Lab 02: Creating a multi-screen iOS app

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

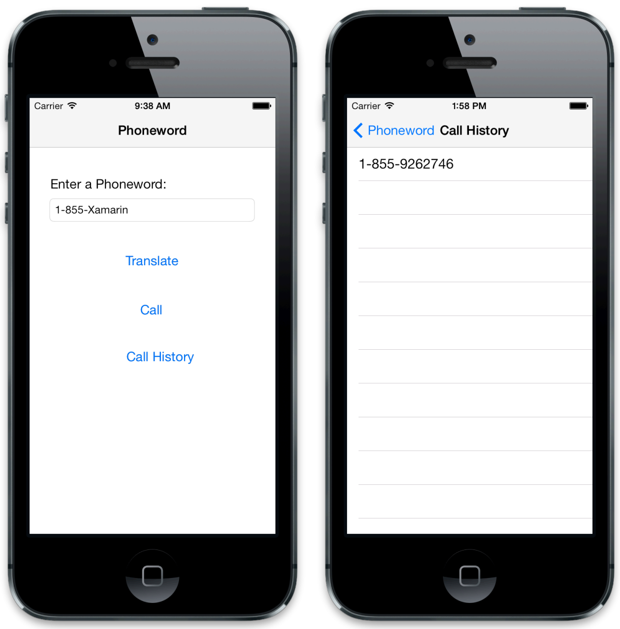
Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is Lab 01 Resources. Make sure you have this folder before you begin.

Lab Goals

In this lab, we'll explore the *Model, View, Controller (MVC)* pattern and see how it is used in iOS to create multi-screened applications.

Additionally, we'll introduce the *UINavigationController* and *UITableViewController* classes. We'll learn how to use these view controllers together to provide a familiar navigation experience in iOS.

Finally, we'll include support for navigation between multiple screens by extending the *Phoneword_iOS* application we created in Lab 1 to include a second screen that contains the call history, as illustrated below:



Steps

In this lab, we'll be extending our Phoneword_iOS application from Lab 1 by adding a *UINavigationController* to add a **Call History** screen. The `UINavigationController` provides a light-gray navigation bar at the top of our application with a **back** button for each screen except for the one at the root of the `UINavigationController`.

Open the Starting Solution

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

   Xamarin.iOS Build Host

2. Launch **Visual Studio 2013.**
3. Navigate to the following folder location in the lab environment:

   `Lab 02 resources/Phoneword_iOS_Start`

4. Double-click the **Phoneword_iOS.sln** file. This solution is similar to the final solution for Phoneword_iOS app we completed in Lab 1 with some minor additions. A `List<string> PhoneNumbers`, was added to both the **Phoneword_iOSViewController** and **CallHistoryController** (to be added later) to hold the list of phone numbers to populate our call history.

Adding a Navigation Controller

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.

A navigation controller will allow us to manage a hierarchy of view controllers. To change our app to start with a navigation controller, we need to make some changes in **AppDelegate.cs**.

1. Double-click the task `// TODO: Step 1a: uncomment to create a Navigation Controller to be the new root.`

In this beginning project, a `UINavigationController` has already been declared for you, but we still need to create an instance of one in the `FinishedLoading` method.

2. Uncomment this task's code.

```
navigationController = new UINavigationController (viewController);
```

This will create a navigation controller where the root screen is the view controller we created in Lab 01.

3. Next we need to remove call to window.RootViewController under `//TODO: Step 1b: delete or comment out the old Root View Controller assignment.`
4. To make the new navigationController the first screen seen when the app loads, we need to set it as the new window.RootViewController. This is done from the next task: `//TODO: Step 1c: uncomment to make the Navigation Controller the new Window root.`

```
window.RootViewController = navigationController;
```

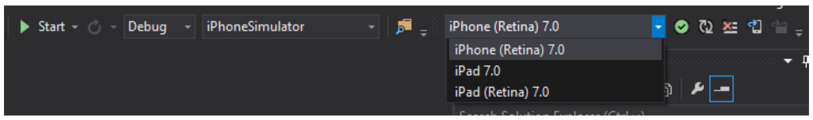
5. Uncomment the code for the next task: `//TODO: Step 1d: uncomment to give View Controller a title.`

```
this.Title = "Phoneword";
```

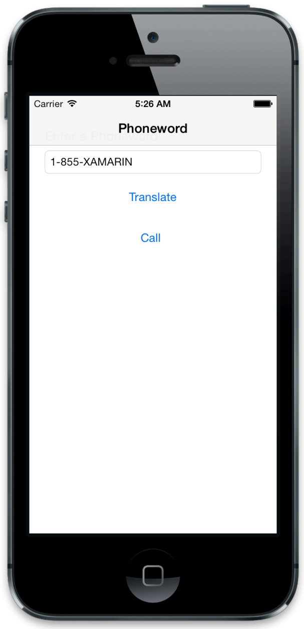
Now that our app is using a navigation controller, this will give a `Title` to our initial view controller that will be shown in the top **Navigation Bar**.

Testing the Application

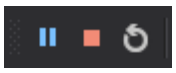
1. From the **iOS Toolbar** choose **Debug**, **iPhone Simulator**, **iPhone (Retina) 7.0** and click the **Start** button:



2. Our app will launch and the Phoneword_iOS screen will now have a **Navigation Bar** with the title **Phoneword** and should resemble the following:

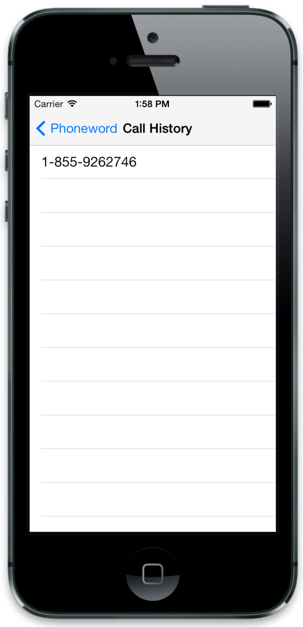


3. Press the **Stop** button in Visual Studio to halt the iOS Simulator:



Adding the Call History

We'll be extending our app by including a **Call History** button to the initial screen of our Phoneword_iOS app to display any calls the user has initiated from within our app. We will display a list of these calls in a **Table View Controller** titled **Call History** as illustrated by the screenshot below:



1. To add the **Call History** button, uncomment the code for the next task: `//TODO: Step 2: uncomment to add new Call History button to the UI.`

```
var callHistoryButton = UIButton.FromType (UIButtonType.RoundedRect);
callHistoryButton.Frame = new RectangleF(20, 346, 280, 30);
callHistoryButton.SetTitle ("Call History", UIControlState.Normal);
View.Add (callHistoryButton);
```

   With this button in place, we need to create the Table View Controller that will be presented when it is clicked.

2. Right-click the project and choose **Add > Existing Item** and grab **CallHistoryController.cs** from the lab resources.

   Uncomment the code for the next task: `//TODO: Step 3a: uncomment - list of phone numbers called for Call History screen`. This will start a `List<String>` that will receive new phone numbers as they are called.

3. In the callButton.TouchUpInside event handler, uncomment the code for `//TODO: Step 3b: uncomment to store the phone number that we're dialing in PhoneNumbersto` to add the called number to the list.

Navigating to the Call History

When the **Call History** button is clicked, we need to create a new `CallHistoryController` with the current list of phone numbers.

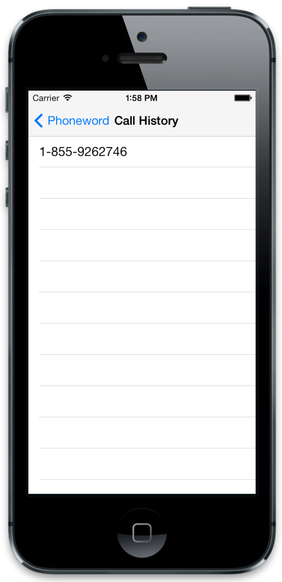
1. Wire up a TouchUpInside event handler for the new callHistoryButton by uncommenting `//TODO: Step 3c: uncomment to wire up the CallHistoryButton.`

```
callHistoryButton.TouchUpInside += (object sender, EventArgs e) => {
    CallHistoryController callHistory = new CallHistoryController();
    callHistory.PhoneNumbers = PhoneNumbers;

    this.NavigationController.PushViewController (callHistory, true);
};
```

Testing our Application

1. Click the **Start** button to build and run our app.
2. Click the **Translate** button.
3. Make a call and click the **Call History** to see the new screen animate into view:



4. Click the **Phoneword** back button to pop the `CallHistoryViewController` and return to the initial **Phoneword** screen.
5. Press the **Stop** button in Visual Studio to halt the simulation.

Summary

Congratulations! In this lab, we built a multi-screened application. We learned how to use a navigation controller to transition between view controllers.