Lab 02: Creating a multi-screen iOS app

Prerequisites

You will need a Mac with Xcode and Xamarin tools installed. 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:

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

Downloads

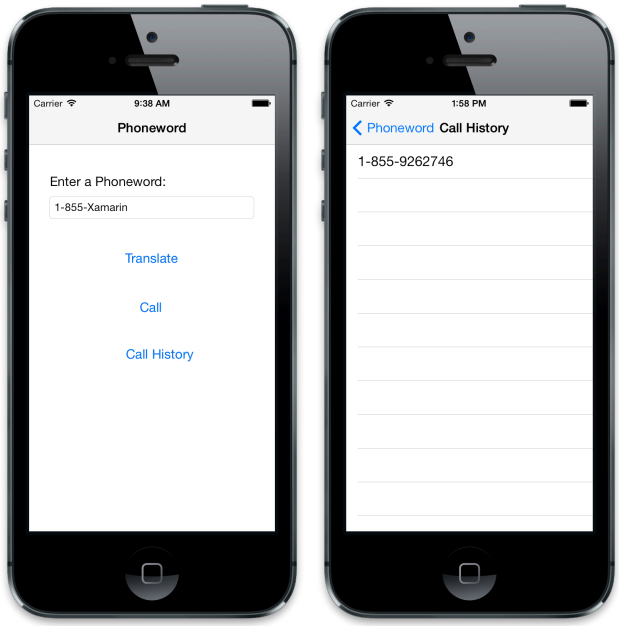
Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is Lab 02 Resources. Make sure you have this folder before you begin.

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

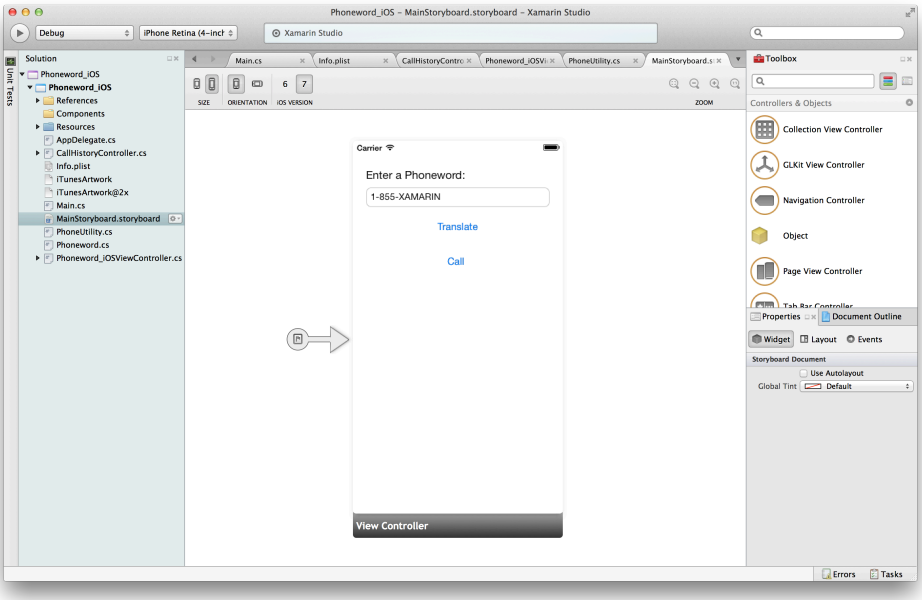
1. Launch **Xamarin Studio**.
2. Navigate to the following folder location in the lab environment:

   ```
   Lab 02 resources/Phoneword_iOS_Start
   ```

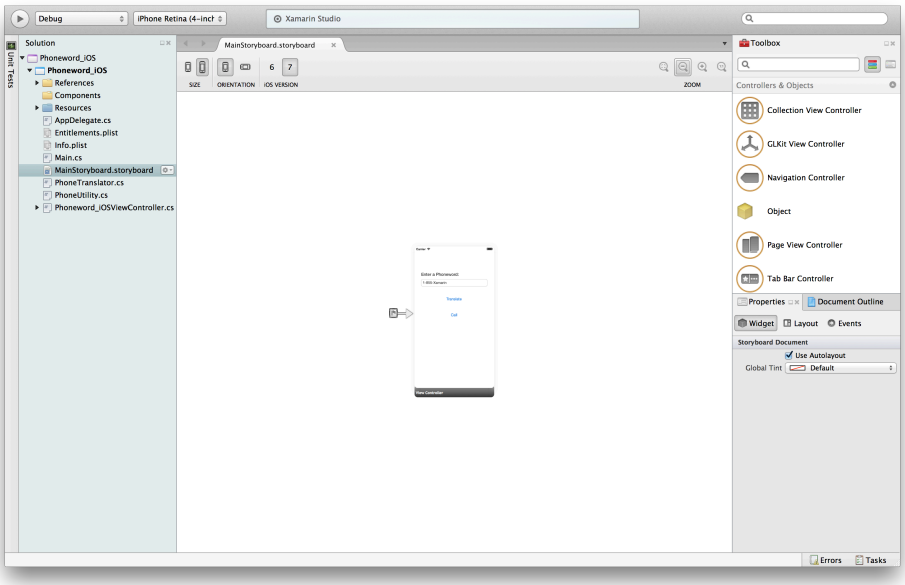
3. Double-click the **Phoneword_iOS.sln** file. This solution is similar to the final solution for Phoneword_iOS app we completed in Lab 1 with some minor additions. A `List<string> PhoneNumbers`, was added to both the **Phoneword_iOSViewController** and **CallHistoryController** (to be added later)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

Adding a Navigation Controller

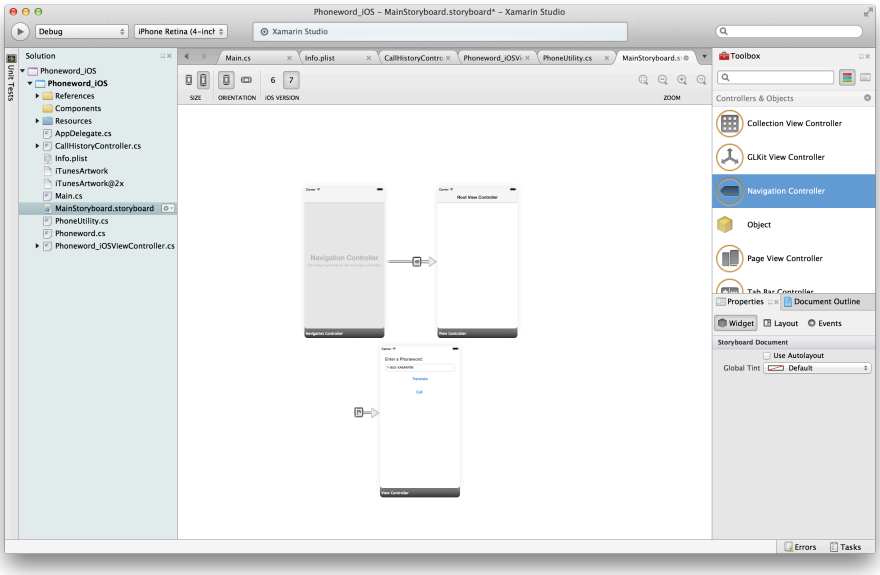
1. Double-click the **MainStoryboard.storyboard** file to display the storyboard in the *Xamarin Studio Designer*:



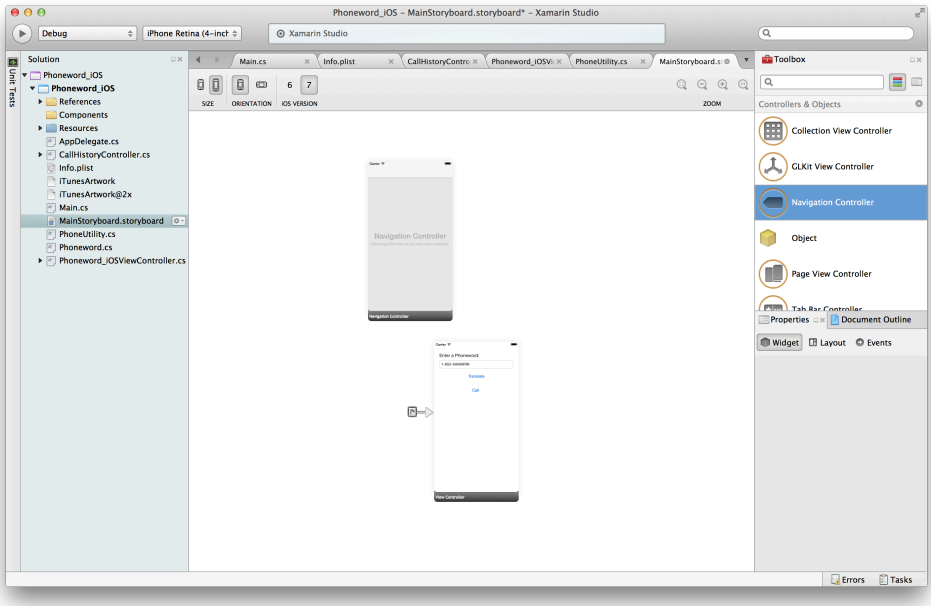
2. Use the **Zoom** tool to zoom out to make room to drag in a **Navigation Controller** as shown below:



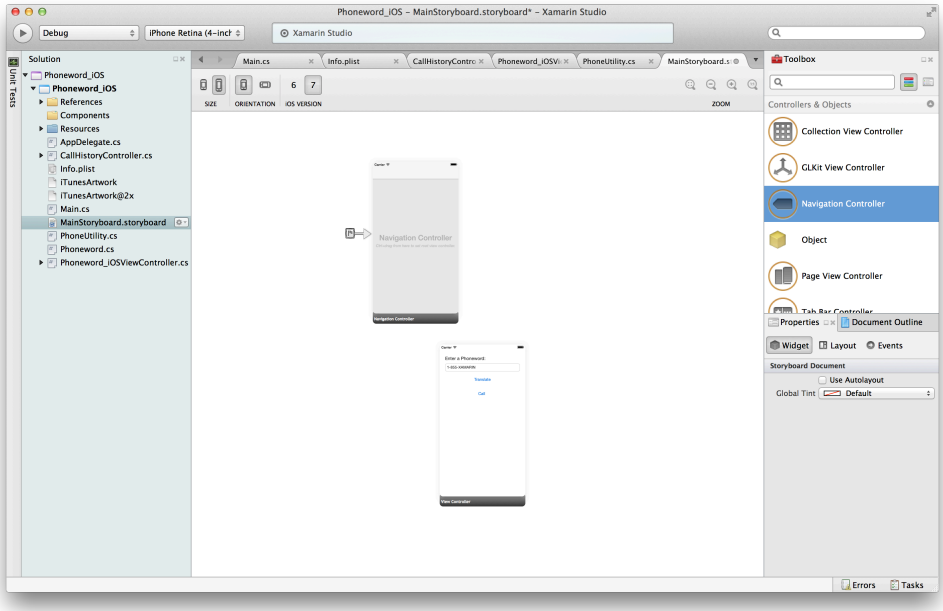
3. Drag a **Navigation Controller** from the **Toolbox** pane to the **Designer** layout pane.
4. Reposition the view controllers so they don't overlap on the screen by dragging them using the solid **black bar** located at the bottom of each view controller. The resulting storyboard should now resemble the following:



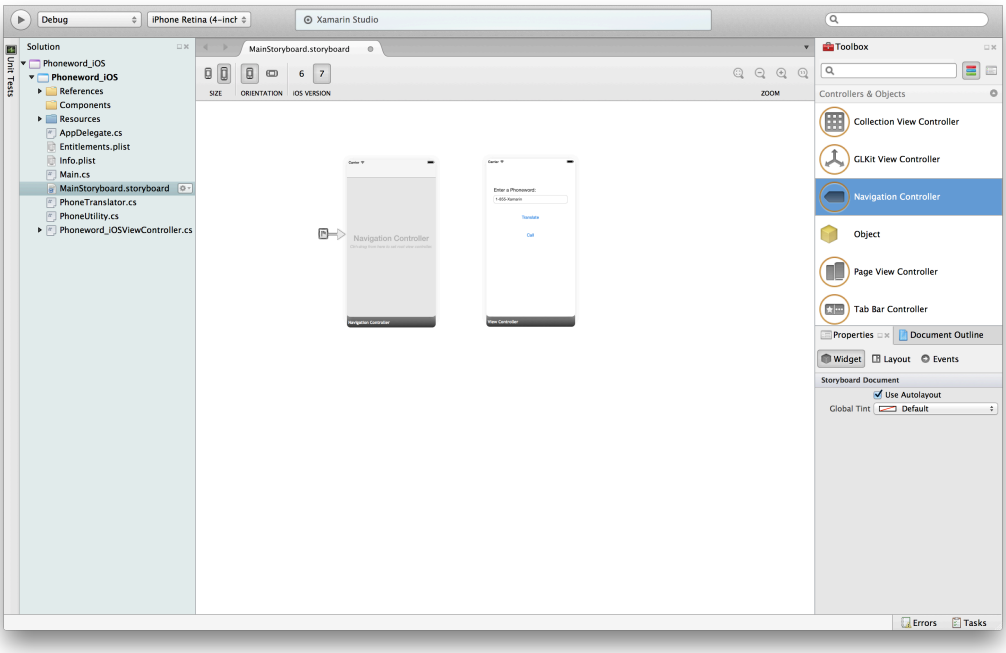
5. Delete the **Root View Controller** attached to the Navigation Controller by single-clicking on the solid **black bar** along the bottom of the view controller and pressing **Delete**:



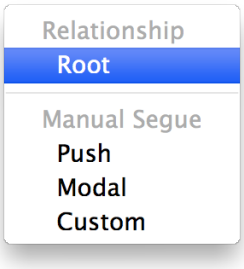
6. Drag the *sourceless segue* from the `Phoneword_iOSViewController` to the left-had side of the **Navigation Controller**, as shown below:



7. Reposition the **Phoneword_iOS** view controller so that it is in line with the **Navigation Controller** as follows:



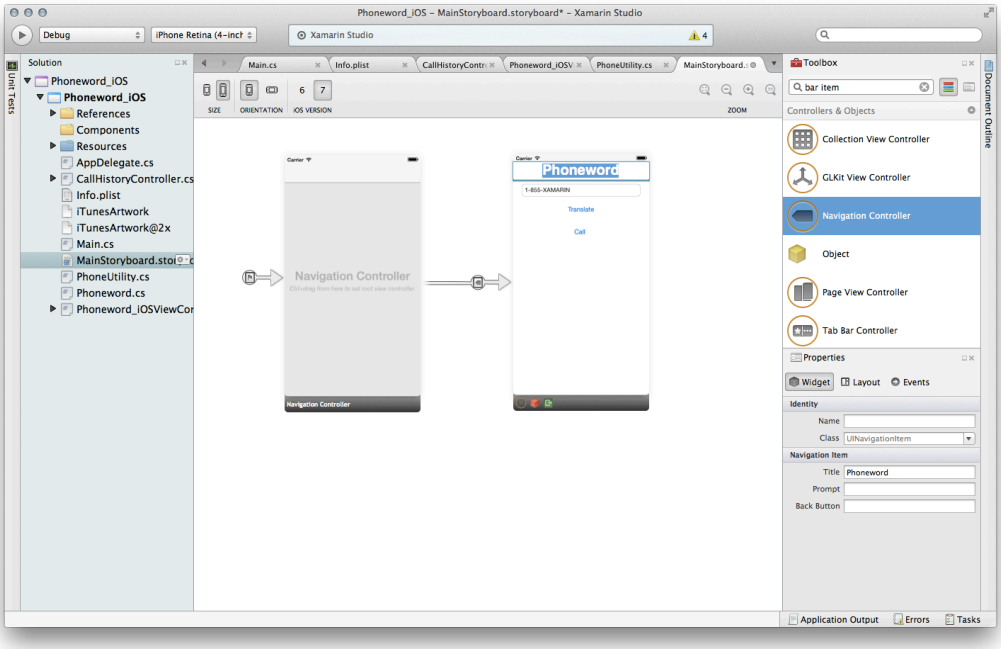
8. Control-drag by holding down the control key while clicking within the view area of the **Navigation Controller** and then dragging until you are over to the **Phoneword_iOS** view controller. A light-blue line will appear during the drag. Lift your dragging finger while it is over the **Phoneword_iOSViewController** and a **Popup Menu** will appear. Select **Relationship > Root** from the menu that appears as a result of the control-drag:



The Phoneword_iOSViewController is now setup as the Root View Controller associated with the Navigation Controller. This means it is the first view controller on the Navigation Controller's stack and the Navigation Controller will automatically add a Navigation Bar. The Navigation Controller will display the screen for the Phoneword_iOS view controller as the initial screen for our app.

Let's add a title to the **Navigation Bar** of our `Phoneword_iOSViewController`.

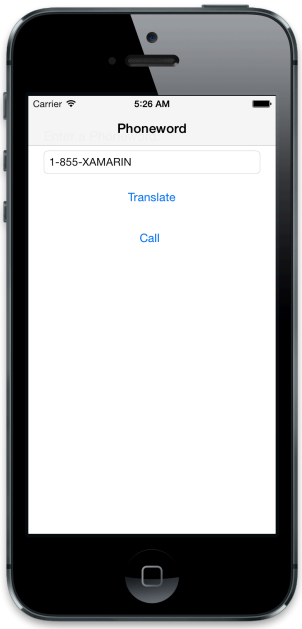
9. Double-click the **Navigation Bar** at the top of our `Phoneword_iOSViewController` and enter `Phoneword` as the title as shown:



10. Select **File > Save All**

Testing our Application

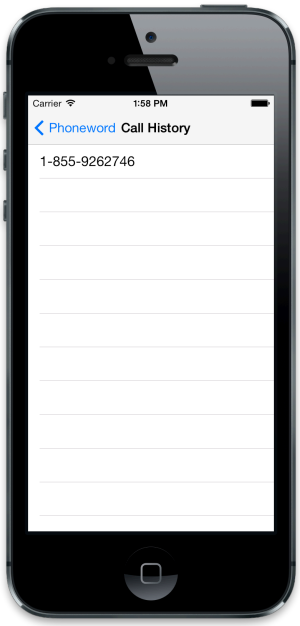
1. Click the **Play** button to build and run our app. Our app will launch and the Phoneword_iOS screen will now have a **Navigation Bar** with the title **Phoneword** and should resemble the following:



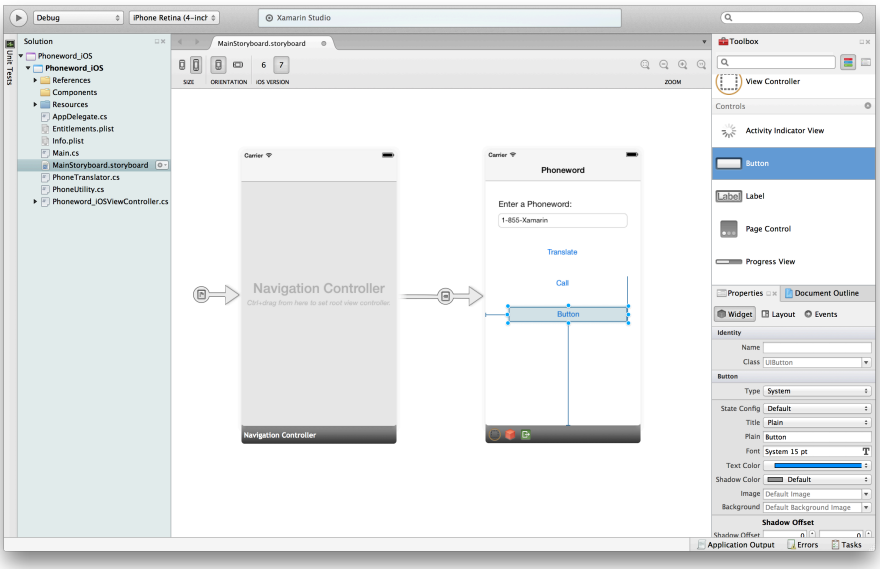
2. In Xamarin Studio, press the **Stop** button to stop the iOS simulator.

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. Drag a new button into the **Designer** and position it under the **Call** button, repositioning and resizing it as necessary to resemble the following:

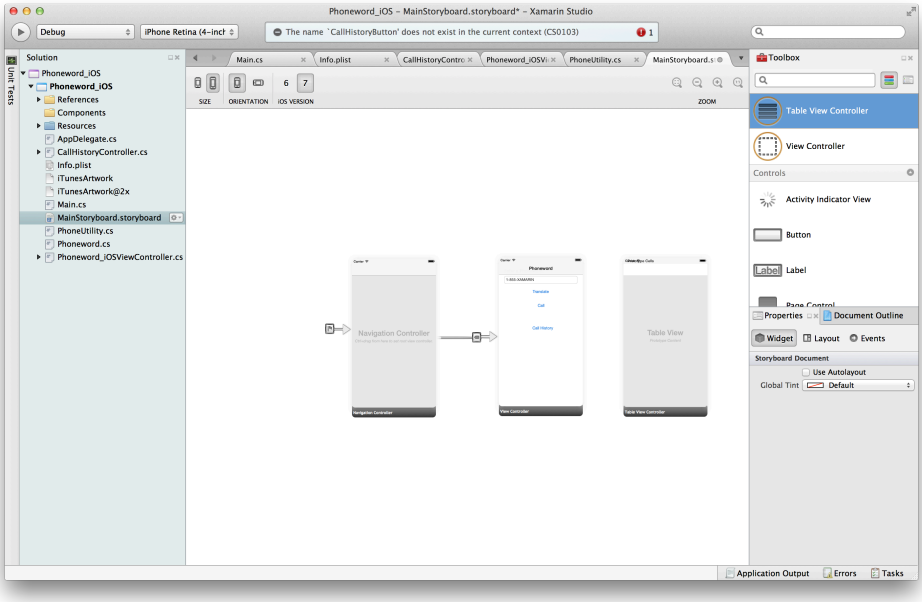


2. Select the **Properties** tab and make the following changes to the Call Button:

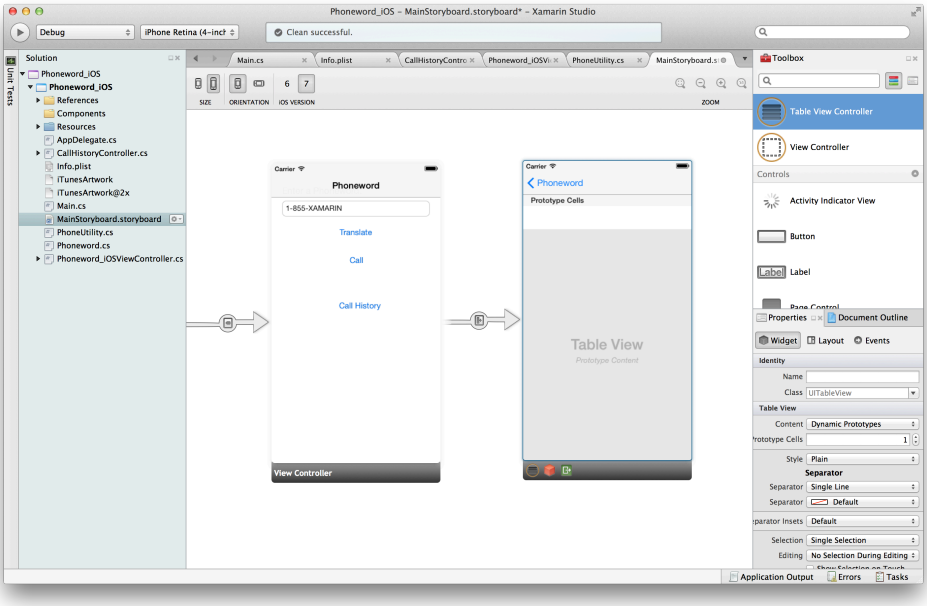
   Identity → Name = `CallHistoryButton`

Button → Plain = `Call History`

3. The Identity Name is how the control is made accessible in code.
4. Drag a **Table View Controller** into the **Designer** and position it to the right of our **Phoneword_iOS** controller:



5. Control-drag by holding down the control key while clicking on the CallHistory button and dragging to the middle of the **Table View Controller.** A **blue line** will appear as you drag. Lift your dragging finger while it is over the **Table View Controller** and a **Popup Menu** will appear. Select **Push** from the **Popup Menu**. A push **Segue** has just been created. Our **Table View Controller** now has a **Navigation Bar** and a **Back Button** called **Phoneword** created for us by the Navigation Controller as shown below:

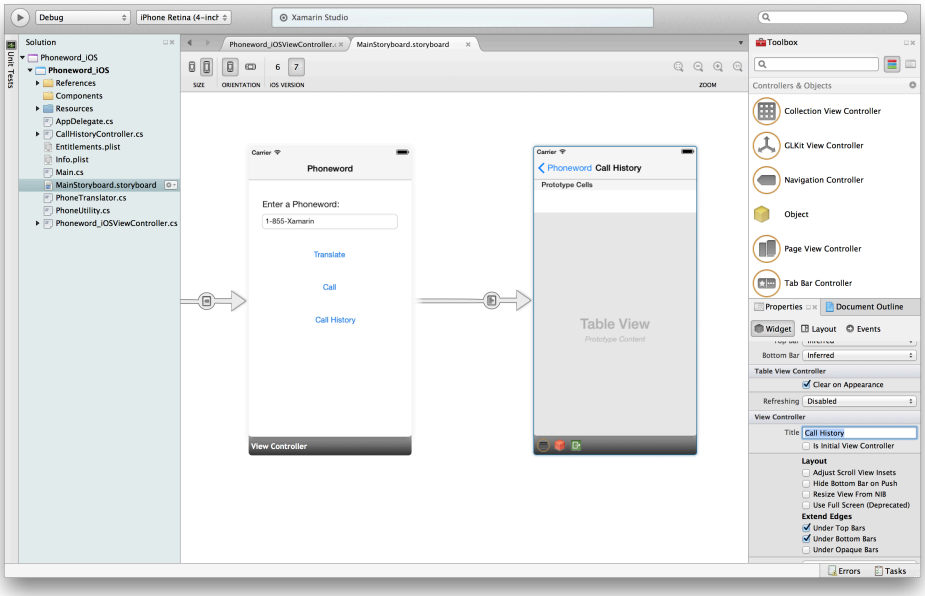


It is important to understand what's happening here. The segue makes the controller get pushed on to the Navigation Controller's stack, and the controller has a view + view hierarchy.

6. Select the **Table View Controller** by single-clicking the **Black Bar** across the bottom of the view controller.

7. Select the **Properties** tab, scroll down to the **View Controller** section and change the **Title** of the **Table View Controller** in the **Navigation Bar** as follows:

View Controller → Title = `Call History`



8. File > Save All

Testing our Application

1. Click the **Play** button to build and run our app.
2. Click the **Call History** button to push the `CallHistoryViewController` on to the navigation stack:

3. We don't expect calls to show up on this list yet because we haven't wired it up to anything yet.
4. Click the **Phoneword** back button to pop the `CallHistoryViewController` and return to the initial **Phoneword** screen.
5. Press the **Stop** button in Xamarin Studio to halt the simulation. Our next step will be to add the code to populate the **Call History** screen.

Implementing the Table View Controller

We will implement the table view controller for our call history by specifying the name of the backing class in the **Properties** tab for the table view controller in the storyboard. Xamarin Studio will automatically generate the backing class when we change the name of the `UITableViewController` class the name of our custom view controller.

1. Select the **Table View Controller** by clicking on the **Black Bar** of the **Table View Controller**. Make the following change in the **Properties** pane:

   Identity → Class = `CallHistoryController`

2. Xamarin Studio automatically creates a `CallHistoryController.cs` file when you change the class name. The `CallHistoryController` is a subclass of `UITableViewController`.
3. The controller comes populated with the following code:

```
using System;
using MonoTouch.Foundation;
using MonoTouch.UIKit;

namespace Phoneword_iOS
{
    public partial class CallHistoryController : UITableViewController
    {
        public CallHistoryController (IntPtr handle) : base (handle)
        {
        }
    }
}
```

We are now ready to insert the code for our `CallHistoryController` that will populate and control the behavior of the table view. We'll need to implement some of the methods for `UITableViewController` and also populate the table's data source from a backing data store. For simplicity, our backing store is an in-memory `List<String>`.

4. In Xamarin Studio, **File > Open...** and navigate to: `/Desktop/Xamarin Studio Labs/Intro to iOS with Xamarin Studio/Lab 02 resources/` Double-click on `Completed_CallHistoryController.cs` to open it. Copy all of this code to the `CallHistoryController.cs`, replacing all of the code currently in the file.
5. Comments have been included in this file to give you a quick idea of how the Table View Controller code works. Table View Controllers are covered in depth in the next course. The important thing to know for now is that the `CallHistoryController` also defines a List<string> with the same name as the `Phoneword_iOSViewController`. This List<string> is called `PhoneNumbers`. `PhoneNumbers` will be used to pass the list of called phone numbers from the `Phoneword_iOSViewController` to the `CallHistoryController`.
6. **File > Save**
7. Close the `Completed_CallHistorycontroller.cs` to avoid confusion.

Adding the Code to populate the Call History

We are now ready to add our implementation to this class to display the call history. The code to do this involves populating the table's data source from a backing data store, which, for simplicity, is an in-memory List<String> in this case.

We need to add a number to **PhoneNumbers** each time the **Call** button is pressed in the `Phoneword_iOSViewController`. We'll populate this list when the **Call** button is pressed, just before launching the dialer.

Navigating to the Call History

8. Let's open `Phoneword_iOSViewController.cs`. We've made a few changes in this lab, notice that we're now keeping a list of the phone numbers in a List<string> called `PhoneNumbers` and we've moved the `translatedNumber` string from `ViewDidLoad` to the controller scope.
9. Let's setup `PhoneNumbers` as an in-memory list to hold the phone numbers called by our app. Double-click `Phoneword_iOSViewController.cs` and uncomment the following line of code under the comment

```
//TODO: Step 2a: uncomment -
 list of phone numbers called for Call History screen
PhoneNumbers = new List<String> ();
```

10. We need to store the phone number we are dialing to the `PhoneNumbers` list. Uncomment the following line of code under the comment

```
//TODO: Step 2b: uncomment to store the phone number that we're dialin
g in PhoneNumbers
PhoneNumbers.Add (translatedNumber);
```

We'll use the `PrepareForSegue` method of `Phoneword_iOSViewController` to set the `PhoneNumbers` property of the `CallHistoryController` just before the transition defined by the segue occurs. The `PrepareForSegue` method is called when the segue is executed, providing an opportunity to prepare the controller that is going to be activated *before* the navigation actually occurs. This gives us the opportunity to perform any necessary initializations. In our case, we want to pass our list of called `PhoneNumbers` to the `CallHistoryController`, so they can be displayed.

11. Uncomment the following section of code under the comment `//TODO: Step 2c: uncomment to implement the PrepareForSegue` method by selecting the lines of code and pressing cmd+/:

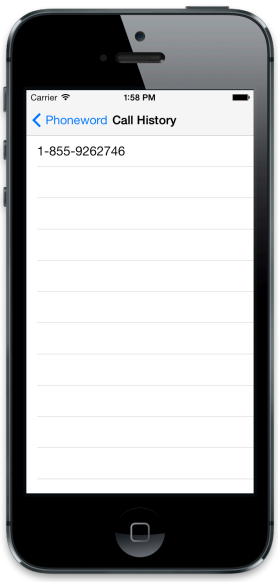
```
//TODO: Step 2c: uncomment to implement the PrepareForSegue method
public override void PrepareForSegue (UIStoryboardSegue segue,
NSObject sender)
{
    base.PrepareForSegue (segue, sender);

    var callHistoryContoller = segue.DestinationViewController
                                as CallHistoryController;

    if (callHistoryContoller != null) {
        callHistoryContoller.PhoneNumbers = PhoneNumbers;
    }
}
```

The segue doesn't "know" that the `DestinationViewController` is a `CallHistoryController`, so we have to cast it to get a strongly typed reference to it.

12. File > Save All.

Testing our Application

1. Click the **Play** button to build and run our app.
2. Click the **Call History** button. No numbers should appear in the **Call History** table.
3. Click the **Phoneword** back button to return to the initial **Phoneword** screen.
4. Click the **Translate** button, followed by the **Call 855-9262746** button.
5. Click the **OK** button to close the **Alert Dialog**.
6. Click the **Call History** button. We see that navigating to the call history after dialing shows the list of numbers dialed:



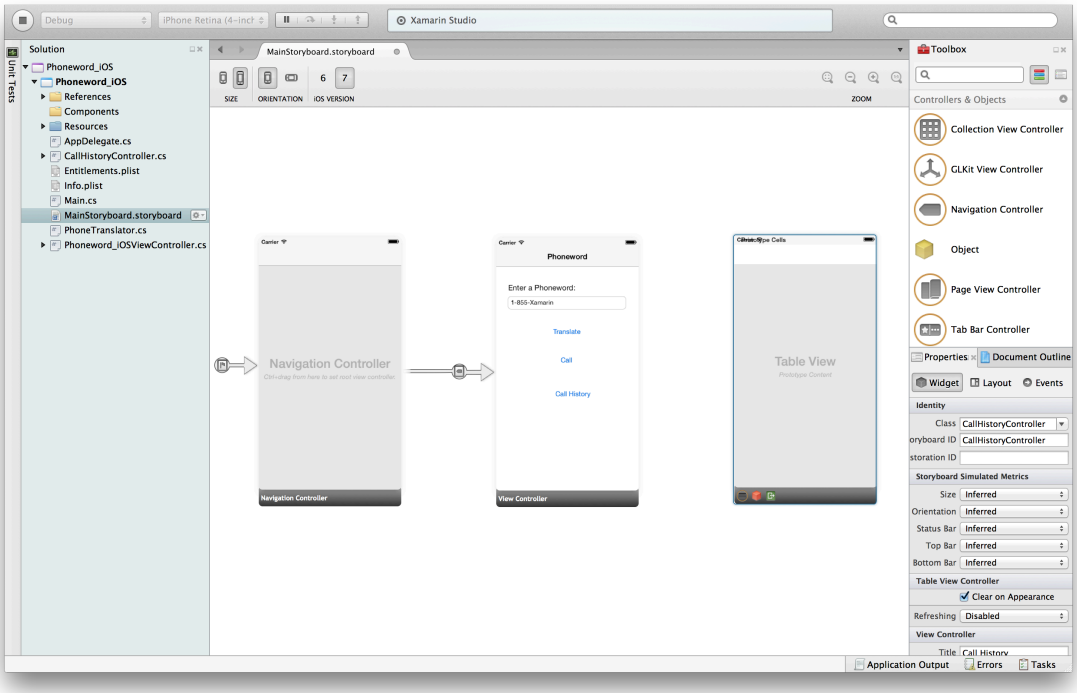
Showing Screens Programmatically with the PushViewController method

Segues are a great way to create transitions between view controllers in a storyboard, but can only be used for known fixed paths. Often times you need to load controllers from places where you can't create a segue. For instance, imagine that we are creating an application that launches different screens from the same button, depending on the current state of an application. In this case, we need to load the view controller programmatically.

The navigation controller exposes a method called *PushViewController* that allows us to do just that. We simply instantiate the view controller we want to push onto the navigation controller and then call the method.

Let's modify our PhoneWord_iOS application again to launch our `CallHistoryController` programmatically, rather than with a segue:

1. Open **MainStoryboard.storyboard.**
2. Select the **Push Segue** connected to the `CallHistoryController` by single-clicking it and press Delete. Since the `CallHistoryController` is no longer connected to the `UINavigationController` the **Navigation Bar** will disappear and our storyboard will look like this:



The PrepareForSegue method we uncommented earlier is no longer being called since the Phoneword_iOSViewController no longer has an outgoing segue. We will ignore this code for the purpose of our lab.

Since we have deleted the segue we must instantiate the `CallHistoryController` programmatically. We'll accomplish this by wiring up the `CallHistoryButton`, instantiating a new `CallHistoryController`, passing in `PhoneNumbers`, and pushing the `CallHistoryController` onto the Navigation Controller's stack.

3. Let's wire up the `CallHistoryButton`. Uncomment the following section code under the comment `//TODO: Step 2d: uncomment to wire up the CallHistoryButton` by selecting the lines of code and pressing cmd+/:

```
//TODO: Step 2d: uncomment to wire up the CallHistoryButton

// Launches a new instance of CallHistoryController
CallHistoryButton.TouchUpInside += (object sender, EventArgs e) => {
    CallHistoryController callHistory = this.Storyboard.InstantiateVie
wController ("CallHistoryController") as CallHistoryController;
    if (callHistory != null) {
        callHistory.PhoneNumbers = PhoneNumbers;
    }
    this.NavigationController.PushViewController (callHistory, true);
};
```

4. File > Save All

We do two things in response to the tapping of the **Call History Button**. First, we create a new instance of our **CallHistoryController** via the current storyboard's `InstantiateViewController` method; passing the Storyboard ID we defined above.

Second, we call `PushViewController` on the current controller's `NavigationController` object. All view controllers have a `NavigationController` property. This property won't be `null` if the view controller has been added to a navigation controller.

`PushViewController` takes two parameters: first, the controller to push, and second, a Boolean value that specifies whether or not the controller should animate onto screen. Usually we want to pass `true`, unless we're pushing the root controller onto it, in which case we would just want it to be there when the app launches.

Testing our Application

1. Click the **Play** button and try tapping the **Call History** button. It will crash. Why?

   Our CallHistoryController code references a **Storyboard ID** to know which view controller in the Storyboard to instantiate programmatically when the **CallHistoryButton** is pressed as shown below:

```
CallHistoryController callHistory = this.Storyboard.InstantiateViewCon
troller ("CallHistoryController") as CallHistoryController;
```

Since we did not set the Storyboard ID, the method fails. Let's fix it.

2. Select the **Table View Controller** by clicking on the **Black Bar** of the **Table View Controller**. Make the following change in the **Properties** tab:

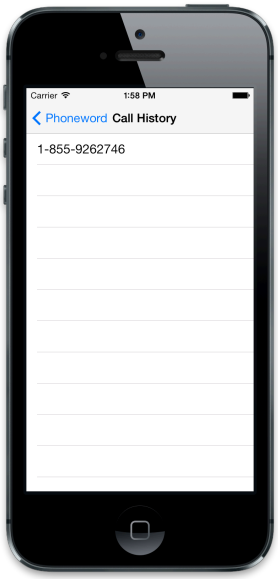
   Identity → Storyboard ID = `CallHistoryController`

   The **Storyboard ID** is used to reference the `CallHIstoryController` in the code below so we can instantiate it when the **CallHistoryButton** is pressed.

3. **File > Save**

Testing our Application

Let's try the application again, go ahead and run it, make a call, and click **Call History**. We should see the Call History screen animate on just as before. Remember, we need to use the app to make a call so the call history is populated.



Summary

Congratulations! In this lab, we built a multi-screened application. We learned how to transition to another screen programmatically and by using segues.