in web development.

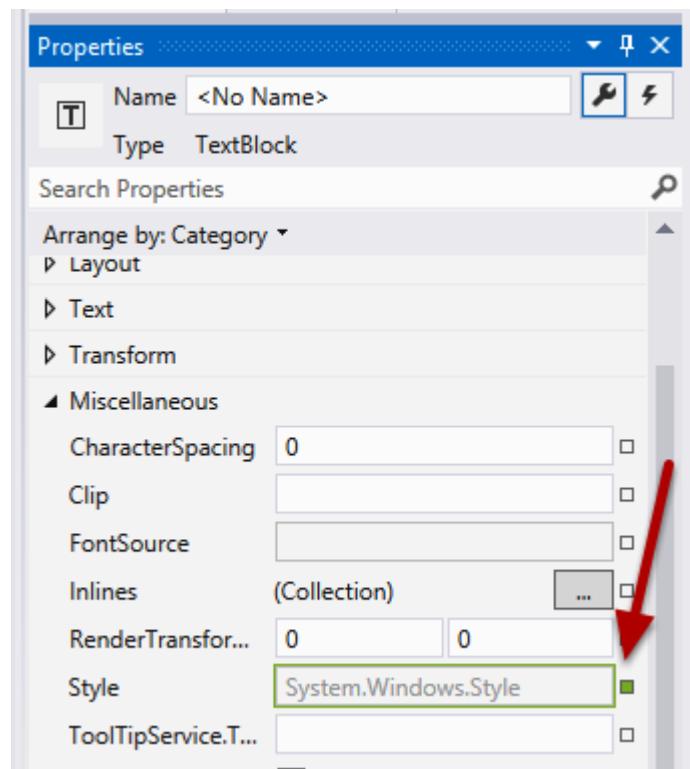
I said earlier that there were two binding expressions in the Windows Phone, the second is {Binding } ... this is used for binding data (i.e., usually generic lists of custom types with their properties set to the data we want to work with in our app) to on page elements. We'll see this at work much later in this series.

## 5. Customizing a Theme Resource by creating a style based on it

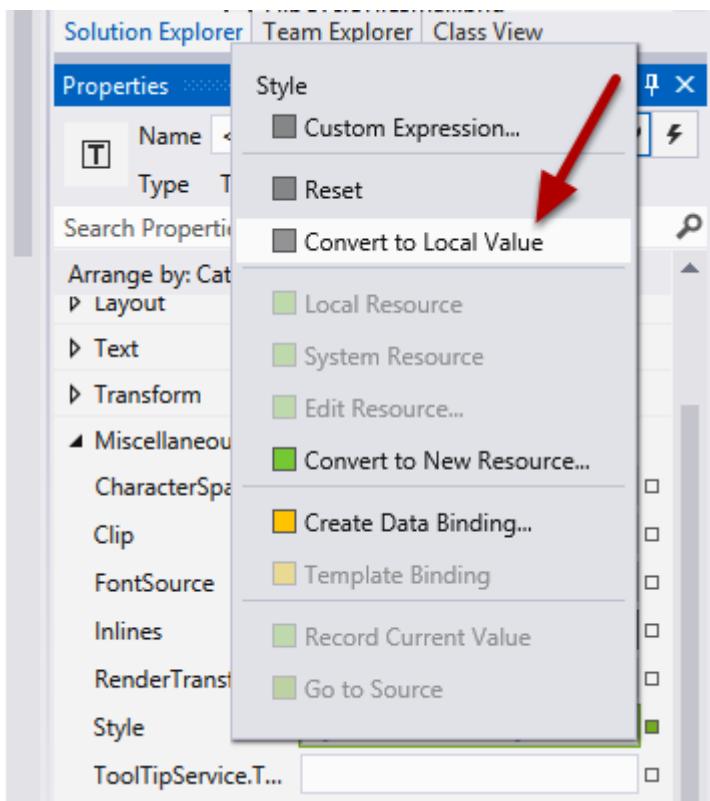
Let's have a little fun. As I said earlier, you typically want to stick with the Phone's Theme Resources. However, we can edit a style if we would like to. I think this might provide an additional insight or two on how this all works.

Make sure your mouse cursor is in the TextBlock with the Text attribute set to "animals".

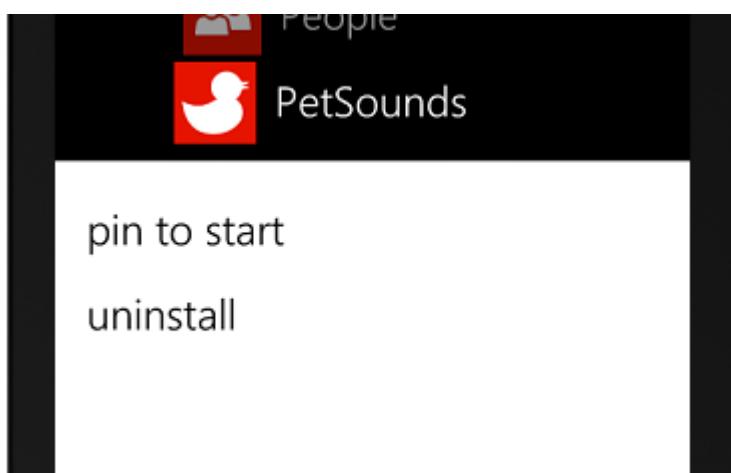
In the Properties window, navigate to the Miscellaneous section, and locate the Style property:



Notice how there's a green border surrounding the text box, and the icon to the right is filled with the same green color. If you click that square or the text box you'll see a context menu:



Here we could potentially change the binding. In fact, I'll click the option "Convert to Local Value". When I do that, notice that the Style property is gone. In its place is a complex property setting for <TextBlock.Style> with a <Style> definition therein:



As you can see, when we converted from the Themed Style to a Local Style, we see part of the definition of that Theme Style ... it's based on the PhoneTextBlockBase Theme Style, and it overrides that style by setting two additional properties: `FontFamily` and `FontSize`. Both of these are defined as Theme Styles as well. Let's override those settings with our own:

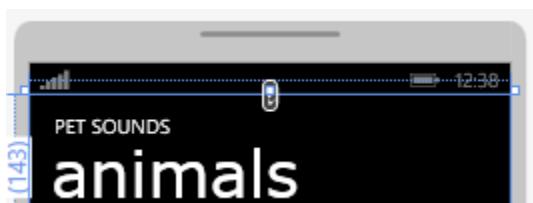
```
32 <TextBlock.Style>
33   <Style BasedOn="{StaticResource PhoneTextBlockBase}"
34     TargetType="TextBlock">
35     <Setter Property="FontFamily"
36       Value="Verdana"/>
37     <Setter Property="FontSize"
38       Value="64"/>
39   </Style>
40 </TextBlock.Style>
```

1

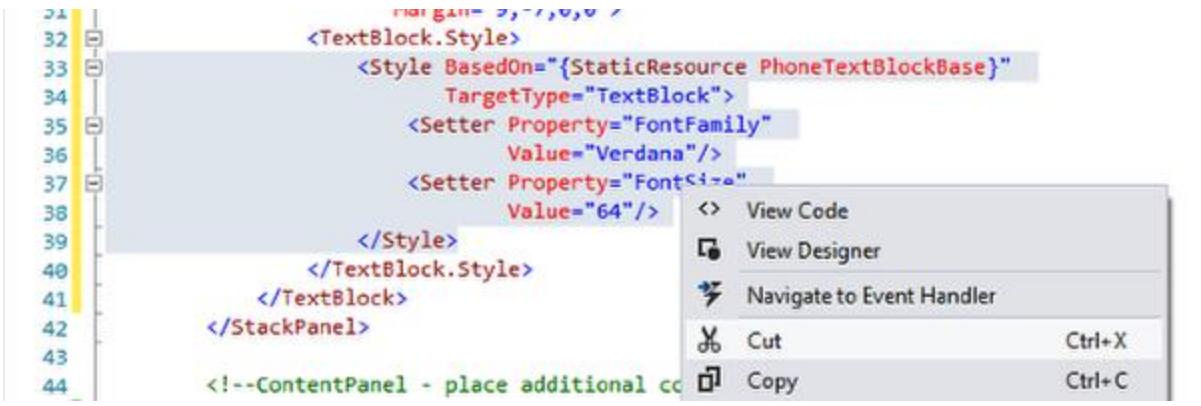
2

1. Set `FontFamily` to "Verdana"
2. Set `FontSize` to 64

That produces the following:



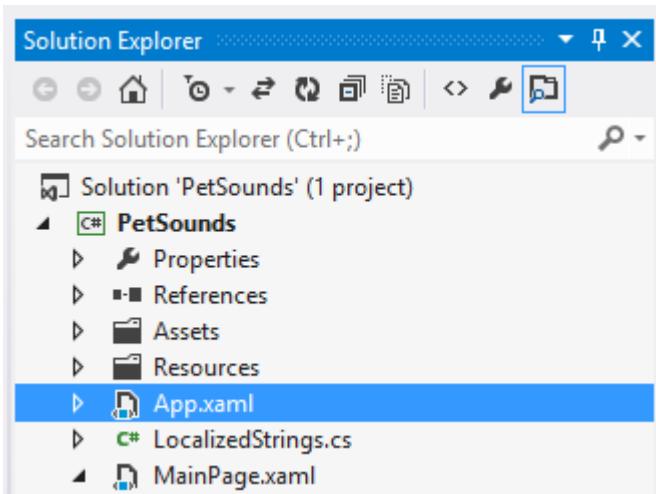
Next, let's make this style available to our entire app by making it a System Resource. I'll highlight everything between the `<TextBlock.Style>` and `</TextBlock.Style>` tags and cut them:



```
31
32     <TextBlock>
33         <TextBlock.Style>
34             <Style BasedOn="{StaticResource PhoneTextBlockBase}" TargetType="TextBlock">
35                 <Setter Property="FontFamily" Value="Verdana"/>
36                 <Setter Property="FontSize" Value="64"/>
37             </Style>
38         </TextBlock.Style>
39     </TextBlock>
40 </StackPanel>
41
42 <!--ContentPanel - place additional controls here-->
43
44
```

A screenshot of the Visual Studio code editor showing XAML code. A context menu is open over the line of code starting with '<TextBlock>'. The menu items are: View Code, View Designer, Navigate to Event Handler, Cut (Ctrl+X), and Copy (Ctrl+C). The 'Copy' option is highlighted.

... then I'll open up the App.xaml file:



... and paste them into the <Application.Resources> section (see lines 12 through 19, below). I also add an attribute:

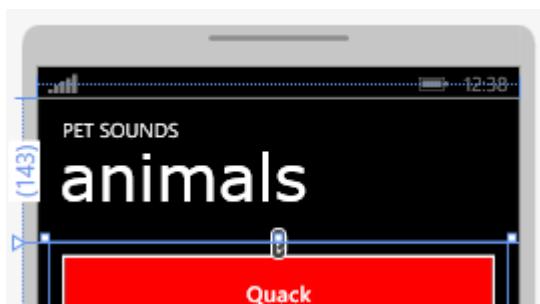
x:Name="MyTitleText":

```
8    <!--Application Resources-->
9    <Application.Resources>
10   <local:LocalizedStrings xmlns:local="clr-namespace:PetSounds" x:Key="LocalizedStrings">
11
12   <Style x:Name="MyTitleText" ↖
13     BasedOn="{StaticResource PhoneTextBlockBase}"
14     TargetType="TextBlock"
15     <Setter Property="FontFamily"
16       Value="Verdana"/>
17     <Setter Property="FontSize"
18       Value="64"/>
19   </Style>
20
21 </Application.Resources>
```

Now, I can return to MainPage.xaml and re-write the TextBlock to use the new Application Resource:

```
30   <TextBlock Text="animals"
31     Margin="9,-7,0,0"
32     Style="{StaticResource MyTitleText}" />
```

... and the result should not change:



Success!

Recap

Just to recap, the big takeaway from this lesson is how we can style our app to make it look like it belongs on the Windows Phone while also expressing our own individuality. We learned how to modify the WMAppManifest.xml file to change the icons and the title of our app, we changed the app's title and page title, and learned how to bind to

StaticResources like Themed Resources for the Windows Phone, and how to create both Local and System Resources based on Themed Resources, and much more.

# Part 7: Localizing the App

Source Code: <http://aka.ms/absbeginnerdevwp8>

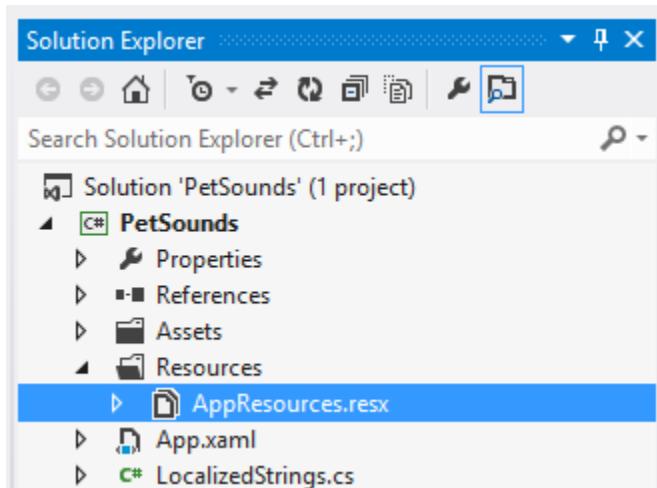
In this lesson I'll explain how to localize your app ... that simply means that we can present the various bits of text in your app in different languages. For example, I may want to support both English speakers and Spanish speakers opening up the number of markets I could potentially sell my app in. Now, there's a bit more to selling in different markets than merely localizing my app, however this would be a big step in the right direction.

Our game plan in this lesson:

1. We'll learn about the AppResources.resx file, and how it allows us to store name / value pairs that we can access in our declarative XAML code (or imperative C# code).
2. We'll see how to create language and region specific variations of the file in Visual Studio.
3. We'll learn how the Windows Phone 8 Operating System will choose the correct version of our AppResources.resx file based on the user's language and region.

1. Modify the AppResources.resx settings and bind to its values

If you expand the Resource folder of our PetSounds project, you'll see a file called AppResources.resx:



If you double-click to open it, it will open with a special designer in the main area:

The screenshot shows the 'AppResources.resx' file in the Visual Studio resources editor. The table contains the following data:

	Name	Value
▶	AppBarButtonText	add
	AppBarMenuItemText	Menu Item
	ApplicationTitle	MY APPLICATION
	ResourceFlowDirection	LeftToRight
*	ResourceLanguage	en-US

This file is a series of Name / Value pairs. On the left are names of settings we will bind to. On the right, the settings for a given language. Which language? See the very last attribute:

ResourceLanguage

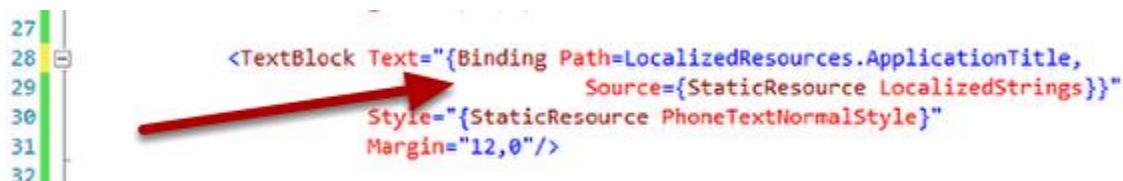
... is set to ...

en-US

"en" means "English". "US" is the region and in this case means "USA". Therefore, these settings should be used in the USA for English speakers. It's also the default AppResources file (you'll see the difference between the default and other languages / regions we support in just a moment).

The Name / Value pairs are mostly snippets of text we'll use throughout the app. However, the "ResourceFlowDirection" is a setting for which direction the characters should be presented. As you know, some languages are read from RightToLeft and this setting would be used throughout the app for this purpose.

As you can see, there's an ApplicationTitle setting set to "MY APPLICATION". What if I wanted to convert the TextBlock on our MainPage.xaml to utilize the AppResources setting? I would use a binding expression like so:



As you can see, I've replaced the hardcoded text to this binding expression:

```
Text="{Binding Path=LocalizedResources.ApplicationTitle,Source={StaticResource LocalizedStrings}}"
```

There's several things at work here that would require a lot of background explanation. For now, just know we're using a binding expression to bind data to an attribute, the Text. The Path attribute of the binding expression refers to the LocalizedStrings.cs file in our project. It creates an instance of an AppResources object which provides us access to the appropriate AppResources file based on the region and preferred language of the Phone's user. We'll see it all come together in a bit. The .ApplicationTitle references the specific Name entry in the AppResources.resx file. The Source attribute is bound to LocalizedStrings ... again, that's where the compiler can find the source of the LocalizedResources property ... it's part of the LocalizedStrings class in the LocalizedStrings.cs file.

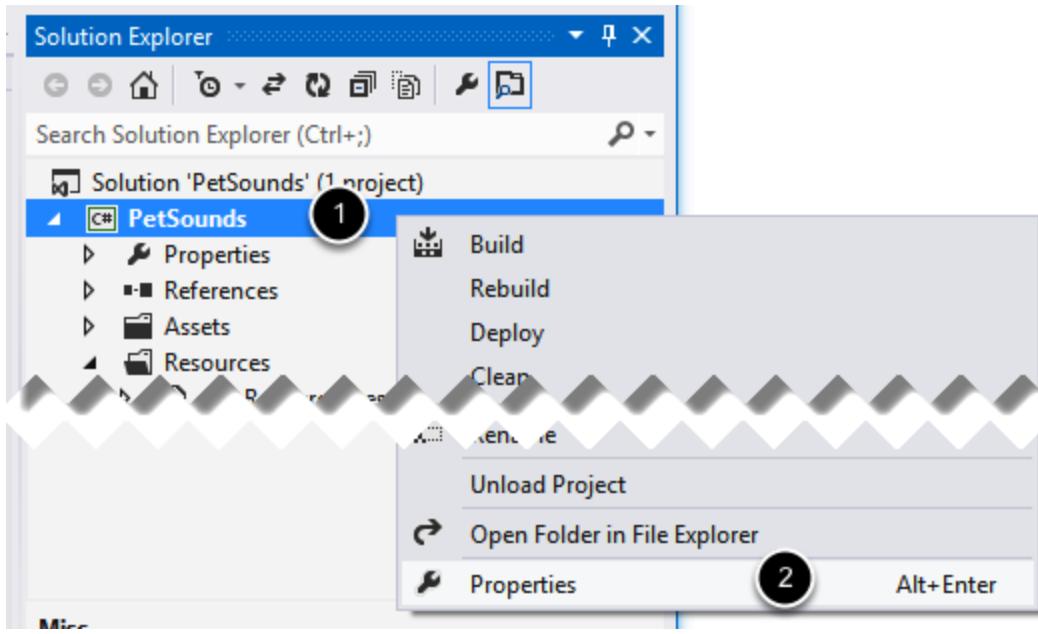
For a more in-depth discussion of how all these ideas interrelate, I would encourage you to check out a great article on MSDN:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff637520\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff637520(v=vs.105).aspx)

The screenshot shows the Windows Phone Dev Center homepage. At the top, there's a navigation bar with links for Design, Develop, Publish, Community, and Dashboard. Below the navigation, there are three buttons: 'SUBMIT APP', 'GET SDK', and 'VIEW SAMPLES'. To the right of these buttons, the title 'How to build a localized app for Windows Phone' is displayed. The page content is partially visible, showing the beginning of the article.

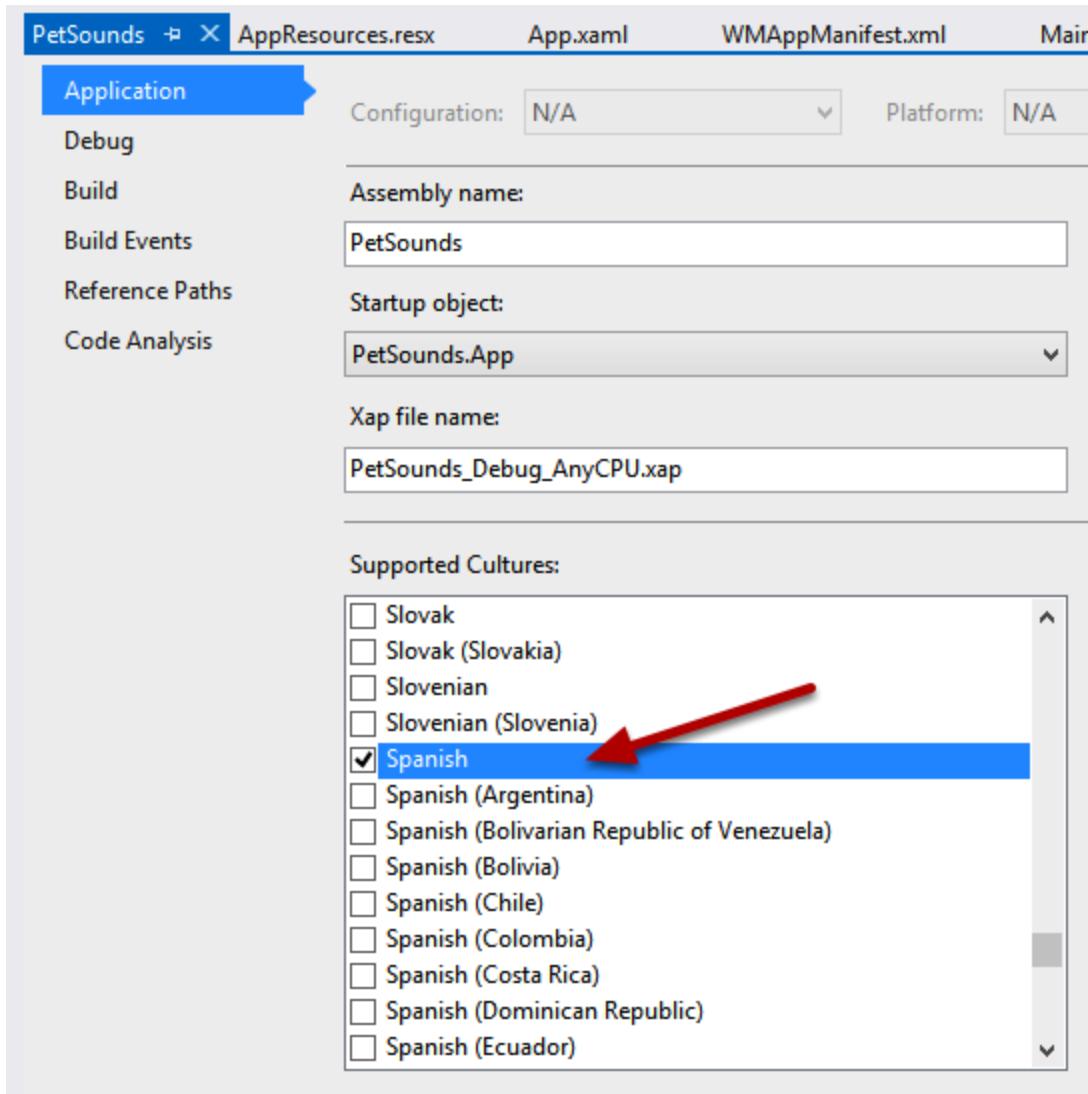
2. Add support for a second language

At this point, we're only supporting English (in the USA). I would like to support the Spanish language no matter which region of the world the user resides. To do that ...



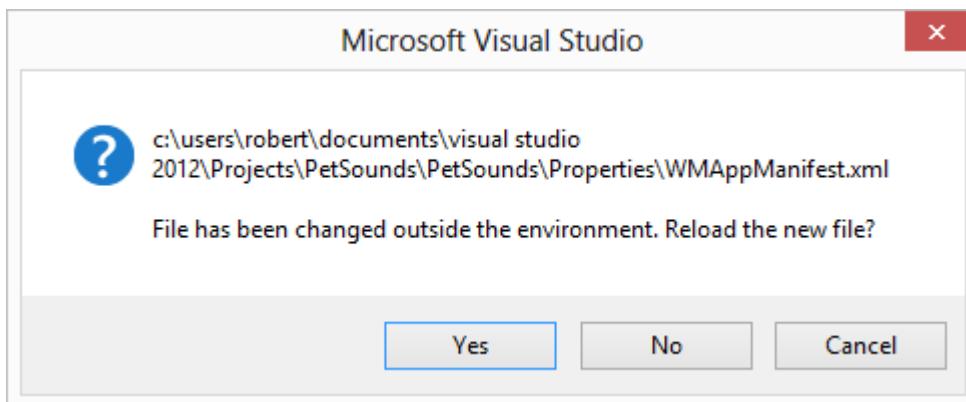
1. In the Solution Explorer, right-click on the Project title PetSounds
2. Select "Properties" from the context-menu

The Project Properties designer appears with the Application tab on the right already selected (if it's not, select it!)



Under "Supported Cultures", place a checkmark next to "Spanish".

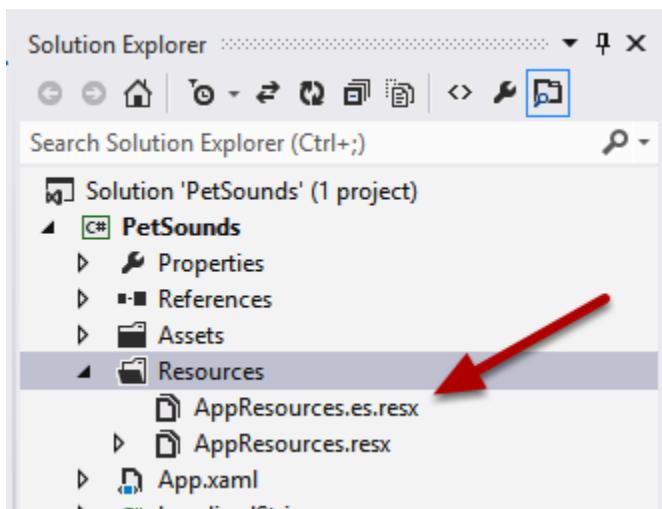
Next, save the project and CLOSE the Project Properties tab. It's possible you will see the following message (depending on what you currently have loaded into the main area of Visual Studio):



If you see this, you can click the "Yes" button.

Now, in the Resources folder, you should see a new, second file appear:

AppResources.es.resx



The .es suffix to the AppResources file indicates the language, Spanish (es for "Espanol"). Double-click that file to open it in the main area:

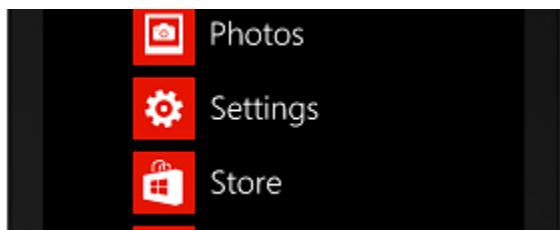
	Name	Value
▶	AppBarButtonText	add
	AppBarMenuItemText	Menu Item
	ApplicationTitle	MY APPLICATION
	ResourceFlowDirection	LeftToRight
*	ResourceLanguage	es

By default, it's in English. With my son's help—he's a third-year high school Spanish student—I translated the Values from English to Spanish. NOTE: The Names MUST BE IDENTICAL IN BOTH THE ENGLISH AND SPANISH VERSIONS OF THIS FILE. Do not translate the Names ... only the Values! The Names are tokens used programmatically. The values will be displayed to the end user:

	Name	Value
	AppBarButtonText	añadir
	AppBarMenuItemText	Elemento de Menú
▶	ApplicationTitle	Pet Sonida
	ResourceFlowDirection	LeftToRight
	ResourceLanguage	es

### 3. Test the different language version of the app

After I've completed the translations, run the app (F5) and use the Phone Emulators' Start button, then I swipe to the alphabetical listing of apps, and scroll down to the Settings app and tap to open it:



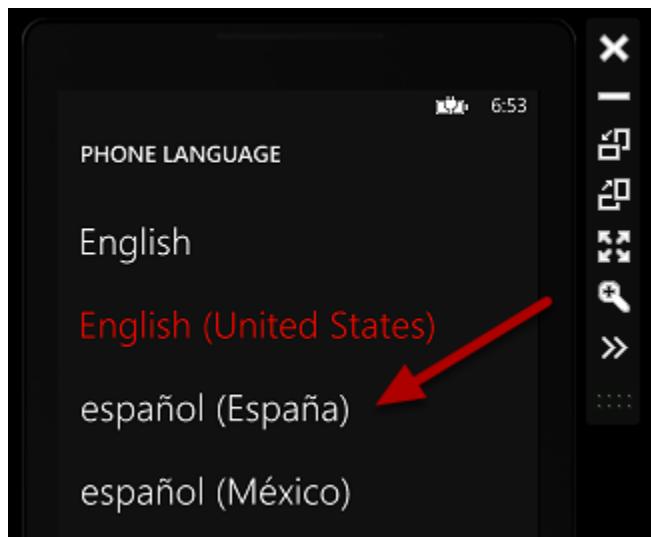
On the "system" page of the Settings app, I scroll down to "language+region" and tap that option:



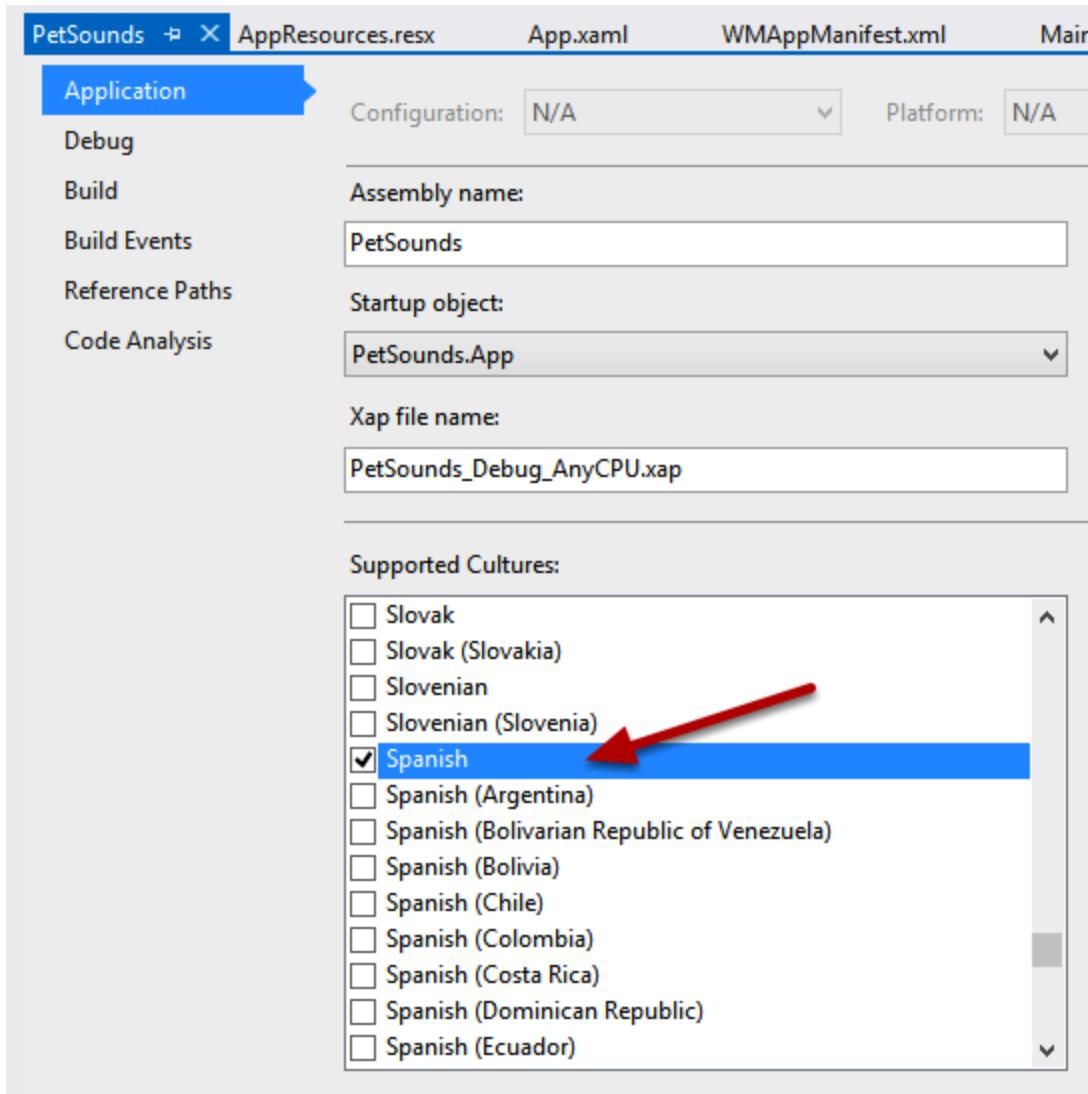
On the "language+region" page, I change the click the Phone Language to edit it from "English (United States)" ...



... and in the list of languages and regions, I select "espanol (Espana)":



I then scroll to the bottom of the page to see the "restart phone" button. Note the message that says "restart required":



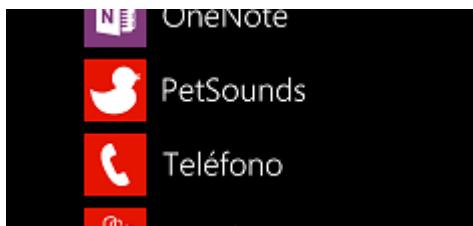
When I restart, I see the message "hasta luego":

hasta luego

At this point Visual Studio may disconnect ... that's ok, you don't need to be in Debug Mode in Visual Studio ... just focus on the Phone Emulator for now. The Phone Emulator will use the last deployed version of the app.

Once the Phone Emulator reboots, you can swipe over to the list of apps to see that they are all in Spanish.

Tap the "PetSounds" app ...



... and when the app opens, you can see the Title has been replaced with the title we entered in the AppResources.es.resx file.



Outstanding! Now, I would simply need to repeat these steps ... especially about creating Name / Value pairs and using binding expressions throughout the app in order to completely localize the app's user interface.

Unless you're bilingual, you'll probably want to re-trace your steps and reset the Phone Emulator to use English. Alternatively, you can completely shut down the Phone Emulator (in the Windows task bar, right-click, select Close) and Visual Studio will restart the Phone Emulator with the default settings.

### Recap

To recap, the big takeaway in this lesson was how to localize your app using AppResources files, the Project Properties, and some binding syntax. We saw how we could set the language and region of our phone to test our localization customizations as well.



# Part 8: Understanding Compilation and Deployment

Source Code: <http://aka.ms/absbeginnerdevwp8>

If you'll recall from the C# Fundamentals series, the C# compiler compiles the code in your project to create a .NET Assembly. The end result is usually (at least, in the case of simple Console applications) an executable file with the file extension ".exe". While we worked exclusively with the debug version of the app in Visual Studio, by changing the Solution Configuration to Release we could create a version of the application suitable for deployment on another user's computer ... as long as they had the same version of the .NET Framework Runtime installed on that computer.

In this lesson, I want to talk about deploying the app we build to a physical device running the Windows Phone 8 Operating System. Up until now, we've merely deployed to the Phone Emulator. The compilation and deployment step is automated for us, and we may not be aware of how the program is packaged and installed into the Emulator. Furthermore, we will want to understand the way in which the app is packaged because we'll want to undoubtedly deploy the app to a physical phone device for testing, and we'll want to package the app so that we can submit it to the Windows Store to be approved and included there for download or sale.

So our game plan in this lesson:

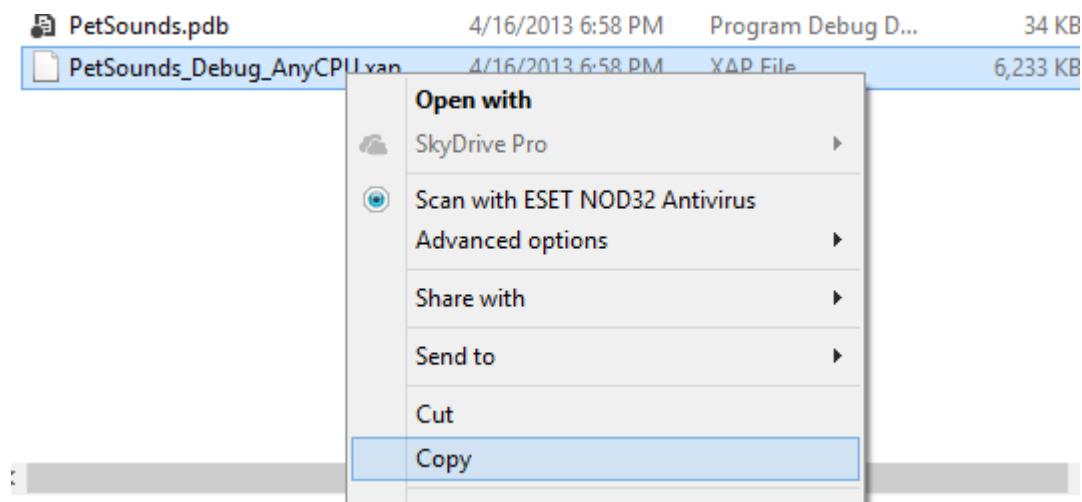
1. I want us to see what happens when we compile our app ... what does Visual Studio create? Maybe we can learn a little about the deployment process as a result.
  2. I want to deploy to an actual physical phone device, just to see our PetSounds app running on a real phone
1. Discovering what happens during compilation and deployment
- Each time we run the app by hitting F5 on our keyboard, or the Run button on the toolbar, Visual Studio is creating a Debug version of our app. If you'll recall from the C# Fundamentals series, it creates a Bin\Debug directory where it places the .NET assembly as well as any additional files required for the app to run. Here's what the Bin\Debug directory looks like for our PetSounds app:

Documents > Visual Studio 2012 > Projects > PetSounds > PetSounds > Bin > Debug				
	Name	Date modified	Type	Size
ds	Assets	4/16/2013 4:08 PM	File folder	
laces	es	4/16/2013 6:51 PM	File folder	
	Properties	4/16/2013 4:08 PM	File folder	
	AppManifest.xaml	4/16/2013 6:51 PM	Windows Markup ...	1 KB
	PetSounds.dll	4/16/2013 6:58 PM	Application extens...	15 KB
	PetSounds.pdb	4/16/2013 6:58 PM	Program Debug D...	34 KB
	PetSounds_Debug_AnyCPU.xap	4/16/2013 6:58 PM	XAP File	6,233 KB

We see the .NET assembly (PetSounds.dll) and a file that we can ignore that helps coordinate Visual Studio's debugger with the running version of the app (PetSounds.pdb). There's some folders that match ones in our Visual Studio project, like the Assets folder, the es folder for the Espanol version of our AppResources.resx, and a Properties folder. Then there's a AppManifest file and a PetSounds\_Debug\_AnyCPU.xap.

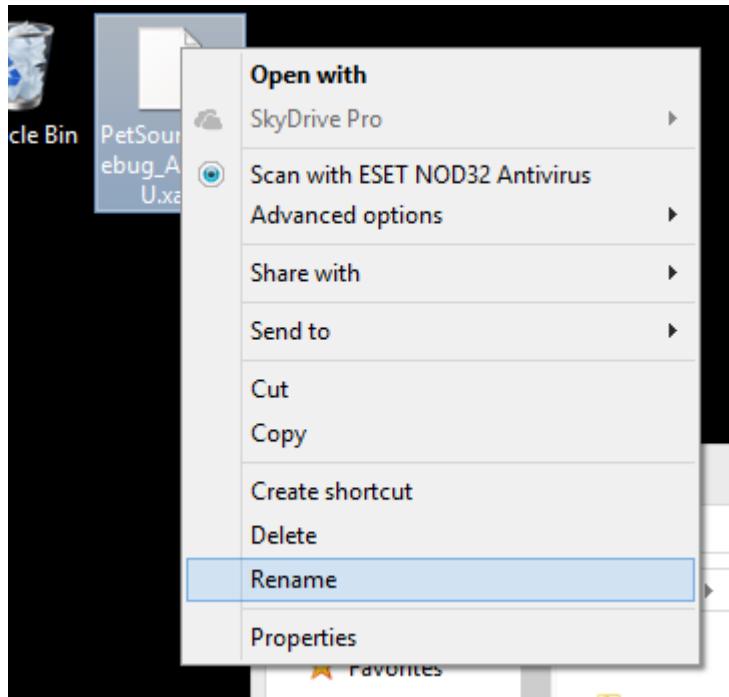
The .xap file is quite large (6 MB). I happen to know that file extension indicates that it is the deployment package ... a file containing all the files and configurations required to deploy our app on the Windows Phone 8 OS.

Let's have a little fun. I'm going to copy that file to my Desktop ... I right-click the file, select Copy ...

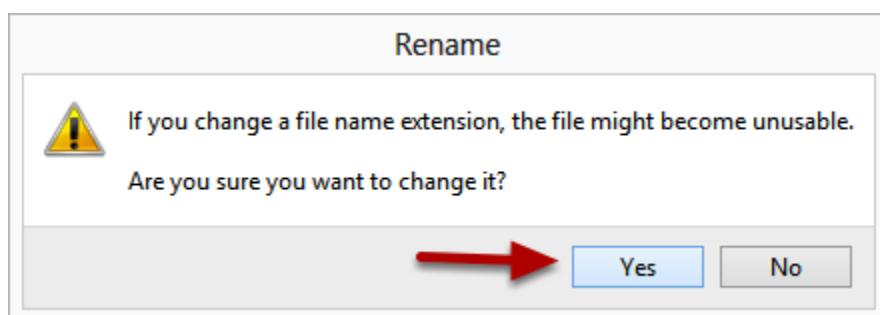


... I right-click my Desktop and select Paste.

Once the file is on my Desktop, I right-click it and select Rename ...

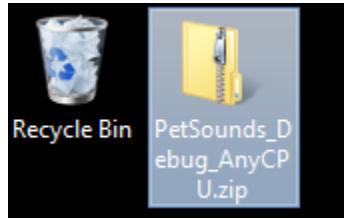


... and I change the file EXTENSION from .xap to .zip ... Windows warns me that this might harm the file:



... this is only a copy, and Visual Studio creates a new .xap each time we deploy, so I select the Yes option in the dialog.

The file now looks like any other .zip file on my computer.

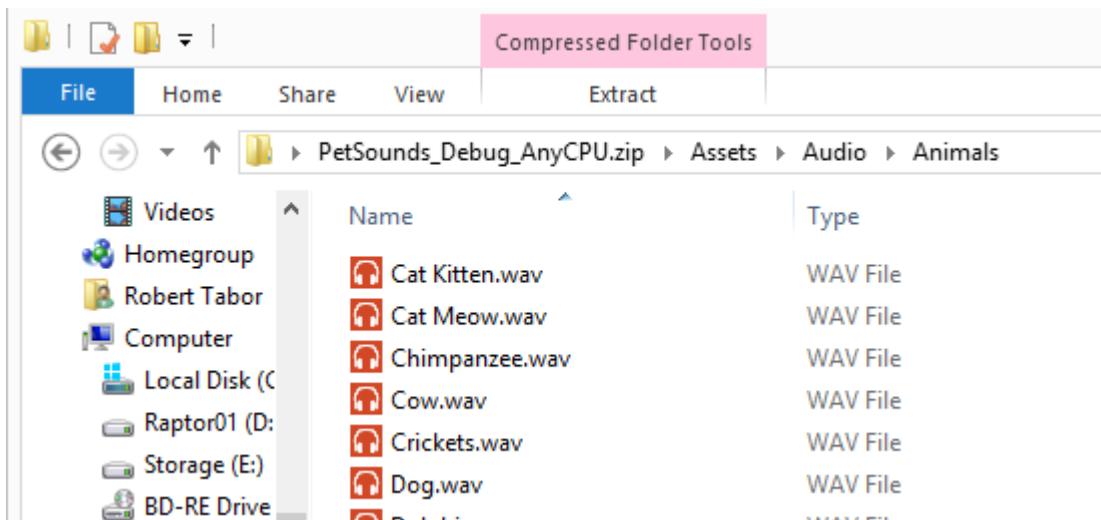


If I select the file in Windows Explorer, I can see its contents (even if I don't choose to Extract the files):

The screenshot shows a Windows Explorer window with the title bar 'PetSounds\_Debug\_AnyCPU.zip'. The file list contains the following items:

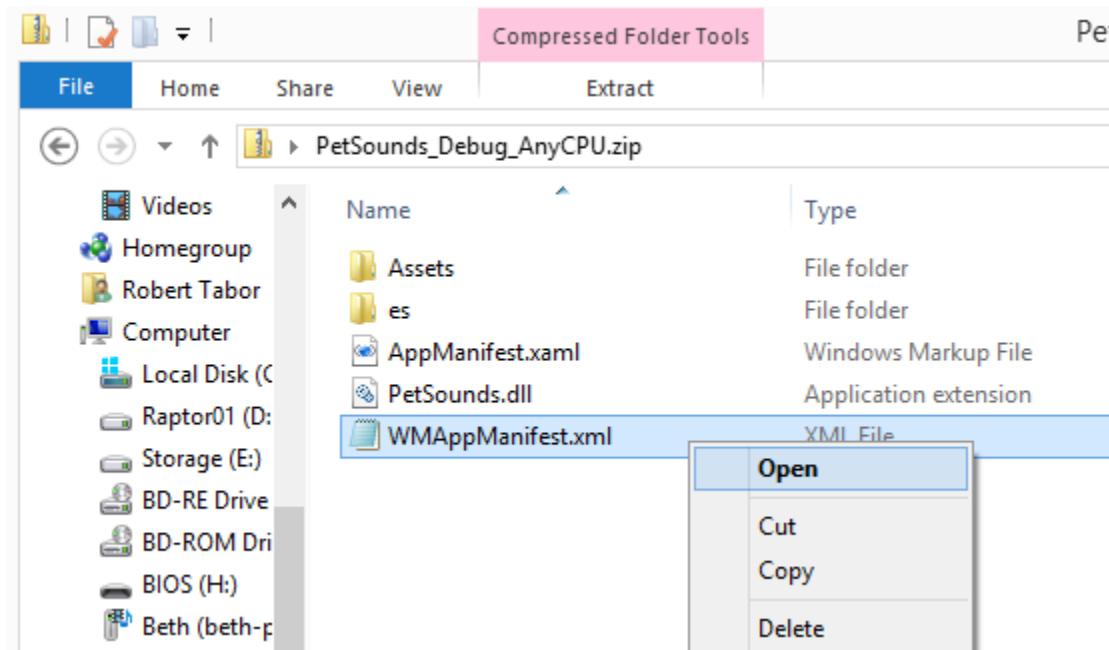
Name	Type	Compressed size
Assets	File folder	
es	File folder	
AppManifest.xaml	Windows Markup File	1 kB
PetSounds.dll	Application extension	6 kB
WMAppManifest.xml	XML File	1 kB

Wow, the .xap file is actually a fancy .zip file containing (essentially) what we saw in the \Bin\Debug directory of our Project! Drilling into the /Assets sub-directory ...

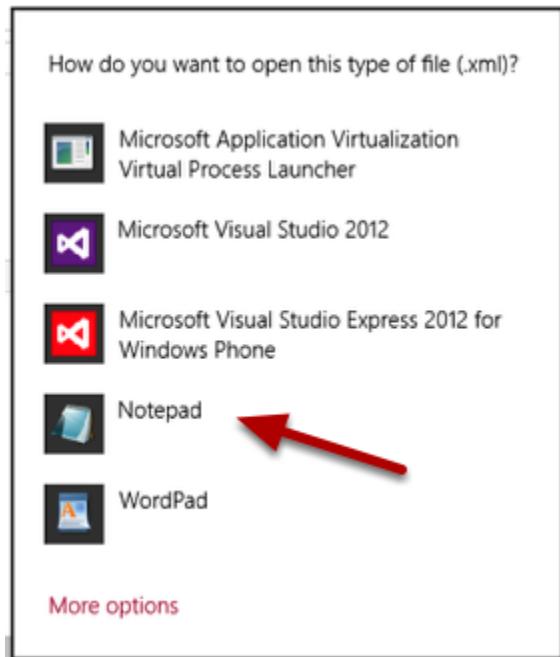


... and eventually into the /Audio and /Animals folder structure, I see the .wav files that we copied into the project a few lessons ago in there.

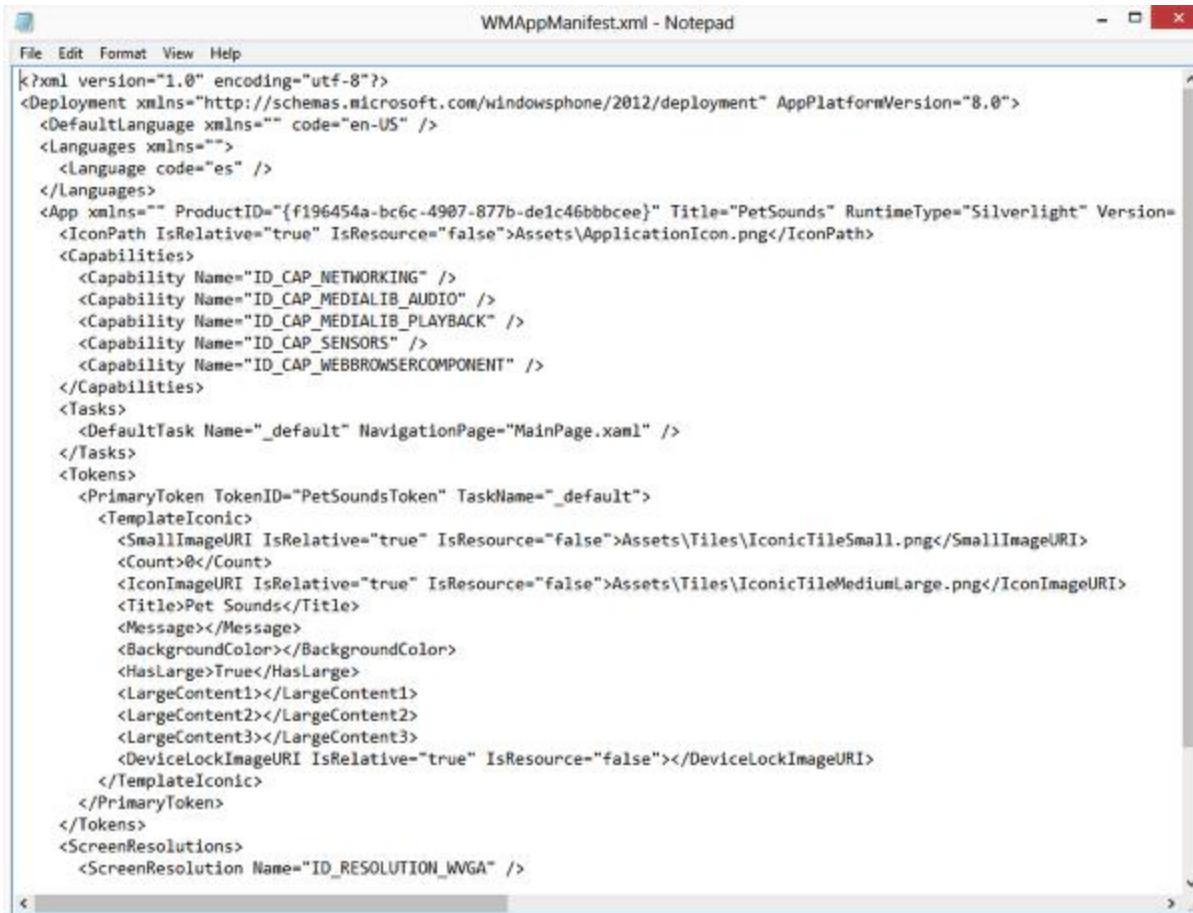
Let's back out to the root of that .zip file ... I want to see what's inside the AppManifest.xaml and WMAppManifest.xml files ... I right-click on the WMAppManifest.xml file ...



You may be asked which application in Windows 8 you want to use to open this file. Choose Notepad.



In Notepad you can see the true nature of the WMAppManifest.xml file ... it's simply XML ... just a lot of it. In Visual Studio, this complexity is hidden from us through a friendly "designer" ... a page in Visual Studio that restricts the changes we can make to the file.



```
WMAppManifest.xml - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="utf-8"?>
<Deployment xmlns="http://schemas.microsoft.com/windowsphone/2012/deployment" AppPlatformVersion="8.0">
  <DefaultLanguage xmlns="" code="en-US" />
  <Languages xmlns="">
    <Language code="es" />
  </Languages>
  <App xmlns="" ProductID="{f196454a-bc6c-4907-877b-de1c46bbbce}" Title="PetSounds" RuntimeType="Silverlight" Version="<IconPath IsRelative="true" IsResource="false">Assets\ApplicationIcon.png</IconPath>
    <Capabilities>
      <Capability Name="ID_CAP_NETWORKING" />
      <Capability Name="ID_CAP_MEDIALIB_AUDIO" />
      <Capability Name="ID_CAP_MEDIALIB_PLAYBACK" />
      <Capability Name="ID_CAP_SENSORS" />
      <Capability Name="ID_CAP_WEBBROWSERCOMPONENT" />
    </Capabilities>
    <Tasks>
      <DefaultTask Name="_default" NavigationPage="MainPage.xaml" />
    </Tasks>
    <Tokens>
      <PrimaryToken TokenID="PetSoundsToken" TaskName="_default">
        <TemplateIconic>
          <SmallImageURI IsRelative="true" IsResource="false">Assets\Tiles\IconicTileSmall.png</SmallImageURI>
          <Count>0</Count>
          <IconImageURI IsRelative="true" IsResource="false">Assets\Tiles\IconicTileMediumLarge.png</IconImageURI>
          <Title>Pet Sounds</Title>
          <Message></Message>
          <BackgroundColor></BackgroundColor>
          <HasLarge>True</HasLarge>
          <LargeContent1></LargeContent1>
          <LargeContent2></LargeContent2>
          <LargeContent3></LargeContent3>
          <DeviceLockImageURI IsRelative="true" IsResource="false"></DeviceLockImageURI>
        </TemplateIconic>
      </PrimaryToken>
    </Tokens>
  <ScreenResolutions>
    <ScreenResolution Name="ID_RESOLUTION_WVGA" />
  </ScreenResolutions>
```

The real question is: "why does it exist?"

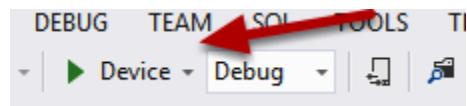
The purpose of the WMAppManifest.xml is to introduce your app to the Phone. It tells the Phone which images to use as Tiles on the Start and Application pages. It tells the Phone which capabilities we hope to use and which languages we can support. It tells the Phone what our name is, and which version of the app this is, which screen resolutions we support, and so on. It is how we integrate our app into the Windows Phone 8 Operating System and ecosystem of apps.

### 3. Deploying to a physical phone

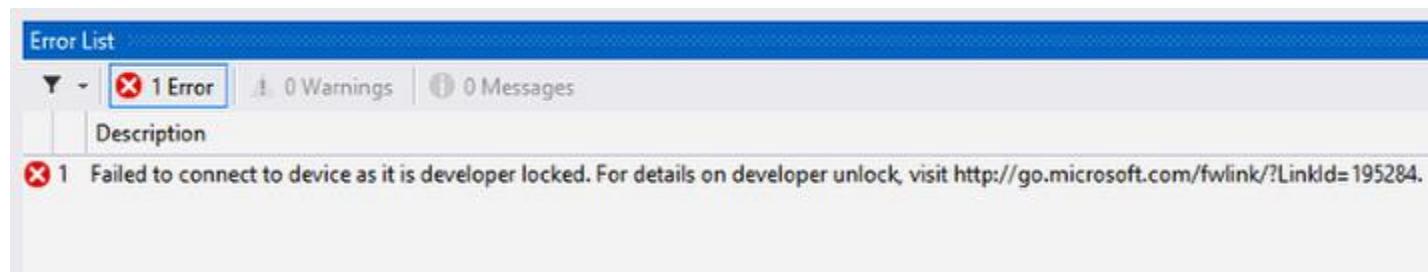
The final thing I want to try in this lesson is to deploy the app to my Lumia 920 phone. The rest of this lesson assumes you already have a Windows Phone Dev Center membership. It costs \$99 per year.

First, I plug my phone into my computer using the USB cord that comes with the phone. I've done this many times before to transfer my music or charge the phone, but never with the intent of deploying an app to it.

In Visual Studio, I'll change from debugging in the Emulator to debugging on the Device using the little down-arrow next to the Run button to choose that option:



Next, I'll click the Run button on the toolbar, but will get an error:

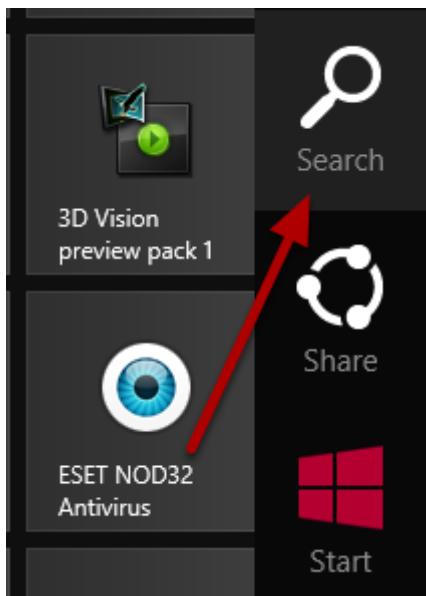


So, the first thing I need to do is to unlock the phone for development. The URL in that message doesn't work ... use this URL instead to learn more:

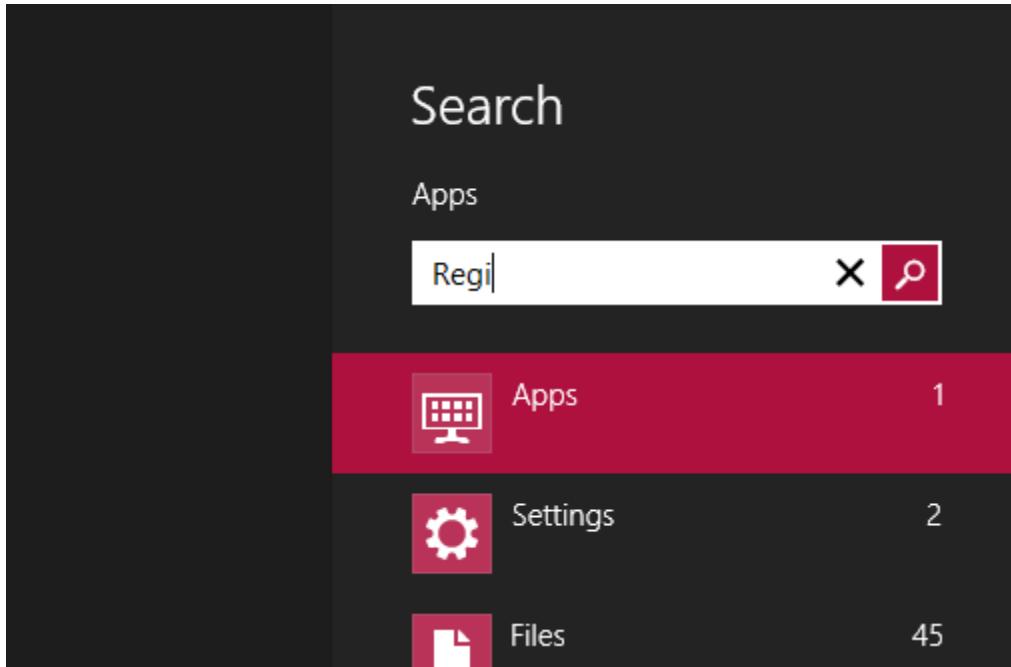
[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769508\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769508(v=vs.105).aspx)

I'll search for the Windows Phone Developer Registration app that was installed when I installed the Windows Phone 8 API on my computer.

In Windows 8, go to the Search charm ...



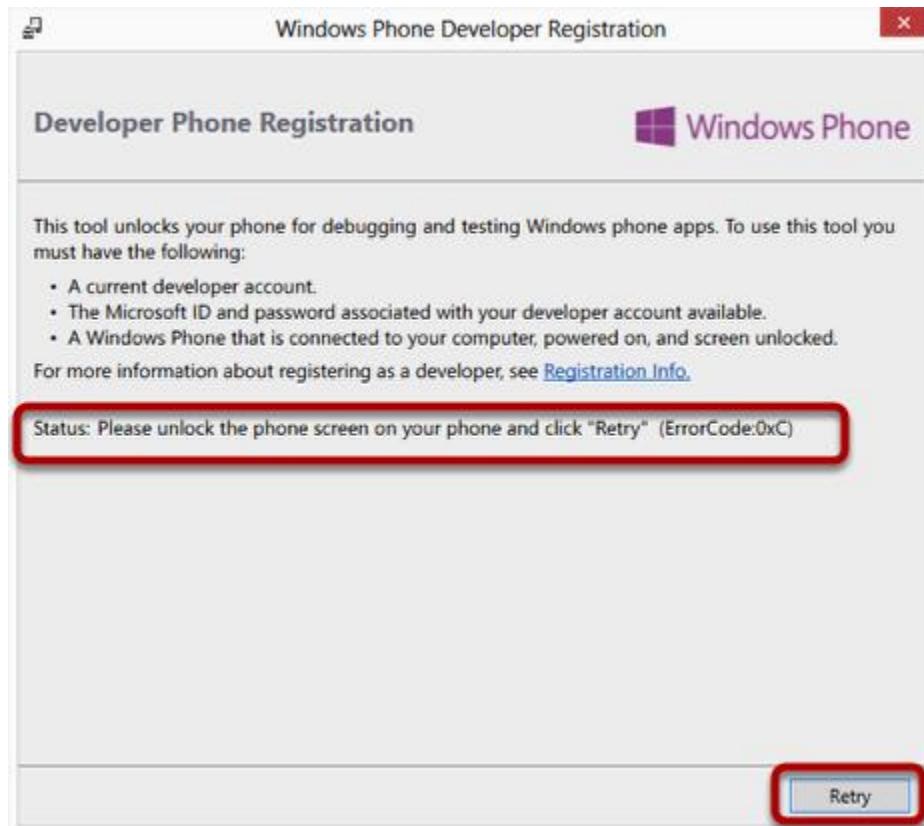
... and type in "Regi" ...



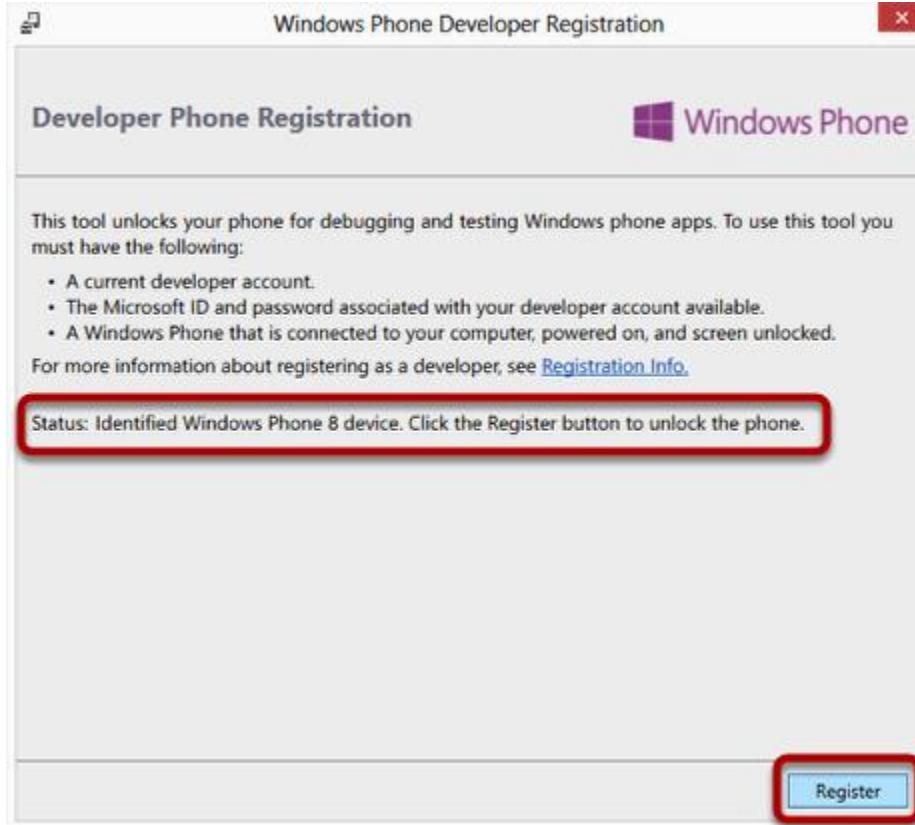
... that should be enough to locate the Windows Phone Developer Registration app ...



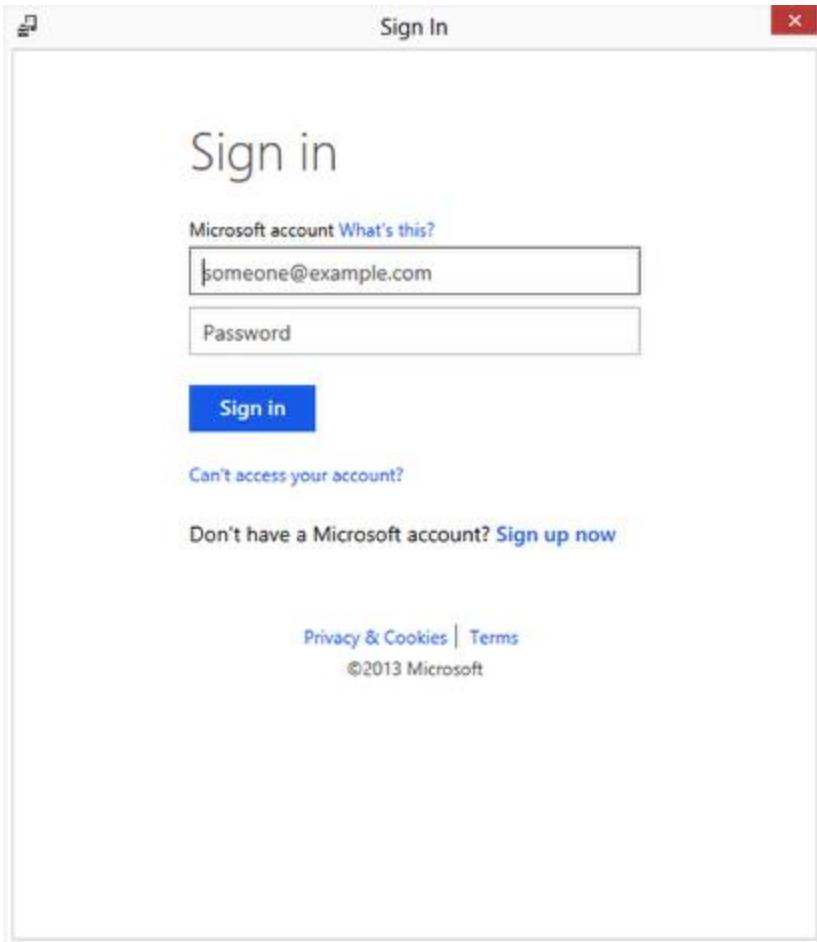
Start the app. It will take you back to the Desktop and display the following dialog ... as you can see, it tried to determine the status of my phone, however the lock screen was locked.



I unlock the lock screen and click the Retry button.

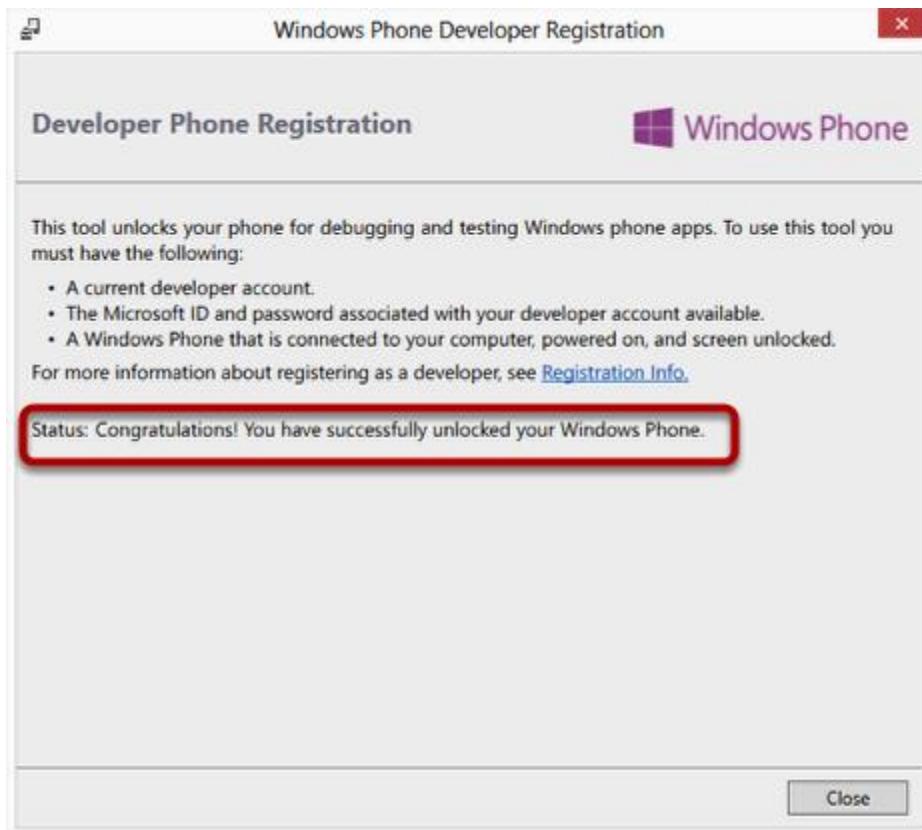


I click The Register button button and it asks me to Sign In to my Microsoft account ...



... Just a note, this always require I sign in twice for some reason. Don't be alarmed if you experience a similar behavior. You can usually get in on the second try.

After a moment, I get a confirmation that the phone was successfully unlocked for development:



I can confirm this by logging into:

<http://dev.windowsphone.com>

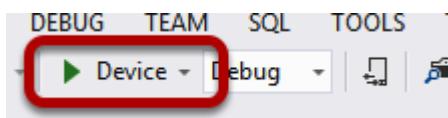
The screenshot shows the Windows Phone Dev Center dashboard. At the top, there are links for 'SUBMIT APP', 'GET SDK', and 'VIEW SAMPLES'. Below these, under the 'Account' section, there is a link to 'Account summary'. Under the 'Phones' section, there is a link to 'Phones'. On the right, a table lists registered phones: 'Bob Tabor's Phone' with a registered date of '5/8/2013' and an expiration date of '5/8/2015'. A 'Remove' button is next to the phone name. The number '4' is circled around the phone entry in the table.

Phone name	Registered date	Expiration date
Bob Tabor's Phone	5/8/2013	5/8/2015

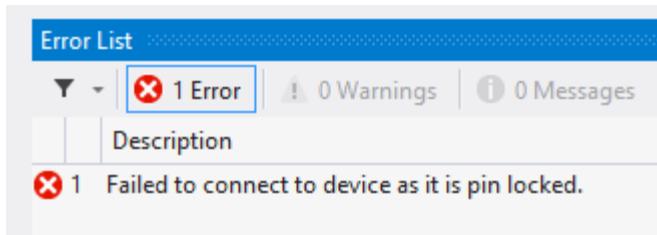
- (1) I navigate to the Dashboard
- (2) Account
- (3) Phones
- (4) The phone I just registered shows up there correctly.

You'll note the name of the phone ... "Bob Tabor's Phone" ... I changed that in Windows Explorer by right-clicking the phone and selecting "Rename". This will allow me to test on multiple phones and tell them apart.

Now that I have the phone registered, I should be able to deploy the app to the Device.



But alas ...



... I must unlock the Device's lock screen and try again.

But after I overcome all the obstacles, I can finally see my PetSounds app running on my phone ...



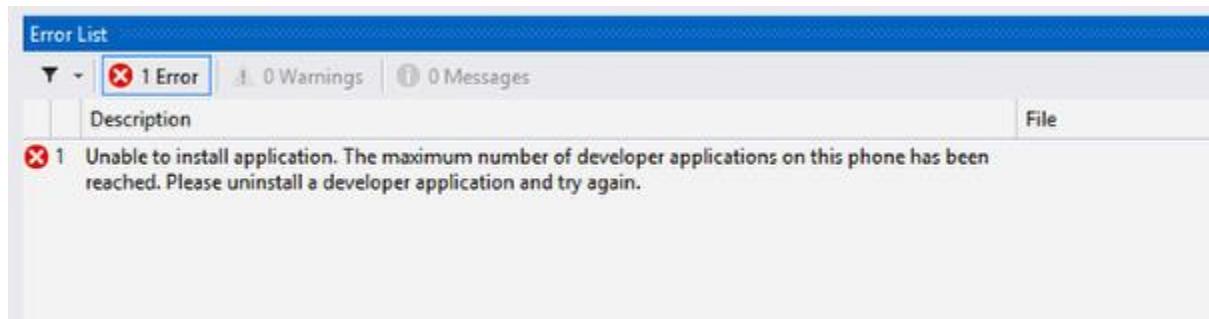
... and not only can I see it, but it actually works! I can annoy my kids and pets with a duck quack!

To exit out of the debug session, use the Stop button on Visual Studio's toolbar just like you would if you were running in the Emulator.



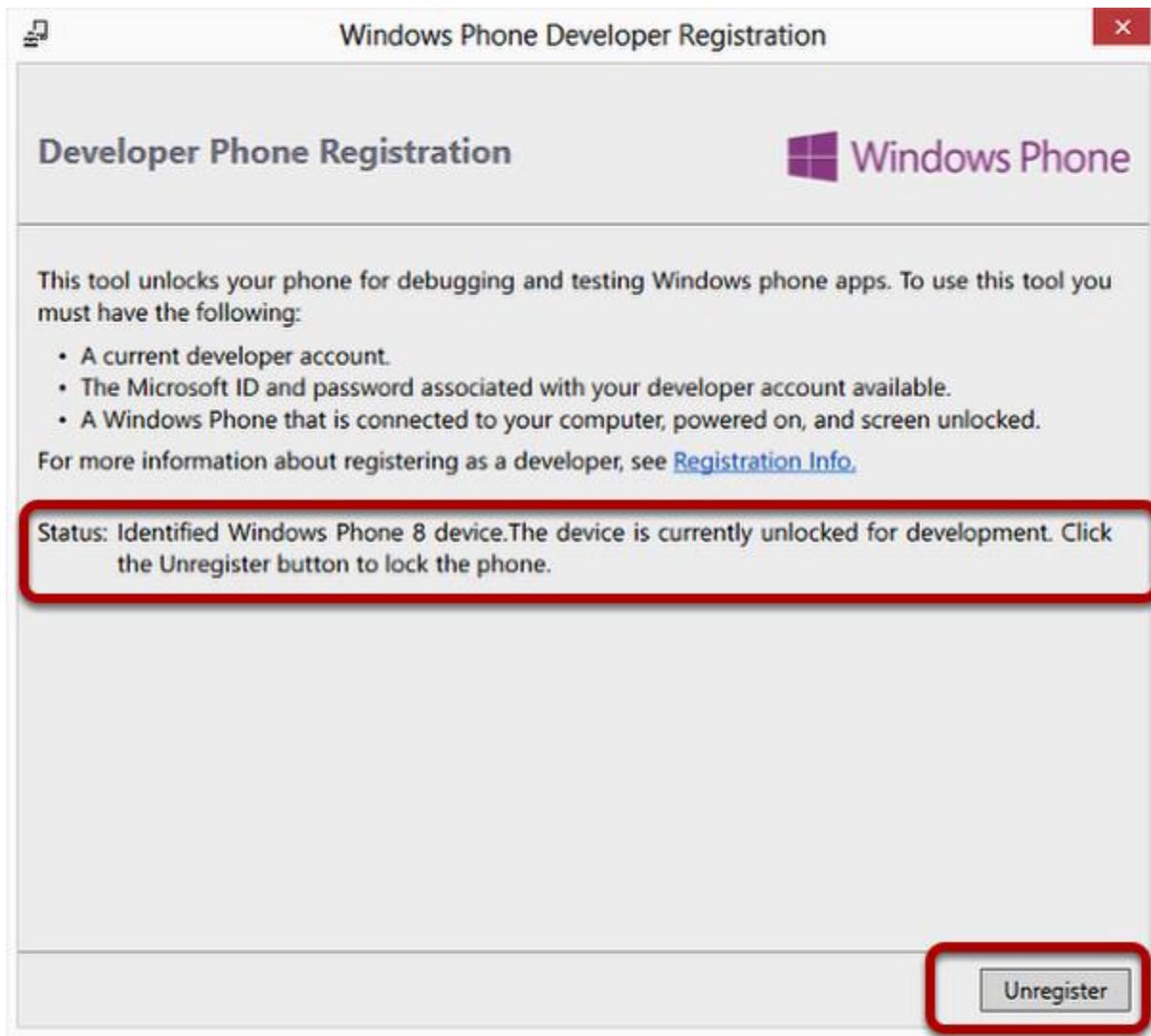
Even though I stop debugging the app on my Device, the app is still present there and I can still run it even if I unplug the Device from the computer. Each time I debug, it will deploy the latest version of my app to the physical device just like it did in the Emulator.

Before we move on, you may see a message like this in the future:



Apparently, there's a limit to the number of apps you can deploy to the phone. I think it is limited to 10. If that happens, just uninstall one or more developer apps the way you would uninstall a regular app. I.e., hold down on the tile and select Uninstall.

And suppose that you want to unregister your phone for development for some reason. You can re-run the Windows Phone Developer Registration Tool and it will identify the phone as being unlocked for development and provide an option to Unregister the phone:



#### 4. Obtaining a Windows Phone Dev Center membership

Finally, as I alluded to earlier, before you can register a phone for development and deploy your apps to it for testing, or later sell your apps in the Marketplace, you'll need a Windows Phone Dev Center membership. It costs \$99 per year if you purchase it directly from Microsoft.

However, there is a second option. Nokia has a Premium Developer Program for Lumia that not only gives you a Windows Phone Dev Center membership, but also gives you Telerik's Rad Controls for Windows Phone, a Buddy.com membership and two Nokia tech support tickets that you can use, presumably, if you run into challenges while developing or deploying your Phone apps. This is the deal I took advantage of and it worked swimmingly.

[http://www.developer.nokia.com/Developer\\_Programs/Lumia\\_developer\\_program.xhtml](http://www.developer.nokia.com/Developer_Programs/Lumia_developer_program.xhtml)

Windows Phone Controls × Nokia Developer - Nokia F × LearnVisualStudio.NET - C × How to register you

← → C www.developer.nokia.com/Developer\_Programs/Lumia\_developer\_program.xhtml

PrintWhatYouLike BibleGateway.com: ... Read Later ChangeDetection - ... Login LVS Login TS h

## NOKIA Developer

Design Develop Distribute Devices Resources Community

# Nokia Premium Developer Program for L

Improve your Windows Phone development productivity with the Nokia Premium Developer Program for Lumia. For \$99 (USD), get a package of benefits worth up to \$1500 (USD).

**Windows Phone Dev Center**  
One year of Windows Phone Developer Center membership. A \$99 (USD) retail value.

**Telerik RadControls**  
A free license for Telerik RadControls for Windows Phone. A \$99 (USD) retail value.

**Buddy.com Cloud APIs**  
Up to 12 months' worth of access to up to 1 million API calls per month with Buddy's cloud APIs during your membership. Up to a \$1200 (USD) retail value.

**Tech Support**  
2 Nokia Tech Support tickets. A \$198 (USD) value.



Join today and get a jump-start on your development

Already a member? Click here for your benefits

FAQ

## Recap

To recap, the big take away in this app was the composition of a deployment package, the purpose of the WMAppManifest.xml file, and deploying to a physical phone device for debugging from Visual Studio. We talked about registering your physical device to unlock it to deploy developer apps to it, and how to obtain a Windows Phone Dev Center account.

# Part 9: Overview of the Windows Phone 8 Emulator

**Source Code:** <http://aka.ms/absbeginnerdevwp8>

We've seen the Windows Phone Emulator at work in this series. It is a crucial component to developing apps for the Windows Phone platform, so I wanted to spend a little time becoming more familiar with it and point you in the right direction for more information.

Our game plan in this lesson ...

1. We'll learn about what the Windows Phone Emulator really is, and how it supplies different versions for different deployment scenarios.
2. We'll learn about the function of the Emulator, including keyboard shortcuts that emulate device buttons.
3. We'll learn about the controls to resize, rotate and simulate the act of handling the virtual device as if it were a physical one, with accelerometer support, GPS support, and more.

## 1. What is the Windows Phone Emulator?

In a nutshell, the Windows Phone Emulator is a desktop application that simulates a Windows Phone device, and actually provides similar performance to a physical Windows Phone device. It provides a virtualized environment in which you can debug and test Windows Phone apps without a physical device—in fact like I said at the outset when installing the Windows Phone 8 API, it's running Microsoft's Hyper-V. For a friendly introduction to Hyper-V on Windows 8, check out this blog post from the Windows 8 team:

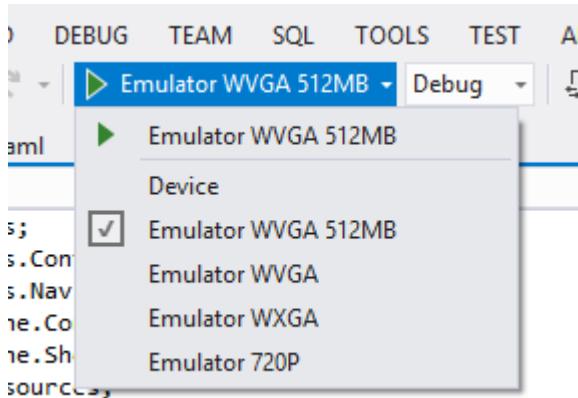
### **Bringing Hyper-V to "Windows 8"**

<http://blogs.msdn.com/b/b8/archive/2011/09/07/bringing-hyper-v-to-windows-8.aspx>

Now, while the Emulator is great for development and quick debugging sessions, before you publish your app to the Windows Phone Store, Microsoft recommends that you actually test your app on a real Windows Phone.

## 2. Choosing different versions of the Emulator for debugging

Up to now, when we clicked the Run / Debug button on Visual Studio's toolbar, we were running the Emulator in its default configuration ... something called WVGA 512MB:



What does WVGA and 512MB really mean?

The 512MB indicates that we're running in a memory constrained environment ... The default emulator image in Visual Studio is Emulator WVGA 512MB, which emulates a memory-constrained Windows Phone 8 phone. So, for example, the Lumia 610 is an inexpensive entry level Windows Phone 8 device which sports only 256MB of RAM. By contrast, the Lumia 920 has 1 GB of RAM. On lower-RAM devices, having multiple apps running at the same time or creating a memory intensive app might cause performance problems. So, to be sure that your app will run well on lower-RAM devices you can test your app using a realistic Emulator image.

There are a number of great articles about RAM usage on MSDN ... let me point out the best starting point to learn more:

#### **App performance considerations for Windows Phone**

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff967560\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff967560(v=vs.105).aspx)

#### **Optimizing Apps for Lower Cost Devices**

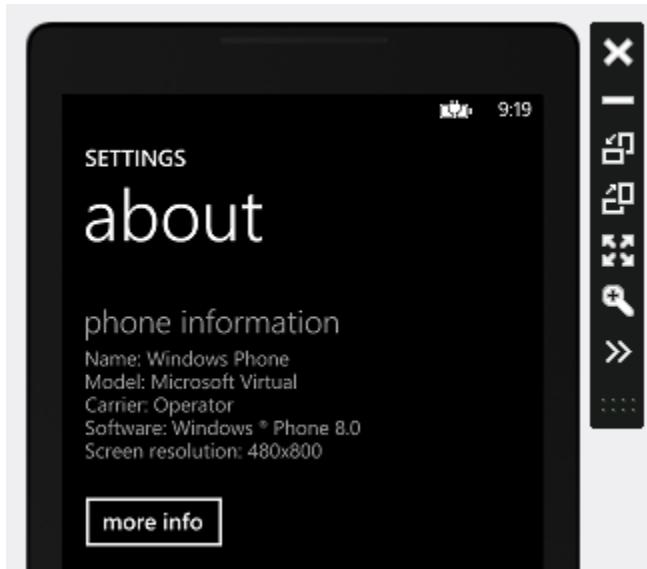
[http://blogs.windows.com/windows\\_phone/b/wpdev/archive/2012/03/07/optimizing-apps-for-lower-cost-devices.aspx](http://blogs.windows.com/windows_phone/b/wpdev/archive/2012/03/07/optimizing-apps-for-lower-cost-devices.aspx)

What does "WVGA" as well as those other acronyms stand for?

The Emulator allows you to test your app on a unique emulator image for each of the screen resolutions supported by Windows Phone. This default selection encourages you to target the largest possible market for your Windows Phone 8 app.

- WVGA (800 × 480)
- WXGA (1280 × 768)
- 720p (1280 × 720)

If you run the default, then go to the Settings app in the About button, you can see this confirmed:



How does this translate into the actual phones on the market?

Lumia 920

Display size: 4.5 inch

Display resolution: WXGA (1280 x 768)

Lumia 820

Display size: 4.3 inch

Display resolution: WVGA (800 x 480)

Lumia 610

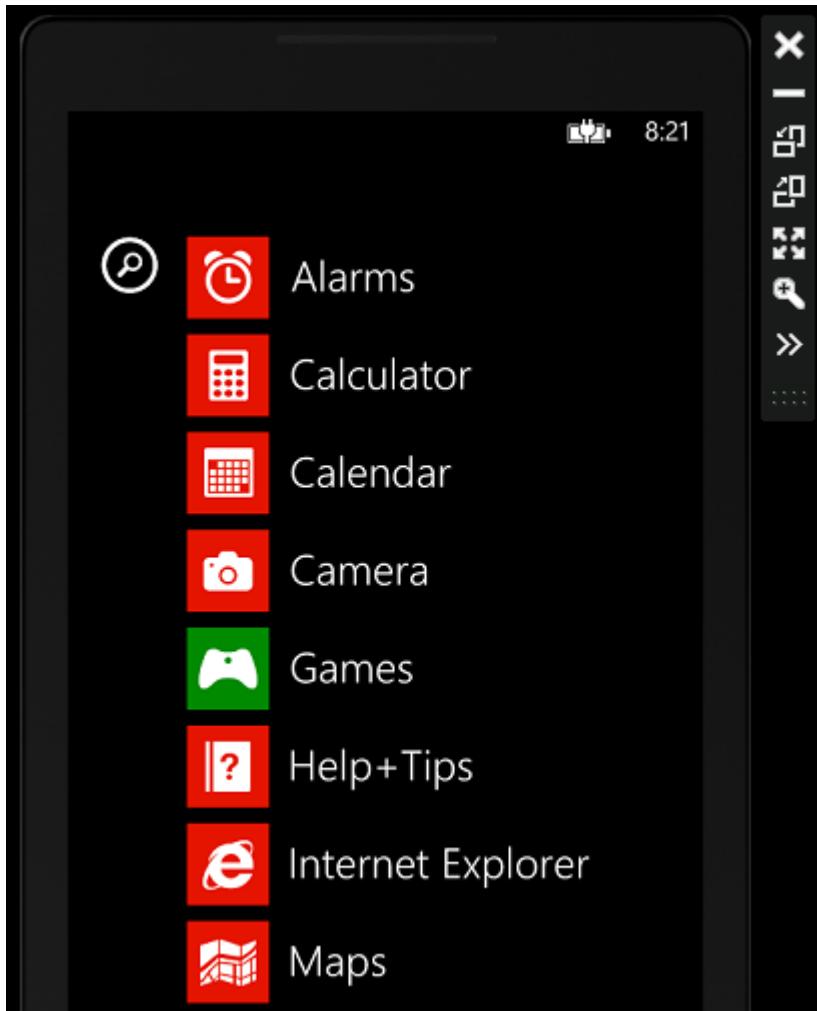
Display size: 3.7 "

Display resolution - WVGA (800 x 480)

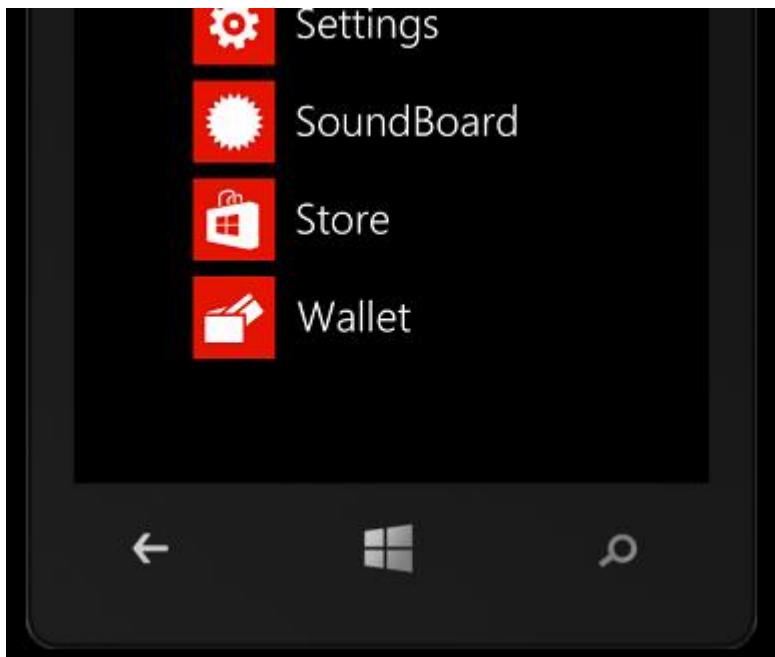
I realize by the time you watch this there may be newer phone models. The point is that you should be aware of the fact that you'll need to support different screen resolutions and memory constraints. Like I talked about in the lesson on Layout using XAML, you will want to eliminate specific pixel sizes for layout (except for margins). Choosing from the different Emulator sizes, you can make sure you're on the right track.

### 3. Working with the Phone Emulator's special features

I'll not spend a lot of time on the phone's basic navigation. A lot of this you'll pick up by using the Emulator throughout the series if you don't already have a physical phone in your possession. Essentially, you'll have an almost identical experience, complete with the Start and Apps page, search, web browsing, clock, battery, etc.:



You have the same hardware buttons on the bottom of the Emulator as you would have on a Windows Phone:



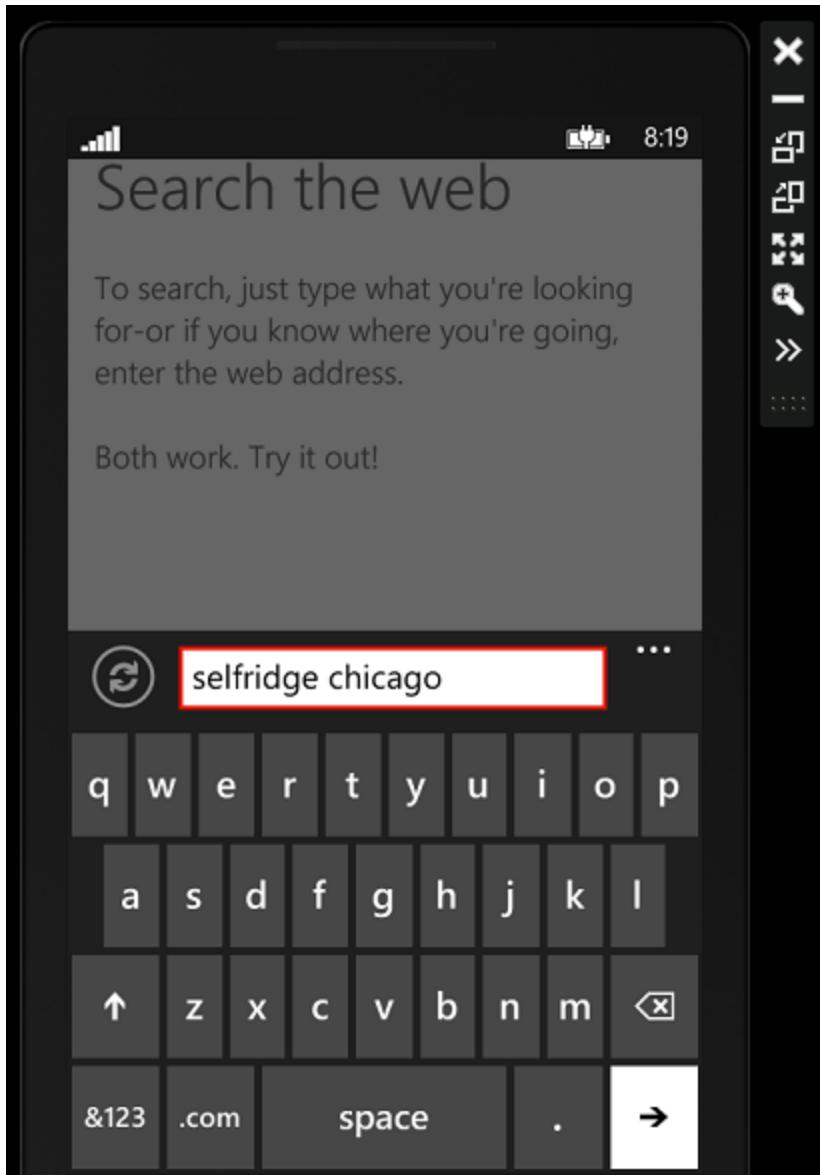
Note that you'll be missing the buttons on the side of the phone. For example, my Lumia 920 has three buttons on the side—a volume up and down, a power button, and a camera button. These can be accessed in the Emulator with keyboard function keys.

There's a list of keyboard mappings here:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff754352\(v=vs.105\).aspx#BKMK\\_KeyboardMapping](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff754352(v=vs.105).aspx#BKMK_KeyboardMapping)

- F6 - Camera button half-way
- F7 - Camera button
- F9 - F10 raise and lower the volume
- F11 - plays / pauses audio ... it simulates an in-line headphone button that pauses music and answers incoming phone calls. If you double-tap the button, it'll skip to the next audio track on your playlist or album
- F12 - Power button / lock screen
- F1 - Back button
- F2 - Windows key
- F3 - Search button

Let's work with the Emulator's keyboard ... F3 to Search:



You'll see the phone's keyboard appear. I can use my mouse to simulate tapping the keys. Most of the time during development, that's a pain. I would rather use my computer's keyboard.

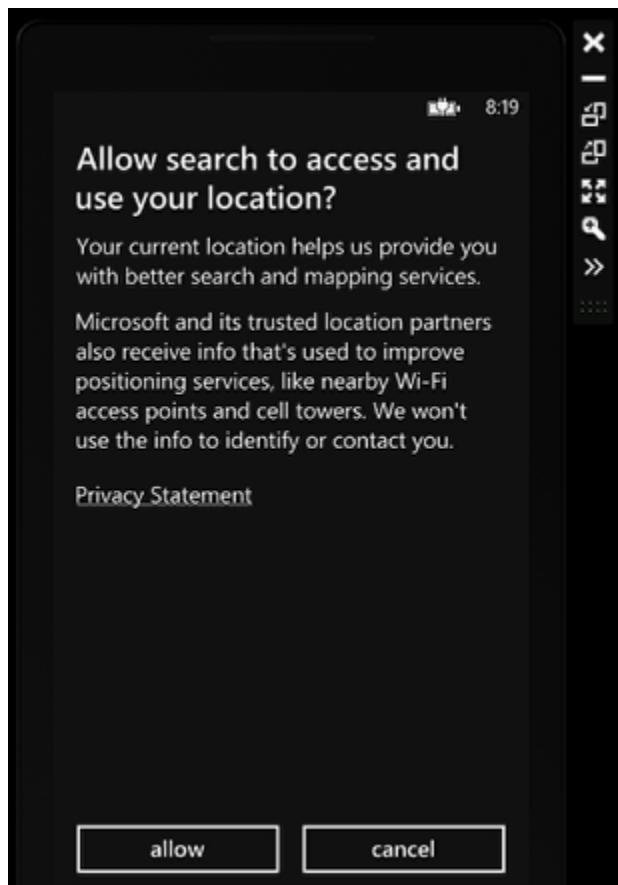
PAGE DOWN button - When a textbox is the target input, PAGE DOWN disables the virtualized "hardware" keyboard down, and you can use your physical keyboard for input.

PAGE UP button - When a textbox is the target input, PAGE UP enables the virtualized "hardware" keyboard.

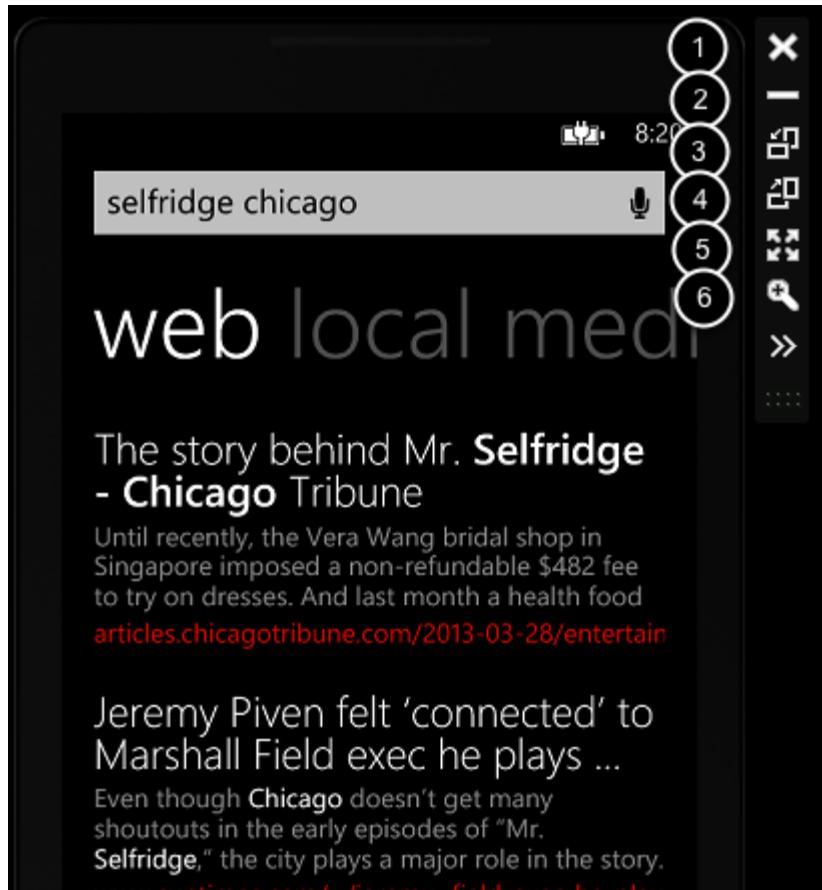
PAUSE/BREAK button - Toggles the keyboard, so you could just use that all the time.

As you can see, I typed in the search phrase "selfridge chicago", looking for articles or a Wikipedia article on one of Chicago's most famous sons, Harry Selfridge who founded the retail store Selfridges of London, and is the subject of a popular TV show on the BBC.

When I click the "Go" keyboard:



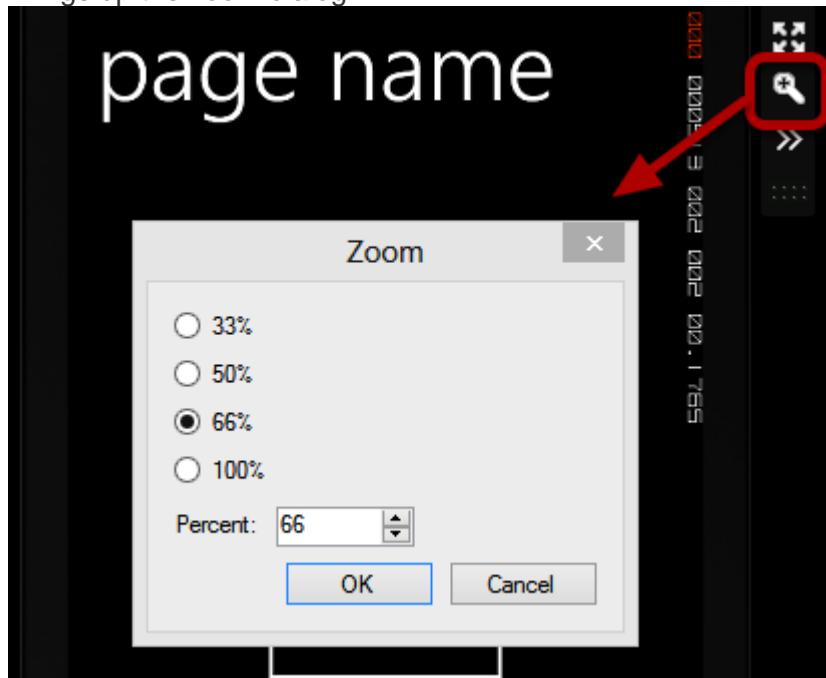
The Bing search works in the Emulator just as it would on a physical device, asking for permission to use your location in case it influences the search results. For example, if I were in London, Bing might show me a map of Selfridges near me. However, since I'm in the USA Bing will deliver different results. I'll show you how the Emulator determines its location in just a moment:



In addition to the "virtualized" phone on screen, there's a floating menu off to the right. A little experimentation will reveal the functions of the first six buttons, in order:

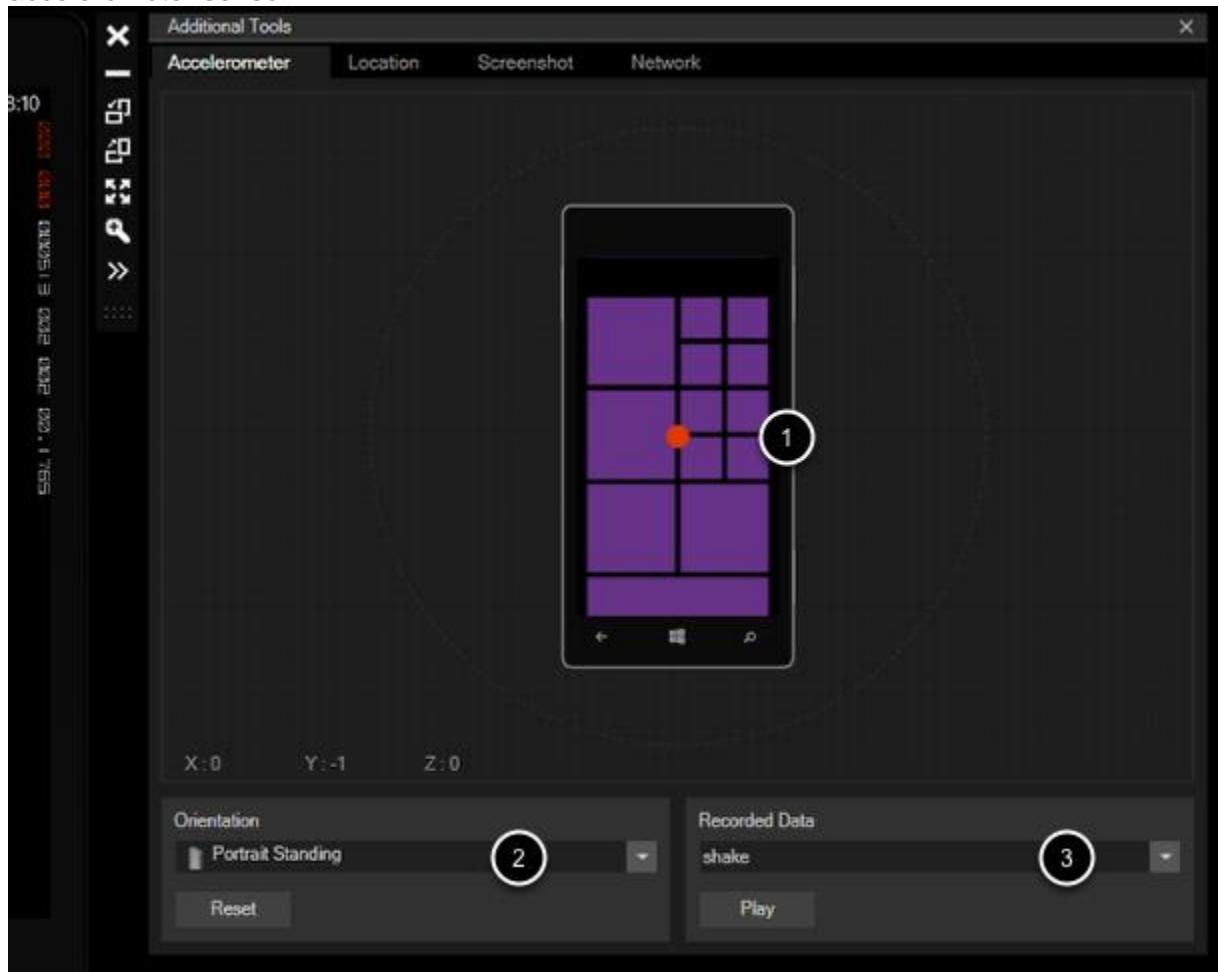
1. Shuts down the emulator
2. Minimizes the emulator
3. Rotates the emulator 90 degrees counter clockwise
4. Rotates the emulator 90 degrees clockwise
5. Expands the emulator to the largest size that can fit comfortably on your computer screen

6. Brings up the zoom dialog



7. Opens up the additional tools dialog. There are four tabs on the Additional Tools dialog. The first is the Accelerometer which can be used to test apps that utilize the

accelerometer sensor:



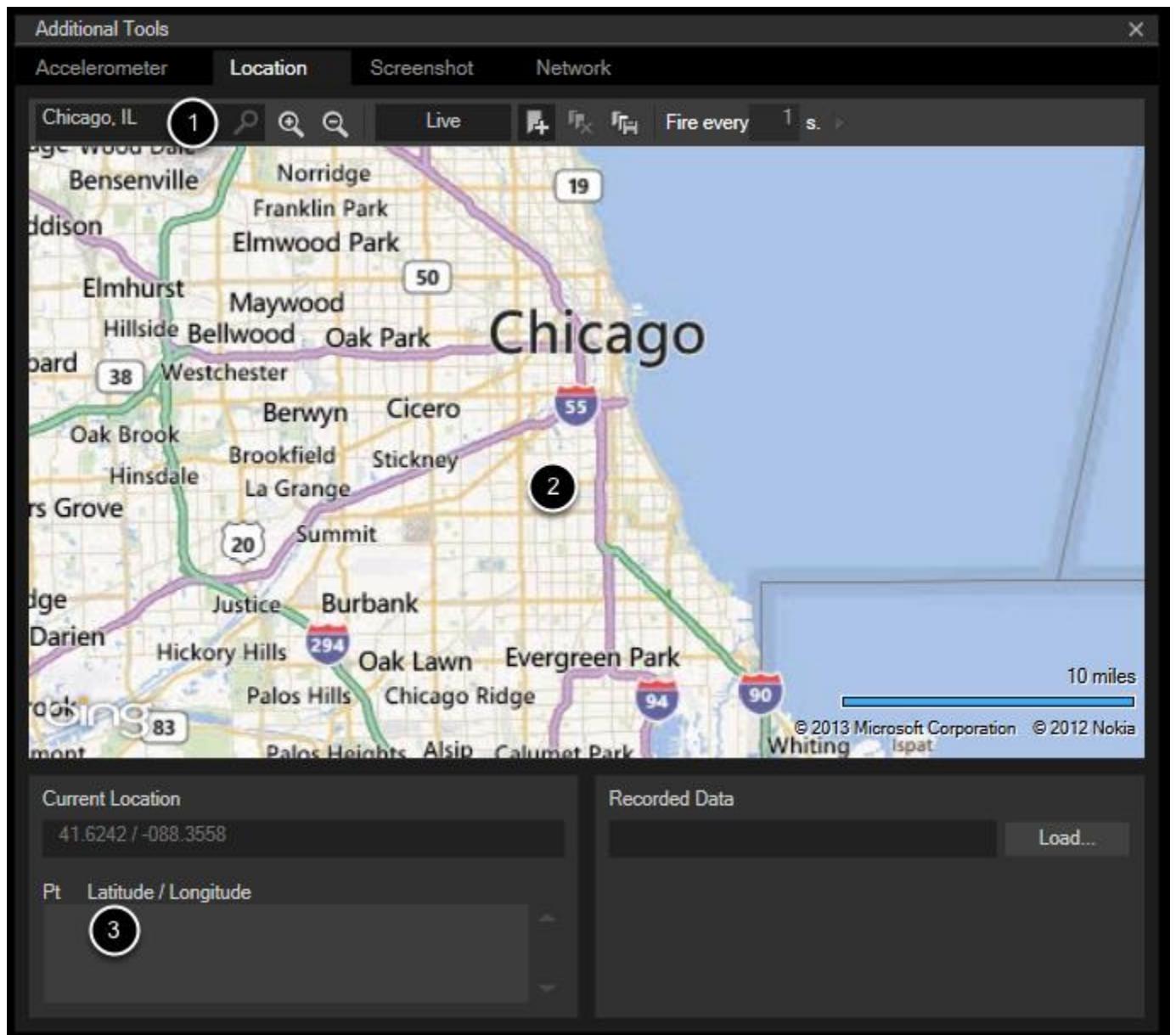
1. You can change the center point of the phone to change its position in 3D space by clicking and dragging the orange dot in the circle.
2. You can change the Orientation to one of the presets.
3. You can play recorded data, like a "shake" motion.

For more information about the Emulator and the Accelerometer sensor in the Windows Phone SDK, a good starting spot would be this article:

How to test apps that use the accelerometer for Windows Phone

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202936\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202936(v=vs.105).aspx)

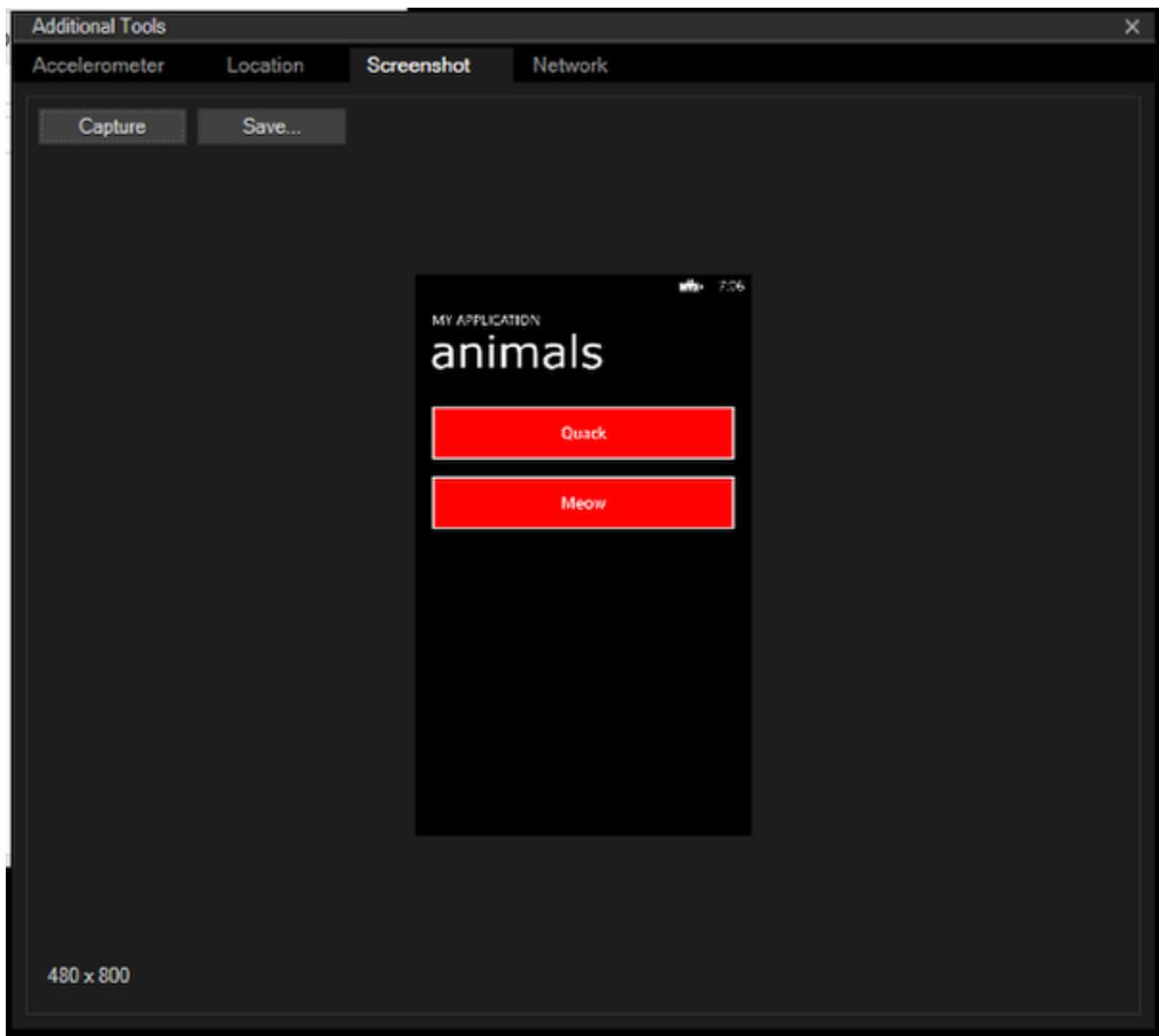
The next tab is the Location tab which allows you to set the current location of the phone. So, even though I'm sitting in Dallas, Texas, I can act like I'm testing my phone in Chicago or any other place in the world. To do this:



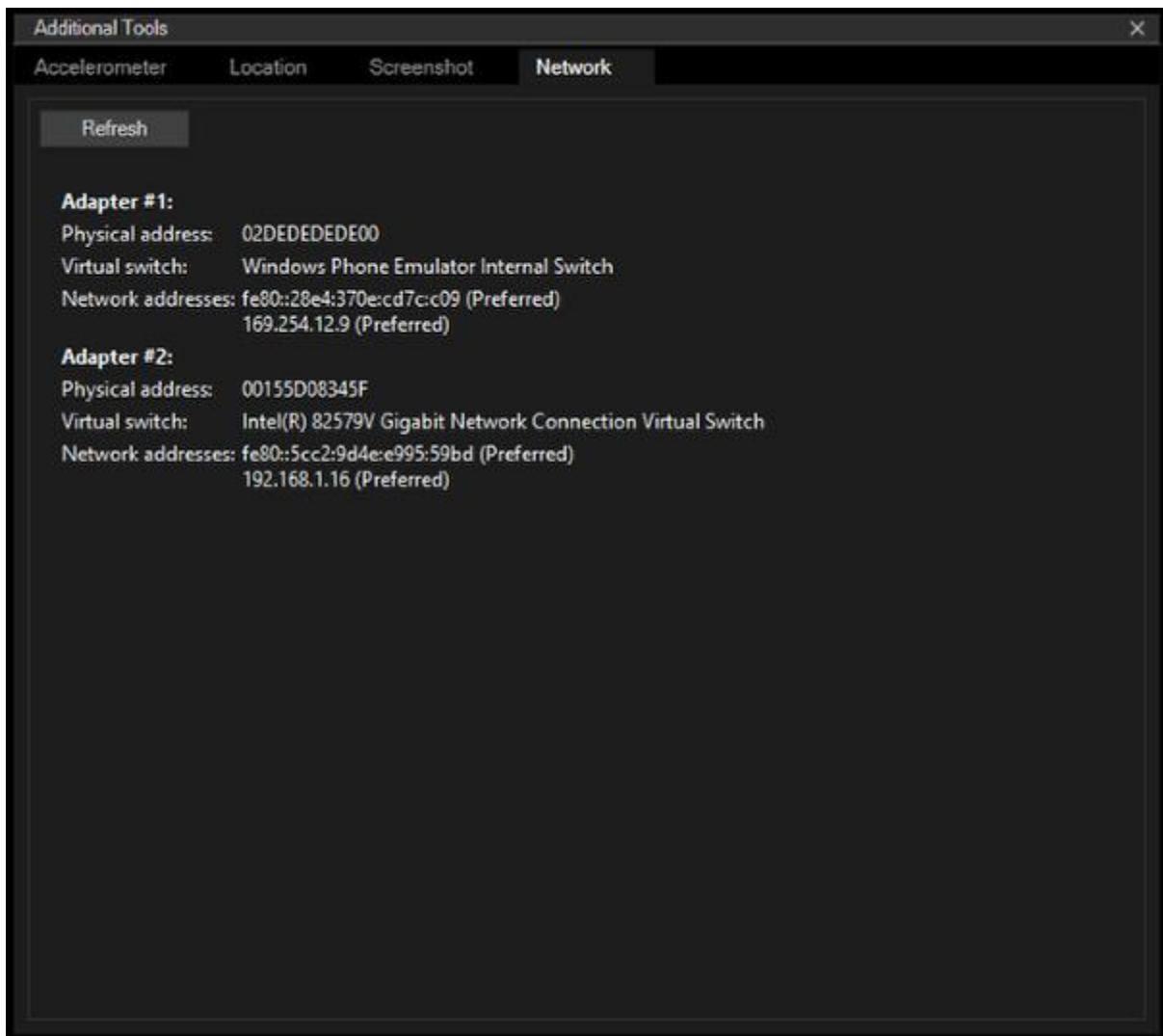
1. Perform a search for a particular location, like "Chicago, IL".
2. Right-click the map to create a pin.
3. I'll verify the exact location in the list of pins marked by their latitude and longitude.

Now, I can go to the Windows Phone Emulator's Map app to see the map zoom to that location. We'll rely on this technique later in this series when we create our AroundMe app.

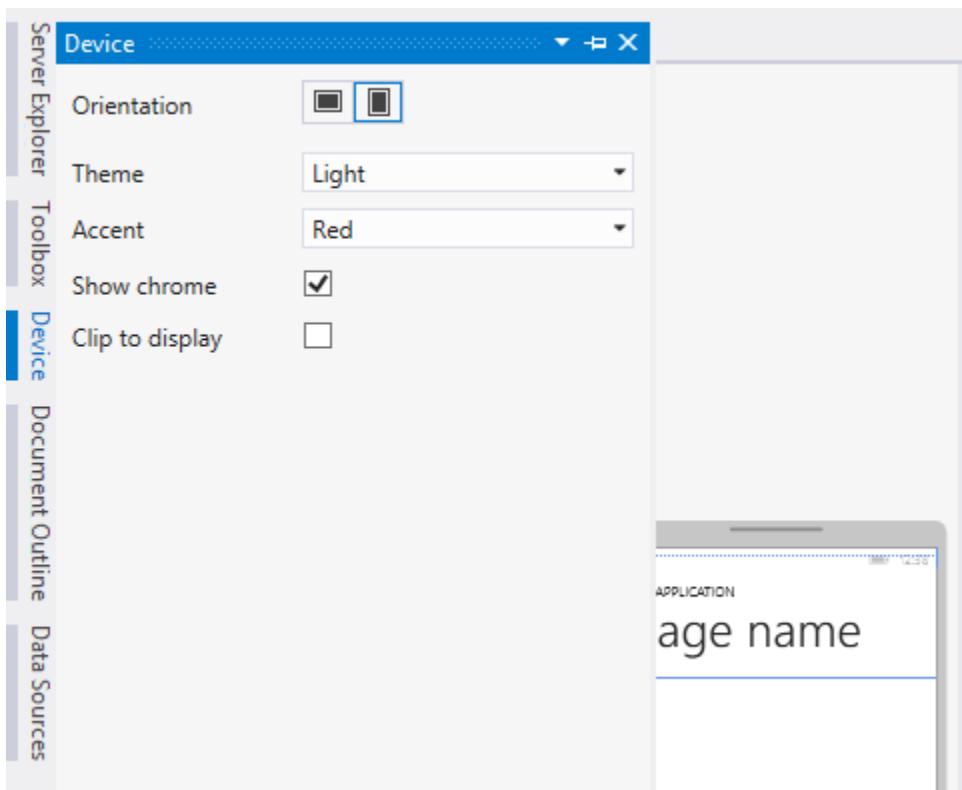
The third tab is for creating Screenshots, which can be helpful when you're creating documentation or bug reports:



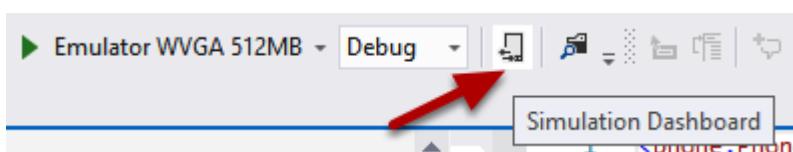
And the final tab is the Network tab. There's not a lot you can do other than see the IP address of the phone on the network:



In addition to the Emulator itself, Visual Studio has some tooling that can affect how the XAML designer displays the phone (orientation, theme and color, chrome, etc.):



There's also a Simulation Dashboard that allows you to validate your Windows Phone apps in various real life conditions.



The Simulation Dashboard gives you the ability to simulate different network conditions, trigger reminders, and check how your app will perform under a locked screen:

Simulation Dashboard

## Control Settings

Use the following settings to test your app in different scenarios that simulate real-world conditions.

Enable Network Simulation

Network Speed

2G      3G      4G      Wi-Fi      No Network

Signal Strength

Good      Average      Poor

Lock Screen

Locked  
 Unlocked

Reminders

You have so many ways to test and monitor your app during development time so you gain some confidence in how it will perform when you distribute it on the Store.

### Recap

The last tip I would give about the emulator is that it's ok to just leave it up and running during development. There's no need to shut it down. When you launch and re-launch your app in debug mode from Visual Studio, part of that process is deploying the latest version of the app to the phone. It's time consuming to relaunch the Emulator each time, and Visual Studio will connect to an existing Emulator instance when it needs to.

So to recap, the Emulator is a great tool that allows you to quickly deploy your app during development to a virtualized Windows Phone. We looked at how to test different device screen sizes and memory constraints, how to manipulate the emulator to test for rotation and motion, how to make the phone think it is located geographically in a specific place, and much more. We'll use these skills extensively throughout this series.

# Part 10: Overview of the Databound App and Pivot App Project Templates

Source Code: <http://aka.ms/absbeginnerdevwp8>

We spent the first 9 lessons learning the absolute basics and we were able to build a very simple app. Yes, PetSounds app is a nice start, but it's a bit limiting. Currently, there's only one category of sounds—animal sounds—and we have two buttons and so, two sounds.

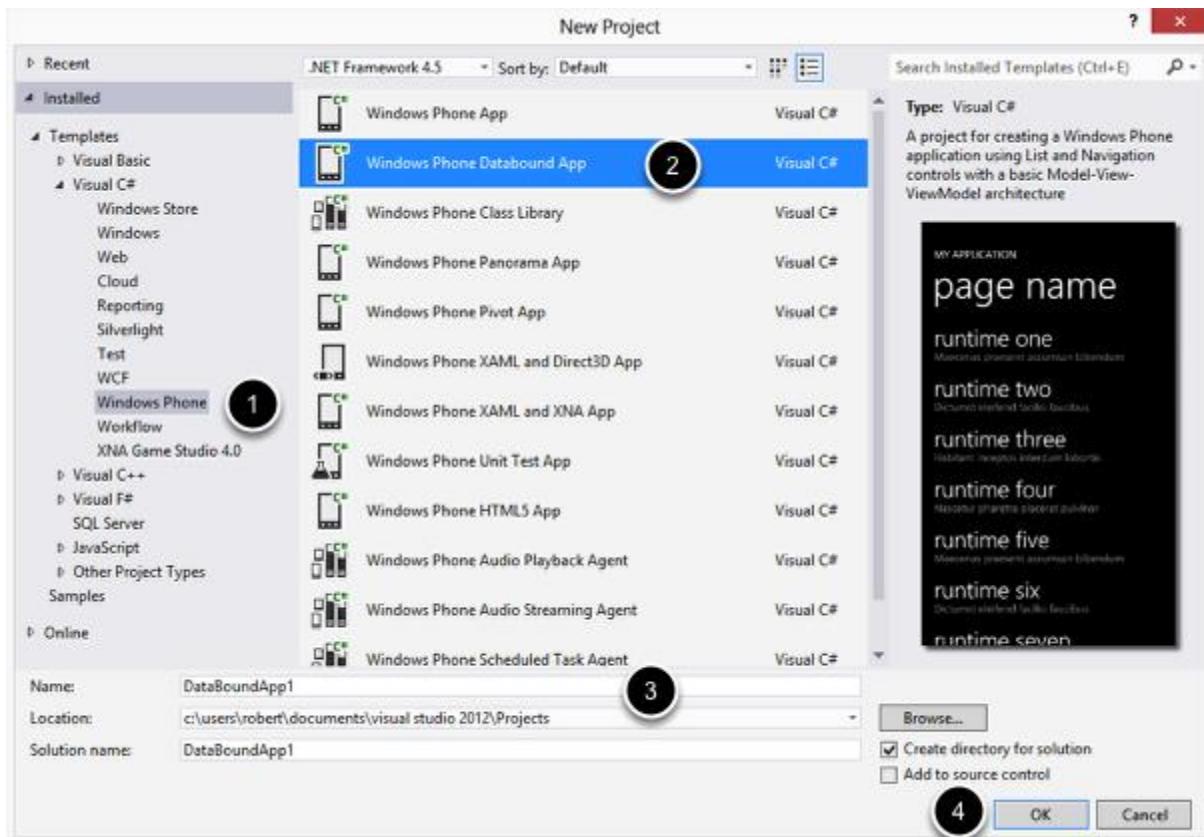
I want us to turn this into a more full-fledged sound board app with different categories of sounds ... perhaps even custom sounds that we can record. We need a good way to represent CATEGORIES of sounds in the app, and I'm willing to bet that there's at least one template available in Visual Studio that will give us a good starting point to help get us pretty close to what I have in mind.

So, in this lesson I want to review two project templates to learn a bit more about what they can do and in so doing, determine if there's a fit between their built-in capabilities and my needs for a new SoundBoard app.

Here's the game plan for this lesson:

1. Create a sample Windows Phone Databound App project template to discover its built-in functionality and look at the code to see how it accomplished those features.
2. Repeat the process for the Windows Phone Pivot App project template.

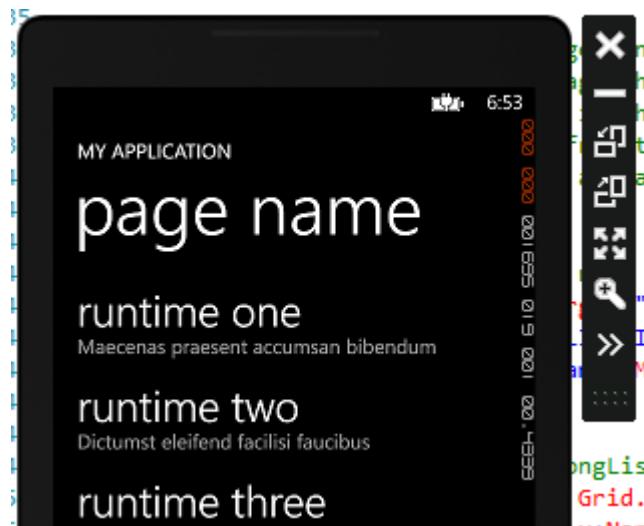
1. Understanding the Windows Phone Databound App project template's functionality  
In Visual Studio, File menu, New | Project ... opens the New Project dialog:



1. Make sure you're in the Installed | Templates | Visual C# | Windows Phone section.
2. Choose the Windows Phone Databound App project template.
3. You can leave the name as is ... we'll probably just delete this project at some point.
4. Click OK.

Once the project is created, before you do anything else, start debugging (F5). This will allow us to observe the functionality "out of the box".

When the app runs, you see the main page containing a list of items named "runtime one", "runtime two", etc. Each of these has a sub-title with "lorem ipsum" text:



Clicking on one of the items will reveal a second page, a details page, containing the details of the item you clicked on. Here you see the full "lorem ipsum" associated with the selected item:



Stop running the app and navigate to the MainPage.xaml. We want to understand how this app works and determine if we can utilize this in our upcoming SoundBoard app.

The list of items is made possible by a control called the LongListSelector between lines 51 and 71:

```

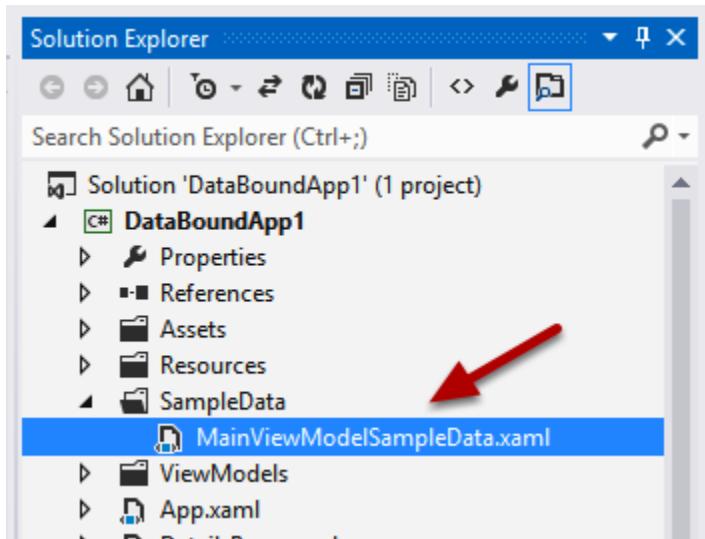
40
49      <!--ContentPanel contains LongListSelector and LongListSelector ItemTemplate. Place a
50      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
51          <phone:LongListSelector
52              x:Name="MainLongListSelector"
53              Margin="0,0,-12,0"
54              ItemsSource="{Binding Items}"
55              SelectionChanged="MainLongListSelector_SelectionChanged">
56
57          <phone:LongListSelector.ItemTemplate>
58              <DataTemplate>
59                  <StackPanel Margin="0,0,0,17">
60                      <TextBlock Text="{Binding LineOne}"
61                          TextWrapping="Wrap"
62                          Style="{StaticResource PhoneTextExtraLargeStyle}"/>
63                      <TextBlock Text="{Binding LineTwo}"
64                          TextWrapping="Wrap"
65                          Margin="12,-6,12,0"
66                          Style="{StaticResource PhoneTextSubtleStyle}"/>
67                  </StackPanel>
68              </DataTemplate>
69          </phone:LongListSelector.ItemTemplate>
70
71      </phone:LongListSelector>
72  </Grid>
73

```

Notice that it has an `ItemsSource` property with a binding expression (line 54). I briefly talked about this type of binding expression in a previous lesson. We use this type of expression to databind a list of data to a visual control. Each item in a generic list, say for example, a `List<T>`, would be displayed using an `ItemTemplate`. You can see the item template defined for the `LongListSelector` between lines 57 and 69. An `ItemTemplate` property is of type `DataTemplate`, which is simply a data type that defines the visual structure of a data object.

Inside the `DataTemplate`, we define the visual structure of each instance of data in our collection ... a `StackPanel` containing two `TextBlocks`.

Inside each of the `TextBlocks` there's a `Text="{Binding LineOne}"` and `Text="{Binding LineTwo}"` attribute value set (lines 60 and 63, respectively). That's what binds a property of a given object to an attribute of a control. We'll see the class hierarchy for the sample data in just a bit. First, let's look at where the data is actually coming from ... open the `SampleData` folder to reveal the `MainViewModelSampleData.xaml` file:



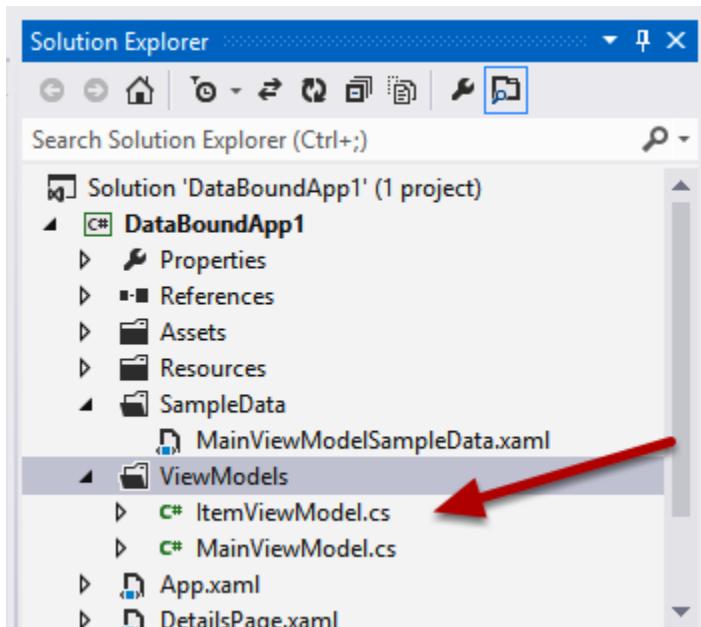
If you open that file in the Solution Explorer, you'll see the XML containing the sample data used in the app. Notice the attributes of each `ItemViewModel` element—`LineOne` and `LineTwo`. They match the names of the instance properties we bind the `Text` attribute to in each of the `TextBoxes` in our `DataTemplate`:

```
1 <vm:MainViewModel
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:vm="clr-namespace:DataBoundApp1.ViewModels"
5   SampleProperty="Sample Text Property Value">
6
7   <vm:MainViewModel.Items>
8     <vm:ItemViewModel ID="0" LineOne="design one" LineTwo="Maecenas praesent
9     <vm:ItemViewModel ID="1" LineOne="design two" LineTwo="Dictumst eleifend
10    <vm:ItemViewModel ID="2" LineOne="design three" LineTwo="Habitant incep
11    <vm:ItemViewModel ID="3" LineOne="design four" LineTwo="Nascetur pharetra
12    <vm:ItemViewModel ID="4" LineOne="design five" LineTwo="Sagittis senectus
13    <vm:ItemViewModel ID="5" LineOne="design six" LineTwo="Torquent ultrices
14  </vm:MainViewModel.Items>
15
16 </vm:MainViewModel>
```

MainViewModelSampleData.xaml cannot be edited in the Design view.

Now, let's take a look at the classes that model this data in C#. In the ViewModels folder, there are two files:

- ItemViewModel.cs
- MainViewModel.cs



The ItemViewModel.cs contains the class definition for the objects we're binding to. Here again we see the LineOne and LineTwo public properties, as well as their private field definitions and other properties that are not utilized in this sample:

```

10
11  namespace DataBoundApp1.ViewModels
12  {
13      public class ItemViewModel : INotifyPropertyChanged
14      {
15          private string _id;
16          /// <summary> ...
17          public string ID...
18
19
20          private string _lineOne;
21          /// <summary> ...
22          public string LineOne...
23
24
25          private string _lineTwo;
26          /// <summary> ...
27          public string LineTwo...
28
29
30          private string _lineThree;
31          /// <summary> ...
32          public string LineThree...
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
}

```

1

2

I want you to be aware of something you may have not seen before ... there's some extras added to this class definition that make it special.

1. The class implements the `INotifyPropertyChanged` interface.
2. As a result of that promise to implement this interface, there's a public event called `PropertyChanged` (and a private method called `NotifyPropertyChanged`) implemented.

The purpose of these additions to the class is to enable the notion of "change management". Whenever one of the properties of this class change, if you look at the "set" of each property (rolled up in the screen capture, above) it will call `NotifyPropertyChanged()` passing in its name. The `NotifyPropertyChanged` method will in turn call the `PropertyChanged` event. Any code that listens for that event will be notified when the `PropertyChanged()` event is triggered.

To further add another clue to this story, take a look at the definition for the `MainViewModel` class:

```
6  namespace DataBoundApp1.ViewModels
7  {
8      public class MainViewModel : INotifyPropertyChanged
9      {
10         public MainViewModel()
11         {
12             this.Items = new ObservableCollection<ItemViewModel>();
13         }
14
15         /// <summary> ...
16         public ObservableCollection<ItemViewModel> Items { get; private set; } 1
17
18         private string _sampleProperty = "Sample Runtime Property Value";
19         /// <summary> ...
20         public string SampleProperty... 3
21
22         private string _localizedSampleProperty...
23         public string LocalizedSampleProperty...
24
25         public bool IsDataLoaded...
26
27         /// <summary> ...
28         public void LoadData()...
29
30         public event PropertyChangedEventHandler PropertyChanged;
31         private void NotifyPropertyChanged(String propertyName)... 2
32
33     }
34 }
```

It, too ...

1. Implements INotifyPropertyChanged, and
2. the public PropertyChanged event  
... however, it also has ...
3. A public ObservableCollection<ItemViewModel> called Items.

First, recall that the LongListSelector had a ItemsSource attribute set to "{Binding Items}". Yes, that is the same "Items" here. That's what ties the collection of ItemViewModel object instances to the list.

The Items property is of type ObservableCollection<ItemViewModel> ... as an OBSERVABLE collection, it is aware when changes are raised from the instances it collects, and then can report those updates to whatever is bound to it. Hopefully you can see where we're headed with this.

The important question is: why would any other code want to be notified about changes going on inside of this collection?

As this example stands right now, there is absolutely no reason any other code would want to be notified because all of the sample data is "static", insomuch that it is being loaded from a static XML file and is not expected to change while the app is running.

However, WHAT IF we wanted to support a new feature in this app where these list items were being updated constantly from some outside source, say, a web service. The web service delivers new "lorem ipsum" text every 30 seconds to each of the ItemViewModel object instances? Silly as it might sound, if we added code that dynamically changed the data in each instance of ItemViewModel every 30 seconds, nothing else in our app would need to change. Each property that was updated would say "Hey, my value changed!" and the entire object instance would say "Hey, I changed!". The ObservableCollection would report this to the LongListSelector, and it would be updated on screen magically.

So, all of this extra code ... implementing the INotifyPropertyChanged interface, the PropertyChanged event, all of the "set" code that calls NotifyPropertyChanged() and so on ... it's all to enable a feature called "observability", and enables an important software development pattern called Model-View-ViewModel ... it enables controls like the LongListSelector and others to automatically update what is displayed when the underlying data has been updated.

Our SoundBoard app doesn't require "observability", so I'm not going to implement all this extra code. However, if I had a good candidate for this style of application—where my data could change often—I would definitely take the approach that has been templated in this project.

The good news is that there's a nice template here for you—you will have to change the class and property names, change the manner in which the data is loaded, etc., but the pattern is nicely implemented here and can be used as a template.

The App.xaml.cs file kicks off the process of loading the data from the XML file into instances of the data model. In the constructor, a new instance of the MainViewModel is created and becomes available to the entire app as a property of the App class called ViewModel:

```

11
12  namespace DataBoundApp1
13  {
14      public partial class App : Application
15      {
16          private static MainViewModel viewModel = null;
17
18          /// <summary>
19          /// A static ViewModel used by the views to bind against.
20          /// </summary>
21          /// <returns>The MainViewModel object.</returns>
22          public static MainViewModel ViewModel
23          {
24              get
25              {
26                  // Delay creation of the view model until necessary
27                  if (viewModel == null)
28                      viewModel = new MainViewModel(); ←
29
30                  return viewModel;
31              }
32          }
33      }

```

Later in the App.xaml.cs, in the Application\_Activated event (see the code comments for when this event is triggered) if the App.ViewModel's IsDataLoaded is false, then it will call the LoadData() method of the MainViewModel class. You could modify this to retrieve data from a web service, a local database or a different XML file:

```

83
84  // Code to execute when the application is activated (brought to foreground)
85  // This code will not execute when the application is first launched
86  private void Application_Activated(object sender, ActivatedEventArgs e)
87  {
88      // Ensure that application state is restored appropriately
89      if (!App.ViewModel.IsDataLoaded)
90      {
91          App.ViewModel.LoadData(); ←
92      }
93  }

```

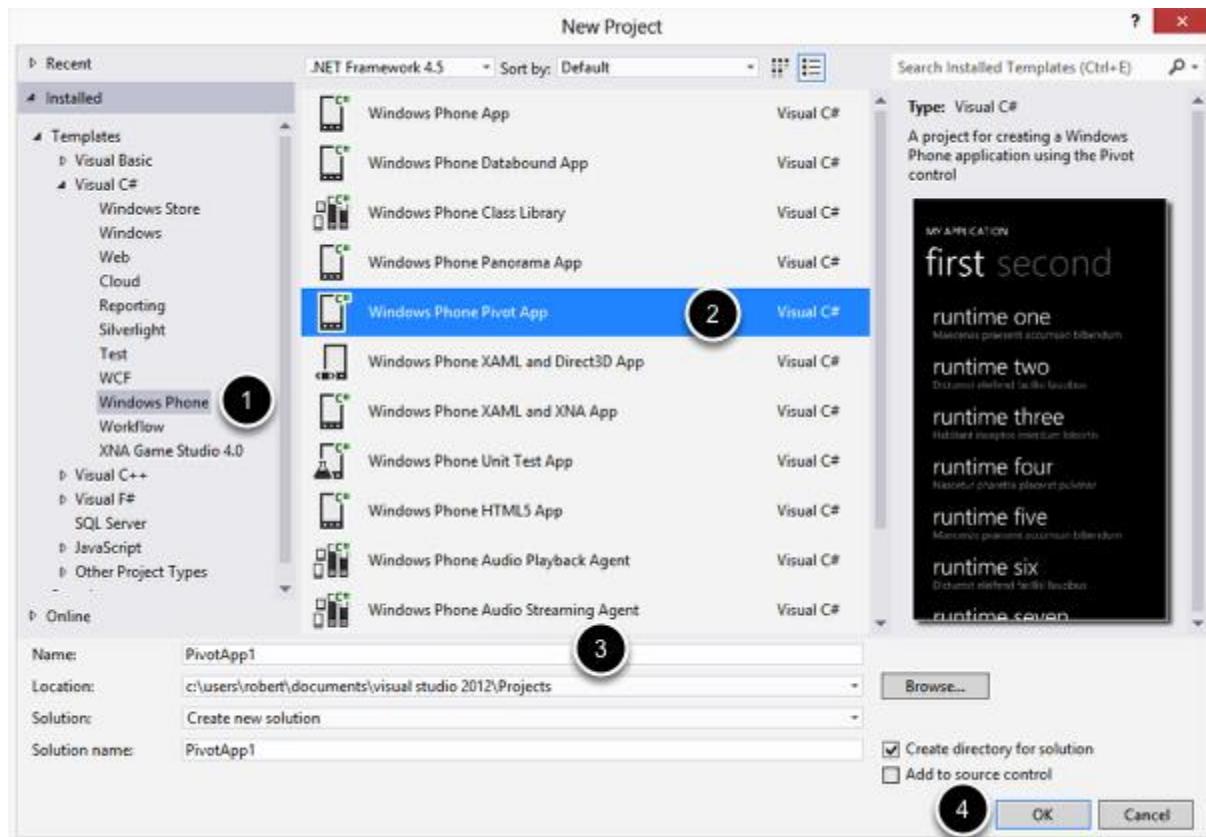
The same check is made on the MainPage.xaml.cs file's OnNavigatedTo() method. If the data has not been loaded, load it now. Both the previous call to LoadData() and this call exist because of how the Windows Phone OS navigates between apps with the back button (more on that later):

```
15  public partial class MainPage : PhoneApplicationPage
16  {
17      // Constructor
18      public MainPage()
19      {
20          InitializeComponent();
21
22          // Set the data context of the LongListSelector control to the sample data
23          DataContext = App.ViewModel;
24
25          // Sample code to localize the ApplicationBar
26          //BuildLocalizedApplicationBar();
27      }
28
29      // Load data for the ViewModel Items
30      protected override void OnNavigatedTo(NavigationEventArgs e)
31      {
32          if (!App.ViewModel.IsDataLoaded)
33          {
34              App.ViewModel.LoadData(); ←
35          }
36      }
37  }
```

Ok, so hopefully at a high level you can see how the Windows Phone Databound App template is wired up to enable data access through a pattern called MVVM, utilizing features in C# and the Windows Phone Runtime called "observability".

## 2. Understanding the Windows Phone Pivot App project template's functionality

While the Databinding project template does accommodate several features we'll want implemented in our SoundBoard app, we still need a way to navigate between categories of sounds. With that in mind, we'll examine the Windows Phone Pivot App template. We'll discard our work in the previous app and File | New | Project ... to create a new Windows Phone Pivot app:

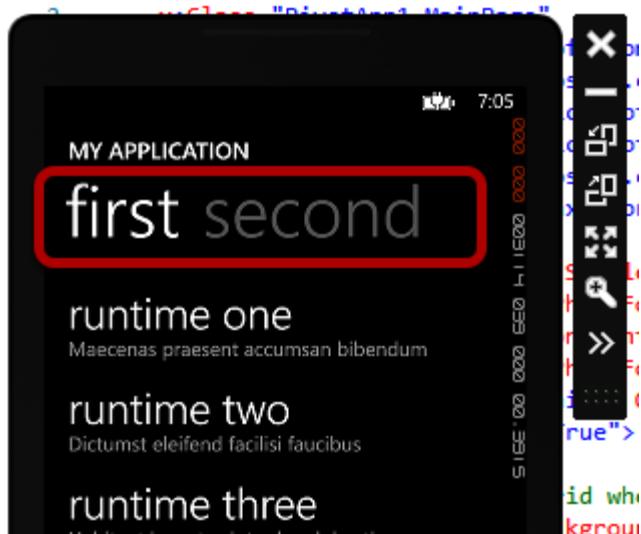


Just as earlier in this lesson:

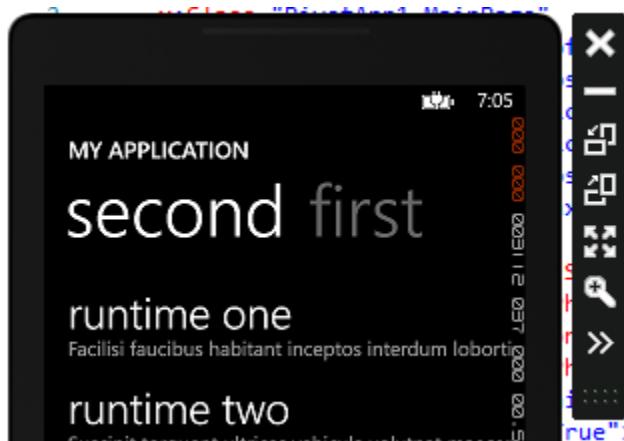
1. Make sure you're in the Installed | Templates | Visual C# | Windows Phone section.
2. This time, choose the Windows Phone Pivot App project template.
3. Again, you can leave the name as is ... we'll probably just delete this project at some point.
4. Click OK.

Again this time we want to immediately run the app to see what it can do without any modifications.

At first glance, the app looks identical, but notice the area below the app title:



You can swipe between views (PivotItems) by click dragging from first to second title:



Admittedly, we're working with sample data, and the LongListSelector is data bound to the same list of objects in both cases, however, the idea is that we can create a Pivot that has multiple PivotItem elements (what I called a view a moment ago), each containing a LongListSelector bound to different data:

```

39      <!--Pivot Control-->
40  <phone:Pivot Title="MY APPLICATION">
41      <!--Pivot item one-->
42      <phone:PivotItem Header="first">
43          <!--Double line list with text wrapping-->
44          <phone:LongListSelector Margin="0,0,-12,0"
45              ItemsSource="{Binding Items}">
46
47          <phone:LongListSelector.ItemTemplate>
48              <DataTemplate>
49                  <StackPanel Margin="0,0,0,17">
50                      <TextBlock Text="{Binding LineOne}"
51                          TextWrapping="Wrap"
52                          Style="{StaticResource PhoneTextExtraLargeStyle}"/>
53                      <TextBlock Text="{Binding LineTwo}"
54                          TextWrapping="Wrap"
55                          Margin="12,-6,12,0"
56                          Style="{StaticResource PhoneTextSubtleStyle}"/>
57                  </StackPanel>
58              </DataTemplate>
59          </phone:LongListSelector.ItemTemplate>
60
61      </phone:LongListSelector>
62  </phone:PivotItem>
63
64      <!--Pivot item two-->
65  <phone:PivotItem Header="second">
66      <!--Double line list no text wrapping-->
67      <phone:LongListSelector Margin="0,0,-12,0" ItemsSource="{Binding Items}">
68          <phone:LongListSelector.ItemTemplate>
69              <DataTemplate>
70                  <StackPanel Margin="0,0,0,17">
71                      <TextBlock Text="{Binding LineOne}" TextWrapping="NoWrap" Mi
72                      <TextBlock Text="{Binding LineThree}" TextWrapping="NoWrap"
73                  </StackPanel>
74              </DataTemplate>
75          </phone:LongListSelector.ItemTemplate>
76      </phone:LongListSelector>
77  </phone:PivotItem>
78
79  </phone:Pivot>

```

The diagram shows four numbered callouts pointing to specific parts of the XAML code:

- 1**: Points to the opening tag of the Pivot control: <phone:Pivot Title="MY APPLICATION">
- 2**: Points to the first PivotItem element: <phone:PivotItem Header="first">
- 3**: Points to the second PivotItem element: <phone:PivotItem Header="second">
- 4**: Points to the ItemTemplate of the first PivotItem, specifically the DataTemplate and its StackPanel content.

1. The Pivot is defined with two "pages", or rather, "PivotItems" ...
2. Here's the first PivotItem
3. Here's the second PivotItem
4. The content of each PivotItem mirrors the databinding example with the LongListSelector, data models, etc.

I can see it all coming together now. We could use the Windows Phone Pivot App template to create categories of sound icons. Each sound icon would be rendered based on its DataTemplate. We would create a data model that would have categories that contained collections of sounds, including information like the sound name and the path to the wav file to be played. So the good news is that we have a clear direction for what needs to happen next—only implementation details remain.

## Recap

To recap, in this lesson we learned about the features of the Windows Phone project templates—the Databound template and the Pivot template share almost identical features. They both databind a `LongListSelector` to a data model that is populated with data from an XML file.

Not only is the data databound to controls inside of a `DataTemplate`, but the templates provide a pattern for monitoring changes to the underlying data and automatically updating the user interface as those changes are made. We won't need that feature in our projects, but it's nice to know there's a simple pattern we can use in these project templates should we ever need it. Finally, we saw the use of the `Pivot` control to create pivot items in our app.

# Part 11: Setting up the SoundBoard App

Source Code: <http://aka.ms/absbeginnerdevwp8>

The PetSounds app was a nice way to learn about playing media, but we have commercial aspirations with our app. So, in this lesson we'll sit down and think about designing and developing a more robust version of PetSounds -- a sound board app with many different types of categories of sounds, and perhaps even the ability to record your own sounds.

So, our game plan in this lesson ...

1. We'll sketch some ideas on the user interface and user's interaction with the app
2. We'll set up the new app, creating a new project, copying assets into the project, setting up the AppResources.resx ... all things we learned about before, but tasks we now need to perform for our brand new project

1. Sketch out the screens in a low-tech mockup

As we set out to think through designing our first commercial app, I would recommend you take some time and read through this:

## Windows Phone 8 Design Process

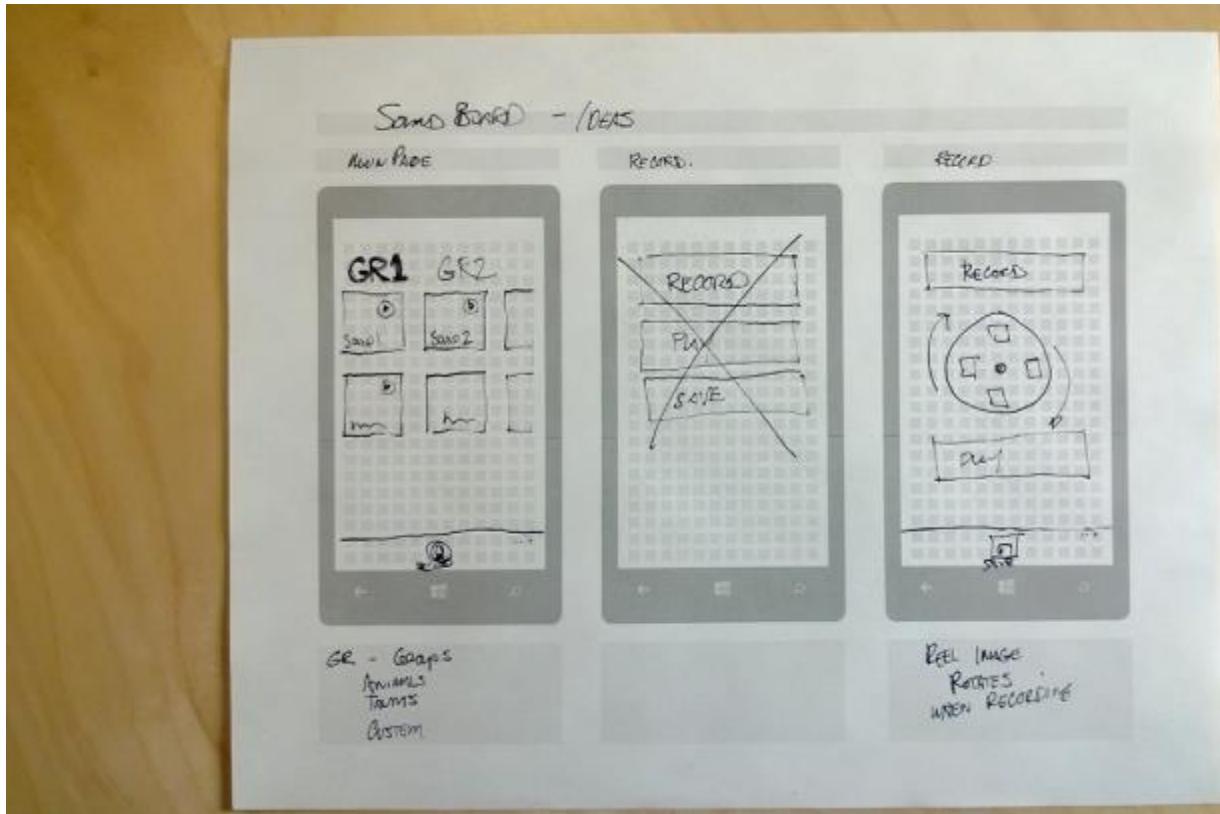
<https://dev.windowsphone.com/en-us/design/process>

Also, there's also a template for sketching a "low tech mockup":

<http://go.microsoft.com/fwlink/?LinkId=266572>

These templates are not only useful for getting your ideas down on paper, but also for thinking through the users interaction with the app, and what feedback we can show the user about the functionality of the app. I can also use a "low tech mockup" to communicate my design and interaction ideas to other to gather their feedback before I waste time developing the app -- just to realize I made a major flaw because I assumed too much.

So, I use that template to sketch my ideas for the SoundBoard app:



The design was largely inspired by the Pivot App Project template ... there will be several PivotItems ("views" or "types") each containing a number of sound tiles. When you tap on a tile, it will play the sound.

I also want to be able to record a sound. I have an app bar at the bottom of the main page ... tapping that will allow you to record a sound. After thinking I needed three buttons, we had the idea that we could provide some visual feedback ... we can easily create a reel, like those old reel-to-reel players from the 1960's, and animate that while recording. That would offer some great visual feedback to the user. Clicking a save application bar button would then allow you to provide a name for the sound and it would be displayed in a PivotItem titled "mine" or "custom" or something like that.

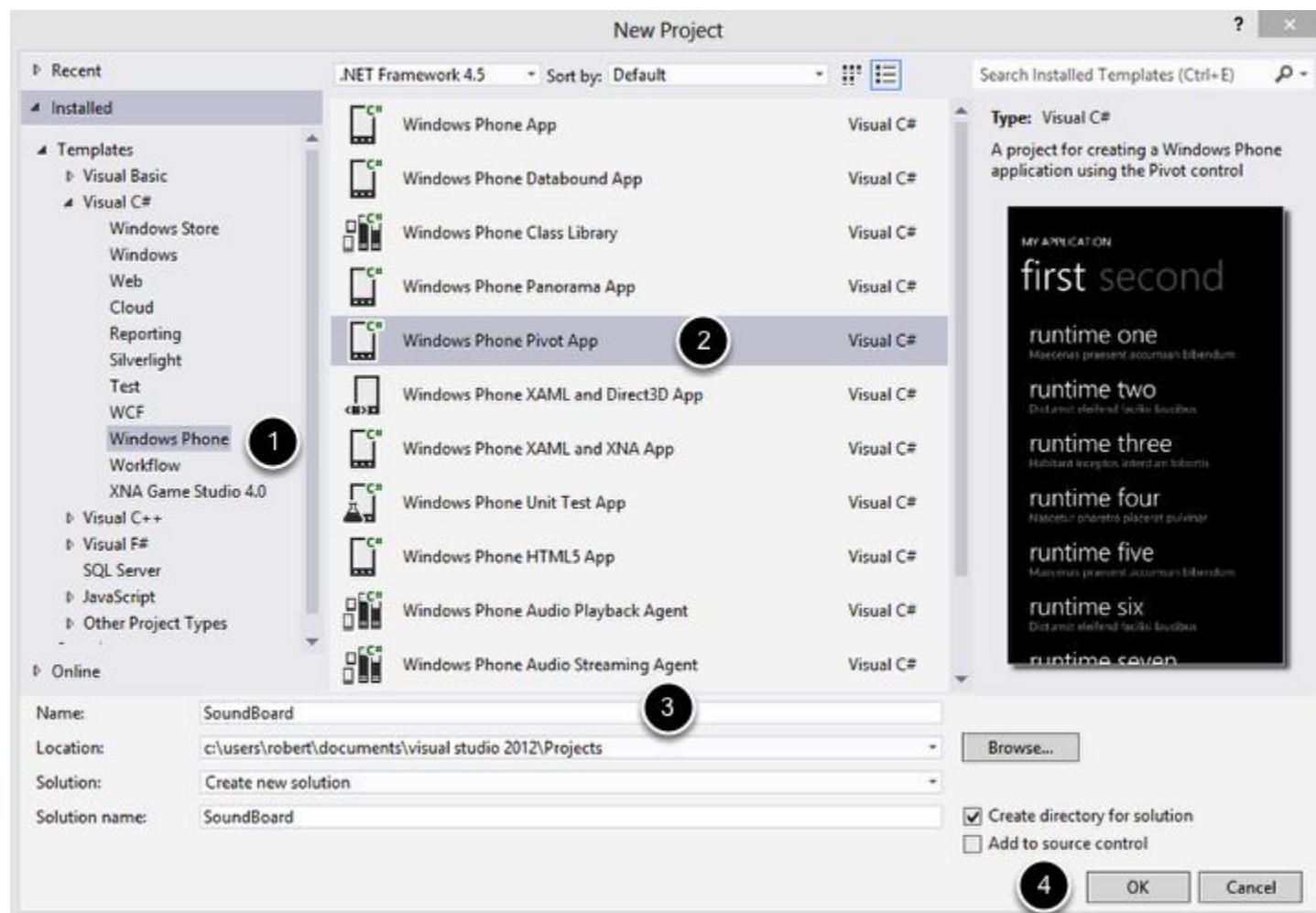
Now, besides the design process, there's always some effort in getting the assets you need for a project like this. In this case, we're providing you with the sounds and the images to follow along. Truth be told, Clint spent a few hours working with a sound guy and a graphic artist. In my experience, the effort required to bring in creative types, or to build creative assets yourself, should not be underestimated. It can be challenging even in a simple app. If you have serious commercial aspirations, I would find someone well versed in those areas that you are not an expert. You can waste a lot of time trying to make your images and sounds just right, and still fall far short of something that can pass as professional quality. Whether it be websites or apps I've worked on, I've never

regretted paying professional designers for their help. It's made me look more professional than I really am.

At any rate, with just a little planning up front and some time spent acquiring the assets I'll need, I have a good direction for my app. Let's go ahead and get started developing the app.

2. Create the SoundBoard project based on the Windows Phone Pivot App project template

File | New | Project ... menu opens the New Project dialog:



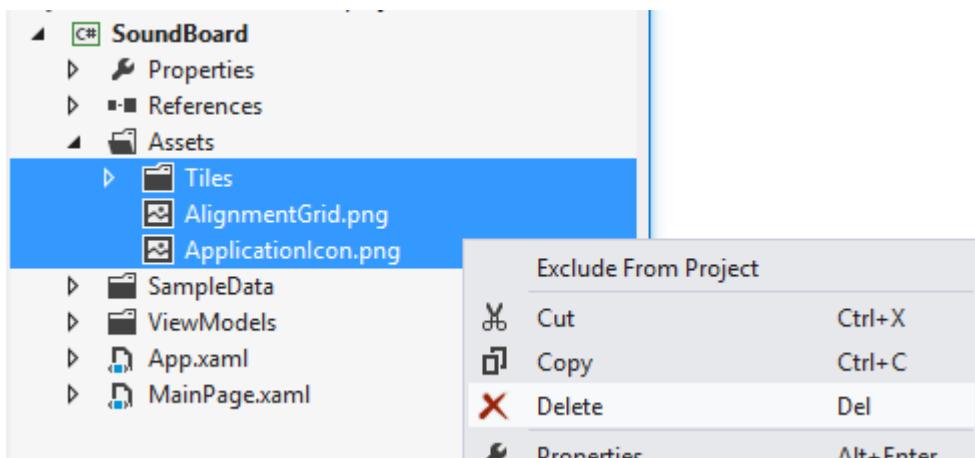
Just like we did earlier in this lesson:

1. Make sure you're in the Installed | Templates | Visual C# | Windows Phone section

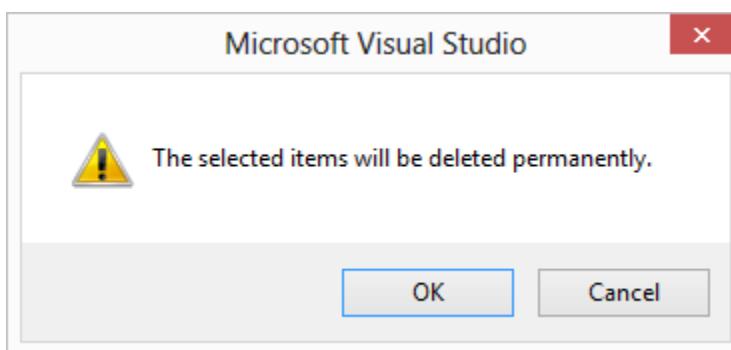
2. This time, choose the Windows Phone Pivot App project template
3. Change name to SoundBoard
4. Click OK

3. Replace project assets

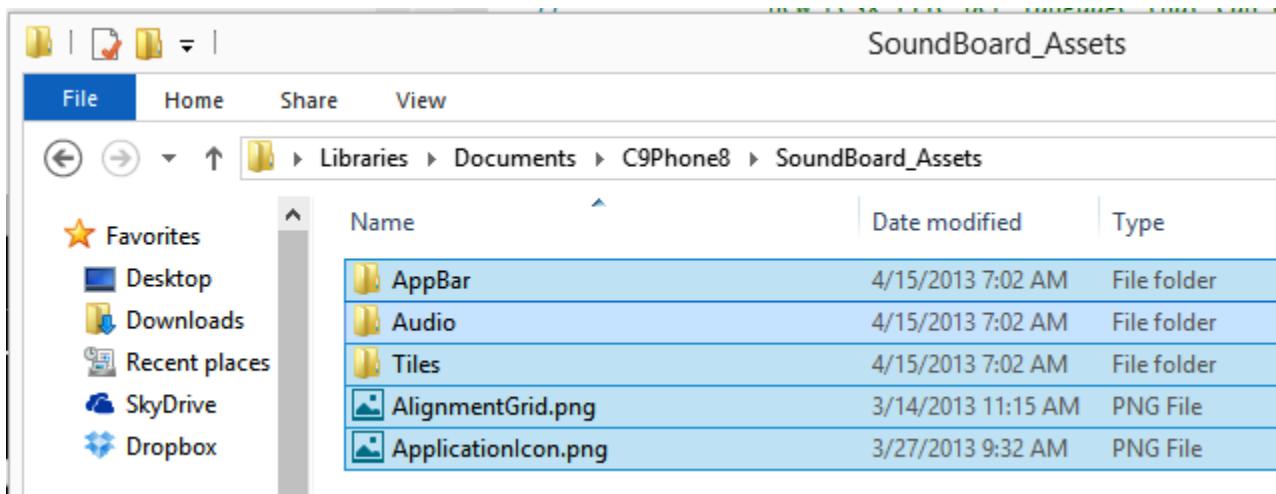
In the Solution Explorer, open the Assets folder and delete all files and subfolders:



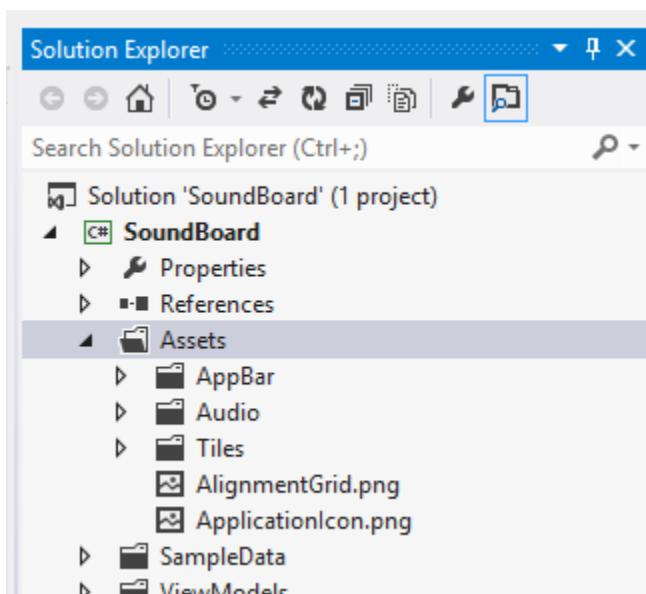
... and confirm the deletions ...



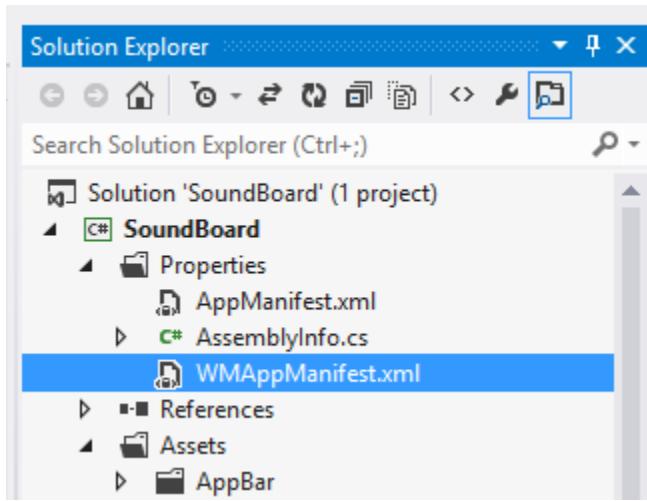
In the assets file that is available wherever you originally downloaded this document or watched the associated videos, there's a subfolder called: SoundBoard\_Assets ... select all files and subfolders ...



... and drag and drop into the SoundBoard project's Assets folder in the Solution Explorer:



4. Confirm the project icons are properly referenced  
In the Properties folder, open the WMAppManifest.xml file:



... and confirm that the App Icon, Small Icon and Medium Icon are set properly ...

WMAAppManifest.xml ✘ X MainPage.xaml

Use this designer to set or modify some of the properties in the Windows Phone app manifest.

Application UI Capabilities Requirements Packa...

Use this page to set the UI details that identify and describe your application.

Display Name: SoundBoard

Description: Sample description

Navigation Page: MainPage.xaml

App Icon:



Supported Resolutions:



wvga     wxga     720p

Tile Template: TemplateFlip

Support for large Tiles

Tile Title: SoundBoard

Tile Images:

Small: 

Medium: 

Windows Phone 8 supports three Tile templates: flip, iconic, and cycle. This link does a nice job explaining the different types:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202948\(v=vs.105\).aspx#BKMK\\_Tiletemplates](http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202948(v=vs.105).aspx#BKMK_Tiletemplates)

In this app, our needs are simple. Later on in this series, we'll use a Cycle template and see how to programmatically give it a list of images to cycle through. Personally? I love any type of data visualization so I love those apps that use the iconic template for anything that has a lot of updatable information that can be displayed in a tile. For example, on my own phone, I have a weather app that updates the tile with the temperature, humidity, wind and visibility, I have another app that displays the amount of battery remaining in an iconic tile, and a countdown app with a count down until I take a vacation. I have actively sought apps like these in the Store because I think they make my Start page look cool.

5. Configure the title of main page to pull from the AppResources.resx file

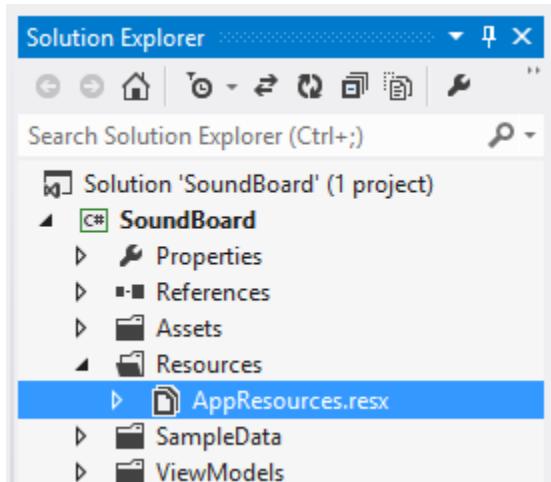
While we don't plan on localizing our application completely, I will go ahead and set up the app so that we could easily add it in the future.

Open the MainPage.xaml and locate the <phone:Pivot> element's Title attribute. Change it to:

```
Title="{Binding Path=LocalizedResources.ApplicationTitle, Source={StaticResource LocalizedStrings}}"
```

```
38
39
40     <!--Pivot Control-->
41     <phone:Pivot Title="{Binding Path=LocalizedResources.ApplicationTitle,
42                               Source={StaticResource LocalizedStrings}}">
        <!--Pivot item one-->
```

Now I'll change the app's title in the AppResources.resx. You'll find it in the Resources folder.



Change the ApplicationTitle value to "SoundBoard". Save and close the file.

The screenshot shows the 'AppResources.resx' editor. At the top, there are tabs for 'AppResources.resx', 'WMAppManifest.xml', and 'MainPage.xaml'. Below the tabs, there are buttons for 'Strings', 'Add Resource', 'Remove Resource', and 'Access Modifier: Public'. The main area is a table with columns 'Name' and 'Value'. The 'ApplicationTitle' row has its 'Value' cell highlighted with a blue background, containing the text 'SoundBoard'.

	Name	Value
	AppBarButtonText	add
	AppBarMenuItemText	Menu Item
..	ApplicationTitle	SoundBoard
	ResourceFlowDirection	LeftToRight
	ResourceLanguage	en-US
	SampleProperty	Sample Runtime Property Value
*		

## Recap

To recap, not much new in this lesson per se, but these are the types of tasks we need to perform each time we undertake a new project. We did talk about a few new topics, like focusing on the design and interaction of the app, and choosing from the different types of tile templates that are available.

# Part 12: Improving the View Model and Sample Data

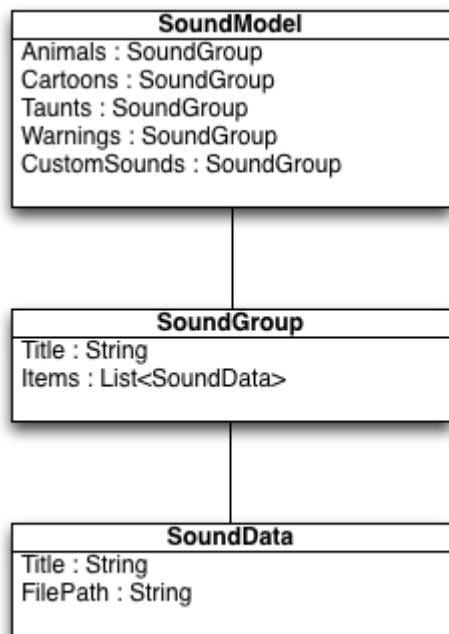
Source Code: <http://aka.ms/absbeginnerdevwp8>

Now that we've completed the basic setup chores, we want to focus on the heart of the application: the data model. This requires we translate our low-tech mockups from the previous lesson into a viable data model that best represents the data and hierarchical relationships between the data.

Here's our game plan for this lesson:

1. We'll briefly analyze our "requirements" (the mockup is all we have) and will quickly diagram a data model that we'll implement in our app.
  2. We'll implement the data model in code.
  3. We'll create some data in an XML file that will be used as DESIGN-TIME data for the purpose of displaying something in our MainPage's XAML Designer.
  4. Modify the MainPage.xaml's binding expressions to point to instances of classes and properties in our data model.
1. Analyze our mockup and design a data model

After looking over the mockups and thinking about how best to delegate responsibilities to the various classes, I come up with a very basic class model that is my first best guess at how to structure the data I'll need for the app:



The SoundModel class will contain references to each of the five groupings of sounds in the app. Notice that each of these properties are of type SoundGroup.

The SoundGroup class represents what I've been calling a "view" or "type". I've struggled for the correct terms, but in essence it is a grouping, not a view (which only connotes the visual aspect of the sound tiles) or a type (which has special connotations in the .NET world). A SoundGroup has a title (that will be used to display as the title of the PivotItem in the Pivot control on the MainPage.xaml) and a generic collection of SoundData.

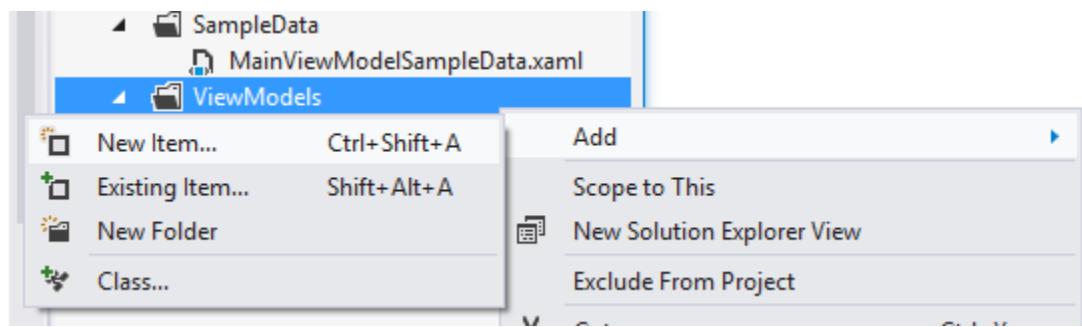
The SoundData class represents the sounds themselves. Each instance of the SoundData will be displayed in a tile, and by tapping the tile, we'll play the sound associated with the SoundData as stored in the FilePath property.

Given this clarity, it's time to implement this diagram in code. I'm sure there will be a few additions as we go along, but this is a good start.

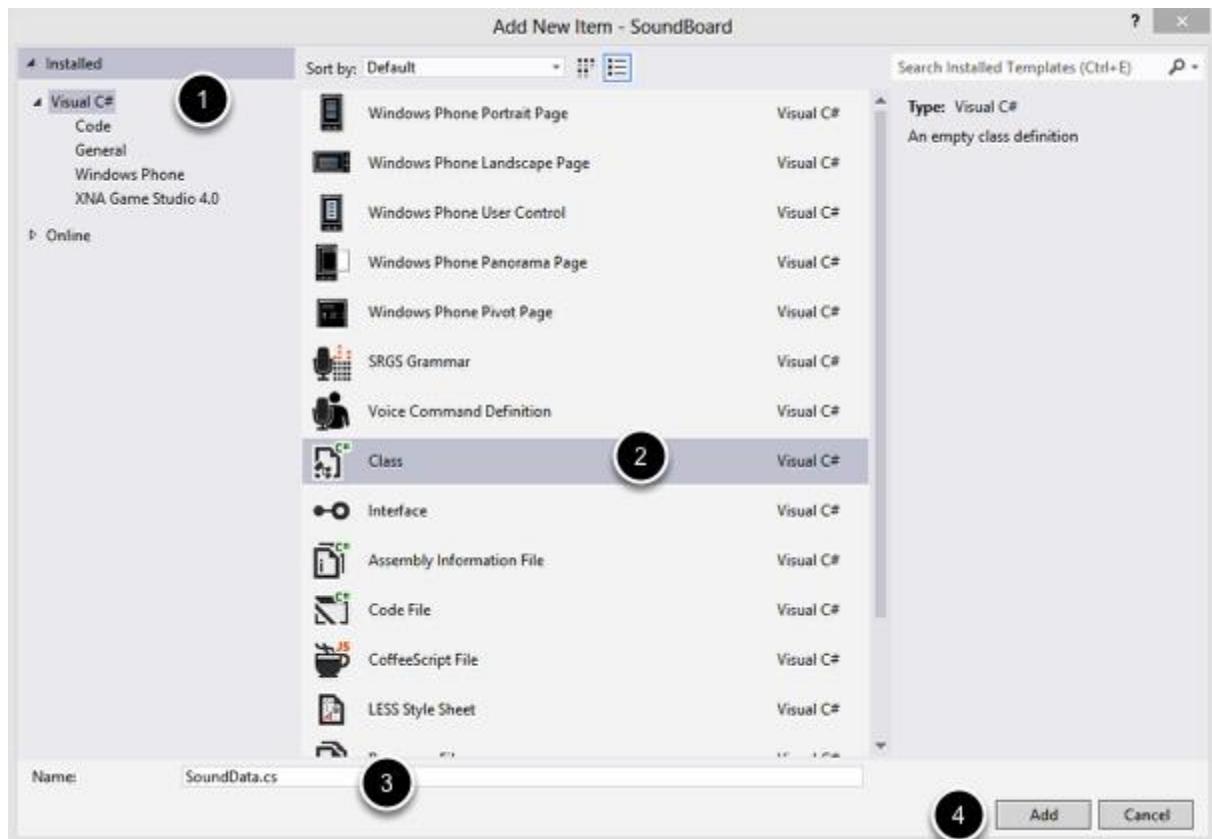
## 2. Create the new data model classes for our app

With a plan and our data model diagram in tow, we'll begin by implementing the new data model.

Right-click ViewModels folder, Add | New Item:



... open the Add New Item dialog:



1. Make sure you're in the Visual C# file templates
2. Select the Class file template
3. Rename to: SoundData.cs
4. Click Add button

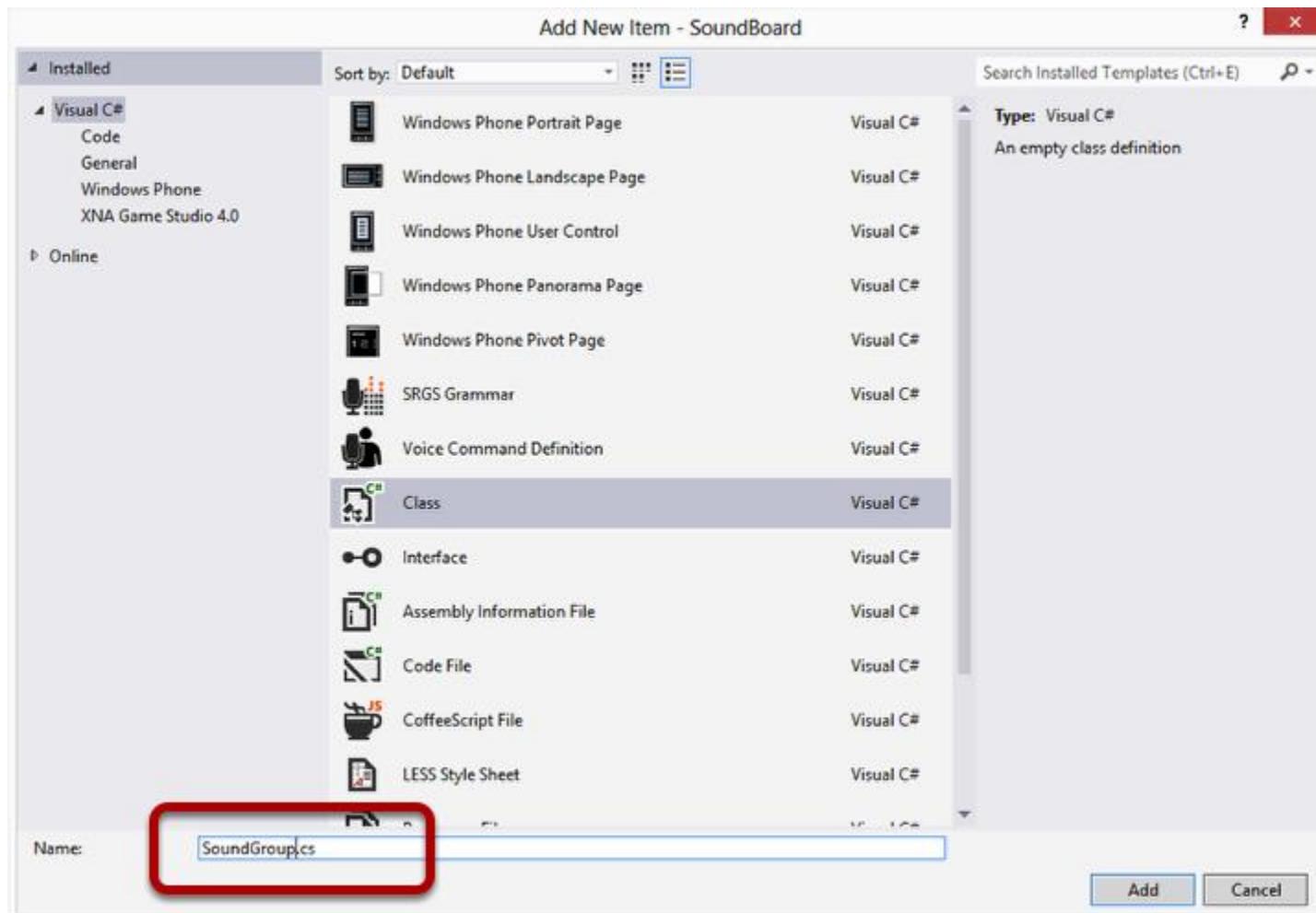
The SoundData.cs file is created and loaded into the main area:

```
SoundData.cs X AppResources.resx WMAppManifest.xml
SoundBoard.ViewModels.SoundData
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace SoundBoard.ViewModels
8 {
9     class SoundData
10    {
11    }
12 }
13
```

... we'll add two public properties: Title and FilePath:

```
ItemViewModel.cs SoundData.cs X AppResources.resx WMAppManifest.xml
SoundBoard.ViewModels.SoundData
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace SoundBoard.ViewModels
9 {
10     public class SoundData
11     {
12         public string Title { get; set; }
13         public string FilePath { get; set; }
14     }
15 }
16
```

Add another new item to the ViewModels folder using the technique described a moment ago. You'll create another Class called: SoundGroup.cs:



In the new SoundGroup.cs, we'll create two public properties and a constructor:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace SoundBoard.ViewModels
8  {
9      public class SoundGroup
10     {
11         public SoundGroup()
12         {
13             Items = new List<SoundData>();    2
14         }
15
16         public List<SoundData> Items { get; set; }  1
17         public string Title { get; set; }
18     }
19 }
20
```

1. Create the Title property and Items property. Items will be a generic list of SoundData.
2. In the constructor, we initialize Items setting it to a new instance of List<SoundData>.

We'll add one more class file to the ViewModels folder called SoundModel. SoundModel will contain a SoundGroup for each "view" or "categories" of sounds we'll display in our app. It will also implement the logic to determine whether data has been loaded into the new data model, and the vast majority of the code therein will create instances of the SoundData classes for each sound we want to use in our app.

We'll start simple and grow this important class throughout this lesson:

```
SoundModel.cs  ✘ X SoundGroup.cs      ItemViewModel.cs      SoundData.cs
SoundBoard.ViewModels.SoundModel
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace SoundBoard.ViewModels
8  {
9      public class SoundModel
10     {
11         public SoundGroup CustomSounds { get; set; }
12         public SoundGroup Animals { get; set; }
13         public SoundGroup Cartoons { get; set; }
14         public SoundGroup Taunts { get; set; }
15         public SoundGroup Warnings { get; set; }
16
17         public bool IsDataLoaded { get; set; }
18
19         public void LoadData()
20         {
21             // Load data into the model
22
23             IsDataLoaded = true;
24         }
25     }
26 }
27
```

1. In lines 11 through 15 we'll create properties representing each SoundGroup (i.e., category) of sounds.
2. We'll use the IsDataLoaded property to determine whether or not we need to perform the LoadData() method to create instances of SoundGroup and SoundData.
3. We'll load the data into the model using a number of private helper method which we'll implement in just a moment.

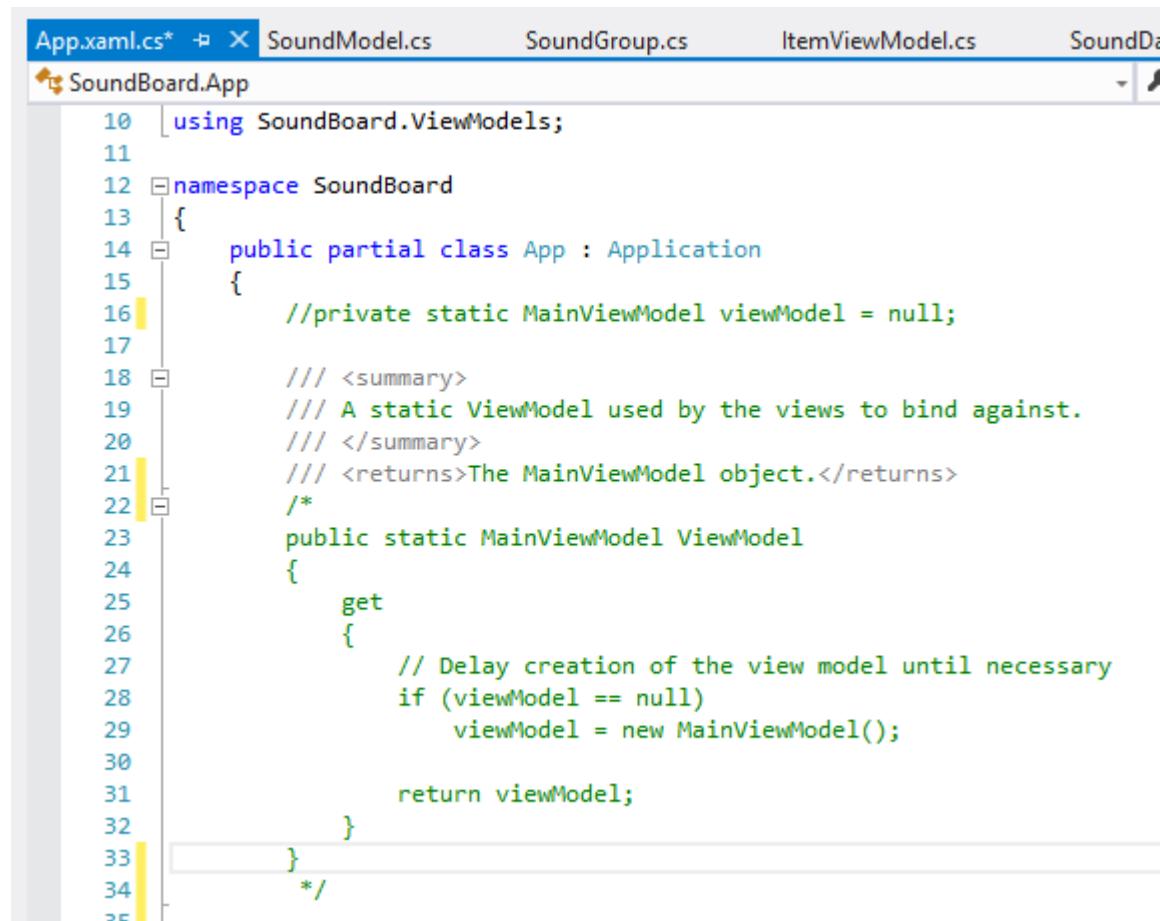
### 3. Modify the App.xaml.cs to use the new data model

Before we perform the bulk of the work in the LoadData() method (and create helper methods to the heavy lifting) I want to (a) replace the OLD data model (from the project template) in the app with our new one we just created by changing the class that the App.xaml.cs uses to back its ViewModel property.

First, I'll comment out references to the OLD model in the App.xaml.cs. I typically comment out things I no longer THINK I need before I actually delete them from the

project. This is a habit I've acquired through the years. I've hastily deleted code thinking I'll replace it, only to realize there was some small detail I was missing. However, since I deleted the old working version and can no longer reference it, it becomes more time consuming and painful to determine where I made the mistake.

Here I comment out the implementation of MainViewModel property, both the private backing field as well as the public getter and setter:



```
App.xaml.cs*  X SoundModel.cs      SoundGroup.cs      ItemViewModel.cs      SoundDa
SoundBoard.App
10  using SoundBoard.ViewModels;
11
12  namespace SoundBoard
13  {
14      public partial class App : Application
15      {
16          //private static MainViewModel viewModel = null;
17
18          /// <summary>
19          /// A static ViewModel used by the views to bind against.
20          /// </summary>
21          /// <returns>The MainViewModel object.</returns>
22          /*
23          public static MainViewModel ViewModel
24          {
25              get
26              {
27                  // Delay creation of the view model until necessary
28                  if (viewModel == null)
29                      viewModel = new MainViewModel();
30
31                  return viewModel;
32              }
33          }
34      */
35  }
```

Next, I'll implement the new version of the ViewModel using our new data model type SoundModel:

```
36     private static SoundModel viewModel = null;
37
38     public static SoundModel ViewModel
39     {
40         get
41         {
42             if (viewModel == null)
43                 viewModel = new SoundModel();
44
45             return viewModel;
46         }
47     }
48
```

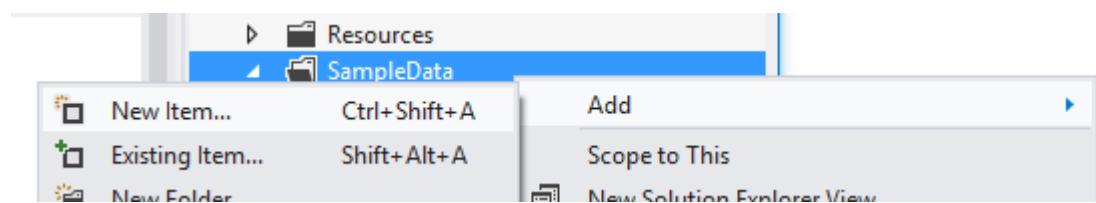
That should be all that's required. References to the SoundModel's IsDataLoaded property and LoadData() method should work just fine.

Now that I've removed the old data model from the app, I'm sure I've broken several other things. At this moment, I'm most concerned about the XAML views. The binding expressions are pointing at properties that no longer exist. Furthermore, at DESIGN-TIME, there's no sample data to display in the XAML designer, and at RUNTIME there's no data either.

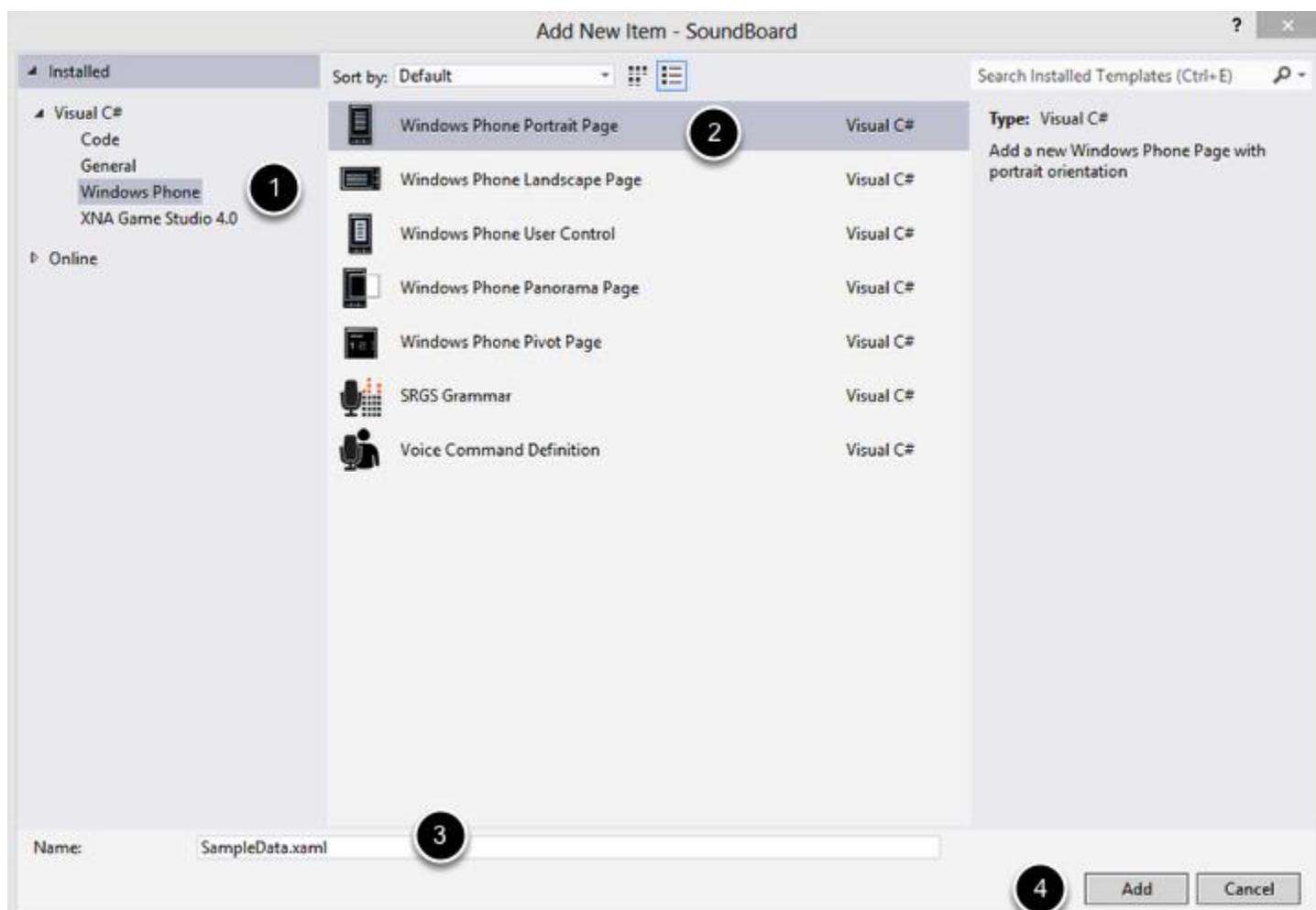
#### 4. Create sample / design-time display data

It will take some time to massage all of this back into working form, so I have to do this in small chunks. My first priority is to make some DESIGN-TIME data available so that I can then edit the XAML views and make sure they're displaying data from my new model correctly.

I'll right-click the SampleData folder, select Add | New Item ... from the context menu:



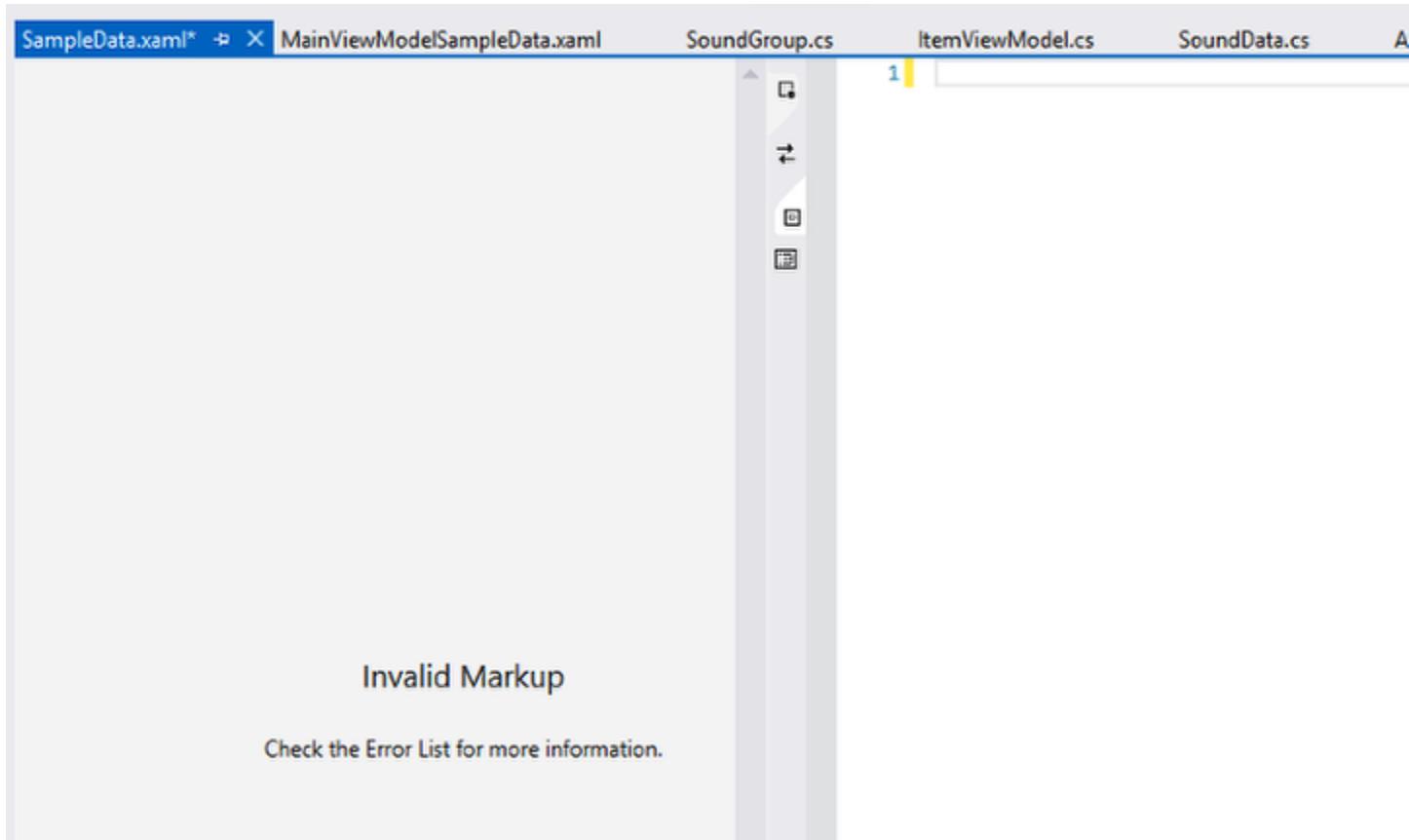
... which opens the Add New Item dialog:



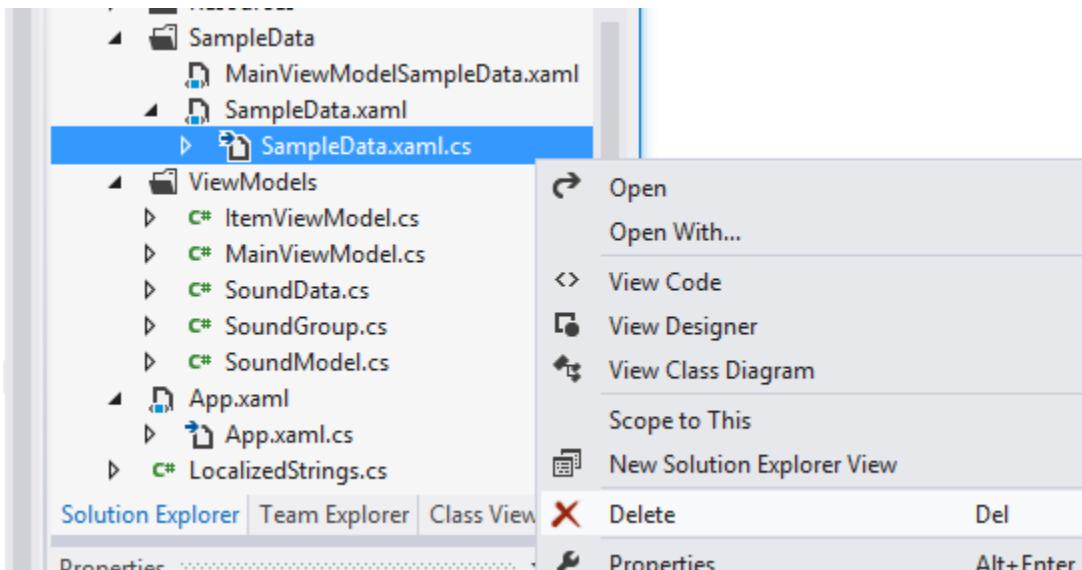
1. Make sure you're in the Windows Phone file templates
2. Select the Windows Phone Portrait Page file template
3. Rename the file to: SampleData.xaml
4. Click Add

I'll use this file for sample data, not as a XAML page, therefore I'll need to remove everything about the templated file that makes it uniquely

When the new SampleData.xaml file opens in the main area, I'll remove all of the XAML inside that file:



In the Solution Explorer, I'll delete the code behind file that's associated with my new SampleData.xaml file by right-clicking and selecting Delete from the context menu:



Back in the SampleData.xaml, I hide the visual XAML designer. This file will contain just data, nothing that could be displayed in the designer:

I add quite a bit of code in the SampleData.xaml file:

```
1 <vm:SoundModel
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:vm="clr-namespace:SoundBoard.ViewModels">
5
6     <vm:SoundModel.Animals>
7         <vm:SoundGroup Title="Animals Sample">
8             <vm:SoundGroup.Items>
9                 <vm:SoundData Title="Animals 1" FilePath="Animals.wav" />
10                </vm:SoundGroup.Items>
11            </vm:SoundGroup>
12        </vm:SoundModel.Animals>
13
14        <vm:SoundModel.Cartoons>
15            <vm:SoundGroup Title="Cartoons Sample">
16                <vm:SoundGroup.Items>
17                    <vm:SoundData Title="Cartoons 1" FilePath="Cartoons.wav" />
18                    <vm:SoundData Title="Cartoons 2" FilePath="Cartoons.wav" />
19                </vm:SoundGroup.Items>
20            </vm:SoundGroup>
21        </vm:SoundModel.Cartoons>
22
23        <vm:SoundModel.Taunts>
24            <vm:SoundGroup Title="Taunts Sample">
25                <vm:SoundGroup.Items>
```

Notice the XAML Namespace prefix "vm"—it references the CLR namespace SoundBoard.ViewModels—the Namespace for the new data model we just created. We want the XAML in this document to create instances of our SoundModel, SoundGroup, and SoundData classes.

## 5. Use the new sample data in the MainPage.xaml

Next, I want to modify the MainPage.xaml file to change the design-time data (i.e., the data we're loading into our design-time) that the XAML Designer loads and displays:

```

1  <phone:PhoneApplicationPage
2      x:Class="SoundBoard.MainPage"
3      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6      xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
9      mc:Ignorable="d"
10     d:DataContext="{d:DesignData SampleData/SampleData.xaml}" ←
11     FontFamily="{StaticResource PhoneFontFamilyNormal}"
12     FontSize="{StaticResource PhoneFontSizeNormal}"

```

Now that we have a data model and real "fake" data, we can see the impact of changes we make in the DataTemplate for our LongListSelector. I'll make changes to the binding expressions in each of the TextBlocks:

The screenshot shows a portion of the MainPage.xaml XAML code. A red arrow points to the line `d:DataContext="{d:DesignData SampleData/SampleData.xaml}"`. Four numbered callouts point to specific parts of the code:

- Callout 1: Points to the `ItemsSource="{Binding Animals.Items}"` line.
- Callout 2: Points to the first `TextBlock Text="{Binding Title}"` line.
- Callout 3: Points to the second `TextBlock Text="{Binding FilePath}"` line.
- Callout 4: Points to the `Header="{Binding Animals.Title}"` line.

```

39     <!--Pivot Control-->
40     <phone:Pivot Title="{Binding Path=LocalizedResources.ApplicationTitle,
41                             Source={StaticResource LocalizedStrings}}">
42         <!--Pivot item one-->
43         <phone:PivotItem Header="{Binding Animals.Title}">
44             <!--Double line list with text wrapping-->
45             <phone:LongListSelector
46                 Margin="0,0,-12,0"
47                 ItemsSource="{Binding Animals.Items}">
48                 <phone:LongListSelector.ItemTemplate>
49                     <DataTemplate>
50                         <StackPanel Margin="0,0,0,17">
51                             <TextBlock Text="{Binding Title}" 1
52                                 TextWrapping="Wrap"
53                                 Style="{StaticResource PhoneTextExtraLargeStyle}"/>
54                             <TextBlock Text="{Binding FilePath}" 2
55                                 TextWrapping="Wrap"
56                                 Margin="12,-6,12,0"
57                                 Style="{StaticResource PhoneTextSubtleStyle}"/>
58                         </StackPanel>
59                     </DataTemplate>
60                 </phone:LongListSelector.ItemTemplate>
61             </phone:LongListSelector>
62         </phone:PivotItem>

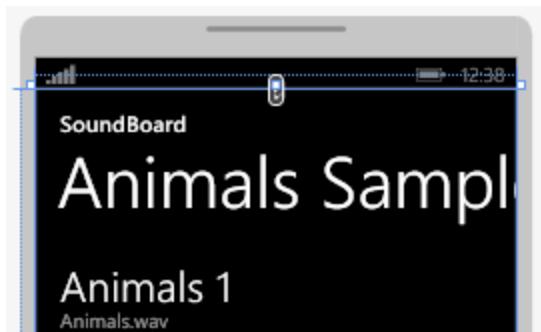
```

1. I'll set the ItemSource binding of the LongListSelector to Animals.Items ... Animals is a public property of the SoundModel, which is the parent class that's defined in the SampleData.xaml file, which we set as the DataContext of the entire MainPage.xaml

page. Therefore, every child property of the SoundModel in that file is accessible (such as Animals, Cartoons, Taunts, etc.).

2. Since the LongListSelector.ItemsSource is set to Animals.Items, and Items is a List<SoundData>, we can reference any property of the SoundData class inside the DataTemplate. So, I set the Text binding to the Title property (of the SoundData class), and ...
3. I set the Text binding to the FilePath property (of the SoundData class).
4. I also set the Header binding in for the PivotItem to Animals.Title. Again, I can reference Animals because it's parent, the SoundModel, is set as the DataContext for the entire MainPage.xaml.

In the XAML designer, we should now see sample data appear:



Hopefully you can see the correlation between the MainPage.xaml's controls and the sample data for Animals:

```
6   <vm:SoundModel.Animals>
7     <vm:SoundGroup Title="Animals Sample">
8       <vm:SoundGroup.Items>
9         <vm:SoundData Title="Animals 1" FilePath="Animals.wav" />
10      </vm:SoundGroup.Items>
11    </vm:SoundGroup>
12  </vm:SoundModel.Animals>
13
```

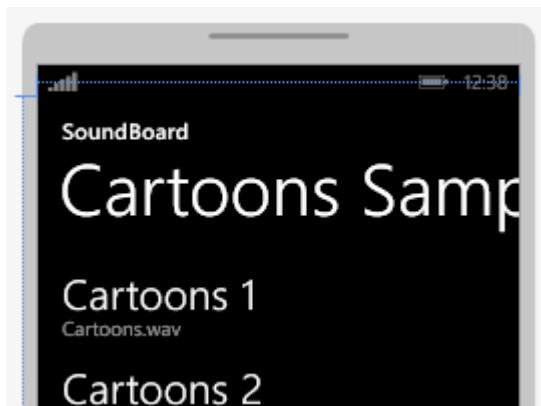
We made a change to the FIRST PivotItem, and now we will change the second pivot item to display the SoundGroup Cartoons. Notice it's practically identical to what I wrote for Animals. I simply substituted Animals for Cartoons and it works.

```

64          <!--Pivot item two-->
65      <phone:PivotItem Header="{Binding Cartoons.Title}">
66          <!--Double line list no text wrapping-->
67          <phone:LongListSelector
68              Margin="0,0,-12,0"
69              ItemsSource="{Binding Cartoons.Items}">
70              <phone:LongListSelector.ItemTemplate>
71                  <DataTemplate>
72                      <StackPanel Margin="0,0,0,17">
73                          <TextBlock Text="{Binding Title}"
74                              TextWrapping="NoWrap"
75                              Margin="12,0,0,0"
76                              Style="{StaticResource PhoneTextExtraLargeStyle}"/>
77                          <TextBlock Text="{Binding FilePath}"
78                              TextWrapping="NoWrap"
79                              Margin="12,-6,0,0"
80                              Style="{StaticResource PhoneTextSubtleStyle}"/>
81                  </StackPanel>
82          </DataTemplate>
83      </phone:LongListSelector.ItemTemplate>
84  </phone:LongListSelector>
85 </phone:PivotItem>

```

When I put my mouse cursor anywhere inside the XAML definition for the second PivotItem, the XAML designer changes its view to that PivotItem:



## Recap

To recap, the big takeaway in this lesson is how we replaced the default data model and sample data in the Pivot project template with our own new data model for working with groups of sounds and bound to the new sample data by modifying the MainPage.xaml's page and control declarations. We learned about data contexts and how they're set and accessed from within the page.

# Part 13: Styling Tiles in the LongListSelector

Source Code: <http://aka.ms/absbeginnerdevwp8>

In the previous lesson, we made a lot of progress in wiring up a new data model to the MainPage.xaml and now we need to focus on the layout of the DataTemplate in our LongListSelector—we'll want to tweak each instance of the SoundData so that they resemble tiles instead of rows. So, if we compare our drawings with the current state of the app's UI, the layout is not quite right. We'll remedy that in this lesson.

Game plan:

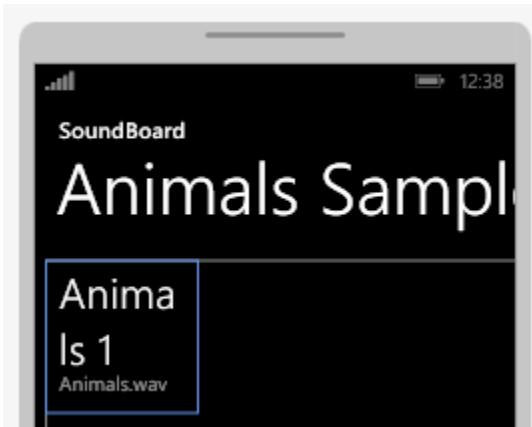
1. We'll modify the LongListSelector to utilize the Grid LayoutMode.
2. We'll completely re-work the layout of each DataTemplate so that it more closely matches the layout of a tile.
3. Since we need five or six PivotItems, each with their own LongListSelector, it doesn't make sense to keep defining the tile layouts, so we'll abstract them into a centralized template that can be used by all the LongListSelectors on our MainPage.xaml.

1. Change the LongListSelector's LayoutMode to Grid

The LongListSelector control sports a LayoutMode property. It accepts an enumeration of either List (which is the default) or Grid. By setting the LayoutMode to Grid, and setting a cell size in terms of width and height, we can quickly change the appearance of the LongListSelector.

```
44
45
46
47
48
49
50
51
    <!--Double line list with text wrapping-->
    <phone:LongListSelector
        Margin="0,0,-12,0"
        ItemsSource="{Binding Animals.Items}"
        LayoutMode="Grid"
        GridCellSize="150,150" <-->
    >
```

By adding lines 49 and 50, the LongListSelector now looks like this:



... where each instance of the DataTemplate looks like a tile.

## 2. Modify the DataTemplate to create the layout we need

Next, we need to modify what's INSIDE the DataTemplate to match our desired layout. I'll highlight and delete everything inside of the DataTemplate:

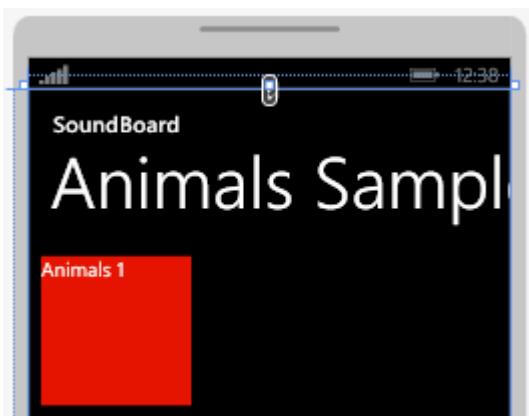
```
52 <phone:LongListSelector.ItemTemplate>
53   <DataTemplate>
54     ...
55       <StackPanel Margin="0,0,0,17">
56         <TextBlock Text="{Binding Title}" ...
57           TextWrapping="Wrap"
58           Style="{StaticResource PhoneTextExtraLargeStyle}"/>
59         <TextBlock Text="{Binding FilePath}" ...
60           TextWrapping="Wrap"
61           Margin="12,-6,12,0"
62           Style="{StaticResource PhoneTextSubtleStyle}"/>
63       </StackPanel>
64     </DataTemplate>
65   </phone:LongListSelector.ItemTemplate>
```

... and then I'll start over with a Grid control with a single cell. Inside the Grid, I'll use a TextBlock for the name of the sound at the bottom of the tile. I'll wrap a StackPanel around it so that I can set its vertical position within the Grid cell later. So, I add the following code inside of the DataTemplate:

```
52     <phone:LongListSelector.ItemTemplate>
53         <DataTemplate>
54             <Grid Background="{StaticResource PhoneAccentBrush}">
55                 <StackPanel>
56                     <TextBlock Text="{Binding Title}" />
57                 </StackPanel>
58             </Grid>
59         </DataTemplate>
60     </phone:LongListSelector.ItemTemplate>
61 </phone:LongListSelector>
62 </phone:PivotItem>
```

In line 55, notice the Background which I set to the built-in PhoneAccentBrush resource.

That code produces this result:

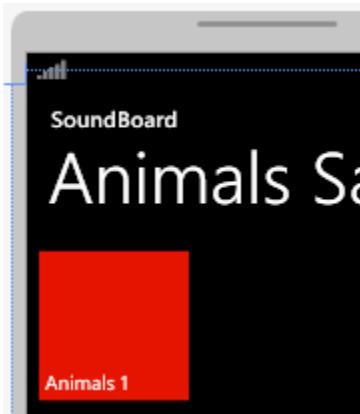


I want to move the TextBlock's position to the bottom and add some padding. I make the following additions:

```
55 <Grid Background="{StaticResource PhoneAccentBrush}">  
56     <StackPanel VerticalAlignment="Bottom">  
57         <TextBlock Text="{Binding Title}" Margin="6, 0, 0, 6" />  
58     </StackPanel>  
59 </Grid>  
60
```

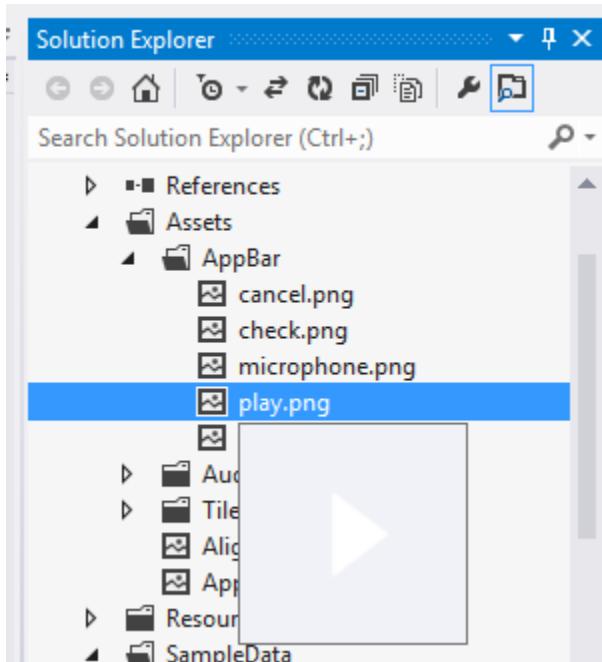
1      2

I set the VerticalAlignment of the StackPanel to "Button", and set margins on the left and bottom sides of the TextBlock. That produces the following:



So far, so good.

Next, I want to add that play button graphic that I sketched in my mockup. To do that, I'll use the Assets\AppBar\play.png icon we added to the project in a previous lesson.



I'll add an inner grid, set its vertical alignment to the top, and its horizontal alignment to the right. The play.png is just an arrow, and I want there to be a circle around it. I guess I could re-work the image, but there's a simple solution in XAML ... I'll just add an Ellipse control around the image.

```

55 <Grid Background="{StaticResource PhoneAccentBrush}">
56   <Grid VerticalAlignment="Top"
57     HorizontalAlignment="Right"
58     Width="40"
59     Height="40"
60     Margin="0, 6, 6, 0">
61     <Ellipse Stroke="{StaticResource PhoneForegroundBrush}"
62       StrokeThickness="3" />
63     <Image Source="/Assets/AppBar/Play.png" />
64   </Grid>
65   <StackPanel VerticalAlignment="Bottom">
66     <TextBlock Text="{Binding Title}" Margin="6, 0, 0, 6" />
67   </StackPanel>
68 </Grid>

```

The code block shows XAML for a grid-based user interface. It includes three numbered callouts: 1 points to the first inner grid, 2 points to the image control, and 3 points to the ellipse control.

1. Add a Grid and set it so that it is small (40 x 40) and anchored to the upper right-hand corner of the tile (via VerticalAlignment and HorizontalAlignment). I also add a small margin to the top and right sides.
2. Use an Image control to load the arrow head image (i.e., the play.png file)

3. Add an Ellipse around the arrow head image. I use a built-in brush for the ellipse ... can you think of why this might be a problem and how I might solve this? Give it some thought. I'm going to leave it as is for now.

These changes make each tile look close to what I had originally envisioned. Now we need to apply this to the other PivotItem's DataTemplates.

3. Adding the DataTemplate's design to the Page's Resources so that we can re-use it  
In the previous lesson, we saw the behavior of the XAML design ... if we put our mouse cursor in a different PivotItem, that PivotItem's DataTemplate appears in the XAML designer view.



Instead of copying and pasting the DataTemplate we created a moment ago into each of the 5 or 6 PivotItems we'll need in the MainPage.xaml, I have a better idea. Let's define the DataTemplate as a page-level resource.

```
1  <phone:PhoneApplicationPage
2      x:Class="SoundBoard.MainPage"
3      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6      xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
9      mc:Ignorable="d"
10     d:DataContext="{d:DesignData SampleData/SampleData.xaml}"
11     FontFamily="{StaticResource PhoneFontFamilyNormal}"
12     FontSize="{StaticResource PhoneFontSizeNormal}"
13     Foreground="{StaticResource PhoneForegroundBrush}"
14     SupportedOrientations="Portrait" Orientation="Portrait"
15     shell:SystemTray.IsVisible="True">
16
17     <phone:PhoneApplicationPage.Resources>
18
19     </phone:PhoneApplicationPage.Resources>
20
--
```

In lines 17 and 19 I've defined a Resources section for the page. I'll merely cut and paste the DataTemplate we created a moment ago here, and then reference it in each PivotItem where we need it.

After I cut and paste the DataTemplate into its new home as a page-level Resource, I add a Key attribute. This Key will allow me to reference my DataTemplate from the rest of the page when I need it.

```
17     <phone:PhoneApplicationPage.Resources>
18         <DataTemplate x:Key="SoundTileDataTemplate">
19             <Grid Background="{StaticResource PhoneAccentBrush}">
20                 <Grid VerticalAlignment="Top"
21                     HorizontalAlignment="Right"
22                     Width="40"
23                     Height="40"
24                     Margin="0, 6, 6, 0">
25                     <Ellipse Stroke="{StaticResource PhoneForegroundBrush}"
26                         StrokeThickness="3" />
27                     <Image Source="/Assets/AppBar/Play.png" />
28                 </Grid>
29                 <StackPanel VerticalAlignment="Bottom">
30                     <TextBlock Text="{Binding Title}" Margin="6, 0, 0, 6" />
31                 </StackPanel>
32             </Grid>
33         </DataTemplate>
34     </phone:PhoneApplicationPage.Resources>
```

Now, I just need to reference the Key, SoundTileDataTemplate, in my LongListSelector's ItemTemplate property like so:

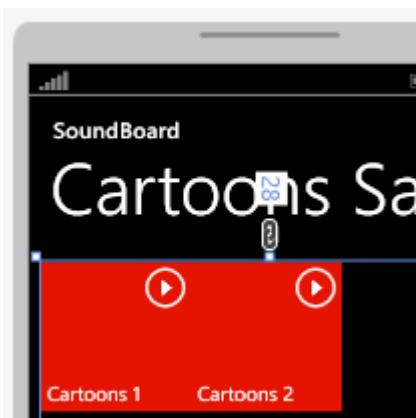
```
62     <!--Pivot item one-->
63     <phone:PivotItem Header="{Binding Animals.Title}">
64         <phone:LongListSelector
65             Margin="0,0,-12,0"
66             ItemsSource="{Binding Animals.Items}"
67             LayoutMode="Grid"
68             GridCellSize="150,150"
69             ItemTemplate="{StaticResource SoundTileDataTemplate}"
70             />
71     </phone:PivotItem>
```



... and I'll repeat that process for the second PivotItem:

```
73      <!--Pivot item two-->
74      <phone:PivotItem Header="{Binding Cartoons.Title}">
75          <!--Double line list no text wrapping-->
76          <phone:LongListSelector
77              Margin="0,0,-12,0"
78              ItemsSource="{Binding Cartoons.Items}"
79              LayoutMode="Grid"
80              GridCellSize="150,150"
81              ItemTemplate="{StaticResource SoundTileDataTemplate}"
82          />
83      </phone:PivotItem>
```

... and something interesting happens. When I look at the XAML designer, I can see two tiles butted up right next to each other:



Why had I not noticed this before? Because the first PivotItem, Animals, only had one SoundData object associated with it. Clearly we'll need to remedy this because it doesn't look right, and as a result it may confuse the user. They may not see distinct tiles but rather one long bar of red or whatever PhoneAccentBrush color they chose.

To remedy this, I add a margin to the bottom and right of each Grid that defines the boundaries of each DataItem tile:

```
16 | 
17 |     <phone:PhoneApplicationPage.Resources>
18 |         <DataTemplate x:Key="SoundTileDataTemplate">
19 |             <Grid Background="{StaticResource PhoneAccentBrush}">
20 |                 Margin="0, 0, 12, 12">
21 |                     <Grid VerticalAlignment="Top">
```



... and that seems to provide the visual separation that I wanted:



Great!

The last task is to create PivotItems for each of the other DataGroups I'll support in my app, including Taunts, Warnings and Custom Sounds. I merely copy and paste one of my existing PivotItems three times like so:

```

84     <phone:PivotItem Header="{Binding Taunts.Title}">
85         <phone:LongListSelector
86             Margin="0,0,-12,0"
87             ItemsSource="{Binding Taunts.Items}"
88             LayoutMode="Grid"
89             GridCellSize="150,150"
90             ItemTemplate="{StaticResource SoundTileDataTemplate}"
91         />
92     </phone:PivotItem>
93
94     <phone:PivotItem Header="{Binding Warnings.Title}">
95         <phone:LongListSelector
96             Margin="0,0,-12,0"
97             ItemsSource="{Binding Warnings.Items}"
98             LayoutMode="Grid"
99             GridCellSize="150,150"
100            ItemTemplate="{StaticResource SoundTileDataTemplate}"
101        />
102    </phone:PivotItem>
103
104    <phone:PivotItem Header="{Binding CustomSounds.Title}">
105        <phone:LongListSelector
106            Margin="0,0,-12,0"
107            ItemsSource="{Binding CustomSounds.Items}"
108            LayoutMode="Grid"
109            GridCellSize="150,150"
110            ItemTemplate="{StaticResource SoundTileDataTemplate}"
111        />
112    </phone:PivotItem>
113

```

1

2

Now, I've warned you about the dangers of copying and pasting code before. There's a chance you'll forget to modify the one little bit of data that makes each pasted item different. So, pay attention to the details ... make sure you're modifying each pasted item in two spots ...

1. Change the PivotItem.Header binding to the correct SoundGroup name, and ...
2. Change each LongListSelector.ItemsSource binding to the correct SoundGroup as well

### Recap

To recap, the big take away from this lesson was how to style the LongListSelector by changing the LayoutMode from List to Grid, and setting the tile sizes. We also learned how we can take things like DataTemplates and defining them at a higher level for ease of re-use. We also learn how to use an Image control and add simple shapes like Ellipses to enhance the visual design. When you build apps, you need to spend a lot of time tweaking the visual appearance and have an eye for detail.

# Part 14: Binding to Real Data at Runtime

Source Code: <http://aka.ms/absbeginnerdevwp8>

So, how far along are we with our SoundBoard app?

Well, we have a new data model in place, and in the previous lesson we added sample data that we used at DESIGN time to help us properly layout the app's user interface, particularly the DataTemplate that is bound to instances of the SoundData class.

Now, IN THIS LESSON, we want to turn our attention to binding to REAL data AT RUN TIME.

Truth be told, we could use this SAME XAML file for the "live data" at run time in our app. If you wanted to implement it that way, you certainly could and you've already got a head start to taking that approach ... just build out that XAML file with more instances of SoundGroup and SoundData, then load that file at RUNTIME in the LoadData() method of the SoundModel class.

In fact, that might be a great way to really challenge yourself ... after you finish with this series of lessons, you could go back and re-create this app but stop at this point and take a different data access approach. You only learn when you struggle, and an exercise like this will force you to struggle with how to load XAML data into our data model at runtime.

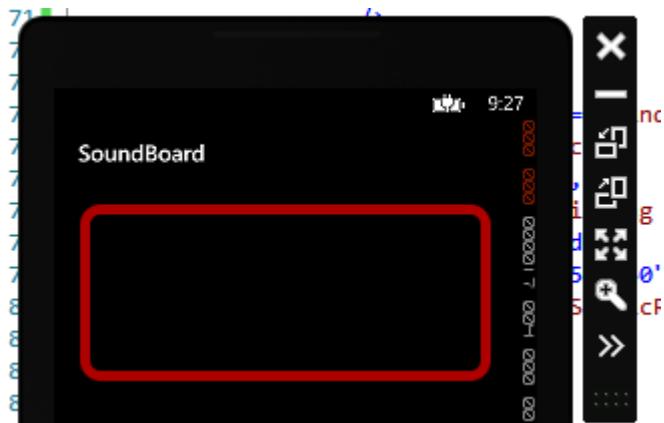
But I digress ...

Our game plan for this lesson:

1. In the SoundModel.cs file, we'll create a series of helper methods, each helper method designed to create instances of the SoundData class which will be added to the SoundGroup's Items collection. So we'll create a helper method called CreateAnimalsGroup() and CreateCartoonsGroup() and so on, each one of these helper methods will create instances of the SoundData class and add them to the proper SoundGroup's Items collection.
2. After we have all of our helper methods complete, we'll modify the LoadData() method and call each of those helper methods. So that, when we call LoadData(), real data will be available at runtime.

1. Adding real run-time data to our app

As you can see, right now were we to run the app:



... we're not loading any data at runtime. What we want to accomplish is to set each of the public properties of our SoundModel class, like Animals for example, to an instance of the SoundGroup object loaded with data. So, here's an example of the interaction I want to enable in my LoadData() method:

```
19     public void LoadData()
20     {
21         Animals = CreateAnimalsGroup();
22
23         IsDataLoaded = true;
24     }
```

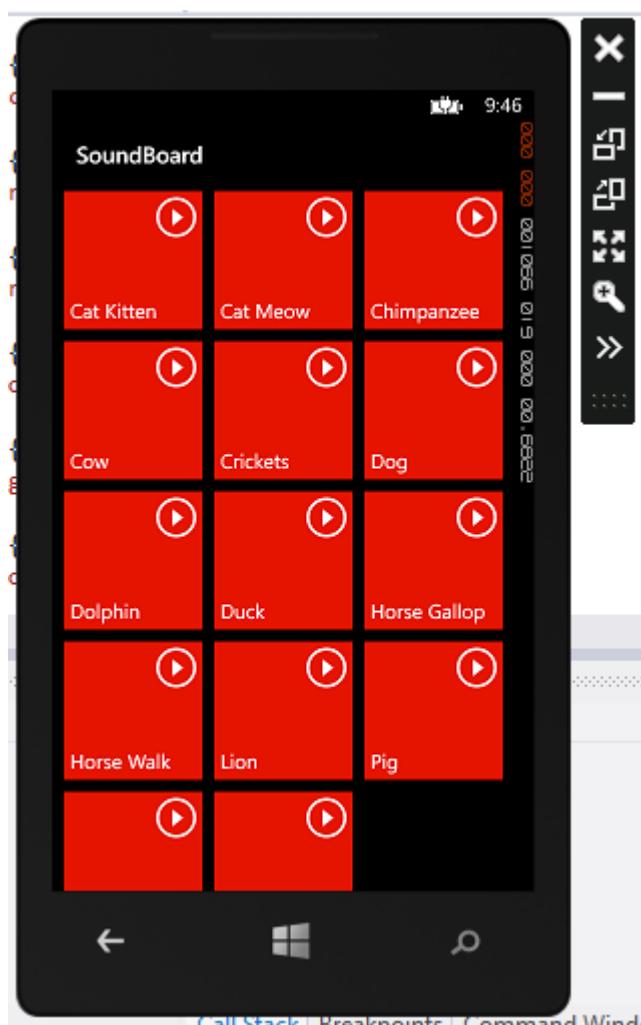


All that's left is to implement the CreateAnimalsGroup() helper method, like so:

```
25 1   private SoundGroup CreateAnimalsGroup()
26 {2
27     SoundGroup data = new SoundGroup();
28     data.Title = "animals";
29     string basePath = "assets/audio/animals/";
30
31     data.Items.Add(new SoundData { Title = "Cat Kitten",
32                     FilePath = basePath + "Cat Kitten.wav" });
33
34     data.Items.Add(new SoundData { Title = "Cat Meow",
35                     FilePath = basePath + "Cat Meow.wav" });
36
37     data.Items.Add(new SoundData { Title = "Chimpanzee",
38                     FilePath = basePath + "Chimpanzee.wav" });
39
40     data.Items.Add(new SoundData { Title = "Cow",
41                     FilePath = basePath + "Cow.wav" });
42
43     data.Items.Add(new SoundData { Title = "Crickets",
44                     FilePath = basePath + "Crickets.wav" });
45
46     data.Items.Add(new SoundData { Title = "Dog",
47                     FilePath = basePath + "Dog.wav" });
48
49     data.Items.Add(new SoundData { Title = "Dolphin",
50                     FilePath = basePath + "Dolphin.wav" });
51
52     data.Items.Add(new SoundData { Title = "Duck",
53                     FilePath = basePath + "Duck.wav" });
54
55     data.Items.Add(new SoundData { Title = "Horse Gallop",
56                     FilePath = basePath + "Horse Gallop.wav" });
57
58     data.Items.Add(new SoundData { Title = "Horse Walk",
59                     FilePath = basePath + "Horse Walk.wav" });
60
61     data.Items.Add(new SoundData { Title = "Lion",
62                     FilePath = basePath + "Lion.wav" });
63
64     data.Items.Add(new SoundData { Title = "Pig",
65                     FilePath = basePath + "Pig.wav" });
66
67     data.Items.Add(new SoundData { Title = "Rooster",
68                     FilePath = basePath + "Rooster.wav" });
69
70     data.Items.Add(new SoundData { Title = "Sheep",
71                     FilePath = basePath + "Sheep.wav" });
72
73         return data;
74     }
75 }
```

1. CreateAnimalsGroup will return an instance of SoundGroup.
2. We'll create an instance of SoundGroup, which we'll build throughout this helper method.
3. We'll set the Title property—this is what gets displayed as the Header property of the PivotItem.
4. Instead of typing out the full path to the audio files, we'll save these in a variable and append it as we're initializing the FilePath property for each new instance of SoundData.
5. Here we add a new instance of SoundData to the Items property (a List<SoundData>) of the SoundGroup and use object initializer syntax to populate the Title and FilePath properties of each.

When we run the app:



... we can now see all of the actual data in the Animals PivotItem.

But wait ... where is the Animals PivotItem Title? We'll have to fix that in a moment. Right now, let's finish adding the rest of the Create\_\_\_\_Group() helper methods.

Here's the listing for the CreateCartoonsGroup():

```
80     private SoundGroup CreateCartoonsGroup()
81     {
82         SoundGroup data = new SoundGroup();
83         data.Title = "cartoons";
84         string basePath = "assets/audio/cartoons/";
85
86         data.Items.Add(new SoundData { Title = "Boing",
87                         FilePath = basePath + "Boing.wav" });
88
89         data.Items.Add(new SoundData { Title = "Bronk",
90                         FilePath = basePath + "Bronk.wav" });
91
92         data.Items.Add(new SoundData { Title = "Bugle charge",
93                         FilePath = basePath + "Bugle charge.wav" });
94
95         data.Items.Add(new SoundData { Title = "Laser",
96                         FilePath = basePath + "Laser.wav" });
97
98         data.Items.Add(new SoundData { Title = "Out Here",
99                         FilePath = basePath + "Out Here.wav" });
100
101        data.Items.Add(new SoundData { Title = "Splat",
102                         FilePath = basePath + "Splat.wav" });
103
104        return data;
105    }
```

Here's the listing for the CreateTauntsGroup():

```
107     private SoundGroup CreateTauntsGroup()
108     {
109         SoundGroup data = new SoundGroup();
110         data.Title = "taunts";
111         string basePath = "assets/audio/taunts/";
112
113         data.Items.Add(new SoundData { Title = "Cackle",
114             FilePath = basePath + "Cackle.wav" });
115
116         data.Items.Add(new SoundData { Title = "Clock Ticking",
117             FilePath = basePath + "Clock Ticking.wav" });
118
119         data.Items.Add(new SoundData { Title = "Dial up",
120             FilePath = basePath + "Dial up.wav" });
121
122         data.Items.Add(new SoundData { Title = "Drum roll",
123             FilePath = basePath + "Drum roll.wav" });
124
125         data.Items.Add(new SoundData { Title = "Elevator Music",
126             FilePath = basePath + "Elevator Music.wav" });
127
128         data.Items.Add(new SoundData { Title = "Laugh",
129             FilePath = basePath + "Laugh.wav" });
130
131         data.Items.Add(new SoundData { Title = "Laugh - Evil",
132             FilePath = basePath + "Laugh - Evil.wav" });
133
134         data.Items.Add(new SoundData { Title = "Wrong Price",
135             FilePath = basePath + "Wrong Price.wav" });
136
137         data.Items.Add(new SoundData { Title = "Sad Trombone",
138             FilePath = basePath + "Sad Trombone.wav" });
139
140         data.Items.Add(new SoundData { Title = "Sarcastic Ooo",
141             FilePath = basePath + "Sarcastic Ooo.wav" });
142
143         data.Items.Add(new SoundData { Title = "Sigh",
144             FilePath = basePath + "Sigh.wav" });
145
146         data.Items.Add(new SoundData { Title = "Snore",
147             FilePath = basePath + "Snore.wav" });
148
149         data.Items.Add(new SoundData { Title = "Yawn",
150             FilePath = basePath + "Yawn.wav" });
151
152         return data;
153     }
154 }
```

Here's the listing for the CreateWarningsGroup():

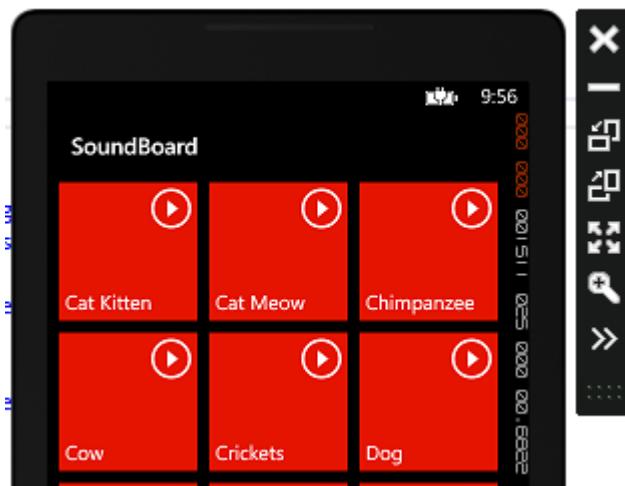
```
154
155     private SoundGroup CreateWarningsGroup()
156     {
157         SoundGroup data = new SoundGroup();
158         data.Title = "warnings";
159         string basePath = "assets/audio/warnings/";
160
161         data.Items.Add(new SoundData { Title = "Air horn",
162                         FilePath = basePath + "Air horn.wav" });
163
164         data.Items.Add(new SoundData { Title = "Air Raid",
165                         FilePath = basePath + "Air Raid.wav" });
166
167         data.Items.Add(new SoundData { Title = "Alarm Clock - Electric",
168                         FilePath = basePath + "Alarm Clock - Electric.wav" });
169
170         data.Items.Add(new SoundData { Title = "Alarm Clock - Bell",
171                         FilePath = basePath + "Alarm Clock - Bell.wav" });
172
173         data.Items.Add(new SoundData { Title = "Backing up",
174                         FilePath = basePath + "Backing up.wav" });
175
176         data.Items.Add(new SoundData { Title = "Bell - Church",
177                         FilePath = basePath + "Bell - Church.wav" });
178
179         data.Items.Add(new SoundData { Title = "Bell - School",
180                         FilePath = basePath + "Bell - School.wav" });
181
182         data.Items.Add(new SoundData { Title = "Fog horn",
183                         FilePath = basePath + "Fog horn.wav" });
184
185         data.Items.Add(new SoundData { Title = "Glass breaking",
186                         FilePath = basePath + "Glass breaking.wav" });
187
188         data.Items.Add(new SoundData { Title = "Missle alert",
189                         FilePath = basePath + "Missle alert.wav" });
190
191         data.Items.Add(new SoundData { Title = "Police - UK",
192                         FilePath = basePath + "Police - UK.wav" });
193
194         data.Items.Add(new SoundData { Title = "Police - US",
195                         FilePath = basePath + "Police - US.wav" });
196
197         data.Items.Add(new SoundData { Title = "Vuvuzela",
198                         FilePath = basePath + "Vuvuzela.wav" });
199
200         return data;
201     }
202 }
```

And now we'll use those helper methods in our LoadData() method to populate the associated Property of each:

```
19     public void LoadData()
20     {
21         Animals = CreateAnimalsGroup();
22         Cartoons = CreateCartoonsGroup();
23         Taunts = CreateTauntsGroup();
24         Warnings = CreateWarningsGroup();
25
26         IsDataLoaded = true;
27     }
28 }
```

## 2. Fixing a data binding problem with the PivotItem Header

If I attempt to run the app, I can't look at the other PivotItems to see the data because we're missing the PivotItem Header:



My first reaction is that it's a binding issue—that the data is not being loaded correctly. I start by looking at the OnNavigatedTo() event handler. Clearly, LoadData() is being called here:

```
--  
12  namespace SoundBoard  
13  {  
14      public partial class MainPage : PhoneApplicationPage  
15      {  
16          // Constructor  
17          public MainPage()...  
18  
19          // Load data for the ViewModel Items  
20          protected override void OnNavigatedTo(NavigationEventArgs e)  
21          {  
22              if (!App.ViewModel.IsDataLoaded)  
23              {  
24                  App.ViewModel.LoadData();  
25              }  
26          }  
27  
28      }  
--
```

Next, I look at the App.xaml.cs file. In the constructor, if the viewModel is null, it will create a new instance of the SoundModel:

```
--  
12  namespace SoundBoard  
13  {  
14      public partial class App : Application  
15      {  
16          private static SoundModel viewModel = null;  
17  
18          public static SoundModel ViewModel  
19          {  
20              get  
21              {  
22                  if (viewModel == null)  
23                      viewModel = new SoundModel();  
24  
25                  return viewModel;  
26              }  
27          }  
28      }
```

I suspect that this is a timing issue. The Pivot must be requesting the App.ViewModel when it's empty, then the DataTemplate is requesting the App.ViewModel after it has been filled.

After staring at this for a few moments, I set breakpoints on the MainPage.xaml.cs:

```

16 |         // Constructor
17 |     public MainPage()
18 |     {
19 |         InitializeComponent();
20 |
21 |         // Set the data context of the listbox control to the sample data
22 |         DataContext = App.ViewModel;
23 |
24 |         // Sample code to localize the ApplicationBar
25 |         //BuildLocalizedApplicationBar();
26 |     }
27 |
28 |     // Load data for the ViewModel Items
29 |     protected override void OnNavigatedTo(NavigationEventArgs e)
30 |     {
31 |         if (!App.ViewModel.IsDataLoaded)
32 |         {
33 |             App.ViewModel.LoadData();
34 |         }
35 |     }

```

Debugging the app reveals that the DataContext is being set prior to calling the LoadData(). This means the PivotItem titles are binding right away, however the PivotItems ItemTemplates / DataTemplates are only binding AFTER we call LoadData():

```

21 |         // Set the data context of the listbox control to the sample data
22 |         DataContext = App.ViewModel;
23 |         // Sample code to local
24 |         //BuildLocalizedApplicationBar();
25 |     }
26 |
27 |     // Load data for the ViewModel Items
28 |     protected override void OnNavigatedTo(NavigationEventArgs e)
29 |     {
30 |

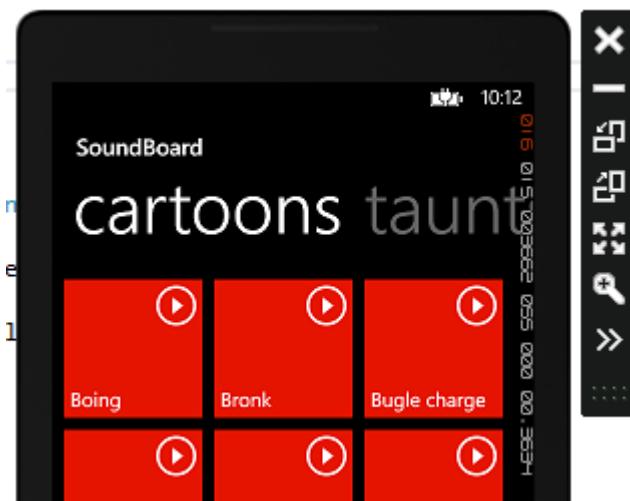
```

This is how we see part of the data, but not the other part. I determine that the remedy is simple—we'll call LoadData() right after we create a new instance of the SoundModel:

```
18     public static SoundModel ViewModel
19     {
20         get
21         {
22             if (viewModel == null)
23             {
24                 viewModel = new SoundModel();
25                 viewModel.LoadData();
26             }
27
28             return viewModel;
29         }
30     }
```



By adding the explicit call to LoadData() in the App.xaml.cs, we can re-run the app:



... and our PivotItem Titles re-appear and we can navigate to each new category to see the data we've loaded into each.

## Recap

To recap, the big take away in this lesson is how we implemented the real data. While there are definitely different approaches we could have taken, we chose to create helper methods containing hard coded instances of our SoundData and SoundGroup

classes in C#. We also saw how to reason our way through an odd timing issue with data binding ... debugging and understanding the order of events is a valuable skill.

Don't forget my challenge at the outset of this lesson ... I hereby challenge you to re-create this app using a different data access technique, such as expanding the SampleData.xaml file with real data, then loading that data at RUN TIME. Can you figure that out? I'll bet if you spend a day working on it, you'll have it working without my help. You might learn more from that challenge than the rest of this series because you only truly learn when you challenge yourself.

# Part 15: Playing a Sound when a ListItem is Selected

Source Code: <http://aka.ms/absbeginnerdevwp8>

Now that we have our data model loaded with live data, not the least of which is the path to the sound files, actually PLAYING those sounds is relatively easy.

Our game plan:

1. We'll add a MediaElement control to the page ... we'll not set the Source declaratively, rather we'll set it programmatically
2. We'll wire up an event handler that will be triggered each time the user taps a different tile
3. In the body of that event handler method, we'll set the Source property of the MediaElement control

1. Add a MediaElement to the MainPage.xaml

Since we need the MediaElement on each PivotItem, I'll add the control near the top of the code, beneath the LayoutRoot Grid control:

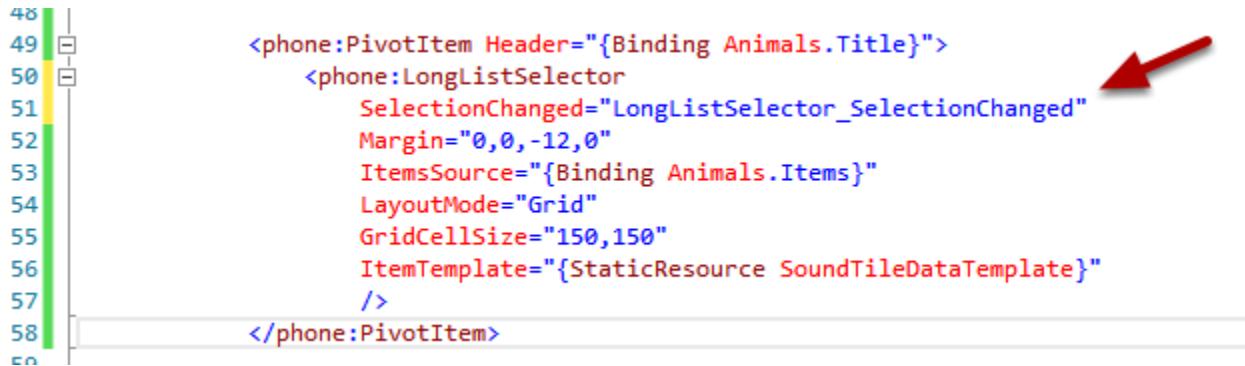
```
37
38    <!--LayoutRoot is the root grid where all page content is placed-->
39    <Grid x:Name="LayoutRoot" Background="Transparent">
40
41        <MediaElement
42            Name="AudioPlayer"
43            Volume="1" />
44
45    <!--Pivot Control-->
```

Notice in line 42 I give the MediaElement a name because I plan on accessing it programmatically in C#, and I set the Volume to 1, the loudest setting possible.

2. Handle the LongListSelector\_SelectionChanged event

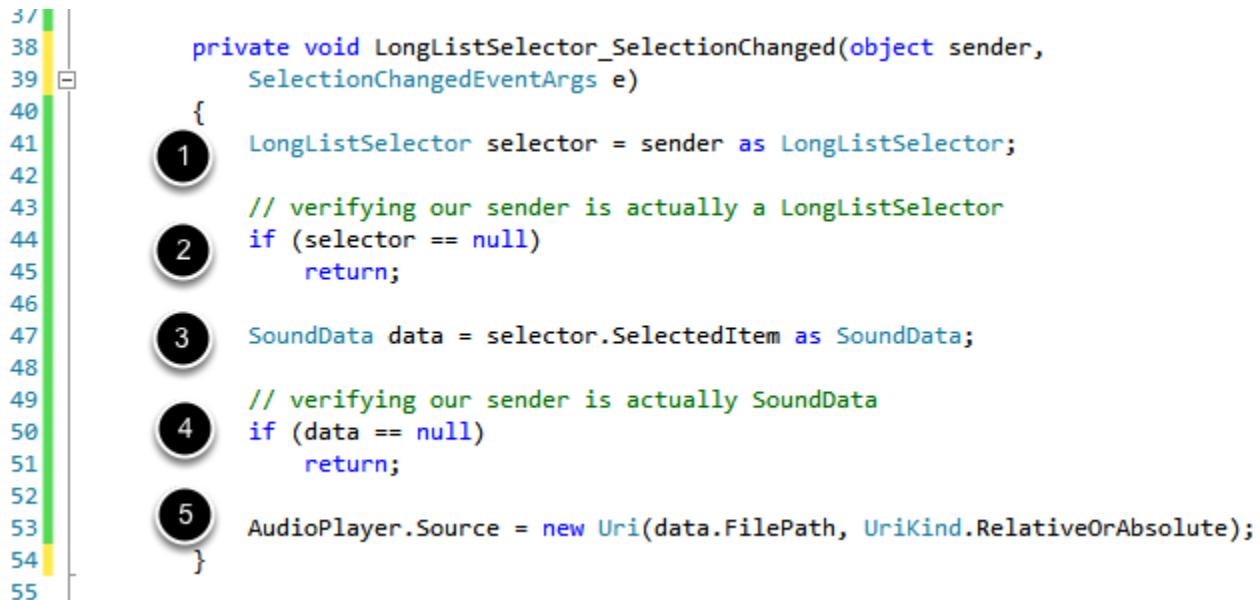
Next, I add an attribute to the LongListSelector that will be triggered each time a tile is tapped. I use the technique I demonstrated in an earlier video to allow Visual Studio to create the event handler method's name and create the event handler method stub.

(Hint: type SelectionChanged=" and when the contextual dialog appears select the Enter key to auto implement the rest of the code). It should look like this (see line 51):



```
48
49     <phone:PivotItem Header="{Binding Animals.Title}">
50         <phone:LongListSelector
51             SelectionChanged="LongListSelector_SelectionChanged"
52             Margin="0,0,-12,0"
53             ItemsSource="{Binding Animals.Items}"
54             LayoutMode="Grid"
55             GridCellSize="150,150"
56             ItemTemplate="{StaticResource SoundTileDataTemplate}"
57         />
58     </phone:PivotItem>
```

Navigate to the event handler method in the code behind and implement the code to play the sound associated with the tile:



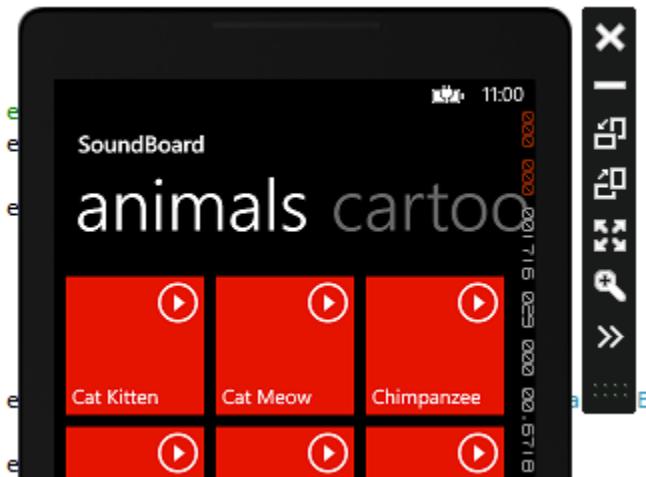
```
3/
38     private void LongListSelector_SelectionChanged(object sender,
39             SelectionChangedEventArgs e)
40     {
41         1     LongListSelector selector = sender as LongListSelector;
42
43         // verifying our sender is actually a LongListSelector
44         2     if (selector == null)
45             return;
46
47         3     SoundData data = selector.SelectedItem as SoundData;
48
49         // verifying our sender is actually SoundData
50         4     if (data == null)
51             return;
52
53         5     AudioPlayer.Source = new Uri(data.FilePath, UriKind.RelativeOrAbsolute);
54     }
55 }
```

1. Here we get a reference to the LongListSelector. Since we have five LongListSelectors defined on our page (because we have five references to the DataTemplate in the Page's Resources) we need to access the one that the user clicked on (or more correctly, find the LongListSelector whose list item was clicked on). Notice the "as LongListSelector" on the end of that line. This is a form of data type conversion ...

sender is object, but we know it must be a LongListSelector. the "as" keyword used here will convert sender to the type LongListSelector.

2. We want to ensure that the selector is not null. While not probable, it is entirely possible for something to have gone awry and some other type was passed in as sender. We're guarding ourselves against that possibly by programming defensively here.
3. Likewise, we want to access the list item that was selected, so we access the selector's SelectItem and attempt to convert that into an object of type SoundData
4. Again, we want to ensure that the data object is not null. While not probably, it's possible for something to have gone awry and some other type was retrieved via the SelectedItem property. By checking for null we're guarding ourselves against this possibility by programming defensively
5. Now that we have the SoundData that was selected, we set the Source property of the MediaElement.

Let's run to test the app:



This seems to work fine at first, but after repeated testing I realize we have a problem. If we tap the same tile twice, I don't get a sound. That's because the selection didn't change and therefore our event handler is not triggered.

### 3. Fixing select selection problem

To fix this, we need to reset the SelectedItem to null after we've finished playing the sound:

```
37
38     private void LongListSelector_SelectionChanged(object sender,
39             SelectionChangedEventArgs e)
40     {
41         LongListSelector selector = sender as LongListSelector;
42
43         // verifying our sender is actually a LongListSelector
44         if (selector == null)
45             return;
46
47         SoundData data = selector.SelectedItem as SoundData;
48
49         // verifying our sender is actually SoundData
50         if (data == null)
51             return;
52
53         AudioPlayer.Source = new Uri(data.FilePath, UriKind.RelativeOrAbsolute);
54
55         // resetting selected so we can play the same sound over and over again
56         selector.SelectedItem = null;
57     }

```



Re-running and testing the app proves our solution to have worked.

## Recap

To recap, the big take away was how to programmatically work with the MediaPlayer element by setting the Source property to dynamically change the sound that is played. We learned about the "as" keyword which performs conversions on compatible reference types. In our case, we needed to cast the input property "object sender" to the LongListSelector and the SelectedItem to SoundData, the type our DataTemplate was bound to. And finally we learned how to unselect a selected item in a LongListSelector by setting the SelectedItem property to null.

# Part 16: Working with the Application Bar

Source Code: <http://aka.ms/absbeginnerdevwp8>

Our app works great as a simple SoundBoard, but we want to push the envelop and enable custom sounds—sounds that the user can record and re-use. This will require a few changes to our app ... in this lesson, we'll add an Application Bar with a Record button. When the user clicks it, we'll navigate the user to a new XAML page where they can record new custom sounds.

You've probably seen an application bar before, even if you didn't realize what it's name was. Application bars appear at the bottom of the viewable area for your app and can contain a number of icons as well as an ellipsis which, when tapped, reveals text below the icon, and possibly a menu of additional options. We'll see it at work before the end of this lesson.

Here's our game plan for this lesson:

1. The project template already creates some boiler plate code for an application bar. We'll un-comment out their code and revise it to display an application bar with a record button and a menu option.
2. We'll work with the AppResources.resx so that the text on our new application bar can be localized in the future.
3. We'll clean up some unused files ... the old data model should go.
4. We'll wire up and create event handler method stubs for the application bar button and menu option. In subsequent lessons, we'll flesh out the functionality in the event handler methods.

1. Enable the boilerplate BuildLocalizedApplicationBar() method

In the MainPage.xaml.cs, the MainPage() constructor has a commented out line of code calling the BuildLocalizedApplicationBar(). I'll un-comment that line of code out:

```
17           // Constructor
18   public MainPage()
19   {
20       InitializeComponent();
21
22       // Set the data context of the listbox control to the sample data
23       DataContext = App.ViewModel;
24
25       BuildLocalizedApplicationBar(); // Un-commented line
26   }
27
```

... and then un-comment out the actual method further down on the code page:

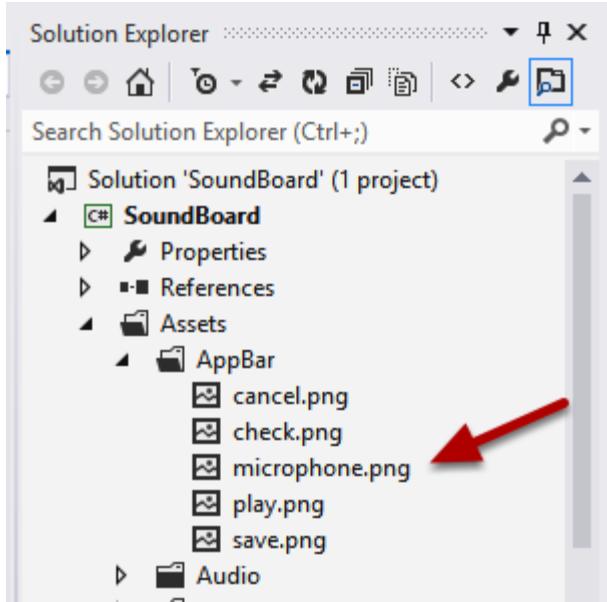
```
57
58     // Sample code for building a localized ApplicationBar
59     private void BuildLocalizedApplicationBar()
60     {
61         // Set the page's ApplicationBar to a new instance of ApplicationBar.
62         ApplicationBar = new ApplicationBar();
63
64         // Create a new button and set the text value to the localized string from AppResources.
65         ApplicationBarIconButton appBarButton =
66             new ApplicationBarIconButton(
67                 new Uri("/Assets/AppBar/appbar.add.rest.png", UriKind.Relative));
68
69         appBarButton.Text = AppResources.AppBarButtonText;
70         ApplicationBar.Buttons.Add(appBarButton);
71
72         // Create a new menu item with the localized string from AppResources.
73         ApplicationBarMenuItem appBarMenuItem =
74             new ApplicationBarMenuItem(AppResources.AppBarMenuItemText);
75
76         ApplicationBar.MenuItems.Add(appBarMenuItem);
77     }

```

By un-commenting out these lines of code, we've added a simple application bar to our app.

## 2. Modifying our Application Bar Button and Text

Obviously, we'll want to change the image that is referenced in line 67 (above) to reference the microphone.png file in the Assets\AppBar folder:

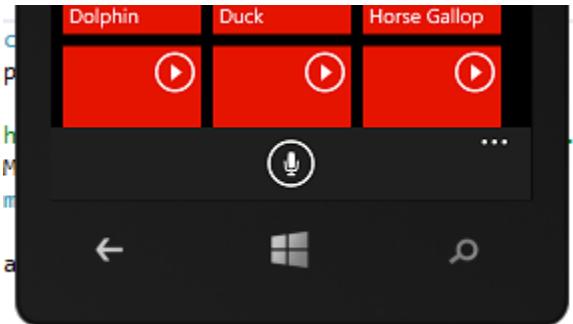


So, I'll make this change:

```
58 // Sample code for building a localized ApplicationBar
59 private void BuildLocalizedApplicationBar()
60 {
61     // Set the page's ApplicationBar to a new instance of ApplicationBar.
62     ApplicationBar = new ApplicationBar();
63
64     // Create a new button and set the text value to the localized string from AppReso
65     ApplicationBarIconButton appBarButton =
66         new ApplicationBarIconButton(
67             new Uri("/Assets/AppBar/microphone.png", UriKind.Relative));
```

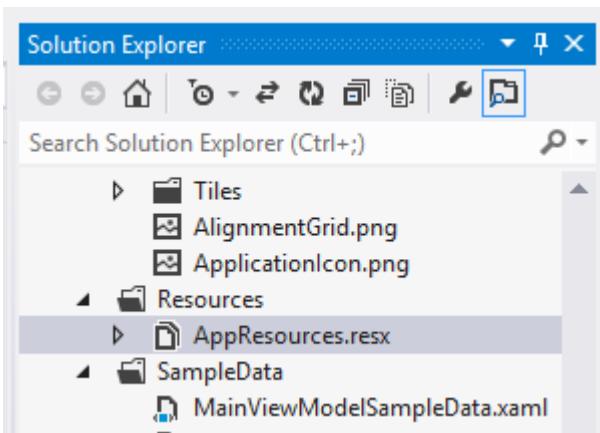
A screenshot of a code editor showing C# code. The code defines a method 'BuildLocalizedApplicationBar' that creates an 'ApplicationBar' and adds an 'ApplicationBarIconButton' to it, pointing to the 'microphone.png' file in the 'Assets/AppBar' folder. A red arrow points to the 'new Uri("/Assets/AppBar/microphone.png", UriKind.Relative)' line.

... and I'll run the app. This reveals the app bar with the microphone icon and the ellipsis which, when tapped, reveals a menu option titled "Menu Option":



Below the microphone icon, the word "add" appears. Let's change both of these.

First, the title of our method, "BuildLocalizedAppBar", suggests that this Application Bar retrieves its textual values from the AppResources.resx, so let's open that file:



... and make the following changes:

	Name	Value
1	AppBarAbout	About
2	AppBarRecord	Record
	ApplicationTitle	SoundBoard
	ResourceFlowDirection	LeftToRight
	ResourceLanguage	en-US
3	AppBarSave	Save

1. I add a property named AppBarAbout and set its value to About
2. I add a property named AppBarRecord and set its value to Record
3. I add a property to AppBarSave and set its value to Save

Also, I remove any properties I no longer need, including:

- AppBarButtonText
- AppBarMenuItemText
- SampleProperty

... and I save this file.

Next, I'll need to re-write some of the BuildLocalizedApplicationBar() to utilize the settings added in the AppResources.resx file:

```

58
59     private void BuildLocalizedAppBar()
60     {
61         ApplicationBar = new ApplicationBar();
62
63         AppBarIconButton recordAudioAppBar =
64             new AppBarIconButton();
65         recordAudioAppBar.IconUri =
66             new Uri("/Assets/AppBar/microphone.png", UriKind.Relative);
67         recordAudioAppBar.Text = AppResources.AppBarRecord;
68
69         AppBarMenuItem aboutAppBar =
70             new AppBarMenuItem();
71         aboutAppBar.Text = AppResources.AppBarAbout;
72
73         ApplicationBar.Buttons.Add(recordAudioAppBar);
74         ApplicationBar.MenuItems.Add(aboutAppBar);
75     }

```

1

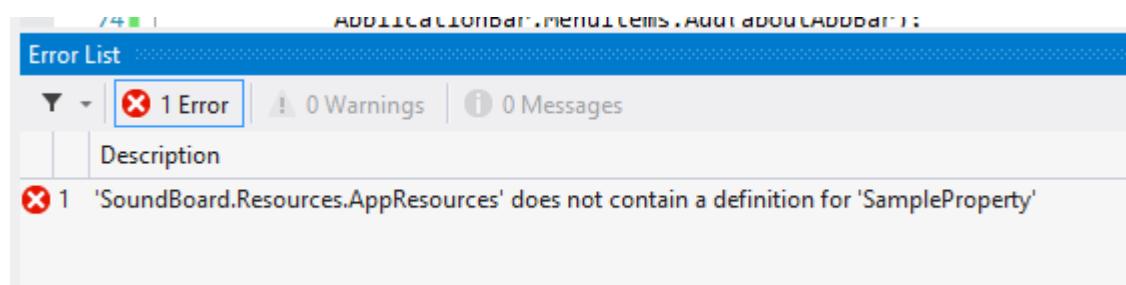
2

3

1. I re-write the creation of the "Record" button
2. I re-write the creation of the "About" menu option
3. I add them both to the Application Bar

### 3. Removing the old data model

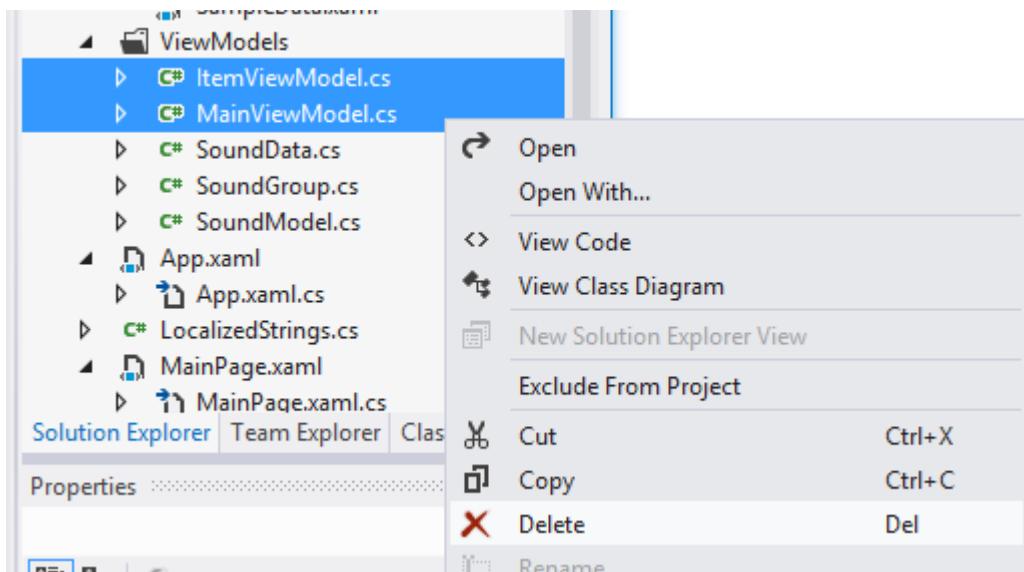
When I attempt to run the app, I see that I may have been too hasty in deleting the "SampleProperty" from the AppResources.resx:



I find that the source of the problem is code that I don't even use anymore ... i.e., the OLD data model code:

```
>MainViewModel.cs X MainPage.xaml.cs MainPage.xaml SampleData.xaml
SoundBoard.ViewModels.MainViewModel
42     /// Sample property that returns a localized string
43     /// </summary>
44     public string LocalizedSampleProperty
45     {
46         get
47         {
48             return AppResources.SampleProperty;
49         }
50     }
51
```

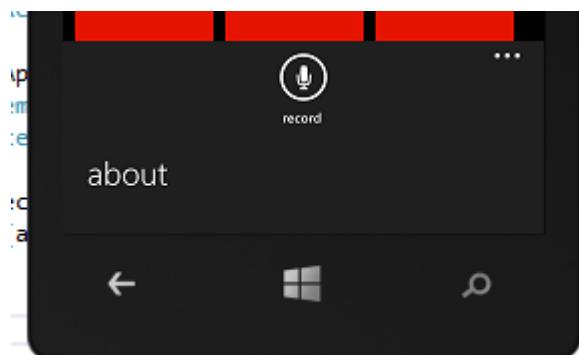
I can kill two birds with one stone by deleting the old data model classes. I select ItemViewModel.cs and MainViewModel.cs, right-click, select Delete from the context menu. That cleans up the code base removing unnecessary code and removes the errant reference to the nonexistent entry in the AppResources.resx:



One of my favorite authors says that you stay organized to stay productive. Keep your work area clean, like a sushi chef. Though I comment out code I don't think I'll need anymore, I also want to comb through the code once a day to keep the codebase pristine. When I open someone else's code and I look at it, I automatically assume it's all being used. Sometimes, however, there are classes or methods no longer called. This adds friction and confusion. When I'm not recording videos and actually writing

code, I rely on a third-party tool called ReSharper ... it will analyze your code and find unused classes and methods, it can locate duplicate code and recommend changes and a thousand other improvements to your code. I highly recommend it.

At any rate, when we run the app this time we can see that the app bar looks has the text we would expect:

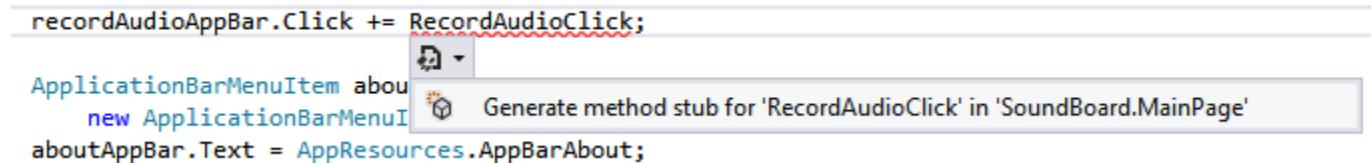


#### 4. Responding to the Click event of the App Bar's Record button

Finally in this lesson, we want to wire up the click event handler methods to our new button and menu items:

```
58
59     private void BuildLocalizedAppBar()
60     {
61         ApplicationBar = new ApplicationBar();
62
63         ApplicationBarIconButton recordAudioAppBar =
64             new ApplicationBarIconButton();
65         recordAudioAppBar.IconUri =
66             new Uri("/Assets/AppBar/microphone.png", UriKind.Relative);
67         recordAudioAppBar.Text = AppResources.AppBarRecord;
68
69         recordAudioAppBar.Click += RecordAudioClick; ←
70
71         ApplicationBarMenuItem aboutAppBar =
72             new ApplicationBarMenuItem();
73         aboutAppBar.Text = AppResources.AppBarAbout;
74
75         ApplicationBar.Buttons.Add(recordAudioAppBar);
76         ApplicationBar.MenuItems.Add(aboutAppBar);
77     }
```

I add line 69 which, as we learned in the lesson on Events, will add the RecordAudioClick method in its list of methods that will be triggered by the click event of the Record button. To create a method stub, use the technique we learned in the Event lesson—hover your mouse cursor over the blue dash under the letter R in RecordAudioClick to reveal a menu, and choose the "Generate method stub ..." menu option:



This will create a method stub for the event handler, complete with a reminder that the method is not yet implemented by throwing an exception:

```
78
79     private void RecordAudioClick(object sender, EventArgs e)
80     {
81         throw new NotImplementedException();
82     }
```

We'll repeat this process for the "about" menu option. In line 75 (below) I associate a method event handler called AboutClick to the click event of the menu option:

```
70
71     ApplicationBarMenuItem aboutAppBar =
72         new ApplicationBarMenuItem();
73     aboutAppBar.Text = AppResources.AppBarAbout;
74
75     aboutAppBar.Click += AboutClick; ← Red arrow points here
76
77     ApplicationBar.Buttons.Add(recordAudioAppBar);
78     ApplicationBar.MenuItems.Add(recordAudioAppBar);
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 private void RecordAudioClick(object sender, EventArgs e)
87 {
88     throw new NotImplementedException();
89 }
90 }
```

... and I use the technique described earlier to create a method stub for RecordAudioClick() (lines 86 through 89).

## Recap

To recap, the big take away is how to create a new application bar, application bar icon, and application bar menu option. We used templated code and the AppResources.resx and wired up event handler method stubs, which we'll fully implement out in the coming lessons.

# Part 17: Introducing the Coding4Fun Toolkit

Source Code: <http://aka.ms/absbeginnerdevwp8>

Recording a custom sound in our app would be difficult to code by hand. Fortunately, your friends at Coding4Fun, and Clint Rutkas specifically, have created a Toolkit for Windows Phone that hides the complexity behind easy to use programmatic interfaces. If you're not familiar with Coding4Fun, it's a Channel9 site that engages in geeky cool projects, not your standard business apps ... they usually use hardware like the Kinect as the interface for controlling things such as boxing robots and even a Ford Mustang that was heavily modified with Microsoft technology ... just to prove that it could be done.

<http://channel9.msdn.com/coding4fun>

Clint is usually in the center of these projects and he is the author and curator of the Coding4Fun Toolkit.

By using this package in our project, we get some additional tools we can utilize in our apps such as a little "About This App" message box (the AboutPrompt) suited for Windows Phone development. In this lesson, I'll demonstrate how to install the Coding4Fun Toolkit package into our app using NuGet. Once we install it, we'll use it to implement the AboutPrompt.

In this short lesson, our game plan is to:

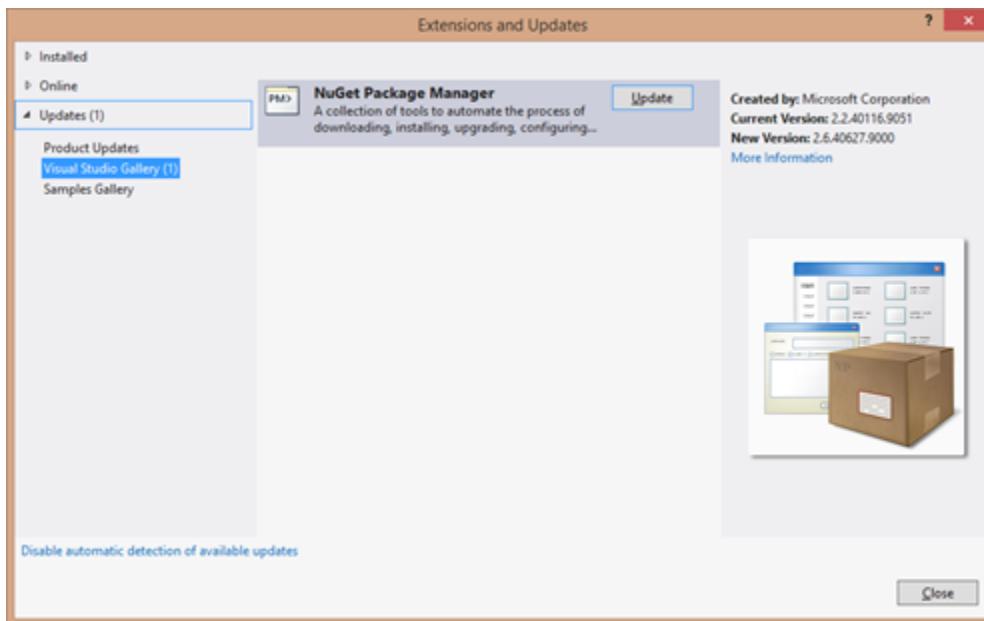
1. Use NuGet to install the Coding4Fun Toolkit package into our solution.
2. Snoop around and see what it added.
3. Add an AboutPrompt to our app.

**Update to series:** Being sure we have the newest version of NuGet.

With a fresh install of Visual Studio 2012, we'll need to update NuGet to get some of the newer packages such as the Coding4Fun Toolkit. It is a super easy process and can be done in a few clicks.

1. Go to the Tools Menu → Extensions and Updates

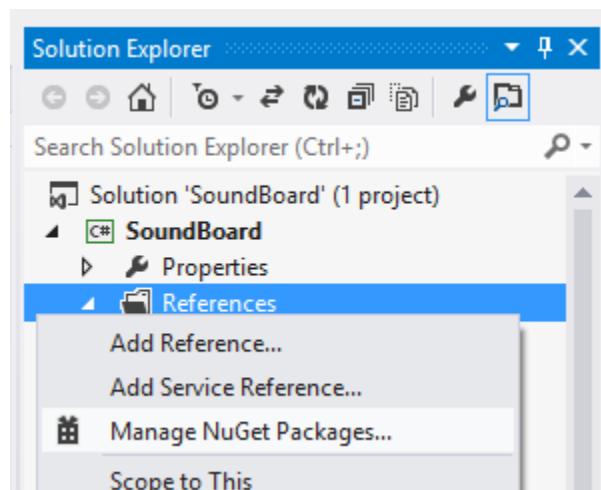
2. Go to the Update Tab → Visual Studio Gallery



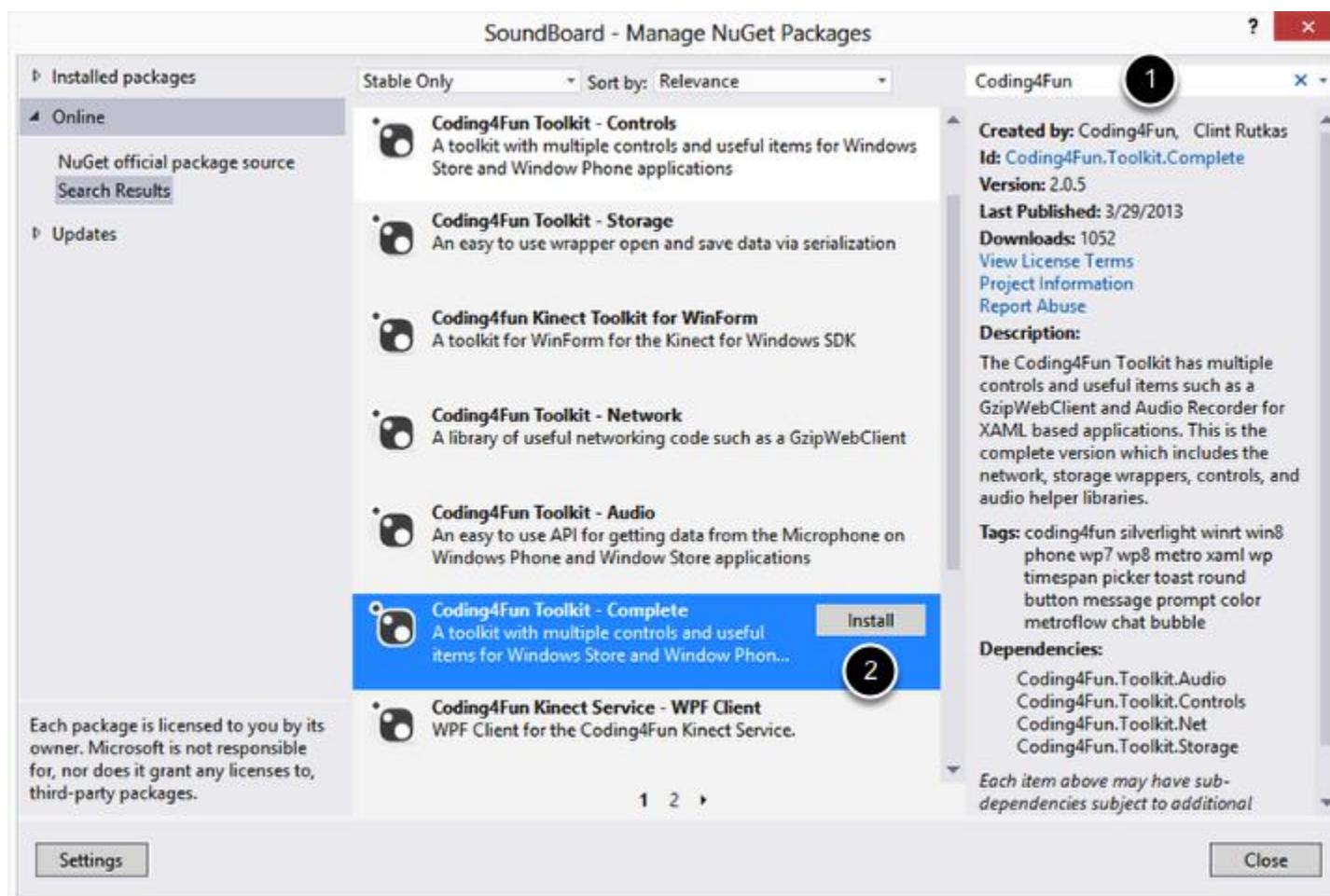
3. Click Update
4. Restart Visual Studio

1. Install the Coding4Fun Package

There are many ways to get to the NuGet package manager ... this time we'll right-click on the References folder and select "Manage NuGet Packages ...":



This will open up the Manage NuGet Packages Dialog:



1. In the Search box, search for: "Coding4Fun".
2. Click the Install button next to the "Coding4Fun Toolkit - Complete" package ... Clint split up the Coding4Fun toolkit into sub-packages so that developers can just pick and choose the parts they want in their project. However, for simplicity—and because we'll use a few different parts and want to familiarize ourselves with the package, we'll choose to install the Complete version.

After a few moments, you'll see green checkmarks next to the packages that were installed:

SoundBoard - Manage NuGet Packages

Installed packages

Online

NuGet official package source

Search Results

Updates

Stable Only Sort by: Relevance

Coding4Fun

Created by: Coding4Fun, Clint Rutkas  
Id: Coding4Fun.Toolkit.Complete  
Version: 2.0.5  
Last Published: 3/29/2013  
Downloads: 1052  
[View License Terms](#)  
[Project Information](#)  
[Report Abuse](#)

Description:

The Coding4Fun Toolkit has multiple controls and useful items such as a GzipWebClient and Audio Recorder for XAML based applications. This is the complete version which includes the network, storage wrappers, controls, and audio helper libraries.

Tags: coding4fun silverlight winrt win8 phone wp7 wp8 metro xaml wp timespan picker toast round button message prompt color metroflow chat bubble

Dependencies:

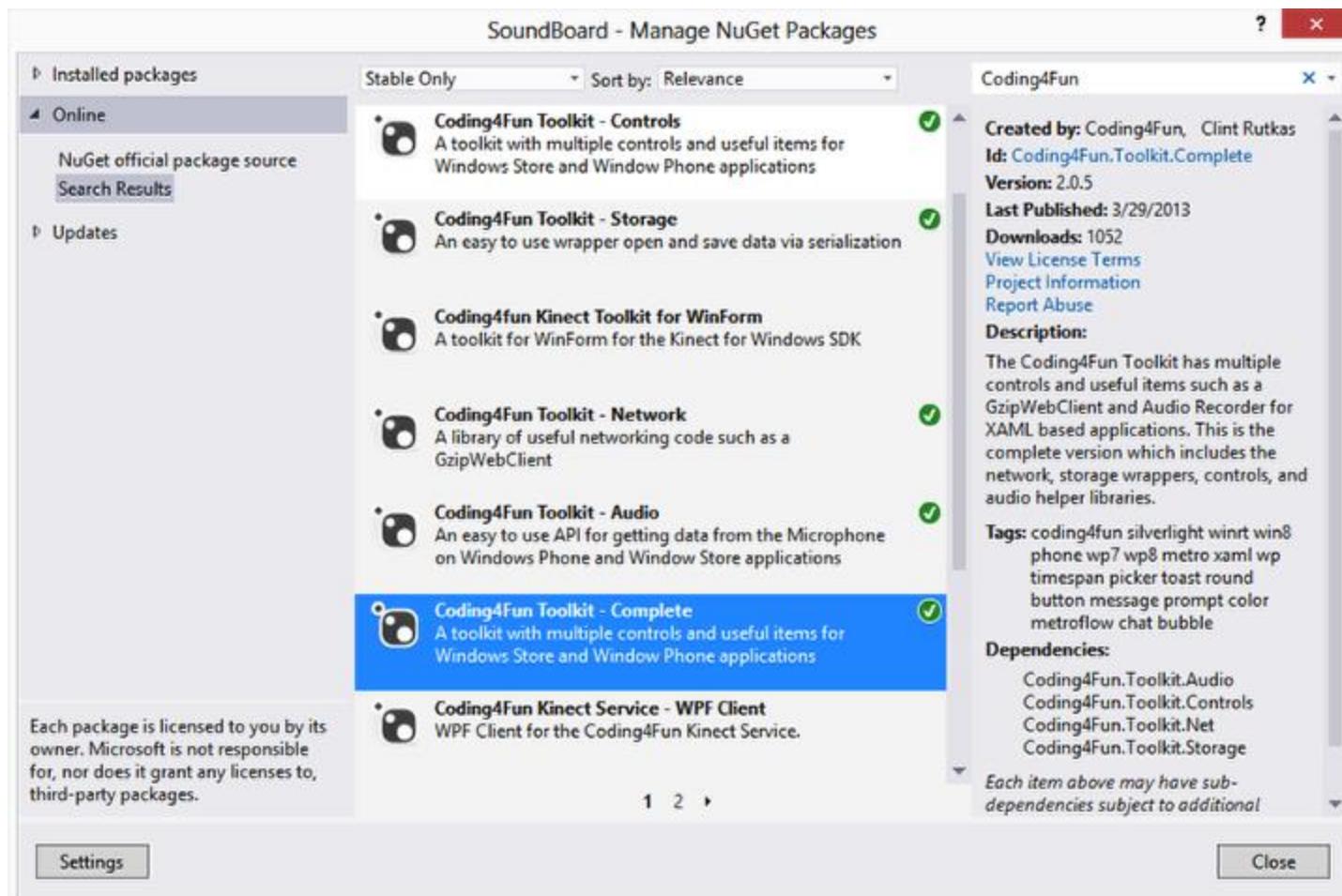
Coding4Fun.Toolkit.Audio  
Coding4Fun.Toolkit.Controls  
Coding4Fun.Toolkit.Net  
Coding4Fun.Toolkit.Storage

Each item above may have sub-dependencies subject to additional

Settings Close

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

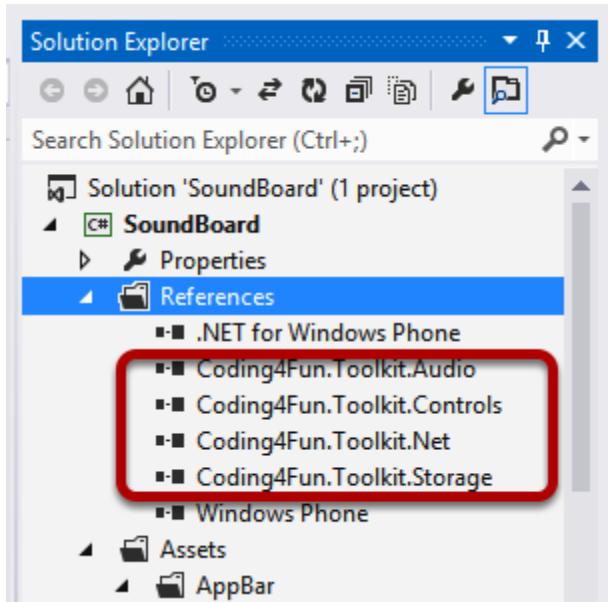
1 2 >



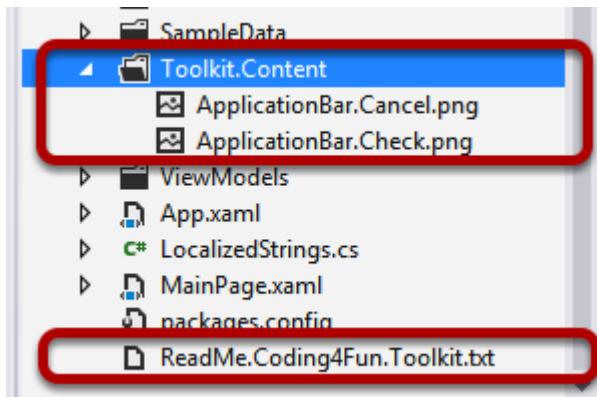
Detailed description: The screenshot shows the 'Manage NuGet Packages' dialog for the SoundBoard project. On the left, there's a sidebar with 'Installed packages' and 'Online' sections, and a search bar. The main area lists several NuGet packages from the 'Coding4Fun' source. The 'Coding4Fun Toolkit - Complete' package is highlighted with a blue background and a green checkmark. To its right, detailed information is shown: developer (Coding4Fun, Clint Rutkas), ID (Coding4Fun.Toolkit.Complete), version (2.0.5), last published date (3/29/2013), download count (1052), and links to view license terms, project information, and report abuse. Below this, a 'Description' section explains the toolkit's features, mentioning controls like GzipWebClient and Audio Recorder for XAML apps. A 'Tags' section lists various technologies. Under 'Dependencies', it lists sub-dependencies like Coding4Fun.Toolkit.Audio and Coding4Fun.Toolkit.Controls. At the bottom, a note states that each item may have sub-dependencies. The bottom right has 'Settings' and 'Close' buttons. A note at the bottom left about package licensing is also present.

... and you can click the Close button that in dialog to close it.

Now, in the Solution Explorer, you can see new References that were added:



... also, a new folder called Toolkit.Content and a ReadMe.Coding4Fun.Toolkit.txt file were added:



Opening the ReadMe.Coding4Fun.Toolkit.txt file, we can see versioning information (what changed) and migration notes:

```
ReadMe.Coding4Fun.Toolkit.txt ✘ × MainPage.xaml.cs SampleData.xaml App.xaml.cs WMAppManifest.xml
| Thanks for using the Coding4Fun Toolkit!

Migration Notes:
2.0.2 -> 2.X
Slider
Why:
    Keep in sync with WinStore and WinPhone 8 Slider control
ToDo:
    Fill property -> Foreground property
    Step property -> StepFrequency property

1.6 -> 2.X
NameSpace changes:
Why:
    Now supporting more than just Phone
ToDo:
    Coding4Fun.Phone -> Coding4Fun.Toolkit

TimeSpanPicker
Why:
    Removed SL Toolkit dependency, simplifying solution
ToDo:
    Reference Coding4Fun.Toolkit.Controls DLL instead of Coding4Fun.Phone.Controls.Toolkit

RoundButton, RoundToggleButton, OpacityToggleButton, Tile, ImageTile:
Why:
    Add support for vector paths, not just Images
ToDo:
    Content property -> Label property
    Content now becomes what is in the center, path, text, whatever you want
```

## 2. Employing the AboutPrompt

We'll take the simple step of adding an AboutPrompt to the app. When the user selects the "About" menu option, we want to display a popup:

```
80
81     private void AboutClick(object sender, EventArgs e)
82     {
83         AboutPrompt aboutMe = new AboutPrompt();
84     }
```

We're missing a using statement, so use the hover-over-the-blue-dash-technique to add the appropriate using statement to the code file:

A screenshot of Microsoft Visual Studio showing code completion. The code editor shows a snippet of C# code:

```
private void AboutClick(object sender, EventArgs e)
{
    AboutPrompt aboutMe = new AboutPrompt();
}

private void RecordAudioClick()
{
    throw new NotImplementedException();
}
```

The cursor is on the word `AboutPrompt` in the first line. A dropdown menu is open, listing:

- { } using Coding4Fun.Toolkit.Controls;
- Coding4Fun.Toolkit.Controls.AboutPrompt
- Generate class for 'AboutPrompt'

A red arrow points to the third item in the list.

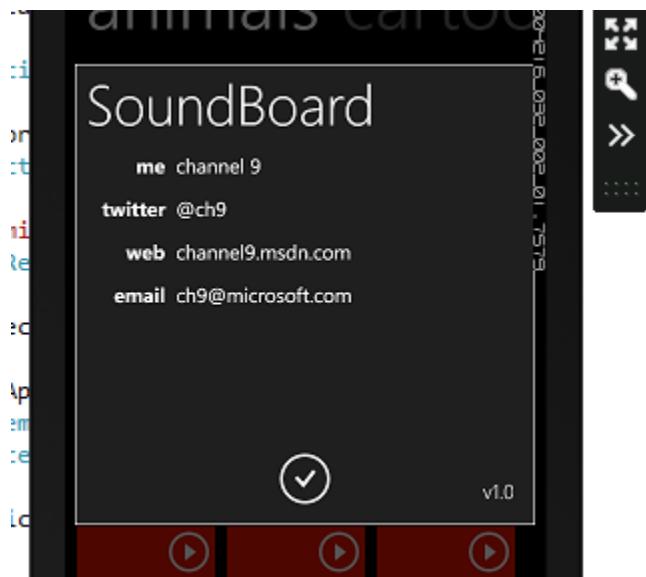
Once we've resolved the reference to the `AboutPrompt` class, we'll call it's `Show()` method. There are many overloaded versions of this method, but we'll pass in the following information ...

A screenshot of Microsoft Visual Studio showing the completed code. The code editor now contains:

```
private void AboutClick(object sender, EventArgs e)
{
    AboutPrompt aboutMe = new AboutPrompt();

    aboutMe.Show("Channel 9", "@ch9", "ch9@microsoft.com", "http://channel9.msdn.com");
}
```

... which produces the following results:



It's quick and simple and for this particular app, we probably need nothing more fancy.

Leveraging open source (or even commercial) packages is a great way to add features quickly to your development projects. The whole .NET community on CodePlex is built on sharing packages, components, templates and so on to help each other quickly implement features. You can save yourself a lot of time during your career by spending a few hours exploring the work of others before going off and writing your own.

Alternatively, use the search box on the NuGet Package Manager to find similar packages and evaluate the strengths and weaknesses of each one relative to the needs of your project.

## Recap

To recap, in this lesson we leveraged the Coding4Fun Toolkit to add one small feature now as well as one large feature later in this series. We learned how to add packages to our app using the NuGet Package Manager dialog, how to read the description of the package to learn more, and how packages can be subdivided to minimize unnecessary references and complexity in our projects.

## Part 18: Navigating Between Pages

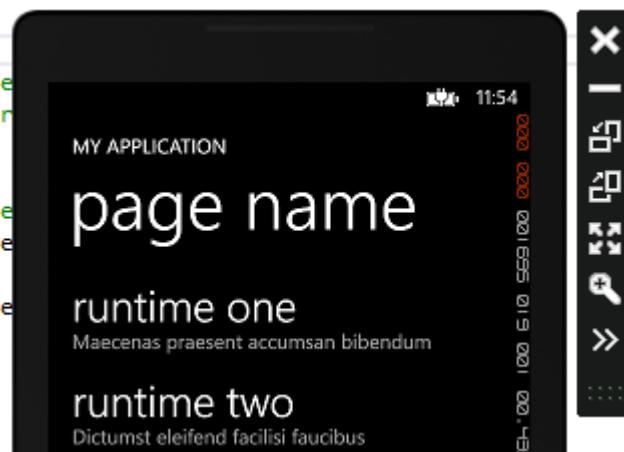
When someone clicks the microphone icon in our app bar, we want to take them to a new page where we can allow them to record a custom sound. To do this, we need to create a second page in our app, and then navigate from the MainPage.xaml to that new page. Navigation in Windows Phone Apps is similar to navigating from one web page to another. In this lesson, we'll learn about the Navigation API and, even though our needs are simple in this particular app, we'll find out the Navigation API capabilities with regards to how the Windows Phone 8 operating system re-hydrates apps shut down due to memory constraints.

Our game plan in this lesson ...

1. Revisit the Databound Project template to observe how it navigated from the main page to the details page.
2. Discuss the classes required for navigation in the Windows Phone 8 API.
3. Implement the code necessary to navigate from our main page to a new page that we'll use to record a custom sound.

1. Revisiting the Databound Project template to learn about navigation

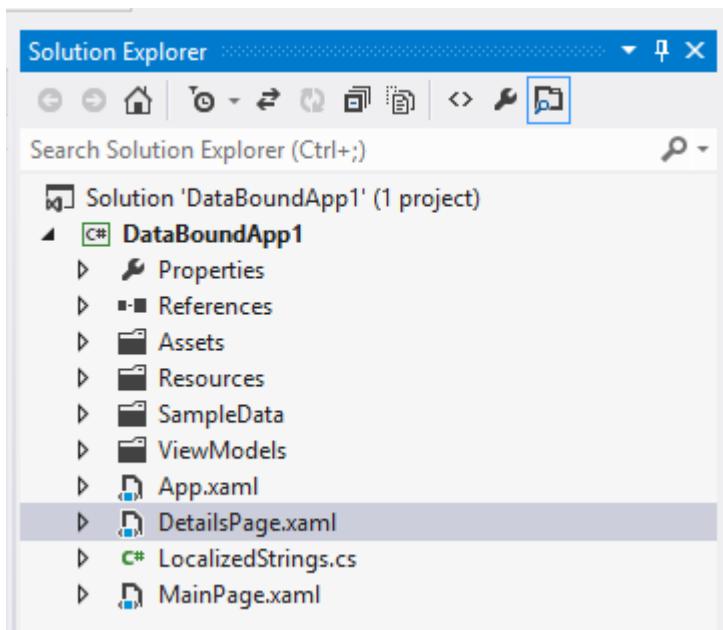
If you'll recall a few lessons ago when we looked at the Databound Project Template, we saw how tapping one of the list items ...



... navigates to another page containing more details about the given item:



In addition to the `MainPage.xaml`, the project template also has a second page called `DetailsPage.xaml`:



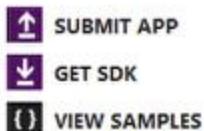
In the `MainPage.xaml.cs` file, in the `MainLongListSelector_SelectionChanged` event handler method, we see the code required to enable this. Focus on lines 47-50:

```
5/
38     // Handle selection changed on LongListSelector
39     private void MainLongListSelector_SelectionChanged(object sender,
40             SelectionChangedEventArgs e)
41     {
42         // If selected item is null (no selection) do nothing
43         if (MainLongListSelector.SelectedItem == null)
44             return;
45
46         // Navigate to the new page
47         NavigationService.Navigate(
48             new Uri("/DetailsPage.xaml?selectedItem="
49                 + (MainLongListSelector.SelectedItem as ItemViewModel).ID,
50                 UriKind.Relative));
51
52         // Reset selected item to null (no selection)
53         MainLongListSelector.SelectedItem = null;
54     }
--
```

The `NavigationService` class is used to simply navigate from one XAML page to another. However, it has a larger role in more complex scenarios. Since it is solely responsible for navigation between XAML pages, it also allows you to inspect the navigation history (called the "back stack"), remove entries from the back stack, and then observe the effect that these changes have on the navigation through the app.

To learn about some of the advanced functionality of the `NavigationService()` class, check out this article on MSDN:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh394012\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh394012(v=vs.105).aspx)



## How to navigate using the back stack for Windows Phone

2 out of 5 rated this helpful - Rate this topic

April 08, 2013

**Applies to:** Windows Phone 8 | Windows Phone OS 7.1

- ▶ Windows Phone development
- ▶ Developing apps
  - ◀ In-app navigation for Windows Phone
    - How to perform page navigation on Windows Phone
    - How to navigate using the back stack for Windows Phone

This topic demonstrates how to modify the navigation of your app by manipulating its navigation history, called the back stack. You can use the `NavigationService` API to inspect and to work with the navigation history. The properties and methods of the `NavigationService` class are used in this topic to inspect the back stack, remove entries, and then observe the effect that these changes have on the navigation through the app.

However in our case, we merely need to navigate from the MainPage.xaml to a new page. We don't need to worry about alternate back page scenarios, passing values between the pages, or the like. Therefore, this other article on MSDN describes the process we'll use in general terms, in case you want to do a little additional research:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626521\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626521(v=vs.105).aspx)

The screenshot shows a Microsoft MSDN article page. On the left, there's a sidebar with three buttons: 'SUBMIT APP', 'GET SDK', and 'VIEW SAMPLES'. Below these are several links under 'Windows Phone development' and 'Developing apps', including 'In-app navigation for Windows Phone', 'How to perform page navigation on Windows Phone', 'How to navigate using the back stack for Windows Phone', 'How to exit or close a Windows Phone app', and 'App page model for Windows'. The main content area has a large title 'How to perform page navigation on Windows Phone'. Below the title, it says '22 out of 28 rated this helpful - Rate this topic' and 'April 08, 2013'. A 'Applies to: Windows Phone 8 | Windows Phone OS 7.1' section follows. The text 'This topic will show you how to navigate back and forth between different pages of content in your app.' is present. A section titled 'This topic contains the following sections.' lists three items: 'Creating an additional page', 'Navigating between pages', and 'Passing parameters'.

Most of the simple navigation scenario is accomplished using the the `Navigate()` method in the code example.

Notice the Uri that is used as an input parameter to the `Navigate()` method. The Uri object represents a Uniform Resource Indicator, similar to a URL but has more utility and as more far reaching. The Uri has two basic parts—a string that represents a location, and a UriKind that should be used to interpret the location string.

Let's start by parsing through the string ... you'll notice that it is part string literal, part dynamic value that retrieves its value from the current selected item in the LongListSelector. They are appended together with the plus + operator:

```
"/DetailsPage.xaml?selectedItem=" + MainLongListSelector.SelectedItem as ItemViewModel).ID
```

The first part of the literal string should be obvious—it is the XAML page we want to navigate to. Everything after that—i.e., after the question mark ? character—is a query string. You've undoubtedly seen a query string before, even if you didn't know what it was called. A query string is a means of sending additional information along with the intended page to be loaded. I said you've undoubtedly seen this before because it is one of two or three primary ways of sending additional data between stateless web pages and is used ubiquitously on the World Wide Web.

If you look at a search for the name "Clint Rutkas" on Bing.com, the navigation bar in your web browser will look like this:

`http://www.bing.com/search?q=clint+rutkas&go=&qs=n&form=QBLH&pq=clint+rutkas&sc=8-9&sp=-1&sk=`

It's simply a clever way of passing information from one web page that's intended to be interpreted and processed by another web page.

In the code example we're examining, when someone taps an item in the LongListSelector we want to send the ID for the item that was selected from the MainPage.xaml to the DetailsPage.xaml ... the ? separates the page name from the query string portion of the URL. The query string is in the form of a name / value pair.

For example:

`selectedItem=3`

... the selectedItem is the name of the pair, and everything after the equals character is the value of the pair.

Two name / value pairs are separated by an & ampersand character. We'll see this used later in this series to send geo-positional latitude and longitude from one page to another ... it will look like this:

`?latitude=41.8986&longitude= 87.6230`

This is how you are able to pass multiple values in one string. Clever.

The second constructor argument for the new Uri takes an enumeration of type UriKind. There are three possible values:

- UriKind.Relative
- UriKind.Absolute
- UriKind.RelativeOrAbsolute

In regards to the differences between the first two, check out this page:

[http://msdn.microsoft.com/en-us/library/system.urikind\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/system.urikind(v=vs.95).aspx)

... which says the following:

"Absolute URIs are characterized by a complete reference to the resource (example: <http://www.contoso.com/index.html>), while a relative Uri depends on a previously defined base URI (example: /index.html)."

In our case, a relative Uri would be "relative to the project's structure". When we prefix the UriString with a forward-slash character / we're specifying the root of the project's deployment package. That should correspond with what we see in the Solution Explorer ... the DetailsPage.xaml is in the root of the project folder. When the project is deployed, those two .xaml files will both be in the root of the package like we saw when we opened up the PetSounds.xap file as a zipped file.

But what about the UriKind.RelativeOrAbsolute? That's a bit trickier. The best I can figure out is that we're simply asking the runtime to figure it out for itself. It will attempt to clean up the UriString we provide and figure out where resources are. I think we could switch that to UriKind.Relative and it would work, too.

Once the NavigationService loads the new page, the DetailsPage.xaml, an event called OnNavigatedTo() fires. Notice its code:

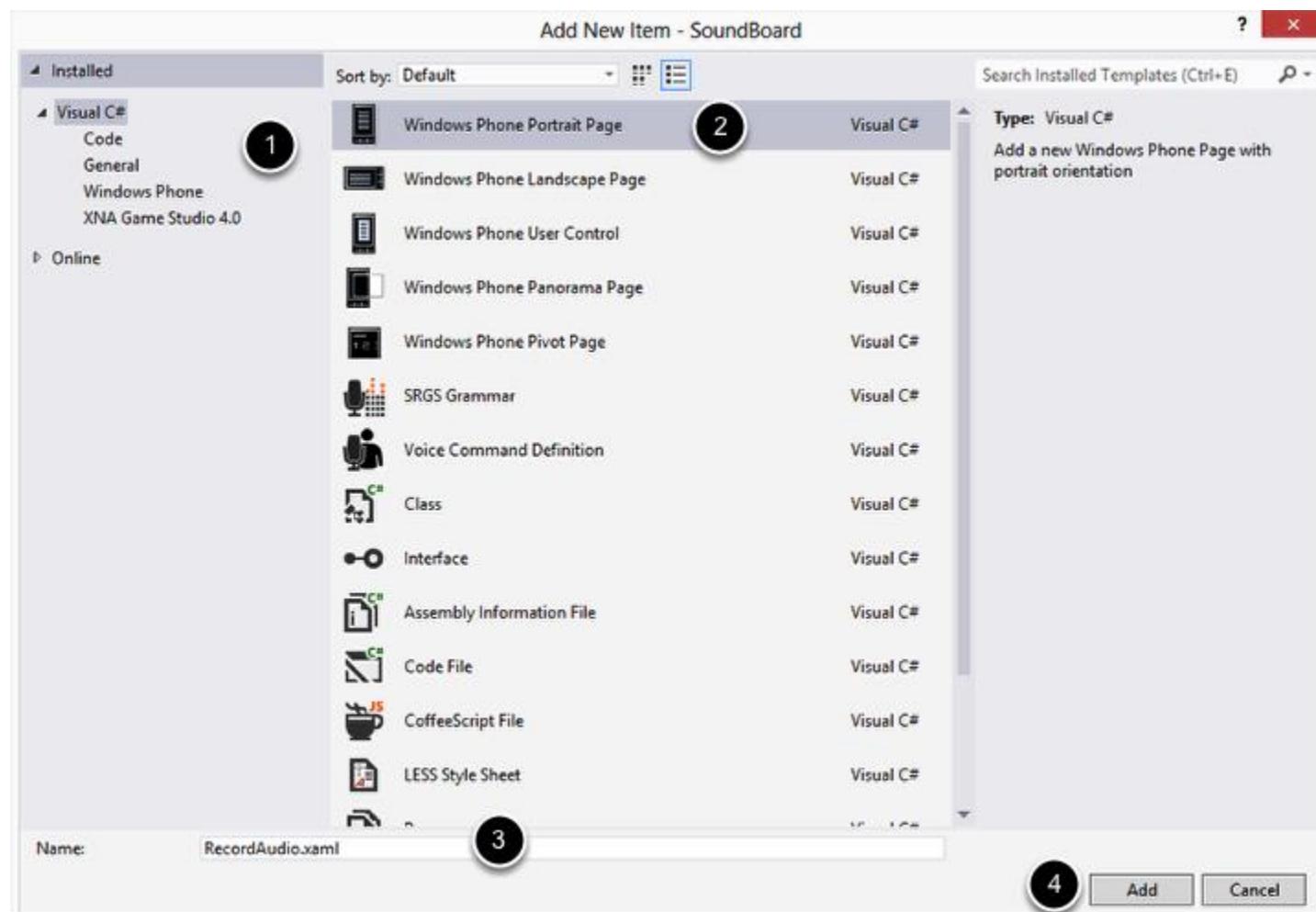
```
25 // When page is navigated to set data context to selected item in list
26 protected override void OnNavigatedTo(NavigationEventArgs e)
27 {
28     if (DataContext == null)
29     {
30         string selectedIndex = "";
31         if (NavigationContext.QueryString.TryGetValue("selectedItem",
32             out selectedIndex))
33         {
34             int index = int.Parse(selectedIndex);
35             DataContext = App.ViewModel.Items[index];
36         }
37     }
38 }
```

In line 31, the NavigationContext.QueryString.TryGetValue("selectedItem") will retrieve the value of the name / value pair in the query string as an out parameter if the parameter name exists. Next, that selectedItem value is used to load the correct item from the data model, and set it as the DataContext for the DetailsPage.xaml so that the various TextBlock controls can bind to it (line 35).

Now that we have seen a full-fledged navigation example at work, we're better prepared to tackle this in our own project.

## 2. Create the RecordAudio.xaml Page

Let's begin by creating that new page we want to navigate to:



1. Make sure you're in the C# file templates section
2. Select "Windows Phone Portrait Page"
3. Rename to: RecordAudio.xaml
4. Click the Add button

Back in the MainPage.xaml page, we'll revisit the event handler method stub we created several lessons ago called RecordAudioClick():

```
80     }
81
82     private void AboutClick(object sender, EventArgs e)
83     {
84         AboutPrompt aboutMe = new AboutPrompt();
85
86         aboutMe.Show("Channel 9", "@ch9", "ch9@microsoft.com", "http://c");
87     }
88
89     private void RecordAudioClick(object sender, EventArgs e)
90     {
91         throw new NotImplementedException();
92     }
93 }
94 }
```

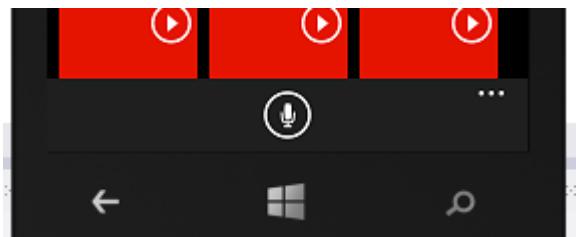
3. Implement the code to navigate to the new page

I'll replace the exception (intended to be a reminder) with the following line of code:

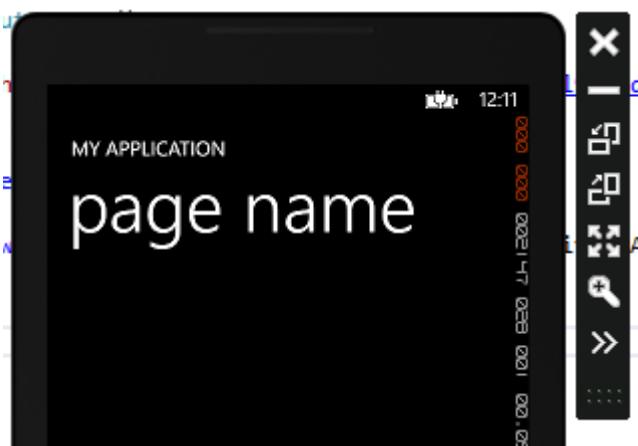
```
88
89     private void RecordAudioClick(object sender, EventArgs e)
90     {
91         NavigationService.Navigate(new Uri("/RecordAudio.xaml", UriKind.RelativeOrAbsolute));
92     }
93 }
94 }
```

Hopefully, this version will be dramatically simplified when compared to the hairy example we saw earlier. We create a new Uri object with a simple address to our new RecordAudio.xaml page, and use the UriKind.RelativeOrAbsolute enum value.

Now, let's test that line of code by running the app and clicking the microphone icon in the app bar:



... if all goes well, we should see a default page template like so:



Great! We'll need to revisit this notion later in this series of lessons with a slightly more complex example in which we pass data between the pages.

## Recap

To recap, the big take away from this lesson is how to navigate between XAML pages using the `NavigationService`. We learned what a `Uri` object is, how to specify a page location and even pass parameters between pages, what the `UriKind` enumeration options mean, and so on.

# Part 19: Setting up the RecordAudio.xaml Page

Source Code: <http://aka.ms/absbeginnerdevwp8>

Now that we can navigate to the new RecordAudio.xaml page, it's time to focus on the layout of this page. If you'll recall from our low-tech UI sketches, we envisioned a simple reel image that would rotate to indicate that the app was recording. There would be a button to start the recording process and a button to play back the sound. Finally, there would be an app bar to save the sound and give it a name. In this lesson, we'll get started by laying out the page and worry about the audio recording functionality and the animation later.

So the game plan in this short lesson is pretty straight forward ...

1. We'll make branding changes, binding to our LocalizedResources.
2. We'll add the buttons and ellipse, and then add some shapes on the ellipse to make it look like an old-fashioned reel.
3. We'll add an app bar like we learned how to do earlier.

1. Perform simple branding changes

We'll begin by modifying the text at the top of the app:

```
“
23      <!--TitlePanel contains the name of the application and page title-->
24      <StackPanel Grid.Row="0"
25          Margin="12,17,0,28">
26          1   <TextBlock Text="{Binding LocalizedResources.ApplicationTitle,
27              Source={StaticResource LocalizedStrings}}"
28                  Style="{StaticResource PhoneTextNormalStyle}"/>
29          2   <TextBlock Text="record audio"
30              Margin="9,-7,0,0"
31                  Style="{StaticResource PhoneTextTitle1Style}"/>
32      </StackPanel>
33
```

1. We'll pull the app's name from our LocalResources.resx file through the familiar syntax we've used many times before.
2. For now, I'll just hardcode the Text attribute as "record audio".

That should produce the following results:



## 2. Perform basic layout of primary controls

I'll add the following XAML in the ContentPanel Grid:

```
34      <!--ContentPanel - place additional content here-->
35      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
36          <StackPanel>
37              <ToggleButton Content="Record" />
38              <Grid Width="200" Height="200">
39                  <Ellipse Fill="{StaticResource PhoneAccentBrush}" />
40              </Grid>
41              <Button Content="Play" />
42          </StackPanel>
43      </Grid>
```

1. We'll use a StackPanel to stack the controls in a vertical fashion.
2. We'll use a ToggleButton, which has two states: on and off. The idea is that we want to turn recording on and off. When the ToggleButton switches states, we'll want to handle the event that is raised to begin or stop the recording process.
3. We'll create a Grid that we'll use to position the Ellipse. We set the color of the Ellipse to the built-in PhoneAccentBrush color, the primary tile color selected for the phone.
4. We'll add one last button for playback of the last sound that was recorded.

It should produce the following results:



The Ellipse control is not complete—we want it to look like an old reel to reel player. We can accomplish this by using several smaller shapes in XAML—a small Ellipse in the center, and small Rectangles aligned to different positions of the main outer Ellipse:

```
38     <Grid Width="200" Height="200">
39         <Ellipse Fill="{StaticResource PhoneAccentBrush}" />
40
41         <Ellipse
42             Fill="{StaticResource PhoneForegroundBrush}"
43             Height="20"
44             Width="20" />
45         <Rectangle
46             Fill="{StaticResource PhoneForegroundBrush}"
47             Height="20"
48             Width="20"
49             VerticalAlignment="Top"
50             Margin="0,20,0,0" />
51         <Rectangle
52             Fill="{StaticResource PhoneForegroundBrush}"
53             Height="20"
54             Width="20"
55             VerticalAlignment="Bottom"
56             Margin="0,0,0,20" />
57         <Rectangle
58             Fill="{StaticResource PhoneForegroundBrush}"
59             Height="20"
60             Width="20"
61             HorizontalAlignment="Left"
62             Margin="20,0,0,0" />
63         <Rectangle
64             Fill="{StaticResource PhoneForegroundBrush}"
65             Height="20"
66             Width="20"
67             HorizontalAlignment="Right"
68             Margin="0,0,20,0" />
69     </Grid>
```

Hopefully most of the code is self explanatory. We use a series of Alignments and Margins to position the shapes on top of the main Ellipse.

If correct, the results should look like this:



### 3. Add an App Bar

We'll uncomment the lines of code provided by the file template to enable an app bar on the page. In the RecordAudio.xaml.cs file:

```
RecordAudio.xaml.cs  ✖  RecordAudio.xaml
SoundBoard.RecordAudio
7  using System.Windows.Navigation;
8  using Microsoft.Phone.Controls;
9  using Microsoft.Phone.Shell;
10 using SoundBoard.Resources;
11
12 namespace SoundBoard
13 {
14     public partial class RecordAudio : PhoneApplicationPage
15     {
16         public RecordAudio()
17         {
18             InitializeComponent();
19
20             1 BuildLocalizedApplicationBar();
21         }
22
23         2 private void BuildLocalizedApplicationBar()
24         {
25             ApplicationBar = new ApplicationBar();
26
27             ApplicationBarIconButton recordAudioAppBar =
28                 new ApplicationBarIconButton();
29             recordAudioAppBar.IconUri =
30                 new Uri("/Assets/AppBar/save.png", UriKind.Relative);
31             recordAudioAppBar.Text = AppResources.AppBarSave;
32
33             recordAudioAppBar.Click += SaveRecordingClick;
34
35             5 ApplicationBar.Buttons.Add(recordAudioAppBar);
36             ApplicationBar.IsVisible = false;
37         }
38
39         7 private void SaveRecordingClick(object sender, EventArgs e)
40         {
41             throw new NotImplementedException();
42         }
43     }
44 }
```

Hopefully none of this is new to you ... I've explained most of this code before, but specific to this app bar:

1. Uncomment the call to BuildLocalizedApplicationBar().
2. Uncomment out the BuildLocalizedApplicationBar() method boilerplate.

3. Create a new ApplicationBarIconButton, and set its IconUri property to the save.png file as well as set its text to the AppResources.resx entry AppBarSave that we created earlier in this series.
4. Create and wire up a new event handler method for the Click event (see #7) ignoring the suggested name when I typed the += characters, instead preferring my own name, and then using the hover-over-the-blue-dash method to reveal the context menu option to generate a method stub.
5. Add the recordAudioAppBar to the app bar.
6. Make the application bar visible.
7. This is the stubbed out event handler you created in step 4 (above).

Running the app and navigating to this page will reveal the app bar with a small disk image titled "save".

#### Recap

To recap, there are no big takeaways per se, just use the techniques we've learned before to implement the special layout for this page. The only new control you met was the ToggleButton which will toggle on or off as opposed to a typically button that can only be tapped / clicked.

# Part 20: Recording an Audio Wav File

Source Code: <http://aka.ms/absbeginnerdevwp8>

In this lesson we'll write the code required to record the custom sound. We'll employ the Coding4Fun Toolkit to help make this easier, but we'll still need to understand things like MemoryStreams and the phone's IsolatedStorage.

Here's our game plan in this lesson:

1. We'll modify the ToggleButton wiring up event handler methods to the Checked and Unchecked events.
2. We'll use the MicrophoneRecord class in the Coding4Fun Toolkit's Audio namespace to begin and stop the recording process.
3. When we stop recording, we'll need to temporarily save the sound stored in the phone's memory to a location on disk so it can be played back or saved permanently.
4. We'll add a MediaElement control so we can enable playback of the sound.
5. Manage the state of the Play sound button, turning it off and on depending on the recording action of the user.

Let me just say that this is perhaps one of the most challenging lessons in this entire series because it deals with some slightly more advanced material. You should embrace this ... you only learn when you struggle, and challenging yourself with difficult concepts will help you grow faster. Be sure to not only watch this video, but also read the MSDN articles that I reference for more information. So put on your thinking cap and let's get started.

1. Modify the ToggleButton Control wiring up Event Handler Methods

Edit the XAML code on the RecordAudio.xaml page for the ToggleButton as follows:



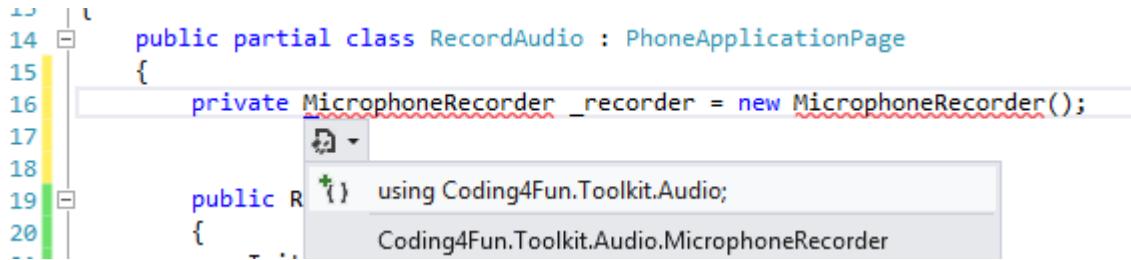
```
37
38 <ToggleButton
39     Checked="RecordAudioChecked"
40     Unchecked="RecordAudioUnchecked"
41     Content="Record" />
42
```

The screenshot shows a code editor with a vertical line of numbers on the left (37, 38, 39, 40, 41, 42) and a horizontal line of numbers at the bottom (41, 42). The code itself is in the center, showing a single `<ToggleButton>` element with three attributes: `Checked="RecordAudioChecked"`, `Unchecked="RecordAudioUnchecked"`, and `Content="Record"`. The word "Content" is highlighted in red, and the entire line is preceded by a yellow vertical bar.

In lines 39 and 40 we wire up method handlers for the two states of the ToggleButton.

2. Create a private instance of the Coding4Fun.Toolkit.Audio.MicrophoneRecord class

In the RecordAudio.xaml.cs file, add the following line of code:

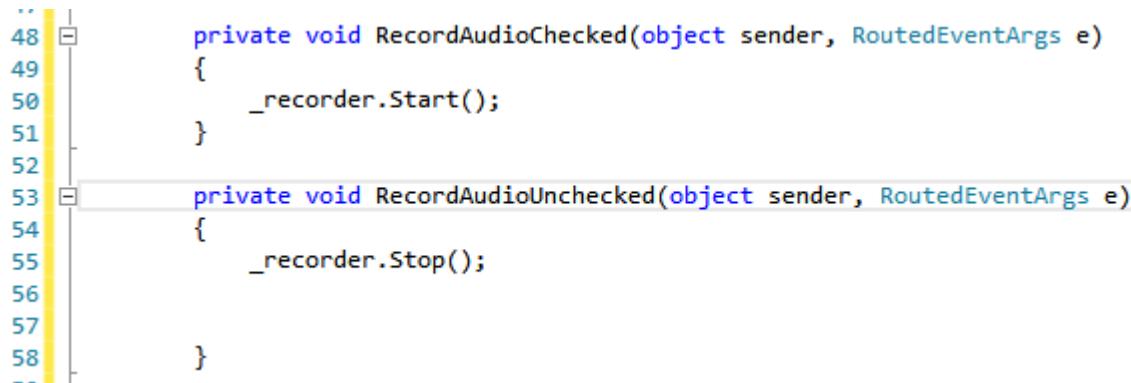


```
14     public partial class RecordAudio : PhoneApplicationPage
15     {
16         private MicrophoneRecorder _recorder = new MicrophoneRecorder();
17
18         public R
19         {
20             ...
21         }
22     }
```

The screenshot shows a portion of C# code in Visual Studio. Line 16 defines a private instance of the `MicrophoneRecorder` class. A tooltip is displayed over the blue-dashed area of the code, showing the full namespace `Coding4Fun.Toolkit.Audio`. The code is part of a class definition for `RecordAudio`.

We create a new private instance of the `MicrophoneRecorder` class, and use the hover-over-the-blue-dash technique to add a using statement for the `Coding4Fun.Toolkit.Audio` namespace.

Now, we can start and stop the `MicrophoneRecorder` by adding code to the `ToggleButton`'s Checked and Unchecked event handler methods:



```
48     private void RecordAudioChecked(object sender, RoutedEventArgs e)
49     {
50         _recorder.Start();
51     }
52
53     private void RecordAudioUnchecked(object sender, RoutedEventArgs e)
54     {
55         _recorder.Stop();
56
57     }
58 }
```

The screenshot shows the implementation of two event handlers: `RecordAudioChecked` and `RecordAudioUnchecked`. Both handlers call the `Start()` and `Stop()` methods of the `_recorder` object respectively.

### 3. Saving the sound data collected by the `MicrophoneRecorder` into a file

As we're recording, the `MicrophoneRecorder` object is collecting the sound information in a buffer. A buffer is just a pocket of memory devoted to storing data. Buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable. So, it may take us 10 seconds to record a 10 second sound and during that time data is being added to the buffer. That said, the computer could process that data in a fraction of a second. The buffer is just a queue ... we can write the data at one rate of speed while reading it at another rate of speed. In programming, buffers are usually used between physical hardware and software, or when moving data from memory to disk and back, or data from memory to a network connection and back. I'm not a computer science guy, so I just think of a buffer as a bucket that you slowly collect things in until you're ready to work with the entire bucket at one time. So, I'm collecting shells on the beach and placing them into my bucket. Once I get a full bucket, I start processing them,

deciding which to keep and which to throw away. I collect over a long period of time, then once I've filled my bucket, I process very quickly. That's how I think of a buffer.

When we call the Stop() method, the MicrophoneRecord stops adding sound information, but is holding that data in memory, in a buffer and NOW we need to process the data in the buffer. In this case, when I use the term "process" what I mean is that I want to grab the sound data out of the buffer and place it as a WAV file into a temporary file.

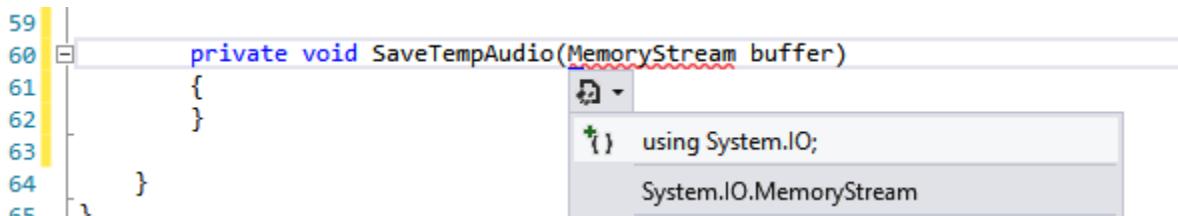
Once it's in a file sitting on my Phone's storage, I'll be able to hand that file over to a MediaElement and direct it to play that temporary WAV file. If the user wants to keep the file, I can re-name it and store it permanently.

So, let's talk about storing files on a Windows Phone. The storage space on the phone is partitioned into isolated areas. Each app installed on the phone gets one of these isolated areas. I use the term "isolated" because one app can't look at the storage area of another app. This is for security ... one app can't rummage through your photos or notes or other secret data and upload it to a secret malicious server or corrupt it in some evil way.

This isolated permanent storage area that's dedicated to your app is called IsolatedStorage. It's just a tidy way of keeping each app's data safe since each app is only able to write and read from its own storage area.

So, back to the problem at hand ... the MicrophoneRecorder object has a bunch on data sitting in a buffer in memory, a MemoryStream object. My job is to save that data from the MemoryStream to a file—a temporary file—in IsolatedStorage. To accomplish this, I'll create a helper method that will take a MemoryStream as an input parameter and then in the body of the helper method, I'll create a new file in IsolatedStorage and then dump all the MemoryStream buffer data into that file.

That's the plan, let's build it:

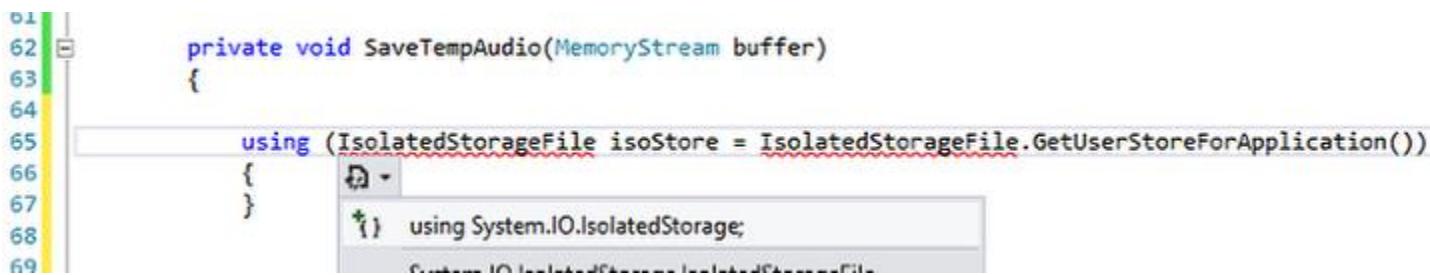


A screenshot of a code editor showing a tooltip for the 'using' keyword. The tooltip contains two entries: 'using System.IO;' and 'System.IO.MemoryStream'. The code in the editor shows a private method 'SaveTempAudio' with a parameter 'buffer' of type 'MemoryStream'.

```
59
60     private void SaveTempAudio(MemoryStream buffer)
61     {
62     }
63 }
64 }
```

As the screenshot indicates, the input parameter is of type MemoryStream and will need a using statement to reference System.IO.

Next, we'll employ a different type of using statement:



In this context, the using statement will create a context for an instance of `System.IO.IsolatedStorage.IsolatedStorageFile`. Any code inside of the code block defined by the using statement will have access to the `isoStore` variable. Once the flow of execution leaves the closing curly brace, the `isoStore` variable will be disposed from memory properly.

You use this using syntax when you want to work with classes that implement `IDisposable` ... typically these are managed types in the .NET Base Class Library (or in our case, the Windows Phone API) that access UNMANAGED resources. So the primary use of the `IDisposable` interface is to release unmanaged resources. The garbage collector automatically releases the memory allocated to a MANAGED object when that object is no longer used. That said, it is not possible to predict when garbage collection will occur. Furthermore, the garbage collector has no knowledge of UNMANAGED resources such as window handles, or open files and streams. The danger is that two or more processes (apps) attempt to access the same resources at the same time and cause an unrecoverable error condition. Types implementing `IDisposable` handle these scenarios correctly. For a more complete explanation of these topics:

#### **using Statement (C# Reference)**

<http://msdn.microsoft.com/en-us/library/yh598w02.aspx>

#### **IDisposable Interface**

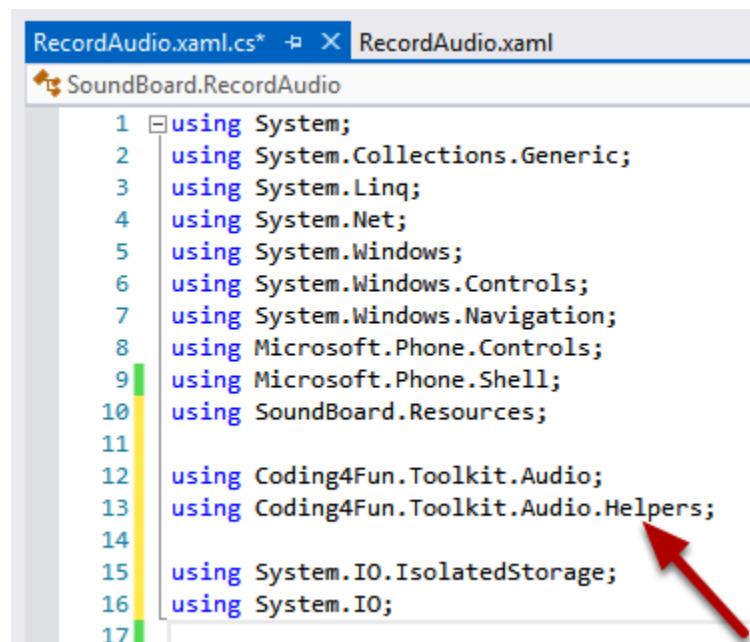
<http://msdn.microsoft.com/en-us/library/system.idisposable.aspx>

The `IsolatedStorageFile` class provides methods that help you manage the files and folder for use by your app. We use the  `GetUserStoreForApplication()` to retrieve the `IsolatedStorage` area for just our app.

Next, we'll grab the buffer and attempt to create a file in our `IsolatedStorage` area:

```
64     private void SaveTempAudio(MemoryStream buffer)
65     {
66
67         using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication())
68         {
69             var bytes = buffer.GetWavAsByteArray(_recorder.SampleRate);
70         }
71
72 }
```

In line 69, I add a line of code that will retrieve values from the buffer (passed in to the helper method) and convert it to the format of a wav (audio) file. As you can see, the `MemoryBuffer` doesn't implement this method. Instead, we want to use an extension method from the `Coding4Fun.Toolkit.Audio.Helpers` namespace. So, in order to apply this extension method, we'll add another using statement at the top of our code file:



```
RecordAudio.xaml.cs*  X  RecordAudio.xaml
SoundBoard.RecordAudio
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Net;
5  using System.Windows;
6  using System.Windows.Controls;
7  using System.Windows.Navigation;
8  using Microsoft.Phone.Controls;
9  using Microsoft.Phone.Shell;
10 using SoundBoard.Resources;
11
12 using Coding4Fun.Toolkit.Audio;
13 using Coding4Fun.Toolkit.Audio.Helpers; -----^
14
15 using System.IO.IsolatedStorage;
16 using System.IO;
17
```

An extension method in .NET allows you to attach a method to any type. So, I could add some utilities to the `int` or `string` or, in this case, the `System.IO.MemoryStream`. The extension method allows you to work with the members of that type just like any public method could. For more information on extension methods, and how to create your own, check out:

<http://msdn.microsoft.com/en-us/library/vstudio/bb383977.aspx>

Note: this is an advanced topic ... as long as you understand what they do and what purpose they serve, that's enough for now. Later you can learn how to create your own.

Ok, so now we have the buffer in wav (audio) format, we just need to actually put it into a new file on the device's IsolatedStorage area:

```
59     private void RecordAudioUnchecked(object sender, RoutedEventArgs e)
60     {
61         _recorder.Stop();
62
63         SaveTempAudio(_recorder.Buffer);
64     }
65
66     private void SaveTempAudio(MemoryStream buffer)
67     {
68
69         using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication())
70         {
71             var bytes = buffer.GetWavAsByteArray(_recorder.SampleRate);
72
73             1         var tempFileName = "tempWav.wav";
74             2         IsolatedStorageFileStream audioStream = isoStore.CreateFile(tempFileName);
75             3         audioStream.Write(bytes, 0, bytes.Length);
76
77             // Play ... SetSource of a MediaElement
78
79         }
80
81     }
82
83 }
```



- (1) We create a temporary name for the wav file. We may discard this in the future (should the user hit the Record button again).
- (2) The CreateFile() method creates a file on the file system using the name we pass in (line 73, above), and gives us back a reference to that we can use to write to.
- (3) Now that we have an empty file to write to, we'll write the data stored in the bytes variable containing our sound. The Write() method's second and third parameters allows us to specify a portion of the sound data we want to write ... in this case, we'll write the entire file from the beginning (0) to the end (bytes.Length).

#### 4. Add a MediaElement to play the new temporary file

In the RecordAudio.xaml file, beneath the ContentPanel grid control, we'll add a MediaElement control, give it a name, and set AutoPlay to false:

```
34      <!--ContentPanel - place additional content here-->
35      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
36          <MediaElement x:Name="AudioPlayer" AutoPlay="False" />
37
38      <StackPanel>
39          <ToggleButton
```



Back in the RecordAudio.xaml.cs, we'll programmatically set the source of our new MediaElement to our new temporarily sound file:

```
76
77      // Play ... SetSource of a MediaElement
78      AudioPlayer.SetSource(audioStream);
79 }
```

Now we're ready to play the sound file ... we just need to wire-up the event handler for the Play button.

#### 5. Handle the Play button's Click Event to Test the Sound

In the RecordAudio.xaml file, in the definition for the Button element with the content "Play", we'll add a Click event handler called "PlayAudioClick":

```
74          Margin="0,0,20,0" />
75      </Grid>
76      <Button Content="Play" Click="PlayAudioClick" />
77  </StackPanel>
78 </Grid>
```

Navigate to the new event handler and add the following line of code to call the Play() method of the MediaElement control:

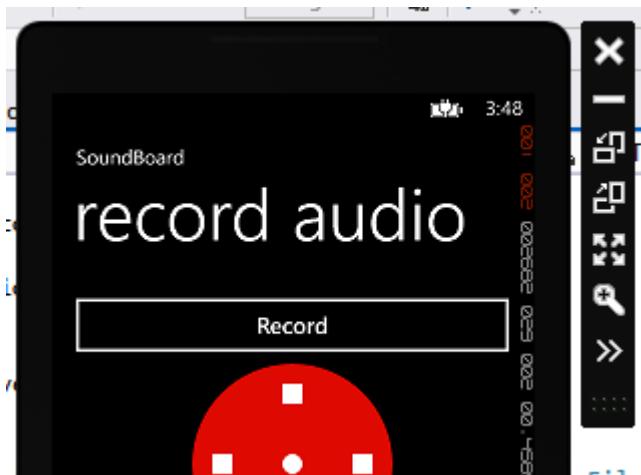
```
81
82      private void PlayAudioClick(object sender, RoutedEventArgs e)
83      {
84          AudioPlayer.Play();
85      }
86
```

At this point we have it all wired up, but it's not pretty ... it's pretty fragile ... so we'll need to comb back through this and program defensively. We'll do that later in this lesson.

For now, let's test our app. We've written a lot of code and, due to the nature of the functionality we've created, it wasn't possible to test it in small parts. We had to implement it all then test it to see where the problems pop up.

## 6. Declaring a Microphone capability for the app

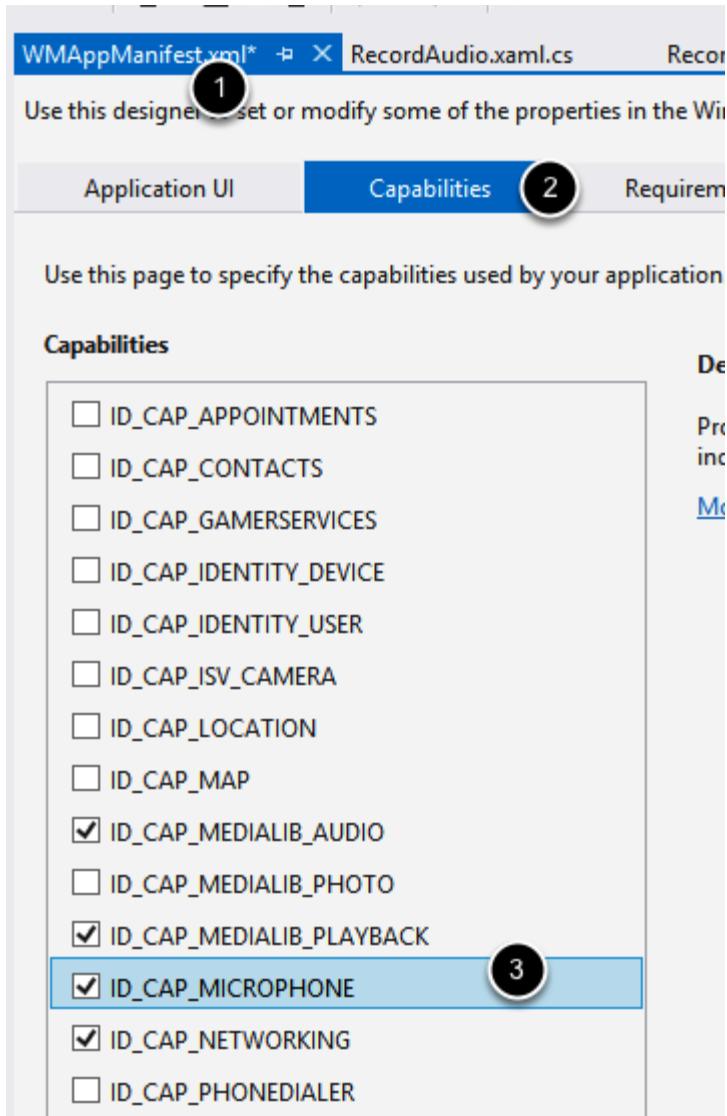
Run the app, click the microphone icon in the application bar, then click the Record ToggleButton:



When you click Record one of two things probably happened ... either the app disappeared silently (no error message), or you'll get an exception, depending on which version of the Coding4Fun Toolkit you're using. The reason this doesn't work is because we do not have permission to use the phone device's microphone. To request permission to use the microphone, we need to declare a capability in our WMAppManifest.xml file.

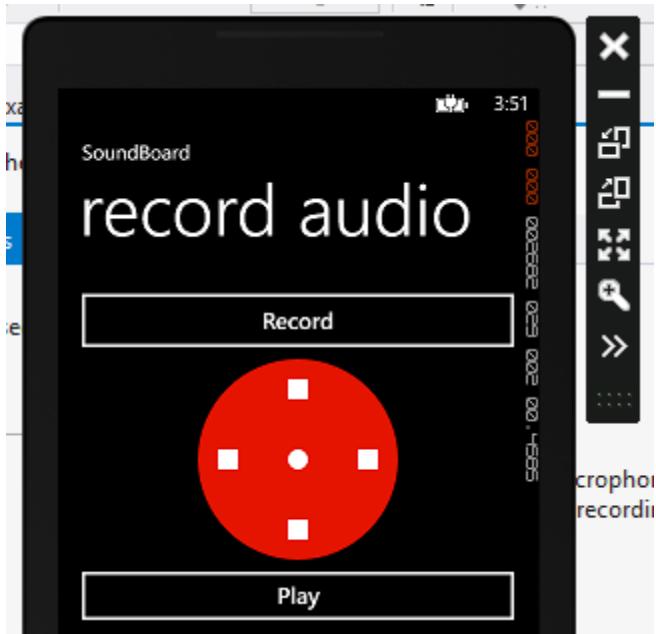
In Windows Phone, you have to request permissions to use certain device capabilities. One purpose of this is to notify the user just how we intend to use their device. They may grow suspicious if our app wants to use their geolocation or their camera when that's not the purpose of our app.

The way to gain access to the capabilities of the phone's hardware is to open up the WMAppManifest.xml file and go to the second tab, "Capabilities":



1. Open up the WMAppManifest.xml file
2. Choose the second tab, Capabilities
3. Add a checkmark next to the ID\_CAP\_MICROPHONE capability

If we re-run the app and go through the same set of steps, it should work. Note: to actually record sound in the Phone Emulator, you will need a working microphone hooked up to the computer. Make sure the mic works prior to starting this so that you can verify that it is not the source of any problems you might encounter:



I think the biggest issue at this point is that (a) the code is pretty fragile and easily broken ... if we were to click the play button and immediately click the record button again, we could probably break the app, and (b) we don't have any visual feedback to let the user know what worked and what didn't, and (c) we are not actually saving the sound permanently, nor are we allowing the user to select a name for the new sound. We'll fix all of these in due time.

For now, let's focus on improving the quality of the code by adding defensive programming statements.

7. Add defensive programming statements to guard against potential exceptions  
First, in the SaveTempAudio helper method, we'll make sure that there's actually data in our buffer before we attempt to create a file and fill it up with the buffer's contents:

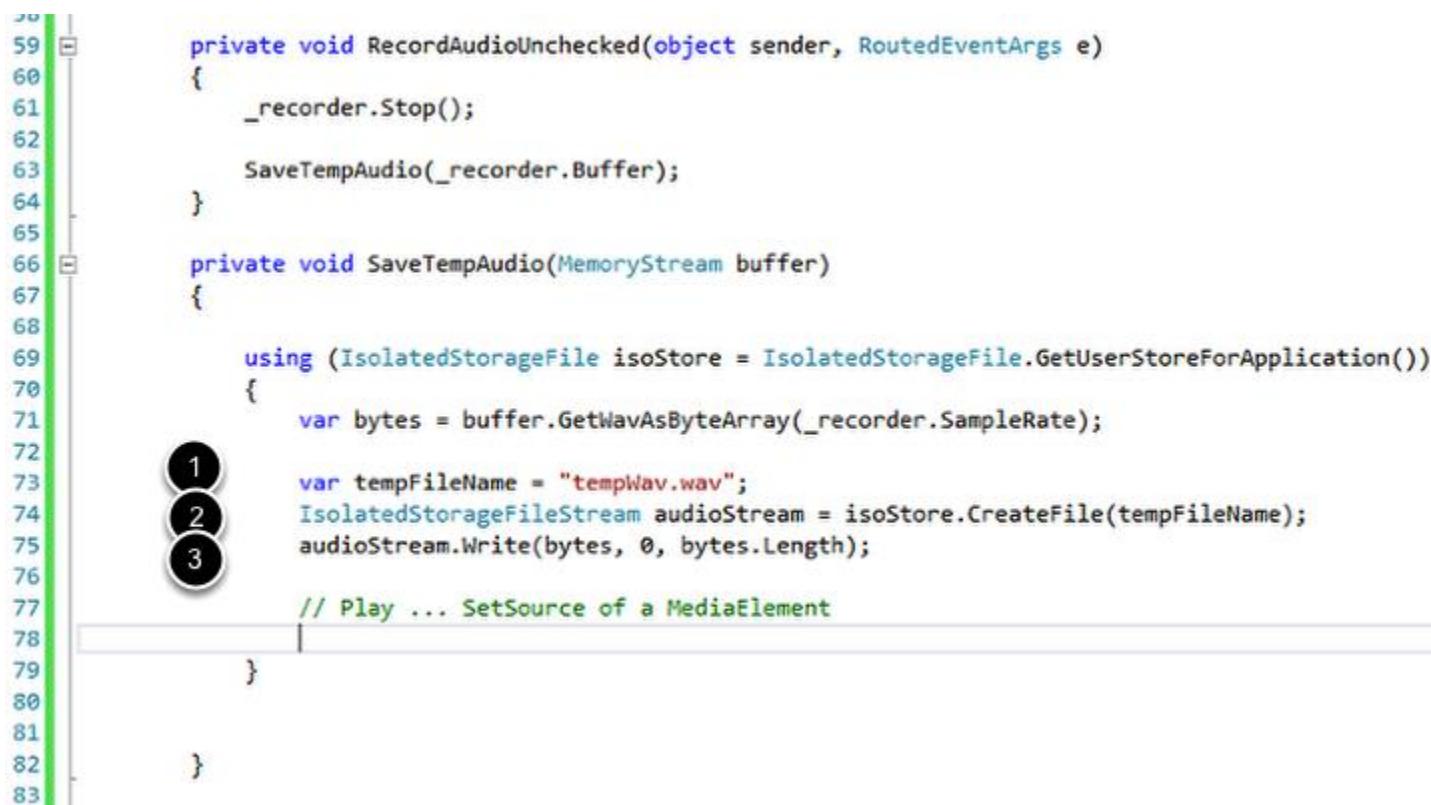
```
--  
65     private void SaveTempAudio(MemoryStream buffer)  
66     {  
67         // Be defensive ... trust no one & nothing  
68         if (buffer == null)  
69             throw new ArgumentNullException("Attempting to save an empty sound buffer.");  
70 }
```

Next, I want to guard against the possibility that the MediaElement is currently playing when the user taps the ToggleButton to record a sound. To do that, I'll refactor my code creating the IsolateStorageFileStream as a private member of the class:

```
18  namespace SoundBoard
19  {
20      public partial class RecordAudio : PhoneApplicationPage
21      {
22          private MicrophoneRecorder _recorder = new MicrophoneRecorder();
23          private IsolatedStorageFileStream _audioStream; ← Red Arrow
24      }
}
```

Now, I'll add code to determine whether the `_audioStream` is empty. If it's not that means I have data that needs to be saved, then I need to empty it out. While I'm there, I'll make sure the MediaElement is no longer playing a sound, and its Source property is set to null. This will release any hold the MediaElement may have on the `_audioStream` from a previous try. It's possible that the MediaPlayer will keep a reference to the `_audioStream` from a previous recording, and I want it to release that reference before I try to create a new file using the same variable name `_audioStream`:

```
59      private void RecordAudioUnchecked(object sender, RoutedEventArgs e)
60      {
61          _recorder.Stop();
62
63          SaveTempAudio(_recorder.Buffer);
64      }
65
66      private void SaveTempAudio(MemoryStream buffer)
67      {
68
69          using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication())
70          {
71              var bytes = buffer.GetWavAsByteArray(_recorder.SampleRate);
72
73              1 var tempFileName = "tempWav.wav";
74              2 IsolatedStorageFileStream audioStream = isoStore.CreateFile(tempFileName);
75              3 audioStream.Write(bytes, 0, bytes.Length);
76
77              // Play ... SetSource of a MediaElement
78
79          }
80
81      }
82
83 }
```



1. Here I check to see whether `_audioStream` is null. If it's not, then I need to make sure that we're not in the middle of using the `_audioStream` in playback mode. In lines 74 and 75 I release the hold that the `MediaElement` has on the `_audioStream`, then ...
2. I call `dispose` on the `_audioStream`. That should properly release any resources `_audioStream` is using. It's now ready to be re-populated with a newly created file handle.
3. Here I re-work the code ... I don't need to create a new instance of `_audioStream` ... it's been cleaned out and all resources have been released. So, I can merely set `_audioStream` to a new file reference.

Now I want to think about how I create and reference the temporary file name. I want to make sure to not leave old temporary files around since they could clutter my apps IsolatedStorage area much like extra files take up too much space on our hard drives. The exception here is that your IsolatedStorage is much smaller than a hard drive, and audio files can be large. So, first I'll move its declaration to a private member variable (line 24, below):

```
--  
20  public partial class RecordAudio : PhoneApplicationPage  
21  {  
22      private MicrophoneRecorder _recorder = new MicrophoneRecorder();  
23      private IsolatedStorageFileStream _audioStream;  
24      private string _tempFileName = "tempWav.wav";  
25  }
```

Next, I'll replace the:

```
var tempFileName = "tempWav.wav";
```

... with the following:

```

66     private void SaveTempAudio(MemoryStream buffer)
67     {
68         // Be defensive ... trust no one & nothing
69         if (buffer == null)
70             throw new ArgumentNullException("Attempting to save an empty sound buffer.");
71
72         // Clean out the AudioPlayer's hold on our audioStream
73         if (_audioStream != null)
74         {
75             AudioPlayer.Stop();
76             AudioPlayer.Source = null;
77
78             _audioStream.Dispose();
79         }
80
81         using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication())
82         {
83
84             1   if (isoStore.FileExists(_tempFileName))
85                 isoStore.DeleteFile(_tempFileName);
86
87             2   _tempFileName = string.Format("{0}.wav", DateTime.NowToFileTime());
88
89             var bytes = buffer.GetWavAsByteArray(_recorder.SampleRate);
90
91             _audioStream = isoStore.CreateFile(_tempFileName);
92             _audioStream.Write(bytes, 0, bytes.Length);
93
94             AudioPlayer.SetSource(_audioStream);
95         }
96     }

```

1. If a temporary file already exists on the user's device from a previous recording attempt, we want to remove that file calling the .DeleteFile() method.
2. We want to give the file a unique name and so we use the DateTime.NowToFileTime() to give it a unique name down to the second. That should be sufficient for our purposes.

Next, I want to disable the Play button when recording. That simple state management should make our lives as developers easier and reduce the possibility that the user is able to corrupt something. I'll give the Button the name "PlayAudio":

```

74         Margin="0,0,20,0" />
75     
```



```

76     </Grid>
77     <Button x:Name="PlayAudio" Content="Play" Click="PlayAudioClick" />
78   
```

Next, I'll perform some state management in the ToggleButton's event handlers:

The screenshot shows a code editor with C# code. The code defines two event handlers for a ToggleButton: `RecordAudioChecked` and `RecordAudioUnchecked`. The `RecordAudioChecked` handler disables the `PlayAudio` button and starts recording. The `RecordAudioUnchecked` handler stops recording and saves the temporary audio buffer. A callout bubble with the number 1 points to the first line of the `RecordAudioChecked` handler. A callout bubble with the number 2 points to the final line of the `RecordAudioUnchecked` handler where the `PlayAudio.IsEnabled` property is set to `true`.

```
54     private void RecordAudioChecked(object sender, RoutedEventArgs e)
55     {
56         PlayAudio.IsEnabled = false;
57         _recorder.Start();
58     }
59
60     private void RecordAudioUnchecked(object sender, RoutedEventArgs e)
61     {
62         _recorder.Stop();
63
64         SaveTempAudio(_recorder.Buffer);
65
66         PlayAudio.IsEnabled = true;
67     }
--
```

1. When I start recording, I'll disable the PlayAudio button, and ...
2. When I stop recording, I'll enable the PlayAudio button.

## Recap

To recap, the big take aways from this lesson include utilizing the Coding4Fun Toolkit's Audio features to greatly reduce the complexity of using the Phone's microphone to record a sound. If you want to peek and see what it does, you can search for the full name of the class in Bing to find a code listing on Codeplex:

<http://coding4fun.codeplex.com/SourceControl/changeset/view/79216#1425409>

We learned about the other using statement in C# to properly dispose of managed classes that work with unmanaged resources. We learned about the function of buffers as a means of collecting data at one rate that could be processed at a different rate. We used the buffer built into the Coding4Fun MicrophoneRecorder class and retrieved the buffer to save it into a file on the device's storage. We learned about IsolatedStorage and how it protects each application's storage area keeping it private from the other apps on the system. We briefly learned about extension methods as we used one that was implemented in the Coding4Fun Audio Helpers namespace and, finally, we practiced some defensive programming techniques to ensure that our app can handle unique situations or edge cases.

# Part 21: Permanently Saving the Audio Wav File

**Source Code:** <http://aka.ms/absbeginnerdevwp8>

At this point, we are recording audio and saving it to a temp file on the app's IsolatedStorage. Next, we need to allow the user to permanently save the sound providing details like the display name for the new custom sound.

The game plan:

1. Add an event handler method to the "save" application bar button
2. We'll manage the state of the application bar ... it should only be visible if a temporary audio file has been created and is ready to be saved permanently
3. We'll use the Coding4Fun Toolkit once again, this time to display an InputDialog to capture the name of the new custom sound audio file
4. We'll serialize the data for the CustomSounds into a JSON file
5. And we'll modify our data model to also load the CustomSounds JSON file to create new instances of the data model for those custom sounds

1. Add an event handler method to the "save" button and manage application bar state  
Earlier we created the application bar for the RecordAudio.xaml page by enabling the BuildLocalizedApplicationBar() method. So all we need to do is make it active:

```

26     public RecordAudio()
27     {
28         InitializeComponent();
29
30         BuildLocalizedApplicationBar();
31     }
32
33     private void BuildLocalizedApplicationBar()
34     {
35         ApplicationBar = new ApplicationBar();
36
37         ApplicationBarIconButton recordAudioAppBar =
38             new ApplicationBarIconButton();
39         recordAudioAppBar.IconUri =
40             new Uri("/Assets/AppBar/save.png", UriKind.Relative);
41         recordAudioAppBar.Text = AppResources.AppBarSave;
42
43         recordAudioAppBar.Click += SaveRecordingClick; 1
44
45         ApplicationBar.Buttons.Add(recordAudioAppBar);
46         ApplicationBar.IsVisible = false; 2
47     }
48

```

1. In line 43 I add an event handler to the Click method using the technique we've utilized throughout this series ... while we're at it, use the other technique I've demonstrated to generate a method stub for the SaveRecordingClick method (hover-mouse-over-blue-dash to reveal an Intellisense menu option to generate the method stub)
2. In line 46 I immediately hide the application bar ... we only want to show it when we have a sound to save (after the user records a custom sound)

Next, we'll enable the application bar after the user stops recording. In the RecordAudioUnchecked() method, we'll set the IsVisible property to true (see line 67, below):

```

59
60     private void RecordAudioUnchecked(object sender, RoutedEventArgs e)
61     {
62         _recorder.Stop();
63
64         SaveTempAudio(_recorder.Buffer);
65
66         PlayAudio.IsEnabled = true;
67         ApplicationBar.IsVisible = true; 67
68     }
69

```



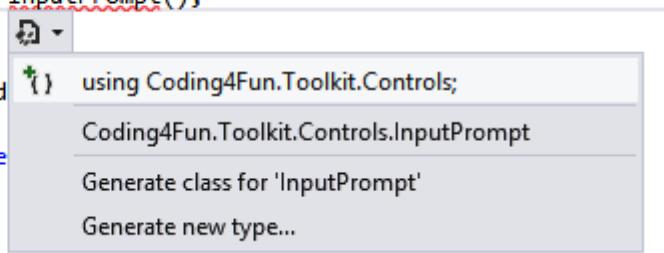
2. Use the Coding4Fun Toolkit to display an InputDialog to capture the name of the new custom sound audio file

In the previous step we added a method stub for the SaveRecordingClick() method.

```
48
49     private void SaveRecordingClick(object sender, EventArgs e)
50     {
51         throw new NotImplementedException();
52     }
53
```

I'll replace the line of code that throws an exception as a reminder and write the following (line 51, below):

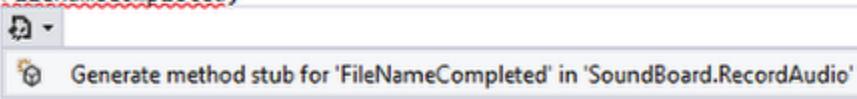
```
49     private void SaveRecordingClick(object sender, EventArgs e)
50     {
51         InputPrompt fileName = new InputPrompt();
52     }
53
54     private void RecordAudioChecked()
55     {
56         PlayAudio.IsEnabled = false;
57         _recorder.Start();
58     }
59
```



Since the InputPrompt is from a different namespace than the others we've utilized so far, we'll need to add a using statement (using the hover-over-the-blue-dash method to reveal the contextual menu).

Next we'll configure and show the InputPrompt:

```
50     private void SaveRecordingClick(object sender, EventArgs e)
51     {
52         InputPrompt fileName = new InputPrompt();
53
54         1 fileName.Title = "Sound Name";
55         fileName.Message = "What should we call the sound?";
56
57         2 fileName.Completed += FileNameCompleted;
58
59         3 fileName.Show();
60     }
61
```



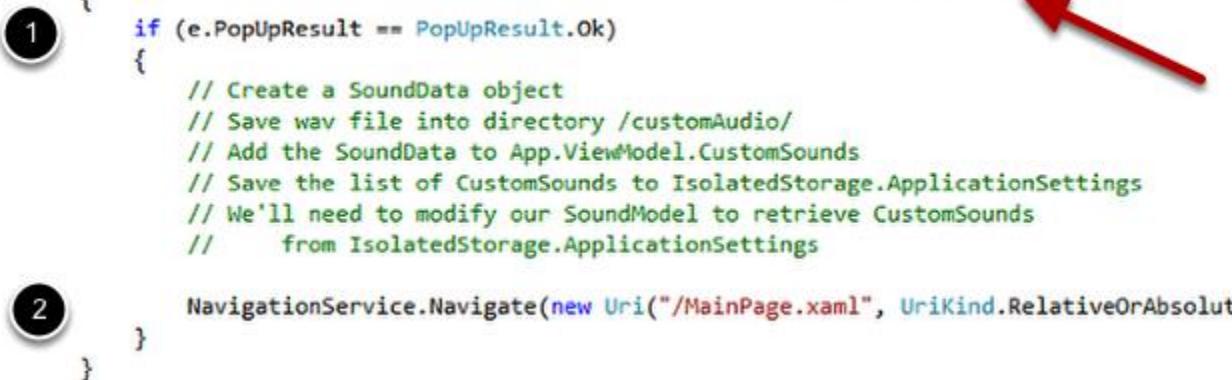
1. Here we set the Title and Message we want to appear in the InputPrompt
2. We attach an event handler method (and generate the method stub using techniques I've demonstrated before) to the Completed event ... we'll tackle that in the next step
3. Once configured, I show the dialog

When the user types in a name for the new custom sound and clicks the checkmark button, the FileNameCompleted() event handler method will fire.

```
61
62     private void FileNameCompleted(object sender, PopUpEventArgs<string, PopUpResult> e)
63     {
64         throw new NotImplementedException();
65     }
66 }
```

We'll ensure that the InputPrompt was exited properly by checking the result. We'll check the PopUpResult that was sent into this event handler method as an input parameter. If the result is "OK" then we can perform the logic necessary to save the temporary file as a new "permanent" sound. See the code I added below, as well as the code comments which give me an outline of the "next steps" I'll want to perform:

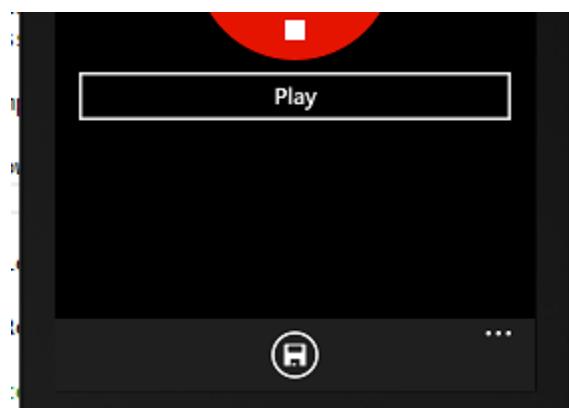
```
54
55
56
57
58
59
60
61
62     private void FileNameCompleted(object sender, PopUpEventArgs<string, PopUpResult> e)
63     {
64         if (e.PopUpResult == PopUpResult.Ok)
65         {
66             // Create a SoundData object
67             // Save wav file into directory /customAudio/
68             // Add the SoundData to App.ViewModel.CustomSounds
69             // Save the list of CustomSounds to IsolatedStorage.ApplicationSettings
70             // We'll need to modify our SoundModel to retrieve CustomSounds
71             //      from IsolatedStorage.ApplicationSettings
72
73             NavigationService.Navigate(new Uri("/ MainPage.xaml", UriKind.RelativeOrAbsolute));
74         }
75     }
76 }
```



1. If the user correctly typed in a new name and clicked the checkmark button to exit the InputDialog, then we'll perform the tasks required to save the custom sound and make it available in the Custom Sounds view of the Sound Board.
2. Finally, we'll navigate back to the MainPage.xaml

Between callouts 1 and 2, above, are an outline of what needs to happen in order for this to work correctly. Before we attempt to implement those ideas, let's make sure the flow works as we would expect so far by running the application.

I record a custom sound by using the ToggleButton. When I stop recording, I in fact see the application bar appear:



When I click the disk icon to save the custom sound, it displays the InputDialog:



And when I type in a new sound name and click the checkmark icon, the dialog disappears and returns me to the MainPage.xaml. Great!

Now, for the hard part ... we'll perform those tasks I outlined in the code comments.

3. Save the sound file into a permanent IsolatedStorage area, serialize the data for the CustomSounds into a JSON file

At this point we have a custom sound recorded and stored as a temporary file and we've just collected a friendly display name for that sound. We want to accomplish two basic tasks:

1. First, we want to add the custom sound to our data model. If information about our new custom sound is never added to the data model, then we'll never be able to render it to the Custom Sounds view on our MainPage.xaml. So, we'll create a new instance of the SoundData class and fill in the FilePath and Title properties appropriately.
2. Next, we'll want to move that file from its temporary location to a permanent subfolder called /customAudio/ ... this is purely to keep all our custom sound files organized in one place.

So, I add the following code to the FileNameCompleted() method:

```

63     private void FileNameCompleted(object sender, PopUpEventArgs<string>, PopUpResult e)
64     {
65         if (e.PopUpResult == PopUpResult.Ok)
66         {
67             // Create a SoundData object
68             SoundData soundData = new SoundData();
69             soundData.FilePath = string.Format("/{0}customAudio/{0}.wav", DateTime.Now.ToFileTime());
70             soundData.Title = e.Result;
71
72             // Save wav file into directory /customAudio/
73             using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication())
74             {
75                 if (!isoStore.DirectoryExists("/customAudio/"))
76                     isoStore.CreateDirectory("/customAudio/");
77
78                 isoStore.MoveFile(_tempFileName, soundData.FilePath);
79             }
80
81             // Add the SoundData to App.ViewModel.CustomSounds
82             App.ViewModel.CustomSounds.Items.Add(soundData);
83
84             // Save the list of CustomSounds to IsolatedStorage.ApplicationSettings
85
86
87             // We'll need to modify our SoundModel to retrieve CustomSounds
88             //      from IsolatedStorage.ApplicationSettings
89
90             NavigationService.Navigate(new Uri("/ MainPage.xaml", UriKind.RelativeOrAbsolute));
91         }
92     }

```

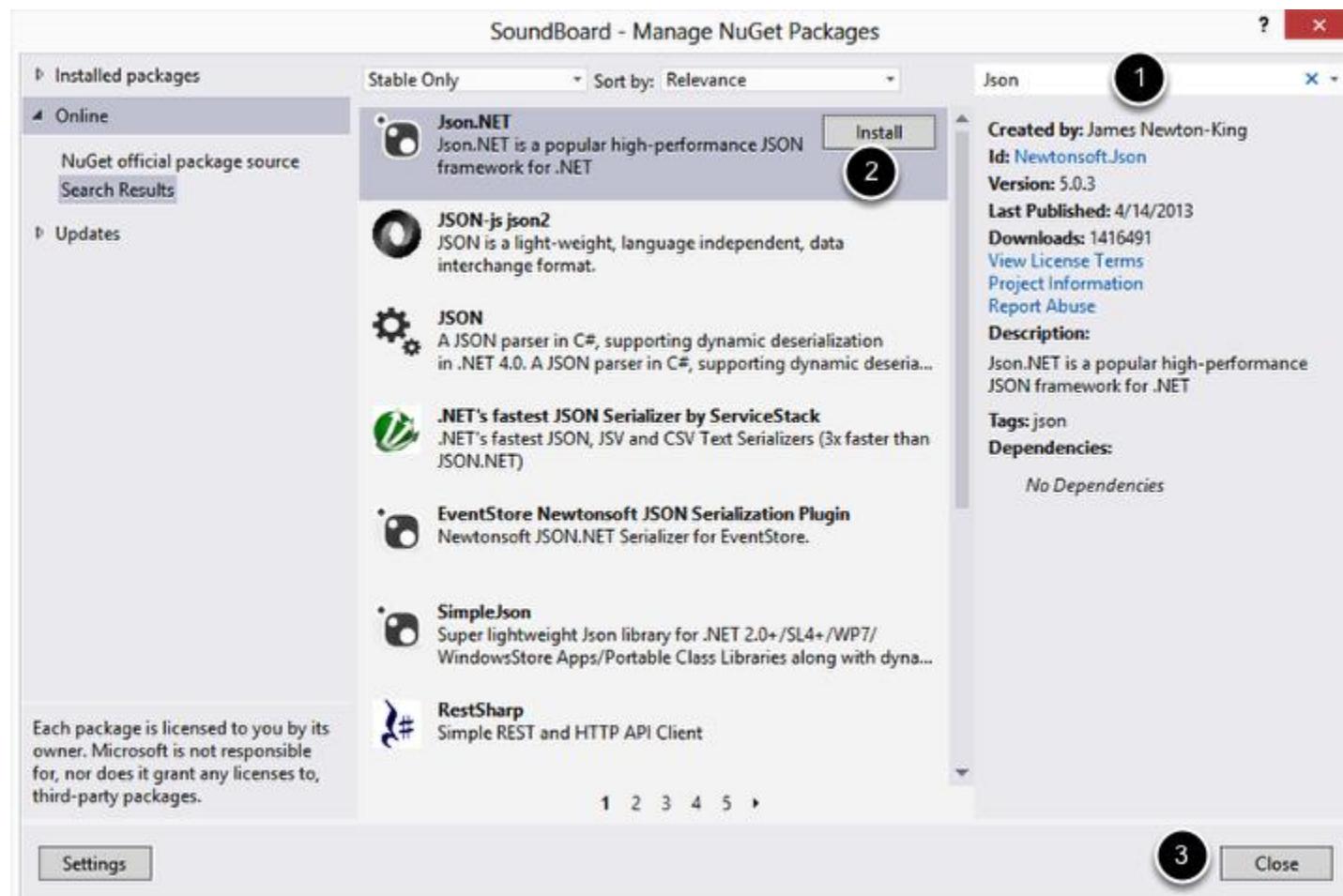
- 1 I create a new instance of SoundData and fill in the Title and FilePath attributes. Notice that we'll be giving our custom sound a new name, but the contents of the file will remain the same.
- 2 Just like when we originally recorded the custom sound, we get a reference to the IsolatedStorage area specifically for our app. We do this with a using statement to properly let go of unmanaged resources (like the Phone's storage). The very first time this code is executed, it may need to create the special folder where we're storing our custom audio files (line 76). Finally, we're moving the temporary file to the new permanent storage area and giving it a new name all in one fell swoop (line 78).
- 3 Next, we're adding the new instance of the SoundData class to our CustomSounds.Items collection. At this moment, we should be able to return back to the MainPage.xaml and see the new custom sound appear in the list of Custom Sounds.

However, what will happen when we close the application and it is completely removed from the Phone's memory? When that happens, the CustomSounds.Items collection will be removed from memory and the next time the app is run, it will have no memory of our custom sounds. We need a way to store our custom sounds data so that we can load it into our data model the next time the user runs our app.

#### 4. Serialize and deserialize the CustomSounds SoundGroup into / out of Json

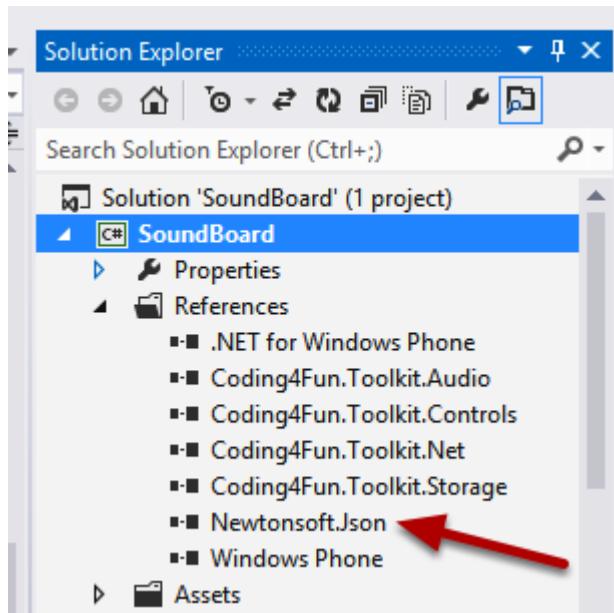
To do this, we'll need to serialize our CustomSounds.Items collection into a data format. There are many data formats we could choose, but we'll pick a very popular, light-weight easy to use format called JSON. It is short for the JavaScript Object Notation. It will allow us to easily represent our collection as JavaScript objects. If we utilize a third-party open source library called Json.NET, we won't even have to think about the data's format ... much of that complexity will be hidden behind simple method calls.

To begin, we'll open up the NuGet Package Manager (using the technique I demonstrated earlier ... right-click the References folder and choose the Manage NuGet Packages ... option.



1. Search for: Json ... one of the top options should be Json.NET.
2. Click the Install button next to the Json.NET package. It will take a few moments to install that package into your project.
3. Click the Close button to continue.

To verify that Json.NET was installed successfully, open up the References folder in the SoundBoard project and verify that Newtonsoft.Json appears there:



Back in the FileNameCompleted() method, the next step is to convert the CustomSounds.Items collection to Json, then store it to disk.

We'll use the Newtonsoft.Json.JsonConvert class to perform the conversion ... you'll need to add the appropriate using statements to accommodate the JsonConvert class:

```
83
84     // Save the list of CustomSounds to IsolatedStorage.ApplicationSettings
85     var data = JsonConvert.
86
87     // We'll ne
88     //   from +} using Newtonsoft.Json;
89
90     Newtonsoft.Json.JsonConvert
91     NavigationService.Navigate(new Uri("mainpage.xaml", UriKind.RelativeOrAbs
```

The code editor shows a tooltip for the 'JsonConvert' class, which is part of the 'Newtonsoft.Json' namespace. The tooltip displays the full class name 'Newtonsoft.Json.JsonConvert'.

Now we're ready to fully implement the storage of the CustomSounds Json file to disk.

```
04  
05 // Save the list of CustomSounds to IsolatedStorage.ApplicationSettings  
06 var data = JsonConvert.SerializeObject(App.ViewModel.CustomSounds);  
07  
08 IsolatedStorageSettings.ApplicationSettings[SoundModel.CustomSoundKey] = data;  
09 IsolatedStorageSettings.ApplicationSettings.Save();  
10  
11 // We'll need to modify our SoundModel to retrieve CustomSounds  
12 // from IsolatedStorage.ApplicationSettings  
13  
14 NavigationService.Navigate(new Uri("/ MainPage.xaml", UriKind.RelativeOrAbsolute));  
15  
16 }  
17  
18 }  
19 }
```

- 1
- 2
- 3

1. We use the JsonConvert.SerializeObject() method to serialize the CustomSounds object (and all it's children, etc.) into Json
2. We'll use a special area of IsolatedStorage called IsolatedStorageSettings to save this object data. This is a simple way to save settings for your app. You can use the name / value pair pattern to save any settings you like for your app. So, in our case, we'll create a key and supply the value ... the value part is obviously the data -- the serialized Json data from the previous line of code. The key part will be a literal string that we'll create as a constant property in the SoundModel class definition. We'll need that key later to RETRIEVE the Json data back out of the IsolatedStorageSettings (we'll do that later in this lesson).
3. We'll call the Save() method to actually save our new ApplicationSetting, the new name / value pair we created in the previous line of code.

Before we forget, let's implement that CustomSoundKey ... in the SoundModel.cs file, I'll add the following line of code (line 19, below):

```
SoundModel.cs  X RecordAudio.xaml.cs      RecordAudio.xaml
SoundBoard.ViewModels.SoundModel
7  namespace SoundBoard.ViewModels
8  {
9      public class SoundModel
10     {
11         public SoundGroup CustomSounds { get; set; }
12         public SoundGroup Animals { get; set; }
13         public SoundGroup Cartoons { get; set; }
14         public SoundGroup Taunts { get; set; }
15         public SoundGroup Warnings { get; set; }
16
17         public bool IsDataLoaded { get; set; }
18
19         public const string CustomSoundKey = "CustomSound"; -----^
20
21         public void LoadData()
22         {
23             Animals = CreateAnimalsGroup();
24             Cartoons = CreateCartoonsGroup();
25             Taunts = CreateTauntsGroup();
26             Warnings = CreateWarningsGroup();
27
28             IsDataLoaded = true;
29         }
30     }
```

As you can see, this is nothing more than a constant string value. We want it to be constant because it should never change. It's simply a unique string we'll use as the means of getting back at the right ApplicationSetting in the IsolatedStorageSettings store.

Next, we'll want to load our custom sounds into memory at the same time we instantiate all of the other SoundGroup objects ... in the SoundModel.cs file, in the LoadData() method:

```

21     public void LoadData()
22     {
23         Animals = CreateAnimalsGroup();
24         Cartoons = CreateCartoonsGroup();
25         Taunts = CreateTauntsGroup();
26         Warnings = CreateWarningsGroup();
27
28         CustomSounds = LoadCustomSounds(); ← Red arrow here
29         IsDataLoaded =
30     }
31
32 }
```

In line 28 (above) we'll call a helper method, LoadCustomSounds(), to populate the CustomSounds property of the SoundModel class. Use the technique I demonstrated earlier to generate a method stub for this new method.

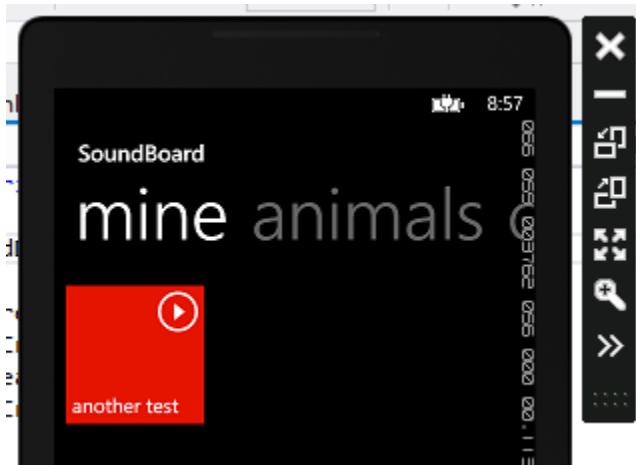
In our new LoadCustomSounds() method, we'll attempt to retrieve the Json that contains the serialized Custom Sound data from the IsolatedStorageSettings:

```

35     private SoundGroup LoadCustomSounds()
36     {
37         SoundGroup data;
38         string dataFromAppSettings;
39
40         if (IsolatedStorageSettings.ApplicationSettings.TryGetValue(CustomSoundKey, out dataFromAppSettings))
41         {
42             data = JsonConvert.DeserializeObject<SoundGroup>(dataFromAppSettings);
43         }
44         else
45         {
46             data = new SoundGroup();
47             data.Title = "mine";
48         }
49
50         return data;
51     }
```

1. We perform a TryGetValue() method on the IsolatedStorageSettings.ApplicationSettings ... if the CustomSoundKey exists in IsolatedStorage, then it should return the value (i.e., the Json we stored previously) into the out parameter, "dataFromAppSettings". If not, then the else code block will create a new (empty) instance of SoundGroup.
2. Now that we have the serialized data, we want to DESERIALIZE that data back into instances of SoundGroup (and SoundData) objects. We call the generic DeserializeObject<T>() method, giving it the type we expect to deserialize it into (i.e., SoundGroup) and pass in the data we retrieved from the IsolatedStorageSettings.
3. Assuming the TryGetValue failed, that means no custom sounds were created (or perhaps there was a problem retrieving the data). In either case, return an empty SoundGroup.

Now we cross our fingers and test the app. I'll record a sound and attempt to save the sound with the name "another test".



All looks good until I return to the MainPage.xaml after I've saved the new custom sound and attempt to play it. However, it doesn't play back! That's because we need to modify the playback code on the MainPage.xaml to load CustomSounds FROM THE NEW FOLDER! It only loads from the /Assets folder right now!

The goal is to set the MediaElement's Source property with the correct location of the sound file associated with the tile our user tapped. We'll look in one of two places ... either in the /Assets folder, or in the IsolatedStorage area.

In the MainPage.xaml.cs, in the LongListSelector\_SelectionChanged() event handler method I add the following:

```
46
47     SoundData data = selector.SelectedItem as SoundData;
48
49     // verifying our sender is actually SoundData
50     if (data == null)
51         return;
52
53     // is file a custom recorded file?
54     if (File.Exists(data.FilePath)) ←
55     {
56         // ...
57     }
58
59     AudioSource = new Uri(data.FilePath, UriKind.RelativeOrAbsolute);
60
```

Here I'm attempting to see if the tile selected by the user is a custom sound. If we can't locate the file name associated with the tile in the Assets\ folder, then we'll look for it in the /customSounds/ sub-folder where in the app's IsolatedStorage area.

For this check, we'll need to access the file system of the Phone, so we'll use the System.IO.File object. The screenshot above shows how we will want to add the appropriate using statement at the top of the code file to include System.IO.

Additionally, we'll want to include a reference to System.IO.IsolatedStorage since we'll be working with classes in that namespace next:

```
 8     using Microsoft.Phone.Controls;
 9     using Microsoft.Phone.Shell;
10    using SoundBoard.Resources;
11    using SoundBoard.ViewModels;
12    using Coding4Fun.Toolkit.Controls;
13    using System.IO;
14    using System.IO.IsolatedStorage;
15
16    namespace SoundBoard
17    {
```

Back in the LongListSelector\_SelectionChanged() event handler method, the File.Exists() will return false if the SoundData object's FilePath cannot be found in the default location. Otherwise it will be in the /Assets folder. So, based on that I write the following code:

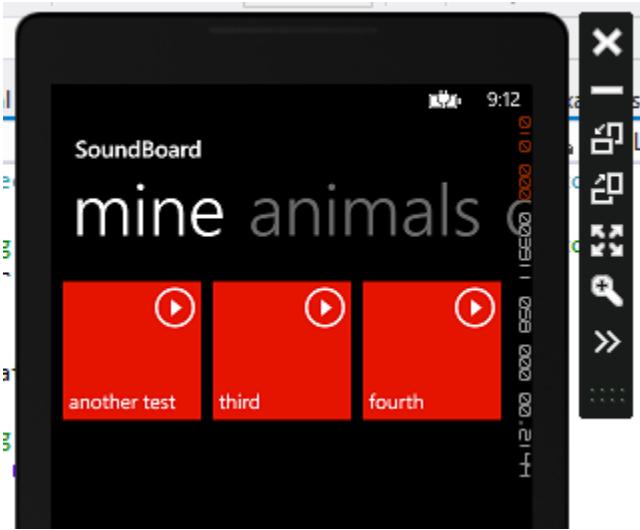
```

50          // verifying our sender is actually SoundData
51          if (data == null)
52              return;
53
54          // is file a custom recorded file?
55          if (File.Exists(data.FilePath))
56          {
57              AudioPlayer.Source = new Uri(data.FilePath, UriKind.RelativeOrAbsolute);
58          }
59          else
60          {
61              using (var storageFolder = IsolatedStorageFile.GetUserStoreForApplication())
62              {
63                  using (var stream = new IsolatedStorageFileStream(data.FilePath, FileMode.Open, storageFolder))
64                  {
65                      AudioPlayer.SetSource(stream);
66                  }
67              }
68          }
69
70          // resetting selected so we can play the same sound over and over again
71          selector.SelectedItem = null;
72      }
73  
```

- 1
- 2
- 3
- 4

1. Here I am setting the MediaElement's Sound property like before using the SoundData's FilePath property
2. In this case, the file was not in the /Assets folder so we'll search for it in IsolatedStorage. Here we create a reference to the IsolatedStorage for our app. Note the using statement so that we can properly dispose of this resource when we're finished.
3. Here we get access to the file using a new technique. We're opening the custom sound as a stream, or more specifically, a IsolatedStorageFileStream. More about streams in a moment.
4. Here we use the SetSource property of the MediaElement to the stream containing our custom sound from IsolatedStorage.

This time when we run the app, record and save a sound, then return to the "mine" (Custom Sounds) category, each of our saved custom sounds should play correctly!



## Recap

To recap, the big take aways from this lesson were Serializing and Deserializing objects into JSON using Newtonsoft Json, which is a very valuable skill that transcends Phone development. We also learned how to work with IsolatedStorage, particularly the IsolatedStorageSettings to store name / value pairs. We used the System.IO.File class to examine the file system, and learned about working with streams. We also used the InputPrompt from the Coding4Fun Toolkit and a bunch more. We are almost done ... we'll just add one last touch to the app to make it fun to use.

# Part 22: Animating the Reel Grid with a Storyboard

Source Code: <http://aka.ms/absbeginnerdevwp8>

The final step will be fun ... we'll animate the reel object on our record audio screen. We want it to rotate as we're recording ... that's a nice visualization of what the app is doing, a nice visual cue to the user that they are in the middle of an important operation with the app.

Our game plan in this lesson:

1. We'll create a Storyboard that includes an animation to rotate the Ellipse and the other objects.
2. We'll programmatically start and stop the animation

... but first we'll need to learn a little about creating animations on the Windows Phone.

1. Declaratively define the animation

Before we dive too deeply into the topic of animation, I want to discuss it at a high level. Once we're comfortable with how animation works in XAML UI frameworks, we'll have a bit more confidence to apply a specific animation to rotate our reel.

First, it's important to realize that XAML controls like the Grid, Button, etc. are defined separately from animation definitions. You can create an animation that is applied to different controls as needed. An animation is associated with a control and then triggered by some event. In our case, we've already defined the Grid and the shapes that will comprise the reel. We'll soon define an animation and associate the two, then we'll write C# code to trigger the animation. Again, we'll do all of this before the end of this lesson.

Animations are made up of a Storyboard object and at least one Animation object.

Let's start with the Animation. First, there are several different types of animation data types. We're using a DoubleAnimation data type because we want to move a property (the Rotation property) from 0.0 to 360.0—the angle of the animation every 4 seconds. In other words, we want the Grid to rotate every 4 seconds. Besides the DoubleAnimation, there's also a ColorAnimation class for animating between two colors, and a PointAnimation for modifying an object's X Y coordinate or its size.

An Animation is basically a timeline combined with a result. The results are determined by the specific Animation class you pick like we just talked about. It's important to know

that all Animation classes inherit from a Timeline class which confers properties related to timing of the animation ... the Begin time, allowing you to delay the start of animation, perhaps waiting for other animations on the storyboard to begin or complete, a Duration property that effects how long the animation should take before delivering the desired result—how long should it take for our fade-in effect, in this case. There are also AutoReverse and RepeatBehavior properties, which do what they suggest.

A Storyboard is a collection of one or more Animations. You group the Animations you want triggered by a specific event such as a button click, a loaded event, and so on. The Storyboard allows you to pair an Animation with a target object. The animation is just a definition of what property should be affected, when it should be affected and how long. We have to APPLY that Animation to a target object. In our case, that target object will be the Grid that contains our shapes representing the reel.

Our Storyboard will be simple since we just want one thing to happen—a rotation from 0 to 360 degrees every 4 seconds. Once we're ready to allow the Storyboard to play, we call its Begin() method. It will in turn kick off all its child Animations and child Storyboards.

There are 4 possible transforms that you can animate as listed and diagrammed on this page:

<http://msdn.microsoft.com/en-us/library/ms750596.aspx>

(I realize that this article is specifically for WPF, but many of the concepts transfer to the Windows Phone API.)

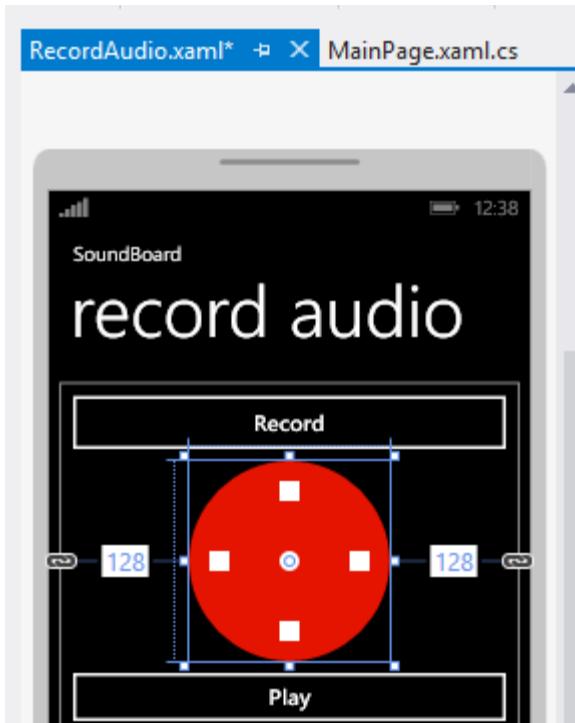
- RotateTransform
- ScaleTransform
- SkewTransform
- TranslateTransform

... as well as other attributes you can change with the ColorAnimation and the like. There's a lot more to learn about animations, and for a more complete explanation of animations, I recommend you start here:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206955\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206955(v=vs.105).aspx)

Now that you have the basics of StoryBoards and Animations, let's modify the Grid containing the shapes that comprise the reel graphic. You'll recall that the large ellipse and the smaller rectangles and ellipse are all contained in a Grid for layout purposes.

To get them to all move correctly at the same time, it is easier to just animate / rotate the grid (rather than each individual shape):



I'll add a name so I can access it programmatically, and I want to indicate that I want to entire grid and all its children to animate together, as a group, or rather as a composite:

```

47
48     <StackPanel>
49         <ToggleButton
50             Checked="RecordAudioChecked"
51             Unchecked="RecordAudioUnchecked"
52             Content="Record" />
53
54     1   <Grid Width="200" Height="200" Name="ReelGrid">
55         <Grid.RenderTransform>
56             <CompositeTransform />
57         </Grid.RenderTransform>
58
59         <Ellipse Fill="{StaticResource PhoneAccentBrush}" />

```

1. Add the programmatic name attribute.
2. I set the RenderTransform property to CompositeTransform. Essentially, what I'm saying is that I will transform the contents of the Grid as one complete unit. I've not provided the details of HOW I'll transform the grid, just the fact that, for the purpose of performing transformations on the Grid, I want to treat the Grid and its children as a group. I'll define HOW I'll transform the grid in a Storyboard.

Near the top of the RecordAudio.xaml page (beneath the PhoneApplicationPage declaration), I will add a Resources section:

```

15  <phone:PhoneApplicationPage.Resources>
16      <Storyboard x:Name="RotateCircle" RepeatBehavior="Forever">
17          <DoubleAnimation
18              Duration="0:0:4" To="360"
19              Storyboard.TargetProperty="(UIElement.RenderTransform).(CompositeTransform.Rotation)"
20              Storyboard.TargetName="ReelGrid"
21              d:IsOptimized="True" />
22      </Storyboard>
23  </phone:PhoneApplicationPage.Resources>
24
25

```

1. We're creating a Page-level resource.
2. We're creating a Storyboard ... because it's Page-level, we could re-use this in other spots in our page. We'll give the Storyboard a name, "RotateCircle" and once it starts, its repeat behavior should be "Forever". We'll control start and stop programmatically. I'll access this Storyboard programmatically by its name in a moment to start / stop it.
3. We'll build a DoubleAnimation object. Why DoubleAnimation? As we discussed at the outset, we want to modify the Rotation property (see the Storyboard.Target property) from 0.0 to 360.0, which are numerical (and therefore will require we use the DoubleAnimation). We set the duration of the animation to 4 seconds and so, in 4 seconds, we want to increase the Rotation value starting at 0 and increasing to 360. We set the TargetName, which is the name we gave the Grid containing our reel. The TargetProperty is tricky ... we have to write it this way due to the fact that we're working with a CompositeTransform ... as I mentioned a moment ago, we want to animate the Grid AND ITS CHILDREN. Otherwise, we could simply set the TargetProperty="Rotation".

## 2. Programmatically start and stop the animation

What comes next is easy now that we've declaratively defined the animation. We'll start and stop the animation based on the state of the ToggleButton:

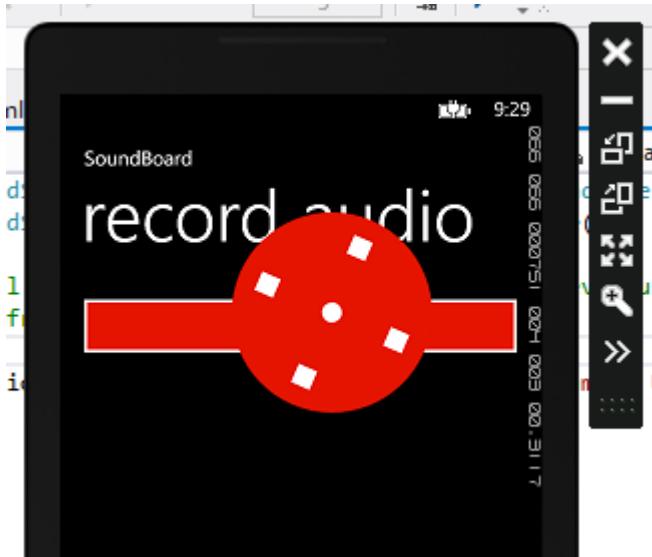
```

100     private void RecordAudioChecked(object sender, RoutedEventArgs e)
101     {
102         PlayAudio.IsEnabled = false;
103         ApplicationBar.IsVisible = false;
104         RotateCircle.Begin();
105
106         _recorder.Start();
107     }
108
109     private void RecordAudioUnchecked(object sender, RoutedEventArgs e)
110     {
111         _recorder.Stop();
112
113         SaveTempAudio(_recorder.Buffer);
114
115         PlayAudio.IsEnabled = true;
116         ApplicationBar.IsVisible = true;
117         RotateCircle.Stop();
118     }

```

1. While recording, start the animation.
2. When finished recording, stop the animation.

If we were to run the application at this point, we would notice an odd behavior while recording:



The reel is not rotating in place, but rather, it is rotating around the page. This is because we didn't define the spot of origin for the transform to take place. Instead of at its default X Y position (0, 0 ... upper left-hand corner), we want it to be in the middle (.5, .5), so we'll set the `RenderTransformOrigin` to that position:

```
53             Content="RECORD" />
54
55         <Grid Width="200"
56             Height="200"
57             Name="ReelGrid"
58             RenderTransformOrigin=".5, .5">
59             <Grid.RenderTransform>
60                 <CompositeTransform />
61             </Grid.RenderTransform>
62
63             <Ellipse Fill="{StaticResource PhoneAccentBrush}" />
64
```

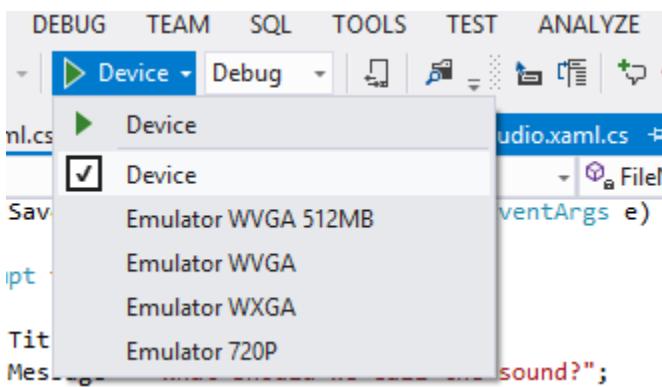
A red arrow points to the `RenderTransformOrigin` attribute in the XAML code, specifically to the value ".5, .5".

Now when we re-run the app, the reel spins instead of traveling:



### 3. Deploy to physical Phone Device

Just for fun, I wanted to see this working on my Nokia Lumia 920, so I connected my device to my computer with a USB cable and used the drop-down next to the Run button on the toolbar to select Device:



I was able to run the app, record and save sounds, even hit break points just like I could using the Emulator in Visual Studio. Excellent.

## Recap

To recap, in this lesson we learned about animation using Storyboard and the relation to Animation classes. We talked about the different types of animations and transforms you can add to your app. Then, we learned how to programmatically trigger the storyboard and how to stop it. We learned about things like how to modify the rotation animation's starting point by changing RenderTransformOrigin attribute. The big takeaway is that if you can imagine it, there's likely a way to accomplish it given the wealth of animation options in XAML and the Windows Phone API.

Now that we've finished our first version of this app, I want to challenge you to add some functionality on your own. I haven't yet added these features myself, and there's no denying that they would require a day or two to research, develop and debug, but here's what I would like you to try and add:

1. Figure out how to add images to each item. You'll need to manipulate the data model to accommodate an associated image file, and you'll have to add image files to the project, add an Image control to the DataTemplate, and bind to it. Actually, if you watch the rest of this series you'll get some hints on how to accomplish this because we'll be binding images to tiles (among other things) as we create our second full app.
2. Figure out how to delete items. Right now, if you add a custom sound, there's no way to get rid of it. That's an unfortunate limitation. Clint suggests you use a Context Menu from the Windows Phone Toolkit (which we'll learn about that when we work on the next app as well).
3. Figure out how to pin sounds to the Start page. That's something I've never tried to implement, but I know there are tutorials out there that can lead you in the right direction.

Are there other features that would make this app cool? Try adding them or share them as challenges to your fellow students in the comments below the video.

Now we have to move on ... we'll begin our second app in the next lesson.

# Part 23: Testing and Submitting to the Store

Source Code: <http://aka.ms/absbeginnerdevwp8>

Now that we've finished the app, it's time to test it and submit it to the Store.

A word of caution before we continue ... at a minimum, your users expect your app to be solid and to look professional.

With regards to creating a solid user experience, there's only so much you can do by yourself because you have the curse of knowledge. As the developer, your knowledge of the flow and "normal operation" of the app creates a myopia. You'll quickly find that your users will brake the app in new and creative ways. So, I would spend a lot of time thinking about the states of the application, particularly any scenario that accepts information from the user in input controls or dialogs. I would spend time thinking about exception handling and messages back to the user to help diagnose the problem. I would think about inputs and outputs and edge case scenarios ... What if they use strange characters outside of the ASCII or Extended ASCII range? What if they add two sounds with the same name? What if the program runs out of battery while the user is recording a sound? What happens if the user clicks one of the hardware buttons while recording or playing back a sound? What if the user fills up Isolated Storage with custom sounds? As it stands now, that would be a problem because we don't have a delete sound feature in the app.

With regards to a professional appearance, here again you have the curse of knowledge and quite possibly a cognitive bias that leads you to think the flow and aesthetics of the app look great when your users think otherwise. Getting feedback, especially from someone who may study design, usability, etc. could prove invaluable. I would get them involved early and often throughout the development process. The beauty of the Windows Design language, formerly known as Metro, is that it pushes you towards a certain aesthetic sensibility. You should study the recommendations from Microsoft in this regard:

<http://developer.windowsphone.com/en-us/design>

At any rate, even when you're getting creative and thinking through the functionality of your app, you'll miss something. That's why it's good to:

1. Let others see your work, especially those who are not your close friends or family. You want honest feedback. Your family and friends won't want to hurt your feelings if they don't like your work.
2. Use the Beta test feature of the store before you attempt to sell the app. Now you're opening your app up to more scrutiny and feedback, however it's better you hear that feedback in beta rather than hear that feedback in the form of negative reviews that pour in and kill your app's chances at success.
3. Even after you've sold the app (or distribute it for free), be responsive to customer emails, questions, feedback, etc. Whether you realize it or not, once you publish an app

you're now running a business. From now on you need to be the friendliest guy on the planet. If someone asks for a new feature, ask them for more details and genuinely give serious thought to whether that's a viable feature to add or a bug to fix. If someone is frustrated and uses harsh language or tone against your app, you gain absolutely nothing by returning the salvo. You need to grow thick skin, swallow your pride, and be the bigger person. From personal experience in handling customers who are displeased over the years, you can turn an enemy into a friend more often by not by choosing your words carefully. May I recommend the time-honored book, How to Win Friends and Influence People by Dale Carnegie.

4. Dedicate a URL and a few pages to your app. Include FAQ's so people can self-serve for the most common issues.
5. Understand the impact of low-memory on your app. I suspect most app crashes have to do with memory constraints, or the failure to close a resource like the file system when you're finished working with them.

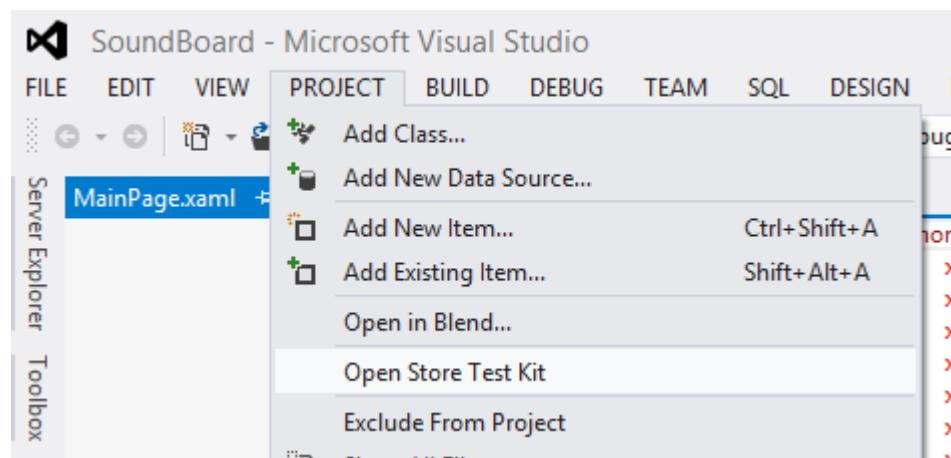
The game plan for this lesson:

1. We'll perform some compliance testing to make sure our app has a chance to be included in the Store
2. We'll submit our app to the store

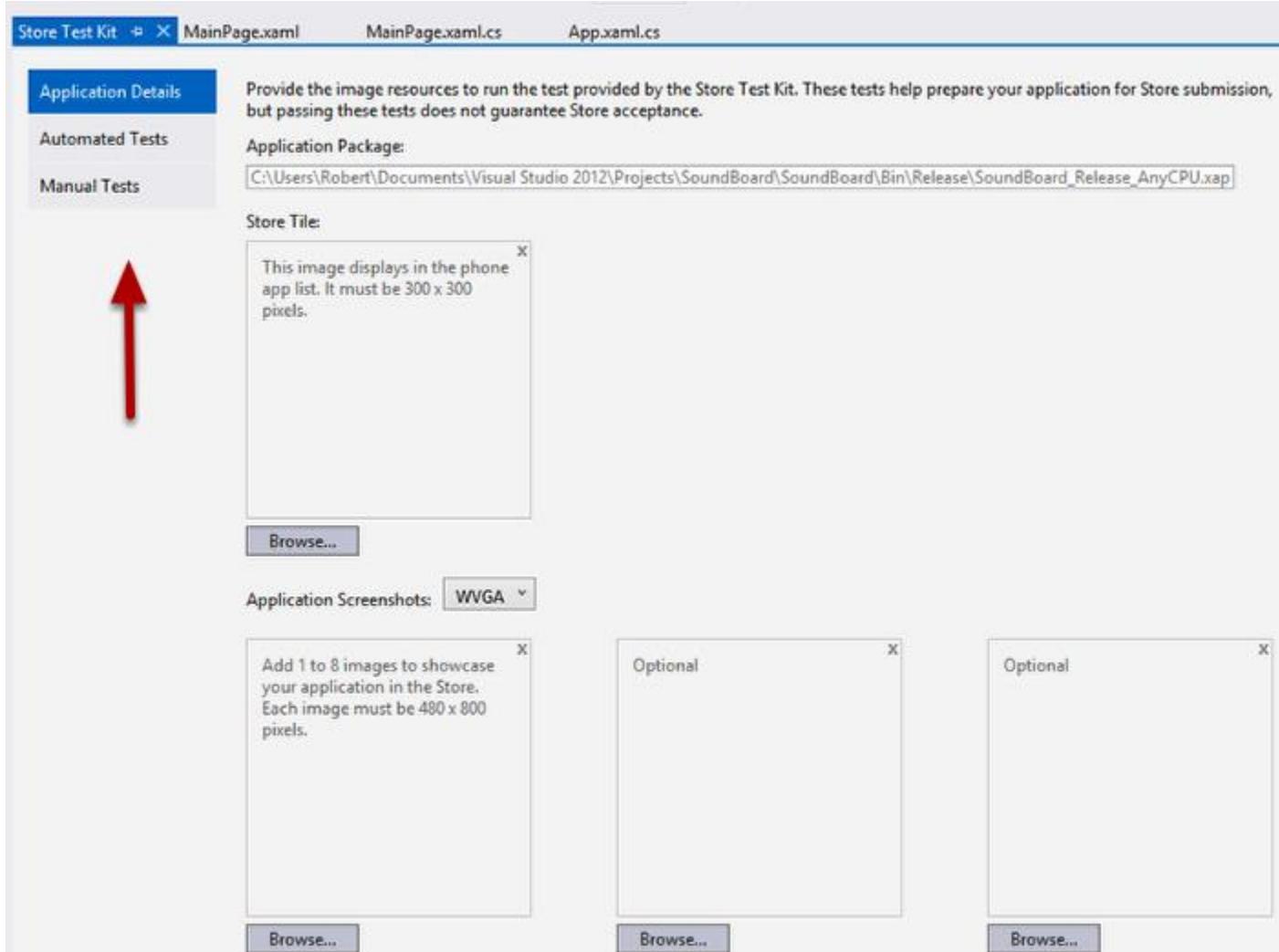
#### 1. Run the Store Test Kit

To evaluate the readiness of our app for the app store, we'll use the Store Test Kit. This will automatically test a few scenarios and suggest other manual tests for you to conduct. Furthermore, it works as a check list for the items you'll need to submit to the store, like promotional images and the like.

To open the Store Test Kit, go to the Project menu select Open Store Test Kit:



This will open a new page in the main area. Notice there are three tabs on the left:



The Application Details collects basic data about the app, including a larger version of the main tile (300 pixels by 300 pixels) for the Store and screenshots of the app. If you use the Phone Store, you can see that each app has a page with descriptive text and details about the publisher, a page with the ratings and reviews for the app, and a page of screenshots.

The Automated Tests tab will perform a few tests to ensure the validity of the XAP deployment package, and will ensure the images (from the Application Details tab) are in place as well.

Store Test Kit MainPage.xaml MainPage.xaml.cs App.xaml.cs

Application Details Click the Run Tests button below to run the automated test cases.

Automated Tests

Manual Tests

Test cases have not been run yet.

Result	Test Name	Test Description
Pending	XAP Package Requirements	Validation of XAP file size and content files
Pending	Iconography	Validation of Application Icons
Pending	Screenshots	Validation of Screenshots

The app is evaluated against the Store submission requirements. Use App Analysis to further analyze the app. Click Start App Analysis to start an analysis session.

The Manual Tests tab has a number of suggestions for you to test on your app before submitting it.

Store Test Kit X MainPage.xaml MainPage.xaml.cs App.xaml.cs

Application Details Pending Required app images

Automated Tests Pending Multiple devices support

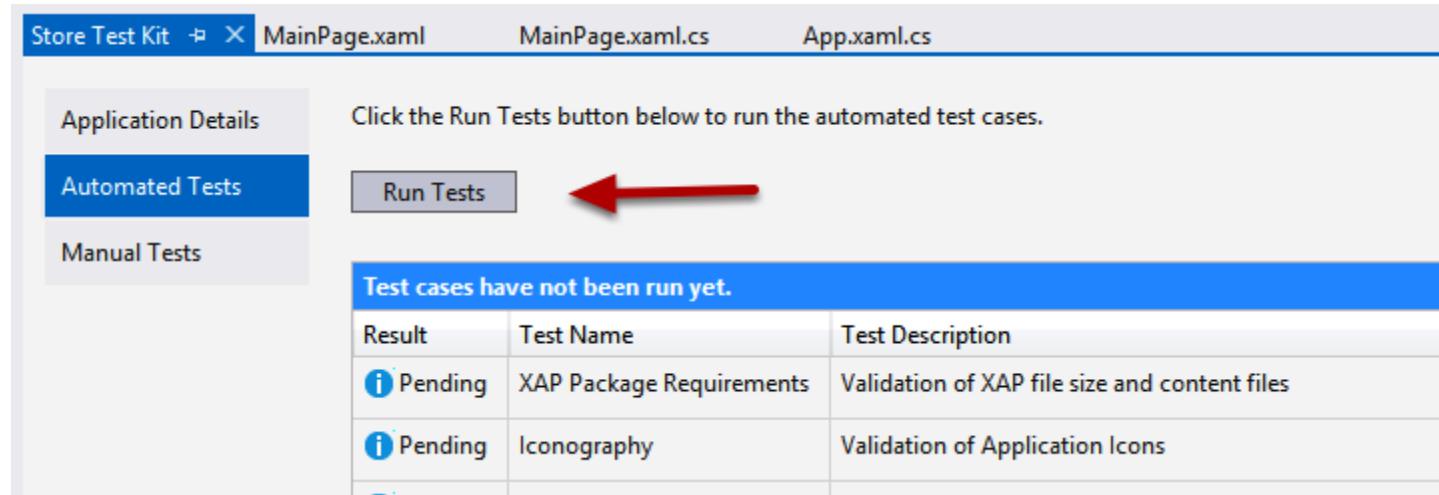
Manual Tests Pending App closure

Pending App responsiveness

To the right of the test status (think of this like a check box that you will manually check off) and the name of the test is the description of the test and how to perform the test, what constitutes failure, etc.

Test Description
<ul style="list-style-type: none"><li>• View the App list.</li><li>• Verify that the App list image is representative of the app.</li><li>• From the App list, tap and hold the App list image and select 'pin to start'.</li><li>• Verify that the default Tile image on the Start screen is representative of the app.</li><li>• If applicable, resize the default Tile and verify that the Tile image is representative of the app.</li></ul> <a href="#">More info...</a>
<ul style="list-style-type: none"><li>• Install your app on two or more Windows Phone devices that are compatible with the app hardware and screen resolution requirements.</li><li>• Verify that the app can install and uninstall without error.</li><li>• After testing the above, ensure your app is installed, and launch it.</li><li>• Comprehensively test app functionality and features to verify that there are no device-specific issues.</li><li>• Verify that the app does not cause the device to stop responding or crash.</li></ul> <a href="#">More info...</a>
<ul style="list-style-type: none"><li>• Launch your app</li></ul>

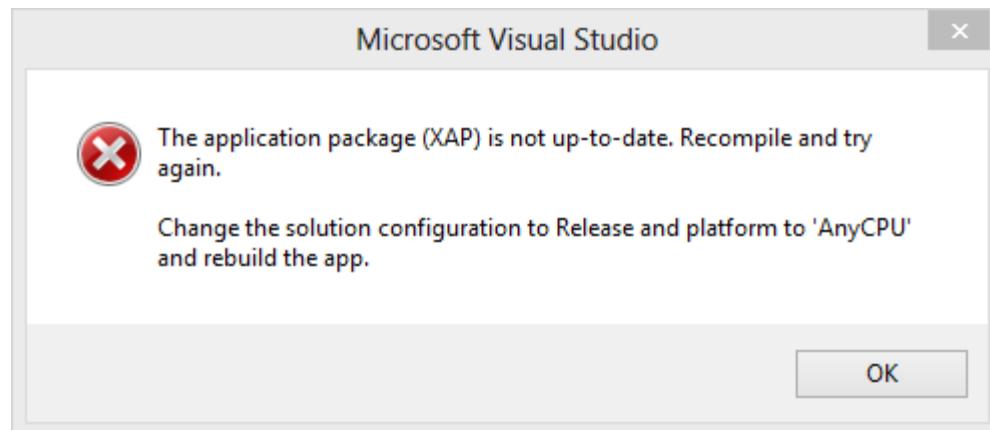
If you attempt to run the tests without changing anything in Visual Studio by clicking the "Run Tests" button on the Automated Tests tab ...



The screenshot shows the 'Store Test Kit' window with three tabs at the top: 'Store Test Kit', 'MainPage.xaml', and 'MainPage.xaml.cs'. The 'MainPage.xaml.cs' tab is active. On the left, there are three buttons: 'Application Details', 'Automated Tests' (which is highlighted in blue), and 'Manual Tests'. To the right of these buttons is a message: 'Click the Run Tests button below to run the automated test cases.' Below this message is a large 'Run Tests' button, which has a red arrow pointing to it from the left. Underneath the 'Run Tests' button is a table titled 'Test cases have not been run yet.' The table has three columns: 'Result', 'Test Name', and 'Test Description'. There are two rows in the table, both of which are marked as 'Pending':

Result	Test Name	Test Description
Pending	XAP Package Requirements	Validation of XAP file size and content files
Pending	Iconography	Validation of Application Icons

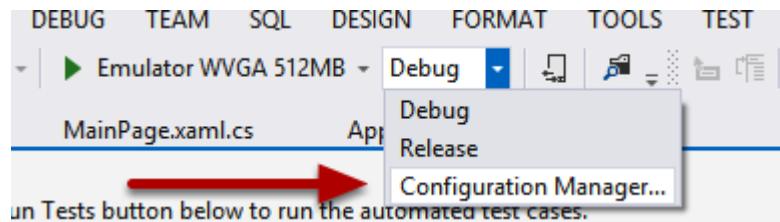
... you'll probably get the following warning / error:



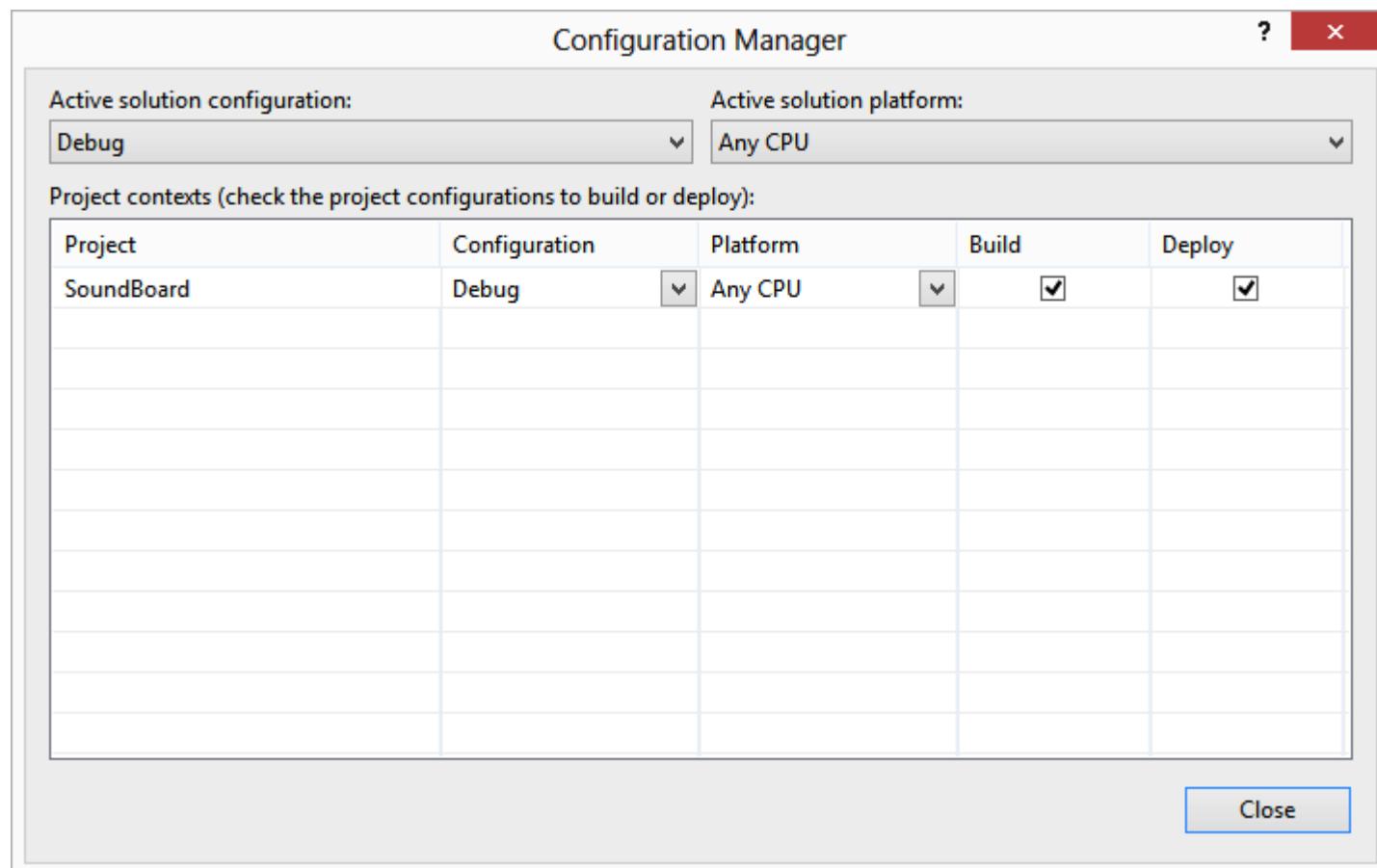
There are several issues we'll need to correct, but the first one is that we need to create a Release version of the deployment package. We've been using a Debug version which includes additional files used to sync up the execution of the code on the Emulator or the Device with Visual Studio so that you can set break points, step through code, and so on.

Now we'll need to create a release version. Before I do that, let's confirm that Visual Studio will create a Release version that is configured to be deployed to "AnyCPU" per

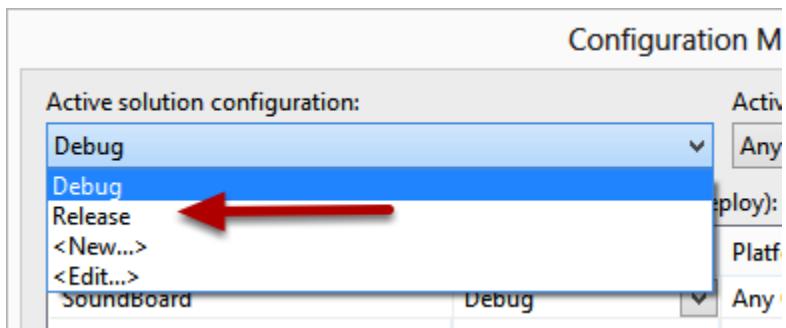
the instructions of this dialog. In the toolbar, I'll select the drop down list next to the Run button and select Configuration Manager:



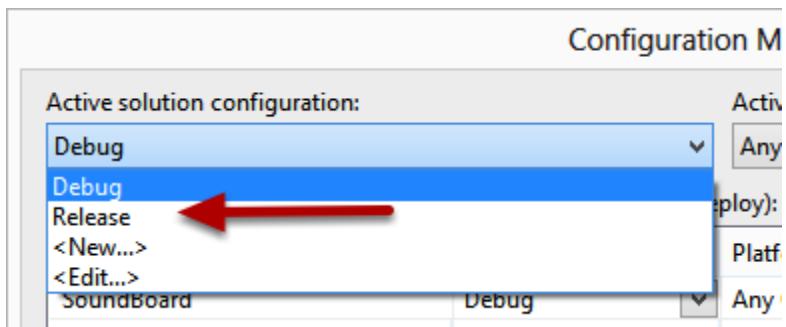
The Configuration Manager dialog appears:



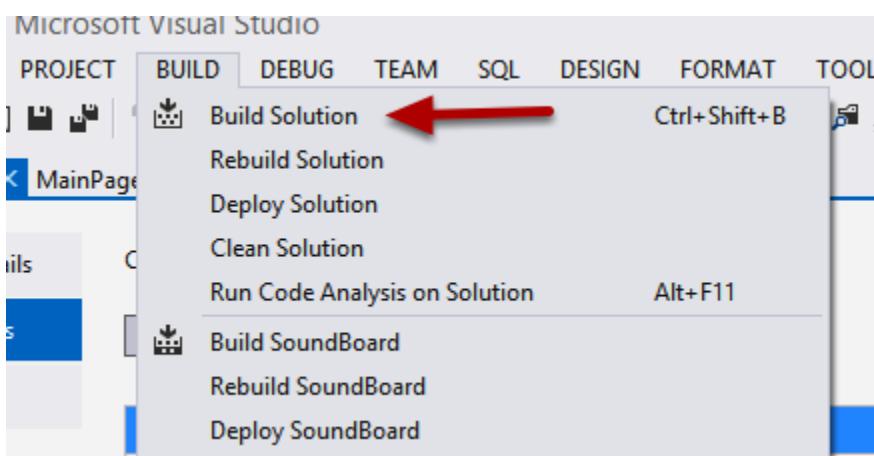
In the Action solution configuration drop down list, I'll select Release ...



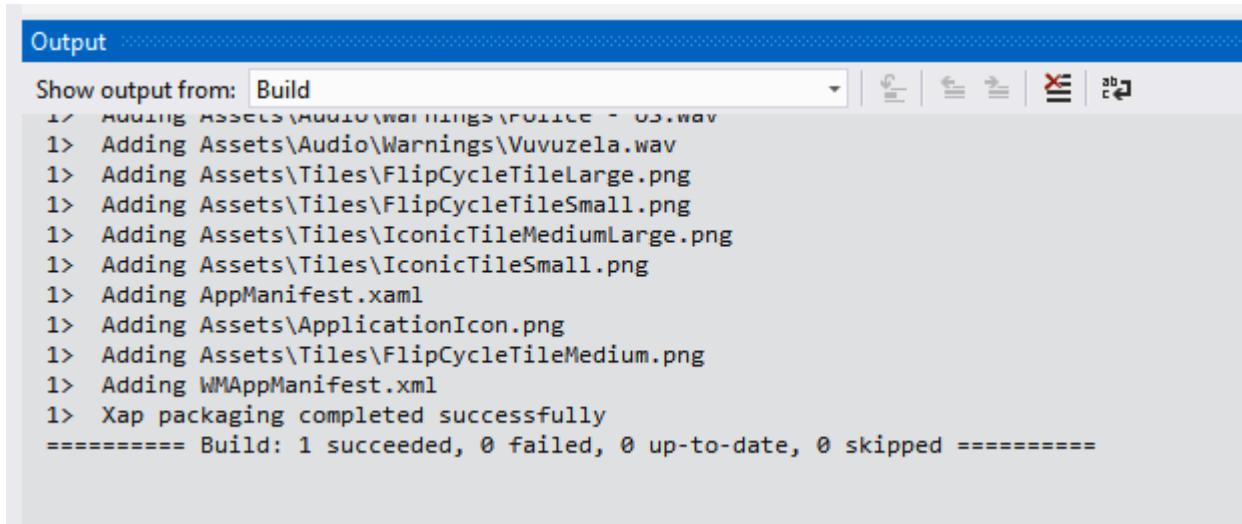
And I'll review the Project contexts in the main area, ensuring that the Configuration is set to Release and the Platform is set to "Any CPU".



If the settings are right (and they should be by default), then I'll Build the Solution (Build menu, select Build Solution).



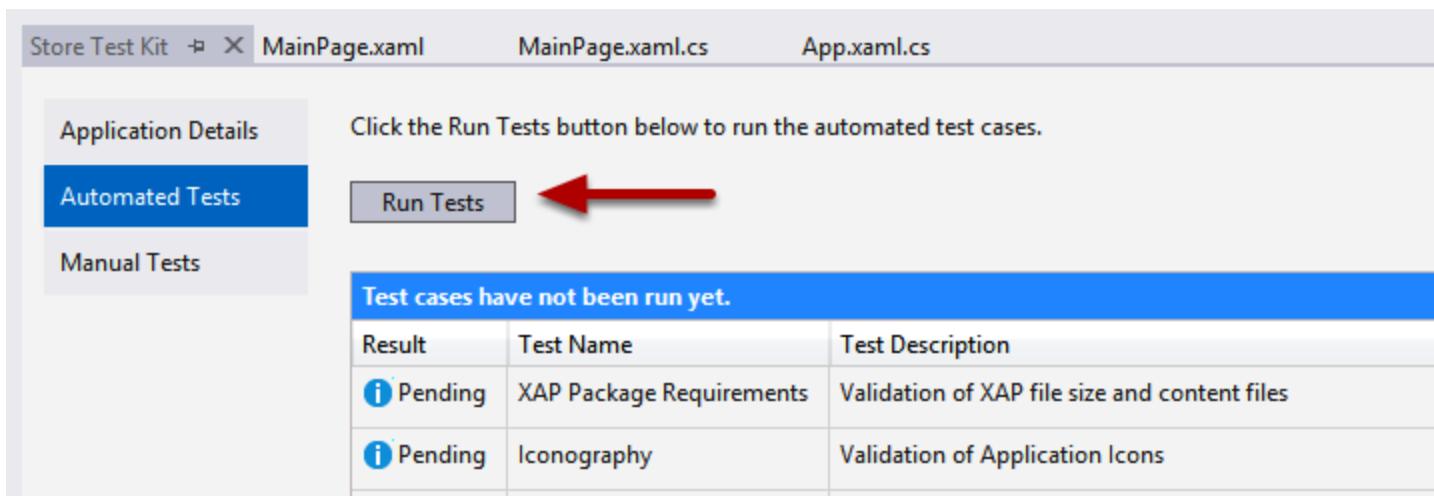
If all goes well, the Output window should indicate success. If it doesn't you'll obviously need to fix that before continuing.



The screenshot shows the Visual Studio Output window titled "Output". The dropdown menu at the top is set to "Build". The main pane displays the following log output:

```
1> Adding Assets\Audio\Warnings\Police - 03.wav
1> Adding Assets\Audio\Warnings\Vuvuzela.wav
1> Adding Assets\Tiles\FlipCycleTileLarge.png
1> Adding Assets\Tiles\FlipCycleTileSmall.png
1> Adding Assets\Tiles\IconicTileMediumLarge.png
1> Adding Assets\Tiles\IconicTileSmall.png
1> Adding AppManifest.xaml
1> Adding Assets\ApplicationIcon.png
1> Adding Assets\Tiles\FlipCycleTileMedium.png
1> Adding WMAppManifest.xml
1> Xap packaging completed successfully
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Now that we have a valid release version of our XAP deployment package, we'll "Run Tests" again ...



The screenshot shows the "Store Test Kit" interface. At the top, there are tabs for "Store Test Kit", "MainPage.xaml", "MainPage.xaml.cs", and "App.xaml.cs". Below this, there are two tabs: "Application Details" and "Automated Tests" (which is selected). A red arrow points to the "Run Tests" button. To the right of the "Run Tests" button is the text "Click the Run Tests button below to run the automated test cases." Below the tabs, there is a section titled "Test cases have not been run yet." containing a table with two rows:

Result	Test Name	Test Description
Pending	XAP Package Requirements	Validation of XAP file size and content files
Pending	Iconography	Validation of Application Icons

... this time, we see that we failed with Iconography and Screenshots.

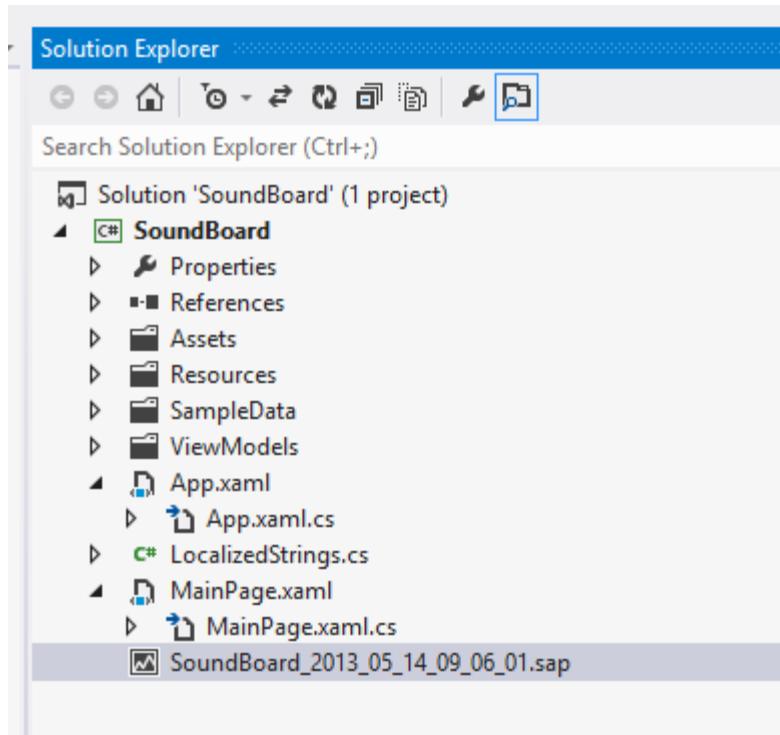
Passed: 1 Failed: 2			
Result	Test Name	Test Description	Result Details
Passed	XAP Package Requirements	Validation of XAP file size and content files	
Failed	Iconography	Validation of Application Icons	[ERROR] : Store application tile is not provided. All icons must be provided.
Failed	Screenshots	Validation of Screenshots	[ERROR] : No screenshot was specified for WVGA. At least one screenshot is required. [ERROR] : No screenshot was specified for WXGA. At least one screenshot is required. [ERROR] : No screenshot was specified for 720P. At least one screenshot is required.

Before I set those, let's take a look at the button further down on that page ... the "Start Windows Phone Application Analysis" ...

The app is evaluated against the Store submission requirements. Use App Analysis to further analyze the app's performance and readiness for Store. Click [here](#) for more details. Click Start App Analysis to start an analysis session.

[Start Windows Phone Application Analysis](#)

Clicking this button will add a new .sap file which is a blank performance log file that will capture data about the performance of your app as it runs. The log file will then be poured over and reported on to let you know how well it runs in certain situations:



That .sap file is loaded into the main area of Visual Studio:

## SoundBoard

Monitoring and profiling your application can help you diagnose performance problems and improve the quality of your application. To begin, choose one of the options below.

### Monitoring (recommended)

- App Analysis (analyzes performance and quality aspects of application)

### Profiling

- Execution (evaluates application performance with advanced visual and code profiling)
  - ▶ Advanced Settings
- Memory (evaluates memory allocation and texture usage)
  - ▶ Advanced Settings

Warning: The app performance observed on the emulator may not be indicative of the actual performance on the device

- 2

[Start Session \(App will start\)](#)

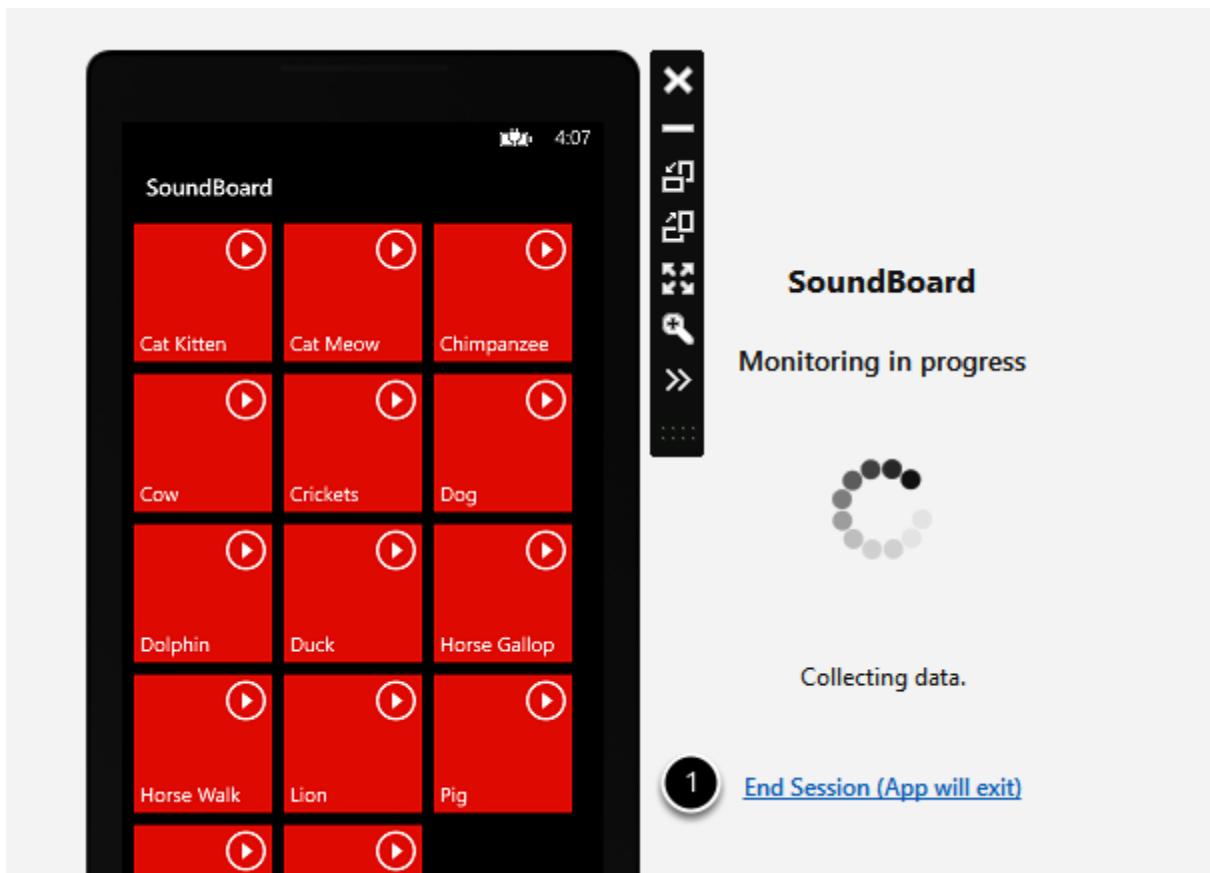
Rather than spend time talking about the different types of data and reports you can generate from the monitoring and profiling options, I would recommend that you read what the help has to say about it:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj215908\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj215908(v=vs.105).aspx)

At this point, I'll merely:

1. Choose the App Analysis monitoring option, then ...
2. Click the Start Session link at the bottom.

The app will load in the Emulator and you'll see Visual Studio display a monitoring message:



At this point you should put the app through its paces and perform your typical use cases, and perhaps even some edge cases. When you're finished testing the app ...

1. Click the "End Session" link

The collected data will be analyzed:

## SoundBoard

Analyzing the data



- ✓ Copying log file to the desktop.
- ✓ Parsing the log file.
- ▶ Analyzing the log file. 

... and a report will be generated ...

## SUMMARY

0 Alert(s)

## REPORT

The different parameters of the app as measured during the analysis session

Startup time 0.54 sec App start time meets requirements.

Responsiveness --- App is responsive.

Total data uploaded 0.00 KB Total data uploaded by app is 0.000 KB

Total data downloaded 0.00 KB Total data downloaded by app is 0.000 KB

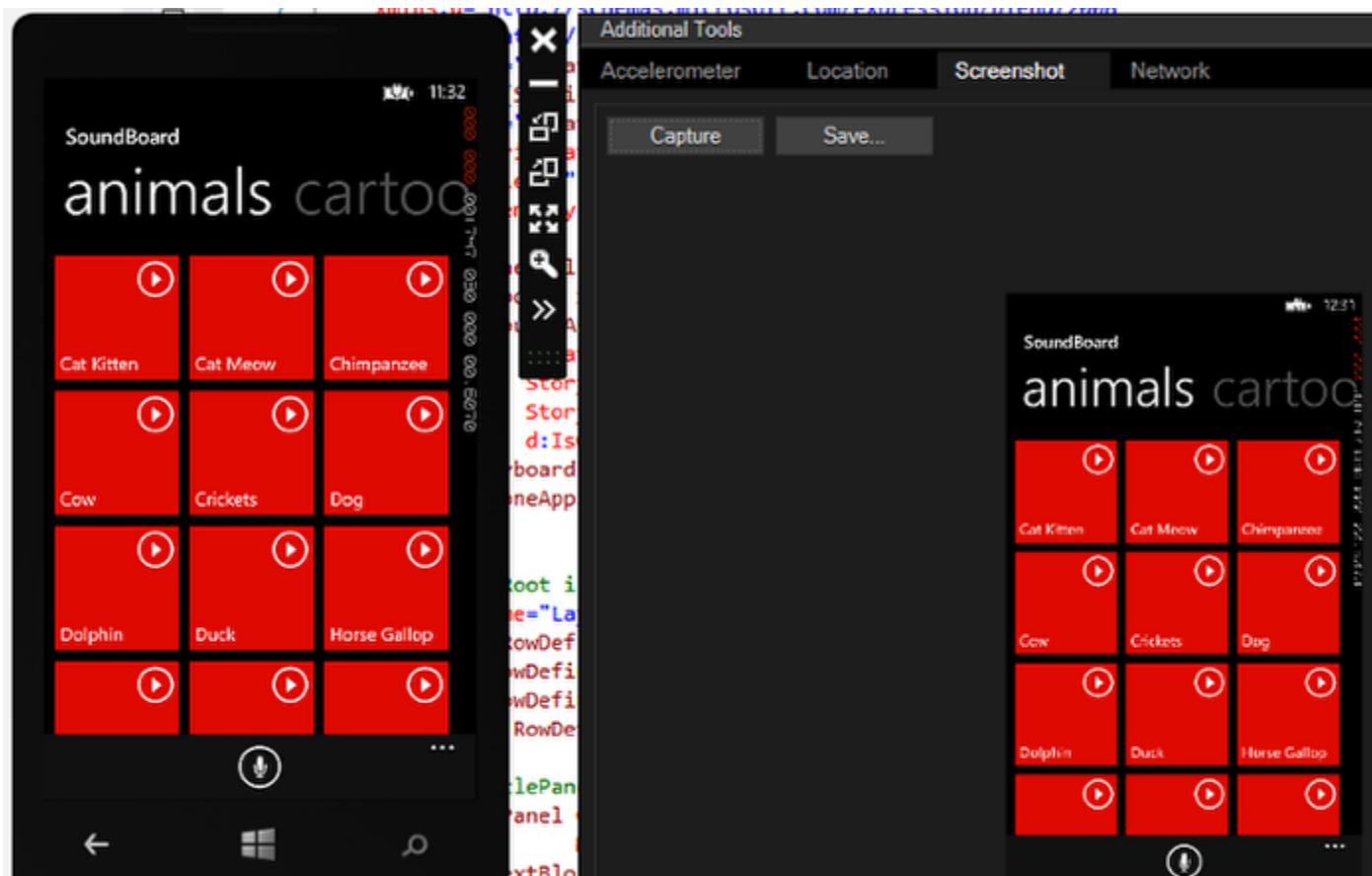
Battery charge remaining 13.15 hours The session consumed 0.98 mAh of battery charge in 31.04 secs. This rate of usage will

Max memory used 13.71 MB App max memory usage is 13.71 MB

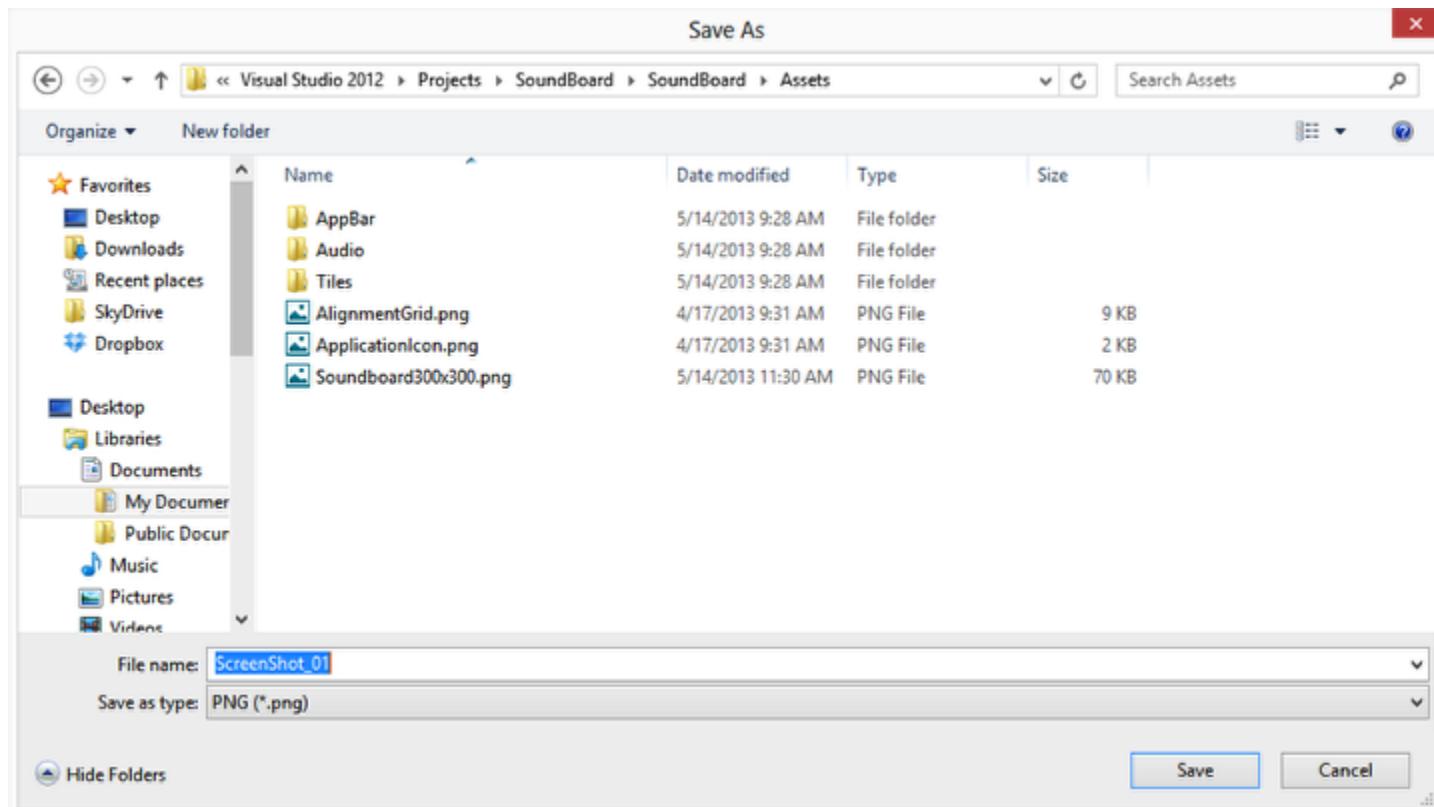
Average memory used 11.92 MB App average memory use is 11.92 MB

The report includes some very interesting details about the app that could help you determine its readiness for the Store. I think one of the most important things to look at is the battery consumption and the max memory used.

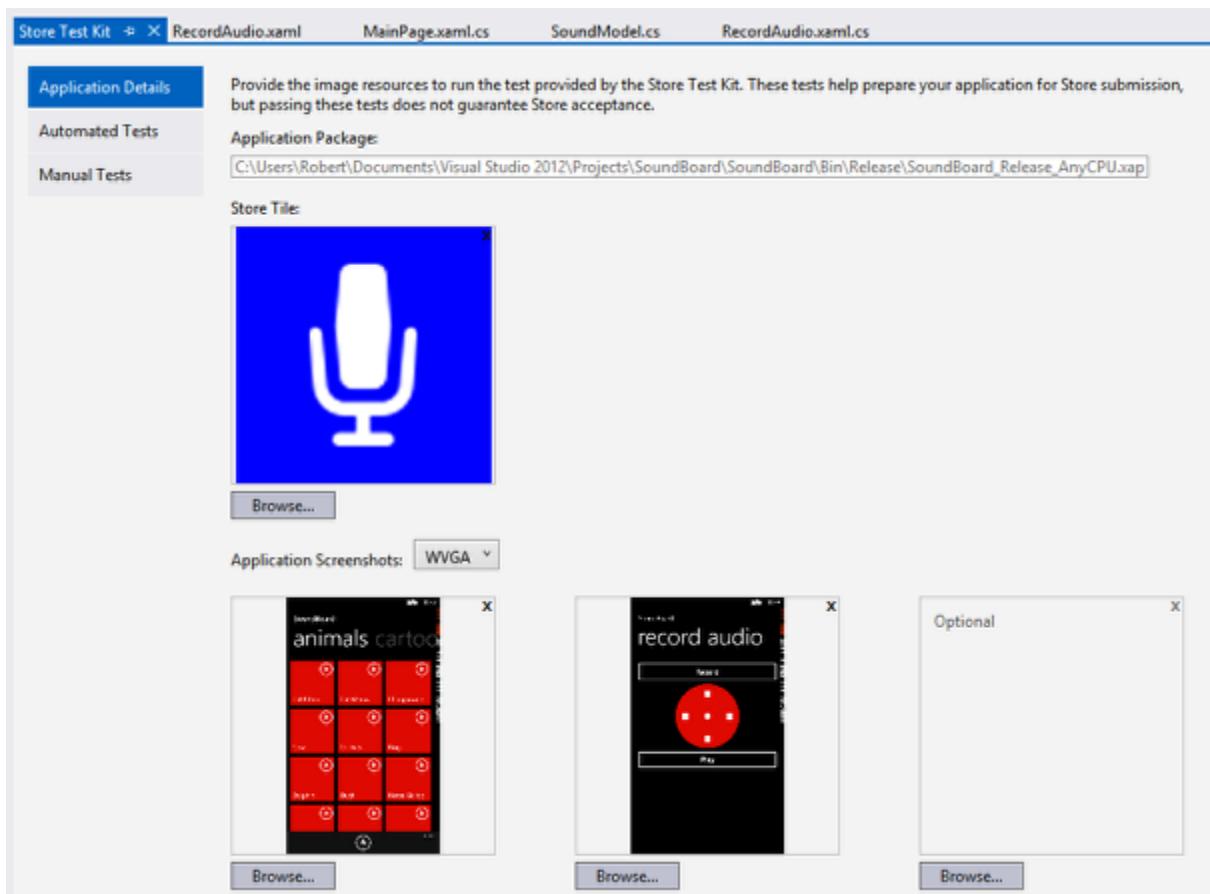
Now that we've completed this test, I'll make sure to ready the images needed for the Store, such as the Store Tile (300x300) and the screenshots. I use the Emulator's Additional Tools to capture two main views of the SoundBoard app:



And I've chosen to save them into the Assets folder of the project. However, in retrospect, I probably would have put them somewhere else ... perhaps another subfolder of the project so I could get back to them more easily and have less confusion about the names and such.

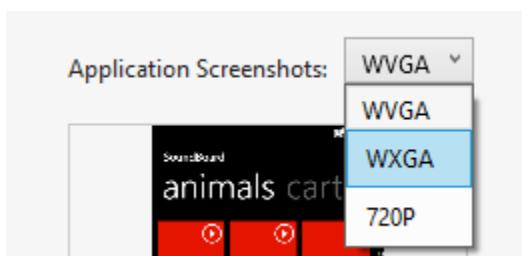


I add the two screenshots to the Application Screenshots section of the Store Test Kit:



However, if you were to run the tests again, it will still fail because you need to provide versions for ALL THREE RESOLUTIONS.

In the Application Screenshots section of the Application Details page, use the drop down to reveal the other screen resolutions and populate them with screen shots using the different versions of the Emulator to capture screen shots at those resolutions.



TIP: When I went to upload the images to the Store, it provided an option to create / scale the various versions of these files based on the WXGA version. However, if getting the Store Test Kit to pass successfully is important to you, then you'll need images for each of the three resolutions.

After supplying all of the required files, I was able to get the automated tests to pass:

Passed: 3 Failed: 0		
Result	Test Name	Test Description
Passed	XAP Package Requirements	Validation of XAP file size and content files
Passed	Iconography	Validation of Application Icons
Passed	Screenshots	Validation of Screenshots

Again, I still have manual tests to consider, as well as my own internal testing I want to perform, but at least this helped me get the assets and the release version of the app ready for submission.

## 2. Submitting the app to the Store

Now I'll attempt to publish my app. This is a multi-day process. I'm not 100% confident that the app will pass the first time. However, I do want to give you an idea of the submission process. It's not scary at all.

It's also important to keep in mind that this site changes often, and it would be impossible to keep up with the changes. So, what you see when you attempt to do this may be dramatically different. You should always read the on-page instructions for yourself. The purpose of this lesson is to merely show you the general process up front before you actually build your app.

Go to: <https://dev.windowsphone.com>

Windows Phone | Dev Center

Design Develop Publish Community Dashboard

Search Dev Center with Bing

bob@learnvisualstudio.net | Sign out

1  
人脉  
3  
1

Publish  
Share your app with the world.

2

SUBMIT APP  
GET SDK  
VIEW SAMPLES

Join the program  
Submit your app  
Renew your account  
Getting paid

Join the program  
Submit your app

Registration info  
Renew your account  
Getting paid

Windows Phone interface screenshot

Windows Phone interface screenshot

1. Click the Publish menu option at the top of the page,
2. Click the Submit App link on the page

That will take you to the Submit app page:

The screenshot shows the Windows Phone Dev Center interface. At the top, there's a navigation bar with links for 'Design', 'Develop', 'Publish', 'Community', and 'Dashboard'. On the right side of the header, there's a user profile for 'bob@learntocreate.com'. Below the header, the main content area has a title 'Submit app'. Underneath the title, a sub-section titled 'Required' lists two steps: '1 App info' and '2 Upload and describe your XAP(s)'. Step 1 is highlighted with a purple background, while step 2 is greyed out.

## Submit app

You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.

Required

- 1** App info  
Give your app a Dev Center alias, price it, and enter other relevant info
- 2** Upload and describe your XAP(s)  
For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.

This page contains a series of tasks, many of which are optional depending on the features of your app and how you hope to monetize it.

1. Click on the App info task

That will take you to the App info page ...

The screenshot shows the Windows Phone Dev Center interface. At the top, there's a navigation bar with links for Design, Develop, Publish, Community, and Dashboard. A user profile for 'bob' is visible on the right. Below the navigation, the title 'Windows Phone | Dev Center' is displayed. The main content area is titled 'App info'. It includes sections for 'App alias\*' (set to 'SoundBoard'), 'Category\*' (set to 'entertainment'), and 'Subcategory' (a dropdown menu). Under the 'Pricing' section, the 'Base price\*' is set to '0.00 USD'. There's also a checkbox for offering free trials.

## App info

The info on this page is used to refer to your app here in the Dev Center, and also controls how it appears in the Store.

### App info

**App alias\***  
This name is used to refer to your app here on Dev Center. The name your customer sees is read directly from your XAP file.

SoundBoard

**Category\***

entertainment

**Subcategory**

0.00 USD

Offer free trials of this app. Before you select this option, make sure you've implemented a trial experience in your app. [Learn more](#).

Here you'll name, categorize, price and choose who you want to distribute the app to. Most of this is self explanatory. There's a lot to consider here, and honestly your choices here reflect your strategy ... who you will target the app to? What is your pricing strategy? If you are going to charge for your app (as opposed to a freemium or advertising model) then will you allow free trials? In some cases, this would make sense. In other cases, it might make less sense. I hear that, in general, trials do help sales but I'm sure that depends on the type of app and your target audience.

Further down on that page ...

https://dev.windowsphone.com/en-us/AppSubmission/Application

Preparin... 8 Must... InTheK...

## Market distribution

- Distribute to all available markets at the base price tier
- Distribute to all markets except China. [Learn more.](#)
- Continue distributing to current markets

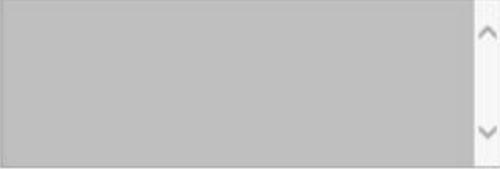
## More options ▲

### Distribution channels

- Public Store
  - Hide from users browsing or searching the Store
- Beta

Choosing Beta allows you to distribute your app to up to 10,000 people for testing. When you're ready to publish your app in the Public Store, you'll need to resubmit it as a new app.

Enter Microsoft account email addresses for beta participants, separated by semi-colons.



You can choose which markets to distribute to, and whether you want to start in the Store or in a beta as we described earlier.

Even further down that page ...

## Publish

- Automatically, as soon as it's certified
- Manually, any time after it's certified

### MPNS certificate

[Learn more](#) about the benefits of authenticated push notifications and MPNS certificates.

Save

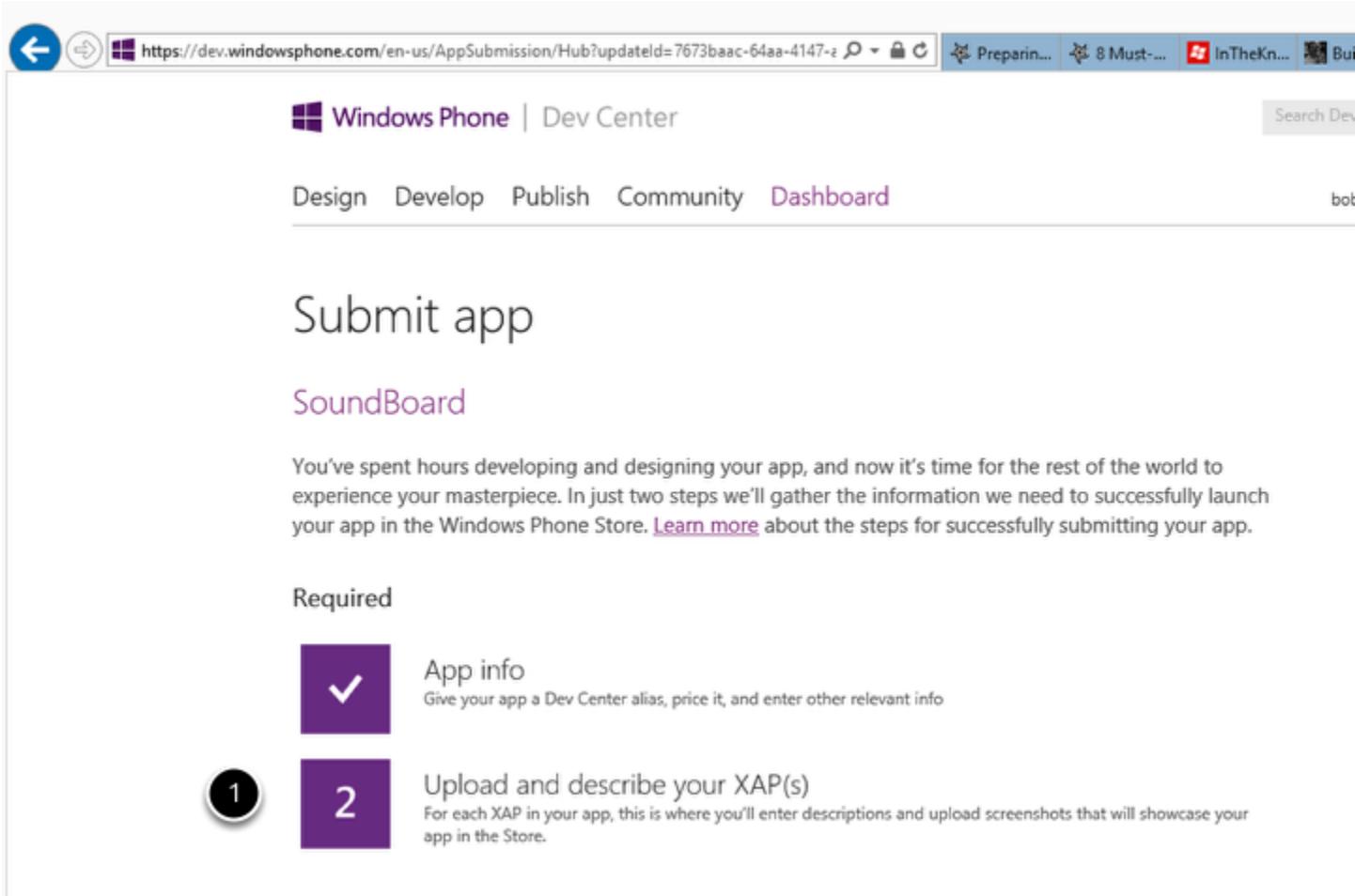
You can choose whether to put it in the store automatically after it has been certified, or whether to

The final option is for MPNS, or rather, the Microsoft Push Notification Service for Windows Phone. This allows you to send messages from a server to a user's phone securely. Imaging a long running massively multi-player casual online game ... when the player's home planet is about to be attacked by another player, the game could notify that an attack is eminent. This might be a premium upgrade in the game. It would give the player a chance to move forces into place. This service costs money, so you need to use it judiciously. For more information, check this out:

Push notifications for Windows Phone

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558(v=vs.105).aspx)

After you click the Save button (above) you'll return to the Submit app page with the first task marked as complete:



The screenshot shows the Windows Phone Dev Center at the URL <https://dev.windowsphone.com/en-us/AppSubmission/Hub?updatedId=7673baac-64aa-4147-z>. The page title is "Windows Phone | Dev Center". The navigation bar includes links for Design, Develop, Publish, Community, and Dashboard. A search bar is located in the top right corner. The main content area is titled "Submit app" and features a sub-section for "SoundBoard". A message encourages users to submit their app, mentioning two steps: "App info" and "Upload and describe your XAP(s)". Both steps are marked as required.

Windows Phone | Dev Center

Design Develop Publish Community Dashboard

Search Dev

## Submit app

### SoundBoard

You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.

Required

1 App info  
Give your app a Dev Center alias, price it, and enter other relevant info

2 Upload and describe your XAP(s)  
For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.

1. Click task 2 to upload and describe the XAP file

That will take you to the Upload and describe your XAP page ...

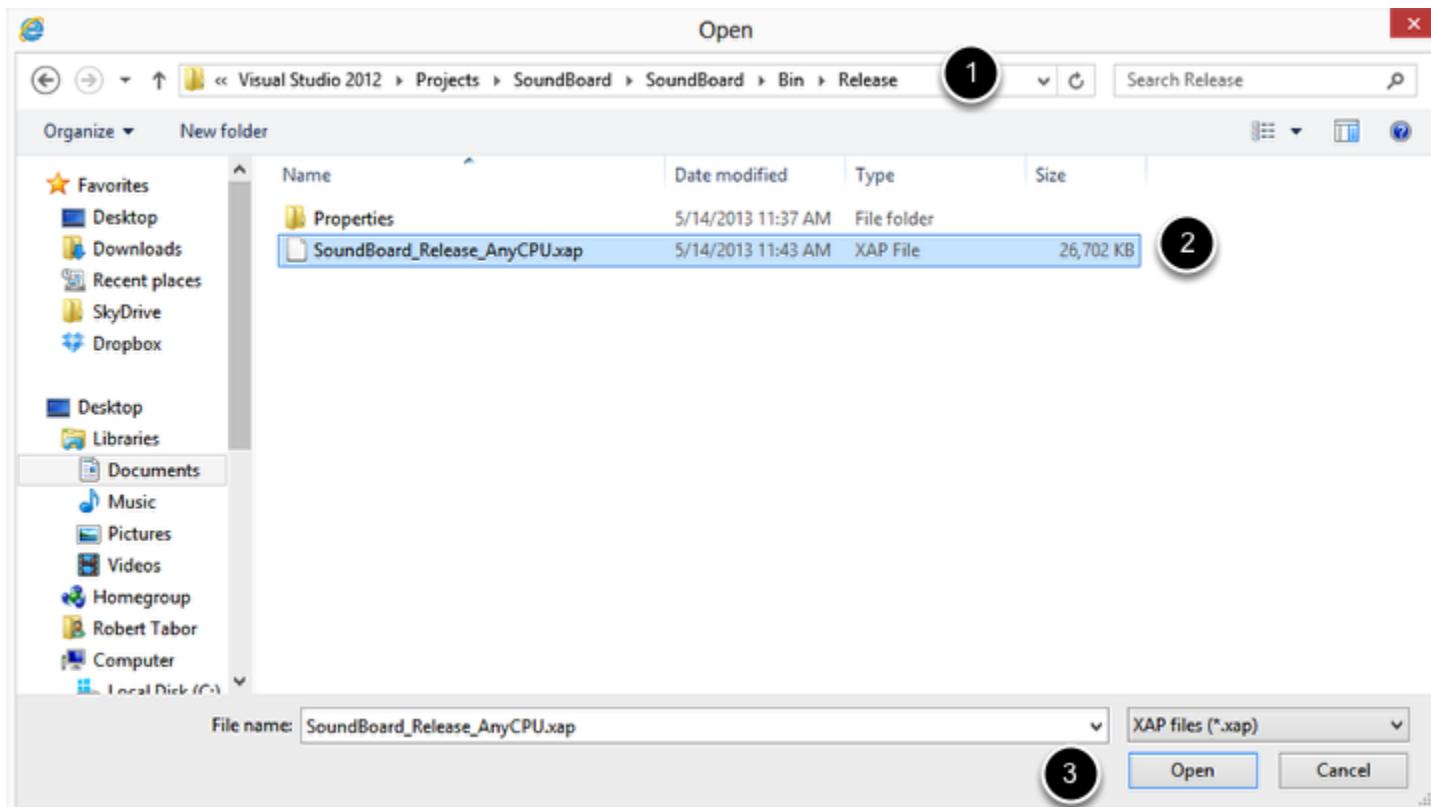
The screenshot shows a web browser window with the URL <https://dev.windowsphone.com/en-us/AppSubmission/Versions?updateId=7673baac-64aa-41>. The page title is "Windows Phone | Dev Center". Below the title, there is a navigation bar with links: Design, Develop, Publish, Community, and Dashboard. The main content area has a heading "Upload and describe your XAP SoundBoard". A sub-section titled "XAPs" is described with the text: "This is an important page, because in addition to uploading your XAP, you're also creating your customer's first impression of your app. The info you provide will be part of the Store's listing of your app. If you're updating an existing app, this page will also include XAPs that you've already uploaded. All these XAPs will be available in the Store after you've published your submission." At the bottom of this section is a button labeled "Add new". To the right of the "Add new" button is a "Save" button.

1 Add new

Save

1. Click the Add new link ...

This will display the Open dialog:



1. Navigate to the RELEASE version of the XAP in the project's Bin\Release directory.
2. Select the version with the suffix: \_Release\_AnyCPU.xap
3. Click Open

It may take a few moments to upload your XAP, especially if you have a lot of image or sound resources embedded in the XAP ...

# Upload and describe your XAP

SoundBoard

If your app contains more than one here, [Learn more](#).

XAPs

Uploading file...

SoundBoard\_Release\_AnyCPU.xap

- - - - -

Cancel

This is an important page, because in addition to uploading your XAP, you're also creating your customer's first impression of your app. The info you provide will be part of the Store's listing of your app. If you're updating an existing app, this page will also include XAPs that you've already uploaded. All these XAPs will be available in the Store after you've published your submission.

Once the XAP is uploaded, you'll see that it is analyzed and its details are displayed:

## XAPs

This is an important page, because in addition to uploading your XAP, you're also creating your customer's first impression of your app. The info you provide will be part of the Store's listing of your app. If you're updating an existing app, this page will also include XAPs that you've already uploaded. All these XAPs will be available in the Store after you've published your submission.

XAP name	Version	OS	Resolution	Language		
● SoundBoard_Release_AnyCPU.xap	1.0.0.0	8.0	WVGA, 720P, WXGA	English	<a href="#">Replace</a>	<a href="#">Delete</a>

[Add new](#)

Select a XAP above to view or edit its Store listing and other info.

XAP version number\*

1 . 0 . 0 . 0

[More XAP options ▾](#)

XAP details detected from file [▲](#)

File name	SoundBoard_Release_AnyCPU.xap
File size	26701 KB
Supported OS	8.0
Resolution(s)	WVGA 720P WXGA

Further down that page, you can modify the XAP's Store listing info ...

## XAP's Store listing info

1 XAP file can contain multiple languages. Select a language to add Store listing info specific for that language.

English



2

Description for the Store\*

Never have another dull conversation! The Channel9 SoundBoard boasts dozens of built in sounds that you can play to liven up even the most boring chats. Sounds in several categories ... Animals, Cartoons, Taunts and Warnings. PLUS you can record and name your own sounds, too!

1721 characters remaining.

3

Specify keywords

Used as exact words or phrases in the Store's search to help people find your app

soundboard

sounds

channel9

sound board

quack



Upload images\*

Use this link to upload all the images in one step, or upload them separately by clicking on each image

1. For each language you support, you'll add ...
2. A description for the app, and ...
3. keywords your potential users might search for to find your app

I would recommend that you give careful thought to your descriptive text (as well as your screenshots). The effectiveness of your writing will either convince someone to give your app a try or move on to another app.

Even further down on that page you'll be able to upload images for the Store ...

## Upload images\*

Use this link to upload all the images in one step, or upload them separately by clicking on each image placeholder below.

[Upload all](#)

\* App tile icon 300 x 300 px



1

[Remove](#)

Background image 1000 x 800 px



2

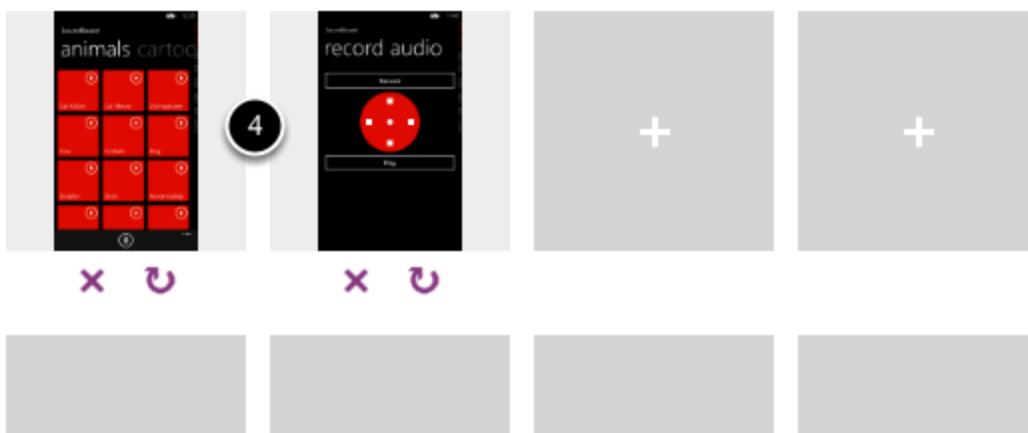
Automatically create lower resolution screenshots from WXGA

3

WXGA | 720p | WVGA

\* We'll use these screenshots to showcase your app.

For WXGA they must be 768 x 1280 px or 1280 x 768 px. [Learn more](#).

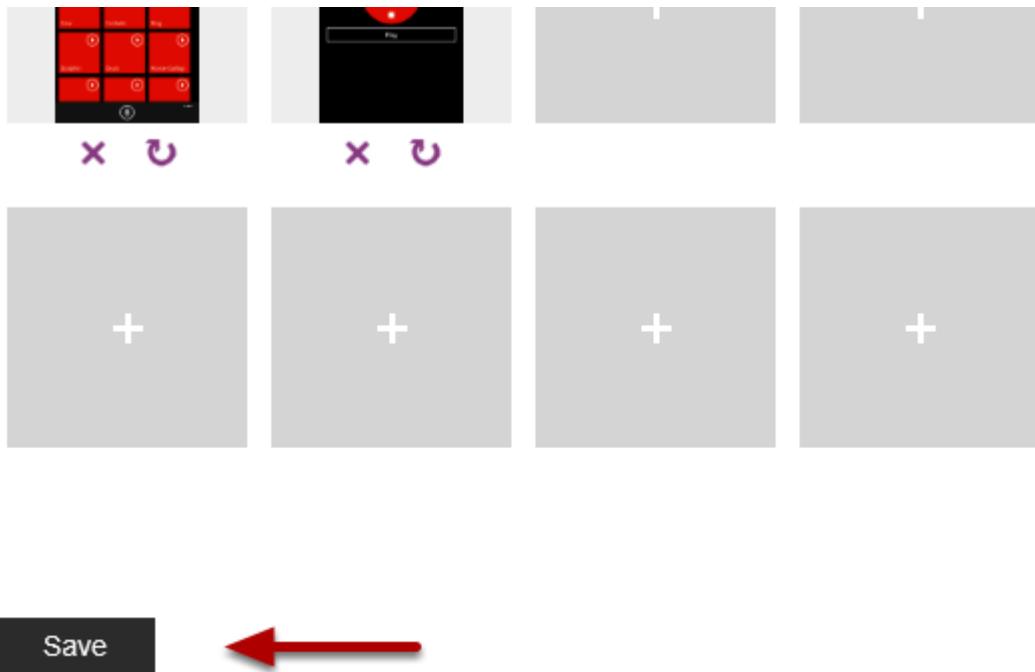


To add an image, you click on the plus symbol in the middle of a given gray square. This will display the Open dialog where you can choose your file.

1. As you can see, I've already selected the 300 x 300 pixel store tile.
2. You can add a background 1000 x 800 that will be displayed for your app. This helps to brand your app and create excitement about the app.

3. I mentioned this earlier ... you can either supply screen shots at each of the screen resolutions you'll support, or you can choose to allow the Store to automatically generate / scale versions of your images from the WXGA version. I chose that in this example.
4. Here I uploaded the WXGA version of the screen shot images.

Even further down the page ...



... I click the Save button. This will return me to the Submit app page with the second task checked as complete ...

# Submit app

## SoundBoard

You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.

### Required



#### App info

Give your app a Dev Center alias, price it, and enter other relevant info



#### Upload and describe your XAP(s)

For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.

Further down that page, you see a number of optional tasks. In our case, we're not using Microsoft's in-app advertising platform, nor will we worry about customizing the pricing on a per-market basis, nor will we need to enable Map service which will provide some API credentials for use in our app. This last option, however, will be important for the next app we'll build. Just put a peg in that idea and we'll revisit it later in this series.

## Optional



### Add in-app advertising

Getting paid through ads? It's all here.



### Market selection and custom pricing

For apps, you have the option to define different pricing and availability for different countries/regions.



### Map services

Get the token required to use map services in your app.

Review and submit



With the two required tasks complete, I click the "Review and Submit" button.

This will display the Review Submission page ...

Passed: 1 Failed: 2			
Result	Test Name	Test Description	Result Details
Passed	XAP Package Requirements	Validation of XAP file size and content files	
Failed	Iconography	Validation of Application Icons	[ERROR] : Store application tile is not provided. All icons i
Failed	Screenshots	Validation of Screenshots	[ERROR] : No screenshot was specified for WVGA. At least [ERROR] : No screenshot was specified for WXGA. At leas [ERROR] : No screenshot was specified for 720P. At least

The Review Submission page is your last chance to catch an error before you submit your app to the Store for review.

Nearer to the bottom of that page, you can see the option to Submit or "Go Back and Edit". I'll click the Submit button ...

Markets - 190 total Edit

## XAPs

XAP name	Version	OS	Resolution	Language
 SoundBoard_Release_AnyCPU.xap	1.0.0.0	8.0	WVGA, 720P, WXGA	English

[Submit](#)

[Go back and edit](#)

... and this will give me the message that the App submission was successful ...

## App submission successful!

### SoundBoard

If your submission needs to be tested, it will take us up to 5 business days to make sure it meets your certification requirements. If not, within a few hours, you can either publish it yourself or it will go live automatically if you chose to automatically publish it. We'll send you an email once it's been processed or if we have additional questions.

Would you like to...

[Submit another app](#)

[Go to the Lifecycle page](#)

[Go to the Dashboard](#)

As the text says, it takes up to 5 business days and you'll be notified by email whether your app is certified for inclusion in the store or not.

Now we wait.

## Recap

To recap, the main take away of this lesson was the thought process of testing, submitting, promoting and supporting your app. There are some tools like the Store Test Kit that can help you get your assets collected and ready for submission, but beyond that you'll want to get your app into the hands of others to try it and provide feedback throughout the entire development process, as well as create a beta program. These steps, along with good artwork, screenshots, descriptions and responsiveness to users after the sale help ensure the success of your app in the Store.

# Part 24: Getting Started with the AroundMe Project

**Source Code:** <http://aka.ms/absbeginnerdevwp8>

Now that we've finished our first app, let's move on to building a second app. This time let's build an app that is both location-aware and social in nature... so, it will use the Phone's GPS to determine where the current user is standing, and then use the latitude and longitude to search for photos that others have taken within a few meters of where the user is standing.

How will we do that? We'll call on Flickr, the social photo sharing website, to provide the data. Whenever you take a photo with a camera that supports Geotagging, extra meta data is added to the photo that includes the latitude and longitude (amongst other) data. Then, when you upload the photos to Flickr, it will store that extra data and make it available so that others can search based on a specific latitude and longitude.

Our app should not only allow me to search for photos "around me", but it should be able to let me pick my favorite Flickr photos as lock screen background images. That would be cool!

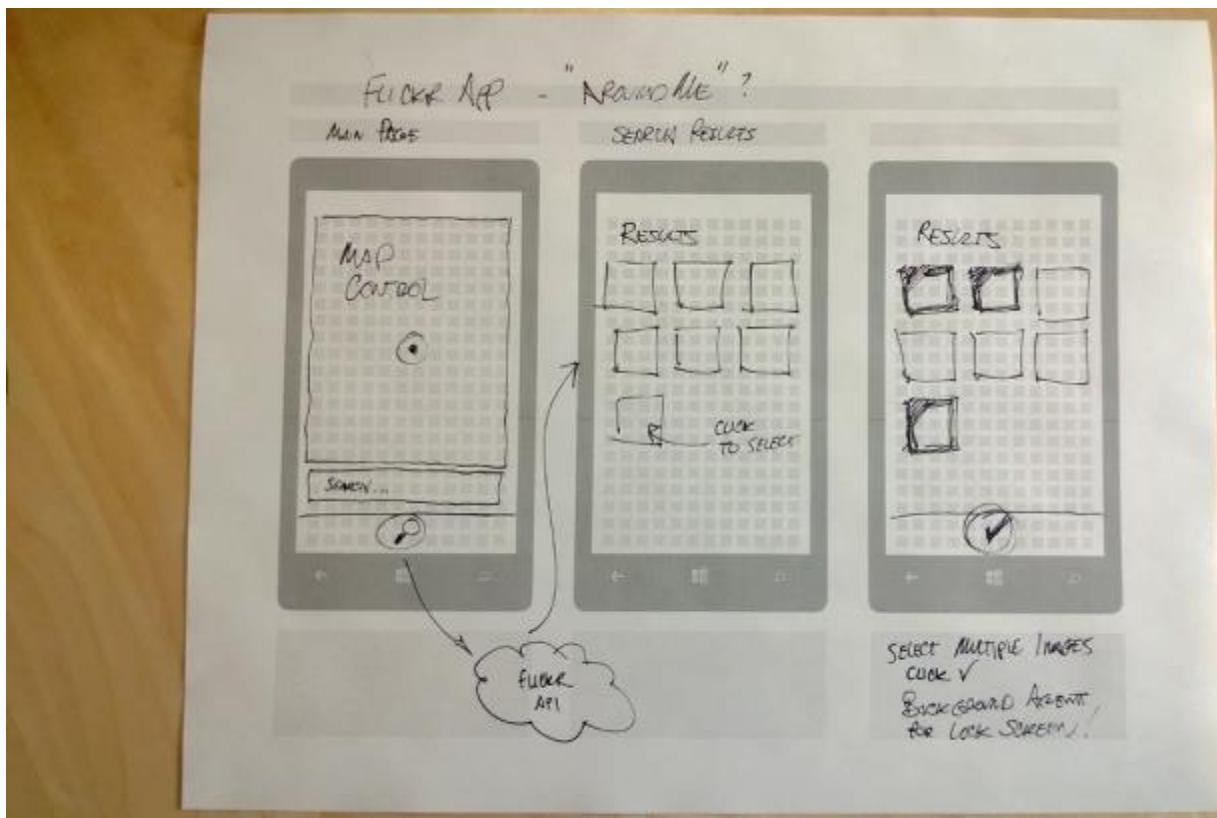
Now that we have some ideas of how it should work, let's spend a few moments mocking up how we want to app to behave. That will be the target that we build to throughout the remainder of this series.

Our game plan in this lesson:

1. We'll start by brainstorming using a low-tech mockup to design the interactions and screens we want in our app.
2. We'll create a new project and take some preliminary steps towards building our app by including a map control in our project ... Even though we haven't designed the app just yet, I anticipate that the map control will come into play at some point so we'll learn how to use it in this lesson.

1. Create a low-tech mockup for the AroundMe app.

Again, we should spend some time thinking about the functionality of the app, about the interaction between the user and the app, how the information should be displayed to the user and we'll do that through a "low-tech mockup" using the templates available from Microsoft:



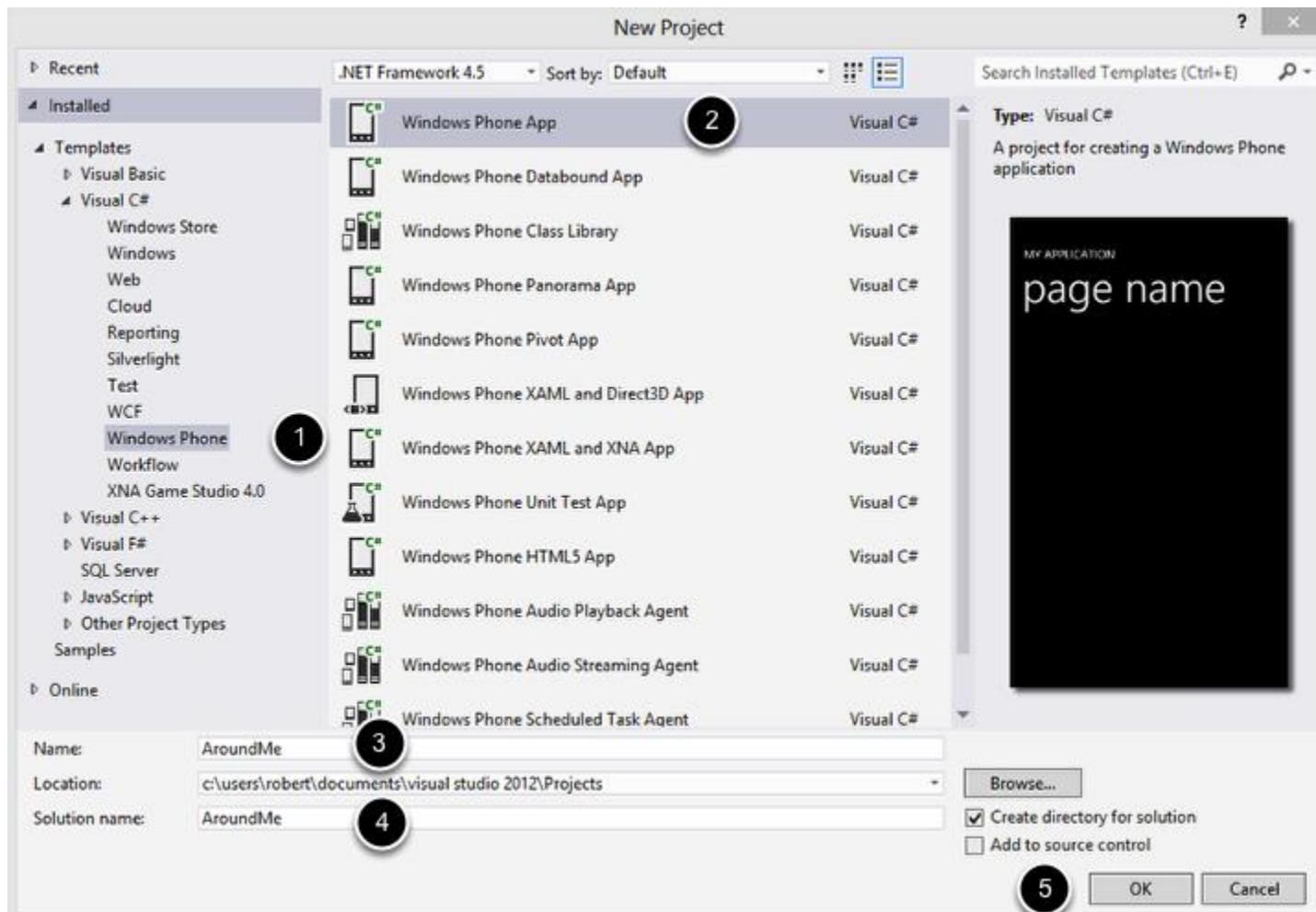
After some brainstorming, we come up with some good design ideas. The MainPage will show the user the map control that displays the user's current location. Below that, a Textbox where the user can add an optional search phrase to refine which photos are returned. And below that, an application bar with a search button.

When the user clicks the search button, we'll make a call to Flickr's web-callable API using the user's current locale and the search phrase. We'll get back a list of images that we'll display in a grid. The user can select photos and click a button in the application bar which will cache those photos on the phone and randomly use them as the lock screen image. This will require we use a background agent to run every 30 minutes or so.

Now that we have a basic idea of the interactions, functionality and screen elements we'll need, let's start building towards that design.

## 2. Create the AroundMe Solution and Project

You should be familiar with this process already, however for the sake of completeness, begin the process by going to the File menu, New | Project ... submenu. That will open the New Project dialog:

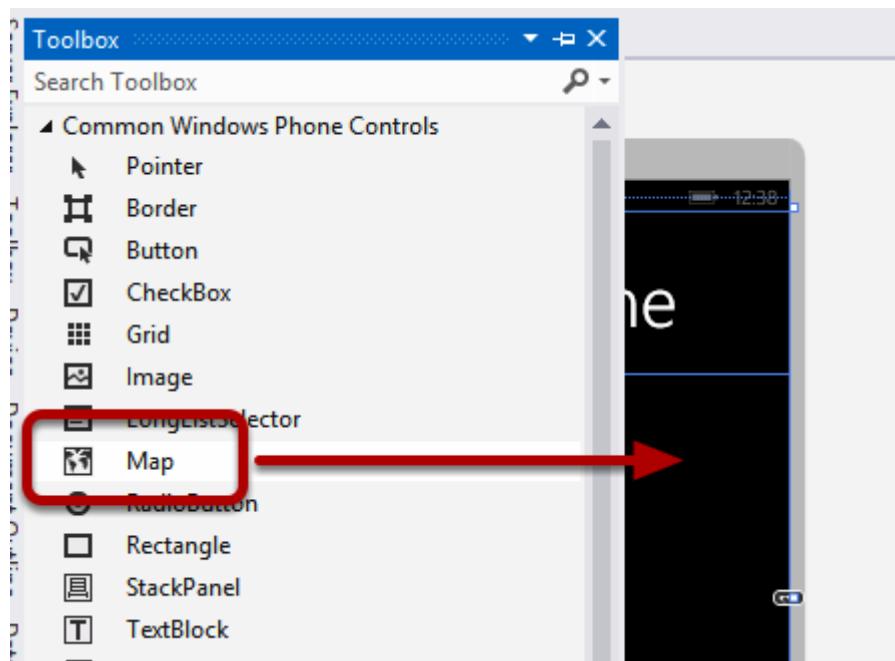


1. Make sure you're in the Visual C# | Windows Phone project templates.
2. Select the "Windows Phone App" project template.
3. Change the name to AroundMe.
4. The Solution name should automatically change as well ... just make sure it has already been changed to AroundMe.
5. Click OK.

The MainPage.xaml should open in the main area of Visual Studio.

3. Add a Map Control from the Toolbox to the visual XAML Editor

Up to now, we've been working with XAML directly. Now that you're comfortable with typing in XAML, I feel comfortable to showing you a shortcut ... you can drag elements from the Toolbox into the visual XAML Editor:



When you do that, there are a few side effects. See what happened to me:



In my case, it created a left-margin of 368 and a top-margin of 101. That's not what I wanted, but I can easily edit that out in the XAML itself.

It's for this reason that I prefer to work directly with XAML ... especially if I remember the name of the control I want to use.

However, in this particular case, there's a huge upside to using the drag and drop technique. Look at the code that was added in line 8:

```
1 <phone:PhoneApplicationPage
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
5     xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8     xmlns:maps="clr-namespace:Microsoft.Phone.Maps.Controls;assembly=Microsoft.Phone.Maps"
9     x:Class="AroundMe.MainPage"
10    mc:Ignorable="d"
```

By dragging and dropping the Maps control, an XAML Namespace was added to the XAML document. This is necessary because the Maps control lives in a different Namespace and Assembly from the other controls I had been using.

Nonetheless, I now have a Map control. I edit it to simply sport a Name attribute. I want this because I know I'll be working with it in C# in just a moment:

```
55
56     <!--ContentPanel - place additional content here-->
57     <Grid x:Name="ContentPanel"
58         Grid.Row="1"
59         Margin="12,0,12,0" >
60
61         <maps:Map Name="AroundMeMap" />
62     </Grid>
63
64 </Grid>
```

If I were to try and run the application at this point (F5), I would experience an error at runtime:

```
16
17     // Constructor
18     public MainPage()
19     {
20         InitializeComponent();
21
22         // Sample code to localise
23         //BuildLocalizedApplication();
24     }
```

! XamlParseException occurred  
A first chance exception of type 'System.Windows.Markup.XamlParseException' occurred in System.Windows.ni.dll

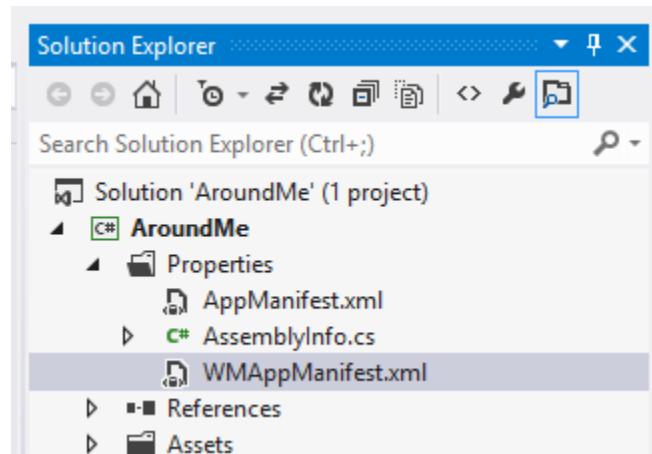
The problem is that the Map control requires the Mapping and Location capabilities of the phone and we've not told the Windows Phone operating system we want to use those capabilities in our app.

The mapping functionality of the Windows Phone 8 Operating System has changed dramatically from the previous version based on Bing maps. The new mapping features were built in conjunction with Nokia and are more integrated into the Phone's operating system than before in an effort to make the maps more performant. So this is why we need to request permission to use this capability of the phone ... it is now a core feature of the phone.

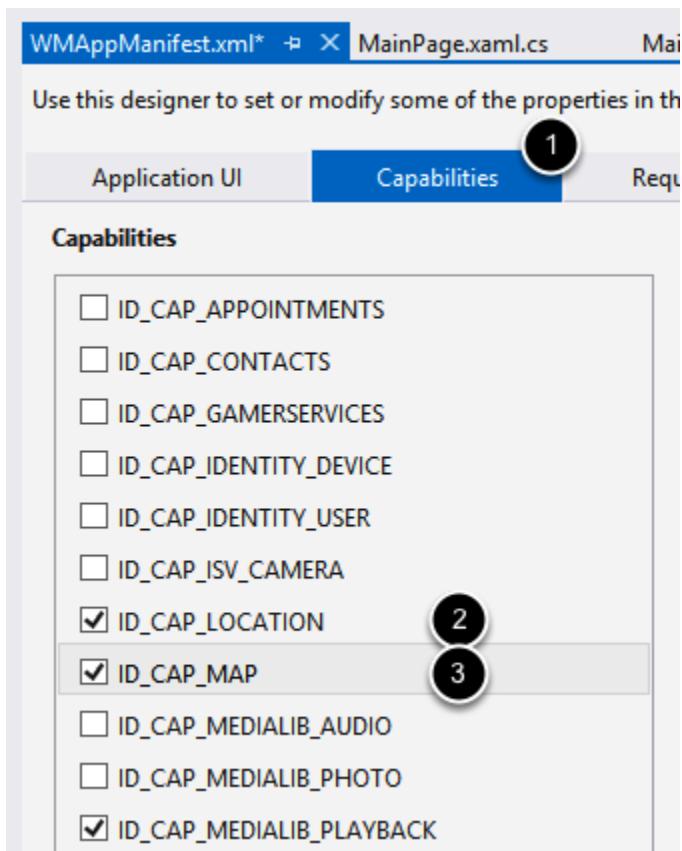
To learn more about the rationale for the changes to mapping in Windows Phone 8:

[http://blogs.windows.com/windows\\_phone/b/wpdev/archive/2013/01/24/the-windows-phone-map-control.aspx](http://blogs.windows.com/windows_phone/b/wpdev/archive/2013/01/24/the-windows-phone-map-control.aspx)

To remedy this, we'll make changes in the WPAppManifest.xml ... open the WMAppManifest.xml file:



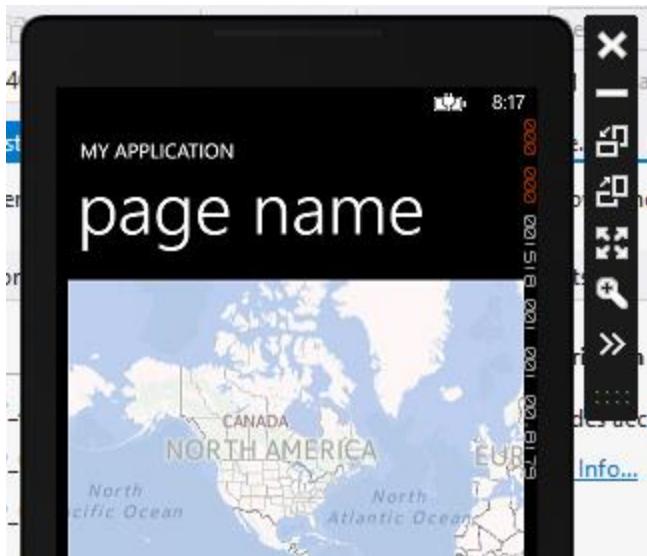
We'll need to add two Capabilities:



In the WPAppManifest.xml visual designer ...

1. Choose the Capabilities tab.
2. Place a check next to ID\_CAP\_LOCATION.
3. Place a check next to ID\_CAP\_MAP.

Save your changes and re-run the app (F5).



The Map control appears, but it's at a very high level because no location is set. We'll do that next.

In the MainPage.xaml.cs:

```
--  
12  Enamespace AroundMe  
13  {  
14    public partial class MainPage : PhoneApplicationPage  
15    {  
16      // Constructor  
17      public MainPage()  
18      {  
19        InitializeComponent();  
20  
21        1 Loaded += MainPage_Loaded;  
22  
23        // Sample code to localize the ApplicationBar  
24        //BuildLocalizedApplicationBar();  
25      }  
26  
27      void MainPage_Loaded(object sender, RoutedEventArgs e)  
28      {  
29        2 UpdateMap();  
30      }  
31  
32      private void UpdateMap()  
33      {  
34        3  
35      }  
36
```

1. We'll handle the Loaded event for the MainPage ... use the little technique I demonstrated earlier to create a method stub for MainPage\_Loaded (hint: hover over the M in "MainPage\_Loaded").
2. We'll create a helper method calls UpdateMap() that will contain the code to set the coordinates and other information to initialize the Map control.
3. Create the method stub for the UpdateMap() method.

Inside the UpdateMap() we'll add the following code:

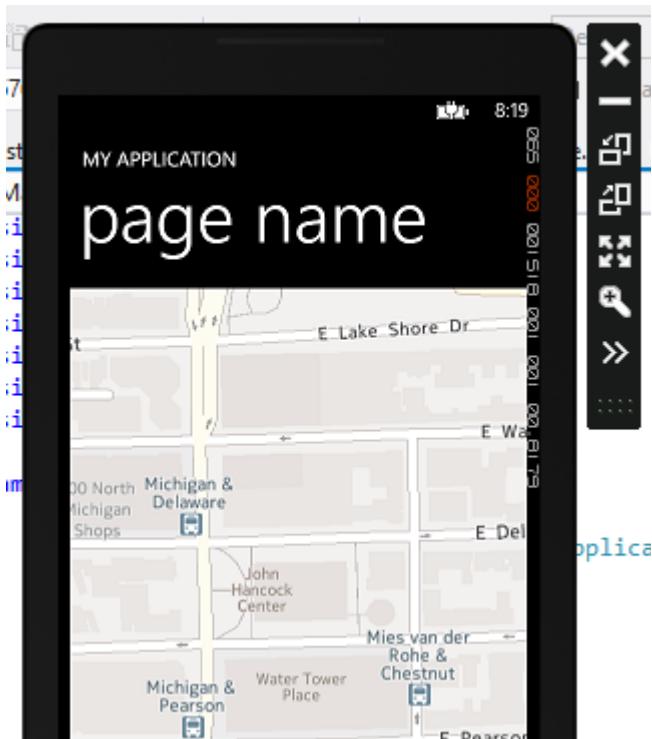
```
32  
33  private void UpdateMap()  
34  {  
35    AroundMeMap.SetView(new GeoCoordinate(41.8988D, -87.6231D), 17D);  
36  }  
37
```

The SetView() method combines several property settings into one convenient call. We'll pass in a GeoCoordinate hardcoded to a specific place (in Chicago ... more on

that in a moment) and a scaling factor of 17. You can experiment with these numbers. 19 is zoomed in really close and 5 is really far away.

By the way, I've suffixed each of the values with the letter "D" which is for Double ... this is just a way to ensure we're working with Double literals.

Now I'll test to make sure it works (F5).



And it does!

### Recap

To recap, the big take away from this lesson is how to use the new Map control in the Windows Phone 8. We'll become even more familiar with it as we go through these lessons, but at a minimum, we know that it requires us to set a Capability in the WMAppManifest.xml file. We also learned how to change the focus of the map and the zoom level. We learned about the GeoCoordinate class that represents a position on the globe based on its latitude and longitude. We still need to learn how to retrieve the current position of the Phone, and we'll do that soon, but this is a good start.

# Part 25: Working with the Geolocator and Geoposition Classes

Source Code: <http://aka.ms/absbeginnerdevwp8>

In the previous lesson I was able to hardcode a GeoCoordinate object and pass it to the Map control to set its position. However, for our app, we will want to retrieve that information from the phone. The Windows Phone API has a Geolocator class that will use the phone's GPS system to determine where in the world it is.

Here's the game plan:

1. We'll experiment with our UpdateMap() method to learn about the Geolocator class, how to set its accuracy and obtain the current position.
2. We'll learn about other classes in the Windows Phone API that allow us to work with the map and the locale of the user
3. We'll configure the Windows Phone Emulator and set its position to a specific place, namely, the John Hancock Center in Chicago. I have a special affinity for Chicago and for that building specifically that I'll tell you about later.

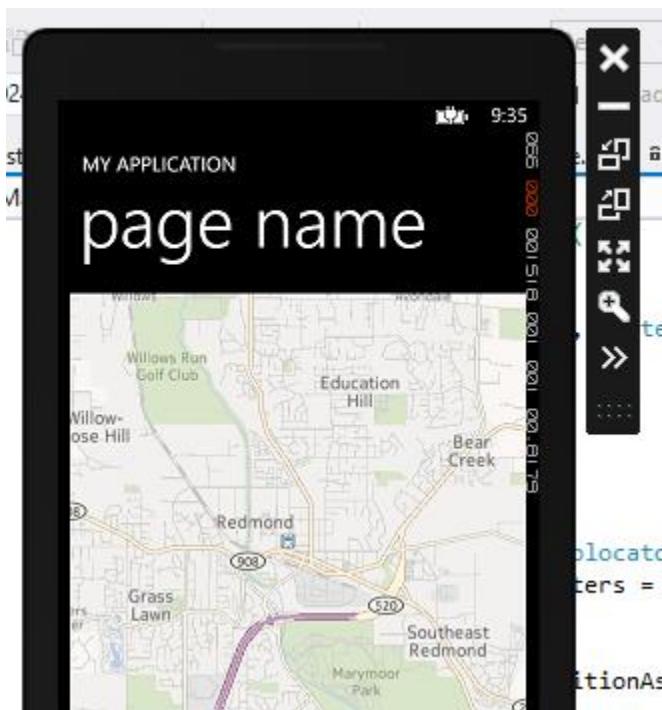
1. Modify UpdateMap() to retrieve its position from the Geolocator class

We'll pick up where we left off in the previous lesson, updating the UpdateMap() method as follows:

```
33
34 3 private async void UpdateMap()
35 {
36 1     Geolocator geolocator = new Geolocator();
37 2     geolocator.DesiredAccuracyInMeters = 50;
38
39 3     Geoposition position =
40 4         await geolocator.GetGeopositionAsync(
41 5             TimeSpan.FromMinutes(1),
42 6             TimeSpan.FromSeconds(30));
43
44 7     var gpsCoorCenter =
45 8         new GeoCoordinate(
46 9             position.Coordinate.Latitude,
47 10            position.Coordinate.Longitude);
48
49 11    AroundMeMap.SetView(gpsCoorCenter, 17);
50
51 12    //AroundMeMap.SetView(new GeoCoordinate(41.8988D, -87.6231D), 17D);
52
53 }
```

1. We create a new instance of the Geolocator class and set its DesiredAccuracyInMeters property to 50 (meters).
2. We call the GetGeopositionAsync method. At this point, I'd like to point out the best article I read on the process of acquiring the Geoposition of the phone and why there are several factors to consider. Actually, it's a blog post on the Windows Phone Developer blog from Daniel Estrada Alva, a software development engineer on the Windows Phone team. [http://blogs.windows.com/windows\\_phone/b/wpdev/archive/2012/11/30/acquiring-a-single-geoposition-in-windows-phone-8.aspx](http://blogs.windows.com/windows_phone/b/wpdev/archive/2012/11/30/acquiring-a-single-geoposition-in-windows-phone-8.aspx)
3. There's a keyword you may not be familiar with in line 40: the await keyword. This corresponds with the async keyword I added to the method's signature in line 34. Just how this works and why I would like to save until a later lesson when I take extensively about the new await functionality in C# 5.0. For now, just understand that the purpose of this is to keep our application responsive while potentially long running tasks are executing. In this case, the phone may take a long time to acquire the current location using its built in GPS hardware. While it is acquiring the location, the user's phone, and even our app, should continue to be responsive to the user's input. Again, more later.
4. The Geoposition object has a collection of coordinates. However, we have to do a little conversion because the Map control requires those coordinates be in a GeoCoordinate object.
5. Finally, we pass the GeoCoordinate we just created to the SetView() method.

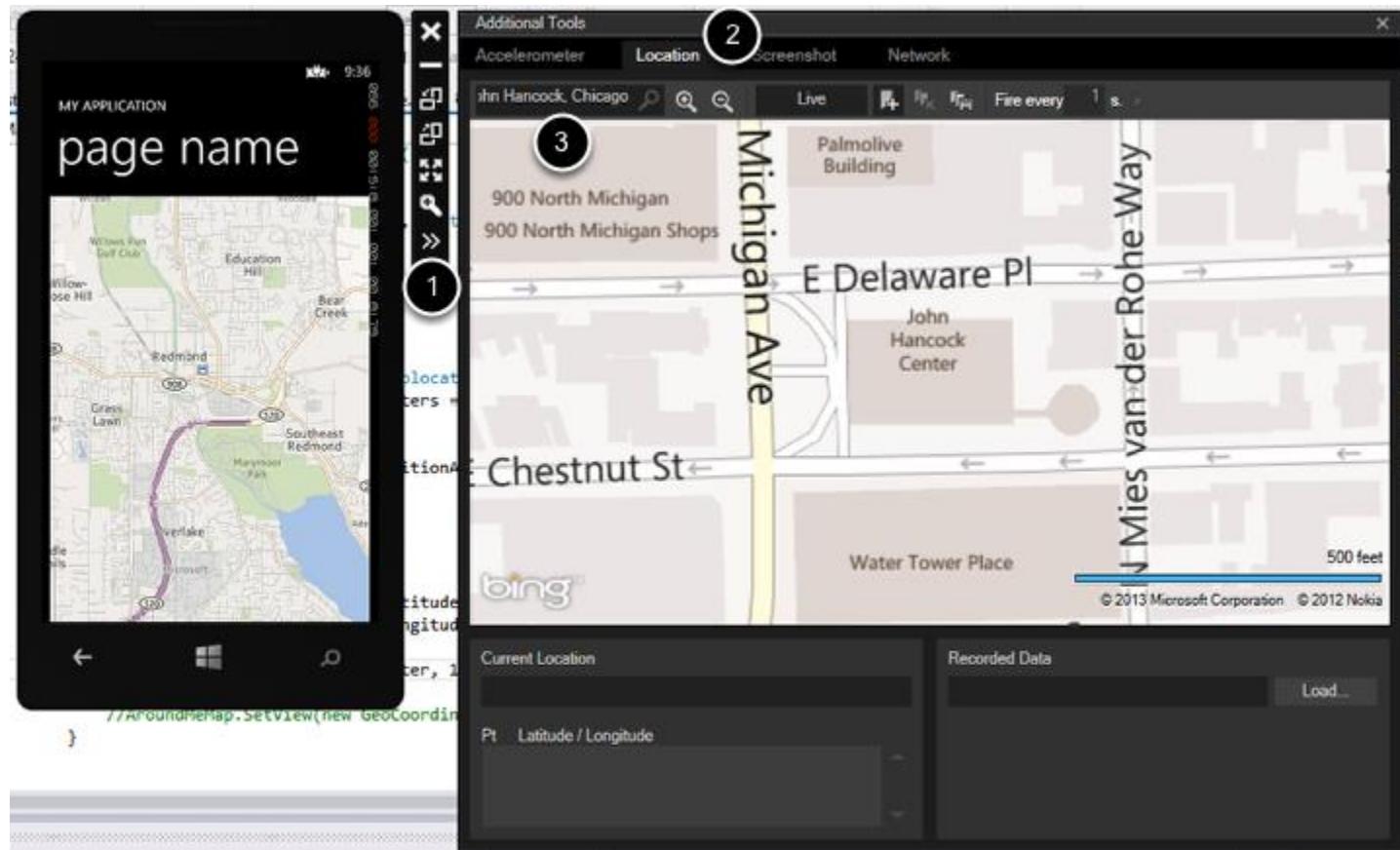
When we run the app (F5), we can see the result:



... the result is that the Map control is zoomed into Redmond, Washington, the home of Microsoft's main campus. This is the default position of the Windows Phone Emulator.

2. Use the Emulator's Additional Tools to change the virtual location of the Emulator for testing

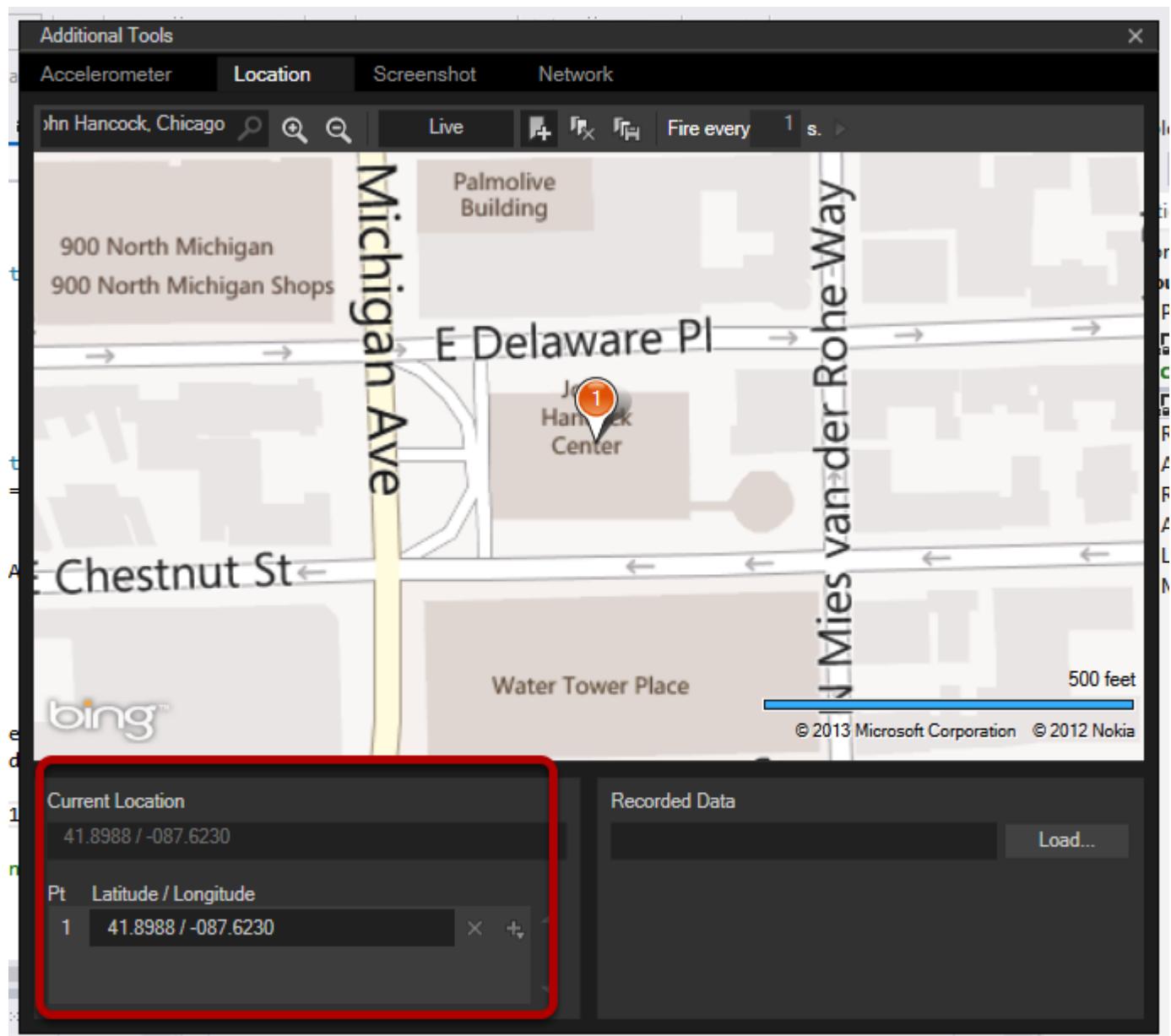
So, how do we change the Emulator to use a different latitude and longitude?



1. I click the double chevron to open up the Additional Tools panel.
2. I click the Location tab at the top.
3. I type the words "John Hancock, Chicago" into the Search textbox and hit the Enter key on my keyboard.

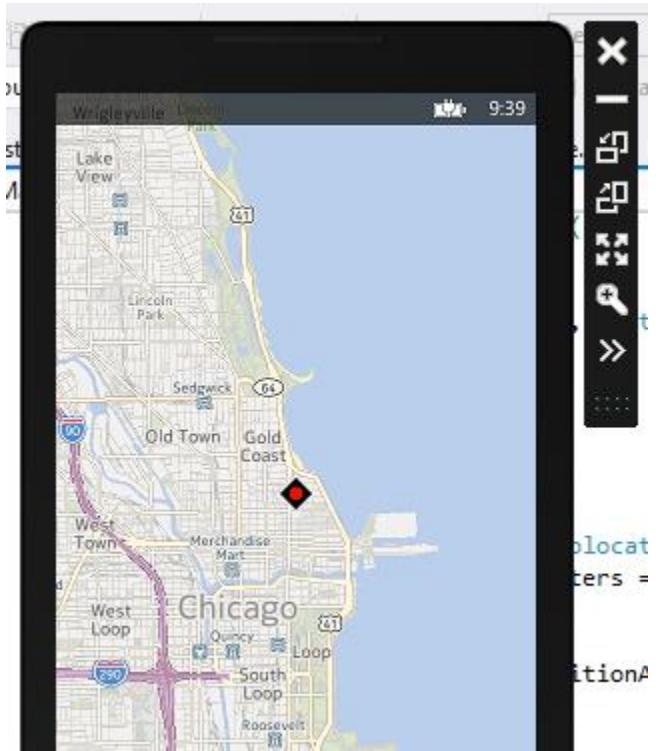
This shows me a different map ... a map of downtown Chicago positioned at the John Hancock Center. Excellent.

I'll use the primary (left) mouse button to add a pin on that location.

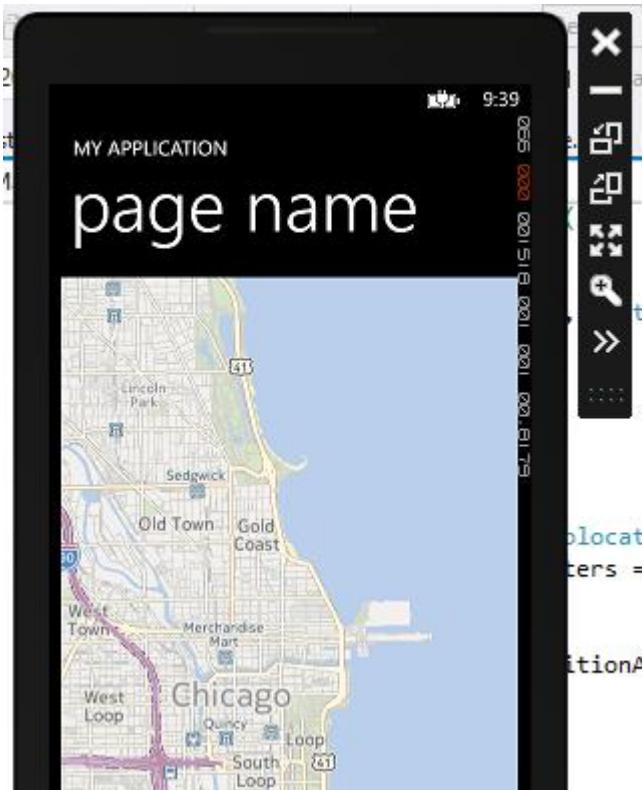


By adding a pin, I've set the Current Location to the correct latitude and longitude for the John Hancock Center.

However, sometimes it takes a while for the phone's GPS to catch up and refresh. What I've found works best is to exit out of the AroundMe app, go to the Windows Phone 8 application list, open Maps and wait for the phone to reposition to the desired location. When I do that, after about 10 seconds, the Maps program repositions to downtown Chicago.



Now, I re-run the app from the Application List (I don't need to stop and re-start the app from Visual Studio) and it should pick up the change in GeoPosition.



It works!

### Recap

Just to recap, the big take away from this lesson is how to use the Geolocator class to interface with the phone's GPS hardware to retrieve an instance of the GeoPosition class. The GeoPosition class has a number of interesting details, but for our purposes we care only about the longitude and latitude, which we can use to construct a new GeoCoordinate class in order to center the Map control. We learned how to configure the Phone Emulator's Location to make it think it is at a specific place in the world to test various location scenarios.

# Part 26: Retrieving a Photo from Flickr's API

Source Code: <http://aka.ms/absbeginnerdevwp8>

In this lesson we'll search for Flickr photos near the geocoordinate determined by our phone.

Many of the most popular phone apps have some interaction with web-based services ... so the app allows users to get at their own data that they've stored "in the cloud", or they provide access to data in order to make it available in some fresh new way.

As developers, we can access the web-based services programmatically IF they expose a web API. A web API is usually just an HTTP based method call ... the URL includes the input parameters of the method call, and the web API will return data back in some standard format. Nowadays, that is usually XML or JSON, JavaScript Object Notation.

Flickr is a great example of a web service we can leverage in our apps ... They have a vast amount of image data that we can search via their web-based API. All we need to do is call one of their methods and then parse the data they return to us.

Our game plan in this lesson:

1. We'll do a little investigation into the Flickr API to see how we can leverage it in our app
2. We'll get setup with Flickr so that we can make calls to their API, and get examples of how to call the method we want to utilize
3. We'll write code in our app to make calls to the Flickr API
4. When we get data back from our call to the Flickr API we'll figure out how to parse through it and actually obtain pictures that we can display in our app

This is a VERY long lesson, but it's crucial because it will serve as a proof of concept that our idea will actually work. We'll see how all the pieces come together and then the rest of the series will be adding improvements and refinements to what we do in this lesson.

1. Become familiar with the flickr API site, sign up for developer access  
Navigate to: <http://www.flickr.com/services/api/>

This is the home page for Flickr's API.

## The App Garden

[Create an App](#) | [API Documentation](#) | [Feeds](#) | [What is the App Garden?](#)

The Flickr API is available for non-commercial use by outside developers. Commercial use is possible by prior arrangement.

**Read these first:**

- [Developer Guide](#)
- [Overview](#)
- [Encoding](#)
- [User Authentication](#)
  
- [Dates](#)
- [Tags](#)
- [URLs](#)
- [Buddyicons](#)
  
- [Flickr APIs Terms of Use](#)

- [API Keys](#)
- [Developers' mailing list](#)

**API Methods****activity**

- [flickr.activity.userComments](#)
- [flickr.activity.userPhotos](#)

**auth**

- [flickr.auth.checkToken](#)
- [flickr.auth.getFrob](#)
- [flickr.auth.getFullToken](#)
- [flickr.auth.getToken](#)

**auth.oauth**

- [flickr.auth.oauth.checkToken](#)
- [flickr.auth.oauth.getAccessToken](#)

Before you can take advantage of the API in your app, you'll need to sign up for a developer account.

There should be an obvious Sign Up link somewhere on the top of the page. I won't walk through that process ... it's likely to change over time. I was able to use my existing Yahoo! ID to sign up for a developer account. It's free and easy to get started, and you'll need your own account so that you can get your own API Key.

Most companies exposing APIs over the web want to ensure that developers are complying with their terms of service, or they want to track usage by app to better understand how their APIs are being leveraged. I suppose in extreme cases the company may want to shut down an app that is abusing the service. This is why most require a unique identification for the developer and the app through the use of an API Key. Flickr is no different.

To get an API Key for the AboutMe app, find the "API Key" somewhere on the Flickr API homepage.

# The App Garden

[Create an App](#) | [API Documentation](#) | [Feeds](#) | [What is the App Garden?](#)

The Flickr API is available for non-commercial use by outside developers.  
Commercial use is possible by prior arrangement.

## Read these first:

- [Developer Guide](#)
- [Overview](#)
- [Encoding](#)
- [User Authentication](#)
  
- [Dates](#)
- [Tags](#)
- [URLs](#)
- [Buddyicons](#)
  
- [Flickr APIs Terms of Use](#)

- [API Keys](#)
- [Developers mailing list](#)

This will list all of the API Keys you were granted for your apps. You'll need to use a different API Key for each app you intend to use with Flickr's API.

Ultimately, you want to create a new API Key by using the "Get Another Key" button.

[The App Garden](#)



**bobtabor**

[Apps By You](#) | [Apps You're Using](#) | [Your Favorite Apps](#)

You haven't created any apps yet. Why not [create your first?](#)

Or have a read through the [App Garden FAQ](#).

Keep 'em coming!

Thanks for contributing to the [App Garden](#). If you have more apps in the works, we'd love to hear about them.



[Get Another Key](#) or [learn more](#).

For now, our app will not make any money, or it's not currently commercial but might be in the future, so choose "Apply for a Non-Commercial Key":

# The App Garden

[Create an App](#) | [API Documentation](#) | [Feeds](#) | [What is the App Garden?](#)

First, we need to know whether or not your app is commercial.

## Choose Non-Commercial if:

- Your app doesn't make money.
- Your app makes money, but you're a family-run, small, or independent business.
- You're developing a product which is not currently commercial, but might be in the future.
- You're building a personal website or blog where you are only using your own images.

[APPLY FOR A NON-COMMERCIAL KEY](#)

OR

## Choose Commercial if:

- You or your agency works for a major brand.
- AND one of the following:
- You want to make a profit.
  - You charge a fee for your product or services.
  - You will bring Flickr content into your product and intend to sell those services.

[APPLY FOR A COMMERCIAL KEY](#)



They want to learn more about the app you're building. You can copy my text if you want. You'll also need to agree to the terms and such:

## Tell us about your app:

Owner bobtabor

This app will be associated with your bobtabor account. You will not be able to change this after you submit your application.

What's the name of your app?

What are you building?  
(And trust us when we say you can't be detailed enough)

I'm building a simple app to learn more about Windows Phone 8 development using the Flickr API and the phone's GPS to find photos near where the user is currently standing.

I acknowledge that Flickr members own all rights to their content, and that it's my responsibility to make sure that my project does not contravene those rights.

I agree to comply with the [Flickr API Terms of Use](#).

or [Cancel](#)

When you fill out the form and click the SUBMIT button, you should receive your API Key.

Since I'm (probably) not supposed to legally reveal my key, I've blurred some of it out. Keep the Key and the Secret available. We'll need them later in this lesson.

Done! Here's the API key and secret for your new app:

The screenshot shows the Flickr API application details page. At the top, it says "AroundMe". Below that, "Key:" is followed by a long string of characters: "bdd5e...b7288". Below "Key:", "Secret:" is followed by another string: "72e8c...ec4". At the bottom of the screenshot, there are three blue links: "Edit app details", "Edit auth flow for this app", and "View all Apps by You".

### What to do next

If your key is to use in an application that someone else developed, for example to display your own or your group's photos on your website or blog, then you're all done!

If your key is for an app that you're developing, here are things to help you build and promote your app:

Back on the main Flickr API page, you see a list of all the web callable APIs available on Flickr. You can literally perform any conceivable operations using Flickr as the backend storage and processor for your photos and create new applications that "mash up" Flickr's functionality with some of your own. That's exactly what we want to do ... combine Flickr's search capability for photos -- specifically searching for photos that were taken geographically in the same place the user of our app.

We want to learn more about the flickr.photos.search API ... how do we call it? What options can we send along to specify geolocation we want to search for?

## 2. Learning about Flickr's search API

### photos

- [flickr.photos.addTags](#)
- [flickr.photos.delete](#)
- [flickr.photos.getAllContexts](#)
- [flickr.photos.getContactsPhotos](#)
- [flickr.photos.getContactsPublicPhotos](#)
- [flickr.photos.getContext](#)
- [flickr.photos.getCounts](#)
- [flickr.photos.getExif](#)
- [flickr.photos.getFavorites](#)
- [flickr.photos.getInfo](#)
- [flickr.photos.getNotInSet](#)
- [flickr.photos.getPerms](#)
- [flickr.photos.getRecent](#)
- [flickr.photos.getSizes](#)
- [flickr.photos.getUntagged](#)
- [flickr.photos.getWithGeoData](#)
- [flickr.photos.getWithoutGeoData](#)
- [flickr.photos.recentlyUpdated](#)
- [flickr.photos.removeTag](#)
- [\*\*flickr.photos.search\*\*](#) 
- [flickr.photos.setContentType](#)
- [flickr.photos.setDates](#)
- [flickr.photos.setMeta](#)
- [flickr.photos.setPerms](#)
- [flickr.photos.setSafetyLevel](#)
- [flickr.photos.setTags](#)

When you find the API you're looking for (i.e., `flickr.photos.search`), click the hyperlink to learn more about that web API method.

## flickr.photos.search

Return a list of photos matching some criteria. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.

### Authentication

This method does not require authentication.

### Arguments

#### api\_key (Required)

Your API application key. [See here](#) for more details.

#### user\_id (Optional)

The NSID of the user who's photo to search. If this parameter isn't passed then everybody's public photos will be searched. A value of "me" will search against the calling user's photos for authenticated calls.

#### tags (Optional)

A comma-delimited list of tags. Photos with one or more of the tags listed will be returned. You can exclude results that match a term by prepending it with a - character.

#### tag\_mode (Optional)

Either 'any' for an OR combination of tags, or 'all' for an AND combination. Defaults to 'any' if not specified.

#### text (Optional)

A free text search. Photos who's title, description or tags contain the text will be returned. You can exclude results that match a term by prepending it with a - character.

#### min\_upload\_date (Optional)

On this page dedicated to the flickr.photos.search web method, we can see which input parameters are optional and which are required, the purpose / meaning of each parameter, the expected format of the parameter value, and so on.

We also can see a list of error codes returned by the web method to the caller ... this might help us interpret any error codes we receive.

#### 115: Invalid XML-RPC Method Call

The XML-RPC request document could not be parsed.

#### 116: Bad URL found

One or more arguments contained a URL that has been used for abuse on Flickr.

### API Explorer

[API Explorer : flickr.photos.search](#)

There's also an "API Explorer : flickr.photos.search" link which takes us to a web page where we can experiment and learn how to call that particular web method.

## flickr.photos.search

### Arguments

Name	Required	Send	Value
user_id	optional	<input type="checkbox"/>	<input type="text"/>
tags	optional	<input type="checkbox"/>	<input type="text"/>
tag_mode	optional	<input type="checkbox"/>	<input type="text"/>
text	optional	<input type="checkbox"/>	<input type="text"/>
min_upload_date	optional	<input type="checkbox"/>	<input type="text"/>
max_upload_date	optional	<input type="checkbox"/>	<input type="text"/>
min_taken_date	optional	<input type="checkbox"/>	<input type="text"/>

### Useful Values

#### Recent public photo IDs:

8662595331 - IMG\_0506

8662595431 - \_DSC1174

8663693596 - Shady waterfall.

#### Popular public group IDs:

16978849@N00 - Black and White

1577604@N20 - Group with Experience

34427469792@N01 - FlickrCentral

By clicking the "Send" check box and supplying a value, we can see the format that our web method call should take. I'll input the a number of options such as the optional latitude and longitude for the John Hancock Center, and add optional text "observatory":

## flickr.photos.search

### Arguments

Name	Required	Send	Value
user_id	optional	<input type="checkbox"/>	<input type="text"/>
tags	optional	<input type="checkbox"/>	<input type="text"/>
tag_mode	optional	<input type="checkbox"/>	<input type="text"/>
text	optional	<input checked="" type="checkbox"/>	<input type="text" value="observatory"/>
min_upload_date	optional	<input type="checkbox"/>	<input type="text"/>
max_upload_date	optional	<input type="checkbox"/>	<input type="text"/>

I'll add an optional radius of "1". I learn that the default "radius\_units" are in kilometers, so by adding a "1" I'm saying "search within a 1 kilometer radius of the geocoordinate I'm specifying".

I also will choose to send the output of my query to JSON, the JavaScript Object Notation.

I'll choose the radio button next to "Sign call with no user token".

Once I've added my settings, I'll click the "Call Method..." button.

lon	optional	<input checked="" type="checkbox"/>	-87.6231
radius	optional	<input checked="" type="checkbox"/>	1
radius_units	optional	<input type="checkbox"/>	
is_commons	optional	<input type="checkbox"/>	
in_gallery	optional	<input type="checkbox"/>	
is_getty	optional	<input type="checkbox"/>	
extras	optional	<input type="checkbox"/>	
per_page	optional	<input type="checkbox"/>	
page	optional	<input type="checkbox"/>	

Output: JSON ▼

- Sign call with no user token?  
 Do not sign call?

Call Method...

[Back to the flickr.photos.search documentation](#)

When I click the "Call Method..." button, a large textbox will appear beneath the button containing sample data in JSON format, as well as the URL that was constructed based on my selected options. Both of these will be important for me in just a moment.

[Call Method...](#)

[Back to the flickr.photos.search documentation](#)

```
{
  "photos": {
    "page": 1,
    "pages": 13,
    "perpage": 100,
    "total": "1223",
    "photo": [
      {
        "id": "8128129712",
        "owner": "20282731@N03",
        "secret": "c5eefccedd",
        "server": "8476",
        "f...
      },
      {
        "id": "8631093478",
        "owner": "44978333@N06",
        "secret": "b75b5074b0",
        "server": "8383",
        "f...
      },
      {
        "id": "8505858710",
        "owner": "66625527@N07",
        "secret": "0fa17b0406",
        "server": "8379",
        "f...
      },
      {
        "id": "5178468286",
        "owner": "46272979@N08",
        "secret": "c60bccaa9f3",
        "server": "1381",
        "f...
      },
      {
        "id": "470862029",
        "owner": "39303693@N00",
        "secret": "d1b743ae4a",
        "server": "204",
        "f...
      },
      {
        "id": "7581122570",
        "owner": "17404515@N00",
        "secret": "00779f24e8",
        "server": "7127",
        "f...
      },
      {
        "id": "5643336607",
        "owner": "29596200@N00",
        "secret": "5e1ce5119d",
        "server": "5027",
        "f...
      },
      {
        "id": "5607856527",
        "owner": "97817658@N00",
        "secret": "d2f96cef06",
        "server": "5142",
        "f...
      },
      {
        "id": "8590930366",
        "owner": "26417560@N07",
        "secret": "f1716784a6",
        "server": "8380",
        "f...
      },
      {
        "id": "5608440118",
        "owner": "97817658@N00",
        "secret": "4387a93531",
        "server": "5065",
        "f...
      },
      {
        "id": "168445998",
        "owner": "28105008@N00",
        "secret": "e8995ba0f1",
        "server": 72,
        "farm": ...
      },
      {
        "id": "168446289",
        "owner": "28105008@N00",
        "secret": "f9acdd852e",
        "server": 45,
        "farm": ...
      },
      {
        "id": "168446684",
        "owner": "28105008@N00",
        "secret": "e615e154aa",
        "server": 56,
        "farm": ...
      },
      {
        "id": "8136473381",
        "owner": "20282731@N03",
        "secret": "705ba5b6c1",
        "server": "8325",
        "f...
      },
      {
        "id": "8139890029",
        "owner": "20282731@N03",
        "secret": "40d6602762",
        "server": "8050",
        "f...
      },
      {
        "id": "8324273864",
        "owner": "20282731@N03",
        "secret": "23d5959095",
        "server": "8493",
        "f...
      },
      {
        "id": "3231626612",
        "owner": "10971105@N03",
        "secret": "15afb746ea",
        "server": "3464",
        "f...
      },
      {
        "id": "4114053652",
        "owner": "52574631@N00",
        "secret": "c3401c7abd",
        "server": "2639",
        "f...
      },
      {
        "id": "4114053372",
        "owner": "52574631@N00",
        "secret": "168dce3091",
        "server": "2707",
        "f...
      },
      {
        "id": "4114052826",
        "owner": "52574631@N00",
        "secret": "2351f44129",
        "server": "2622",
        "f...
      },
      {
        "id": "4114053232",
        "owner": "52574631@N00",
        "secret": "b5227eb266",
        "server": "2733",
        "f...
      },
      {
        "id": "...",
        "owner": "...",
        "secret": "...",
        "server": "...",
        "farm": ...
      }
    ]
  }
}
```

URL: <http://api.flickr.com/services/rest/>

method=flickr.photos.search&api\_key=57f3406acd95fd7a1919f7b4e38264fb&text=observatory&lat=41.8988&lon=-87.6231&radius=1&format=json&nojsoncallback=1

### 3. Setting up our project to use the Flickr API

Back in my Visual Studio project, I'll need to prepare for making my first call into the Flickr API. I'll want to appropriately brand the app by changing the app and page titles. I'll just hardcode them this time and not use the resources file. I realize this will limit my app to just English speakers. I can always come back later and change this.

```

50   </> TitlePanel contains the name of the application and page title
51   <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
52     <TextBlock
53       Text="AROUND ME"
54       Style="{StaticResource PhoneTextNormalStyle}"
55       Margin="12,0"/>
56     <TextBlock
57       Text="pictures near ..."
58       Margin="9,-7,0,0"
59       Style="{StaticResource PhoneTextTitle1Style}"/>
60   </StackPanel>

```

1. Change the Text attribute to "AROUND ME"
2. Change the Text attribute to "pictures near ..."

My next changes will be in the ContentPanel.

```

61
62      <!--ContentPanel - place additional content here-->
63      <Grid x:Name="ContentPanel"
64          Grid.Row="1"
65          Margin="12,0,12,0" >
66          <Grid.RowDefinitions>
67              <RowDefinition Height="50" />
68              <RowDefinition Height="200" />
69              <RowDefinition Height="*" />
70          </Grid.RowDefinitions>
71
72          <TextBlock Name="ResultTextBlock" Grid.Row="0" />
73          <Image Name="FlickrImage" Grid.Row="1" Stretch="UniformToFill" />
74          <maps:Map Name="AroundMeMap" Grid.Row="2" />
75      </Grid>
76

```

1. Add three RowDefinitions at various heights
2. Put a control in each of those new rows. I create a TextBlock called "ResultTextBlock" for row 1. I create an Image control called "FlickrImage" for row 2.
3. Move the Map to row 3.
4. Programmatically calling the Flickr API via HTTP

My goal is to populate the Image control with a single image. Later, I'll add a search results page that can display all the results. For now, I just want to hack something together to figure out how to call into the Flickr search API programmatically and get results. I'll make it work correctly later.

```
37     private async void UpdateMap()
38     {
39         Geolocator geolocator = new Geolocator();
40         geolocator.DesiredAccuracyInMeters = 50;
41
42         Geoposition position =
43             await geolocator.GetGeopositionAsync(
44                 TimeSpan.FromMinutes(1),
45                 TimeSpan.FromSeconds(30));
46
47         var gpsCoorCenter =
48             new GeoCoordinate(
49                 position.Coordinate.Latitude,
50                 position.Coordinate.Longitude);
51
52         //AroundMeMap.SetView(new GeoCoordinate(41.8988D, -87.6231D), 17D);
53
54         AroundMeMap.Center = gpsCoorCenter;
55         AroundMeMap.ZoomLevel = 15;
56
57         ResultTextBlock.Text = string.Format("{0} - {1}",
58             AroundMeMap.Center.Latitude,
59             AroundMeMap.Center.Longitude);
60
61         HttpClient client = new HttpClient();
62     }
```

1. I comment out the SetView() and instead use the Map's Center and ZoomLevel properties just to show how to perform the same task using only the
2. The ResultTextBlock should not be here
3. I want to make a call via the web and I know there's a class in the Windows Phone 8 API called HttpClient. However, I don't have that package installed in my project.

I can find that package on NuGet at: <http://nuget.org/packages/Microsoft.Net.Http>

 **nuget** gallery

Log On Register

Home Packages Upload Package Statistics Documentation Blog Search Packages 

This is a prerelease version of HTTP Client Libraries.

## HTTP Client Libraries 2.1.3-beta

 367,821 Downloads

5,481 Downloads of v 2.1.3-beta

2/18/2013 Last update

[Project Site](#) [License](#) [Report Abuse](#) [Contact Owners](#) [How to Download](#)

This package provides a programming interface for modern HTTP applications on .NET Framework 4, Silverlight 4 and 5, Windows Phone 7.5 and 8. This package includes HttpClient for sending requests over HTTP, as well as HttpRequestMessage and HttpResponseMessage for processing HTTP messages.

This package also supports Portable Class Libraries.

This package is not supported in Visual Studio 2010, and is only required for projects targeting .NET Framework 4.5 or .NET for Windows Store apps when consuming a library that uses this package. For known issues, please see: <http://go.microsoft.com/fwlink/?LinkId=279987>.

To install HTTP Client Libraries, run the following command in the [Package Manager Console](#)

```
PM> Install-Package Microsoft.Net.Http -Pre
```

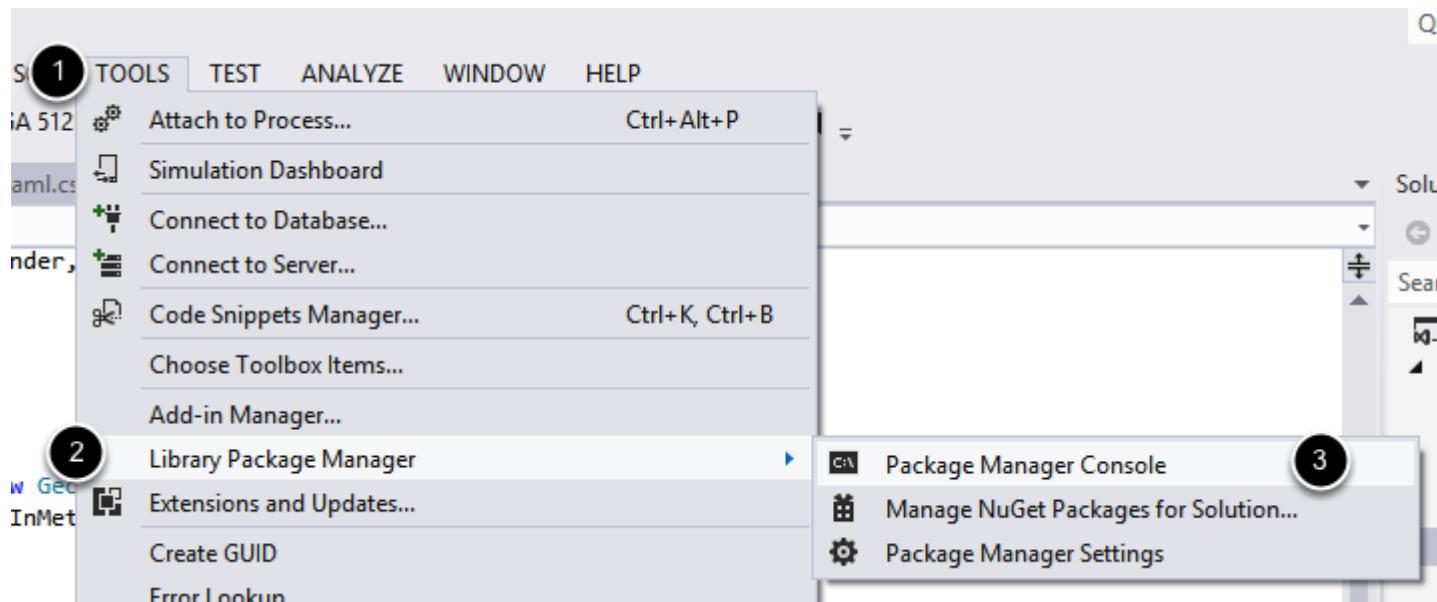
 [Tweet](#) 6

 [Like](#) 11 people like this. Sign Up to see what your friends like.

NOTE: Since I took this screenshot, this package no longer requires the -Pre argument.

I pay particular attention to the way you install the package using the Package Manager Console: `Install-Package Microsoft.Net.Http`

To open the Package Manager Console:



1. Tools menu
2. Library Package Manager
3. Package Manager Console

This will open the Package Manager Console by default in the bottom area of Visual Studio (unless you docked it somewhere else in a previous session).

You'll type in the `install-package` command at the Package Manager prompt: `Install-Package Microsoft.Net.Http`

The screenshot shows the 'Package Manager Console' window. At the top, there are dropdown menus for 'Package source' (set to 'NuGet official package source') and 'Default project' (set to 'AroundMe'). Below the header, a message states: 'Each package is licensed to you by its owner. Microsoft is not responsible for third party packages. Follow the package source (feed) URL to determine additional licenses.' The console displays the following text:  
Package Manager Console Host Version 2.2.40116.9051  
Type 'get-help NuGet' to see all available NuGet commands.  
PM> install-package Microsoft.Net.Http -pre

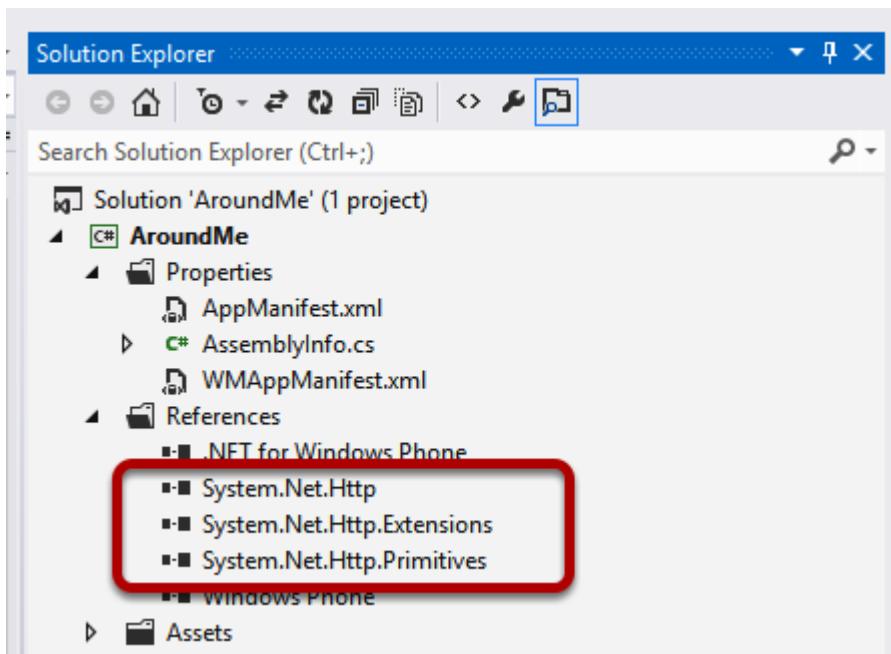
If all goes smoothly, you'll see a number of messages appear indicating success:

```
Package Manager Console
Package source: NuGet official package source | Default project: AroundMe
additional licenses. Follow the package source (feed) URL to determine a
Package Manager Console Host Version 2.2.40116.9051
Type 'get-help NuGet' to see all available NuGet commands.

PM> install-package Microsoft.Net.Http -pre
Attempting to resolve dependency 'Microsoft.Bcl (>= 1.0.16-rc)'.
Attempting to resolve dependency 'Microsoft.Bcl.Build (>= 1.0.4)'.
'Microsoft.Net.Http 2.1.3-beta' already installed.
Successfully added 'Microsoft.Bcl.Build 1.0.4' to AroundMe.
Successfully added 'Microsoft.Bcl 1.0.19' to AroundMe.
Successfully added 'Microsoft.Net.Http 2.1.3-beta' to AroundMe.

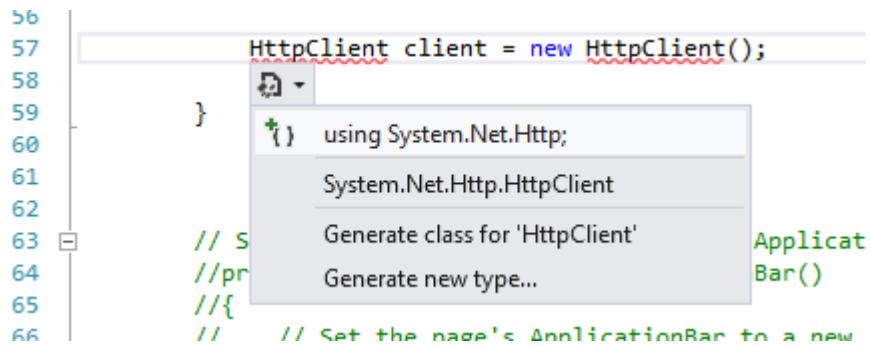
PM>
```

Now when you look at the Solution Explorer under the References folder, you'll see three new references System.Net.Http.\*



Now we should be able to resolve the reference for the HttpClient class. However your mouse cursor over the blue dash under the "H" in HttpClient to reveal a drop-down menu. Choose the: **using System.Net.Http;**

... option to add a using statement to your MainPage.xaml.cs.



A screenshot of a code editor showing a dropdown menu. The menu has a grey header with a magnifying glass icon and the text 'using System.Net.Http;'. Below this are three items: 'System.Net.Http.HttpClient' (highlighted in light blue), 'Generate class for 'HttpClient'' (disabled, shown in grey), and 'Generate new type...' (disabled, shown in grey). The background shows some C# code with line numbers 56 through 66. Line 57 contains the problematic 'HttpClient client = new HttpClient();' line. Lines 63 through 66 are comments starting with '// S', '//pr', and '//' respectively.

```
56
57     HttpClient client = new HttpClient();
58 }
59 }
60 }
61 }
62 }
63 // S
64 //pr
65 //{
66 //
```

Now I work my way through the process of building a URL to call Flickr's search web API. I'm using that URL from the "API Explorer" as my template, substituting the optional parts like the api\_key, license, lat, lon and so on.

```

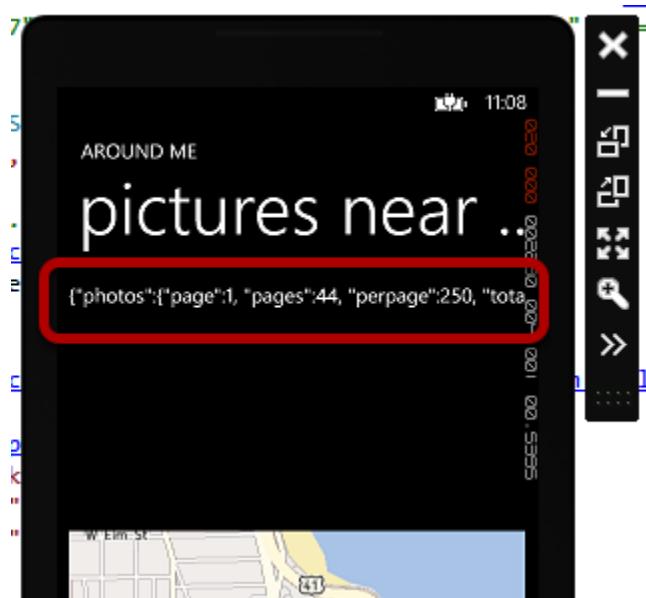
60
61     HttpClient client = new HttpClient();
62
63     // About licenses:
64     // http://www.flickr.com/services/api/flickr.photos.licenses.getInfo.html
65     /*
66      * <license id="4" name="Attribution License" url="http://creativecommons.org/licenses/by/2.0/">
67      * <license id="5" name="Attribution-ShareAlike License" url="http://creativecommons.org/licenses/by-sa/2.0/">
68      * <license id="6" name="Attribution-NoDerivs License" url="http://creativecommons.org/licenses/by-nd/2.0/">
69      * <license id="7" name="No known copyright restrictions" url="http://flickr.comcreativecommons.org/licenses/no-nd/2.0/">
70     */
71     string[] licenses = { "4", "5", "6", "7" };
72     string license = String.Join(", ", licenses);
73     license.Replace(",", "%2C");
74
75     // Your API key ... REPLACE THIS WITH YOURS:
76     // http://www.flickr.com/services/api/keys/
77     string flickrApiKey = "bdd5b7288";
78
79     // Search API
80     // http://www.flickr.com/services/api/flickr.photos.search.html
81
82     string url = "http://api.flickr.com/services/rest/" +
83         "?method=flickr.photos.search" +
84         "&license={0}" +
85         "&api_key={1}" +
86         "&lat={2}" +
87         "&lon={3}" +
88         "&radius=2" +
89         "&format=json" +
90         "&nojsoncallback=1";
91
92     var baseUrl = string.Format(url,
93         license,
94         flickrApiKey,
95         gpsCoorCenter.Latitude,
96         gpsCoorCenter.Longitude);
97
98     string flickrResult = await client.GetStringAsync(baseUrl);
99
100    ResultTextBlock.Text = flickrResult;
101
102}
103
104
```

1. First I add licenses that I want to search through. I only want to include licenses that have limited or no copyright licenses associated with them to avoid any legal problems. I could hardcode this string, however I would prefer to construct it so that I could modify it easier and more consistently in the future. As you can see I'm switching out the commas in this string for %2C ... this is a form of URL Encoding. Leaving certain punctuation in a

URL like commas, periods, quotation marks, question marks, ampersands and so on could produce unpredictable results because they have a specific meaning and usage when the URL is parsed by the web server. So, to bypass that, we URL encode those characters. After the web server software is finished parsing the URL, those characters will be decoded and understood the way we want them to be understood. It's a form of packing and unpacking a suitcase that you'll take on a flight with you.

2. Here I've separated out my Flickr API Key because I may want to replace it later.
3. Here I create the URL as a string with the replacement codes -- like {0} and {1} -- for use in the next line of code ...
4. Here we use string.Format to perform the replacement of the code -- {0} and {1}, etc. -- with the actual values we'll use for the web api call.
5. Here we actually make the call to Flickr by passing our newly constructed URL to the HttpClient's GetStringAsync() method. When it returns, we'll display it in the TextBlock I added earlier. I do this so I can actually see what we're getting back from the web service.

Let's run the app (F5) and see what we have so far:



What we get back from Flickr is a string of JSON as we expected.

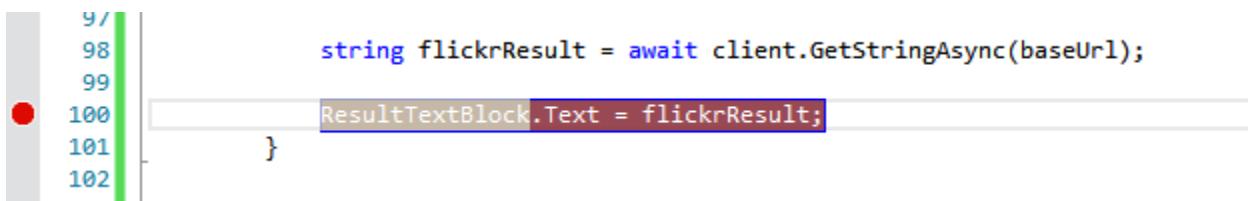
## 5. Deserializing the JSON result into classes

We will need to convert that string of JSON into something we can use, like CLR types (or rather, classes). There's a little trick for this that I learned from Clint that will create classes that represent the data structures used in the JSON so that we can deserialize

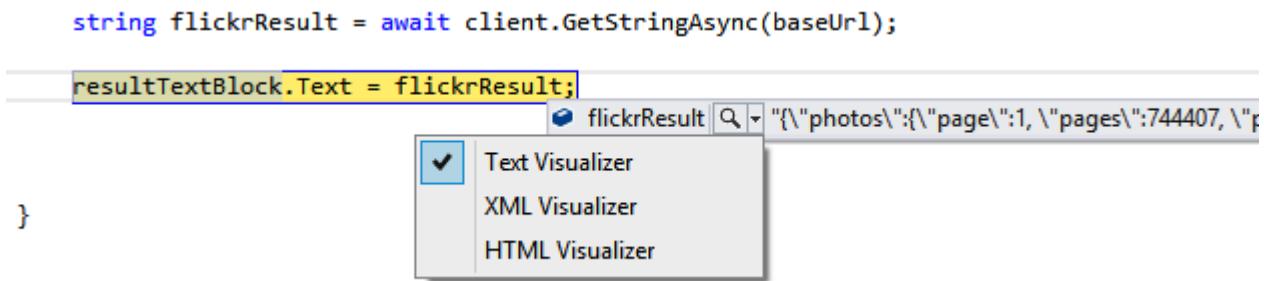
it into instances of those classes. Once we have the JSON data into a class hierarchy, we can work with it in C# quite easily.

First, we'll need to grab the actual JSON that has been returned by the web service call to Flickr.

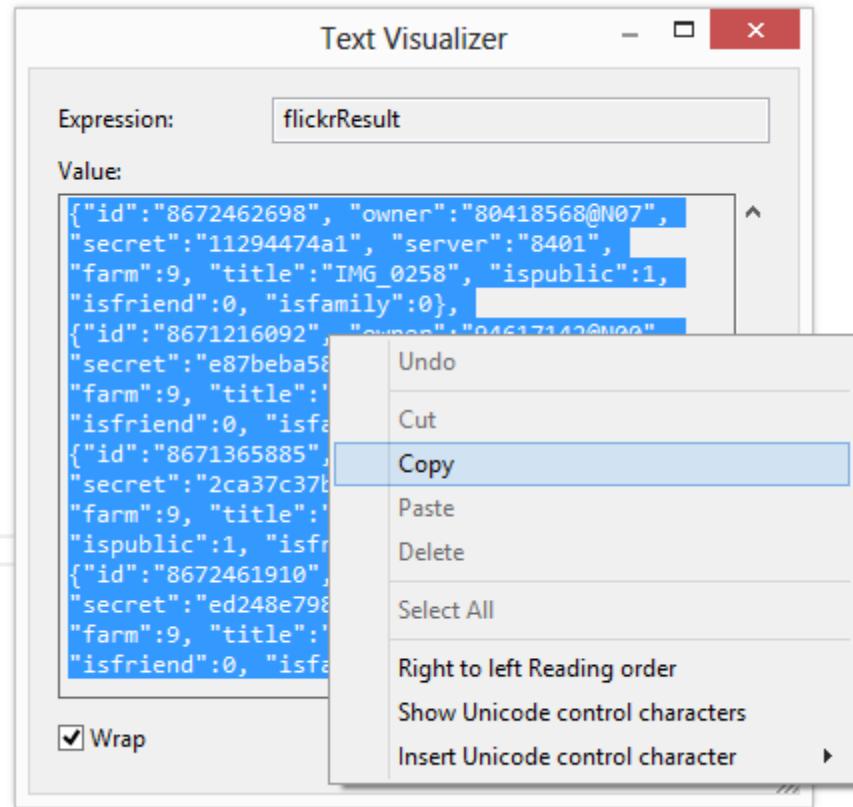
I set a breakpoint on the line of code where we set the ResultTextBlock's Text property to the result from Flickr:



And then I run / debug (F5) the app. When we get to that breakpoint, I hover over the `flickrResult` variable to see the data. If I click on the little magnifying glass icon next to the data I get a menu that will allow me to see the data in one of several visualization dialog windows:



Choosing "Text Visualizer" will open the Text Visualizer dialog. Here, I can copy all of the JSON on to my clipboard:



... and now that I have that JSON copied to my clipboard, I navigate to:

<http://json2csharp.com>

... and paste the JSON into the large text box on that page, and click the Generate button:

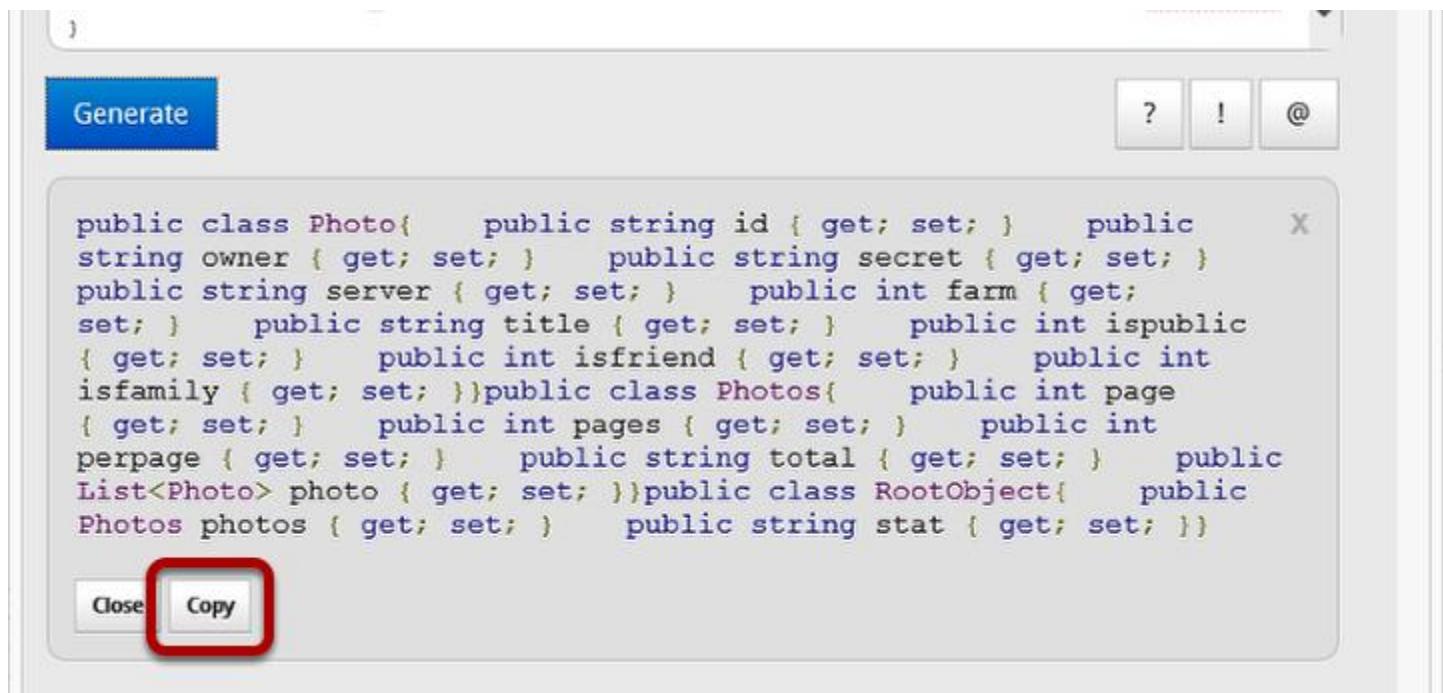
The screenshot shows a web browser window with the URL <http://json2csharp.com/>. The page title is "json2csharp - generate c# classes from json". On the left, there's a large text area containing JSON data:

```
farm":9, "title":"IMG_0258", "ispublic":1, "isfriend":0, "isfamily":0}, {"id":"8671216092", "owner":"94617142@N00", "secret":"e87beba58b", "server":"8543", "farm":9, "title":"IMG_8811", "ispublic":1, "isfriend":0, "isfamily":0}, {"id":"8671365885", "owner":"30335727@N00", "secret":"2ca37c37b1", "server":"8259", "farm":9, "title":"Adopt A Highway (7 of 136)", "ispublic":1, "isfriend":0, "isfamily":0}, {"id":"8672461910", "owner":"94617142@N00", "secret":"ed248e7984", "server":"8524", "farm":9, "title":"IMG_3009", "ispublic":1, "isfriend":0, "isfamily":0}}], "stat":"ok"} )
```

Below the JSON input is a blue "Generate" button. To the right of the button are three small icons: a question mark, an exclamation mark, and an '@' symbol. At the bottom right, there's a logo for "AppHarbor" featuring a stylized dragon icon.

Beneath the Generate button, I get the output: C# classes that match the JSON data structure. Quite awesome!

I click the "Copy" button ...



The screenshot shows a software window with a blue header bar containing a 'Generate' button and three icons. Below the header is a code editor window displaying C# code. The code defines three classes: Photo, Photos, and RootObject. The Photo class has properties for id, owner, secret, server, title, farm, ispublic, isfriend, isfamily, and methods for page, pages, total, and photo. The Photos class has properties for page, pages, total, and photo. The RootObject class has properties for photos and stat. At the bottom of the code editor are two buttons: 'Close' and 'Copy', with 'Copy' being highlighted by a red rectangle.

```
public class Photo{    public string id { get; set; }    public string owner { get; set; }    public string secret { get; set; }    public string server { get; set; }    public int farm { get; set; }    public string title { get; set; }    public int ispublic { get; set; }    public int isfriend { get; set; }    public int isfamily { get; set; }}public class Photos{    public int page { get; set; }    public int pages { get; set; }    public int total { get; set; }    public List<Photo> photo { get; set; }}public class RootObject{    public Photos photos { get; set; }    public string stat { get; set; }}
```

... and click Ctrl + C to copy the C# code to my clipboard ...

X

```
public class Photo
{
    public string id { get; set; }
    public string owner { get; set; }
    public string secret { get; set; }
    public string server { get; set; }
    public int farm { get; set; }
    public string title { get; set; }
    public int ispublic { get; set; }
    public int isfriend { get; set; }
    public int isfamily { get; set; }
}

public class Photos
{
    public int page { get; set; }
    public int pages { get; set; }
    public int perpage { get; set; }
    public string total { get; set; }
    public List<Photo> photo { get; set; }
}

public class RootObject
```

[Close](#)[Back](#)

press CTRL+C now to copy this code to your clipboard

... and paste it (carefully) beneath the Class definition for the MainPage class, but inside the AroundMe namespace like so:

```

37     private async void UpdateMap()
38     {
39         Geolocator geolocator = new Geolocator();
40         geolocator.DesiredAccuracyInMeters = 50;
41
42         Geoposition position =
43             await geolocator.GetGeopositionAsync(
44                 TimeSpan.FromMinutes(1),
45                 TimeSpan.FromSeconds(30));
46
47         var gpsCoorCenter =
48             new GeoCoordinate(
49                 position.Coordinate.Latitude,
50                 position.Coordinate.Longitude);
51
52         //AroundMeMap.SetView(new GeoCoordinate(41.8988D, -87.6231D), 17D);
53
54         AroundMeMap.Center = gpsCoorCenter;
55         AroundMeMap.ZoomLevel = 15;
56
57         ResultTextBlock.Text = string.Format("{0} - {1}",
58             AroundMeMap.Center.Latitude,
59             AroundMeMap.Center.Longitude);
60
61         HttpClient client = new HttpClient();
62     }

```

On line 134, I will rename "RootObject" to:

FlickrData

... it's more descriptive of what its purpose is, and I believe in clear names for variables and classes. The result should look like this:

```

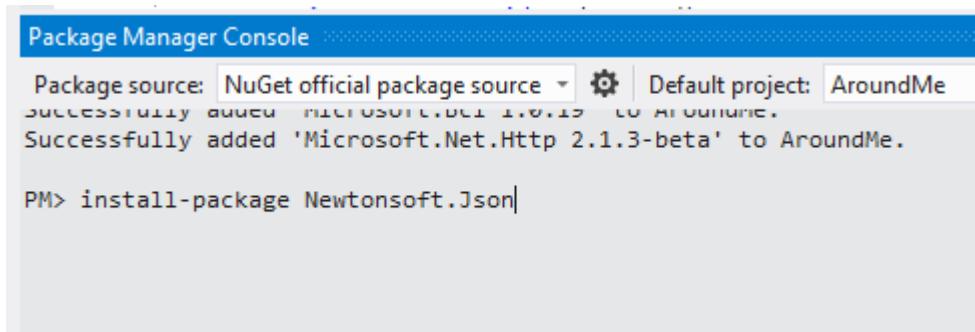
134     public class FlickrData
135     {
136         public Photos photos { get; set; }
137         public string stat { get; set; }
138     }

```

Next, I want to actually deserialize the JSON data into instances of these new classes. To do that, I'll use a third-party but heavily utilized package called Newtonsoft.Json.

I open the Package Manager Console (like I did earlier in this lesson) and type the following: **install-package Newtonsoft.Json**

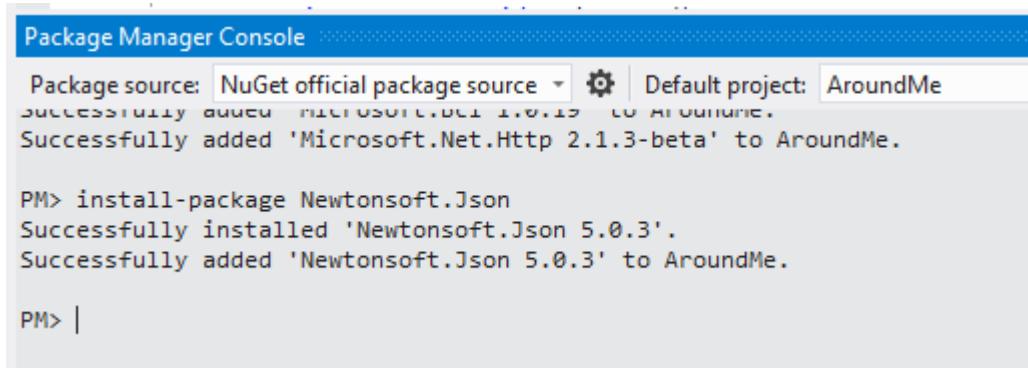
... and hit the enter key.



The screenshot shows the Windows Taskbar at the bottom with icons for File Explorer, Start, Task View, and others. The Package Manager Console window is open, displaying the following text:

```
Package Manager Console
Package source: NuGet official package source | Default project: AroundMe
Successfully added 'Microsoft.Net.Http 2.1.3-beta' to AroundMe.
Successfully added 'Newtonsoft.Json' to AroundMe.
```

If all goes well, you should get messages indicating success like you see below:



The screenshot shows the Windows Taskbar at the bottom with icons for File Explorer, Start, Task View, and others. The Package Manager Console window is open, displaying the following text:

```
Package Manager Console
Package source: NuGet official package source | Default project: AroundMe
Successfully added 'Microsoft.Net.Http 2.1.3-beta' to AroundMe.
Successfully added 'Newtonsoft.Json 5.0.3' to AroundMe.
```

Now, we need to revisit the `UpdateMap()` method and modify it to actually perform the deserialization:

```

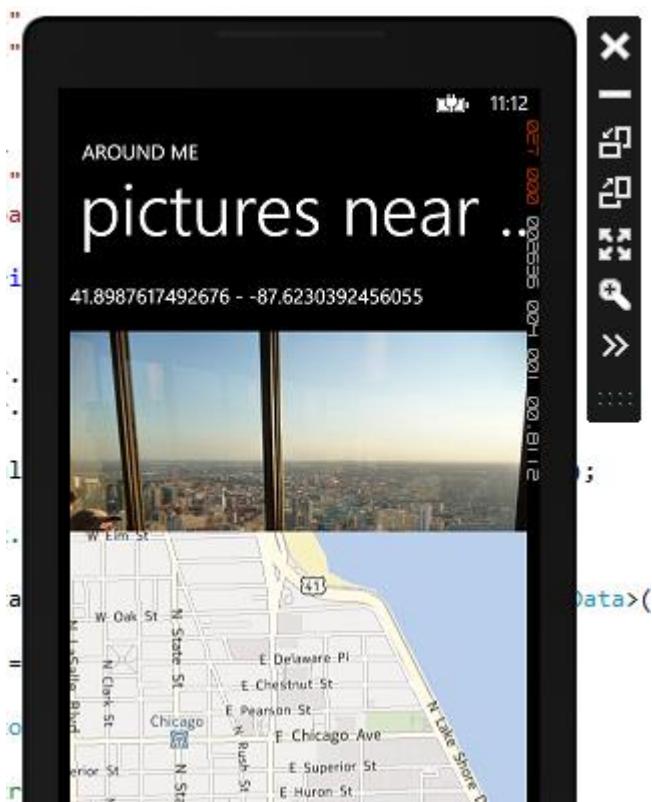
99
100    //ResultTextBlock.Text = flickrResult;
101
102    FlickrData apiData = JsonConvert.DeserializeObject<FlickrData>(flickrResult);
103
104    if (apiData.stat == "ok")
105    {
106        foreach (Photo data in apiData.photos.photo)
107        {
108            // To retrieve one photo, use this format:
109            //http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}{size}.jpg
110
111            string photoUrl = "http://farm{0}.staticflickr.com/{1}/{2}_{3}_n.jpg";
112
113            string baseFlickrUrl = string.Format(photoUrl,
114                data.farm,
115                data.server,
116                data.id,
117                data.secret);
118
119            // FlickrImage is just an Image control in XAML for now
120            FlickrImage.Source = new BitmapImage(new Uri(baseFlickrUrl));
121
122            // Just trying to grab 1 photo, so break out of this
123            break;
124        }
125    }
126
127

```

1. The deserialization is pretty easy ... just one line of code. We call the static `DeserializeObject<T>` method of the `JsonConvert` object from `Newtonsoft.Json` package (not shown here ... you'll need to add a using statement using the technique I demonstrated earlier).
2. The `FlickData` class has a "stat" property that represents whether the call was successful and contains data. If the "stat" property has the value "ok", we're good to iterate through the data.
3. Here we iterate through the `photos.photo` property which is a `List<Photo>`. For each photo ...
4. We want to construct a URL that will retrieve each individual photo (line 111). That URL is interesting and gives us an insight into how Flickr stores its data ... photos are saved in different server farms on different servers. Each photo has an ID, and just to protect the service from a hacker trying to hack URLs to find private or hidden data, a property called "secret" is added. Notice that I combine that with the letter "n" indicating the size of the photo I want to retrieve. I'll have more to say about that later.
5. I want to take this newly constructed URL and retrieve that photo from Flickr. To do this, I'll create a new `BitmapImage` object passing in the URI for where the photo is located. I'll give the new `BitmapImage` to the `FlickrImage` control's `Source` property.
6. Just for testing purposes, I only need one photo, so I'll immediately break out of the for each. I'll revisit this later to grab a bunch of photos and add them for display in a grid.

Again, that will come later. For now, I just want to make sure I can grab the data and display one photo in my app.

I run the app (F5) to test:



And it seems to work!

#### Recap

To quickly recap, there were quite a few important takeaways from this lesson ... we learned about the Flickr API which contains hundreds of method calls giving us access to practically every feature of Flickr so we can integrate it in our apps. We learned about getting an API Key and how to call the search API and integrate it into our app. We learned a little about JSON, how to use the json2.csharp.com website to create class definitions that match the JSON we want to work with. We learned about the Newtonsoft.JSON to deserialize the JSON into instances of those classes so we can work with them in C#, and a lot more.

# Part 27: Navigating and Passing Data to the SearchResults Page

Source Code: <http://aka.ms/absbeginnerdevwp8>

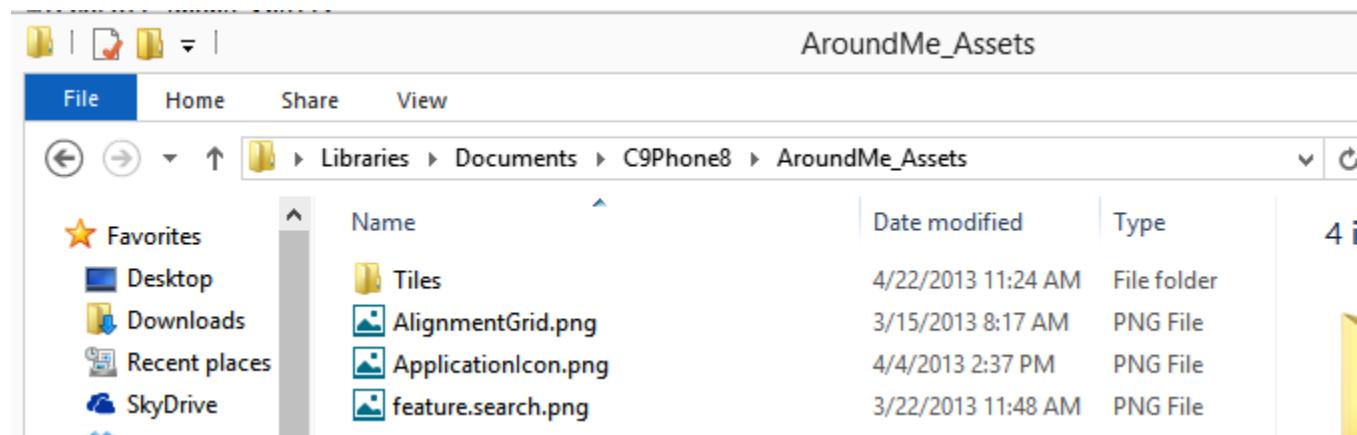
Now we have a map and a single image returned from Flickr, we have the basic building blocks for our app. But our eventual aim is to show a list of images instead of just one. So, in this lesson we'll do that by adding a SearchResults page and displaying all the images returned by our search.

Our game plan in this lesson:

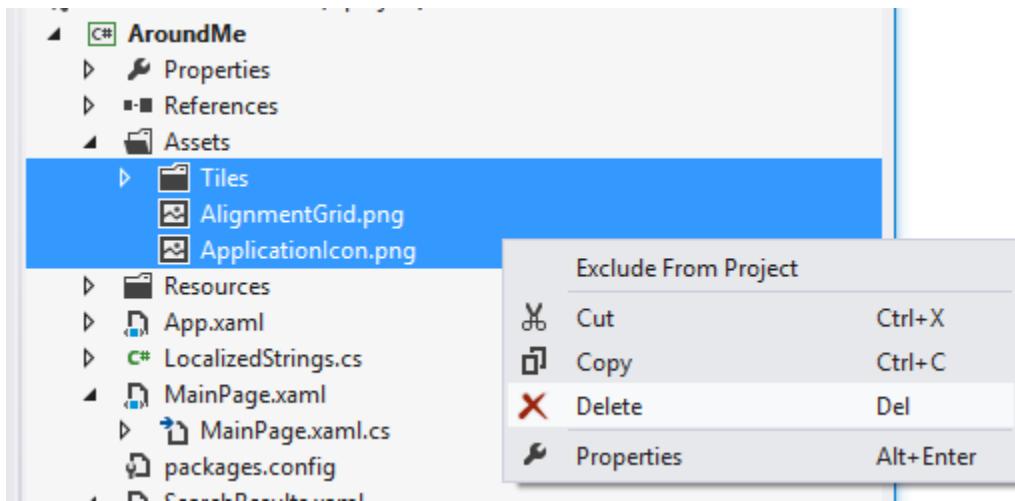
1. We'll add the AroundMe assets for the tile image and so on
2. We'll add an Application Bar and Search button that will navigate to a new page that will ultimately display Flickr images, sending along the latitude and longitude as parameters
3. We'll add a new page, the SearchResults.xaml page, we'll add a text block to the page, and we'll add code that retrieves the parameters sent from the MainPage to the new SearchResults page and display the latitude and longitude values, just to make sure we are correctly passing that data
4. We'll clean up and refactor the code we've written into class files, and clean up the code that calls our new class file methods so that it more closely resembles our original diagrams / low-tech mock ups of the user interface and user interaction

## 1. Add AroundMe assets to project

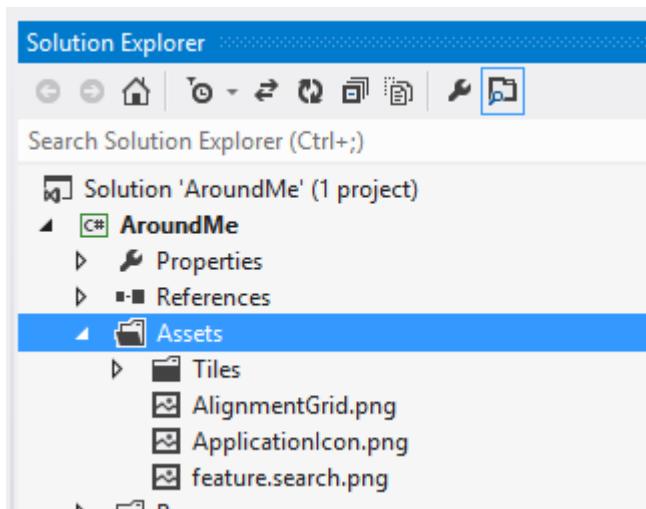
In the downloads available for this series, there's a subfolder called AroundMe\_Assets. We'll be copying (dragging and dropping) the files and subfolders in that directory to Visual Studio to make them part of our project's Assets.



First, delete everything inside of the Assets folder ...



... then in Windows Explorer, drag and drop the files in the AroundMe\_Assets folder to Visual Studio's Solution Explorer beneath the Assets folder. When you're finished it should look like this:

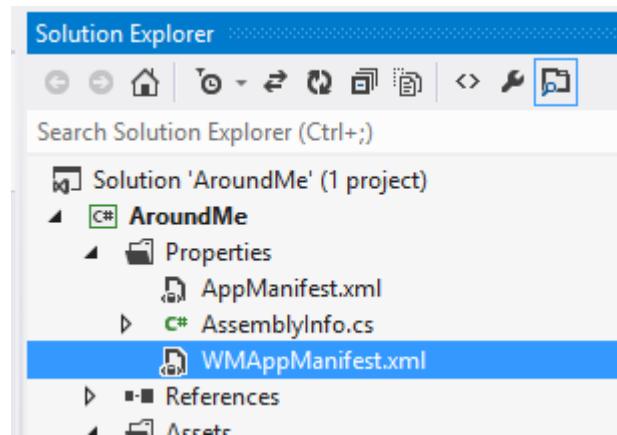


The number of files may look the same, but their contents are different ... i.e., the ApplicationIcon.png is for the AroundMe application. There's also an icon called feature.search.png ... it will be used in this lesson as the icon for the Application Bar's Search button.

2. Make sure assets appear in the WMAppManifest.xml correctly

The assets (images) we copied into our project's Assets folder should match the names of the default assets added by the project template. Just to make sure, let's take a look at the WMAppManifest.xml and see if the new image assets appear as expected.

Find the WMAppManifest.xml file in the Properties folder of the project. Double-click to open:



In the WMAppManifest.xml designer, the App Icon should have the appearance of the skyline of my home town (and Clint's home town), Chicago.

WMAAppManifest.xml X SearchResults.xaml.cs SearchResults.xaml

Use this designer to set or modify some of the properties in the Windows Phone app

Application UI Capabilities Requirements Properties

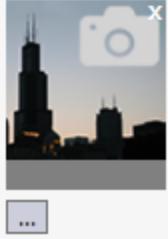
Use this page to set the UI details that identify and describe your application.

Display Name: AroundMe

Description: Sample description

Navigation Page: MainPage.xaml

App Icon:



...

Furthermore, we'll want to make sure that Tile Template is set to TemplateCycle, and that the Tile Images (Small, Cycle 1) look like the App Icon:

Supported Resolutions:



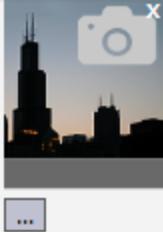
wvga     wxga     720p

Tile Template: TemplateCycle

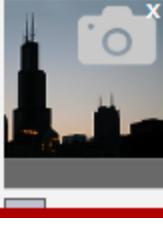
Support for large Tiles

Tile Title:

Tile Images:

Small: 

... 

Cycle 1: 

Cycle 2: 

Cycle 3: 

### 3. Add an Application Bar and Search Button

We've added an Application Bar in a previous lesson, so much of this should look familiar:

```
118
119     // Sample code for building a localized ApplicationBar
120     private void BuildLocalizedApplicationBar()
121     {
122         // Set the page's ApplicationBar to a new instance of ApplicationBar.
123         ApplicationBar = new ApplicationBar(); 1
124
125         AppBarIconButton appBarButton =
126             new AppBarIconButton(
127                 new Uri("/Assets/feature.search.png", UriKind.Relative)); 2
128         appBarButton.Text = "Search";
129         appBarButton.Click += SearchClick; 3
130         ApplicationBar.Buttons.Add(appBarButton); 4
131     }
132 }
```

1. I'll create a new instance of the ApplicationBar class
2. I'll create a new instance of the AppBarIconButton class ... in the constructor, I'll reference the location of the Search icon we added a moment ago.
3. I'll add the text that will be displayed under the button, and attach the SearchClick event handler method to the Click event of the button. Obviously, that method doesn't exist just yet. We'll come back to this in a moment and implement it.
4. I'll add the new Search button to the Application Bar.

Now, let's go back and implement the SearchClick method (from #3, above). There's a little blue bar under the 'S' in 'SearchClick'. Hover your mouse cursor until a tiny menu appears. It will drop down to another menu option to generate a method stub for the SearchClick method:

```
127
128     new Uri("/Assets/feature.search.png", UriKind.Relative));
129     appBarButton.Text = "Search";
130     appBarButton.Click += SearchClick;
131     ApplicationBar.Buttons.Add(appBarButton);
132 }
```

 Generate method stub for 'SearchClick' in 'AroundMe.MainPage'

If you click that menu option, it will generate a method stub as promised. We'll implement that method in the next step...

4. Navigate to a new page, pass data to the page

I add the following code to the event handler method:

```
133     private void SearchClick(object sender, EventArgs e)
134     {
135         1     string navTo = string.Format(
136             "/SearchResults.xaml?latitude={0}&longitude={1}",
137             AroundMeMap.Center.Latitude,
138             AroundMeMap.Center.Longitude);
139
140         2     NavigationService.Navigate(new Uri(navTo, UriKind.RelativeOrAbsolute));
141     }
142 }
```

1. In this line, we create a string that represents the path to a page called SearchResults.xaml. We've not yet added that page to our application -- we'll do that in a moment. If you look at the string we're building, we're passing additional information in the form of a query string.

We've already talked about query strings and passing data in a previous lesson. The only wrinkle is that this time we're sending two bits of data from the MainPage.xaml to a new page we've yet to create called SearchResults.xaml. You use the ampersand character & to separate name / value pairs in a query string. Additionally, we're using String.Format() to substitute the actual latitude and longitude values into the Uri string.

2. In line 140, we're using a class called the NavigationManager to Navigate() from the MainPage.xaml to a new page as defined in the new Uri object. The NavigationService not only knows how to navigate from one page to another, but is also responsible for keeping a history of previous navigations so that, when you use the back button on your Windows Phone 8, it can take you back to both the page AND the state of that page as you left it previously.

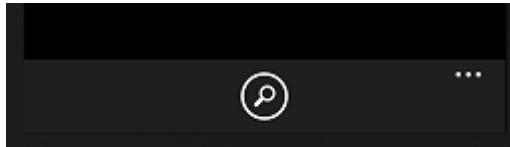
Now that we've created the Application Bar's Search button and have implemented the Click event handler for that button, we'll want to display the Application Bar.

## 5. Show the Application Bar

We'll merely add a call to the BuildLocalizedApplicationBar() method we created a moment ago.

```
20     public partial class MainPage : PhoneApplicationPage
21     {
22         // Constructor
23         public MainPage()
24         {
25             InitializeComponent();
26
27             Loaded += MainPage_Loaded;
28
29             BuildLocalizedApplicationBar(); ←
30         }
31     }
```

Let's test the app:

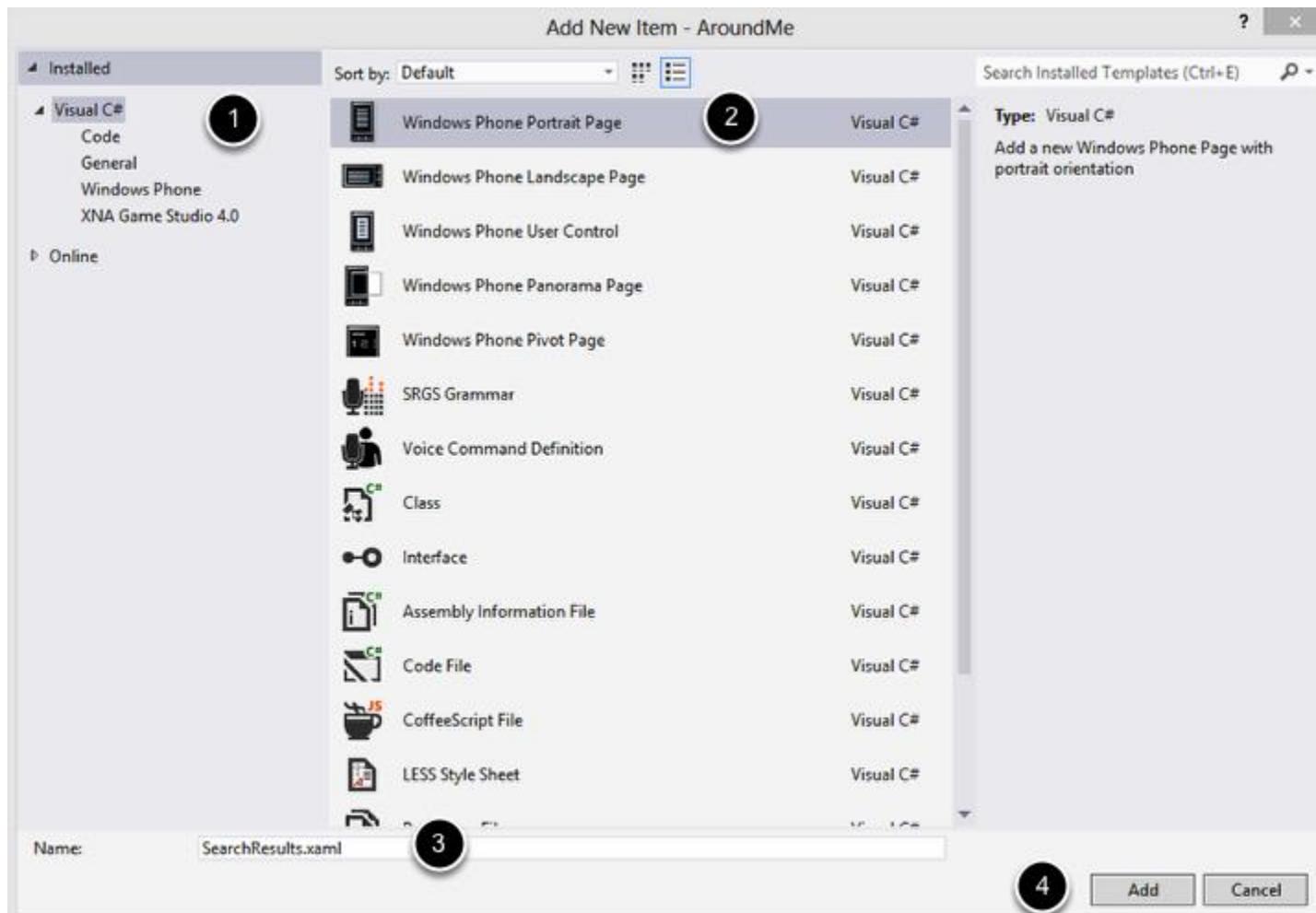


... and now at the bottom of our app, we can see the Search icon. Clicking the ellipsis in the upper-right hand corner of the App Bar, we can see it expand to show the Text property of the button as well.

If we were to click it, the `NavigationService.Navigate()` method would probably throw an exception because we do not have a page called `SearchResults.xaml` in our project. Let's add it.

6. Add a new item to the project, call it `SearchResults.xaml`

Right-click project name in Solution Explorer, select Add | New Item ... In the Add New Item dialog:



1. Make sure you're in the Visual C# templates
2. Select Windows Phone Portrait Page
3. Make sure you call this: SearchResults.xaml
4. Click Add

Next, we'll want to quickly test to make sure the code we wrote correctly navigates to our new page AND sends along those query string values.

7. Validate that the values are being passed correctly between the MainPage and SearchResults page

In the content panel of the SearchResults.xaml page, I've removed everything, then added a TextBlock control and named it LocationTextBlock.

```
~  
29      <!--ContentPanel - place additional content here-->  
30      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">  
31          <TextBlock Name="LocationTextBlock" />  
32      </Grid>  
33  </Grid>  
~
```

Now, I'll write C# code to retrieve the values from the query string and display them in the LocationTextBlock ... just to make I can grab those values that were passed in the query string.

```
~  
13      public partial class SearchResults : PhoneApplicationPage  
14  {  
15      private double _latitude;  
16      private double _longitude;  
17  
18  public SearchResults()  
19  {  
20      InitializeComponent();  
21  
22      Loaded += SearchResults_Loaded;  
23  }  
24  
25  void SearchResults_Loaded(object sender, RoutedEventArgs e)  
26  {  
27      LocationTextBlock.Text = string.Format("Location: {0} & {1}",  
28          _latitude.ToString(),  
29          _longitude.ToString());  
30  }  
31  
32  protected override void OnNavigatedTo(NavigationEventArgs e)  
33  {  
34      base.OnNavigatedTo(e);  
35  
36      _latitude = Convert.ToDouble(NavigationContext.QueryString["latitude"]);  
37      _longitude = Convert.ToDouble(NavigationContext.QueryString["longitude"]);  
38  }  
39  }  
~
```

1. I create two private fields (variables) to hold the values I'll retrieve from the query string.
2. The OnNavigatedTo() event will fire after the page class has been instantiated, but before the Page\_Loaded event fires. This is a perfect opportunity to process the query string. I use the NavigationContext class' Query string property array to get at the individual items in the query string by their name. I convert their data type from string to Double.
3. I wire up the Loaded event for this page.

4. When the Loaded event fires, I set the Text value Formatting in the latitude and longitude.

Does it work?



Great!

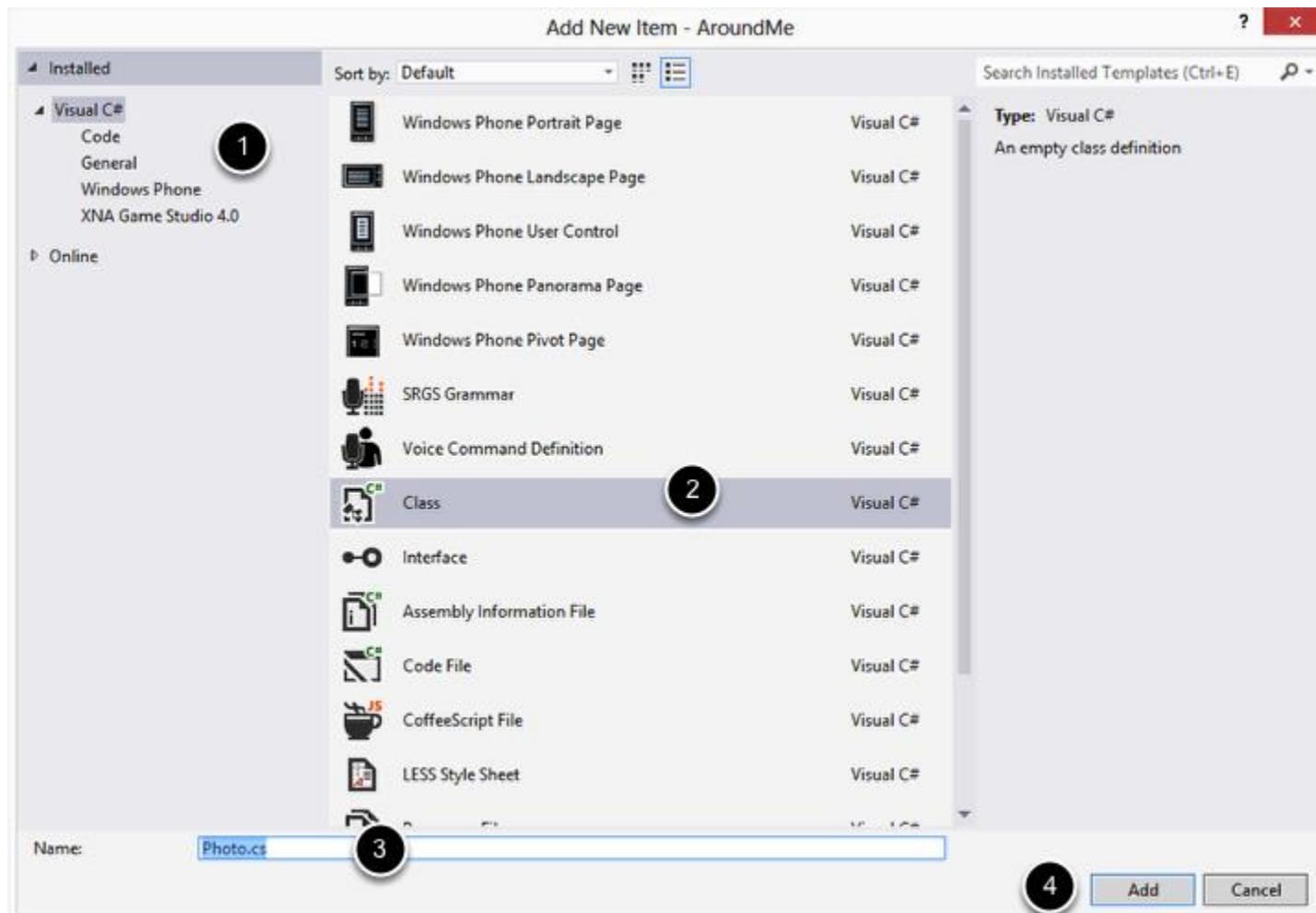
Next, I'll take my first step towards refactoring the code I've written. I'm using that term (refactor) loosely in this context to mean I'm going to re-organize the code more logically. Refactoring is the process of improving the quality (readability, structure, organization, etc.) of the code without changing its function.

## 8. Refactor the classes into a separate file

At the moment, I have code in places it should not be. For example, I've implemented three classes:

- Photo
- Photos
- FlickrData

... at the bottom of my MainPage.xaml.cs. We can do better than that.



Right-click on the project name, select Add | New Item ... from the context menu. The Add New Item dialog appears ...

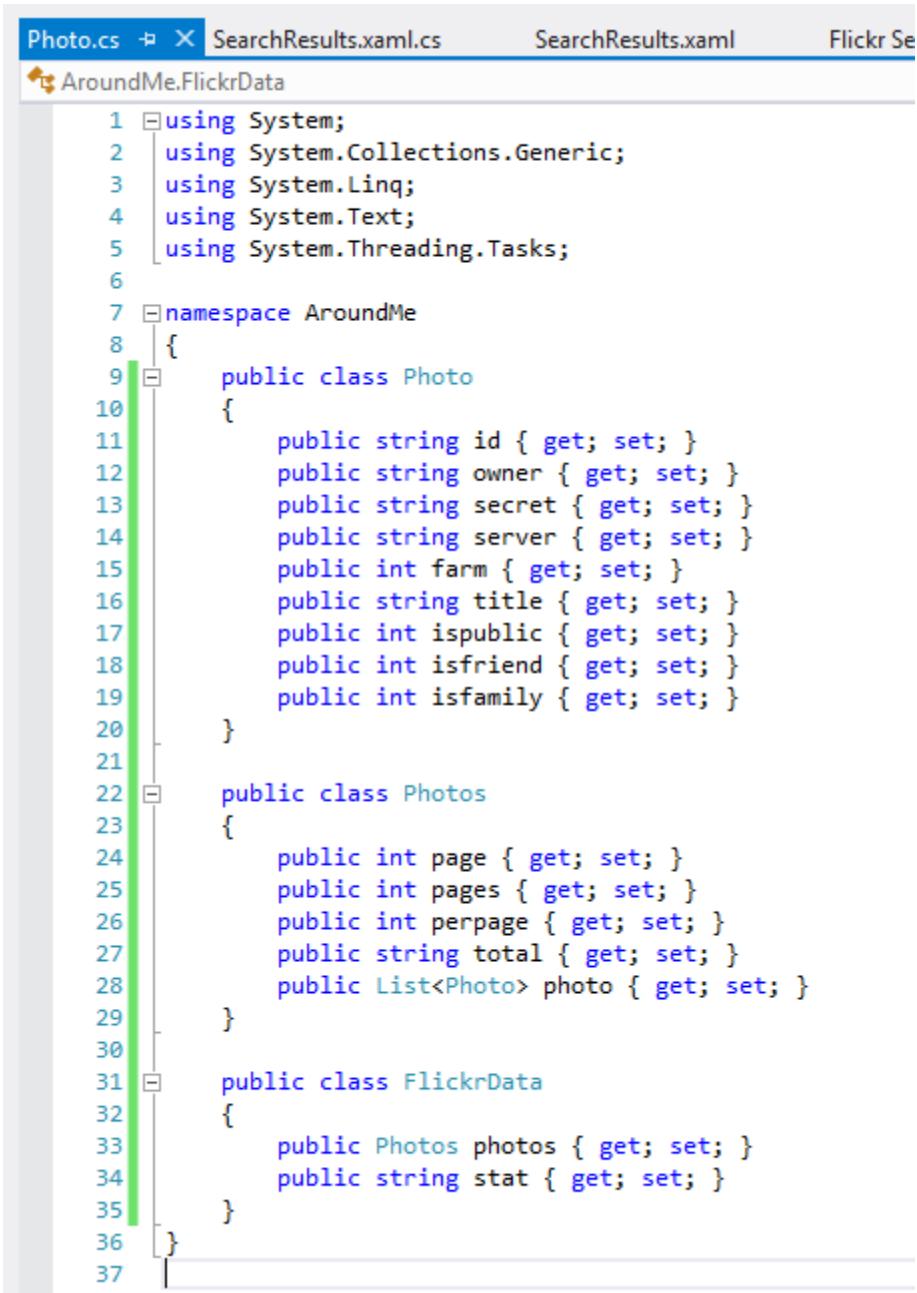
1. Make sure you're in the Visual C# item templates
2. Select the Class item template
3. Change the name to: Photo.cs
4. Click Add

Next, open the MainPage.xaml.cs file, highlight the Photo, Photos and FlickrData class definitions ...

```
153
154     public class Photo
155     {
156         public string id { get; set; }
157         public string owner { get; set; }
158         public string secret { get; set; }
159         public string server { get; set; }
160         public int farm { get; set; }
161         public string title { get; set; }
162         public int ispublic { get; set; }
163         public int isfriend { get; set; }
164         public int isfamily { get; set; }
165     }
166
167     public class Photos
168     {
169         public int page { get; set; }
170         public int pages { get; set; }
171         public int perpage { get; set; }
172         public string total { get; set; }
173         public List<Photo> photo { get; set; }
174     }
175
176     public class FlickrData
177     {
178         public Photos photos { get; set; }
179         public string stat { get; set; }
180     }
181 }
```

... cut them out of that file, and then paste them into the new Photo.cs file.

Photo.cs should look like this:



The screenshot shows a Windows application window with a tab bar at the top. The active tab is "Photo.cs". Other tabs include "SearchResults.xaml.cs", "SearchResults.xaml", and "Flickr Se...". Below the tabs is a code editor containing C# code. A vertical green bar highlights the entire class definition.

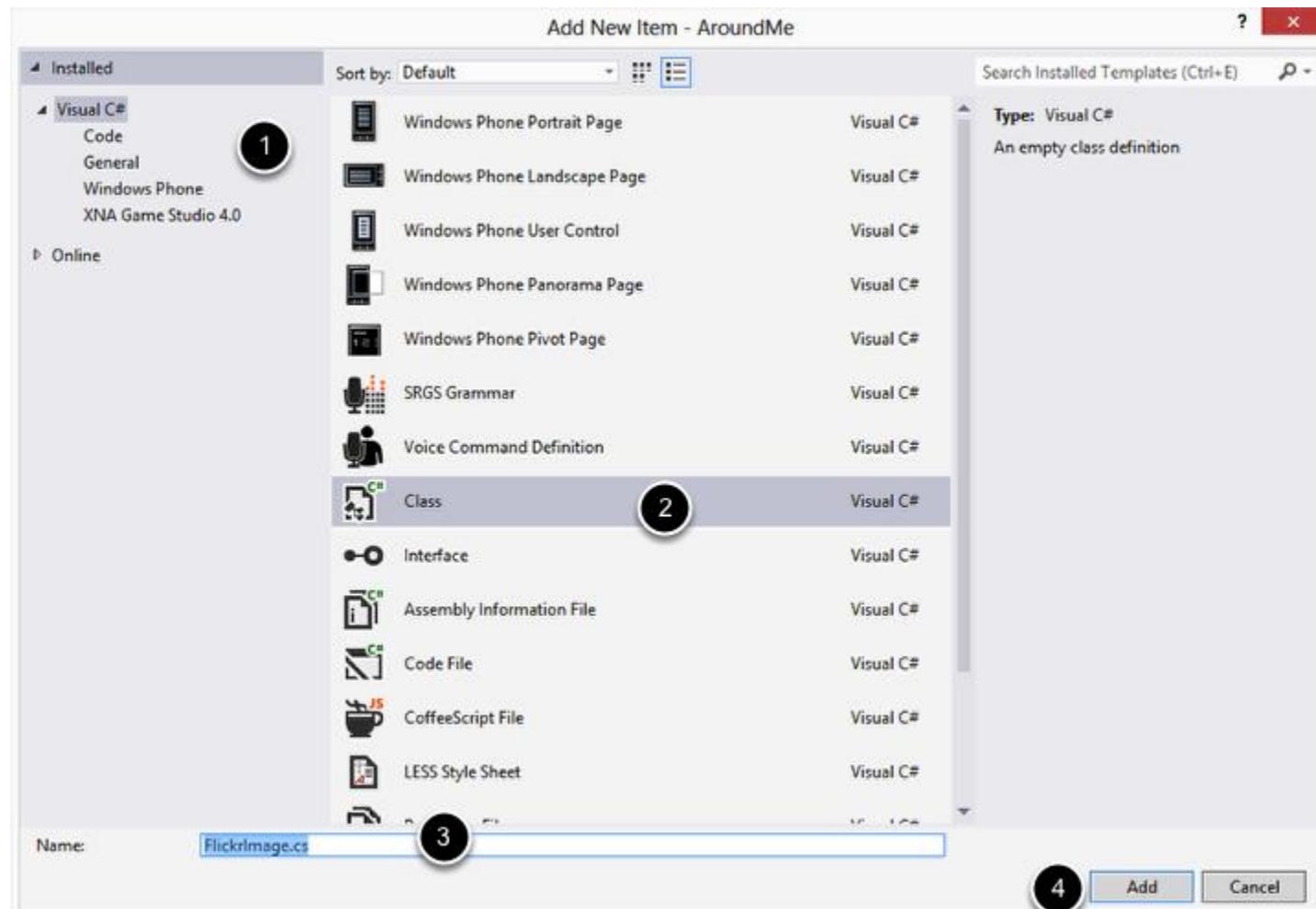
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AroundMe
8  {
9      public class Photo
10     {
11         public string id { get; set; }
12         public string owner { get; set; }
13         public string secret { get; set; }
14         public string server { get; set; }
15         public int farm { get; set; }
16         public string title { get; set; }
17         public int ispublic { get; set; }
18         public int isfriend { get; set; }
19         public int isfamily { get; set; }
20     }
21
22     public class Photos
23     {
24         public int page { get; set; }
25         public int pages { get; set; }
26         public int perpage { get; set; }
27         public string total { get; set; }
28         public List<Photo> photo { get; set; }
29     }
30
31     public class FlickrData
32     {
33         public Photos photos { get; set; }
34         public string stat { get; set; }
35     }
36 }
37
```

If this were a real project, I would probably separate each of these into their own files. I do this as a convention for consistency ... each class should be in its own file so that I can easily locate every class in my project via the Solution Explorer. But that is merely a stylistic preference of mine. This will work just fine.

Next, I'm going to implement a class that will wrap the call to the Flickr API web service call.

## 9. Implement the FlickrImage.cs class

Right-click the project name in the Solution Explorer, select Add | New Item ... from the context menu. When the Add New Item dialog appears ...



1. Make sure the Visual C# item templates are selected
2. Select the Class item template
3. Change the name to: FlickrImage.cs
4. Click Add

Next will involve writing a lot of code. I'll explain it after the code passage below:

```

11  public class FlickrImage
12  {
13      public Uri Image320 { get; set; }
14      public Uri Image1024 { get; set; }
15
16      public async static Task<List<FlickrImage>> GetFlickrImages(
17          string flickrApiKey,
18          double latitude = double.NaN,
19          double longitude = double.NaN)
20      {
21          HttpClient client = new HttpClient();
22
23          string baseUrl = getBaseUrl(flickrApiKey, latitude, longitude);
24
25          string flickrResult = await client.GetStringAsync(baseUrl);
26
27          FlickrData apiData =
28              JsonConvert.DeserializeObject<FlickrData>(flickrResult);
29
30          List<FlickrImage> images = new List<FlickrImage>();
31
32          if (apiData.stat == "ok")
33          {
34              foreach (Photo data in apiData.photos.photo)
35              {
36                  FlickrImage img = new FlickrImage();
37
38                  // To retrieve one photo, use this format:
39                  //http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}{size}.jpg
40
41                  string photoUrl = "http://farm{0}.staticflickr.com/{1}/{2}_{3}";
42
43                  string baseFlickrUrl = string.Format(photoUrl,
44                      data.farm,
45                      data.server,
46                      data.id,
47                      data.secret);
48
49                  img.Image320 = new Uri(baseFlickrUrl + "_n.jpg");
50                  img.Image1024 = new Uri(baseFlickrUrl + "_b.jpg");
51
52                  images.Add(img);
53              }
54          }
55      }
56  }

```

The FlickrImage class has a dual purpose. (a) An instance of this class represents a single image's Flickr URIs at different image resolutions, and (b) It has a public static method (GetFlickrImages()) that knows how to call Flickr's API and retrieve a generic List<FlickrImage>. Perhaps it would be a good idea to separate these into two separate classes and class files, but I'll leave it as is for now.

1. I implement the two instance member properties representing the URIs pointing to two sizes of each Flickr image we'll use in this project. The smaller image will be used in the SearchResults page, and the larger image will be used on the lock screen much later.

The rest of this code passage is the public static method that will create a `List<FlickrlImage>` by calling Flickr's API. We moved most of this code from the `MainPage.xaml.cs` `UpdateMap()` method to its new home here.

2. This public static method returns a `Task<List<FlickrlImage>>` ... what is this `Task<>` that surrounds the `List<FlickrlImage>`? I'm going to devote a whole lesson to talking about the `async` features we're using in this app, and explain the new `Async` features in C#. It's the reason behind the `Task<>` keyword as well as the `async` keyword. For now, I just ask that you trust me ... it needs to be there and I'll explain why later.

The method's signature requires the Flickr API key is passed in along with a latitude and longitude input parameter each defaulted to `double.NaN`, which means, Not a Number. In other words, if the caller of this method doesn't pass in a value, we don't want it defaulting to 0 because 0 represents a real coordinate. We'd rather use `NaN` and check for that later to ensure that we are in fact working with a real number or not. We'll implement that check later as we try to harden our application against potential problems.

3. Working with Flickr's API requires a URL complete with query string parameter values. I foresee that the process of creating that query string can be messy, so I'm going to remove that and put it in a helper method that I'll implement later in this lesson. To construct that URL I'll need many of the same parameters that were passed into this method, so I pass them along as input parameters to the soon-to-be-implemented `getBaseUrl()` method. By moving the complex string manipulation required to call the search API into its own helper method, it keeps this method more readable.
4. We want to convert the `Photo` class instances into the new instances of the `FlickrlImage` class since this is just the information we'll need to display in our results. In other words, we only really need to know the two sizes of images ... one for the smaller thumbnails and one for the actual lock screen on our phone. We'll need to build URLs for these two versions of the image based on other information provided to us in the `Photo` class. So, we're basically just simplifying the data returned back to us by creating a new, simplified, streamlined list of objects, each with two properties.
5. Here we set the two properties of our new `FlickrlImage` object to URIs. The suffix identifies the two versions / sizes of the photos we want to download.
6. Finally, we'll return the `List<FlickrlImages>` to the caller.

Let's circle back and revisit callout #3 (above) ... we wanted to separate all the complex string manipulation logic required to construct a URL for the search API query to its own private helper method called `getBaseUrl()`.

```
5/
58     private static string getBaseUrl(string flickrApiKey,
59         double latitude = double.NaN,
60         double longitude = double.NaN)
61     {
62         // About licenses:
63         // http://www.flickr.com/services/api/flickr.photos.licenses.getInfo.html
64         /*
65             * <license id="4" name="Attribution License" url="http://creativecommons.org/licenses/by/2.0/"/>
66             * <license id="5" name="Attribution-ShareAlike License" url="http://creativecommons.org/licenses/by-sa/2.0/"/>
67             * <license id="6" name="Attribution-NoDerivs License" url="http://creativecommons.org/licenses/by-nd/2.0/"/>
68             * <license id="7" name="No known copyright restrictions" url="http://flickr.com/commons/usage/"/>
69         */
70         string[] licenses = { "4", "5", "6", "7" };
71         string license = String.Join(", ", licenses);
72         license = license.Replace(", ", "%2C");
73
74         if (!double.IsNaN(latitude))
75             latitude = Math.Round(latitude, 5);
76
77         if (!double.IsNaN(longitude))
78             longitude = Math.Round(longitude, 5);
79
80         // Search API
81         // http://www.flickr.com/services/api/flickr.photos.search.html
82
83         string url = "http://api.flickr.com/services/rest/" +
84             "?method=flickr.photos.search" +
85             "&license={0}" +
86             "&api_key={1}" +
87             "&lat={2}" +
88             "&lon={3}" +
89             "&radius=2" +
90             "&format=json" +
91             "&nojsoncallback=1";
92
93         var baseUrl = string.Format(url,
94             license,
95             flickrApiKey,
96             latitude,
97             longitude);
98
99         return baseUrl;
100    }
101 }
102 }
103 }
```

Hopefully nothing here requires much explanation. We're merely moving this functionality out of our MainPage.xaml.cs's unwieldy UpdateMap() method to a new

location. We're hiding all this complex string manipulation behind a friendly method name to make the code more readable.

**IMPORTANT:** I did make two significant changes here due to small bugs that I discovered in the original code.

1. I replaced:

```
license.Replace(", ", "%2C");
... with ...
license = license.Replace(", ", "%2C");
... because the Replace method returns a string. It does not merely operate and set the
string it is called on.
```

2. The second problem will only surface if you attempt to use this app on your physical Phone device, or perhaps if you choose a different location than I've selected in the Emulator. Actually, it's not a problem with our app, per se, it's a problem with Flickr's API. It will only accept 5 digits of precision after the decimal point. If you send it more than 5 digits of precision, it will never return a result. So, we need to use the Math.Round() method to ensure that we only send up to 5 digits, or rather, places of precision.

## 8. Refactor the layout and display logic

Now the UpdateMap() method can be greatly simplified because we've offloaded the responsibility of displaying results to the SearchResults.xaml.cs page.

I'll remove everything except just the code required to update the map with the current GPS position of the phone:

```
--  
36     private async void UpdateMap()  
37     {  
38         Geolocator geolocator = new Geolocator();  
39         geolocator.DesiredAccuracyInMeters = 50;  
40  
41         Geoposition position =  
42             await geolocator.GetGeopositionAsync(  
43                 TimeSpan.FromMinutes(1),  
44                 TimeSpan.FromSeconds(30));  
45  
46         var gpsCoorCenter =  
47             new GeoCoordinate(  
48                 position.Coordinate.Latitude,  
49                 position.Coordinate.Longitude);  
50  
51         //AroundMeMap.SetView(new GeoCoordinate(41.8988D, -87.6231D), 17D);  
52  
53         AroundMeMap.Center = gpsCoorCenter;  
54         AroundMeMap.ZoomLevel = 15;  
55  
56         /*  
57             ResultTextBlock.Text = string.Format("{0} - {1}",  
58                 AroundMeMap.Center.Latitude,  
59                 AroundMeMap.Center.Longitude);  
60         */  
61     }  
62  
63 }
```

Next, I can remove the (temporary) XAML controls I was using to check my results ... namely, I remove the Image control named FlickrImage and the TextBlock control named ResultTextBlock.

```
61
62     <!--ContentPanel - place additional content here-->
63     <Grid x:Name="ContentPanel"
64         Grid.Row="1"
65         Margin="12,0,12,0" >
66         <Grid.RowDefinitions>
67             <RowDefinition Height="50" />
68             <RowDefinition Height="200" />
69             <RowDefinition Height="*" />
70         </Grid.RowDefinitions>
71
72         <!--
73             <TextBlock Name="ResultTextBlock" Grid.Row="0" />
74             <Image Name="FlickrImage" Grid.Row="1" Stretch="UniformToFill" />
75             -->
76             <maps:Map Name="AroundMeMap" Grid.Row="1" />
77
78     </Grid>
```

Now I need to visit the SearchResults.xaml.cs page and wire it up to:

1. retrieve the latitude and longitude from the query string once the page has been navigated to
2. kick off the search to Flickr once the page has been loaded.

```

11  namespace AroundMe
12  {
13      public partial class SearchResults : PhoneApplicationPage
14      {
15          private double _latitude;
16          private double _longitude;
17
18          // Your API key ... REPLACE THIS WITH YOURS:
19          // http://www.flickr.com/services/api/keys/
20          private const string FlickrApiKey = "bdd5-----288";
21
22      public SearchResults()
23      {
24          InitializeComponent();
25
26          Loaded += SearchResults_Loaded;
27      }
28
29      protected async void SearchResults_Loaded(object sender, RoutedEventArgs e)
30      {
31          /*
32          LocationTextBlock.Text = string.Format("Location: {0} & {1}",
33              _latitude.ToString(),
34              _longitude.ToString());
35          */
36
37          var images = await FlickrImage.GetFlickrImages(
38              FlickrApiKey,
39              _latitude,
40              _longitude);
41
42          DataContext = images;
43      }
44
45      protected override void OnNavigatedTo(NavigationEventArgs e)
46      {
47          base.OnNavigatedTo(e);
48
49          _latitude = Convert.ToDouble(NavigationContext.QueryString["latitude"]);
50          _longitude = Convert.ToDouble(NavigationContext.QueryString["longitude"]);
51      }
52  }
53

```

1. Rather than hard code it in the FlickrImage class, I decided to move the FlickrApiKey out to the SearchResults.xaml.cs file as a constant. The reason is so that I could potentially re-use the FlickrImage class again later in another app. I would need a different FlickrApiKey. It makes sense to pass that in at runtime.
2. I comment out the outdated reference to the LocationTextBlock. We no longer need that. It was just to help us get our bearings, but we're ready to move past that. Later, I'll delete these comments.

3. Here I make the call to `FlickrlImage.GetFlickrlImages()`. This kicks off the search. It's awaited, meaning our code is calling an await-able / `async` method. The flow of code can proceed, as long as we don't need the results (i.e., the images) from this call.
4. We set the `DataContext` for the `SearchResults.xaml` page. We'll use this as we bind a `LongListSelector` to the thumbnail images returned from line 37.

Finally, we'll add a `LongListSelector`, style its `DataTemplate` and set its bindings.

```

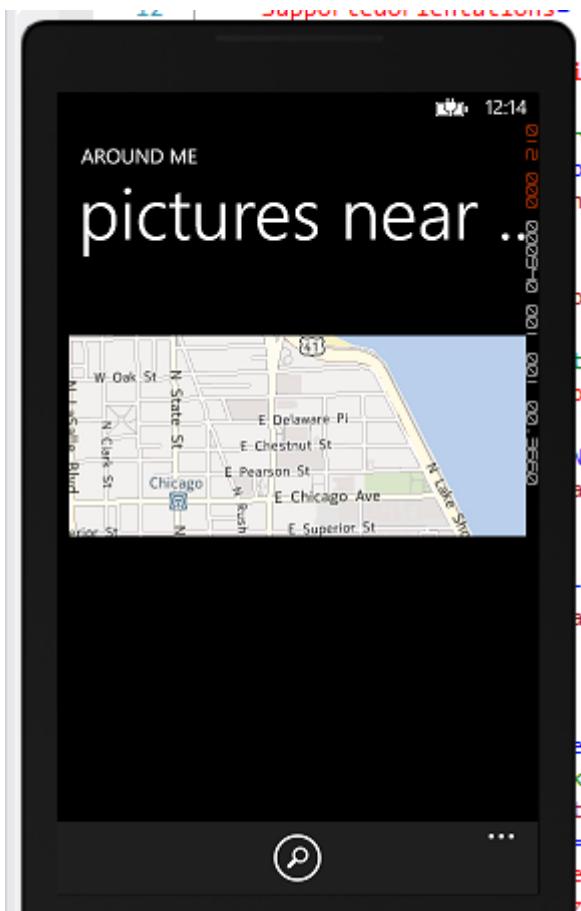
22
23      <!--TitlePanel contains the name of the application and page title-->
24      <StackPanel Grid.Row="0" Margin="12,17,0,28">
25          <TextBlock
26              Text="AROUND ME"
27              Style="{StaticResource PhoneTextNormalStyle}"/>
28          <TextBlock
29              Text="pick for background"
30              Margin="9,-7,0,0"
31              Style="{StaticResource PhoneTextTitle1Style}"/>
32      </StackPanel>
33
34      <!--ContentPanel - place additional content here-->
35      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
36          <!-- <TextBlock Name="LocationTextBlock" /> -->
37          1   <phone:LongListSelector
38              LayoutMode="Grid"
39              ItemsSource="{Binding}"
40              GridCellSize="105, 105">
41                  <phone:LongListSelector.ItemTemplate>
42                      2   <DataTemplate>
43                          <Image
44                              Source="{Binding Image320}"
45                              Stretch="UniformToFill" />
46                      </DataTemplate>
47                  </phone:LongListSelector.ItemTemplate>
48              </phone:LongListSelector>
49          </Grid>
50      </Grid>
51

```

1. The `SearchResults.xaml` page will mostly be comprised of a `LongListSelector` bound to the `DataContext`, i.e., the result of our call to `FlickrlImage.GetFlickrlImage()`, which will be a `List<FlickrlImage>`. The `FlickrlImage` has two properties ... one representing the `Uri` for the thumbnail, and one representing the larger image (that we'll use for the lock screen). This `LongListSelector` will be in `Grid` layout mode, with each cell `105 x 105`. We're binding to the `DataContext` of the page, so no need for additional information beyond the binding expression `{Binding}`.

2. Here we build the content for each cell of the LongListSelector grid ... an Image control in each cell. It will bind to the Image320 property of the DataSource (which was a List<FlickrImage>, so we're binding to the FlickrImage.Image320 property).

When we run the app ...



... and perform a search by clicking the magnifying glass icon ...



We see results in or around the John Hancock Center. Awesome!

### Recap

To quickly recap, the big takeaway in this lesson was navigating to a new page, how restructured and refactored the code so that its implementation is cleaner and more obvious -- you'll see the value of this reorganization as we continue this series and reuse this code from other places. I'm really just planning ahead a little. While I don't think we learned anything new with regards to binding and such, we did get another chance to exercise things that we learned when working on the SoundBoard again.

# Part 28: Understanding Async and Awaitable Tasks

Source Code: <http://aka.ms/absbeginnerdevwp8>

In this lesson, I want to talk about the keywords we've added to several spots in our source code. The keywords I'm referring to are:

1. `async`
2. `await`
3. `Task<T>`

You see all three of them in the `GetFlickrImages()` method:

```
15
16 public async static Task<List<FlickrImage>> GetFlickrImages(
17     string flickrApiKey,
18     double latitude = double.NaN,
19     double longitude = double.NaN)
20 {
21     HttpClient client = new HttpClient();
22
23     string baseUrl = getBaseUrl(flickrApiKey, latitude, longitude);
24
25     string flickrResult = await client.GetStringAsync(baseUrl);
26
27     FlickrData apiData =
28         JsonConvert.DeserializeObject<FlickrData>(flickrResult);
29 }
```

The code snippet shows the `GetFlickrImages()` method. Three specific lines are highlighted with red arrows pointing to them from the left margin:

- `public async static Task<List<FlickrImage>> GetFlickrImages(`
- `string flickrResult = await client.GetStringAsync(baseUrl);`
- `FlickrData apiData =`

These keywords are newly added to C# 5.0 and are generally referred to as the new "async"—short for "asynchronous" or "asynchrony"—feature. In a nut shell, this new feature is a simplified way of improving the performance of the app and making it more responsive to the user without the complexity of writing code to use multiple threads. If you call a method of the Windows Phone API or some other library supporting `async` that could potentially take a "long time", the method will say, "I promise to get those results to you as soon as possible, so go on about your business and I'll let you know when I'm done". The app can then continue executing, and can even exit out of method contexts. Another word for this in computer programming terminology is a "promise".

Under the hood, when you compile this source code, the Compiler picks apart the source code and implements a complex series of statements in the Intermediate Language to allow this to happen flawlessly and it does it without the use of multiple threads in most cases.

Async is best used for operations that have a high degree of latency, but are not compute intensive. So, for example, we want to retrieve data from a Web API like we're

doing in the `GetFlickrImages()` method. Before `async`, our app would block the execution of code waiting for Flickr's web service to reply with the data we requested. That could take a second, or two, or three, which is an eternity in computing. But the Phone's PROCESSOR is not busy at all. It's just sitting there, waiting for a reply from the Flickr web service. This is known as "I/O Bound" ... I/O means "In / Out" ... so things like the file system, the network, the camera, etc., involve "I/O bound operations" and are good candidates for `async`. In fact, the Windows Phone API designers decided to bake `async` into all I/O Bound operations, forcing you to use this to keep the phone responsive to the end user.

Contrast that to a compute-intensive operation such as a photo filter app that must take a large image from the camera and run complex mathematical algorithms to change the colors or the position of each pixel in the image. That could take a long time, too, but in that case, the Phone's processor is hard at work. This type of operation is known as "CPU Bound". This is NOT a good use of `async`. In this case, you would want to consider a Background Worker which helps to manage threads on the Windows Phone platform. If you are developing in .NET, you may prefer to work with threading via the Task Parallel Library instead, or revert back to managing threads manually, which has been an option since the very earliest versions of .NET.

Understanding multi-threading, parallel programming, the Task Parallel Library, even Background Workers on the Windows Phone API are WAY beyond the scope of this series and this lesson. It's a large topic and I'll not lie to you—it makes my head spin. If you want to learn a little more about `async` and how it applies to the Windows RunTime, check out:

### **Working with Async Methods in the Windows Runtime**

<http://channel9.msdn.com/Series/Windows-Store-apps-for-Absolute-Beginners-with-C-/Part-12-Working-with-Async-Methods-in-the-Windows-Runtime>

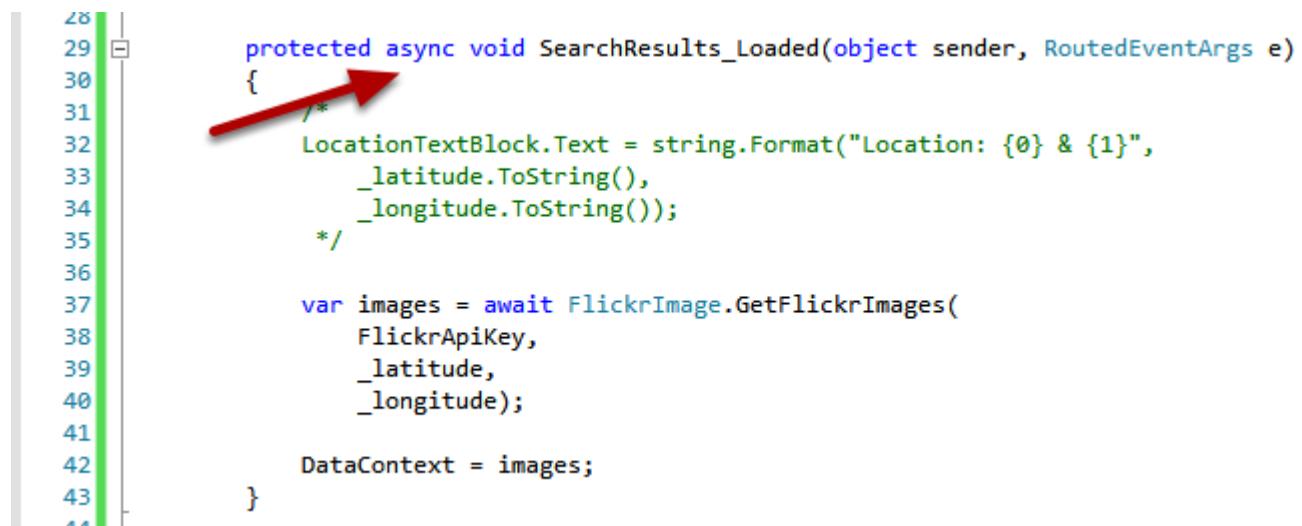
... and be sure to read the comments where I further explain this idea.

But back to the topic at hand ... `async` is for those common occasions when you have a blocking operation that is not compute intensive, such as our case here where we are waiting for the `HttpClient.GetStringAsync()` operation to complete. In this case, we use the `await` keyword, which says, "I'll get back to you when I'm finished". The thread of execution can continue on through the other lines of code until it absolutely must have the results of that operation. Now, in our case, we attempt to call `JsonConvert.DeserializeObject<FlickrData>` in the very next line of code, so little advantage is gained. However, we must still use the `await` keyword because the `HttpClient.GetStringAsync()` method returns a `Task<T>`. That method requires we `await`. You'll see `await`-able methods with a common convention ... they all end with the suffix `Async`.

Furthermore, any method that uses an `await`-able method must be marked with the `async` keyword in their method signature, AND if it is supposed to return a value it must

return the value wrapped with a Task<T>. If the method is void, it will merely be marked void in the method signature. Since the HttpClient.GetStringAsync() is marked with await, and our method returns a List<FlickrImage> we must mark our method with async and wrap the List<FlickrImage> with a Task<T>. That essentially means that any method that calls our method can continue executing until it absolutely needs that List<FlickrImage>.

Therefore, take a look at the method that calls our FlickrImage.GetFlickrImages() method:



```
28
29 protected async void SearchResults_Loaded(object sender, RoutedEventArgs e)
30 {
31     /*
32     LocationTextBlock.Text = string.Format("Location: {0} & {1}",
33         _latitude.ToString(),
34         _longitude.ToString());
35     */
36
37     var images = await FlickrImage.GetFlickrImages(
38         FlickrApiKey,
39         _latitude,
40         _longitude);
41
42     DataContext = images;
43 }
```

That's why in line 37 (above) we must use the await keyword—our FlickrImage.GetFlickrImages() was forced to be await-able (since it used the HttpClient.GetStringAsync()). Notice that we also had to modify the event handler method's method signature adding the async keyword. Since this is what they refer to as a "top level method", the chain of adding async stops here.

Again, remind me, why are we doing all of this? In hopes of making our app more responsive. This operation should not be blocking the Phone's processor from taking on other tasks like answering a phone call or running background tasks on behalf of other apps. It won't prevent the user from selecting any of the hardware buttons on the phone and getting an instant response.

## Recap

To recap, the big take away in this lesson is what async is, how it works, the scenarios it was designed to address, and its overall purpose—to keep the phone and your apps responsive during long running I/O bound operations.

# Part 29: Filtering the Results by Keyword

Source Code: <http://aka.ms/absbeginnerdevwp8>

Currently, we can find all images on Flickr based on the locale of the device, however in our original "low tech mockup" design, we wanted to allow the user to filter the results by a keyword or search phrase. We'll enable that feature in this lesson.

Our game plan in this lesson ...

1. We'll modify the layout of the MainPage.xaml to include a Topic textbox that will allow the end user to type in specific search terms to include in our search of images from Flickr.
2. We'll then take that topic and send it to Flickr's web service. That will require that we change the code in a few places to accommodate the passing and receiving of the topic value.
3. We'll then add the concept of radius to the app allowing us to specify the distance from the latitude and longitude for Flickr to include in its search. I'll also refactor the code to harden it, or rather, make it impervious to accidental or malicious values that are nonsensical. We'll practice "defensive programming".

1. Edit MainPage.xaml: clean up layout, add Search text box

The screenshot shows the XAML code for MainPage.xaml. A red box highlights a section of code from line 38 to line 41. A green vertical bar is on the left, and a red circle with the number 1 is on the right. Another red box highlights a section of code from line 45 to line 52. A green vertical bar is on the left, and a red circle with the number 2 is on the right.

```
34
35     <Grid x:Name="ContentPanel"
36         Grid.Row="1"
37         Margin="12,0,12,0" >
38     <Grid.RowDefinitions>
39         <RowDefinition />
40         <RowDefinition Height="Auto" />
41     </Grid.RowDefinitions>
42
43     <maps:Map Name="AroundMeMap" />
44
45     <StackPanel Grid.Row="1">
46         <TextBlock
47             Foreground="{StaticResource PhoneSubtleBrush}"
48             Text="Topic" />
49         <TextBox
50             Name="SearchTopic"
51             Margin="-12, 0" />
52     </StackPanel>
53 </Grid>
```

Not depicted in this screenshot, I remove almost all of the boilerplate comments and the comments I've added previously.

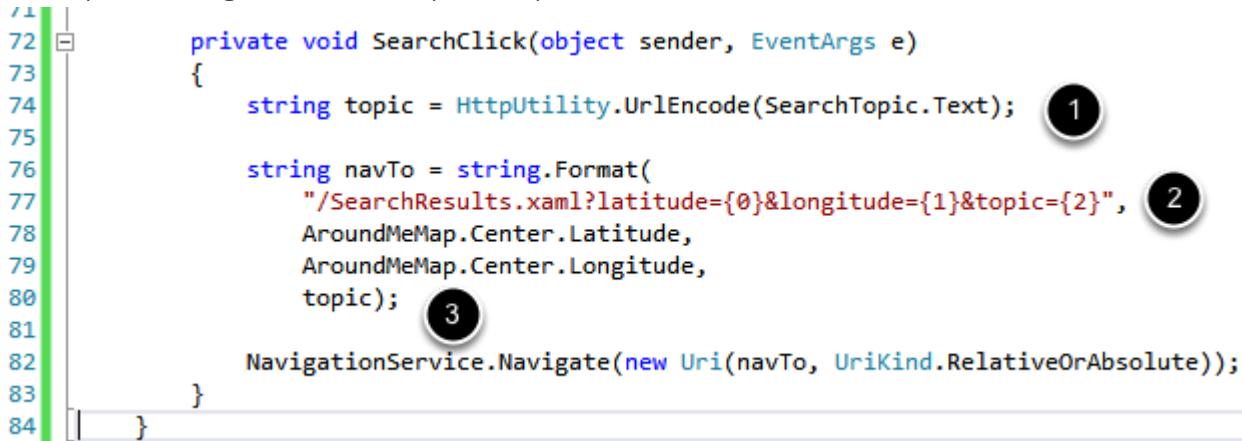
1. I change the number of RowDefinitions, as well as their heights. In line 39 I didn't specify a height, meaning I want the default value of \*, or rather, star sizing ... "take the remaining available space". Line 40 sets the row height to Auto which means "distribute space evenly based on the size of the content that is within a column or row."
2. We add a StackPanel as a container for the layout of our new Search bar at the bottom of the MainPage.xaml. The StackPanel contains a TextBlock and TextBox to allow for entry. The margin looks like it is missing two values, however that's simple a shortcut for writing:

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

... it simply means "repeat these values".

Now, we'll need to add this notion of "Topic" throughout the entire application. We'll pass it along with the position to the SearchResults.xaml page, we'll pass it to the Flickr web service call, and so on.

2. Modify the Navigation code to pass Topic to SearchResults.xaml



```

72     private void SearchClick(object sender, EventArgs e)
73     {
74         string topic = HttpUtility.UrlEncode(SearchTopic.Text); 1
75
76         string navTo = string.Format(
77             "/SearchResults.xaml?latitude={0}&longitude={1}&topic={2}", 2
78             AroundMeMap.Center.Latitude,
79             AroundMeMap.Center.Longitude,
80             topic); 3
81
82         NavigationService.Navigate(new Uri(navTo, UriKind.RelativeOrAbsolute));
83     }
84 }
```

In the MainPage.xaml.cs, edit the SearchClick() method ...

1. We are passing data in query strings ... We talked about query strings already. In this case, I'm concerned that a user may enter characters in the SearchTopic.Text that could be interpreted incorrectly. For example, using an ampersand or question mark could harm the interpretation of the query string. The UrlEncode() method can be used to encode the query-string values or even the entire URL. If characters such as blanks and punctuation are passed in an HTTP stream without encoding, they might be misinterpreted at the receiving end. URL encoding converts characters that are not allowed in a URL into character-entity equivalents. For more information about URL Encoding, check out: <http://msdn.microsoft.com/en-us/library/4fkewx0t.aspx>

However, whenever you're working with query strings, whether in HTML or XAML, it's always a good idea to URL Encode, ESPECIALLY if what you're passing in the query string is input by the end user.

2. In the string.Format, I add the "&topic={2}" to pass the Search Topic along to the next page, and ...
3. Add the Url Encoded topic variable to substitute for the third position in the string.

3. Modify the Navigation code to RECEIVE the search topic in the SearchResults.xaml  
 First I'll create a private variable to hold the topic value I'm passing to the  
 SearchResults.xaml page ...

```

11  namespace AroundMe
12  {
13      public partial class SearchResults : PhoneApplicationPage
14      {
15          private double _latitude;
16          private double _longitude;
17          private string _topic; ←
18      }

```

Next, I'll retrieve the topic value from the query string and save it in the new private variable ...

```

39
40      protected override void OnNavigatedTo(NavigationEventArgs e)
41      {
42          base.OnNavigatedTo(e);
43
44          _latitude = Convert.ToDouble(NavigationContext.QueryString["latitude"]);
45          _longitude = Convert.ToDouble(NavigationContext.QueryString["longitude"]);
46          _topic = NavigationContext.QueryString["topic"]; ←
47      }

```

While I'm here, I decide to go ahead and modify my call to the FlickrlImage.GetFlickrlImage() method. I'll need to pass the topic value to the GetFlickrlImages() method that ultimately calls the Flickr web api. Since I've already made the latitude and longitude optional / defaulted input parameters to my method, the topic value must be passed in prior to those (unless I choose to make it optional as well ... but I've chosen not to.)

```
47
30     protected async void SearchResults_Loaded(object sender, RoutedEventArgs e)
31     {
32         var images = await FlickrImage.GetFlickrImages(
33             FlickrApiKey,
34             _topic, ←
35             _latitude,
36             _longitude
37         );
38
39         DataContext = images;
40     }
41
```

Now that I've added the topic in the call, I need to modify the `FlickrImage.GetFlickrImages()` method itself:

```
15
16     public async static Task<List<FlickrImage>> GetFlickrImages(
17         string flickrApiKey,
18         string topic, ①
19         double latitude = double.NaN,
20         double longitude = double.NaN
21     )
22     {
23         HttpClient client = new HttpClient();
24
25         string baseUrl = getBaseUrl(flickrApiKey, topic, latitude, longitude); ②
26
27         string flickrResult = await client.GetStringAsync(baseUrl);
28     }
```

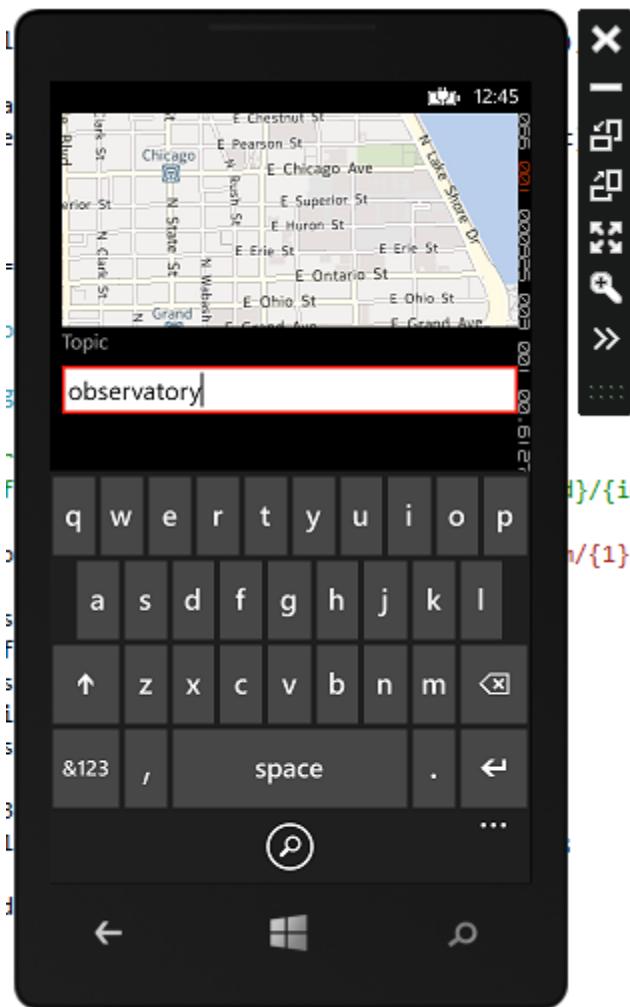
1. I modify the method signature by adding the topic input parameter, and ...
2. I pass topic to the `getBaseUrl()` method call. Now I'll need to modify THAT method as well ...

```

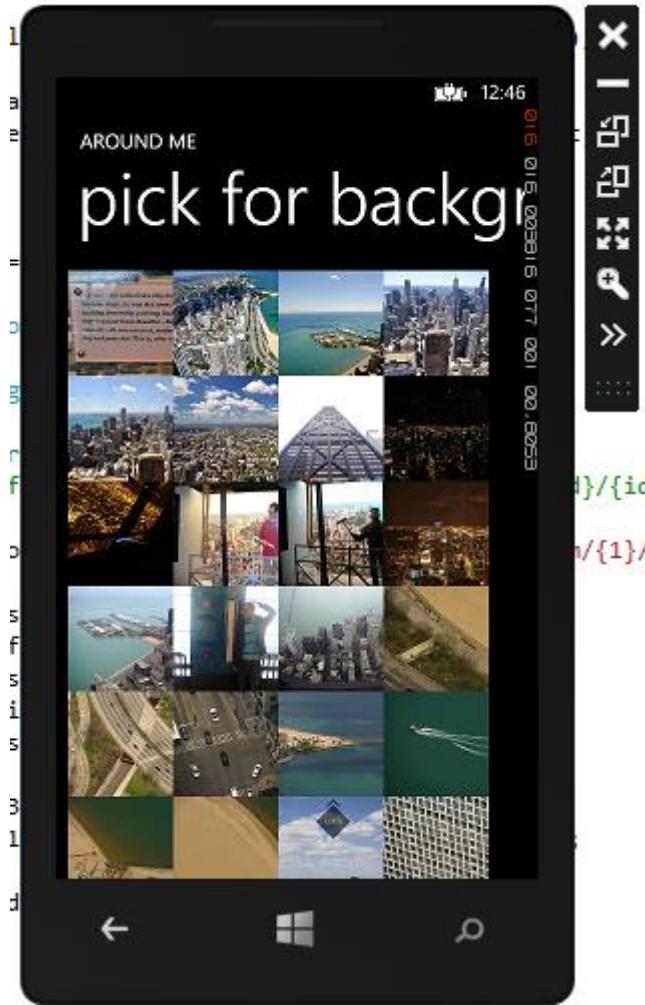
60     private static string getBaseUrl(string flickrApiKey,
61         string topic, 1
62         double latitude = double.NaN,
63         double longitude = double.NaN)
64     {
65         // About licenses:
66         // http://www.flickr.com/services/api/flickr.photos.getInfo.html
67         /*
68          * <license id="4" name="Attribution License" url="http://creativecommons.org/licenses/by/2.0/">
69          * <license id="5" name="Attribution-ShareAlike License" url="http://creativecommons.org/licenses/by-sa/2.0/">
70          * <license id="6" name="Attribution-NoDerivs License" url="http://creativecommons.org/licenses/by-nd/2.0/">
71          * <license id="7" name="No known copyright restrictions" url="http://flickr.com/commons/usage/">
72         */
73         string[] licenses = { "4", "5", "6", "7" };
74         string license = String.Join(", ", licenses);
75         license.Replace(",", "%2C");
76
77         // Search API
78         // http://www.flickr.com/services/api/flickr.photos.search.html
79
80         string url = "http://api.flickr.com/services/rest/" +
81             "?method=flickr.photos.search" +
82             "&license={0}" +
83             "&api_key={1}" +
84             "&lat={2}" +
85             "&lon={3}" +
86             "&text={4}" + 2
87             "&radius=2" +
88             "&format=json" +
89             "&nojsoncallback=1";
90
91         var baseUrl = string.Format(url,
92             license,
93             flickrApiKey,
94             latitude,
95             longitude,
96             topic); 3
97
98         return baseUrl;
99     }
100 }
101 }
```

- 1 I modify the getBaseUrl() method signature by adding the topic input parameter.
- 2 I add the "text" parameter to the Flickr API call, per their list of parameters.
- 3 I modify the line of code that replaces the placeholders with the actual values, adding topic at the end to be substituted into the new baseUrl string.

Now, I'm ready to test the results ... I'll search for "observatory" to limit the pics that are returned by the Flickr web service call to only those that have the term "observatory" in their description ...



I get a list of photos back that (a) look different from the list of images I was getting back, and (b) they do all look like they have something to do specifically with the observatory as opposed to a street level view.



However, I suppose I have no easy way to validate these results other than to perform a similar search on Flickr's own website look for similarities between the list I'm displaying in the AroundMe app and the results from their website. I'm satisfied with the results, so I'm not going to take the time to thoroughly check the results. I'll trust that it's working for now.

We've satisfied the title of this lesson by adding filtering on the results based on keywords. However, in the time that remains I want to add two improvements to our `GetFlickrImages()` method in the `FlickImage.cs` file ...

4. Adding radius and programming defensively
- The two improvements:

First, I'll add the concept of radius ... the Flickr web api allows us to pass in the radius it will search.

Second, I'll add some code "defensive code" ... the phrase "defensive programming" is a thought process or approach to writing code. In this case, I'll apply that thought process to mean that I want to do some checking of the query string values for latitude, longitude and topic that I'm passing from the MainPage to the Results, and I want to add double-quotes around the topic I want to pass to the Flickr API so the user can add a phrase. If you want to learn more about defensive programming from a high level, a great place to start is Wikipedia:

[http://en.wikipedia.org/wiki/Defensive\\_programming](http://en.wikipedia.org/wiki/Defensive_programming)

Let's begin by adding the notion of "radius" to our app. I'm not going to take the time to fully implement this in this series. Clint's original application that he gave me will adjust the radius we sent to Flickr based on how zoomed in or zoomed out we are in the Map control. However, I decided to leave that code out of this video series ... first, it will add a LOT of code to the application with some complicated calculations to convert map pixels to radius. Secondly, there's no way to simulate the pinch-in or pinch-out motion in the Windows Phone emulator. Perhaps once I'm finished with this series, I'll come back and add an addendum to how to tackle this scenario. So, for now, I'll merely add the feature for radius, and I'll hard-code the radius to 5 kilometers.

I'll modify the FlickrImage.cs ... specifically, I'll modify the getBaseUrl() method's signature to include a radius input parameter ... I default radius the same way I default latitude and longitude: to NaN, or rather, "Not A Number":

```
58  
59  
60  
61     private static string getBaseUrl(string flickrApiKey,  
62                                         string topic,  
63                                         double latitude = double.NaN,  
64                                         double longitude = double.NaN,  
65                                         double radius = double.NaN) ←  
66     {
```

I'll continue to modify the getBaseUrl() method by splitting out the line of code where I build the url that will later be used in the String.Format(). I want to test the topic, latitude, longitude and radius before adding them to the baseUrl returned from this method. In these cases, I want to only add values to the baseUrl that are valid. Therefore:

```

75     string[] licenses = { "4", "5", "6", "7" };
76     string license = String.Join(",", licenses);
77     license.Replace(", ", "%2C");
78
79     // Search API
80     // http://www.flickr.com/services/api/flickr.photos.search.html
81
82     string url = "http://api.flickr.com/services/rest/" +
83         "?method=flickr.photos.search" +
84         "&license={0}" +
85         "&api_key={1}" +
86         "&format=json" +
87         "&nojsoncallback=1";
88
89     var baseUrl = string.Format(url,
90         license,
91         flickrApiKey);
92
93     if (!string.IsNullOrWhiteSpace(topic))
94         baseUrl += string.Format("&text=%22{0}%22", topic);
95
96     if (!double.IsNaN(latitude) && !double.IsNaN(longitude))
97         baseUrl += string.Format("&lat={0}&lon={1}", latitude, longitude);
98
99     if (!double.IsNaN(radius))
100        baseUrl += string.Format("&radius={0}", radius);
101
102     return baseUrl;
103 }

```

1. I re-work this entire section, removing about half of the string I'm constructing. Now that I've removed many of the variable parts (which I'll add back into the baseUrl in #2, #3, and #4, below), I can remove the variables I'll be replacing in the String.Format statement in line 89 and following.
2. I add this gated check to ensure that the user typed something, anything into the Topic textbox on the MainPage.xaml. Assuming the user did in fact type something in, we'll surround it with double quotes (%22 is the ASCII encoding for double quotes) and add it to the baseUrl.
3. I add this gated check to ensure that latitude and longitude are indeed numbers. If they are, then we'll add them in the baseUrl.
4. I add this gated check to ensure that radius is indeed a number. If it is, I'll add it to the baseUrl.

Next, I'll modify the GetFlickrImages() method in the FlickrImage.cs file:

```
15
16     public async static Task<List<FlickrImage>> GetFlickrImages(
17         string flickrApiKey,
18         string topic,
19         double latitude = double.NaN,
20         double longitude = double.NaN,
21         double radius = double.NaN
22     )
23     {
24         HttpClient client = new HttpClient();
25
26         string baseUrl = getBaseUrl(flickrApiKey, topic, latitude, longitude, radius);
27
28         string flickrResult = await client.GetStringAsync(baseUrl);
29     }
```

1. I add radius and default it to NaN
2. I include radius in the call to getBaseUrl()

Now, I'll modify the code in the SearchResults.xaml page that calls into the GetFlickrImages() method:

```

11  namespace AroundMe
12  {
13      public partial class SearchResults : PhoneApplicationPage
14      {
15          private double _latitude;
16          private double _longitude;
17          private double _radius;
18          private string _topic;
19
20          // Your API key ... REPLACE THIS WITH YOURS:
21          // http://www.flickr.com/services/api/keys/
22          private const string FlickrApiKey = "bdd563b503e3b86bd7645afbacab7288";
23
24          public SearchResults()
25          {
26              InitializeComponent();
27
28              Loaded += SearchResults_Loaded;
29          }
30
31          protected async void SearchResults_Loaded(object sender, RoutedEventArgs e)
32          {
33              var images = await FlickrImage.GetFlickrImages(
34                  FlickrApiKey,
35                  _topic,
36                  _latitude,
37                  _longitude,
38                  _radius
39                  );
40
41              DataContext = images;
42          }
43
44          protected override void OnNavigatedTo(NavigationEventArgs e)
45          {
46              base.OnNavigatedTo(e);
47
48              _latitude = Convert.ToDouble(NavigationContext.QueryString["latitude"]);
49              _longitude = Convert.ToDouble(NavigationContext.QueryString["longitude"]);
50              _radius = Convert.ToDouble(NavigationContext.QueryString["radius"]);
51              _topic = NavigationContext.QueryString["topic"];
52          }
53      }
54  }

```

Similar to what I did with topic a few moments ago:

1. I create a private variable, `_radius`, to hold the radius value
2. I include the `_radius` variable in the call to `GetFlickrImages()`
3. When the `SearchResults.xaml` page loads, I will grab the query string value "radius".

Now I need to PASS "radius" in the query string FROM MainPage.xaml TO SearchResults.xaml ... in the MainPage.xaml:



```
71
72     private void SearchClick(object sender, EventArgs e)
73     {
74         // http://msdn.microsoft.com/en-us/library/system.net.httputility.urlencode(v=VS.95)
75         string topic = HttpUtility.UrlEncode(SearchTopic.Text);
76
77         string navTo = string.Format(
78             "/SearchResults.xaml?latitude={0}&longitude={1}&topic={2}&radius={3}",
79             AroundMeMap.Center.Latitude,
80             AroundMeMap.Center.Longitude,
81             topic,
82             5); ←
83
84         NavigationService.Navigate(new Uri(navTo, UriKind.RelativeOrAbsolute));
85     }
86 }
87
```

... I merely hard-code this to 5, indicating 5 kilometers.

I test to make sure it works, and it does.

Again, I've merely hardcoded this value. Ideally, I would allow the radius to match the visible map on the MainPage.xaml. If you want to see a better implementation, take a look at Clint's original implementation.

## Recap

To recap, the big take away from this lesson was how to grab values from XAML input controls, how to program defensively to make sure we check the input values to ensure they are valid, how to URL Encode whenever you pass values (especially, end user generated input values) in query strings of any kind. Along the way, we were able to expand the usability of our app by allowing the user to specify the type of images he or she is looking for.

# Part 30: Adding a Progress Indicator

Source Code: <http://aka.ms/absbeginnerdevwp8>

When our app launches, it will need to ascertain the current GPS position and update the Map control. That could take a little and we want to give the user of our app some visual feedback that all is fine and the application is still working.

So, here's the gameplan:

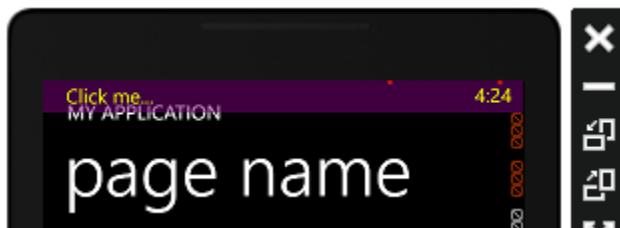
1. We'll create a new ProgressIndicator object and tell our SystemTray to use it when it needs to display a ProgressIndicator
2. I'll create a helper method that will show or hide the new ProgressIndicator

1. Understanding the Progress Indicator

It's hard to visualize exactly what the SystemTray and Progress indicator look like in an app. I used an example on this page:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626537\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626537(v=vs.105).aspx)

To help me see what it is I'm working with here:



In the screenshot (above) I was able to set the SystemTray to a transparent purple color, and set the foreground color to yellow.

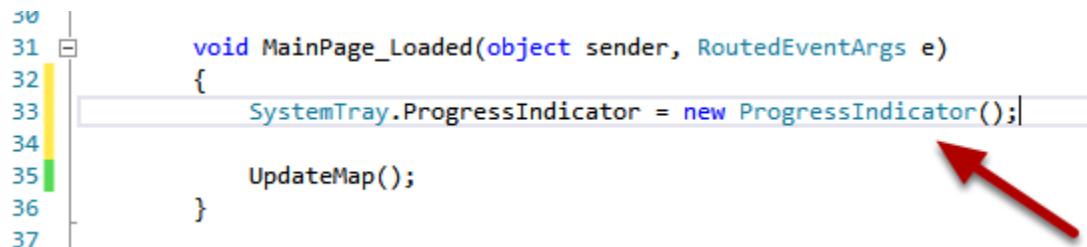
The SystemTray is ever present even if it's hidden at times depending on what we're doing with our phone. The current time is part of the SystemTray as well.

We can create a new ProgressIndicator object and give it to the SystemTray to use as the current ProgressIndicator for our app. We can then show and hide it at will.

In our app, I'll use the ProgressBar to provide feedback initially while the app is determining it's GPS position and populating the map. I'll write some logic that will hide and show the ProgressIndicator.

First step: create a new ProgressIndicator when our MainPage has loaded ...

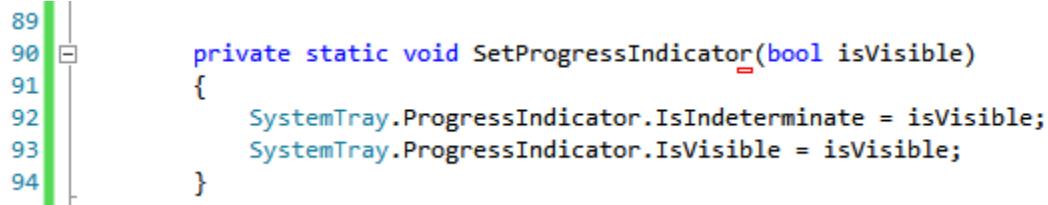
2. Create a new ProgressIndicator() object and set it as the SystemTray's current ProgressIndicator.



```
50
31     void MainPage_Loaded(object sender, RoutedEventArgs e)
32     {
33         SystemTray.ProgressIndicator = new ProgressIndicator();
34
35         UpdateMap();
36     }
37
```

Adding the ProgressIndicator was easy ... now we need to invoke it or hide it as needed.

3. Create a helper method to show / hide the ProgressIndicator



```
89
90     private static void SetProgressIndicator(bool isVisible)
91     {
92         SystemTray.ProgressIndicator.IsIndeterminate = isVisible;
93         SystemTray.ProgressIndicator.Visible = isVisible;
94     }
```

The helper method will merely take a boolean which will then turn the ProgressIndicator's animation on or off (line 92) and will show or hide the ProgressIndicator itself (line 93).

4. Modify the UpdateMap() method to use the ProgressIndicator and helper method  
Now that we have this helper method, we'll use it in our potentially long running UpdateMap() method. In fact, we'll update the user on the progress by changing the text of the ProgressIndicator throughout the various states of the UpdateMap() method:

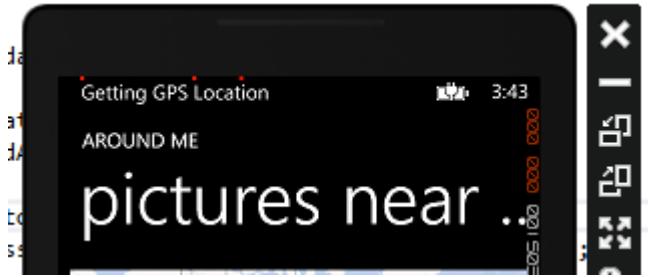
```

>/
38     private async void UpdateMap()
39     {
40         Geolocator geolocator = new Geolocator();
41         geolocator.DesiredAccuracyInMeters = 50;
42
43         SetProgressIndicator(true);
44         SystemTray.ProgressIndicator.Text = "Getting GPS Location"; 1
45
46         try
47         {
48             Geoposition position =
49                 await geolocator.GetGeopositionAsync(
50                     TimeSpan.FromMinutes(1),
51                     TimeSpan.FromSeconds(30));
52
53             SystemTray.ProgressIndicator.Text = "Acquired"; 2
54
55             var gpsCoorCenter =
56                 new GeoCoordinate(
57                     position.Coordinate.Latitude,
58                     position.Coordinate.Longitude);
59
60             AroundMeMap.Center = gpsCoorCenter;
61             AroundMeMap.ZoomLevel = 15;
62
63             SetProgressIndicator(false); 3
64         }
65         catch (UnauthorizedAccessException) {
66             MessageBox.Show("Location is disable in phone settings.");
67         }
68         catch (Exception ex) {
69             MessageBox.Show(ex.Message);
70         }
71     }
72

```

1. Line 43 calls the new helper method to show the ProgressIndicator and turn on the animation. Line 44 sets the initial text of the ProgressIndicator ... at this point, we're starting the process of getting the GPS location.
2. We update the ProgressIndicator's text to let it know we've acquired the GPS location.
3. We hide the ProgressIndicator and turn off the animation.

Let's see it in action (F5).



If your internet connection is fast, you might blink and miss the action. If the user has poor reception on their phone, it might be the only thing letting them know that the app is still functioning.

### Recap

Just to recap, the big take away from this lesson is learning what the SystemTray is, and how to pass it a ProgressBar that we can show and hide and change the text of to keep users informed about the progress of long running operations in the app.

# Part 31: Multiple Selection with the LongListMultiSelector

Source Code: <http://aka.ms/absbeginnerdevwp8>

Our ultimate vision for the app is to allow the user to select one or more photos.

Game plan:

1. We'll add a reference to the Windows Phone toolkit so that we can utilize the LongListMultiSelector, an improved version of the LongListSelector we've been using that allows a user to make multiple selections in the list.
2. We'll convert our LongListSelector to a LongListMultiSelector and add attributes to its definition to enable its unique functionality

1. Install the Windows Phone toolkit package and review the samples

The Windows Phone Toolkit extends what we get out of the box with the Windows Phone 8 API, adding new components and functionality. I'm going to download the toolkit and samples just to see what this can do for us.

Visit: <http://phone.codeplex.com/>

And click on the Downloads link.

The screenshot shows the Windows Phone Toolkit page on CodePlex. At the top, there's a navigation bar with back and forward buttons, a search bar, and a URL field showing <http://phone.codeplex.com/>. To the right of the URL is the text "The Windows Phone Toolkit". Below the navigation bar, the CodePlex logo and the text "Project Hosting for Open Source Software" are visible, along with "Register" and "Sign In" links.

The main title "The Windows Phone Toolkit" is centered above a horizontal menu bar. The menu bar contains five items: "HOME" (highlighted in blue), "SOURCE CODE", "DOWNLOADS" (which is highlighted with a red rectangle), "DOCUMENTATION", and "ISSUE TRACKER". Below the menu bar, there are two links: "Page Info" and "Change History (all pages)".

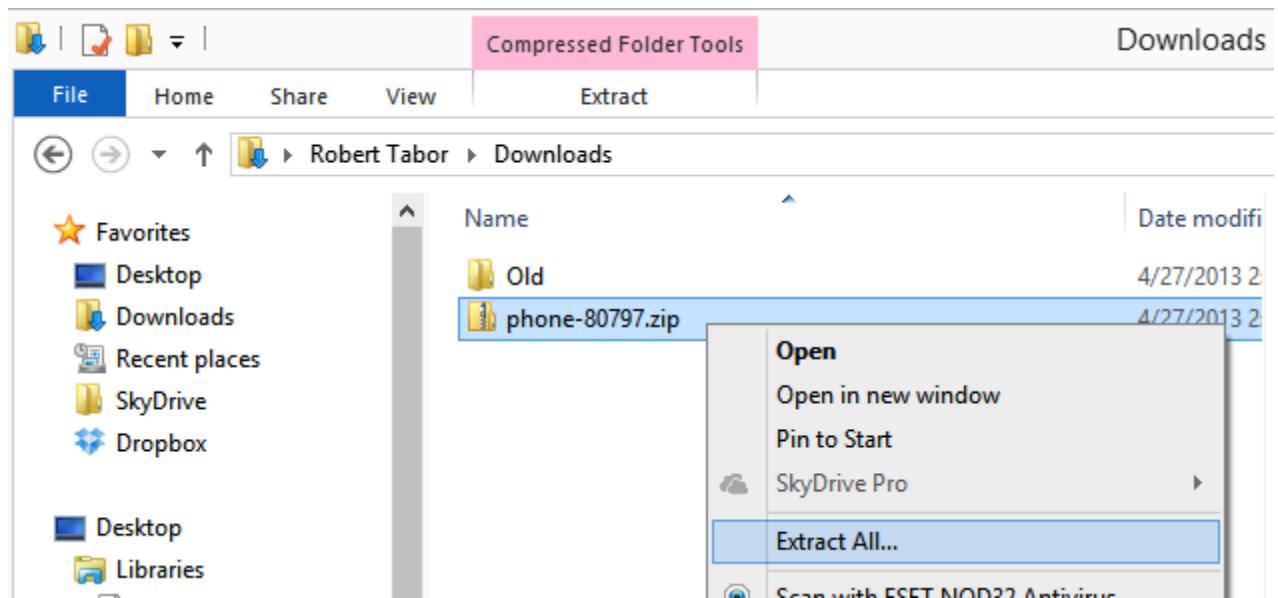
The main content area starts with a brief introduction: "Straight from Microsoft Windows Phone developer platform team - Windows Phone Toolkit provides the developer community with new components, functionality, and an efficient way to help shape product development. WPToolkit releases include open source code, samples & docs, plus design-time support for the Windows Phone platform."

A large red box highlights the "Windows Phone Toolkit" section. Inside this box, the text "Latest release date – October 30th 2012" is displayed. Below this, a list of steps for using the toolkit is provided:

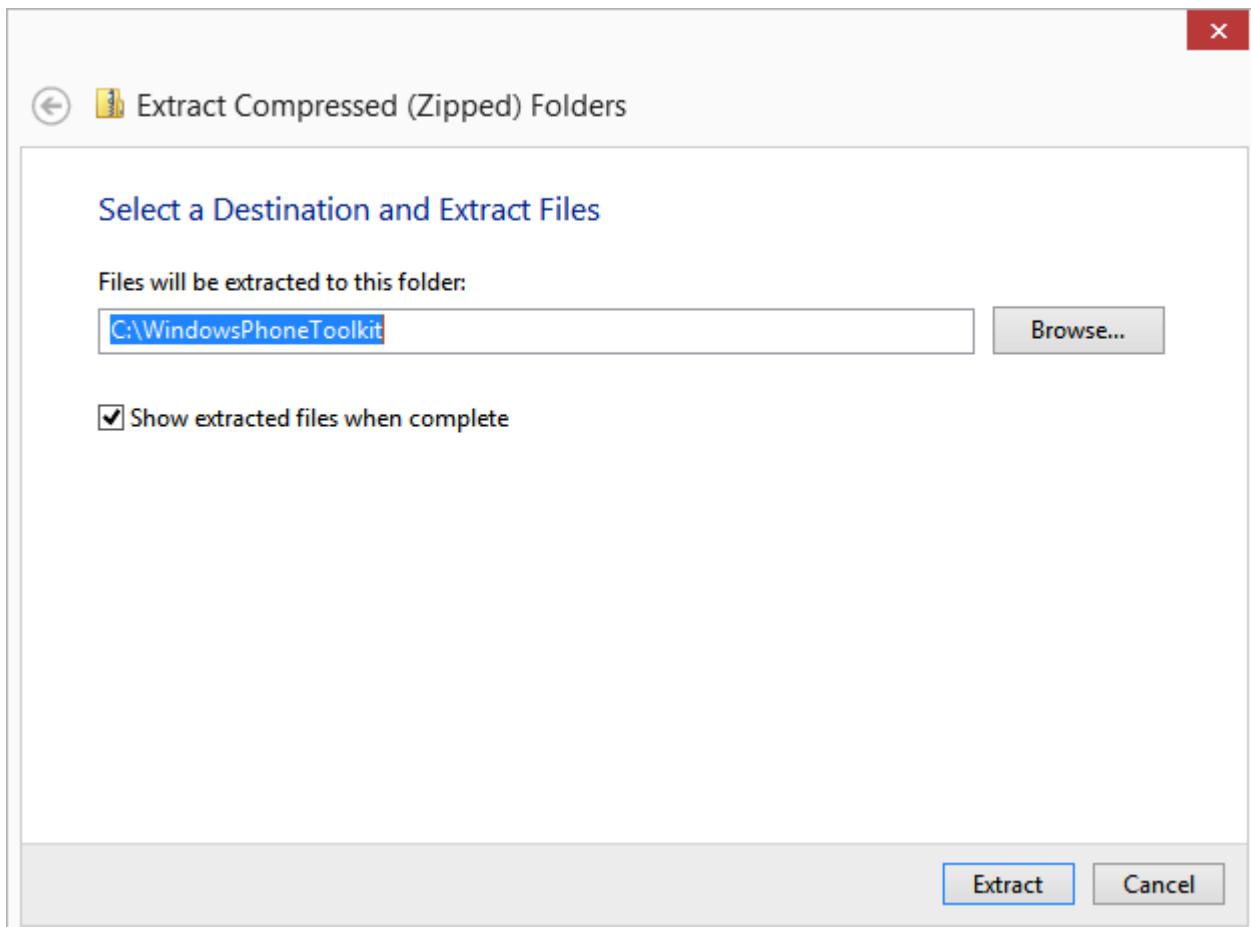
- » Ensure you have [Nuget](#) version >= 2.1 for WP8 projects.
- » [Download WPToolkit Package on NuGet](#)
- » [Download WPToolkitTestFx Package on Nuget for Test Framework](#)
- » [Download source code and samples](#)
- » [Release notes](#)
- » [Discuss in the Developing for Windows Phone forum](#)

NOTE: THIS CHANGED SINCE THE SCREENSHOT ... Download the solution from the Source Code tab!

There's a single download that contains a solution for both Windows Phone 7 and 8 toolkits, as well as sample apps containing all the controls and features. I download it, right-click the zip file and Extract All:

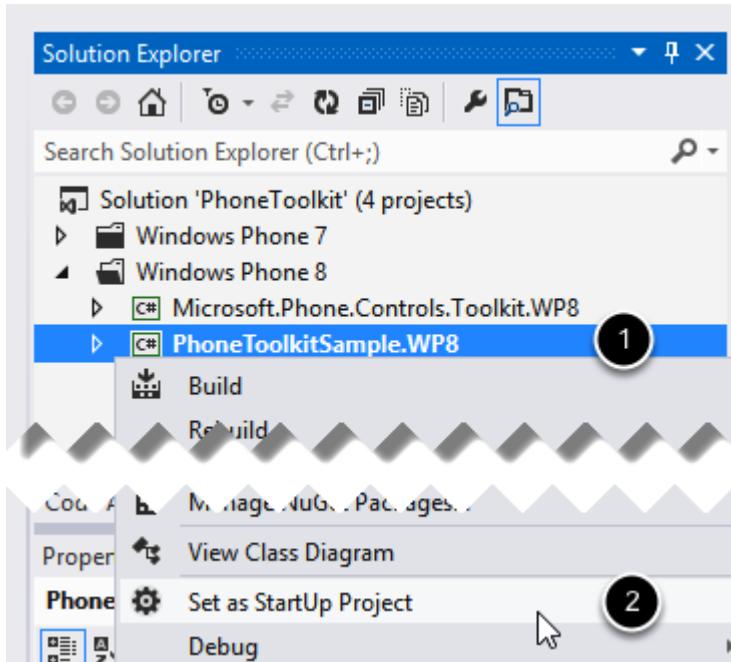


I'll put it in a temporary spot on my hard drive:



And I'll open up the Solution folder, agreeing to the dialogs that warn me about downloading things from the Internet.

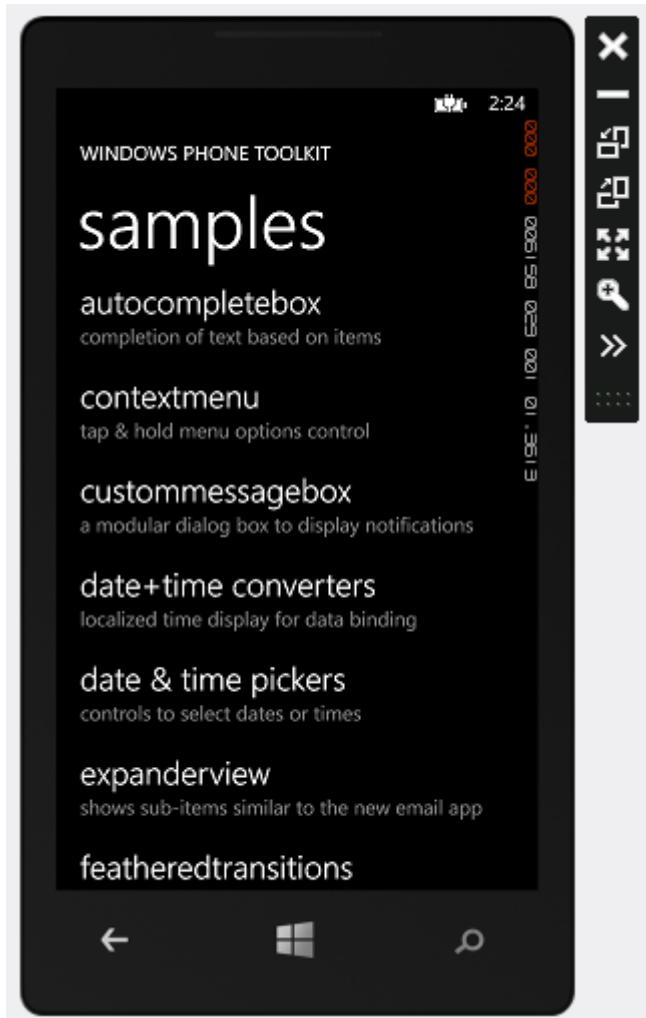
To run the Sample app:



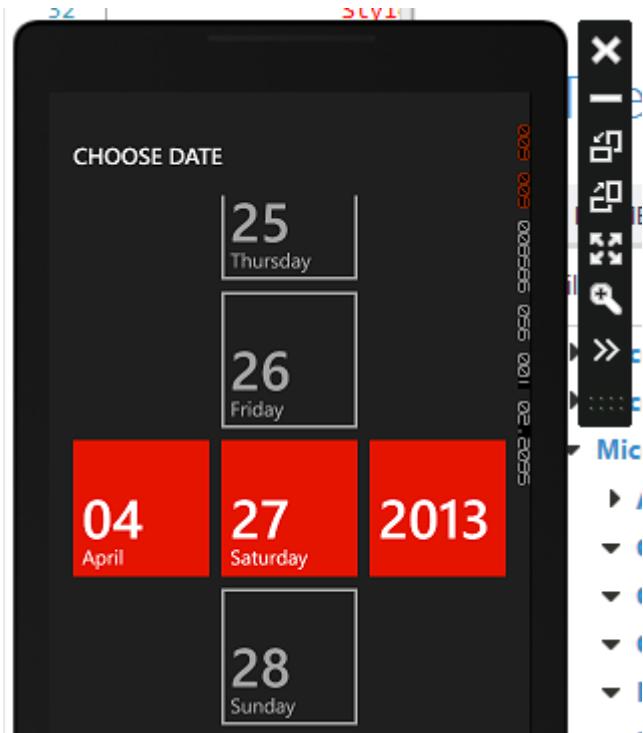
1. Right-click the project in the Solution Explorer named: PhoneToolkitSample.WP8.
2. Select "Set as StartUp Project" from the context menu.

Now, run the Solution in debug mode (F5).

There are dozens of samples featuring the controls and animations / transitions available in the toolkit:



I love the date picker:

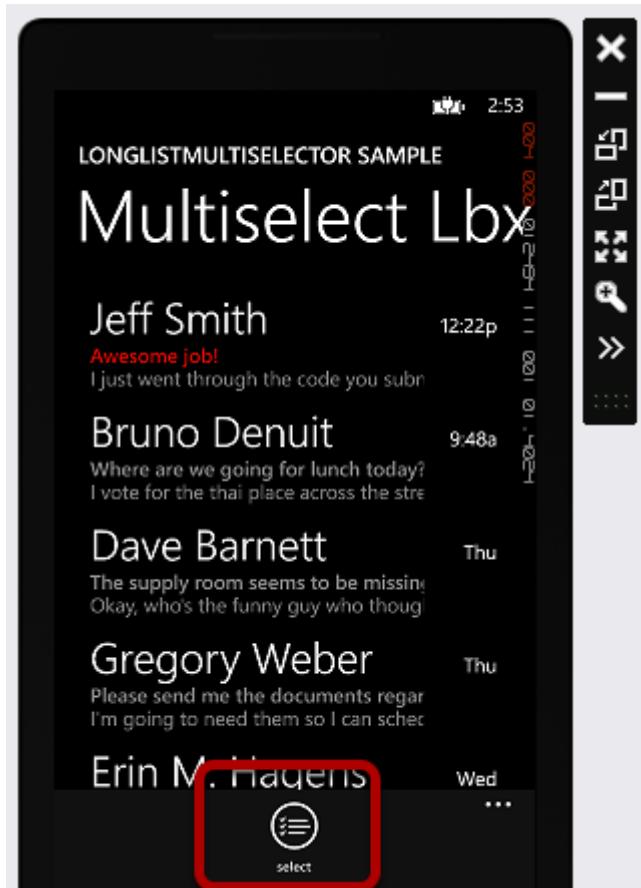


... and the HubTile control which is strangely calming if you watch it for a few minutes:

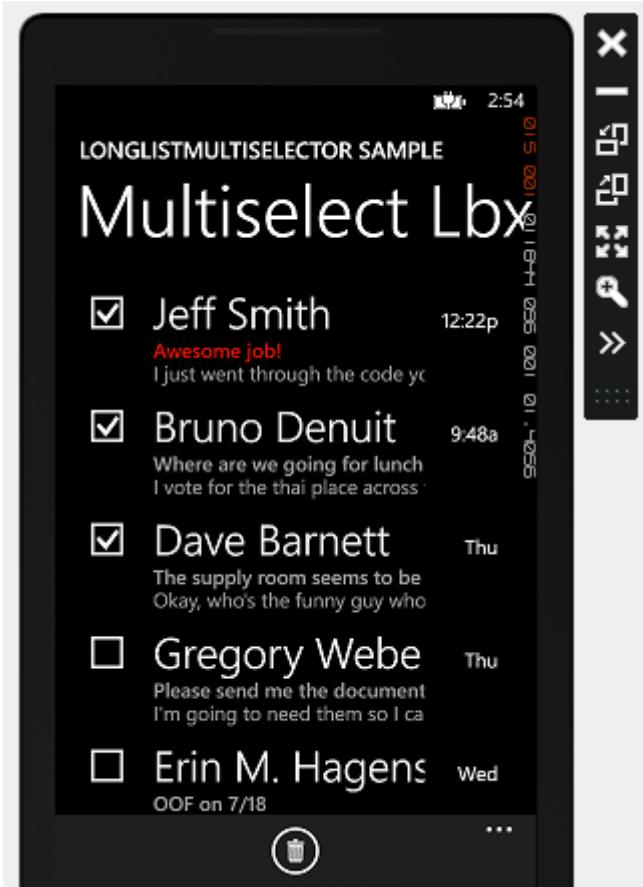


We're most interested in the LongListMultiSelector sample. It demonstrates several different modes that can be used to enable multi-selectable lists.

Click the "Select" button at the bottom:



... and that puts the list into selection mode. Now, you can select the checkmarks next to each item, then click the icon in the App Bar which has changed to a Delete button:



That's great, however not exactly what we need for our AroundMe project. Fortunately, the control's appearance and functionality can be configured in multiple ways.

If you swipe over to the Grid mode panel, there's another example that involves image thumbnails. If you click on an image, it shows a larger version in a light box. However, when you click the icon in the App Bar at the bottom, you can switch to Grid Selection Mode. This allows you to select one or more images. Each thumbnail that's selected will show a red triangle with a white checkmark icon in the upper-right-hand corner.



That's what we'll want for the results of our Flickr API call results. This will allow the user to select multiple images to be saved to the phone and randomly used for the lock screen every 30 minutes.

## 2. Install the NuGet package into the project

To use the Windows Phone Toolkit package in our existing AroundMe project, the easiest way is to add it with NuGet.

Navigate to: <https://nuget.org/packages/WPtoolkit/>

 **nuget** gallery

Log On Register

Home Packages Upload Package Statistics Documentation Blog

Search Packages 



22,003  
Downloads

22,003  
Downloads of v  
4.2012.10.30

10/30/2012  
Last update

[Project Site](#)  
[License](#)

## Windows Phone toolkit 4.2012.10.30

Windows Phone toolkit provides a collection of controls, extension methods and page animations to help create beautiful and consistent Windows Phone user interfaces and make common programming tasks easier. Documentation and source are on CodePlex at <http://phone.codeplex.com>.

To install Windows Phone toolkit, run the following command in the Package Manager Console

```
PM> Install-Package WPToolkit
```

 Tweet 4

 Like 19 people like this. Sign Up to see what your friends like.

### Release Notes

Release notes - <http://phone.codeplex.com/releases/view/96743>

Includes -

I love the obvious package installation instructions on the NuGet site.

If the Package Manager Console is not already open, go to Visual Studio's Tools menu, select Library Package Manager, then the Package Manager Console sub menu option.

At the Package Manager command prompt, type: **install-package WPToolkit**

```
Package Manager Console

Package source: NuGet official package source | Default project: AroundMe
Each package is licensed to you by its owner. Microsoft is not responsible for
additional licenses. Follow the package source (feed) URL to determine
what other terms apply.

Package Manager Console Host Version 2.2.40116.9051

Type 'get-help NuGet' to see all available NuGet commands.

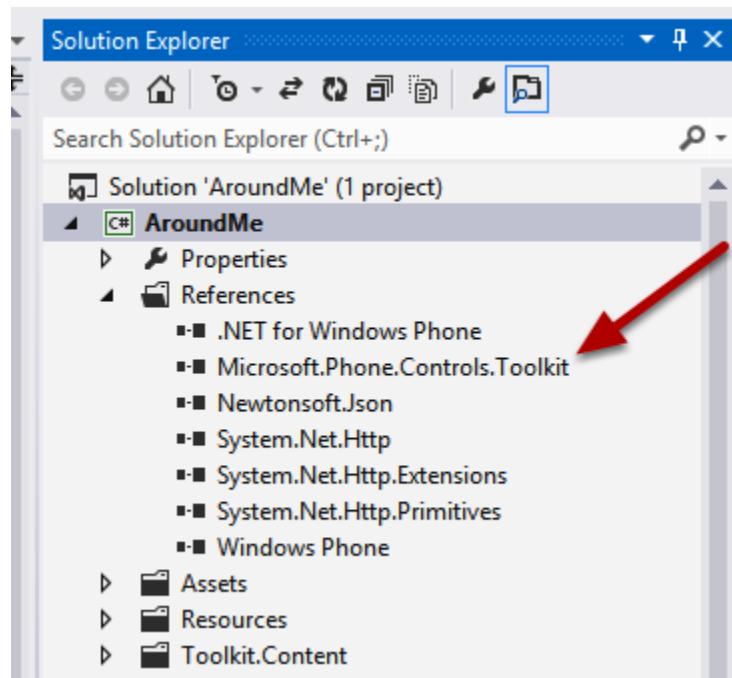
PM> install-package WPToolkit
```

Assuming you typed it correctly and it encountered no problems, you should see this:

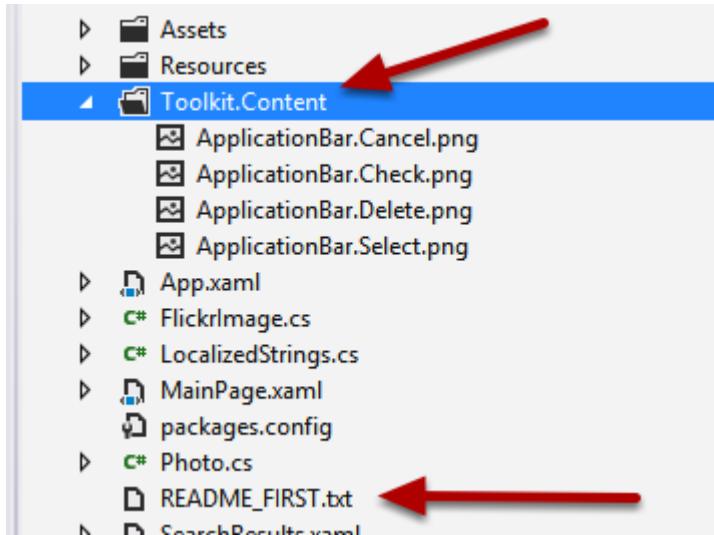
```
PM> install-package WPtoolkit
Successfully installed 'WPtoolkit 4.2012.10.30'.
Successfully added 'WPtoolkit 4.2012.10.30' to AroundMe.

PM>
```

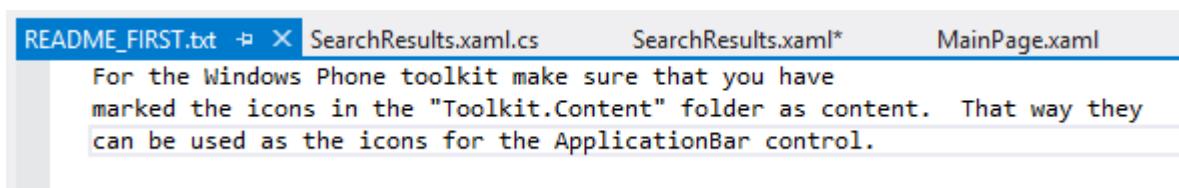
... and you should now see a reference to the Microsoft.Phone.Controls.Toolkit in the References section of the Solution Explorer for the AroundMe project:



This NuGet package included some additional content ... a folder called Toolkit.Content with a few icon files, and a README\_FIRST.txt file:



The README\_FIRST.txt file explains the purpose of the icons:



### 3. Implement the LongListMultiSelector

Ok, so now we're ready to modify our SearchResults.xaml page. We'll be upgrading the LongListSelector control to the LongListMultiSelector control, changing a few attributes to enable the enhanced features we'll utilize:

```

53
54      <!--ContentPanel - place additional content here-->
55      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
56          <!-- <TextBlock Name="LocationTextBlock" /> -->
57          <phone:LongListSelector
58              LayoutMode="Grid"
59              ItemsSource="{Binding}"
60              GridCellSize="105, 105">
61              <phone:LongListSelector.ItemTemplate>
62                  <DataTemplate>
63                      <Image
64                          Source="{Binding Image320}"
65                          Stretch="UniformToFill" />
66                  </DataTemplate>
67              </phone:LongListSelector.ItemTemplate>
68          </phone:LongListSelector>
69      </Grid>
70  </Grid>

```

Before we can use it, we'll need to create an XML Namespace entry in the PhoneApplicationPage element at the very top of the page. See line 9:

```

1  <phone:PhoneApplicationPage
2      x:Class="AroundMe.SearchResults"
3      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
6      xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
7      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
9      xmlns:toolkit="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit"
10     FontFamily="{StaticResource PhoneFontFamilyNormal}"
11     FontSize="{StaticResource PhoneFontSizeNormal}"
```



This XAML Namespace basically says this ... when you see an XAML element prefixed with toolkit, then look in the following assembly and the following CLR namespace for that class' definition. Somewhere in the Windows Phone Toolkit code, there's a namespace defined as Microsoft.Phone.Controls.Toolkit, and it has a class named LongListMultiSelector, among others. So, the compiler parses the element <toolkit:LongListMultiSelector />, and it knows to create an instance of that specific class.

In a nutshell, that's all there is to it, but you can learn even more about XAML Namespaces and mapping to custom classes and assemblies

here: [http://msdn.microsoft.com/en-us/library/ms747086.aspx#Mapping\\_To\\_Custom\\_Classes\\_and\\_Assemblies](http://msdn.microsoft.com/en-us/library/ms747086.aspx#Mapping_To_Custom_Classes_and_Assemblies)

Now that we have the XAML Namespace defined, we'll be able to reference the LongListMultiSelector like so:

The diagram shows the XAML code structure with three numbered callouts:

- Callout 1: Points to the opening tag of the LongListMultiSelector element (line 38).

```
<toolkit:LongListMultiSelector
```
- Callout 2: Points to the EnforceIsSelectionEnabled="True" attribute (line 39).

```
    LayoutMode="Grid"  
    ItemsSource="{Binding}"  
    GridCellSize="105, 105"  
    EnforceIsSelectionEnabled="True"
```
- Callout 3: Points to the toolkit:LongListMultiSelector.ItemTemplate element (line 44).

```
    SelectionChanged="PhotosForLockscreen_SelectionChanged">  
    <toolkit:LongListMultiSelector.ItemTemplate>
```

```
54  
55      <!--ContentPanel - place additional content here-->  
56  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">  
57      <!-- <TextBlock Name="LocationTextBlock" /> -->  
58      <toolkit:LongListMultiSelector  
59          LayoutMode="Grid"  
60          ItemsSource="{Binding}"  
61          GridCellSize="105, 105"  
62          EnforceIsSelectionEnabled="True" 1  
63          SelectionChanged="PhotosForLockscreen_SelectionChanged">  
64          <toolkit:LongListMultiSelector.ItemTemplate> 2  
65              <DataTemplate>  
66                  <Image  
67                      Source="{Binding Image320}"  
68                      Stretch="UniformToFill" />  
69              </DataTemplate>  
70          </toolkit:LongListMultiSelector.ItemTemplate>  
71      </toolkit:LongListMultiSelector>  
72  </Grid>  
73  
74
```

We can salvage most of the code, however we'll ...

1. Change from <phone:LongListSelector to <toolkit:LongListMultiSelector as well as the closing element (line 51).
2. Add the EnforceIsSelectionEnabled="True" attribute and setting. This property is important: it's what actually allows for the multi-selection behavior.
3. Change from <phone:LongListSelector.ItemTemplate to <toolkit:LongListMultiSelector.ItemTemplate as well as the closing element (line 50).

Finally, we'll test to make sure we can actually multi-select:



Success!

### Recap

Just to recap, the big take away from this lesson is the Windows Phone toolkit ... what it is, what additional features it supplies, how to incorporate it into our app through NuGet and adding the XAML namespace, and so on. In most cases, all we have to do is think about the features we want to support and focus on those ... most of the heavy lifting has been done by someone else. Nonetheless, we need to be aware that these resources exist and know how to add them into our project.

# Part 32: Animating Image Search Results

Source Code: <http://aka.ms/absbeginnerdevwp8>

In this lesson we'll refine our search results page by animating the images as they are loaded from Flickr. We want the images to ease in, fade in to view. We also want to provide feedback to the user of the app for long running operations like retrieving these photos over a slow internet connection by adding a progress indicator. And finally, I want to add some feedback in case the user searches their current location and topics and Flickr has no results to return ... we don't want to leave the user in the dark waiting for photos to appear that never come.

So, our game plan in this lesson:

1. We'll modify our LongListMultiSelector's DataTemplate -- we'll prepare the Image control for the fade in effect we'll implement
2. We'll create an animation and storyboard targeting the image control to enable the fade in effect
3. We'll make provisions for the possibility that there are no photos on Flickr that match our search criteria by creating and hiding a Text block that we'll un-hide when there are no photos to show
4. We'll add a progress indicator that will run while we're performing the web service call to Flickr and loading the images

1. Prepare the Image control for a fade-in animation

The first step we'll take is to set the initial Opacity attribute of the Image to 0, indicating that we want to hide the contents of the image. 0 is completely transparent and 1 is completely opaque. 0.5 would be partially transparent / opaque.

Then, for each image in our DataTemplate, as that particular image has downloaded from Flickr, the ImageOpened event will fire. We'll handle that event and write code to animate the change of the Opacity attribute.

So, we'll make the following changes to the Image control:

```

38     <toolkit:LongListMultiSelector
39         LayoutMode="Grid"
40         ItemsSource="{Binding}"
41         GridCellSize="105, 105"
42         EnforceIsSelectionEnabled="True"
43         SelectionChanged="PhotosForLockscreen_SelectionChanged">
44             <toolkit:LongListMultiSelector.ItemTemplate>
45                 <DataTemplate>
46                     <Image
47                         Opacity="0"
48                         Source="{Binding Image320}"
49                         Stretch="UniformToFill"
50                         ImageOpened="Image_ImageOpened" />
51                 </DataTemplate>
52             </toolkit:LongListMultiSelector.ItemTemplate>
53         </toolkit:LongListMultiSelector>

```

1. Add an Opacity attribute, set it to 0
2. Add an ImageOpened event handler attribute and wire it up to a new method called "Image\_ImageOpenend", which we'll implement in the next step.

Right-click anywhere on that line of code with the ImageOpened attribute or its setting, select "Navigate to Event Handler" from the context menu.

## 2. Implement the Image\_ImageOpened Event Handler

Here we'll write C# code to animate the fade in effect. Many examples of animation you'll see use XAML to define both the Storyboard and Animations associated with the storyboard, then use C# code to kick off the animation when a certain event is triggered. We could do it that way, but we've chosen to do it all with C# because we're dealing with a special situation (more about that in a moment).

Animations are made up of a Storyboard object and at least one Animation object.

Let's start with the Animation. First, there are several different types of animation data types. We're using a DoubleAnimation data type because we want to move a property from 0.0 to 1.0. There's also a ColorAnimation class for animating between two colors, and a PointAnimation for modifying an objects X Y coordinate or its size.

An Animation is basically a timeline combined with a result. The results are determined by the specific Animation class you pick like we just talked about. It's important to know that all Animation classes inherit from a Timeline class which confers properties related to timing of the animation ... the Begin time, allowing you to delay the start of animation, perhaps waiting for other animations on the storyboard to begin or complete, a Duration property that effects how long the animation should take before delivering the desired

result -- how long should it take for our fade in effect, in this case. There's also AutoReverse and RepeatBehavior properties which do what they suggest.

A Storyboard is a collection of one or more Animations. You group the Animations you want triggered by a specific event, like a button click, a loaded event and so on. The Storyboard allows you to pair an Animation with a target object. The animation is just a definition of what property should be affected, when it should be affected and how long. We have to APPLY that Animation to a target object. In our case, that target object will be an Image control.

In our case, our Storyboard is simple ... we just want one thing to happen. We could add multiple Animations targeting multiple properties of multiple objects, and could even create child Storyboards to better refine how our user's experience will play out. Once we're ready to allow the Storyboard to play, we call its Begin() method. It will in turn kick off all its child Animations and child Storyboards.

For a more complete explanation of animations, I'd recommend you start here:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206955\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206955(v=vs.105).aspx)

Back to our situation ... we need to animate each image as it's opened. We'll create an Animation to move the Opacity property from 0 to 1 thereby making it transform from completely transparent into completely opaque. We'll add the Animation to a Storyboard and then call Begin() on the Storyboard.

```
--  
60     private void Image_ImageOpened(object sender, RoutedEventArgs e)  
61     {  
62         Image img = sender as Image;  
63  
64         if (img == null)  
65             return;  
66  
67         // using System.Windows.Media.Animation  
68         Storyboard s = new Storyboard();  
69  
70         DoubleAnimation doubleAni = new DoubleAnimation();  
71         doubleAni.To = 1;  
72         doubleAni.Duration = new Duration(TimeSpan.FromMilliseconds(500));  
73  
74         Storyboard.SetTarget(doubleAni, img);  
75         Storyboard.SetTargetProperty(doubleAni, new PropertyPath(OpacityProperty));  
76  
77         s.Children.Add(doubleAni);  
78  
79         s.Begin();  
80     }
```

Sometimes the animation is subtle to the untrained eye. To make the effect more dramatic, you can stretch the time from 500 milliseconds to something like 2000 or 3000 milliseconds, or rather, 2 or 3 seconds just for testing purposes.

Lines 74 and 75 use a SetTarget and SetTargetProperty syntax to pair up the animation and its target. You see a lot of Set\_\_ and Get\_\_ styled methods in the Windows Phone API, and that's due to the Windows Phone property system we talked about at the very outset of this series.

The Windows Phone property system allows us to use attached properties. If lines 68 through 75 were in XAML, it might look something like this:

```
<Storyboard>
<DoubleAnimation
Storyboard.TargetName="ImageControlName"
Storyboard.TargetProperty="Opacity"
To="1.0" Duration="0:0:0.5" />
</Storyboard>
```

(Now, this approach is actually flawed because we would be targeting a single Image control. And as far as I know, you can't use a binding expression inside of the Storyboard.TargetName to make it dynamically apply to every image control in our data template, so this is one reason why a C# approach works better.)

I wanted to show the XAML version (flawed though it is) to illustrate what we're doing in our C# code ... in lines 74 and 75 we're ATTACHING the Storyboard.SetTarget and Storyboard.SetTargetProperty attached properties TO the DoubleAnimation object doubleAni, and setting their values appropriately. It looks like we're setting attributes of the Storyboard, but we're ATTACHING attached properties TO the DoubleAnimation.

One more curiosity about line 75. We're setting the Storyboard.TargetProperty to a new PropertyPath(OpacityProperty) ... what does this mean?

The rules of the Windows Phone property system require the target property of an animation must be set to a Dependency Property. Remember: that's one of the special powers of a Dependency Property -- that it can be animated, unlike regular old CLR properties. Fine, then why do we have to wrap it with the new PropertyPath object. That's difficult for me to explain, and for the sake of brevity let me point you to another article that can explain it with a good example:

### **PropertyPath XAML Syntax**

Specifically, this link points to the anchor "PropertyPath for Animation Targets"  
[http://msdn.microsoft.com/en-us/library/ms742451.aspx#databinding\\_sa](http://msdn.microsoft.com/en-us/library/ms742451.aspx#databinding_sa)

In a nutshell, the PropertyPath is used to specify the property that is the target of an animation. We have a simple case, and so for now, I think it's easier just to understand

that the `PropertyPath` object provides a map to find the dependency property we want to animate.

Let's make sure our images fade in by debugging the app:



Mine works, and did you notice the half-second fade in? Very nice.

We've satisfied the topic of this lesson, but I wanted to add a few more features to our app while we're here. We'll provide feedback to the user to let them know the status of their searches against the Flickr API.

3. Add a `TextBlock` for "No Photos Found", a `ProgressBar` and `TextBlock` for "Loading"  
Now, lets edit the `MainPage.xaml` again, sandwiching the `LongListMultiSelector` with two new passages of XAML:

```

34
35     <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
36
37         1 <TextBlock
38             x:Name="NoPhotosFound"
39             Visibility="Collapsed"
40             Style="{StaticResource PhoneTextTitle2Style}"
41             Text="No photos found :(" />
42
43         <toolkit:LongListMultiSelector
44             Name="PhotosForLockscreen"
45             LayoutMode="Grid"
46             ItemsSource="{Binding}"
47             GridCellSize="105, 105"
48             EnforceIsSelectionEnabled="True"
49             SelectionChanged="PhotosForLockscreen_SelectionChanged">
50             <toolkit:LongListMultiSelector.ItemTemplate>
51                 <DataTemplate>
52                     <Image
53                         Opacity="0"
54                         Source="{Binding Image320}"
55                         Stretch="UniformToFill"
56                         ImageOpened="Image_ImageOpened" />
57                 </DataTemplate>
58             </toolkit:LongListMultiSelector.ItemTemplate>
59         </toolkit:LongListMultiSelector>
60
61         2 <StackPanel VerticalAlignment="Center" x:Name="Overlay" Visibility="Collapsed">
62             <TextBlock HorizontalAlignment="Center" Text="Loading ..." />
63             <ProgressBar x:Name="OverlayProgressBar" IsIndeterminate="True" />
64         </StackPanel>
65
66     </Grid>

```

1. We create a TextBlock element. A TextBlock is a simple XAML element for displaying small amounts of flow content. By "flow content" I mean textual content that dynamically adjusts and reflows the text content based on run-time variables such as window size, device resolution and other user preferences. There's quite a bit to Flow Documents in XAML ... I would point you to this resource for more information:  
<http://msdn.microsoft.com/en-us/library/aa970909.aspx>

At any rate, the TextBlock is just a way to put some text on our page.

We style it using a built-in Text Style: [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769552\(v=vs.105\).aspx#BKMK\\_TextStyles](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769552(v=vs.105).aspx#BKMK_TextStyles) and most importantly: we set its Visibility attribute to Collapsed meaning we want it hidden by default. We'll write some logic that will potentially change that in a moment.

2. Add a StackPanel that contains another TextBlock. It also has a ProgressBar element with an IsIndeterminate="True" set, meaning we just want it to keep animating until we're finished with it. We'll toggle this property on and off as we need it. Most importantly, we set the Visibility of the StackPanel to "Collapsed" ... here again, we'll write some logic that will toggle the Visibility as we're performing the web service call in the background.

Next, in the `SearchResults_Loaded` event handler, we'll write the logic for both code passages (above) that I just alluded to.

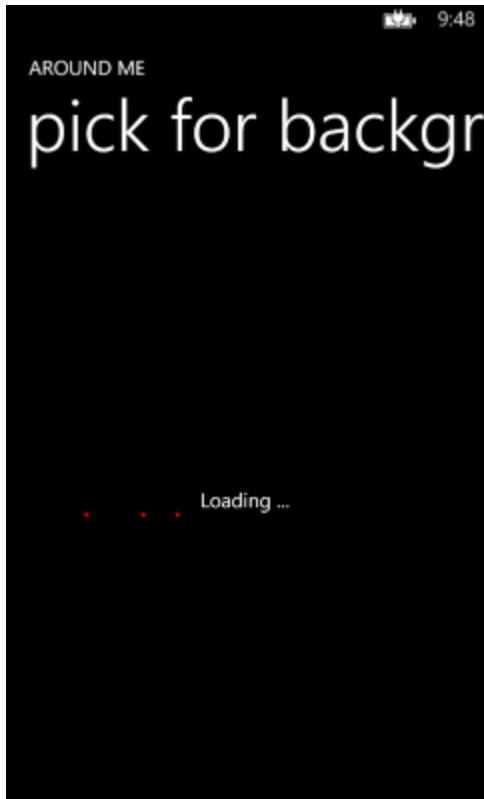
```
32     protected async void SearchResults_Loaded(object sender, RoutedEventArgs e)
33     {
34         Overlay.Visibility = Visibility.Visible;
35         OverlayProgressBar.IsIndeterminate = true; 1
36
37         var images = await FlickrImage.GetFlickrImages(
38             FlickrApiKey,
39             _topic,
40             _latitude,
41             _longitude,
42             _radius
43         );
44
45         DataContext = images;
46
47         if (images.Count == 0)
48             NoPhotosFound.Visibility = Visibility.Visible;
49         else
50             NoPhotosFound.Visibility = Visibility.Collapsed;
51
52         Overlay.Visibility = Visibility.Collapsed; 2
53         OverlayProgressBar.IsIndeterminate = false;
54     }
```

We added code before and after our web services call to Flickr to show and hide the StackPanel's contents, as well as toggle the ProgressBar's animation.

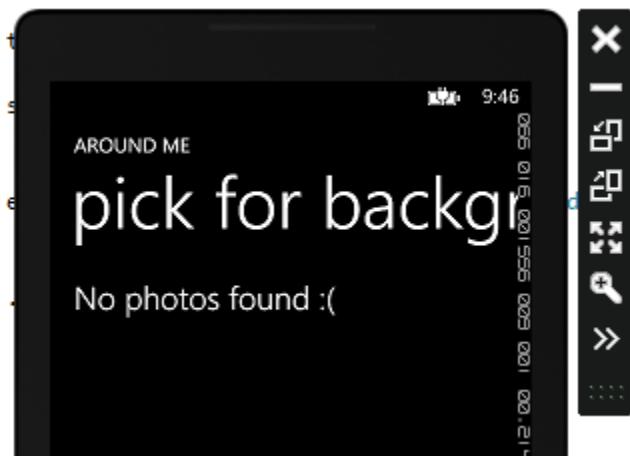
1. Here we show the StackPanel we added (named Overlay). Also, we turn on the ProgressBar's animation by setting the `IsIndeterminate = true`.
2. In lines 47 through 50, we decide to toggle the Visibility of the TextBlock to reveal the run stating "No Photos found :" in, in fact, no photos were returned from the Flickr web service call.

In lines 52 and 53, we reverse what we did in lines 34 and 35 ... we toggle the Visibility back to Collapsed to hide it, then set `IsIndeterminate = false` to stop the animation.

I'll test both scenarios. First, I'll search for some non-sensical term:



Since it takes a moment to run the query, depending on the speed of your internet connection, you should briefly see the "Loading ..." text and the animated progress bar. Ultimately, we want to see the results:



And it works!

## Recap

Just to recap, the big take away from this lesson is animation in the Windows Phone API. We create an Animation object to control the change of a dependency property over time, and pair that Animation object with a dependency object and add that pairing to a Storyboard. These little touches, almost imperceptible, make our apps stand out from the competition and earn the respect of our users.

We also added feedback to the user about the state of the app while it's making that long call to the Flickr API with a Progress Bar and a message, and let the user know when there were no Flickr images that matched their criteria. Again, we're trying to consider this from a user's perspective ... I've used plenty of apps where I wasn't sure if the app was still working or it had locked up. These little touches help the user gain confidence in the app and enhance their emotional ties to it.

# Part 33: Working with the Lock Screen to Display an Image

Source Code: <http://aka.ms/absbeginnerdevwp8>

Our app is coming together nicely, but there's one last feature we want to implement. When the user selects one or more Flickr images in the LongListMultiSelector, we'll take the FlickrImage data and serialize it to disk using JSON. In this first pass, we'll randomly select one of those FlickrImage instances as the lock screen image once. Then in the next lesson, we'll randomly select from that list of serialized FlickrImages once every 30 minutes, which is ultimately our goal.

So, our game plan in this lesson:

1. Add an App Bar to the SearchResults.xaml page
2. When at least one item is selected from our LongListMultiSelector, we'll show our app bar, otherwise we'll hide it
3. We'll handle the click event for the new Set App Bar button, getting a list of selected Flickr images
4. We'll add a new LockScreenHelpers.cs class and add functionality to: (a) clean out any images that were already in local storage, (b) save the new list of images into local storage, and (c) select a random image and make it the phone's lock screen image.

This is a very code-intensive lesson. For the most part, I'll provide an overview of several lines of code at a time. In some cases, I'll dive deeper to explain an individual class or method. However, if I ever skip something or if I don't explain it to your satisfaction, Microsoft's documentation is always the definitive source you should be referencing. You can't become a serious developer without consulting MSDN several times a day.

## 1. Add an App Bar

We've done this several times already so I won't explain it in depth. I add the following method:

```
59     private void BuildLocalizedApplicationBar()
60     {
61         // Set the page's ApplicationBar to a new instance of ApplicationBar.
62         ApplicationBar = new ApplicationBar();
63         ApplicationBar.IsVisible = false;
64
65         // Create a new button and set the text value to the
66         //localized string from AppResources.
67         AppBarIconButton appBarButton =
68             new AppBarIconButton(
69                 new Uri("/Toolkit.Content/ApplicationBar.Check.png",
70                     UriKind.RelativeOrAbsolute));
71
72         appBarButton.Text = AppResources.AppBarSet;
73         appBarButton.Click += appBarButton_Click;
74
75         ApplicationBar.Buttons.Add(appBarButton);
76     }
77
78     private void appBarButton_Click(object sender, EventArgs e)
79     {
80         throw new NotImplementedException();
81     }
82 }
```

In this code we simply create a new ApplicationBar() object (line 62) and create a single AppBarIconButton (line 67) with a checkmark icon. On that button, we'll set the text (line 72) and wire up an event handler (line 73). Then we'll add the button to the ApplicationBar (line 75). Notice in line 63 that we set the IsVisible to false initially. We'll write display logic -- when one or more Flickr images are selected in our LongListMultiSelector, we'll make the ApplicationBar visible.

Since we're creating a LOCALIZED ApplicationBar, and using AppResources in line 72 to set the button's text, we'll need to modify the AppResources.resx file located in the Resources folder of our project.

The screenshot shows the 'AppResources.resx' file in the Visual Studio resources editor. The table contains the following entries:

	Name	Value
AppBarMenuItemText		Menu Item
AppBarSearch		Search
ApplicationTitle		MY APPLICATION
ResourceFlowDirection		LeftToRight
ResourceLanguage		en-US
▶	AppBarSet	Set
*		

I add an AppBarSet entry and set it to the word "Set" ... this will be the text under our checkmark icon.

Finally, we'll need to trigger our BuildLocalizedApplicationBar() method in the SearchResults.xaml.cs constructor like so:

```

26     public SearchResults()
27     {
28         InitializeComponent();
29
30         Loaded += SearchResults_Loaded;
31
32         BuildLocalizedApplicationBar(); ←
33     }

```

Now that we've created the Application Bar, we need to write the display logic to show it only when one or more Flickr images are selected.

2. Write display logic in the SelectionChanged event handler of the LongListMultiSelector  
When I first created the LongListSelector (prior to it becoming the LongListMultiSelector), I added a SelectionChanged event handler. I knew I would need it eventually. Now is that time.

When the list of SELECTED items changes, we want to evaluate the number of items selected and show or hide the Application Bar accordingly.

First, we need to add a Name attribute to the LongListMultiSelector so we can reference it programmatically in C#:

```
42 </TextBlock>
43 <toolkit:LongListMultiSelector
44     Name="PhotosForLockscreen"
45     LayoutMode="Grid"
46     ItemsSource="{Binding}"
47     GridCellSize="105, 105"
48     EnforceIsSelectionEnabled="True"
49     SelectionChanged="PhotosForLockscreen_SelectionChanged">
```



Next, right-click line 49 (above):

```
SelectionChanged="PhotosForLockscreen_SelectionChanged" >
```

and select Navigate to Event Handler from the context menu. That will open the SearchResults.xaml.cs file again and allow us to add code to the PhotosForLockscreen\_SelectionChanged event handler. We'll merely add a simple if else statement like so:

```
-->
94     private void PhotosForLockscreen_SelectionChanged(
95         object sender,
96         SelectionChangedEventArgs e)
97     {
98         if (PhotosForLockscreen.SelectedItems.Count == 0)
99             ApplicationBar.IsVisible = false;
100        else
101            ApplicationBar.IsVisible = true;
102    }
103
```

Here we're checking the number of selected items in the PhotosForLockscreen LongListMultiSelector object and setting the visibility of the Application Bar accordingly.

If we were to test our code, we should be able to select one item and see the Application Bar with the checkmark icon appear:



... and if we were to deselect that item, the Application Bar should disappear.

3. Write code to handle the Set Application Bar Button's Click event

When I wrote this line of code earlier in the BuildLocalizedApplicationBar() method:

```
appBarButton.Click += appBarButton_Click;
```

... Visual Studio created a stubbed out method called appBarButton\_Click(). This is where we'll kick off much of the hard work in this application around saving the selected images and picking a random one to display on the lock screen, at least, initially.

In the appBarButton\_Click() method, I add the following code:

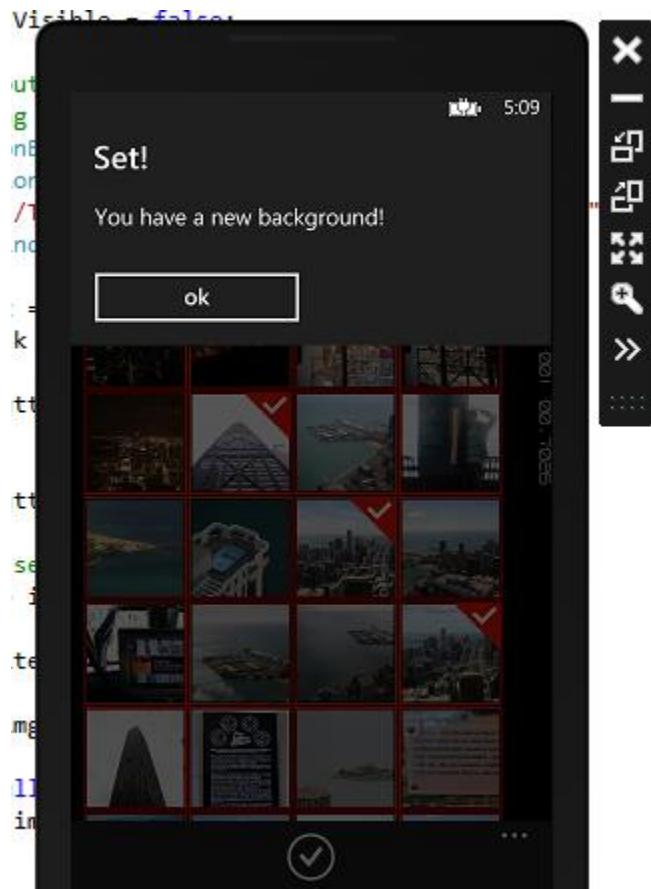
```
78     private void appBarButton_Click(object sender, EventArgs e)
79     {
80         // Get a list of selected images
81         List<FlickrImage> imgs = new List<FlickrImage>();
82
83         foreach (object item in PhotosForLockscreen.SelectedItems)
84         {
85             FlickrImage img = item as FlickrImage;
86
87             if (img != null)
88                 imgs.Add(img);
89         }
90
91         // Clean out / remove all images currently in IsolatedStorage
92         // Save this new list of selected images to IsolatedStorage
93         // Randomly select one item and set it as the lockscreen
94
95         MessageBox.Show("You have a new background!", "Set!", MessageBoxButtons.OK);
96     }

```

After create a new generic List<FlickrImage>, I iterate through each selected item from the LongListMultiSelector and adding each one to the new List<FlickrImage> collection.

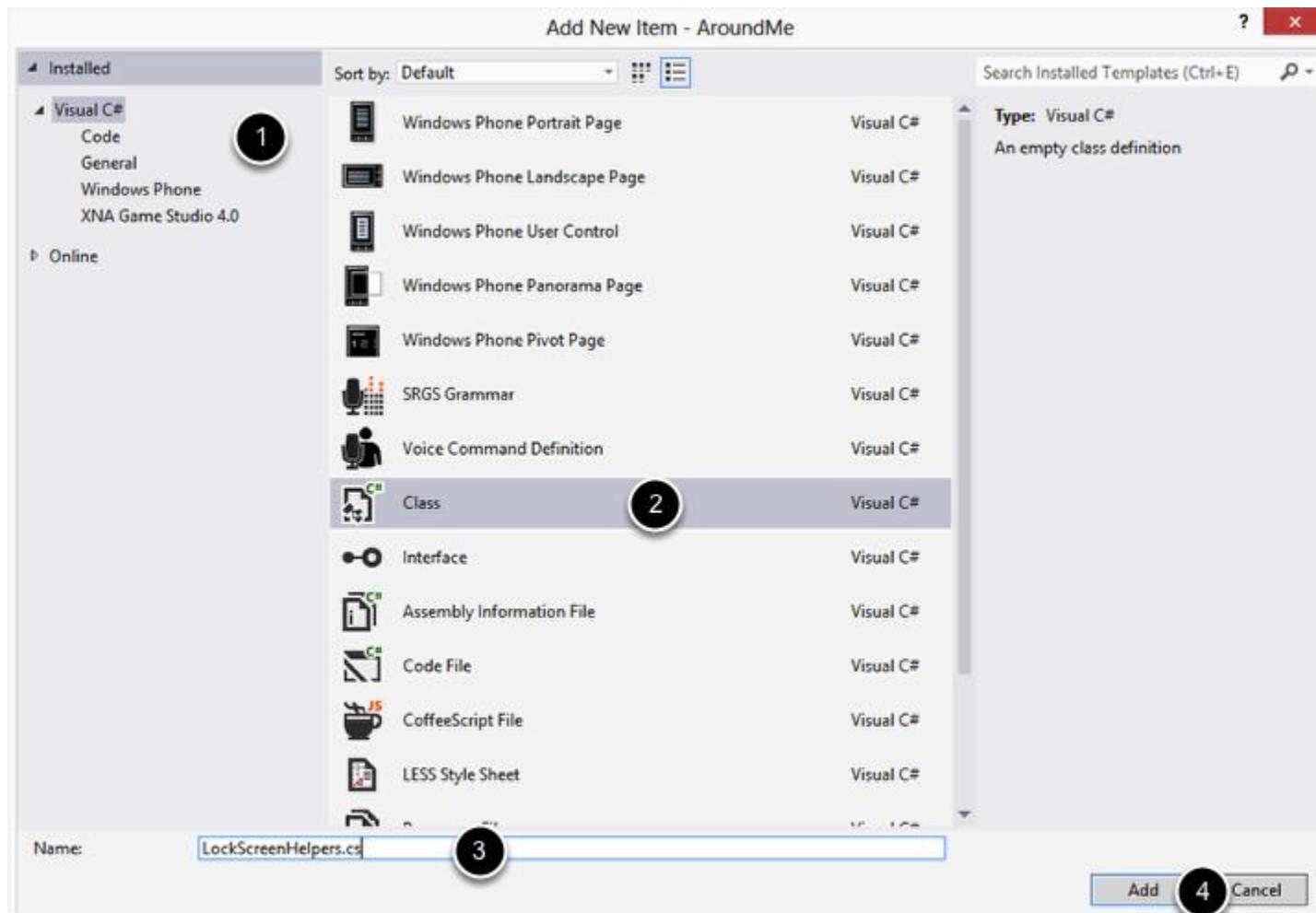
In lines 91 through 93 I write a few comments ... these are TODOs ... I have more work to do prior to completing these steps. However, this list will drive the development of the three major features I'll implement in the remainder of this lesson. I'll implement them in a new helper class I'll create in a moment. I use the term "helper class" to talk about a class of related utility methods intended to interface with some part of the underlying system. Some may call this a facade ... it's a friendly face on top of, in this case, local or Isolated Storage and the Lock screen APIs.

Finally, I'll notify the user that the operation completed successfully with a MessageBox. I want to see this in action so I can visualize how this will all work when it's doing what I have in my mind's eye.



Perfect. That will be great ... but we have a lot of code to write to get this working correctly.

4. Add the LockScreenHelper.cs Class File and write the desired functionality  
We'll add a new item (Class) to the AroundMe project.



1. In the Visual C# templates ...
2. Make sure you create a Class file template
3. And most importantly, name it: LockScreenHelpers.cs
4. Click Add

Recall from our gameplan for this lesson AND from the TODO list comments we added a moment ago that we'll need to enable three vital functionalities in this helper class:

1. Clean out any images that were already in local storage,
2. Save the new list of images into local storage, and ...
3. Select a random image and make it the phone's lock screen image

We'll start with the first task, deleting any existing files that may be around from previous AroundMe sessions. I write the following code:

```

8  namespace AroundMe
9  {
10     class LockScreenHelpers
11    {
12        private const string IconRoot = "Shared/ShellContent/";
13        private const string BackgroundRoot = "Images/";
14
15    public static void CleanStorage()
16    {
17        using (IsolatedStorageFile storageFolder =
18            IsolatedStorageFile.GetUserStoreForApplication())
19        {
20            // can't call Delete Directory as some files may
21            // be in use for background or icons
22            TryToDeleteAllFiles(storageFolder, BackgroundRoot);
23            TryToDeleteAllFiles(storageFolder, IconRoot);
24        }
25    }
26
27    private static void TryToDeleteAllFiles(
28        IsolatedStorageFile storageFolder,
29        string directory)
30    {
31        if (storageFolder.DirectoryExists(directory))
32        {
33            try
34            {
35                string[] files = storageFolder.GetFileNames(directory);
36
37                foreach (string file in files)
38                {
39                    storageFolder.DeleteFile(directory + file);
40                }
41            }
42            catch (Exception)
43            {
44                // could be in use
45            }
46        }
47    }
48
49 }

```

I have two methods: (1.) CleanStorage(), and a private method called (2.) TryToDeleteAllFiles() which is a helper function. Both of these are static meaning we will not have to create an instance of the LockScreenHelpers class. Helper classes and helper methods are usually static ... they don't need to maintain "state" per se ... we just need them to be a convenient container for our methods. We'll pass in the state (values) we need to operate on.

CleanStorage() merely grabs a reference to the local storage for our app, and passes it to the TryToDeleteAllFiles() helper method, along with the sub folder we want to clean out. These two sub folder names / locations are stored as constants at the top of the class.

The Images folder is something we will create in the next step to store copies of the images we selected from the LongListMultiSelector. The Shared/ShellContent folder is used for start page tiles, the app list icons and such.

Back in the AppBarButton\_Click() event handler, next we call the LockScreenHelpers.CleanStorage() method:

```
90
91     // Clean out / remove all images currently in IsolatedStorage
92     LockScreenHelpers.CleanStorage(); ← Red arrow here
93
94     // Save this new list of selected images to IsolatedStorage
95
96
97     // Randomly select one item and set it as the lockscreen
98
99     MessageBox.Show("You have a new background!", "Set!", MessageBoxButton.OK);
100
101 }
```

Next, we'll turn our attention to saving the list of selected images to IsolagedStorage. We'll implement another method in the LockScreenHelpers.cs file:

```
51 1 public static void SaveSelectedBackgroundScreens(List<FlickrImage> data)
52 {
53     2 var stringData = JsonConvert.SerializeObject(data);
54
55     3 using (var storageFolder = IsolatedStorageFile.GetUserStoreForApplication())
56     {
57         4 using (var stream = storageFolder.CreateFile(LockScreenData))
58         {
59             5 using (StreamWriter writer = new StreamWriter(stream))
60             {
61                 writer.Write(stringData);
62             }
63         }
64     }
65 }
```

1. In the appBarButton\_Click() event handler, we've already added the images selected in the LongListMultiSelector to a List<FlickrImage> called "imgs". We'll pass that collection in here because we want those images saved to IsolatedStorage.
2. We'll convert (serialize) the List<FlickrImage> to Json for easy storage
3. We'll get this app's IsolatedStorage area ... remember why we use the "using" statement? To properly release unmanaged resources like IsolatedStorage.
4. We'll create a file and save the handle to it in a variable ... we'll use that handle to write to that new file in a moment. Handles to files involve unmanaged resource, so we utilize the using statement again here. Note the name of the file ... I'll create a constant string with the name in a moment.
5. We employ a StreamWriter to write the Json into a new file.

Back to callout #4 (above) ... I'll create a constant with the name of the file:

```

10  namespace AroundMe
11  {
12      class LockScreenHelpers
13      {
14          private const string IconRoot = "Shared/ShellContent/";
15          private const string BackgroundRoot = "Images/";
16          public const string LockScreenData = "LockScreenData.json";
17
18          public static void CleanStorage()
19      }

```



Back in the appBarButton\_Click() event handler, we'll pass our List<FlickrImage> called imgs to our new LockScreenHelpers.SaveSelectedBackgroundScreens() method.

```
78     private void appBarButton_Click(object sender, EventArgs e)
79     {
80         // Get a list of selected images
81         List<FlickrImage> imgs = new List<FlickrImage>();
82
83         foreach (object item in PhotosForLockscreen.SelectedItems)
84         {
85             FlickrImage img = item as FlickrImage;
86
87             if (img != null)
88                 imgs.Add(img);
89         }
90
91         // Clean out / remove all images currently in IsolatedStorage
92         LockScreenHelpers.CleanStorage();
93
94         // Save this new list of selected images to IsolatedStorage
95         LockScreenHelpers.SaveSelectedBackgroundScreens(imgs); ←
96
97         // Randomly select one item and set it as the lockscreen
98
99         MessageBox.Show("You have a new background!", "Set!", MessageBoxButtons.OK);
100    }
101}
```

The next part is a bit more difficult. We want to choose an image from IsolatedStorage randomly to be the lock screen image. We'll tackle this in three parts:

1. We'll open the Json file containing our List<FlickrImage>
2. We'll randomly choose one of the items in that collection
3. We'll call a helper method to actually do the dirty work of assigning that image as the new lock screen image

```

71     public static async Task SetRandomImageFromLocalStorage()
72     {
73         string fileData;
74
75         1   using (IsolatedStorageFile storageFolder
76             = IsolatedStorageFile.GetUserStoreForApplication())
77         {
78             if (!storageFolder.FileExists(LockScreenData))
79                 return;
80
81             using (IsolatedStorageFileStream stream
82                 = storageFolder.OpenFile(LockScreenData, FileMode.Open))
83             {
84                 using (StreamReader reader = new StreamReader(stream))
85                 {
86                     fileData = reader.ReadToEnd();
87                 }
88             }
89         }
90
91         List<FlickrImage> images
92         = JsonConvert.DeserializeObject<List<FlickrImage>>(fileData);
93
94         2   if (images != null)
95         {
96             Random rndNumber = new Random();
97             int index = rndNumber.Next(images.Count);
98
99             Debug.WriteLine(index + ":" + images[index].Image1024);
100
101            3   await SetImage(images[index].Image1024);
102         }
103     }
104

```

1. Since we've been reading and writing to IsolatedStorage quite a bit in this series, hopefully much of this code should look familiar, or at the very least, you should be able to figure out what's going on here. Lines 75 through 92 are opening up the Json file from IsolatedStorage and deserializing it back into a List<FlickrlImage>. We do a check to make sure the Json file actually exists, and we open it and read it using a StreamReader.
2. Assuming the List<FlickrlImage> is not empty, we will choose a random number with an upper bounds set to the number of items in the List<FlickrlImage> collection. If we have the debugger open while we're testing this, we will be able to see which image was chosen.
3. We'll send the image to a helper method that will set it as the lock screen image ... we'll work on that next ...

```

101
102     public static async Task SetImage(Uri uri)
103     {
104         string fileName = uri.Segments[uri.Segments.Length - 1];
105
106         string imageName = BackgroundRoot + fileName;
107         string iconName = IconRoot + fileName;
108
109         using (IsolatedStorageFile storageFolder
110             = IsolatedStorageFile.GetUserStoreForApplication())
111         {
112             if (!storageFolder.DirectoryExists(BackgroundRoot))
113                 storageFolder.CreateDirectory(BackgroundRoot);
114
115             if (!storageFolder.FileExists(imageName))
116             {
117                 using (IsolatedStorageFileStream stream
118                     = storageFolder.CreateFile(imageName))
119                 {
120                     HttpClient client = new HttpClient();
121
122                     byte[] flickrResult = await client.GetByteArrayAsync(uri);
123
124                     await stream.WriteAsync(flickrResult, 0, flickrResult.Length);
125
126                     storageFolder.CopyFile(imageName, iconName);
127                 }
128             }
129         }
130
131         await SetLockScreen(fileName);
132     }

```

1. Here, we want to isolate just the file name portion of the Uri. Uri.Segments splits up the Uri path into an array of strings containing the parts between the forward-slashes.

For more information about Uri.Segments: <http://msdn.microsoft.com/en-us/library/system.uri.segments.aspx>

Essentially, by saying we want the last segment (0 based, so we get the number of segments and subtract 1) we're asking for just the file name ... i.e., the last segment.

2. Here we attempt to locate the larger version of the requested image from IsolatedStorage. Those images should be stored in the Images sub-folder (i.e., the constant BackgroundRoot). If that folder doesn't exist, create it.
3. If the image file we're looking for doesn't exist, then we'll need to download it from Flickr. First, create a new file and hang on to that reference. We'll use it in a moment.
4. Then, create a new HttpClient and try to download the image using the Uri for the current image. It will come back as a byte array.
5. Attempt to save the byte array to the file we created in callout #3 (above).

6. Not only do we want the image copied into our Images folder, but we also want it copied into the Shared/ShellContent folder because if the app is pinned to the Start page, we'll change it to be the image on the tile as well.
7. Now that we can confirm the images (both images, for the lock screen AND the Start page tile) are in place, we can attempt to set them as the current images. We'll encapsulate that functionality into it's own helper, SetLockScreen() method which I'll implement next:

```

150
137 1 private static async Task SetLockScreen(string fileName)
138 {
139 2     // This article describes how to programmatically
140     // take control of the lockscreen background:
141     // http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206968\(v=vs.105\).aspx
142
143     // Right-click WMAppManifest.xml, select Open With ...
144     // Choose 'XML (Text) Editor with Encoding'
145     // Choose AutoDetect when asked about which Encoding
146
147     // IMPORTANT: Add this under </Tokens> section:
148     //
149     /*
150         <Extensions>
151             <Extension ExtensionName="LockScreen_Background"
152                 ConsumerID="{111DFF24-AA15-4A96-8006-2BFF8122084F}"
153                 TaskID="_default" />
154         </Extensions>
155     */
156     // That will make our app a Lock Screen Background "provider".
157
158     // Lockscreen Design Guidelines
159     // http://msdn.microsoft.com/en-us/library/windowsphone/design/jj662927\(v=vs.105\).aspx
160
161 3     bool hasAccessForLockScreen
162         = LockScreenManager.IsProvidedByCurrentApplication;
163
164 4     if (!hasAccessForLockScreen)
165     {
166         // If you're not the provider, this call will prompt the user for permission.
167         // Calling RequestAccessAsync from a background agent is not allowed.
168         var accessRequested = await LockScreenManager.RequestAccessAsync();
169
170         // Only do further work if the access was granted.
171         hasAccessForLockScreen = (accessRequested == LockScreenRequestResult.Granted);
172     }
173
174 5     if (hasAccessForLockScreen)
175     {
176         Uri imgUri = new Uri(
177             "ms-appdata:///local/" + BackgroundRoot + fileName,
178             UriKind.Absolute);
179         LockScreen.SetImageUri(imgUri);
180     }
181

```

1. Since we'll be using features of the Windows Phone 8 API (specifically around the LockScreenManager) that are async, we'll need to declare this entire helper method as async.

2. This comment is pretty important. Not only does it describe the entire process of setting the lock screen, it also calls you attention to the need to manually edit the WMAppManifest.xml file with an Extension. We'll implement that step in just a moment. I also provide the link to important guidelines if you want to integrate your app with the phone's lock screen. "With great power comes great responsibility."
3. Only one app can be the LockScreenManager at a time. The LockScreenManager has the power to modify the LockScreen. Obviously, we want OUR app to be the LockScreenManager, at least while we're attempting to change the lock screen's image. Other apps could change other features of the lock screen, too, but as long as our app is running in the background and the user has selected images, it will attempt to change the lock screen image. Here, we're simply trying to ascertain if our app is indeed the current LockScreenManager and therefore, whether it has access to the lock screen.
4. If our app does NOT currently have access to the lock screen, then we'll request it. This could take a little while so it is an async method. If its successful, the we'll update the hasAccessForLockScreen bool, setting it to "true".
5. Now if we have access to the lock screen, then we'll set the lock screen image.

Continuing on in the SetLockScreen() method declaration, we'll now focus on the Start page tile:

```

181
182   1     var mainTile = ShellTile.ActiveTiles.FirstOrDefault();
183
184   2     if (null != mainTile)
185   {
186       Uri iconUri = new Uri("isostore:/// + IconRoot + fileName, UriKind.Absolute);
187       var imgs = new List<Uri>();
188       imgs.Add(iconUri);
189
190   4     CycleTileData tileData = new CycleTileData();
191     tileData.CycleImages = imgs;
192     //tileData.IconImage = imgUri;
193
194   5     mainTile.Update(tileData);
195   }
196
197 }
```

1. ShellTile.ActiveTiles returns a collection of an application's Tiles pinned to Start. Here we want to grab the first one (or null, if the app is not pinned to Start).
2. Now, as long as the app IS PINNED to Start, then we can modify the tiles that are cycled through on the Start page.
3. We construct a Uri for the new image that has been randomly selected and add it to a List<Uri>.
4. We create a CycleTileData object and set its CycleImages property to the List<Uri> we create ... which only contains a single image. Obviously, with a little ingenuity, we could

expand this to include ALL the images from our selection on the Start tile and have them cycle. That would be a cool feature you could add.

5. Finally we tell the Tile for our Start page to update using the latest CycleTileData object we just created.

Before I forget, we need to add an Extension in the WMAppManifest.xml file as explained in the long comment I added at the top of SetLockScreen() helper method.

Right-click the WMAppManifest.xml file and select Open With ... XML Editor, then add the following code in the <Extensions> element, below:

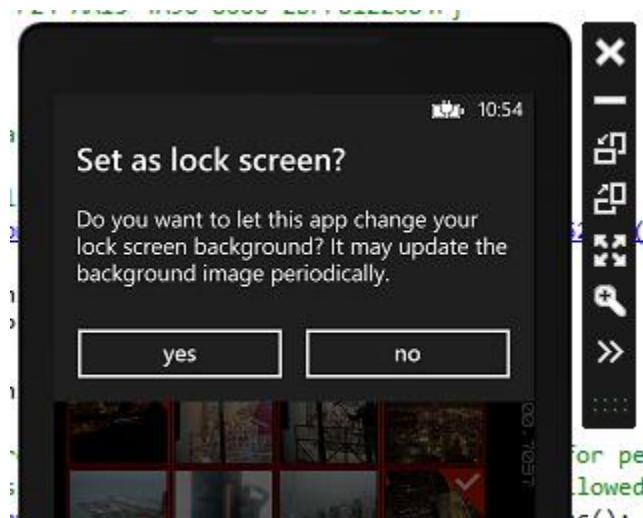


The reason we're doing this is because the WMAppManifest doesn't have a visual way to add extensions. To my knowledge, the only Extension you can even add at this point is for the Lock screen.

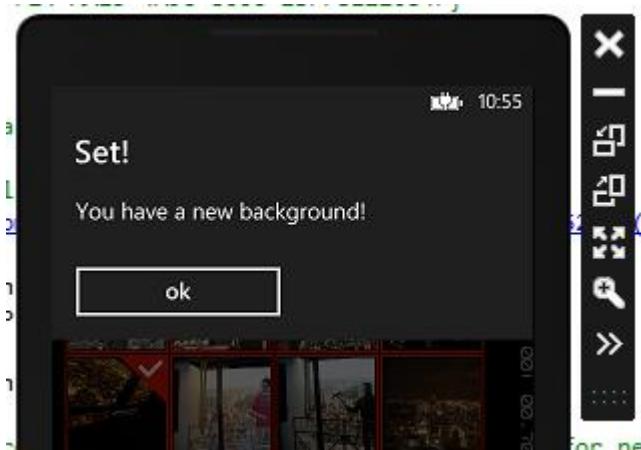
Back in the appBarButton\_Click() event handler, we'll call the LockScreenHelpers.SetRandomImageFromLocalStorage() method:

```
78     private async void appBarButton_Click(object sender, EventArgs e)
79     {
80         // Get a list of selected images
81         List<FlickrImage> imgs = new List<FlickrImage>();
82
83         foreach (object item in PhotosForLockscreen.SelectedItems)
84         {
85             FlickrImage img = item as FlickrImage;
86
87             if (img != null)
88                 imgs.Add(img);
89         }
90
91         // Clean out / remove all images currently in IsolatedStorage
92         LockScreenHelpers.CleanStorage();
93
94         // Save this new list of selected images to IsolatedStorage
95         LockScreenHelpers.SaveSelectedBackgroundScreens(imgs);
96
97         // Randomly select one item and set it as the lockscreen
98         await LockScreenHelpers.SetRandomImageFromLocalStorage(); ←
99
100        // Test by hitting F12 twice in the emulator
101
102        MessageBox.Show("You have a new background!", "Set!", MessageBoxButtons.OK);
103    }
104}
```

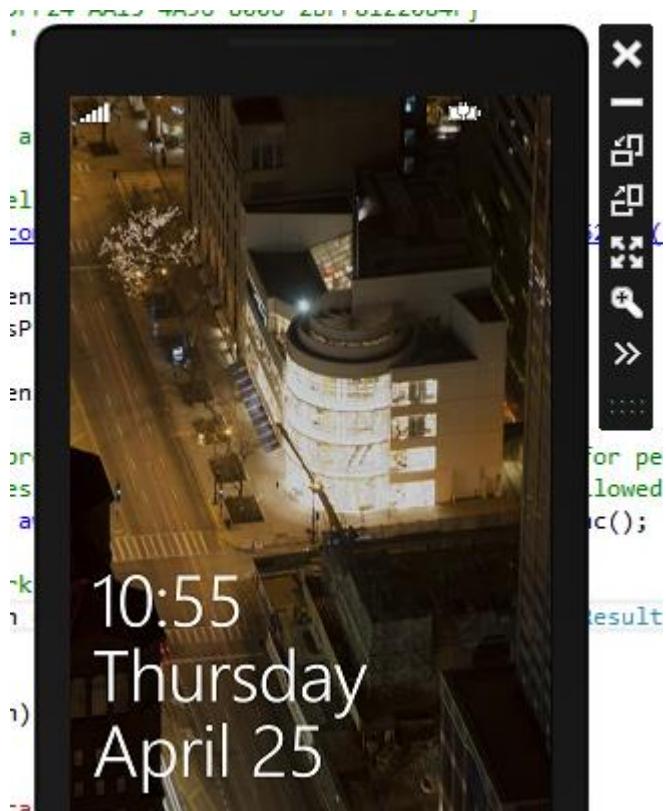
Now it's time to test. Run through our usual routine to select a number of images you want on your lock screen, then select the checkmark icon. It should reveal the "Set as lock screen" dialog as before:



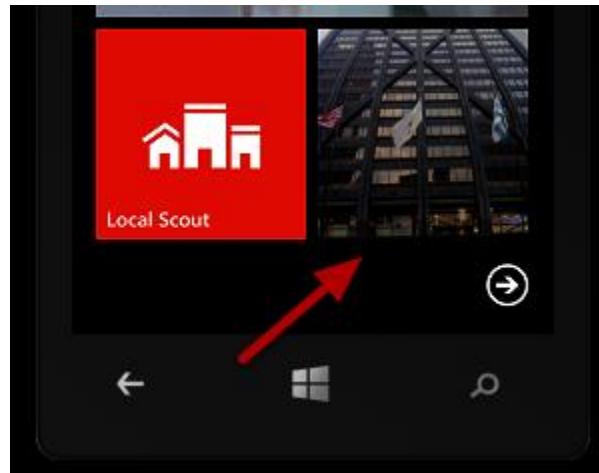
When you select "yes" you see the notification that the background has been set:



And now when you hit the F12 button twice on your keyboard (to simulate clicking the power button twice) you should see a different lock screen:



Furthermore, as the lock screen changes, so does the current Start page tile:



Outstanding!

## Recap

To recap, most of what we did in this lesson was very specific to creating an app that controls a lock screen. While it's interesting, very few developer will need to create this specific style of app. That's not really the big take away from this lesson, in my opinion. Instead, this should give you a sense of the depth of the API and just how many aspects of the app and its integration with the Windows Phone 8 operating system you can actually control. It also should provide a good sense of how you would go about it and the skills you would need to have to make those changes, many of which we covered in this lesson.

# Part 34: Creating a Background Agent for Scheduled Tasks

Source Code: <http://aka.ms/absbeginnerdevwp8>

At this point, our AroundMe app is missing just one final feature: the ability to randomly change the lock screen to one of the pictures we've selected every few moments, even if the AroundMe app isn't currently running.

This functionality requires we use a Scheduled Task Agent. The Scheduled Task Agent runs in the background and executes at specific intervals even when our program is not in the foreground -- in other words, when it is not the app you're working with on screen. In our case, we'll schedule the task to run every 30 seconds. When the scheduled task executes, it will call into our LockScreenHelper class' `SetRandomImageFromLocalStorage()` method.

To enable this functionality we will add a new Windows Phone Scheduled Task Agent project to our current solution. This poses a problem ... we will need to execute the code in our `LockscreenHelper.cs` from our new project. However, currently this file (as well as its dependencies, the `FlickrImages.cs` and `Photo.cs` class files) reside in our main project at the moment. We could merely cut, copy and paste the code files so that they reside in both projects, however the better solution (from a code maintenance perspective) would be to create a third project, a Windows Phone Class Library project, that will house the shared code files, then reference it from both projects. We typically want to share code rather than duplicate it so that any changes or updates can be made in one place and shared by all dependents.

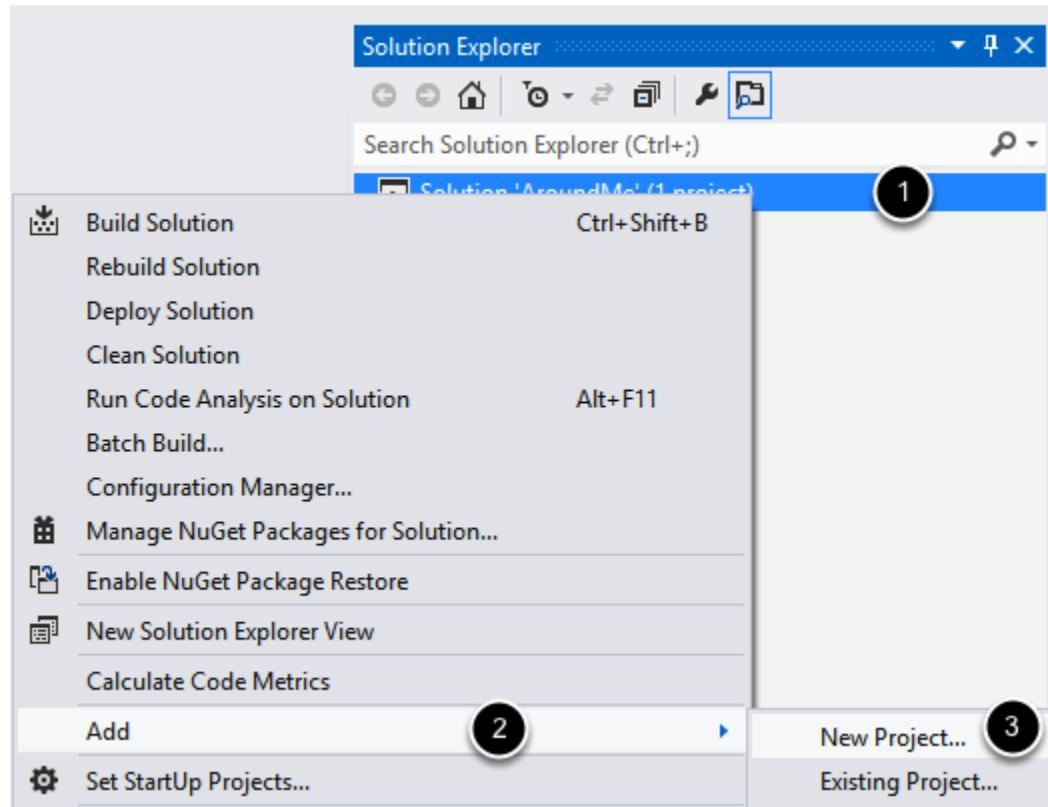
Our game plan:

1. We'll add a new Windows Phone Scheduled Task Agent project to our Solution
2. We'll add a couple lines of code that will call the `SetRandomImageFromLocalStorage()` method and allow us to test it in the emulator
3. We'll add a new Windows Phone Class Library project in our solution and will move our three class files into it that will be shared between the other two projects.
4. We'll move those code files into the Windows Phone Class Library, we'll use NuGet to add in Newtonsoft's JSON library and the Microsoft.Net.Http library package, we'll clean up the namespaces and so on.
5. We'll create references from the AroundMe project and the Scheduled Task Agent project to our new Class Library project.
6. In the main AroundMe project, I'll need to introduce the Scheduled Task Agent to the operating system by launching it when AroundMe starts up, and configure the `WMAppManifest.xml` to be an extension in order to allow our app to be a Lock screen Background Provider.
7. And if all goes well, we'll run it and watch the fireworks.
8. I'm not going to submit this app to the store ... we've already seen that process in lesson 23, but I will show you some last steps required before submitting the app to the store,

including how to get a token to properly license and use Map Services in your Phone Store app.

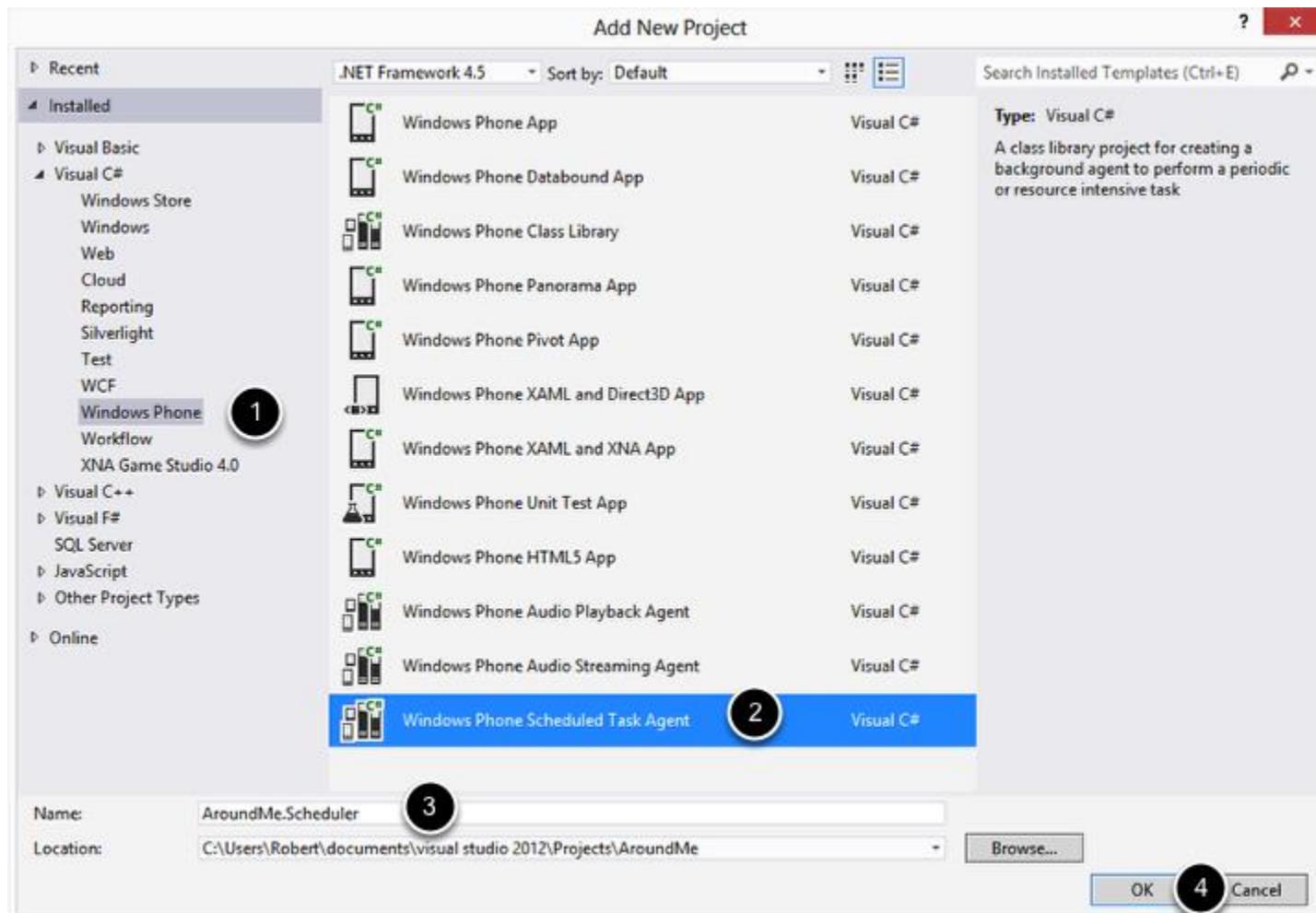
1. Add a new Windows Phone Scheduled Task Agent project to our solution called AroundMe.Scheduler

I don't believe I've ever demonstrated how to add a second project to a solution in the C# Fundamentals series. Assuming you do not know how to do this ...



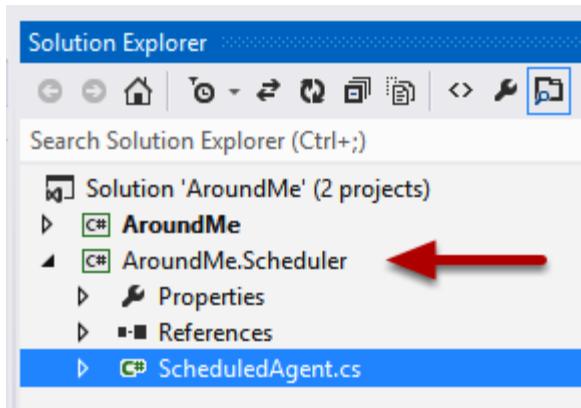
1. Right-click the solution name in the Solution Explorer
2. Select Add
3. New Project ...

The Add New Project dialog appears:



1. Select the Visual C# Windows Phone templates
2. Select Windows Phone Scheduled Task Agent project template
3. Name it: AroundMe.Scheduler ... we'll use a common notation when naming projects, the dot notation, to match what we want the default namespace for our code in this project to have
4. Click OK

You should now see a second project in the Solution Explorer. The most important part of this project is the `ScheduledAgent.cs` class.



Most of code in the ScheduledAgent.cs file is "boiler plate" meaning we will not need to change it, but it must be there in order to create a valid Scheduled Task Agent. However, if you scroll down near the bottom of the file, in line 41 you'll find the `OnInvoke()` method with a TODO comment. This is where we'll implement our code as follows:

```
41     /// <summary>
42     /// </summary>
43     protected override void OnInvoke(ScheduledTask task)
44     {
45         //TODO: Add code to perform your task in background
46
47
48
49         // From:
50         // http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.sch
51
52         // "Use this method during application development to test your background agent
53         // implementation. Background agents are launched by the operating system
54         // according to the type of agent and the current state of the system. This
55         // scheduling logic is described in Background agents for Windows Phone. You
56         // can use this method to launch an agent more frequently from your foreground
57         // application or from the agent itself in order to test the agent execution.
58         // This method should only be used during development. You should remove calls
59         // to this method from your production application."
60
61         ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(30));
62
63         NotifyComplete();
64     }
65 }
```

As you can see, I added a TODO. We'll come back here later and add code that will set the random image from local storage. We have a little setup work to do first.

Before we do that, I've added one line of code (with about 10 lines of comments) that explain it's purpose. I copied it from this source:

[http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.scheduler.scheduledactionservice.launchfortest\(v=vs.105\).aspx](http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.scheduler.scheduledactionservice.launchfortest(v=vs.105).aspx)

So, line 61 of the code snippet above:

```
ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(30));
```

... is merely for testing purposes in the Phone Emulator. This allows developers to watch the behavior of the task every 30 seconds. Otherwise, as we'll learn in a moment, when our app is deployed to a real Phone device, it will only run once every 20 to 40 minutes. More about that in just a moment.

As the comments above it explain, we need to either remove this line of code, or at least wrap it in an if statement like this:

```
[code language="csharp"]if (Debugger.IsAttached) {  
}[/code]
```

The original article references this additional background information:

[http://msdn.microsoft.com/en-US/library/windowsphone/develop/hh202942\(v=vs.105\).aspx](http://msdn.microsoft.com/en-US/library/windowsphone/develop/hh202942(v=vs.105).aspx)

That article is very important for our purposes, and if you think you need to do some background processing or scheduled processing, you MUST read that article. In a nutshell, there are two types of background tasks: ResourceIntensiveTasks and PeriodicTasks. Resource Intensive tasks run in the background because they may take a long time to complete ... they're processor intensive. PeriodicTasks run at set intervals. In our case, we're creating a PeriodicTask.

There are different rules and constraints depending on the type of background task we're working with. Both ResourceIntensiveTasks and PeriodicTasks share the following constraints:

1. An application may have only one background agent, and only one instance of the agent runs at a time.
2. We're limited by which of the Phone's APIs we can utilize from a background task

3. We're limited by the amount of the phone's resources we can use ... no more than 11 megabytes for an application like ours
4. Our scheduled task will only work for two weeks before it needs to be rescheduled by re-running the AroundMe app
5. If our app crashes twice in a row, it will be unscheduled by the operating system

In our case, since we're working with PeriodicTasks, we need to be aware of a few additional things:

6. Our task will run about every 30 minutes, however the phone's operating system decides when to run the tasks. In fact, it might run ALL scheduled tasks at the same time because it's more efficient to use the phone's battery once every 30 minutes than a bunch of times during that same period for each individual Scheduled Task that wants to run. So, that 30 minute time frame can drift as much as 10 minutes in either direction. So, when we deploy our AroundMe app to a phone, it will change image every 20 to 40 minutes or so.
7. Our Scheduled task can only run for 25 seconds before it is shut down by the operating system. That shouldn't affect us, but we should be aware of that for future reference.
8. If the battery is low and the phone goes into Battery Saver mode, then our Scheduled Task may not run at all. If you have a Windows Phone 8 and your battery gets low, you'll see a little heart icon over the battery meaning that it has shut down scheduled tasks, turned off some of the functionality of the phone like the GPS and Bluetooth radio and more. Users can configure how aggressive they want Battery Saver to be in the Settings screen. I have a battery monitoring app on my start page, and I was getting low on battery, then I noticed that Battery Saver mode kicked in and I had 20 hours left. That's because it was throttling what was going on in the background. So, be aware of that when you create your own Scheduled Tasks.
9. Finally, depending on each phone's configuration, there may be a limit to how many background agents can run at the same time. A few days ago I hit that limit and it asked me to choose some background tasks to shut down because the phone was low on memory.

So, if you plan on submitting a phone to the Store you'll need to be willing to support users. If you want to support your users, you'll need to know a little about how the phone and the operating system work with your app. If the user complains that their background is not changing often enough, you might want to ask a few questions about how many apps are currently running.

To see what's running in the background, go to your Windows Phone's settings, swipe over to the applications page. There you will find "background tasks" setting at the top of the list. Once inside the settings, you'll see a list of all the apps running in the background. To shut it down, just tap on the app and tap the "block" button. To confirm that you've shut down the app, press and hold the back button to view any open apps.

At any rate, that article also does a nice job of explaining the general lifecycle of a background agents. From the article:

"When the agent is launched, the operating system calls OnInvoke(ScheduledTask) ... When the agent has completed its task, it should call NotifyComplete() or Abort() to let the operating system know that it has completed. NotifyComplete should be used if the

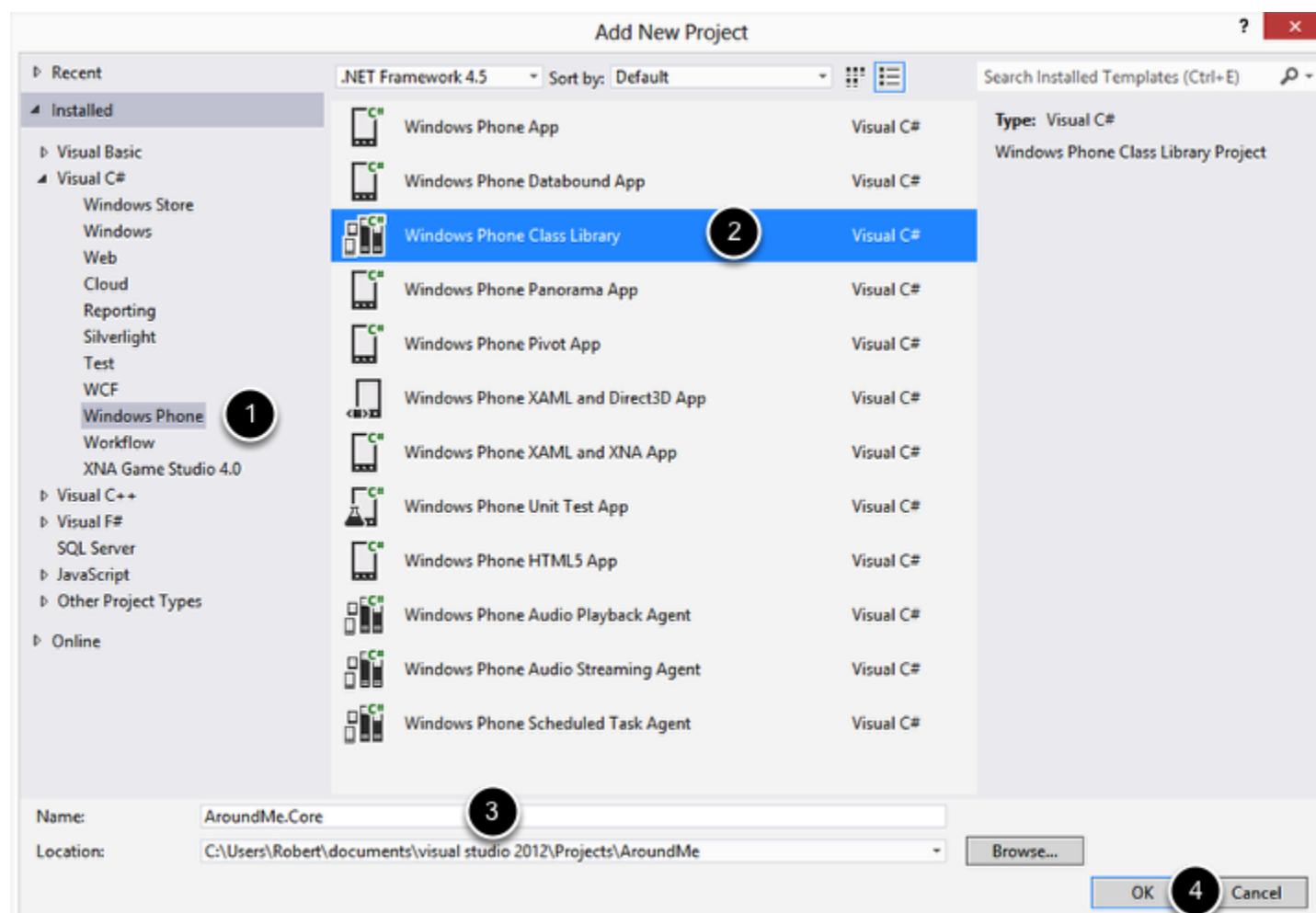
task was successful. If the agent is unable to perform its task – such as a needed server being unavailable - the agent should call Abort, which causes the IsScheduled property to be set to false."

So that is the purpose of the NotifyComplete() in line 63 of the code snippet above.

Again, we'll come back and revisit the TODO comment in line 45 after we've moved our class files from our main AroundMe project to a new Windows Phone Class Library project, which we'll do next.

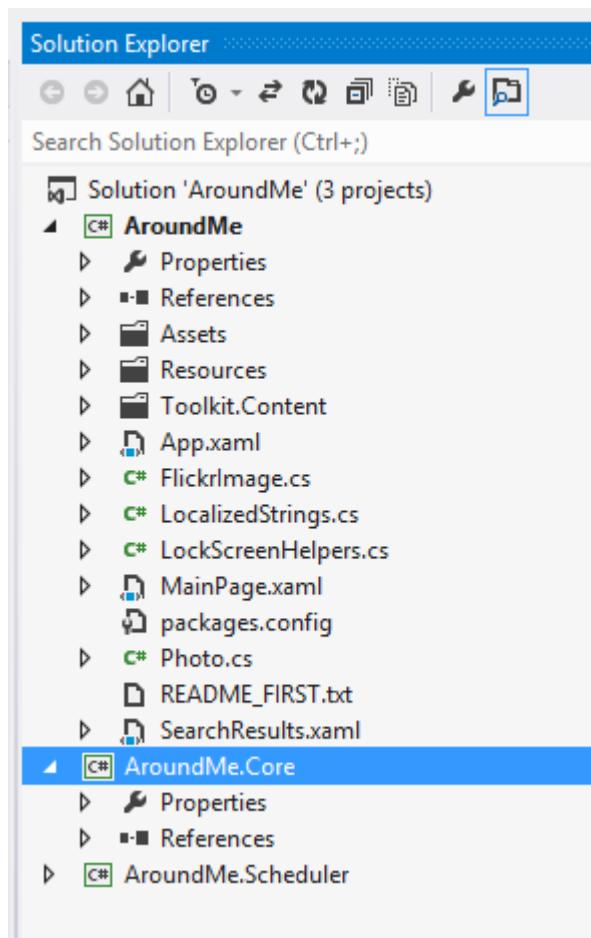
2. Create a new Phone Class Library called AroundMe.Core and move class files to it  
Just like in step 1, we'll add a new project to our solution. So, right-click the our solution entry in the Solution Explorer, select Add, New Project ... In the Add New Project dialog

...



1. Select Visual C# Windows Phone templates
2. Select the Windows Phone Class Library project template
3. Rename to AroundMe.Core ... again, we're following a naming convention that will produce a namespace that we want to use in our solution. The term "Core" indicates essential functionality that will be utilized in other projects
4. Click OK

Your Solution Explorer should look like this, with one solution and three projects:



NOTE: I deleted the Class.cs file from the new AroundMe.Core project. We won't need it. Selected it and hit the delete key on your keyboard.

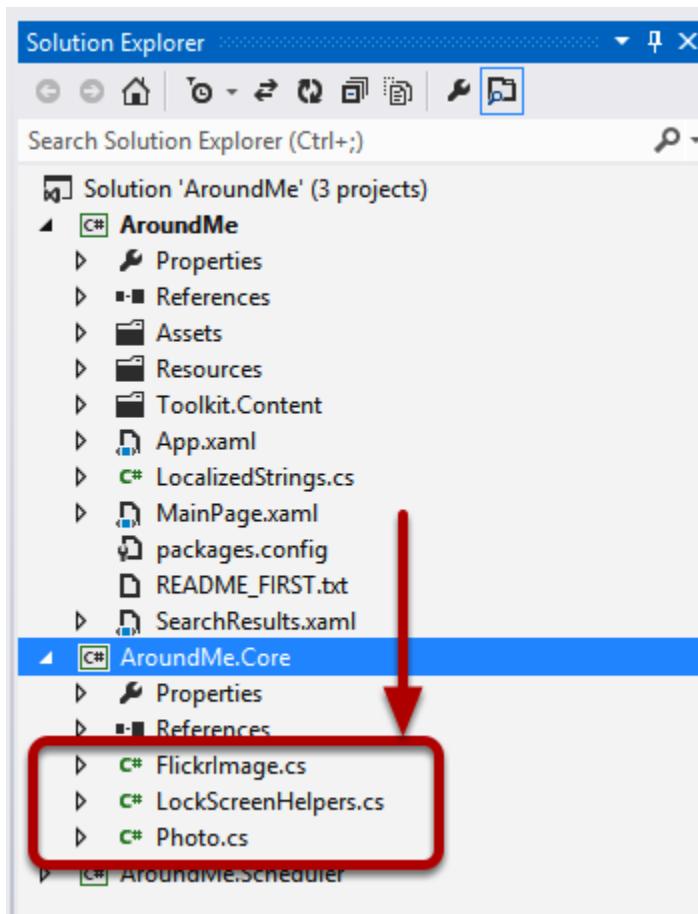
Next, I'll drag my three class files:

- FlickrlImage.cs
- LockScreenHelper.cs

- Photo.cs

... from the AroundMe project to the AroundMe.Core project. THIS WILL MERELY COPY THE FILES ... you now have two copies of the files.

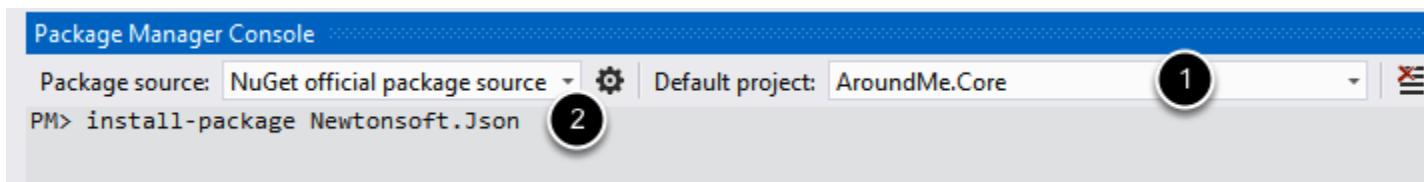
In the AroundMe project, select those three files and delete them. Those three files should only appear in your AroundMe.Core project:



Next, since these files depend on third-party or optional packages, we'll need to use NuGet to import them into our new AroundMe.Core project.

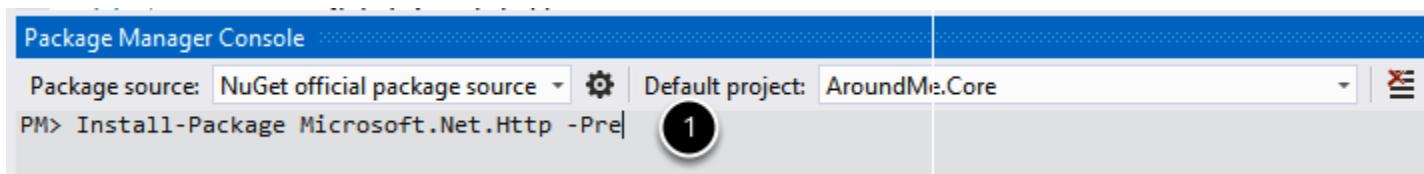
If it's not already open in Visual Studio, go to the Tools menu, select Library Package Manager menu option, then the Package Manager Console sub menu.

In the Package Manager Console:



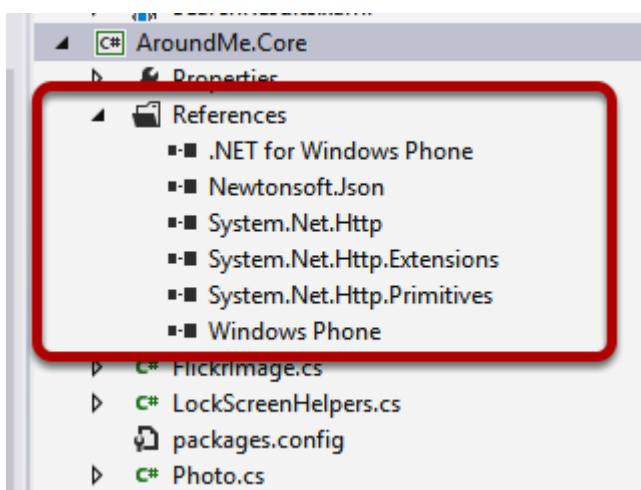
1. Make sure the Default project is set to: AroundMe.Core
2. At the Package Manager prompt, type: **install-package Newtonsoft.Json**
3. Hit enter.

Assuming that completes successfully ...



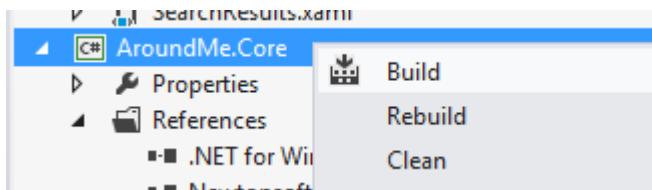
1. At the Package Manager prompt, type: **install-package Microsoft.Net.Http**
2. Hit enter.

We've talked about both of these before, so I'll not review that now. These are simply dependencies that the class files we moved will need. Confirm everything installed correctly in the Solution Explorer:



... your references should match what I have in the AroundMe.Core project.

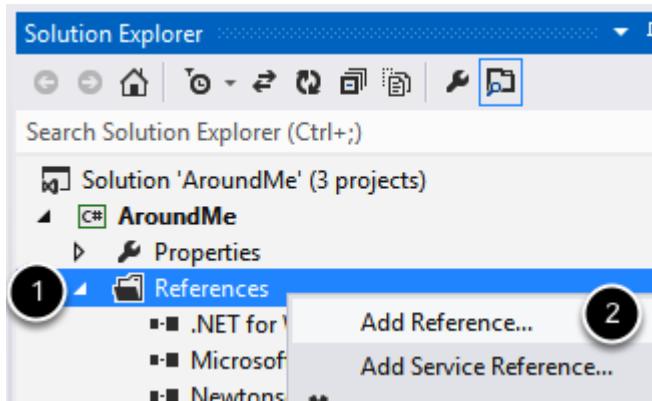
Just to make sure everything works IN THAT PROJECT, right-click the AroundMe.Core project name in the Solution Explorer and select Build from the context menu:



... assuming everything builds without any errors, we're ready to reference our new Windows Phone Class Library project from our main AroundMe project and the Background Task Agent project.

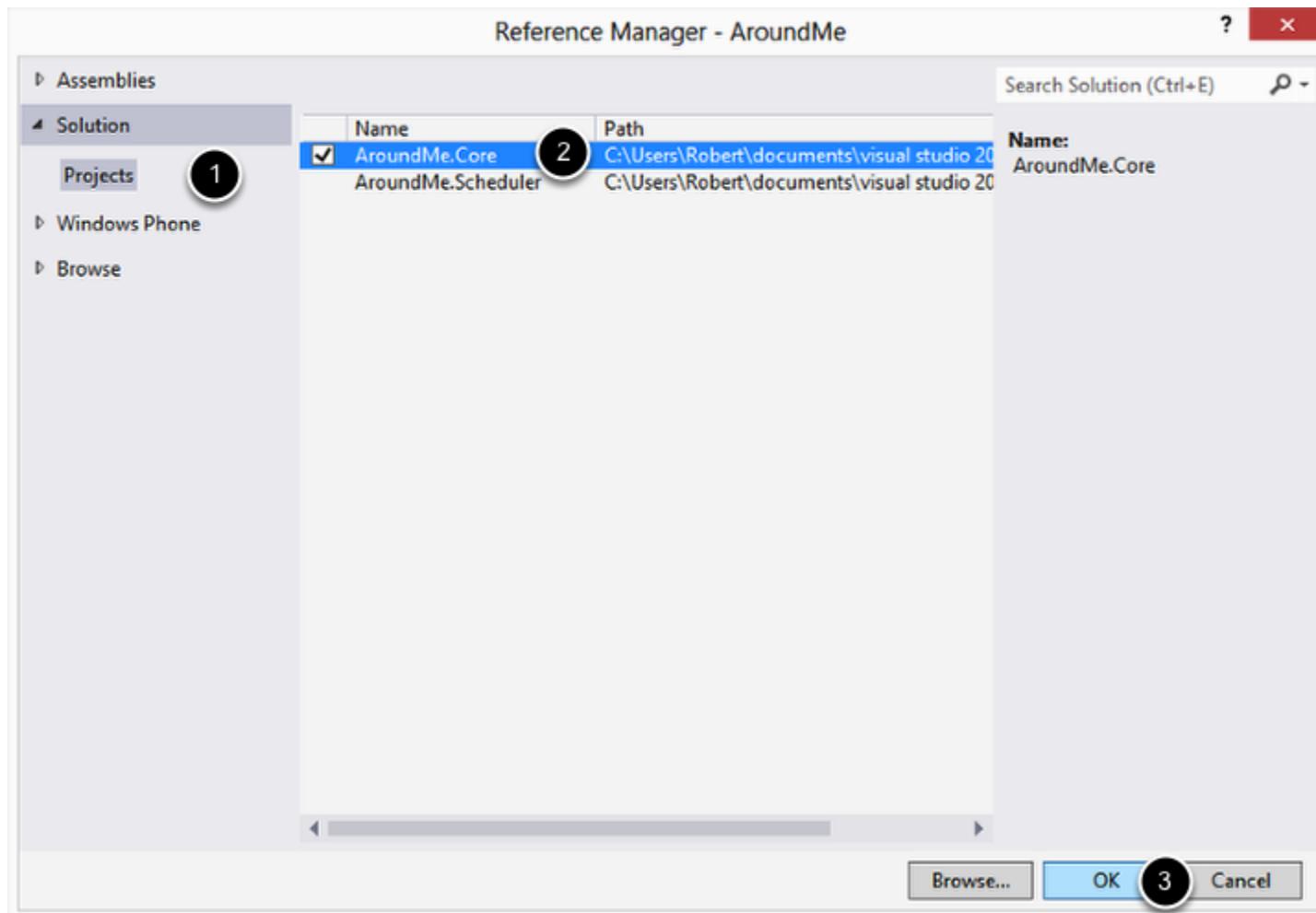
In the Solution Explorer, in the AroundMe project, add a reference to the AroundMe.Core project:

3. Add reference from AroundMe to AroundMe.Core and fix any lingering problems from the refactoring



1. Right-click References
2. Select Add Reference ...

When the Reference Manager dialog appears ...



1. Navigate to the Solution | Projects tab
2. Place a check-mark next to AroundMe.Core
3. Click OK

Now it's time to clean up our AroundMe project's source code. When you move things from one project to another, there's bound to be some problems. In our case, we need to comb through our code and make sure our AroundMe project can find the same code in the new AroundMe.Core Class Library project.

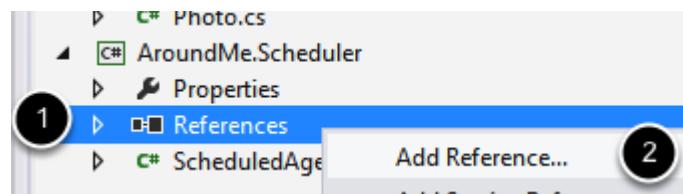
In the `SearchResults.xaml.cs` class, in the `AppBarButton_Click` click event, I find my first problem ... I hover my mouse cursor over the blue squiggly line:

```
--  
91     // Clean out / remove all images currently in IsolatedStorage  
92     LockScreenHelpers.CleanStorage();  
93  
94     'AroundMe.LockScreenHelpers' is inaccessible due to its protection level  
95     LockScreenHelpers.SaveSelectedBackgroundScreens(imgs);  
96  
97     // Randomly select one item and set it as the lockscreen  
98     await LockScreenHelpers.SetRandomImageFromLocalStorage();  
aa
```

... and it tells me: "AroundMe.LockScreenHelpers is inaccessible due to its protection level".

When I see the term "protection level", that usually means a visibility problem ... in other words, our old code can't "see" our new code. And usually that's because I'm missing an accessibility modifier in the new code.

In my LockScreenHelpers.cs file I add the public accessibility modifier to the class declaration:



At this point you may need to re-build the AroundMe.Core project in order for the warnings in the SearchResults.xaml.cs class to go away.

But now that I'm looking at my LockScreenHelpers.cs class, I see something else ... I've moved these three files to a new project, and named the project deliberately to be in a different namespace. However, since I merely copied those files, the declared namespace remains as it did before ... simply:

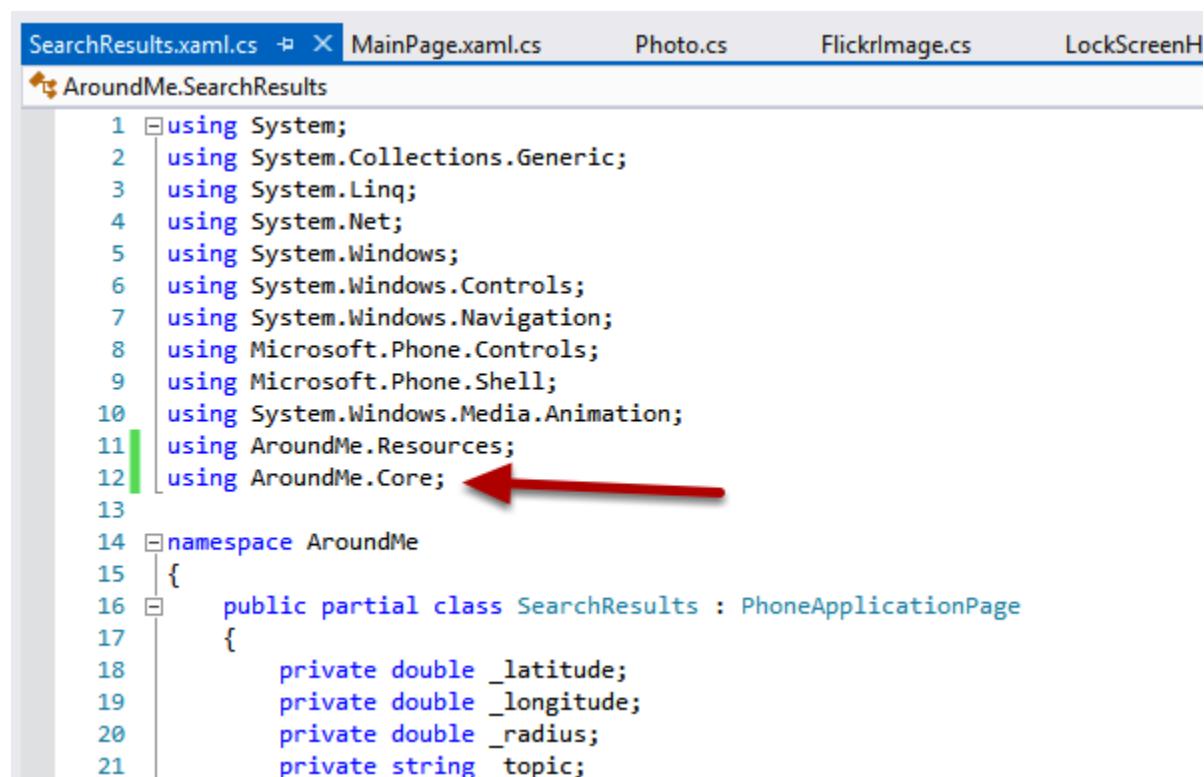
```
[code language="csharp"]namespace AroundMe { }[/code]
```

But do I want the namespace to be AroundMe? I think I would prefer it to match the project name, AroundMe.Core ... so I make that change to each of the class files in my AroundMe.Core project:

```
8
9  namespace AroundMe.Core ←
10 {
11     public class FlickrImage
12     {
13         public Uri Image320 { get; set; }
14         public Uri Image1024 { get; set; }
15     }
```

After I make that change, I Build the AroundMe.Core project again.

Now, I'll need to add a using statement to the SearchResults.xaml.cs file so that the compiler knows where to look for the LockScreenHelpers class:



```
SearchResults.xaml.cs  X MainPage.xaml.cs      Photo.cs      FlickrImage.cs      LockScreenH
找了 AroundMe.SearchResults
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Net;
5  using System.Windows;
6  using System.Windows.Controls;
7  using System.Windows.Navigation;
8  using Microsoft.Phone.Controls;
9  using Microsoft.Phone.Shell;
10 using System.Windows.Media.Animation;
11 using AroundMe.Resources; ←
12 using AroundMe.Core; ←
13
14 namespace AroundMe
15 {
16     public partial class SearchResults : PhoneApplicationPage
17     {
18         private double _latitude;
19         private double _longitude;
20         private double _radius;
21         private string _topic;
```

By simply adding the:

using AroundMe.Core;

... to my using statements, that should fix any namespace problems I introduced with this change.

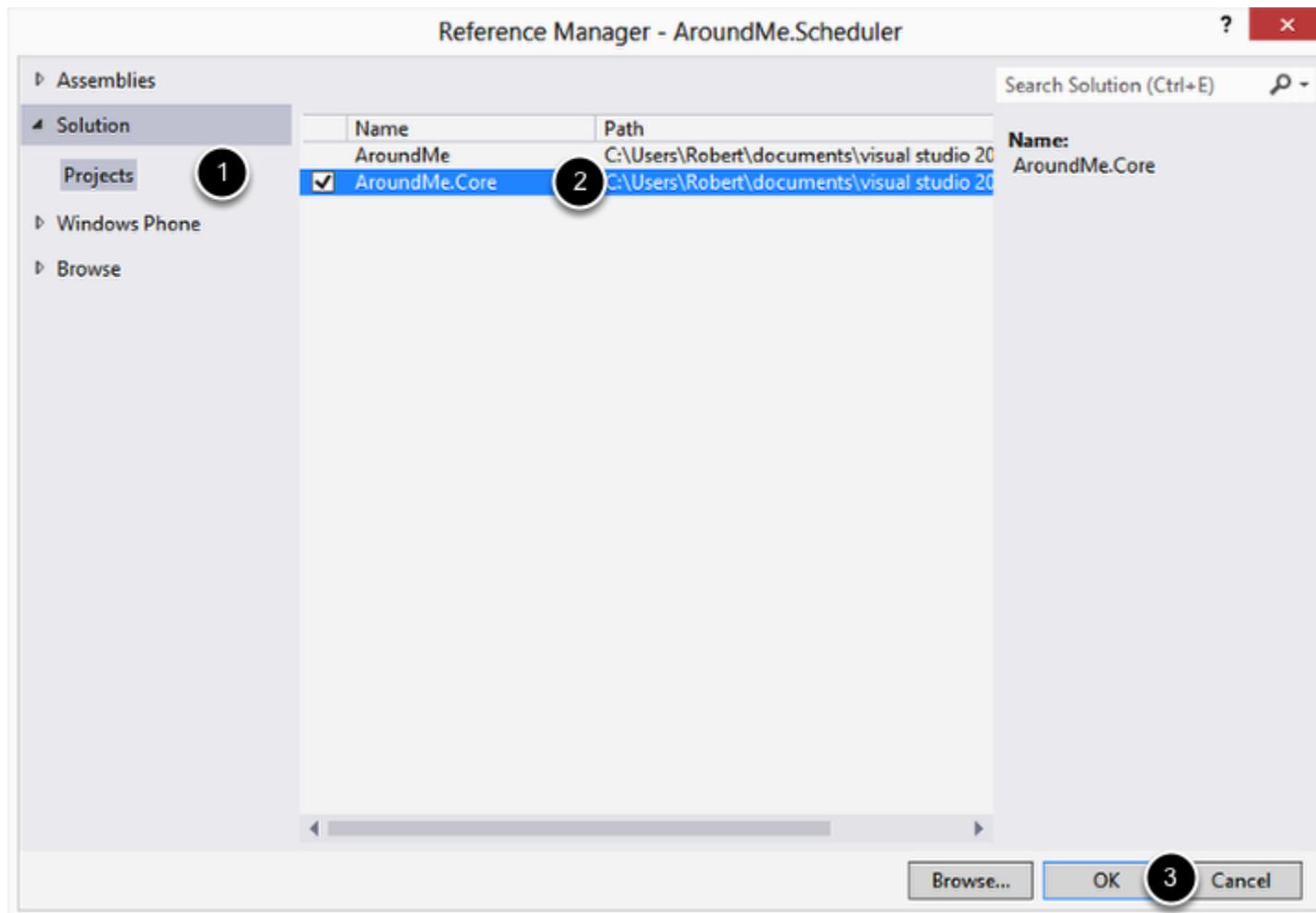
4. Create a reference from AroundMe.Scheduler to AroundMe.Core and write code to set background lock screen image

In the Solution Explorer:

```
11  using System.Threading.Tasks;
12  using Windows.Phone.System.UserProfile;
13
14  namespace AroundMe
15  {
16      public class LockScreenHelpers
17      {
18          private const string IconRoot = "Shared/ShellContent/";
19          private const string BackgroundRoot = "Images/";
```

1. Right-click the References entry of associated with the AroundMe.Scheduler project
2. Select Add Reference...

That will display the Reference Manager dialog ...



1. Select the Solution | Projects tab
2. Place a check-mark in the box next to the AroundMe.Core project
3. Click OK

Next, add the following using statement to the ScheduledAgent.cs file:

```
using AroundMe.Core
```

```
ScheduledAgent.cs  X  SearchResults.xaml.cs      MainPage.cs
AroundMe.Scheduler.ScheduledAgent
1  using System.Diagnostics;
2  using System.Windows;
3  using System;
4  using Microsoft.Phone.Scheduler;
5  using AroundMe.Core; ←
6
7  namespace AroundMe.Scheduler
8 }
```

In that same file, in the `OnInvoke` method near the bottom, it's time to revisit the TODO comment we added earlier.

We'll make the following changes to that method:

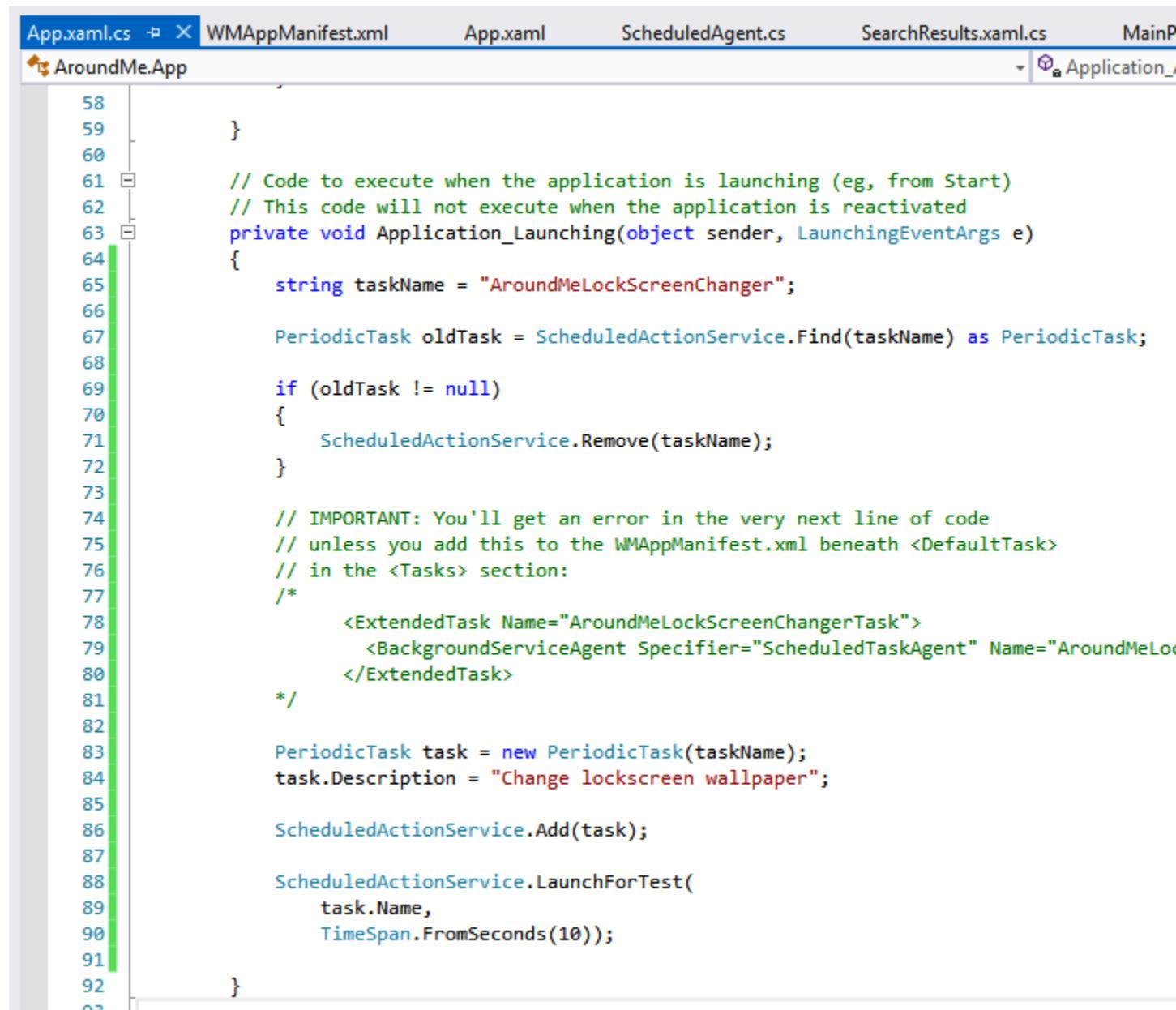
```
42 2 protected async override void OnInvoke(ScheduledTask task)
43 {
44     await LockScreenHelpers.SetRandomImageFromLocalStorage(); 1
45
46     // From:
47     // http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.sch
48
49     // "Use this method during application development to test your background agent
50     // implementation. Background agents are launched by the operating system
51     // according to the type of agent and the current state of the system. This
52     // scheduling logic is described in Background agents for Windows Phone. You
53     // can use this method to launch an agent more frequently from your foreground
54     // application or from the agent itself in order to test the agent execution.
55     // This method should only be used during development. You should remove calls
56     // to this method from your production application."
57
58     ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(30));
59
60     NotifyComplete();
61 }
```

1. Add this line of code at the top, removing the TODO comments:  
`ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(30));`
2. Add the `async` modifier to the method declaration. We've talked about this at length in lesson 27.

5. Modify the AroundMe project's App.xaml.cs to introduce the background scheduled task agent to the operating system

Finally, we'll need to register our new background scheduled task agent with the operating system. We'll do this when our AroundMe app first starts up, so we'll be adding the code to the App.xaml.cs which handles the Application\_Launching event.

In the App.xaml.cs file, add the following code to the Application\_Launching event handler method stub:



The screenshot shows the Visual Studio IDE with the App.xaml.cs file open. The code editor displays the following C# code for the Application\_Launching event handler:

```
58
59
60
61     }
62
63     // Code to execute when the application is launching (eg, from Start)
64     // This code will not execute when the application is reactivated
65     private void Application_Launching(object sender, LaunchingEventArgs e)
66     {
67         string taskName = "AroundMeLockScreenChanger";
68
69         PeriodicTask oldTask = ScheduledActionService.Find(taskName) as PeriodicTask;
70
71         if (oldTask != null)
72         {
73             ScheduledActionService.Remove(taskName);
74
75             // IMPORTANT: You'll get an error in the very next line of code
76             // unless you add this to the WMAppManifest.xml beneath <DefaultTask>
77             // in the <Tasks> section:
78             /*
79                 <ExtendedTask Name="AroundMeLockScreenChangerTask">
80                     <BackgroundServiceAgent Specifier="ScheduledTaskAgent" Name="AroundMeLoc
81                 </ExtendedTask>
82             */
83
84             PeriodicTask task = new PeriodicTask(taskName);
85             task.Description = "Change lockscreen wallpaper";
86
87             ScheduledActionService.Add(task);
88
89             ScheduledActionService.LaunchForTest(
90                 task.Name,
91                 TimeSpan.FromSeconds(10));
92         }
93     }
```

There are two basic activities in this code passage:

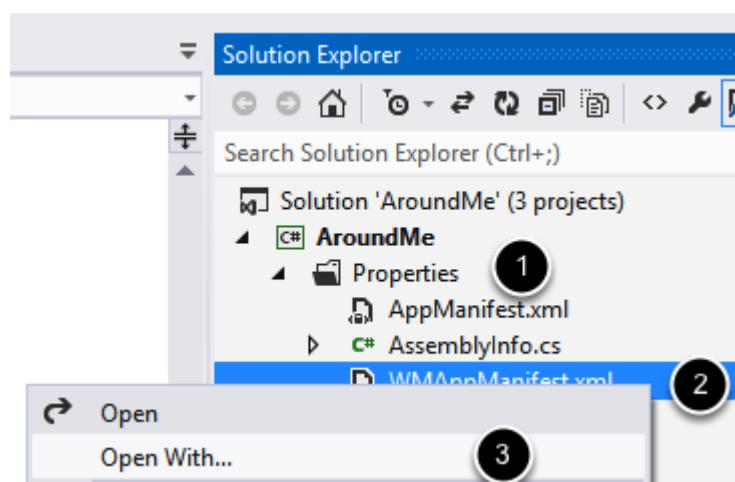
- Lines 65 - 72 -- Here we are trying to find any running versions of our background scheduled task agent. If it is running, we'll want to remove it from the ScheduledActionService which acts as a registrar for the operating system's scheduled tasks.
- Lines 83 - 90 -- Here we create an instance of our background scheduled task agent and register it with the ScheduledActionService.

Why do we try to remove it, then re-add it? Keep in mind what we said earlier ... after 2 weeks, our scheduled tasks will be de-scheduled. This little dance simply keeps our background scheduled task agent scheduled and operating for another two weeks.

Notice that we set the task's Description property in line 84 ... this will be displayed to the end user if they look at the background tasks in Settings at the very bottom of the page where you can choose to block the app. It's a good reminder to the user of what this particular background scheduled task agent is doing ... in case they've forgotten that they installed your app.

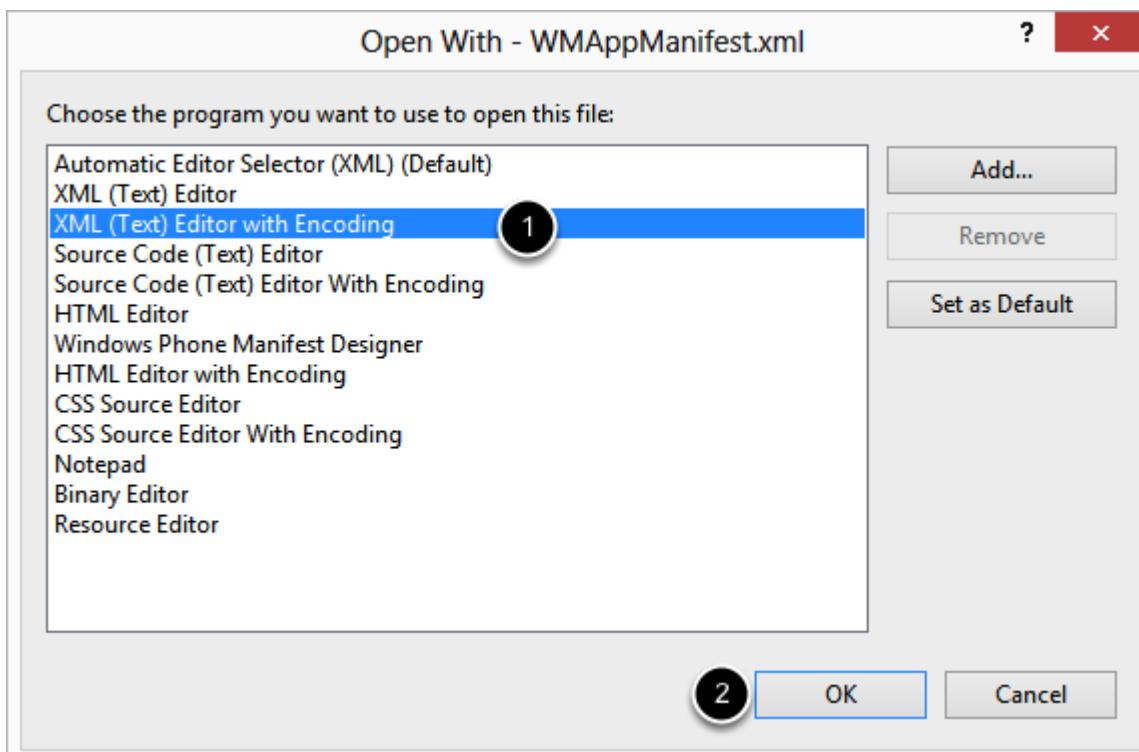
- In line 88 I'm creating this LaunchForTest() method like I did in the AroundMe.Scheduler to schedule the background task for its first execution for development purposes in the Phone Emulator.
- Lines 74 - 81 -- this is a reminder to me and you ... we'll need to open up the WMAppManifest.xml file and add this ExtendedTask element.

We've already done this once or twice, but just to refresh your memory:



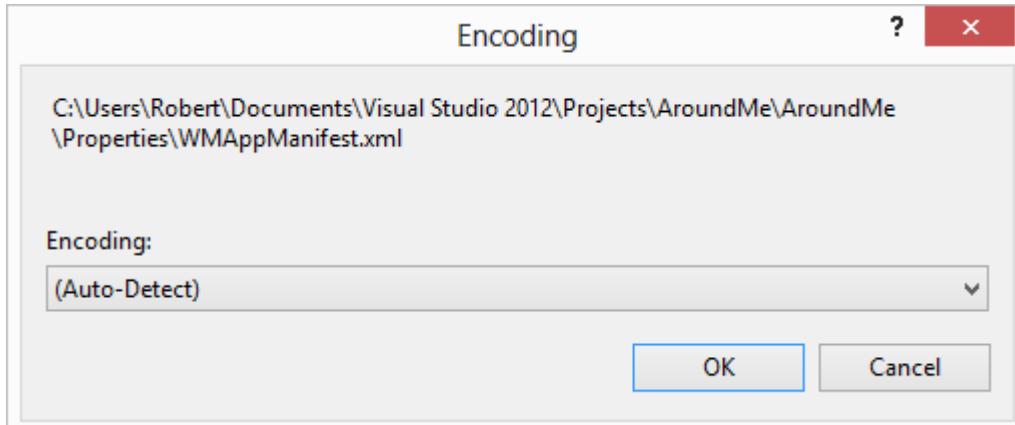
1. In the Properties section / folder ...
2. Right-click on the WMAppManifest.xml file ...
3. Choose Open With ... from the menu

From the Open With dialog:



1. Select XML (Text) Editor with Encoding
2. Click OK

You will be asked about the "encoding" part in the next dialog:



Simply leave the "auto-detect" option and click OK.

We'll want to make changes INSIDE of the <Tasks> element, below the <DefaultTask> element like so:

```
<?xml version="1.0" encoding="utf-8"?>
<Deployment xmlns="http://schemas.microsoft.com/windowsphone/2012/deployment" AppPlatformVersion="7.0">
    <DefaultLanguage xmlns="" code="en-US" />
    <App xmlns="" ProductID="{c44a5ba2-474f-4106-a598-b9ba9b433727}" Title="AroundMe" RuntimeType="Windows Phone" Version="1.0.0.0">
        <IconPath IsRelative="true" IsResource="false">Assets\ApplicationIcon.png</IconPath>
        <Capabilities>
            <Capability Name="ID_CAP_NETWORKING" />
            <Capability Name="ID_CAP_MEDIALIB_PLAYBACK" />
            <Capability Name="ID_CAP_SENSORS" />
            <Capability Name="ID_CAP_WEBBROWSERCOMPONENT" />
            <Capability Name="ID_CAP_LOCATION" />
            <Capability Name="ID_CAP_MAP" />
        </Capabilities>
        <Tasks>
            <DefaultTask Name="_default" NavigationPage="MainPage.xaml" />
            <ExtendedTask Name="AroundMeLockScreenChangerTask">
                <BackgroundServiceAgent Specifier="ScheduledTaskAgent" Name="AroundMeLockScreenChanger" Specification="BackgroundTaskSpecification">
                    </BackgroundServiceAgent>
                </ExtendedTask>
            </Tasks>
            <Tokens>
                <PrimaryToken TokenID="AroundMeToken" TaskName="_default">
                    <TemplateCycle>
                        <SmallTemplateIRT TcRelative="true" TcResource="false">Assets\Tiles\FlinCvleATileSmall.png</SmallTemplateIRT>
                    </TemplateCycle>
                </PrimaryToken>
            </Tokens>
        </App>
    </Deployment>

```

We'll add the following XML:

```
<ExtendedTask Name="AroundMeLockScreenChangerTask">
<BackgroundServiceAgent Specifier="ScheduledTaskAgent"
Name="AroundMeLockScreenChanger" Source="AroundMe.Scheduler"
Type="AroundMe.Scheduler.ScheduledAgent" />
</ExtendedTask>
```

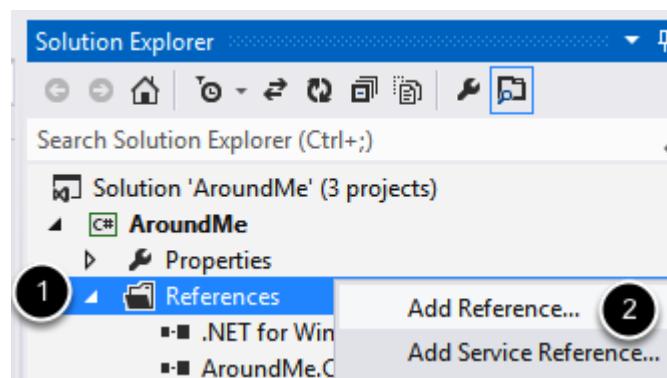
What does this do? Per this web page: [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769509\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769509(v=vs.105).aspx)

... in the section titled "ExtendedTasks Element" about mid-way through, it says:

"The ExtendedTasks element is a child of the Tasks element and contains BackgroundServiceAgent elements. This element defines the use of multiple tasks by an app. Extended tasks are named by the developer. Currently, you can use the ExtendedTasks element only to define background tasks."

Remember, the WMAppManifest.xml is there mostly for certification purposes into the Store, and is used to introduce your app to the phone's operating system and allow it to become part of the phone's ecosystem. Here's a great example ... we tell the phone's operating system that we want this part of our app, the AroundMe.Scheduler to be included with the other background task agents that are currently running. We're registering with the background task agent scheduler to let it know what we want it to run within our app, namely, the AroundMe.Scheduler, we give it a name, a description and so on.

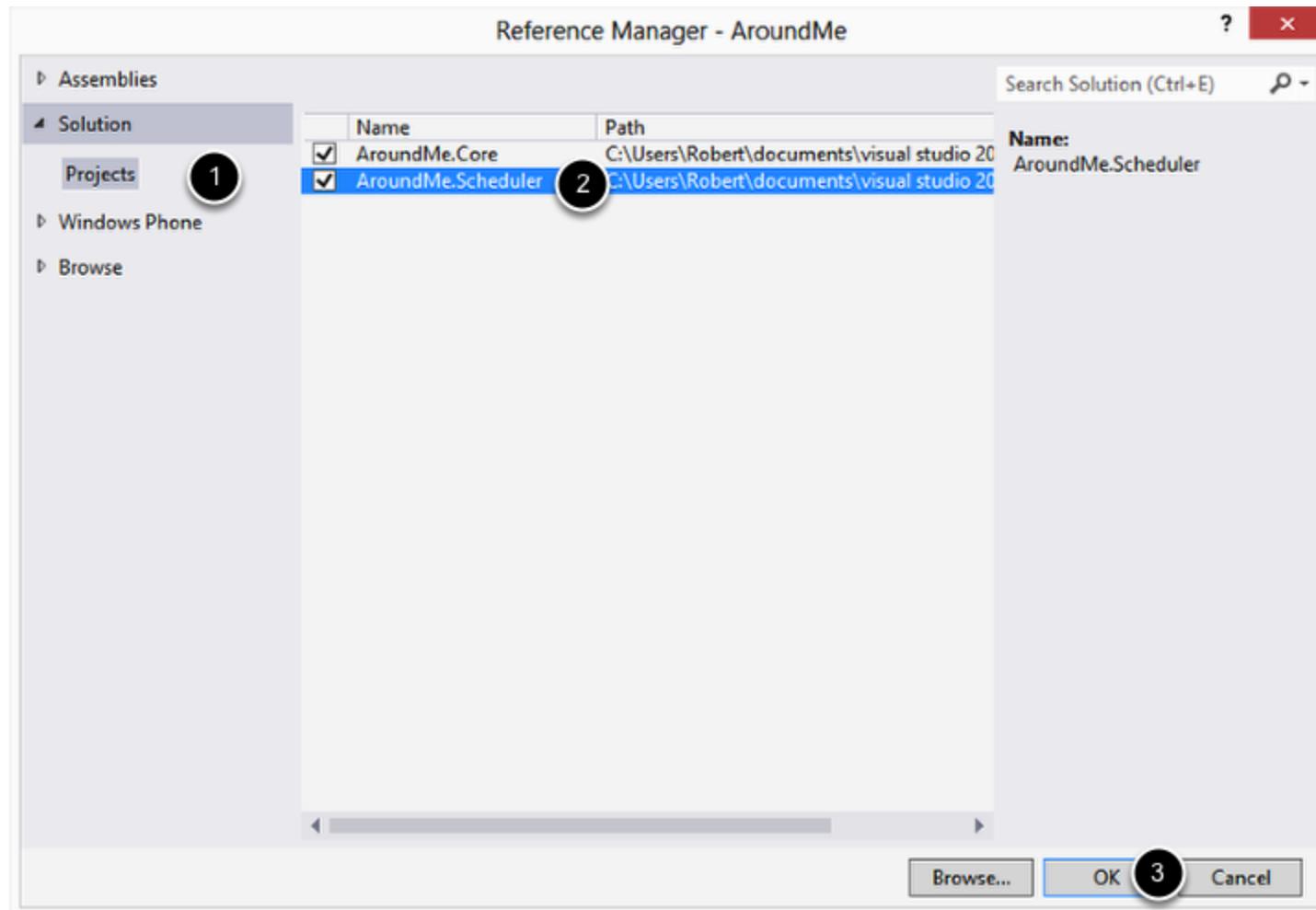
In order for the main AroundMe app to register our AroundMe.Scheduler, we'll need to add a reference to it:



1. In the AroundMe project, right-click the References folder

2. Choose Add Reference ...

This will pop up the Reference Manager dialog ...



1. Choose the Solutions | Project tab
2. Place a check-mark next to the AroundMe.Scheduler project
3. Click OK

Now we should be able to run our application in the Phone Emulator to see it at work.

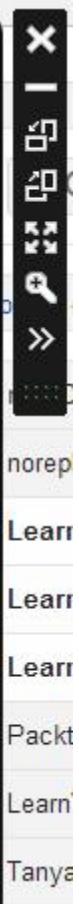
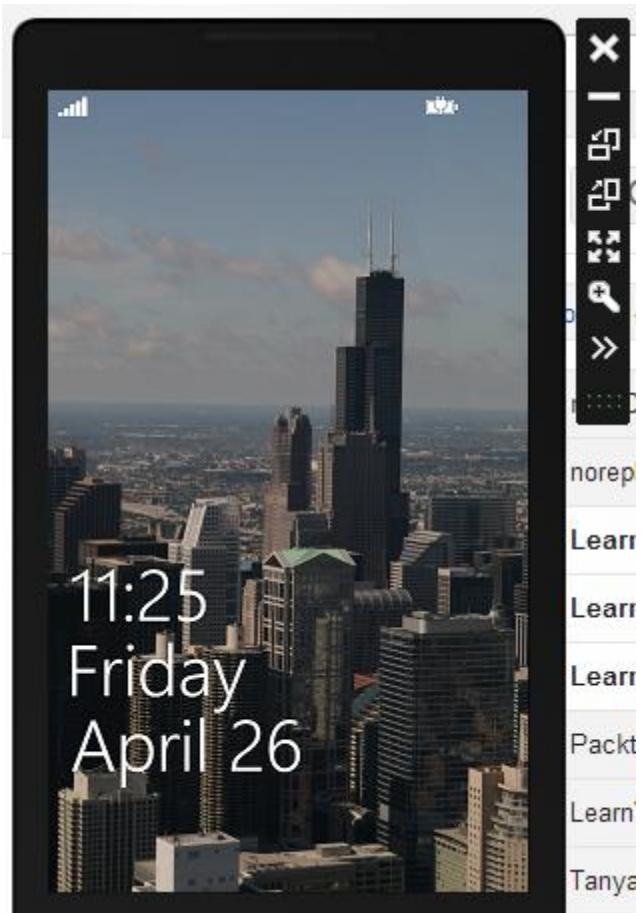
5. Run the app in the Phone Emulator to make sure it works.

When running the app from a previous try, you may have left it on the lock screen to see whether or not it worked. If you did leave the Emulator on the lock screen, you may see this error message:

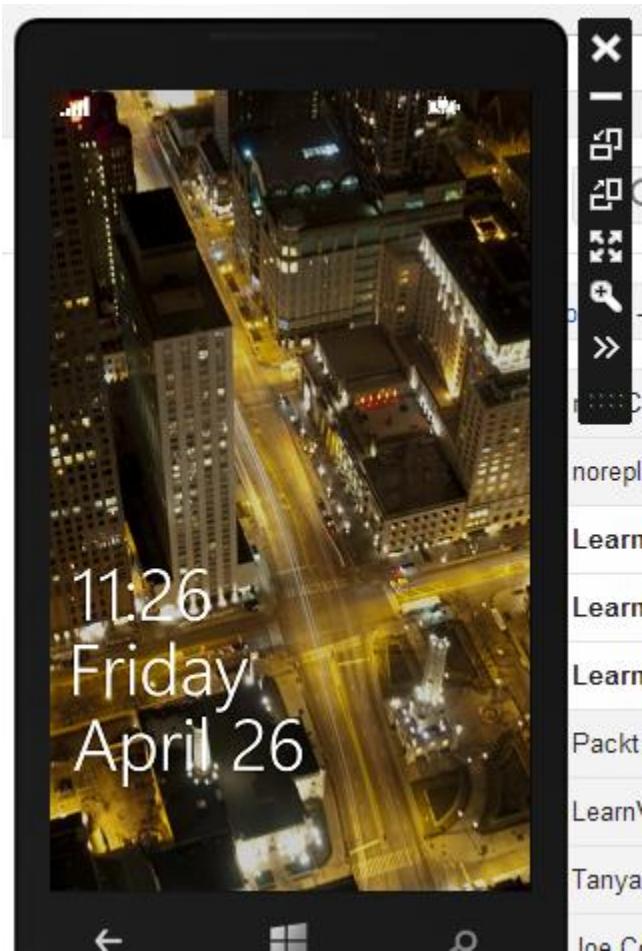


You merely need to unlock the Phone Emulator's lock screen by hitting F12 on the keyboard and / or use the swipe motion to unlock the phone. Then you can re-run / debug the app.

To test this, you just need to go through our usual script ... i.e., search, select a few Flickr images, click the Set button, approve the dialog boxes that appear, then hit the F12 key on your computer's keyboard twice to simulate clicking the phone's power button twice to lock then awaken the phone. Undoubtedly your photos will look different from mine.



After 30 seconds (maybe a little more, maybe a little less) it should change images to another of your selections.



And so we've completed the development of our project successfully!

But wait, there's just one or two more things we need to do before we submit this for inclusion in the store.

6. Clean up the code removing development only code, adding other details before submitting to the Store

There are a few final steps that we would need to take before we're ready to ship this and submit it to the Store for inclusion.

First, you'll recall in this lesson we used the `LaunchForTest()` method a couple of times to test the scheduled background task agent. Remember this?

```
42     protected async override void OnInvoke(ScheduledTask task)
43     {
44         await LockScreenHelpers.SetRandomImageFromLocalStorage();
45
46         // From:
47         // http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.scheduler.scheduledagent.cs#L11
48
49         // "Use this method during application development to test your background agent
50         // implementation. Background agents are launched by the operating system
51         // according to the type of agent and the current state of the system. This
52         // scheduling logic is described in Background agents for Windows Phone. You
53         // can use this method to launch an agent more frequently from your foreground
54         // application or from the agent itself in order to test the agent execution.
55         // This method should only be used during development. You should remove calls
56         // to this method from your production application."
57         ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(30));
58
59         NotifyComplete();
60     }
61 }
62 }
```



We'll need to search the entire solution for the LaunchForTest method ... recall we have a reference to it as depicted (above) in the AroundMe.Scheduler project's ScheduledAgent.cs file as well as in the AroundMe project's App.xaml.cs file. As I said earlier, you either want to comment these out OR wrap them in a check for the debugger:

```
if (Debugger.IsAttached) {  
}
```

Obviously we'll want to thoroughly test our app. I would recommend allowing friends and family to use it, or better yet, find a group of developers in your area that would be willing to take a look and make recommendations. I was at a Nokia developer event recently and there were a lot of developers exchanging business cards and showing each other what they were working on. This can be a great way to inexpensively find testers, and to build some contacts in the development community. So, I would recommend getting involved by checking in on their Events page:

<http://www.dvlup.com/Events>

Another option is to locate the Microsoft developer evangelist in your area and ask them if they have any events scheduled. I would search for something like:

- Microsoft developer evangelist [Your city, state, region or country here]
- windows phone user group [Your city, state, region or country here]

NOTE: You may have to be creative and try different wording to find a group or evangelist near you.

Developer evangelists usually have blogs, twitter accounts or mailing lists through MSDN or TechNet that you can subscribe to in order to learn about meet-ups, user groups or code camps in your local area. Or you could always write the developer evangelist an email and introduce yourself.

Once you're ready to submit to the store, you should pay particular attention to the APIs you'll be calling. For example, we used the Windows Phone 8 Maps control and Flickr. Make sure the account you're using is up to date, and you're using the correct credentials.

With regards to the Map control, we will need to get an ApplicationId and an Authentication Token from the developer center. For a full explanation of this, check out this page:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207033\(v=vs.105\).aspx#BKMK\\_appidandtoken](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207033(v=vs.105).aspx#BKMK_appidandtoken)

In a nut shell, I'll add a Loaded event handler:

```
34
35     <Grid x:Name="ContentPanel"
36         Grid.Row="1"
37         Margin="12,0,12,0" >
38     <Grid.RowDefinitions>
39         <RowDefinition />
40         <RowDefinition Height="Auto" />
41     </Grid.RowDefinitions>
42
43     <maps:Map Name="AroundMeMap" Loaded="AroundMeMap_Loaded" />
44
45     <StackPanel Grid.Row="1">
46         <TextBlock
47             Foreground="{StaticResource PhoneSubtleBrush}"
48             Text="Topic" />
49         <TextBox
50             Name="SearchTopic"
51             Margin="-12, 0" />
52     </StackPanel>
53 </Grid>
```

And will implement the code like so ... I've comment it out for now:

```
57  
38     void AroundMeMap_Loaded(object sender, RoutedEventArgs e)  
39     {  
40         // before you push to the store, these must be set.  
41         // you get these values from the dev center  
42         //MapsSettings.ApplicationContext.ApplicationId = "ApplicationID";  
43         //MapsSettings.ApplicationContext.AuthenticationToken = "AuthenticationToken";  
44     }  
45
```

If you recall from Lesson 23, we saw an optional set of tasks including Map Services ...

#### Optional



Add in-app advertising  
Getting paid through ads? It's all here.



Market selection and custom pricing  
For apps, you have the option to define different pricing and availability for different countries/regions.



Map services  
Get the token required to use map services in your app.



[Review and submit](#)

... this is where you would retrieve the token and the application id required to use map services in your app. You would merely replace those literal strings with the supplied tokens from the store.

#### Recap

To recap, the big takeaway was how to create a scheduled task and share common code in a library, and how to test the scheduled task in the Emulator. We also learned about final steps, not the least of which is how to receive a token for Map Services.

In the next lesson, I'll have just a few comments and will wrap up this series.

## Part 35: Where to go from here

Congratulations, you made it to the end of another video series. That's no small task—it took a lot of commitment to do that, not to mention time and attention, but hopefully you enjoyed the series and were compiling ideas for building your own apps and gaining confidence—not to mention the knowledge of the API, the techniques for working in the Visual Studio IDE and the Phone Emulator, and so on.

Moving forward, I recommend a few things ...

1. Stay on top of updates with the Windows Phone blog:

[http://blogs.windows.com/windows\\_phone/b/windowsphone/](http://blogs.windows.com/windows_phone/b/windowsphone/) ... I just realized I hadn't been there in a few days and there was a great promotion that I'm going to take advantage of when I finish recording this video. It contains the news and updates you'll want to know about straight from Microsoft.

Also, the great WPCentral site: <http://www.wpcentral.com/> ... will keep you abreast of new devices and apps that come out for the platform.

Also, more pertinent to this series, the Windows Phone Developer blog:

[http://blogs.windows.com/windows\\_phone/b/wpdev/](http://blogs.windows.com/windows_phone/b/wpdev/)

2. To continue your learning, check this out:

What's new in Windows Phone SDK 8.0

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206940\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206940(v=vs.105).aspx)

It gives a great overview of many of the new and advanced features.

3. If you haven't already, you'll want to get a Phone Developer Center Membership. It will take a while for verification through Symantec, even longer if you're like me and have a tiny business with no history with Dun & Bradstreet or a phone number you want made public.
4. If you haven't done so already, you'll want to get a Windows Phone 8 device. I highly recommend the Lumia 920 ... I've been using mine for about a month. And I'm just an individual here—I don't speak on behalf of Microsoft when I say that I'm no stranger to the devices of other manufacturers and platforms of other companies—but this phone and the ecosystem around it, including the apps, the Surface, and my new over-powered Windows 8 computer have been a blast. I love pinning tiles and seeing them update with new information. I love the built-in Skydrive support, especially in OneNote and Office for the Windows Phone 8. And the camera is awesome. And I've never owned a Nokia product before, but I'm really impressed with the phone itself and, moreover, with the dedication to the Windows Phone 8 platform.
5. So, my last recommendation is the dvlup.com program at Nokia. <http://www.dvlup.com>

They are incentivizing developers to create apps and register them on the site to win prizes through a currency called DVLUP Reward Points. You can trade them in to get free phones, advertisements, or special placement in the store. They also have challenges ... at this moment, they have 42 challenges running. Complete the challenge and depending on the complexity of the challenge you earn points.

I want to sincerely thank Randy Arnold, who is a Nokia Developer Ambassador in my area, for taking me under his wing and explaining Nokia's strategy, showing me how DVLUP works and why it's a great opportunity for developers as well as how it supports the Windows Phone 8 platform.

I want to thank Larry Lieberman and Desiree Lockwood for their support in getting me the assets I needed for this series.

As always, thanks to Golnaz and Dan Fernandez because they're always awesome.

And most of all, thanks to Clint Rutkas who is "the man". Not only did he do most of the heavy lifting up front by building these great apps, which we had the pleasure of learning from, he also patiently answered over a hundred emails from me asking for help and guidance as I was learning the new features of the Windows Phone 8 as well as his approach and thought process as he set out to build these apps. It was a valuable learning experience for me, and hopefully for you, too.

If you like what we did here and want more like it, make sure you let Channel 9 and Microsoft know. They will get you the resources you want.

And if you liked this series, there's plenty more where this came from. Please take 3 minutes to check out my website:

<http://www.LearnVisualStudio.NET>

... where I have tons of video training just like this on a wide variety of .NET topics.

Finally, thank you for your kind attention and as we part ways I sincerely wish you the best. Let me know what you've built—I'd love to check it out. If I can ever help, email me at: [bob@learnvisualstudio.net](mailto:bob@learnvisualstudio.net)

Thank you.

**Source Code:** <http://aka.ms/absbeginnerdevwp8>