Lab 01: Creating our First iOS Application

Prerequisites

You will need a Windows machine with both Visual Studio (2010 or better) and Xamarin tools installed for a Business or Enterprise plan. You will also need a Mac with Xcode and Xamarin tools installed to serve as a build host for Visual Studio to use. We will be using the iOS Simulator to test the code we are building. If you need help setting up your environment, go through the online instructions here for Xamarin installation on Mac:

http://docs.xamarin.com/guides/ios/getting_started/installation/

and here for getting iOS development working in Visual Studio:

http://docs.xamarin.com/guides/ios/getting_started/introduction_to_xamarin_ios_for_visual_studio/

Downloads

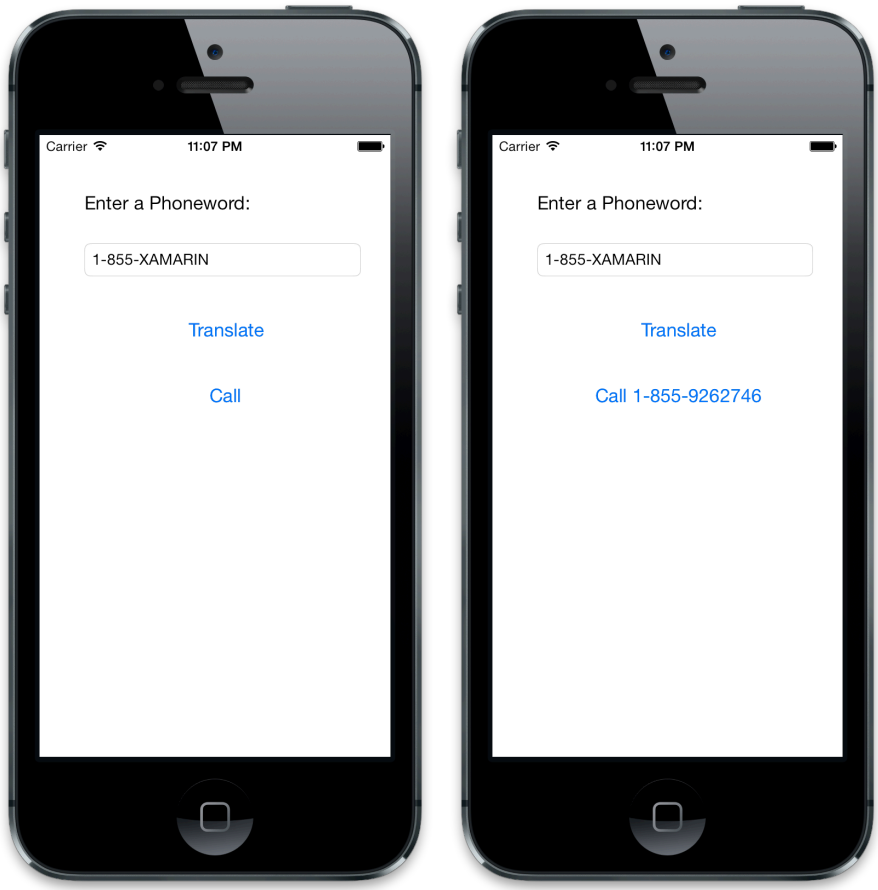
Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is Lab 01 Resources. Make sure you have this folder before you begin.

Lab Goals

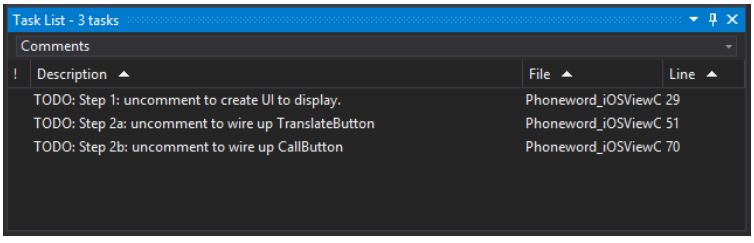
The goal of this lab will be to create our first Xamarin.iOS application. During the process we will become familiar with the tools within Visual Studio, and the various parts of our application. By completing this lab you will gain experience with the following tools:

- Visual Studio with Xamarin.iOS – Overview of iOS Toolbar in Visual Studio.
- Anatomy of a Xamarin.iOS Application – The component parts of a Xamarin.iOS application.
- Creating your application's user interface (UI) directly in code.
- Deployment – How to deploy your application to the iOS Simulator.

In addition, by completing this lab, you will be introduced to the core parts of a Xamarin.iOS application through the creation of a simple application that translates a phone number with letters in it, i.e. 1.855.XAMARIN, and then allows the user to dial the number as shown in the following screen shots:



The lab has been provided as a starter solution with most of the code already filled in for you – as you following along with the instructor you will make small changes for each step, either writing a little code or uncommenting a block of code. Most of these steps are clearly marked in the supplied solution with `// TODO:` comments. These comments are picked up by Visual Studio and shown in the Task List, which you can make visible through the **View > Task List** menu item. When the Tasks List is open, make sure it is showing Comments in the drop-down filter, it will look like this:



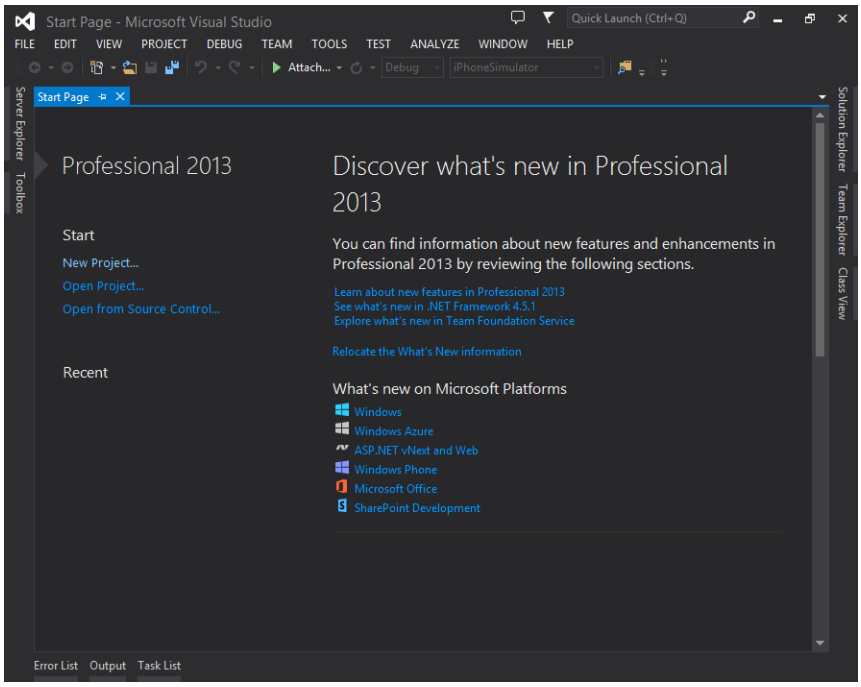
You can quickly jump to the code by clicking on the task itself to keep up with the lecture as the instructor runs through this lab. If you need additional time to complete a task or need some help please let the instructor know – the goal is for you to work through this lab in the class itself.

Steps

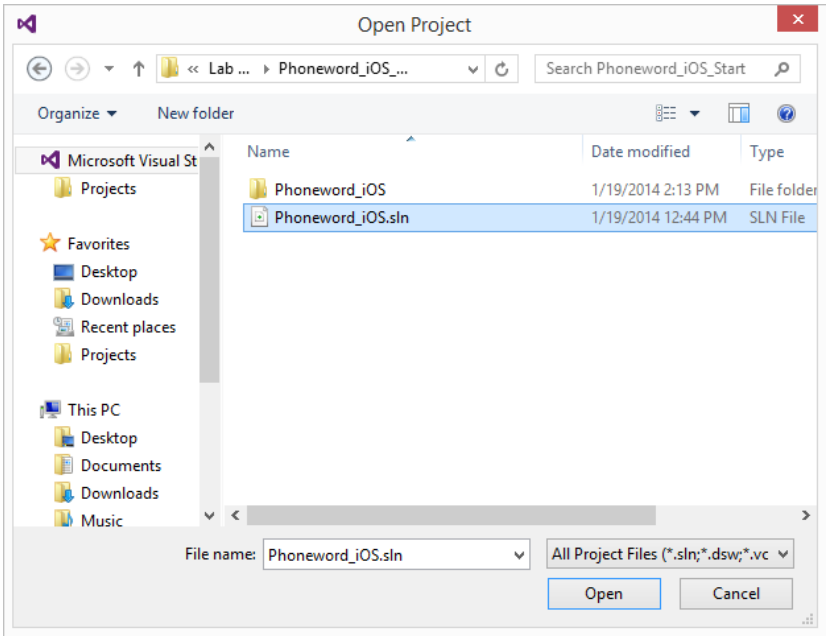
Open the Starting Solution

1. If it isn't already running, launch **Xamarin.iOS Build Host** from the Mac's **Application** folder.

2. Launch **Visual Studio 2013.** It should open and look something like:



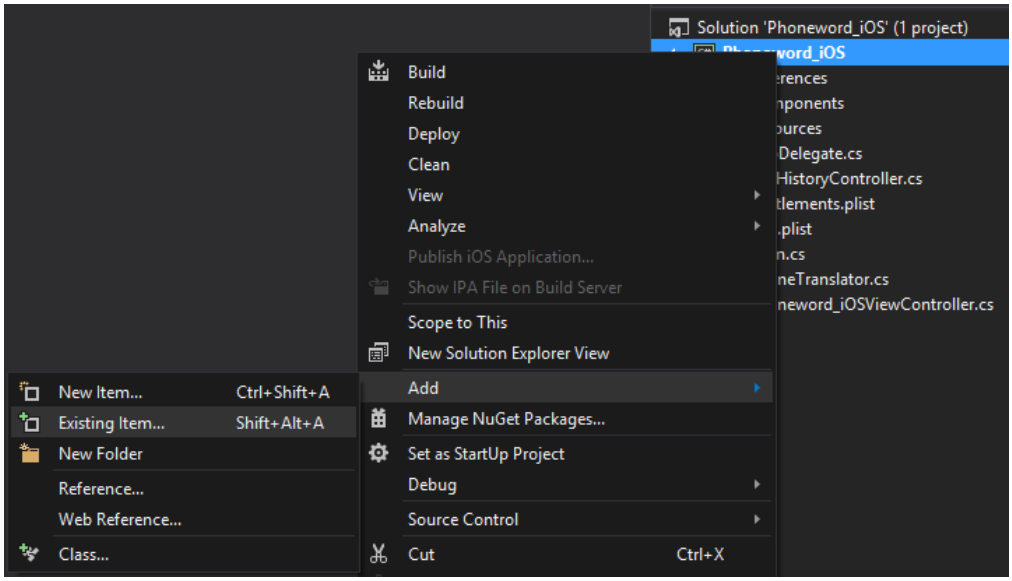
3. Click **Open Project...** on the Visual Studio Start Page and navigate to the **Lab 01 Resources** folder included with this document.
4. Locate the **Phoneword_iOS_Begin** folder and open the `Phoneword_iOS.sln` file:



Add PhoneTranslator.cs file to our project

We need to add the code to translate the alphanumeric phone number into a numeric phone number.

1. Right-click on the **Phoneword_iOS** project folder and choose **Add > Existing Files...** as shown below:



2. Navigate to the lab resources (Lab 01 Resources) that were included with this document.
3. Select `PhoneTranslator.cs` and click the **Open** button. The file will be copied to your current project directory.

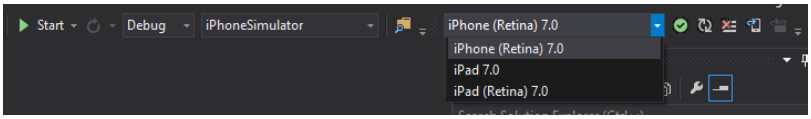
Create the User Interface

> This task has already been completed for you, however the instructor will walk through these steps and create the UI from scratch so you can see how it is built. If you would like to build the UI on your own after class, the instructions are supplied below. Once the instructor is done with the UI, you can go to the next task: Testing the Application.

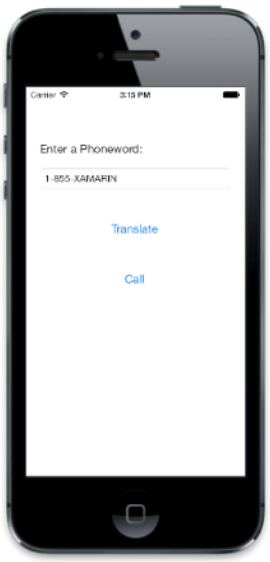
1. Let's look at how iOS UI is created in code. Double-click the task `//TODO: Step 1: uncomment to create UI to display`.
2. Uncomment this task's code
3. Each control is declared, given a location and size, and then added to the View Controller's `View` after setting any control-specific settings like `Text` or `Title`.

Testing the Application

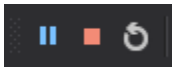
1. From the **iOS Toolbar** choose **Debug**, **iPhone Simulator**, **iPhone (Retina) 7.0** and click the **Start** button:



2. The iOS Simulator should show up after several seconds, and if everything has gone well our app will display:



3. Press the **Stop** button in Visual Studio to halt the iOS Simulator:

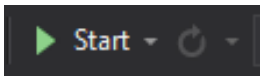


Implement the TouchUpInside Event for TranslateButton

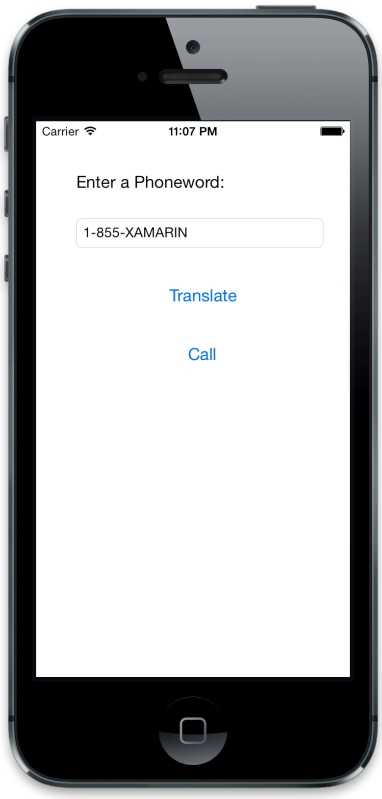
1. Let's wire up the TranslateButton. Double-click `Phoneword_iOSViewController.cs` and uncomment the section of code under the comment `//TODO: Step 1a: uncomment to wire up TranslateButton.`
2. File > Save.

Testing the Touch Event Handler

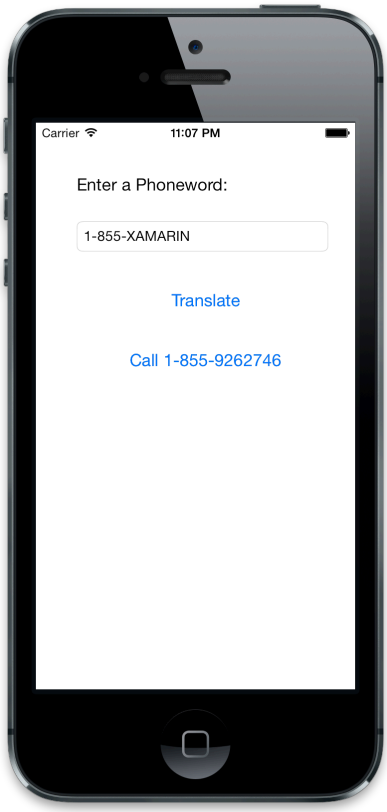
1. Our TranslateButton is now wired. Let's test it. Click the **Start** button again:



2. The iOS Simulator should show up after several seconds and if everything has gone well our app will display:



3. If there are any errors, check the error messages and compare the code to the lab instructions. It may also help to check the completed version of the lab in the `Phoneword_iOS_Completed` project.
4. Click the **Translate** button in our app and verify the **Call** button title changes to `Call 1-855-9262746`, as shown below:



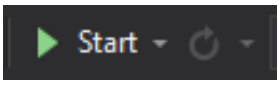
5. Press the **Stop** button in Visual Studio to halt the iOS Simulator:

Implement the TouchUpInside Event for CallButton

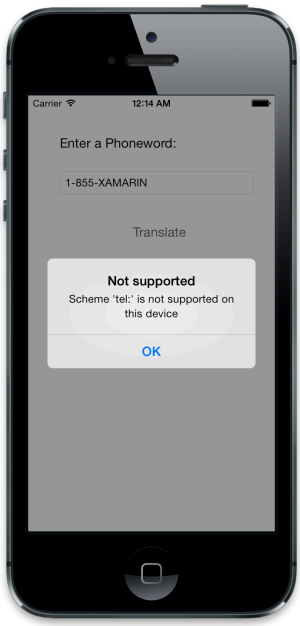
1. Let's wire up the CallButton. Double-click `Phoneword_iOSViewController.cs` and uncomment the section of code under the comment `//TODO: Step 1b: uncomment to wire up CallButton`.
2. **File > Save**.

Testing the Application

1. Click the **Start** button:



2. Click the **Translate** button to populate our CallButton with `Call 1-855-9262746`.
3. Press the **Call 1-855-9262746** button and an *Alert Dialog* should appear with a **Not Supported** message since the **iOS Simulator** can't make a call. The Alert dialog should resemble the following:



Congratulations

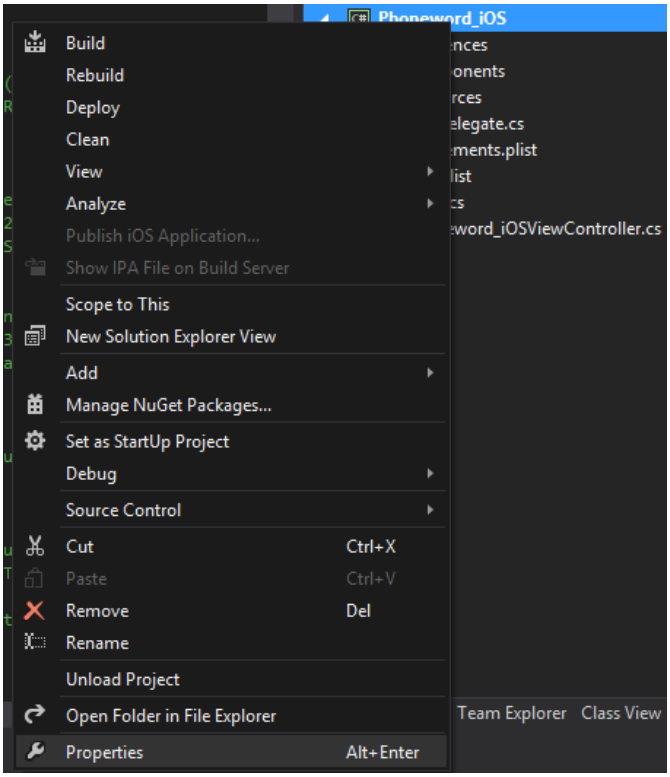
You've now completed your very first Xamarin.iOS application for the iPhone!

Finishing Touches

Application Name

1. While the **iOS Simulator** is still running, click the circular *Home Button* on the **iOS simulator** screen. Our app name defaults to the name of our project, **Phoneword_iOS**, which is a bit too long to display under the app icon. Additionally, the icon is the default app icon, as shown below:

2. In Visual Studio, right-click on the Project `Info.plist` file.

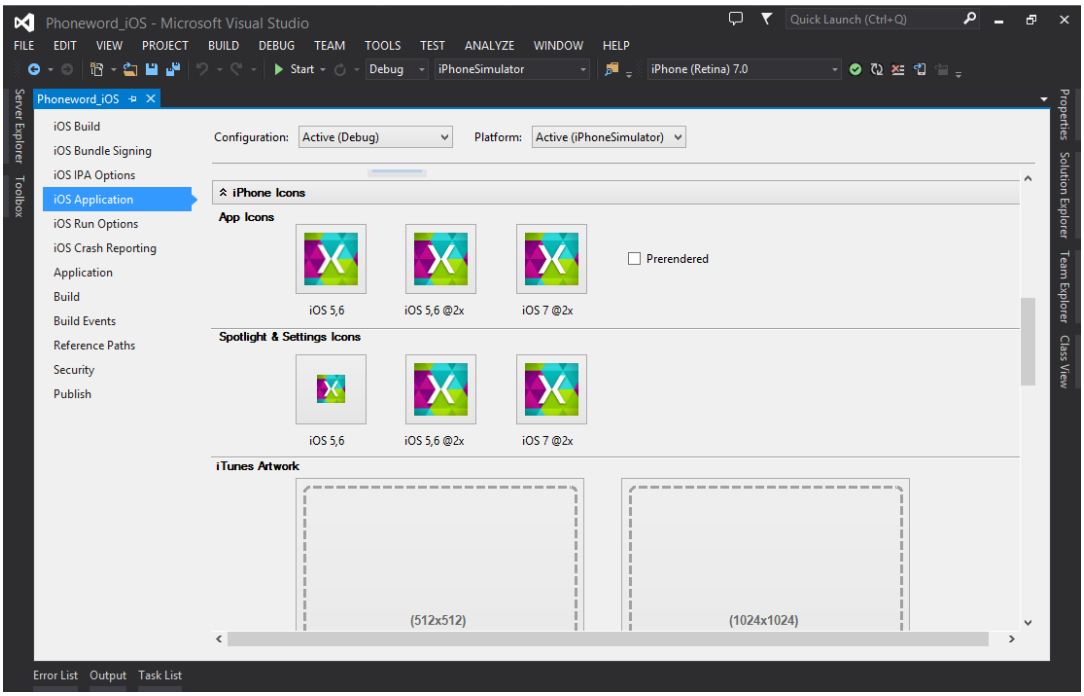


3. Under **iOS Application** change the **Application Name** to `Phoneword`
4. Click the **Start** button to rerun our app.

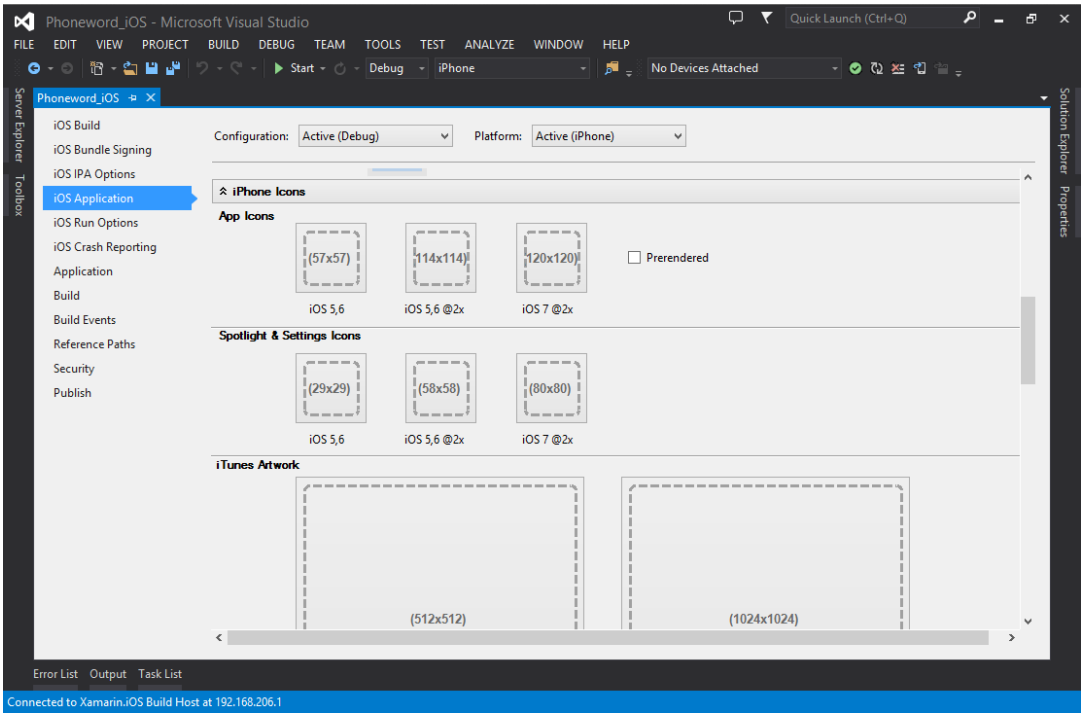
5. Verify our new app name by clicking on the **Home Button** to stop the **iOS Simulator,** so we can see the new **app name** under the icon as shown below:

Icons and Launch Images

Let's add some custom app icons and launch images to our project. Our finished project will resemble:



1. Right-click the `info.plist` file and scroll down until you can see the **iPhone icons** section as shown below:



2. In **iPhone Icons** section, single-click on the **placeholder for the 57 x 57 icon.**
3. Navigate to the **Lab 01 App Resources** and double-click on icon-57.png to add it to our Phoneword_iOS project.
4. In a similar way, double-click on each box representing an app icon and select the png file using the table below:

Icon Image Size	Import Image Name
57x57	icon-57.png (already done)
114x114	icon-114.png
120x120	icon7-120.png
29x29	icon-29.png
58x58	icon-58.png
80x80	icon7-80.png

5. Scroll down if necessary until you see the **iPhone Launch Images** section, add the launch images from the Resources folder by double-clicking on each **placeholder for the launch Images,** selecting the appropriate image for each of the following:

Launch Image Size	Import Image Name
320x480	Default.png
640x960	Default@2x.png
640x1136	Default-568h@2x.png

6. Open the Resources folder in the Solution Explorer to see our new imported app icons. The names were changed as they were imported to the correct names for our iOS app. For your reference, the following table lists how names of the app icons and launch images were changed during import:

Original Name	Name inside Project
icon-57.png	Icon.png
icon-114.png	Icon@2x.png
icon7-120.png	Icon-60@2x.png
icon-29.png	Icon-Small.png
icon-58.png	Icon-Small@2x.png
icon7-80.png	Icon-40@2x.png
Default.png	no change
Default@2x.png	no change
Default-568h@2x.png	no change

7. File > Save All.

Testing the Application

1. Click the **Start** button to run our app. As the app launches, it will briefly display the **Xamarin launch image**:

2. Click the Home button on the iOS Simulator to halt our app and we will see our custom icon:

Congratulations!

We covered a lot of ground here. You now have a solid understanding of the components of a Xamarin.iOS application, as well as the tools used to create them.

Summary

In this lab, we have built our first **Xamarin.iOS** application and learned how to use Visual Studio to do so. In the next lab, we're going to build on this application by adding another screen and introducing the Model View Controller (MVC) pattern in iOS.