# Mobile Navigation Patterns

Evolve Fundamentals Track, Chapter 7

Xamarin Inc.

# Overview

This tutorial introduces some different user interface navigation metaphors for mobile platforms, and shows how to implement them using Xamarin.

Because the different mobile platforms (such as iOS, Android and Windows Phone) have similar goals and limitations they share some common navigation elements. In addition, each platform has its own signature style and popular patterns that are driven either by the vendor or by 3rd party apps.

The code for this chapter is already complete – there are no exercises to do. There are seven sample projects included in the solution:

- AndroidStack – Hierarchical navigation in Android.
- AndroidTabs – Tabs in Android.
- AndroidOptions – Building an Options in Android.
- ActionBarSherlock & ActionBarSherlock demo – Binding that provides a backwards-compatible implementation of the Android ActionBar.
- iOSStack – Hierarchical navigation with UINavigationController.
- iOSTabs – Tabbed interface for iOS.
- iOSFlyout – Adding a Flyout Navigation Component to an iOS application.

These projects each contain a basic working example of the relevant user interface pattern.

# iOS

iOS set the benchmark for mobile user interface navigation patterns when it was released (as iPhoneOS) in 2007. Apple introduced a complete platform with its own, new user-interface metaphors that were different from their OS X platform and also from what was available on other mobile devices at the time. Two of those navigation patterns – the stack and tabs – almost completely dominated the built-in apps and the first wave of App Store apps. Since the initial launch, changes in the way people use their phones, along with competition and ideas from other platforms, has resulted in new UI idioms appearing on the iOS platform, such as the Flyout Navigation Menu popularized by the Facebook app.

## Stack (Hierarchy)

One of the main navigation mechanisms in iOS is the “stack” or hierarchy of screens, where the user can drill-down through screens by making a selection on each screen. A context-sensitive “back” button is provided to help them navigate up to previous choices, and the current screen title provides context for the data they can see.

This navigation pattern can be used for many different things, including a hierarchical menu structure (such as in the Mail and Music apps) or to navigate a tree of data. The iOSStack example screenshots below show how some navigation steps might look (a selection on each screen leads to the next screen):


iOS provides a purpose-built class – `UINavigationController` – to enable this type of navigation. Once you have created an instance of a `UINavigationController` you must provide it with the first view to display. Within that view, subsequent user-selections simply access the NavigationController property to "push" the new view onto the display.

### Example: iOSStack

The following steps were taken to implement the iOSStack example:

1) Create a UINavigationController and set the MenuTableViewController (which has already been implemented as a simple table view) as the first screen to display in the `AppDelegate FinishedLaunching` method:

```
//TODO: DemoStack: we create a UINavigationController to be the 'root' of our app
evolveNavigationController = new UINavigationController ();
//TODO: DemoStack: then push our first table as the first view that's shown
evolveNavigationController.PushViewController (new MenuTableViewController (), false);
```

2) In the menu's RowSelected method the code "pushes" a different view controller to the screen depending on which menu item was selected from the list:

```
if (indexPath.Row == 0)
    controller.NavigationController.PushViewController (new SessionsViewController (), true);
else if (indexPath.Row == 1)
    controller.NavigationController.PushViewController (new SpeakersViewController (), true);
else
    controller.NavigationController.PushViewController (new AboutViewController (), true);
```

3) In subsequent view controllers (such as the SpeakersViewController) the associated table view data source uses the selected object (in this case a Speaker) to create the view controller to "push"

```
var speaker = data [indexPath.Row];
controller.NavigationController.PushViewController (new
SpeakerViewController (speaker), true);
```

Although this navigation pattern is often implemented with table views in iOS, you are not limited to that type of view. Most view controllers can be contained within a UINavigationController and can then contain instructions to push new view controllers to the screen.

# Tabs

The other common iOS navigation control is the tab bar. It lets users choose between different functions in an application, while still being able see some context about what data they are viewing (because the tab stays highlighted).

The sample looks like this with each different tab selected:


Within each tab we can still host a UINavigationController, so that we can present different hierarchies of information and let the user skip between them via the tab bar.

### Example: iOSTabs

The following steps were taken to implement the iOSTabs example:

1) The first step is to create a subclass of UITabBarController (in the Navigation folder) so we can add our tab choices. The only customization required is in the `ViewDidLoad` method:

```
public override void ViewDidLoad ()
{
    base.ViewDidLoad ();
    var vc1 = new UINavigationController ();
    vc1.PushViewController (new SessionsViewController (), false);
    var vc2 = new UINavigationController ();
    vc2.PushViewController (new SpeakersViewController(), false);
```

```
    var vc3 = new AboutViewController();
    // add them to the tab bar controller
    var vcs = new UIViewController[] {vc1, vc2, vc3};
    ViewControllers = vcs;
    // now set the tab displays for each one (text and image)
    vc1.TabBarItem = new UITabBarItem ("Sessions",
UIImage.FromBundle("images/tabsession"), 0);
    vc2.TabBarItem = new UITabBarItem ("Speakers",
UIImage.FromBundle("images/tabspeaker"), 1);
    vc3.TabBarItem = new UITabBarItem ("About",
UIImage.FromBundle("images/tababout"), 2);
}
```

2) Create the UITabBarController subclass set it as the first screen to display in the AppDelegate

```
evolveTabController = new EvolveTabBarController ();
```

The view controllers that appear for each tab don't need to know about the tab controller that is displaying them.

### Tab Formatting

In addition to providing custom text and images for a tab, you can use system-provided icons in the `UITabBarSystemItem` enumeration. Options include Bookmarks, Downloads, Search and other common functions.

### Red Badge Indicator

iOS offers an additional feature on the tab bar, where you can make a red "badge" appear on a tab. This is used to indicate new items are available to read. You can make the badge appear by setting the BadgeValue property as shown:

```
vc1.TabBarItem.BadgeValue = "1";  // badge will appear automatically
```

and to clear the badge

```
vc1.TabBarItem.BadgeValue = null; // badge will disappear
```

### Too Many Tabs

Only 5 tabs can appear on an iPhone screen (or 9 on an iPad). More tabs can be added, but the fifth tab will automatically be replaced by a "More" tab which will display the remaining options in a table view for the user to select.

### Changing the Tab Order

It is possible to let the user change the position of the tabs by setting the `CustomizableViewControllers` property of the tab bar controller, and then writing code to detect the changes and save them.

## Flyout

A more recent addition to the iOS platform is the Flyout Navigation Menu that was popularized by the Facebook app. Other popular apps like the Gmail and Google Maps app have implemented a similar style of menu.

One key advantage of this user interface is that it frees up most of the screen for content – the user only sees the menu when they swipe to reveal it. The other

benefit is that the menu can be quite long (compared to only showing 5 tabs) and also nicely formatted.

The iOSFlyout sample looks like this (with the menu revealed, and when different content screens are visible):


Example: iOSFlyout

The example project uses the FlyoutNavigation component from the Xamarin Component Store at http://components.xamarin.com or available from within the IDE. The example already has the component added to the project. For more information on how to add components, refer to the Social + Components chapter.

1) The first step is to create a subclass of FlyoutNavigationController (in the Navigation folder) so we can add our menu choices. The only customization required is in the constructor:

```
public EvolveFlyoutNavigationController () : base()
{
    var vc1 = new UINavigationController ();
    vc1.PushViewController (new SessionsViewController (), false);
    var vc2 = new UINavigationController ();
    vc2.PushViewController (new SpeakersViewController(), false);
    var vc3 = new UINavigationController();
    vc3.PushViewController (new AboutViewController(), false);

    // Create the navigation menu
    NavigationRoot = new RootElement ("Navigation") {
        new Section () {
            new StyledStringElement ("Sessions")     { BackgroundColor =
UIColor.Clear, TextColor = UIColor.LightGray },
            new StyledStringElement ("Speakers")     { BackgroundColor =
UIColor.Clear, TextColor = UIColor.LightGray },
            new StyledStringElement ("About Evolve"){ BackgroundColor =
UIColor.Clear, TextColor = UIColor.LightGray },
        }
    };
    // Supply view controllers corresponding to menu items:
```

```
        ViewControllers = new UIViewController[] {
            vc1, vc2, vc3
        };
        NavigationTableView.BackgroundView = new UIImageView
    (UIImage.FromBundle ("images/Background-Party.png"));
        NavigationTableView.SeparatorColor = UIColor.DarkGray;
    }
```

2) In the AppDelegate set the flyout navigation subclass as the root view controller in the `AppDelegate FinishedLaunching` method:

```
FlyoutNav = new EvolveFlyoutNavigationController ();
window = new UIWindow (UIScreen.MainScreen.Bounds);
window.RootViewController = FlyoutNav;
```

3) Although some implementations of the flyout navigation rely on the user to "swipe" to reveal the menu, it is helpful to also provide a more discoverable option. The example app provides a button on the navigation bar to reveal the menu, implemented like this:

```
var bbi = new UIBarButtonItem(UIImage.FromBundle ("images/slideout.png"),
UIBarButtonItemStyle.Plain, (sender, e) => {
        AppDelegate.FlyoutNav.ToggleMenu();
});
NavigationItem.SetLeftBarButtonItem (bbi, false);
```

As with the tab bar controller, each view controller that appears doesn't need to know about the flyout navigation controller that is displaying them. This makes it easy to add flyout navigation to an existing app.

## Other iOS Navigation Controls

In addition to these basic iOS navigation elements, applications can be creative about how they present information and let the user navigate. Other examples that you will find elsewhere in the course materials include:

- Satellite Menu – Another component, explained in the Social + Components chapter.
- Collection View Controller – A control that can be used for data display or navigation. This is discussed in detail in the Advanced chapter on Collection Views.

Some other iOS-specific elements to be aware of include:

- Split View Controller – Introduced by Apple for the iPad, the split view creates a "master detail" screen where the master view is usually a scrolling table that can be shown or hidden.
- Pager View Controller – Provides an animated "page turning" effect useful for simulating book or magazine reading.

# Android

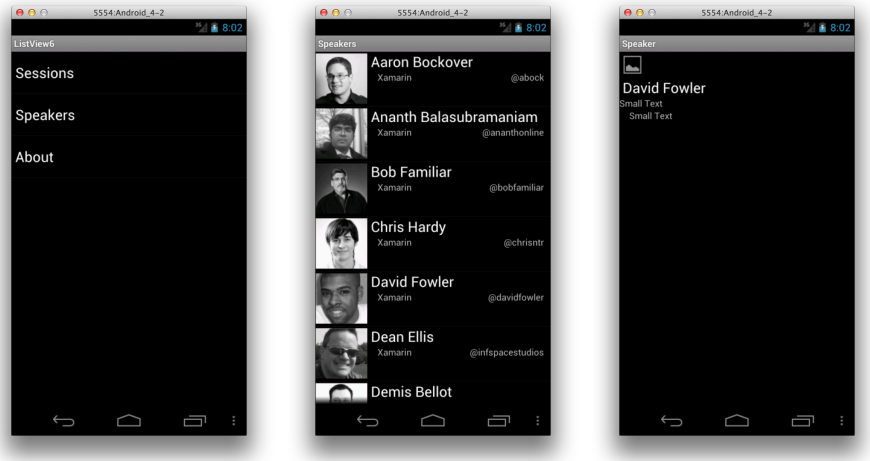
Android application navigation patterns are often similar to those in iOS, although they may be rendered quite differently. Because Android devices originally had

hardware “back” and “options menu” buttons, these featured prominently in Android applications. More recent versions of the Android operating system call for these hardware buttons to be replaced by software-driven controls, but the functionality is usually the same.

## Stack

The “stack” (or hierarchical) navigation pattern on Android looks similar to iOS – it can be driven by lists of options or any other view, and lets the user drill down into different options. One key difference is the lack of the context-aware “back” button, instead the application code expects the user to press the hardward or system-provided software back button (which appears outside of the application itself).

The AndroidStack example application looks like this:




### Example: AndroidStack

The following steps were taken to implement the AndroidStack example:

1) The first step is to create a simple menu in a list view, in the MenuActivity:

```
items = new string[] { "Sessions", "Speakers", "About" };
ListAdapter = new ArrayAdapter<String>(this,
Android.Resource.Layout.SimpleListItem1, items);
```

2) Then implement the OnListItemClick to show the next level of menu based on the selection:

```
protected override void OnListItemClick(ListView l, View v, int position,
long id)
{
    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);
}
```

3) In subsequent activities, simply start a new intent. Android will automatically take care of navigating backwards. You can use the PutExtra method to add parameters for the next activity:

```
protected override void OnListItemClick(ListView l, View v, int position,
long id)
{
    var speakerName = adapter[position].Name;
    var intent = new Intent(this, typeof(SpeakerActivity));
    intent.PutExtra("Name", speakerName);
    StartActivity(intent);
}
```

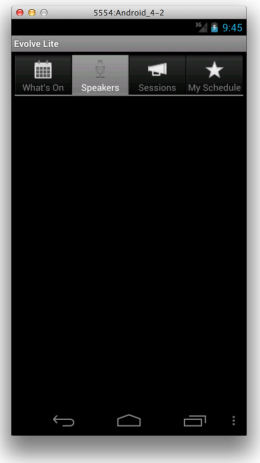
4) Use the `Intent.Extras` property to retrieve parameters passed to the subsequent activities, which can be used to populate the screen with relevant data:

```
var name = Intent.Extras.GetString ("Name");
```

The activities that appear for each tab don't need to know about the tab view that is displaying them.

# Tabs

Android tabs are typically rendered along the top of the screen (unlike iOS which is rendered at the bottom).




### Example: AndroidTabs

1) Ensure that the MainActivity inherits from `TabActivity`:

```
 [Activity (Label = "Tabs", MainLauncher = true,
Icon="@drawable/ic_launcher")]
public class MainActivity : TabActivity
```

2) Implement a helper method to create tab items:

```
private void CreateTab(Type activityType, string tag, string label, int
drawableId)
{
    var intent = new Intent(this, activityType);
    intent.AddFlags(ActivityFlags.NewTask);
```

```
    var spec = TabHost.NewTabSpec(tag);
    var drawableIcon = Resources.GetDrawable(drawableId);
    spec.SetIndicator(label, drawableIcon);
    spec.SetContent(intent);
    TabHost.AddTab(spec);
}
```

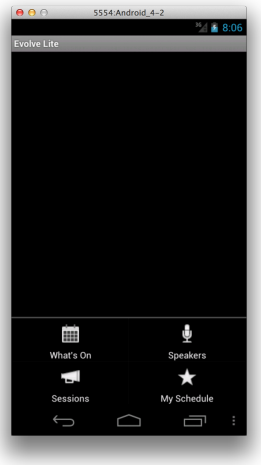
3) Create the tabs, specifying text and images:

```
CreateTab(typeof(WhatsOnActivity), "whats_on", "What's On",
Resource.Drawable.ic_tab_whats_on);
CreateTab(typeof(SpeakersActivity), "speakers", "Speakers",
Resource.Drawable.ic_tab_speakers);
CreateTab(typeof(SessionsActivity), "sessions", "Sessions",
Resource.Drawable.ic_tab_sessions);
CreateTab(typeof(MyScheduleActivity), "my_schedule", "My Schedule",
Resource.Drawable.ic_tab_my_schedule);
```

Each activity will then appear inside the tab control when selected.

# Options Menu

In older Android devices the Options menu is triggered from a hardware button. Newer Android devices show a software-driven icon (three vertical dots) to trigger it.




Example: AndroidOptions

1) Because we want the same menu to appear in every screen of the sample, create a BaseActivity class and override `OnCreateOptionsMenu` with the following code (menus can also be defined with XML resources):

```
public override bool OnCreateOptionsMenu(IMenu menu)
{
        var item = menu.Add(Android.Views.MenuConsts.None, 1, 1, new
Java.Lang.String("What's On"));
        item.SetIcon(Resource.Drawable.ic_tab_whats_on);
        item = menu.Add(Android.Views.MenuConsts.None, 2, 2, new
Java.Lang.String("Speakers"));
        item.SetIcon(Resource.Drawable.ic_tab_speakers);
        item = menu.Add(Android.Views.MenuConsts.None, 3, 3, new
Java.Lang.String("Sessions"));
```

```
        item.SetIcon(Resource.Drawable.ic_tab_sessions);
        item = menu.Add(Android.Views.MenuConsts.None, 4, 4, new
Java.Lang.String("My Schedule"));
        item.SetIcon(Resource.Drawable.ic_tab_my_schedule);
        return true;
}
```

2) Implement the OnOptionsItemSelected method in the base class, to start a new intent (or whatever you want the menu item to do):

```
public override bool OnOptionsItemSelected(IMenuItem item)
{
    var intent = new Intent();
    switch (item.TitleFormatted.ToString()) {
    case "What's On":
        intent.SetClass(this, typeof(WhatsOnActivity));
        intent.AddFlags(ActivityFlags.ClearTop);
        StartActivity(intent);
        return true;
// others removed for brevity
```

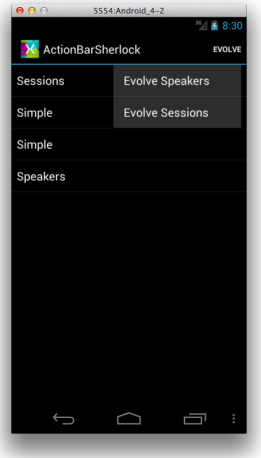
3) Ensure the other activities inherit from the BaseActivity that defines the options menu:

```
[Activity (Label = "SessionsActivity")]
public class SessionsActivity : BaseActivity
```

While the options menu is still supported, applications that only target Android 4.0 and newer are encouraged to use ActionBar instead. There are also libraries such as ActionBarSherlock (discussed next) that provide a compatibility layer to allow apps to target old and new versions of the operating system while sharing common menu code.

## ActionBarSherlock

ActionBar was introduced with Android 3.0 (4.0?) to provide a new navigation framework for phone and tablets apps. ActionBarSherlock provides a more modern Options Menu than the previous example.



## Other Android Navigation Controls

This chapter has primarily discussed using Activities to navigate and display data. Android fragments were introduced in Android 3.0 to provide more layout flexibility and sharing of views across different devices (such as phones and tablets).

# Summary

In this chapter we have looked at how to implement some of the most common navigation user interfaces for iOS and Android. Understanding the basics of implementing these controls will help you start building applications quickly, but also provide a foundation for combining them to build more complex user interfaces.

### Further Reading

To learn more about different types of mobile navigation user interfaces, check out some of these links:

- http://www.mobile-patterns.com/
- http://inspired-ui.com/
- http://pttrns.com/ (iOS)
- http://www.androiduipatterns.com/ (Android)