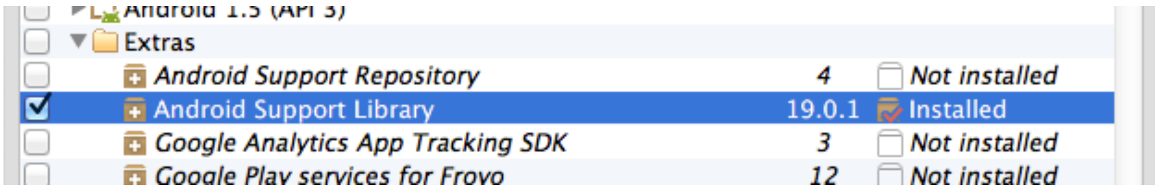
Lab 01: Stack Navigation

Prerequisites

This lab can be done with iOS, Android, or both. For Android, you will need a development environment for Android setup; either a Mac or PC with Xamarin Studio or a Windows PC with Visual Studio and the Xamarin tools installed. For iOS, remember that if you are using Windows, you will still need an accompanying Mac on your network with XCode and the Xamarin tools to build and run the code.

Many of the Android projects require Android Support Libraries (v4 and v7 AppCompat). If you do not already have them, they can be installed from the Android SDK Manager (`Tools->Open Android SDK Manager...` from the Xamarin Studio menu). Scroll down to the `Extras` and choose to install the `Android Support Library` item.



☐	Android 1.5 (API 3)		
☐	Extras		
☐	Android Support Repository	4	Not installed
✓	Android Support Library	19.0.1	Installed
☐	Google Analytics App Tracking SDK	3	Not installed
☐	Google Play services for Froyo	12	Not installed

See the `Xamarin.Android` setup documentation if you need help getting your Android environment setup:

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

See the `Xamarin.iOS` setup documentation if you need help getting your iOS environment setup:

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

Downloads

There are several projects in a single solution that we will use to explore the various mobile navigation patterns on iOS and Android. You can download the projects from the Xamarin University website:

http://university.xamarin.com/

Lab Goals

The goal of this lab will be to get an understanding of how iOS and Android handle navigation stacks. By completing this lab, you will learn about:

- The iOS and Android stack history – how items are added to the back stack on either platform.
- How to create a stack, or hierarchy, navigation system on iOS or Android (or both).

Steps

Open the Starting Solution

1. Launch Xamarin Studio and open MobileNavigationPatterns solution file included in your lab resources.
2. Depending on the platform you want to use, make sure either AndroidStack or iOSStack is the startup project.
3. Proceed to the next step for iOS or skip to the next section for Android.

iOS Stack Navigation

Creating the stack UI

1. Within the iOSStack project, open AppDelegate.cs.
2. To present a stack navigation system on iOS, you can use the provided UINavigationController class. We create one of those for a class-level field in `FinishedLaunching`.

```
UINavigationController evolveNavigationController;

public override bool FinishedLaunching (UIApplication app, NSDictionar
y options)
{
    // ...
    evolveNavigationController = new UINavigationController ();
    // ...
}
```

3. Navigation controllers need a root view controller to start the stack. In this case we will push on a MenuTableViewController with `PushViewController`. Alternatively, you could pass it in to a UINavigationController constructor overload.

```
evolveNavigationController.PushViewController (new MenuTableViewContro
ller (), false);
```

4. Then, as normal, we set our controller to be the RootViewController for the window.

```
window.RootViewController = evolveNavigationController;
```

5. Run the application in the simulator to see the styling that is automatically provided by the use of UINavigationController, then return to Xamarin Studio. Feel free to leave the app running in the simulator.

Adding to the back stack

The speaker and session lists are basic UITableViewControllers with simple backing UITableViewSources.

6. To see where we actually add to the back stack, open SpeakersTableSource.cs.
7. Inside `RowSelected`, we grab the data for the selected speaker from the data set.

```
var speaker = data [indexPath.Row];
```

8. Then we create a SpeakerViewController to display the details and push it onto the navigation stack. We access the current navigation controller in use, if any, from a controller's `NavigationController` property.

```
controller.NavigationController.PushViewController (new SpeakerViewCon
troller (speaker), true);
```

9. Return to the application in the simulator and click the Speakers item. This will animate in a new controller with the speaker list. The navigation bar will now have a back button that can return you to the root list. Return to Xamarin Studio. Feel free to leave the app running in the simulator.

Popping off the back stack

10. To pop back off the stack, UINavigationController provides a back button in the navigation bar.
11. In iOS 7, users can also use a swipe from the left-edge to trigger a back navigation as well.
12. Return to the application in the simulator and click the back button to return to the root menu.
13. [iOS 7 only] Click through to another item. This time swipe from the left edge of the screen to see the use of the back gesture popping you back to the root menu.

Android Stack Navigation

1. Within the AndroidStack project, open MenuActivity.cs.
2. Stack navigation is a pervasive concept in Android; any time you move to a new Activity, by default, you are adding to the back stack. This is a simple ListActivity, getting its data from a string array used in an ArrayAdapter.

```
items = new string[] { "Sessions", "Speakers", "About" };
ListAdapter = new ArrayAdapter<String>(this, Android.Resource.Layout.S
impleListItem1, items);
```

3. By overriding OnListItemClick, we can present the appropriate next activity based on the item clicked.

```
var intent = new Intent(this, typeof(SessionsActivity));

if (position == 1)
    intent = new Intent (this, typeof(SpeakersActivity));
else if (position == 2)
    intent = new Intent (this, typeof(AboutActivity));
```

```
StartActivity(intent);
```

4. Run the application in the emulator and click into **Sessions** and then click into any session's detail. This list is now the top activity on the stack, with the session list behind it and the root at the last item in the stack.
5. To pop it from the back stack, simply hit the Android-provided back button. You will end up back at the root menu screen first launched by this app. This functionality is extremely common to Android apps, though other actions can be added to the Android back stack.
6. Return to Xamarin Studio. Feel free to leave the app running in the emulator.

Master/Detail system

It is important to consider different device form factors when designing your apps. One way you can take advantage of larger screens when presenting these sorts of master/detail data sets is by using Fragments and an alternative layout.

7. Open **Resources/layout/speakers_screen.axml** to see the Android designer with an alternate layout available. The alternate layouts are pulled form the other layout folders in your **Resources** directory, in this case an identically-named .axml file in **Resources/layout-land**. The "-land" suffix means these layouts will be used when a device is in landscape mode.

Note: In this lab, we are using "-land" so that only one emulator is needed. For tablets, "-large" is likely a better choice. There are a number of layout configurations that can be targeted by alternative layouts. To learn more, visit Supporting Multiple Screens on the Android developer portal

8. Return to the app running in the emulator.
9. From the root menu of the app, click into **Speakers** while still in portrait mode.
10. Click through to any speaker's details to see the portrait process. This details screen has been added to the stack in this case.
11. Press the **Back** button to return to the speaker list.
12. Now, switch the emulator to landscape orientation.

Tip: to change the orientation of an Android emulator device, hit Left Ctrl+F12. This key combination will also return it to portrait.

13. This will cause Android to recreate your activity and load the layout-land view, splitting the screen into two parts: list and current details. It will also automatically select the first speaker. Because we are using Fragments, we encapsulate the list and details views in a way that they can be reused for multiple layouts like this.

For more information about Android Fragments, read the Xamarin docs about getting started with Fragments.

14. Select a new speaker from the list. Now that the details are always visible, this will simply change the details shown without adding to the back stack.
15. The logic behind this dual-layout system is in our list fragment. Open `SpeakerListFragment.cs`.
16. In `OnActivityCreated`, we use the availability of a details pane to indicate we are in dual-pane mode.

```
var detailsFrame = Activity.FindViewById<View>(Resource.Id.details);
_isDualPane = detailsFrame != null && detailsFrame.Visibility == ViewS
tates.Visible;
```

17. When a list item is clicked (`OnListItemClick`), we load the details based on `_isDualPane` in `ShowDetails`.
18. If we have a details pane, we check the current item and swap in a fragment with the selected details.

```
ListView.SetItemChecked(speakerId, true);

// Check what fragment is shown, replace if needed.
var details = FragmentManager.FindFragmentById(Resource.Id.details) as
 SpeakerDetailsFragment;
if (details == null || details.ShownSpeakerIndex != speakerId)
{
    // Make new fragment to show this selection.
    details = SpeakerDetailsFragment.NewInstance(speakerId);

    // Execute a transaction, replacing any existing
    // fragment with this one inside the frame.
    var ft = FragmentManager.BeginTransaction();
    ft.Replace(Resource.Id.details, details);
    ft.SetTransition(FragmentTransaction.TransitFragmentFade);
    ft.Commit();
}
```

19. If we do not have the details pane available, we fall back on the standard `StartActivity` approach that will push it onto the stack.

```
var intent = new Intent();

intent.SetClass(Activity, typeof (SpeakerDetailsActivity));
intent.PutExtra("current_speaker_id", speakerId);
StartActivity(intent);
```

Summary

In this lab, we saw how iOS and Android offer stack navigation solutions for applications. We reviewed how items are added and popped off the stack. For Android, we also saw a method for taking advantage of large-screen devices with alternative layouts that can keep the user from needing to build up the back stack.