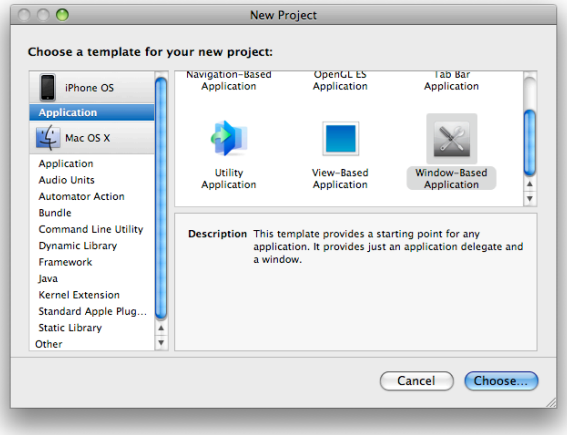
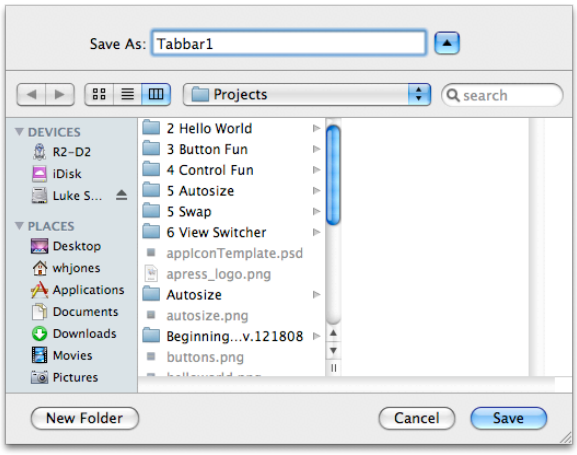
# Tab Bar Controllers and Navigation Controllers

## Part 1: Tab Bar Controllers

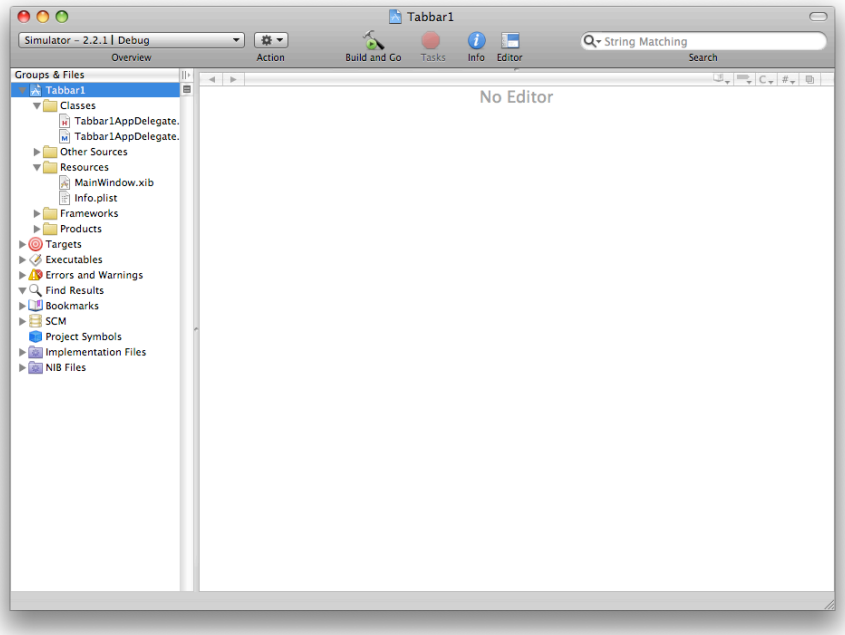
The purpose of this tutorial is to show how to use tabbar controllers and navigation controllers by themselves, then combine them to use them both in the same application. Several iPhone applications use both controllers concurrently, and some even use different navigation controllers for different tabs on the tabbar.

To start, let's fire up XCode and start a new Window-based application, called Tabs1.

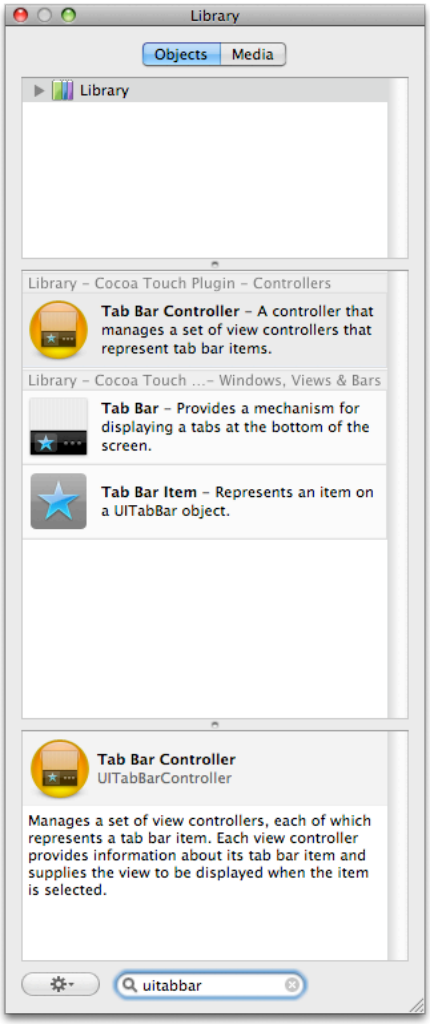


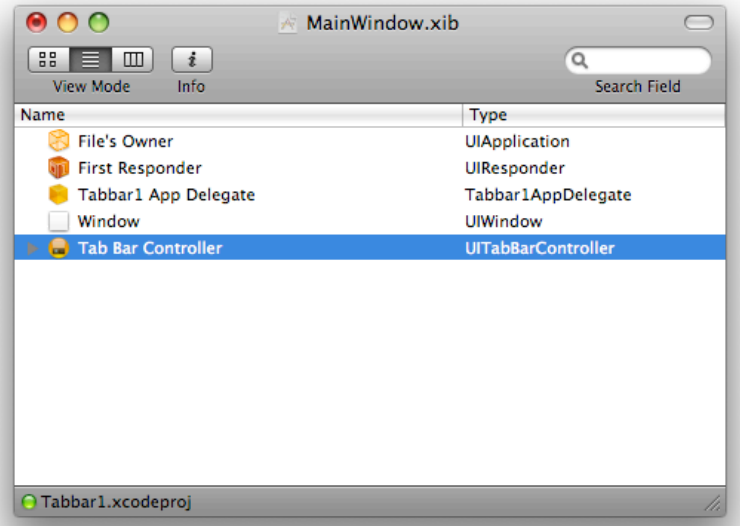



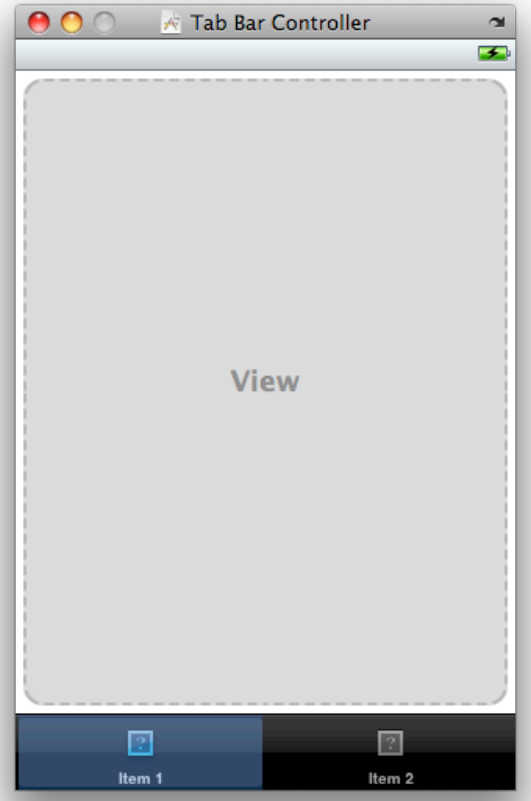

Next, expand Classes and Resources - not too much to your app yet, huh? We'll fix that pretty quickly.



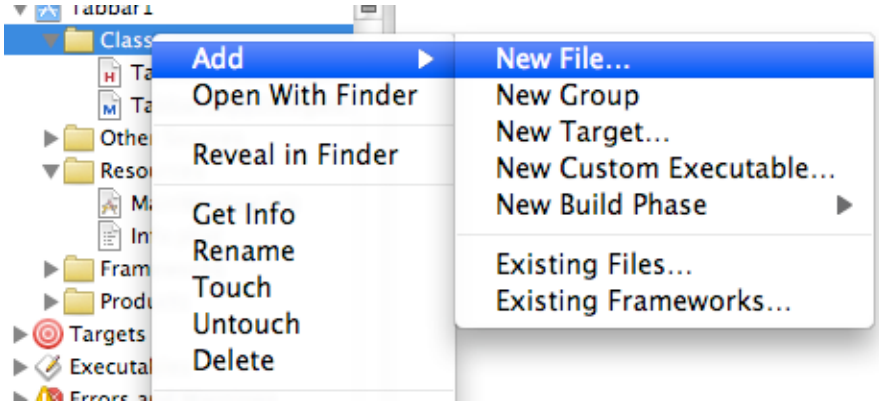
Next, fire up Interface Builder by double-clicking on MainWindow.xib in the Resources section.  Find a UITabbarController in the Library and drag it to the MainWindow.xib window.  You should notice another view window appear, labeled Tab Bar Controller.  You can close out the other view Window, as you will work with the Tab Bar Controller view window from now on.

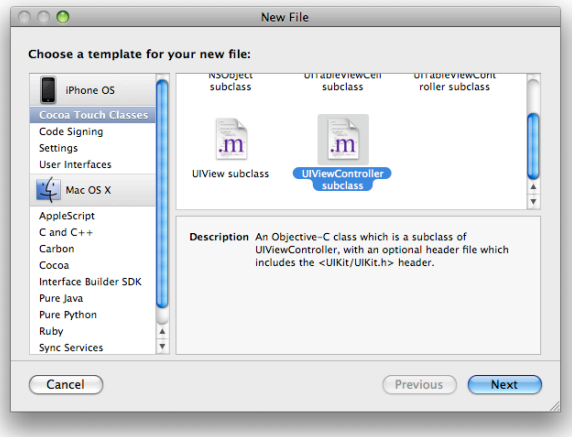


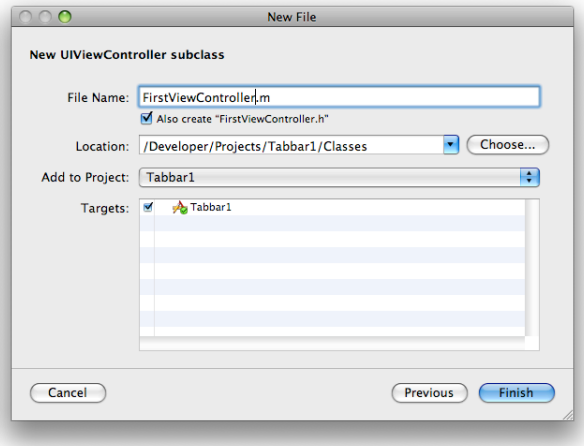



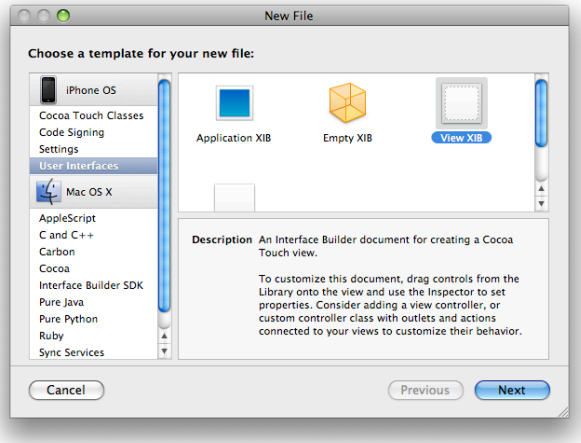



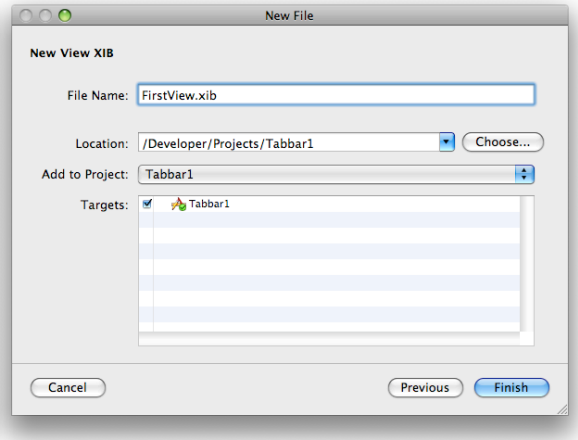
Next, you'll have to create two new View Controller classes to display in your Tab Bar Controller window. Let's do that now. Create two new UIViewController classes, called FirstViewController and SecondViewController. You'll also need to create two new Empty View XIBs, called FirstView and SecondView.



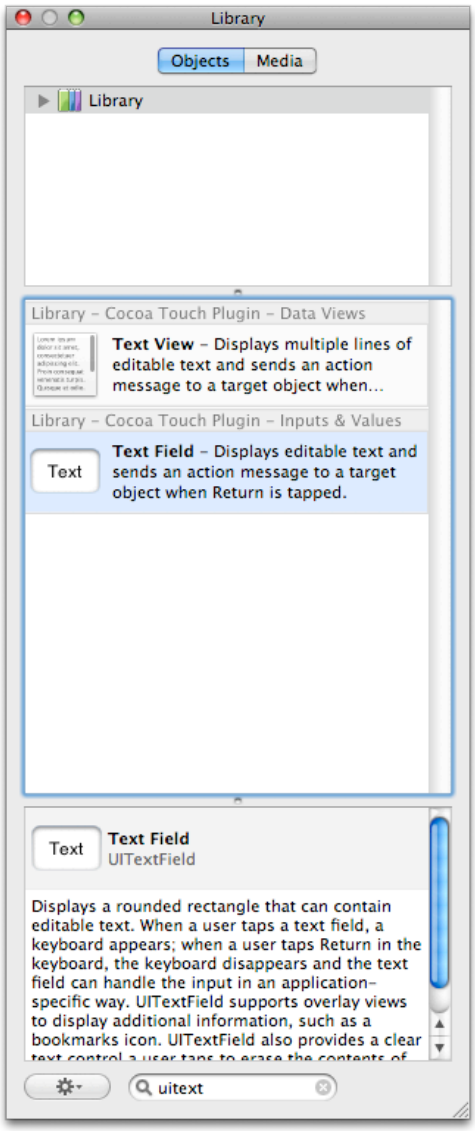



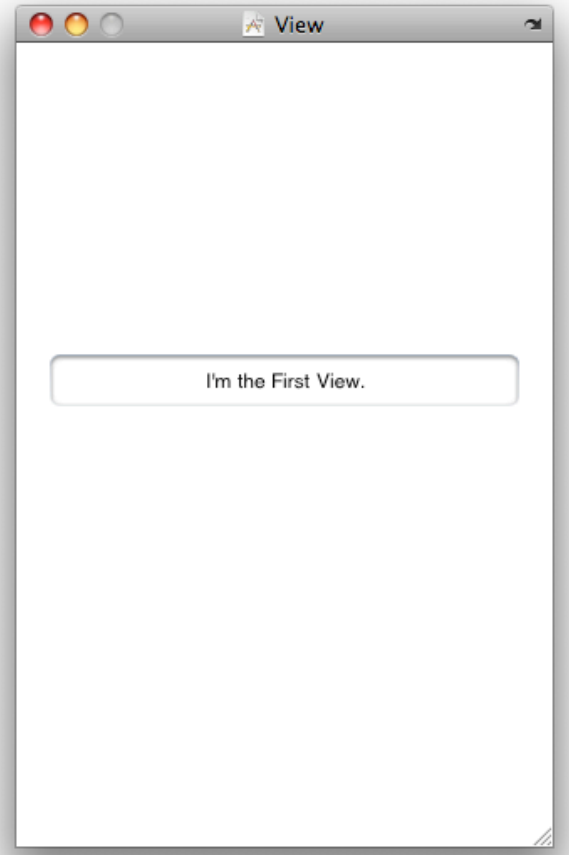



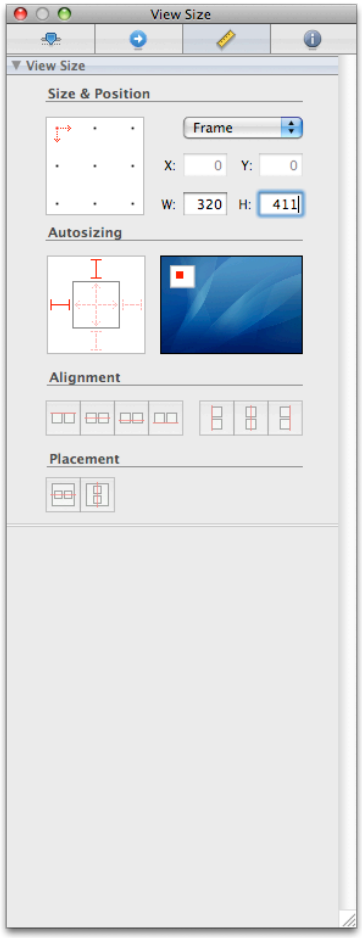



Now, add a UITextField to both the FirstView.xib and SecondView.xib to show that we're really displaying what we want to see. Open up both .xibs, then from the Library drag a UITextField onto each of their view windows, then type something like "I'm the First View." and "I'm the Second View." or whatever you like into the text fields.



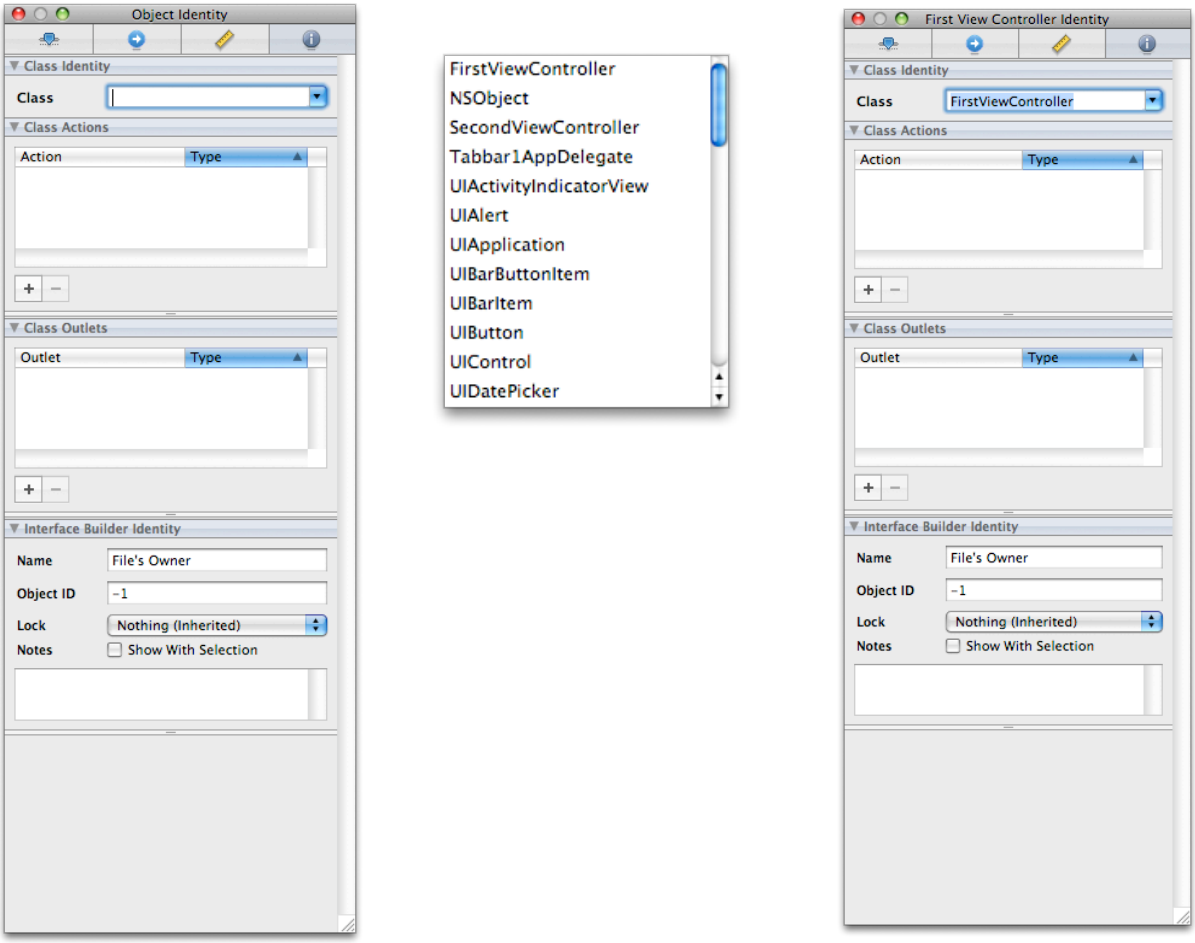




The next thing you must do is change the height of the view to 411 pixels. Why, you ask? Well, the tab bar sits on the bottom of the viewing area. Normally, your viewing area is 460 pixels high, but the tab bar itself is 49 pixels, leaving you with 411 pixels for your maximum viewing area. Click on the Size tab (the third one) and change the height to 411 pixels there. You will notice that the FirstView view window immediately changes to the new dimensions. Do the same for SecondView.xib.

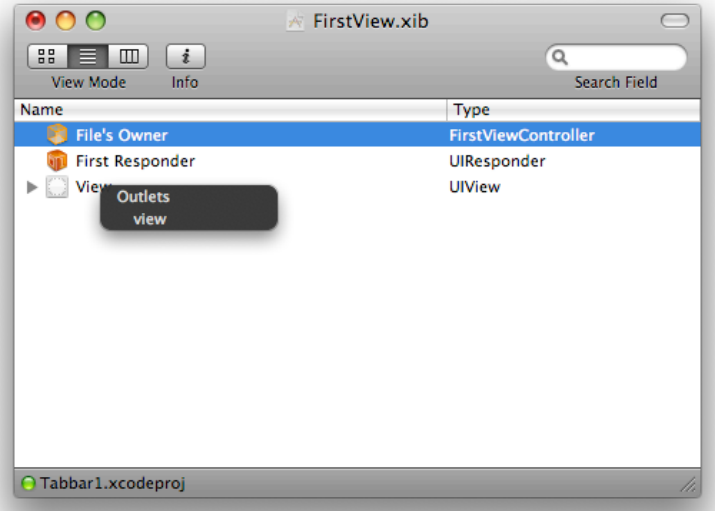




Now, you have to tell Interface Builder the class to which this .xib file belongs.  In the FirstView.xib and SecondView.xib windows, click to select File's Owner and in the Inspector window, click the last tab on the right, the Identity tab, and choose the appropriate class from the drop-down menu in the topmost section.

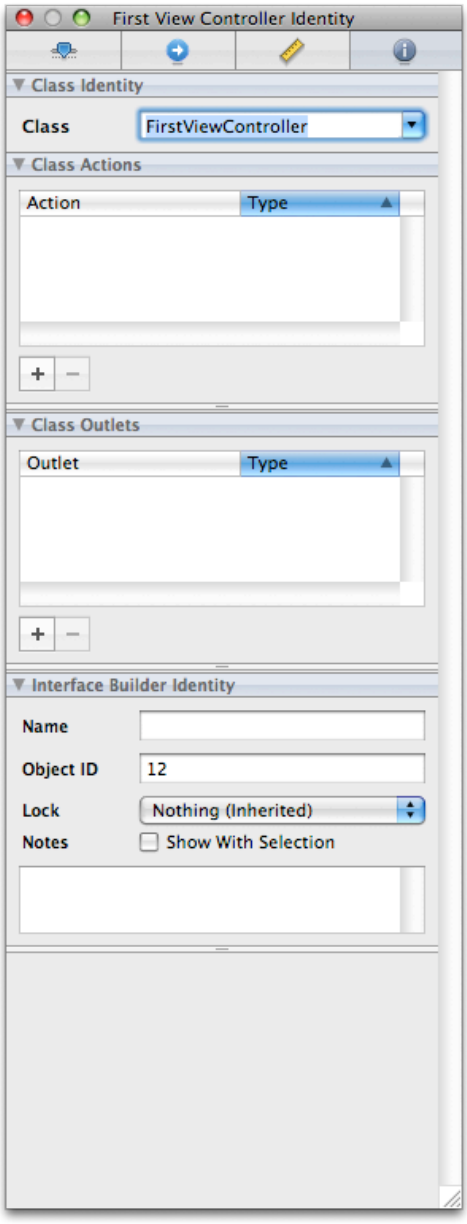


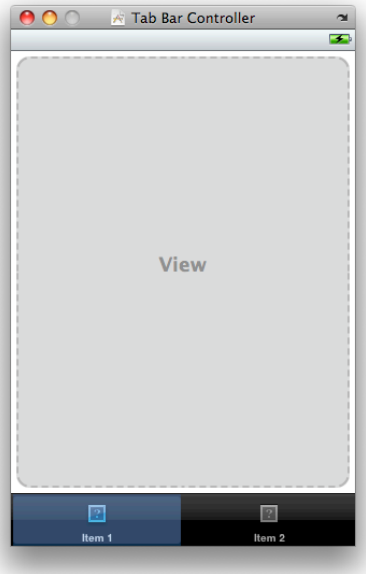

Now, control-drag from the File's Owner object to the View object in both the FirstView.xib and SecondView.xib windows and select the view Outlet in the dark menu that appears.

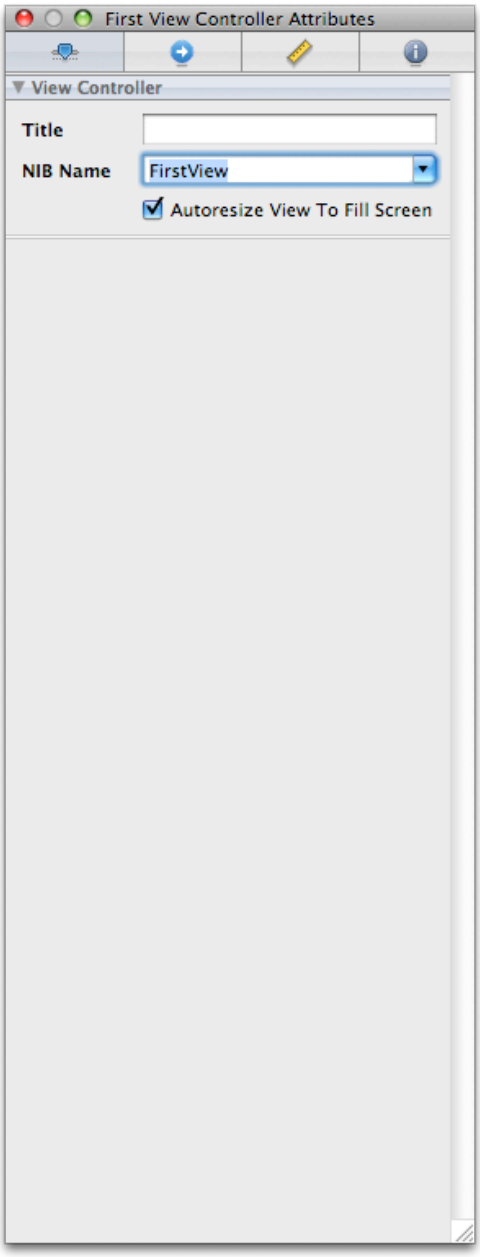


That tells the .xib to show the View window (the one on which you just placed a UITextField) when the .xib file is loaded, which is exactly the action you desire. At this point, you can safely save and close your FirstView.xib and SecondView.xib files.

In the MainWindow.xib, click to select the left tab in the tabbar (if you click and the tabbar item, labeled Item1, is highlighted, click in the View section, then click back on the left tabbar button). In the Inspector, click the Identity tab and select FirstViewController as the class. Lastly, click the Attributes tab and select FirstView as the nib file. You will now notice that the View has changed to reflect the nib file to be loaded when that tabbar button is pressed. Repeat the same steps for the right button, but change its class to SecondViewController and have it show SecondView as its nib.







You probably noticed a textfield labeled Name in the Attributes tab of the Inspector.  Tab bars do not use this field, but navigation controllers do set this variable, although we will set this in code and not in the Inspector.

Speaking of code, now we get to go back to XCode.  You will need to add an ivar (instance variable) for the tabbar controller in your app delegate class.  Make your Tabbar1AppDelegate.h file look like this:

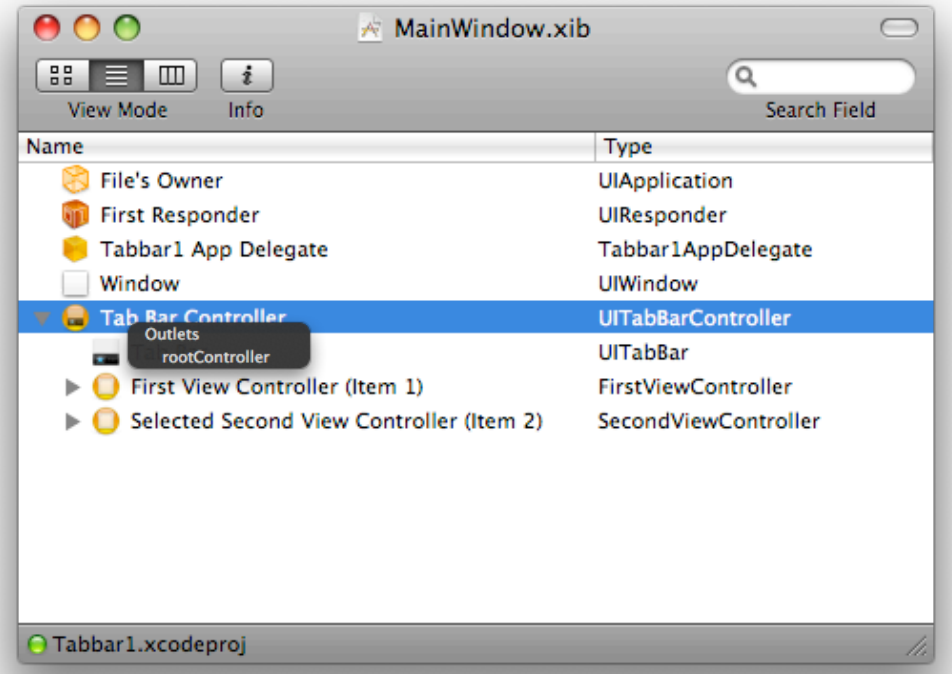
```
//
//  Tabbar1AppDelegate.h
//  Tabbar1
//
//  Created by William Jones on 6/14/09.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface Tabbar1AppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    IBOutlet UITabBarController *rootController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UITabBarController *rootController;

@end
```

Now, make your Tabbar1AppDelegate.m file look like this:

```
//
//  Tabbar1AppDelegate.m
//  Tabbar1
//
//  Created by William Jones on 6/14/09.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "Tabbar1AppDelegate.h"

@implementation Tabbar1AppDelegate

@synthesize window;
@synthesize rootController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    [window addSubview:rootController.view];
    [window makeKeyAndVisible];
}


- (void)dealloc {
    [window release];
    [super dealloc];
}


@end
```

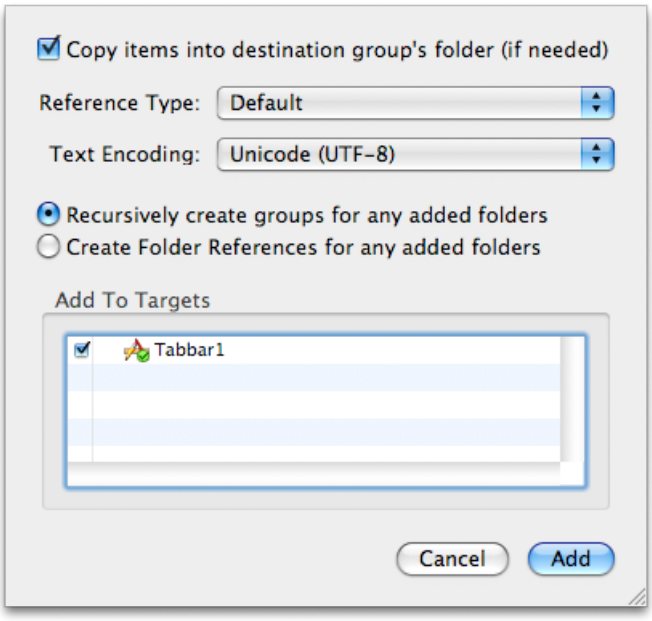
The first thing you should notice is that the rootController ivar is an IBOutlet. This basically tells Interface Builder that it can be used for connections. We will get to that in a moment. You will also notice that in the applicationDidFinishLoading: method, the view of the rootController is the one being added as a subview. You haven't connected up the rootController in IB yet, however, so let's do that now.

Double-click on MainWindow.xib to open it. Control-drag from the Tabbar1 App Delegate object to the Tab Bar Controller object and click on the rootController outlet in the dark menu.

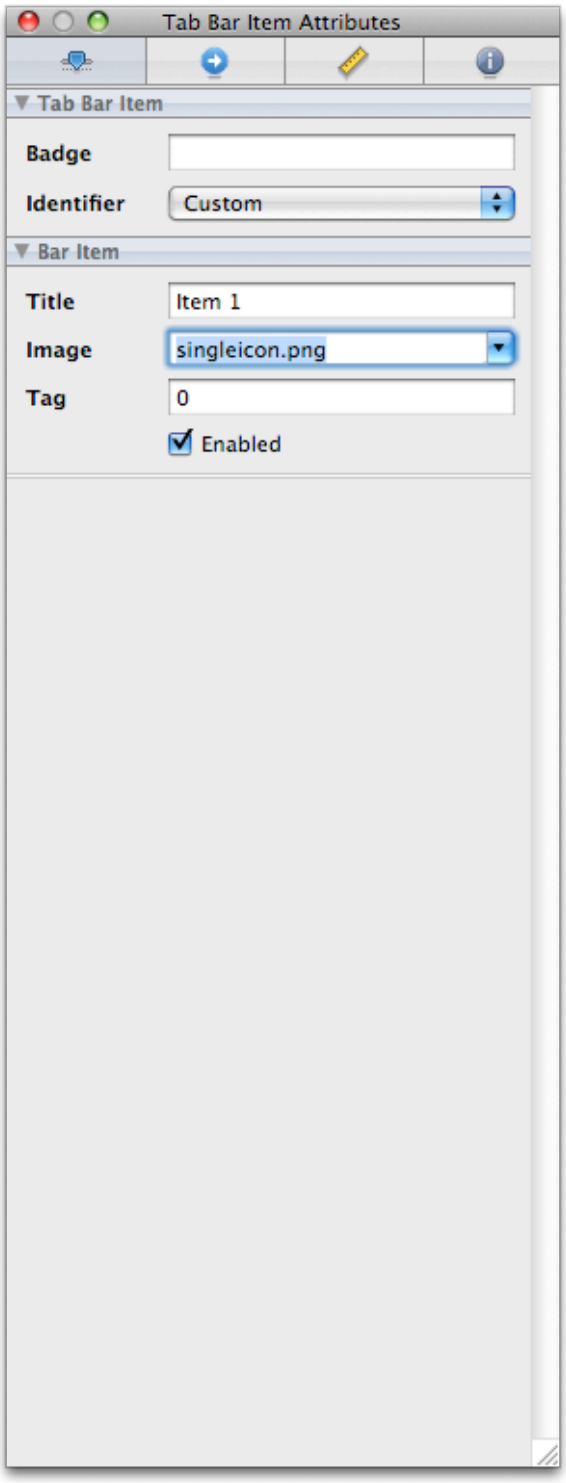



You should be set to go here.  Click Build and Go and observe your handiwork!  It should look like the following:

You can also change the background color of each view to better differentiate between the two views. Also, you can create icons for the tab bar items (they sit above the tab bar item labels of Item1 and Item2). They should be 24 pixels x 24 pixels and be a medium gray color. All you need do is drag them to the Resources section in XCode and choose to copy them over (check the top box).

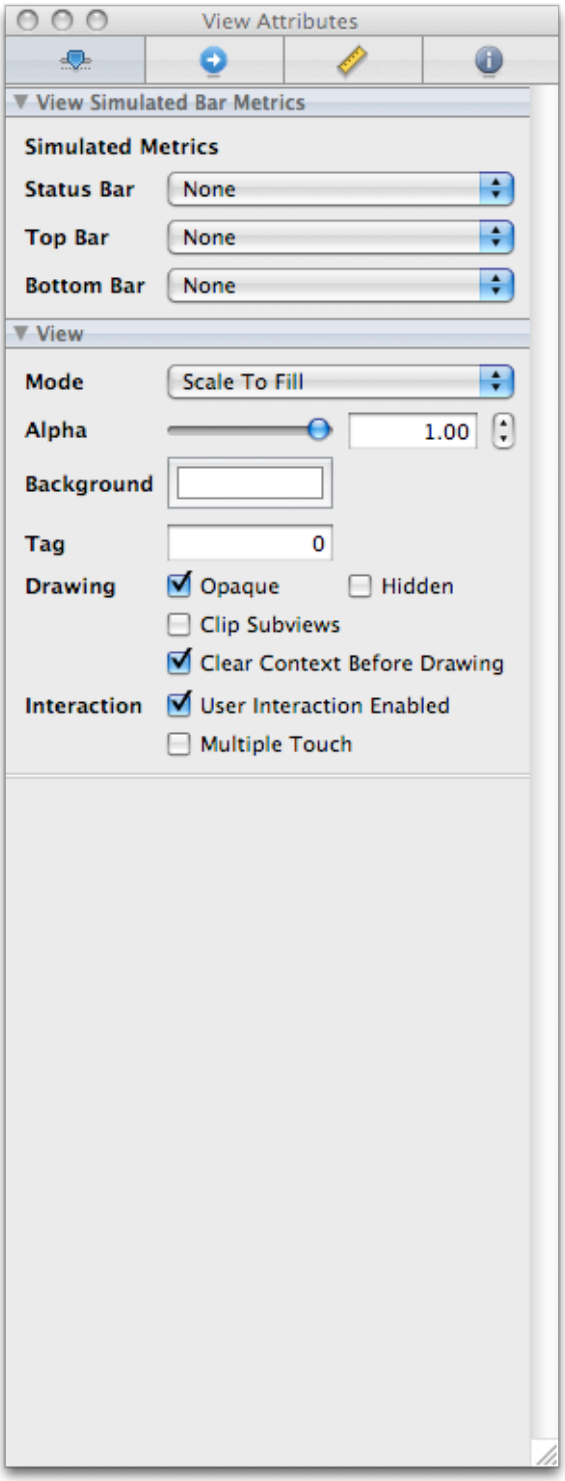


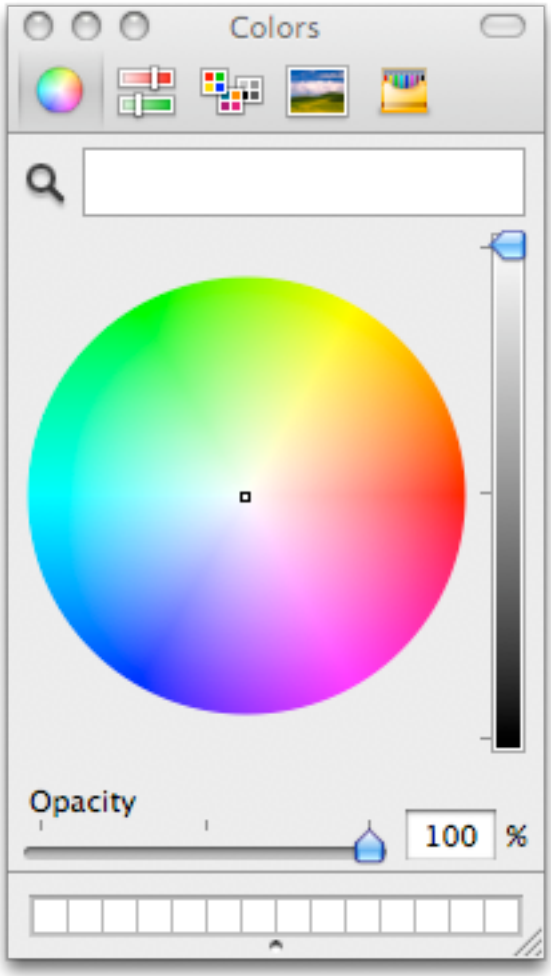

Then, in the MainWindow.xib, click once on the first tab bar item, then once again to select it, then in the Inspector click on the Attributes tab and from the list in the drop-down menu, select the desired image file.


You will notice the new icon show up in the MainWindow.xib file in the appropriate place on the tab bar.  Repeat this process for the second tab, then save the .xib file and click on Build and Go to observer your progress.


To change the background of your .xib, in the Attributes tab of the Inspector, click on the colorwheel beside the Background label and select your desired color.

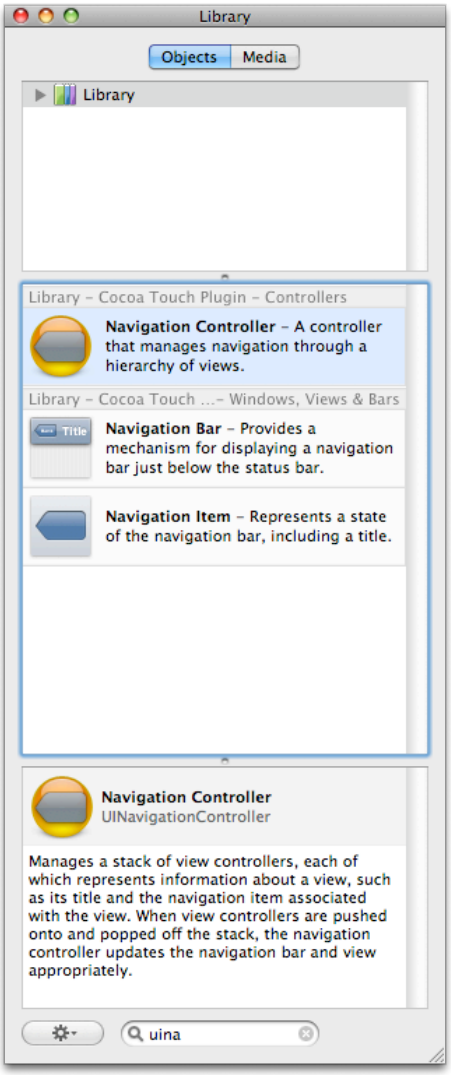

View Attributes
View Simulated Bar Metrics
Simulated Metrics
Status Bar
None
Top Bar
None
Bottom Bar
None
View
Mode
Scale To Fill
Alpha
1.00
Background
Tag
0
Drawing
Opaque
Hidden
Clip Subviews
Clear Context Before Drawing
Interaction
User Interaction Enabled
Multiple Touch

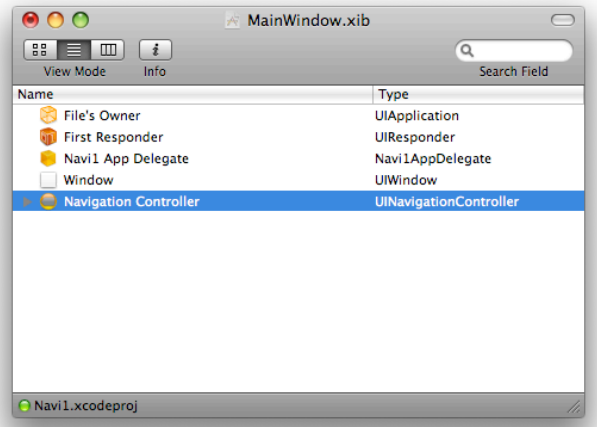

Colors
Opacity
100
%

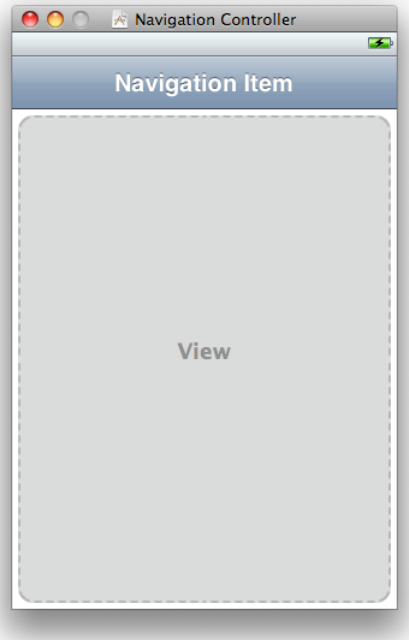
## Part 2. Navigation Controllers

Well, that seems like a bit, doesn't it? It really wasn't all that intense, once you get the hang of it. Next up, we'll learn about navigation controllers, which sit on the top of the view, just below the status bar. They also sport back arrows to let you know the previous view. Navigation controllers push and pop views onto the view controller stack. This makes them somewhat similar to tab bar controllers, but still just a little bit different.

For this section, fewer screenshots will be used, as the steps are quite similar to those performed in Part 1. Create a new Window-based project, called Navi1. We can choose to use IB to create the navigation controller or do the same in code. I normally do it in code, but let's use IB to get it done. Double-click on MainWindow.xib to open it up in IB. Next, find a navigation controller in the Library and drag it over the the MainWindow.xib window. Another view window will appear, with the navigation controller visible. Close the plain view window as you will no longer need it.



Navigation controllers have a “root view” that controls the bottom-most view controller in the stack, which is to say the first view presented to the user. For that reason, create a new UIViewController subclass called RootViewController. This class will not need an accompanying .xib file, as it is being used abstractly. We will be adding an IBOutlet named navController to control the actual navigation. Create the accompanying .xib file as well.

Now, just as in Part 1, create two new UIViewController subclasses, FirstViewController and SecondViewController, as well as their accompanying .xib files, then put something on the .xib files to differentiate them from each other, and change their height to 411 pixels (the navigation controller is the same height as the tab bar controller). Finally, connect File’s Owner to the correct class and connect up the view outlet. If you forgot how to do that, refer to Part 1. We’re going to navigate from the RootViewController to the FirstViewController, then to the SecondViewController, and back again.

Next, make your Navi1AppDelegate.h file look like this:

```
//
//  Navi1AppDelegate.h
//  Navi1
//
//  Created by William Jones on 6/14/09.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface Navi1AppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    IBOutlet UINavigationController *navController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UINavigationController *navController;

@end
```

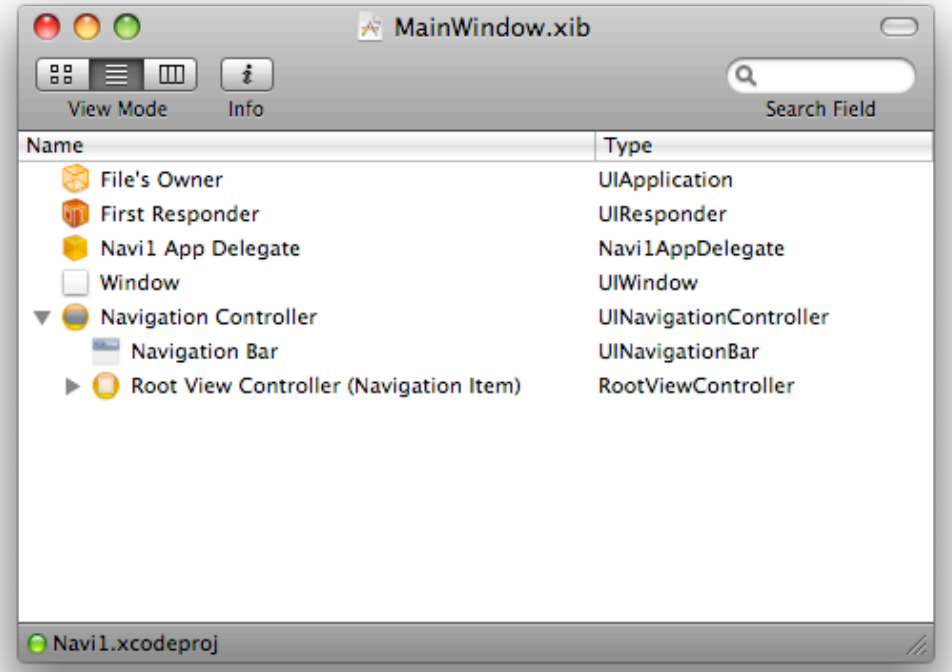
Next, make your Navi1AppDelegate.m file look like this:

```
//
//  Navi1AppDelegate.m
//  Navi1
//
//  Created by William Jones on 6/14/09.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "Navi1AppDelegate.h"
#import "FirstViewController.h"

@implementation Navi1AppDelegate

@synthesize window;
@synthesize navController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [window addSubview:navController.view];
    [window makeKeyAndVisible];
}


- (void)dealloc {
    [navController release];
    [window release];
    [super dealloc];
}


@end
```

The applicationDidFinishLaunching: method is very similar to the one used in the tab bar application. We add the navController view and the view that appears when the user launches the app. How is that accomplished? I'm glad you asked.

Double-click to bring up MainWindow.xib and put the objects into list view. Single-click on the Navi1 App Delegate object and select the Identity tab in the Inspector to verify that the navController outlet exists (it does, because once you enter it into the appropriate class and save that file, the ivars are able to be used in IB). Next, control-drag from the Navi1 App Delegate object to the Navigation Controller object and select the navController outlet from the dark menu.

Next, expand the Navigation Controller object and single-click the View Controller (Navigation Item) to select it. Click the Identify tab in the Inspector and select RootViewController in the class drop-down list. You'll notice the class change in the MainWindow.xib file as well.

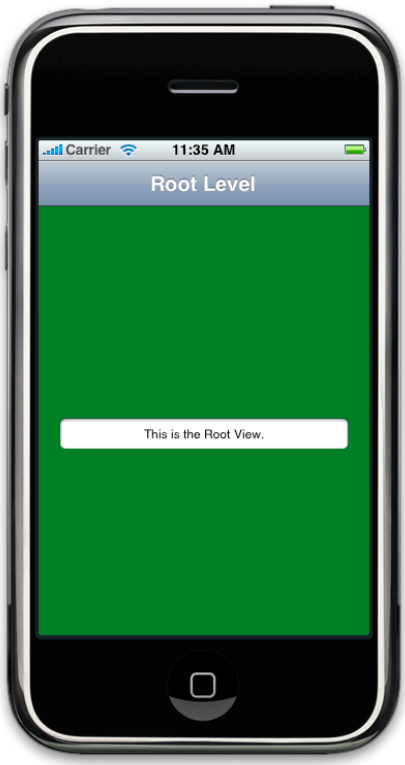



What will the user see, you ask? Well, we have to put some content here, if you have not done so already. Double-click to open RootView.xib, change its height to 411 pixels and put whatever content your heart desires. This is the initial view presented to the user when the application launches.

Now, to make sure everything works as advertised, make your Navi1AppDelegate.m's viewDidLoad method look like this:

```
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    self.title = @"Root Level";
    [super viewDidLoad];
}
```

You will be using the viewDidLoad method extensively when working with a navigation controller. You should change the title for all of your views, so as to delineate between them, hence the self.title line in this method. You should be able to Build and Go to verify your progress. I got this:




So, how do we pull up FirstView and SecondView? It will be the same process on each, so I'll show you how to do the first and leave the second up to you. The first think you'll need to do is provide the user a way to address the next view. This is normally done with a button, or if your first view is a tableview, by the didSelectRowAtIndexPath: method. For simplicity's sake, I chose an ordinary view, so we will just use buttons.

Double-click to open RootView.xib and drag a UIButton over from the Library. The default button looks pretty weak, so I suggest you go to http://developer.apple.com/iphone/library/samplecode/UICatalog/index.html, download the .zip file and pull out the bluebutton.png and whitebutton.png files from the images folder. Next add them to your Resources section. You now need to add the button to both your RootViewController.h and .m files. Make your RootViewController.h file look like this:

```
//
//  RootViewController.h
//  Navi1
//
//  Created by William Jones on 6/14/09.
//  Copyright 2009 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface RootViewController : UIViewController {
    IBOutlet UIButton *pressMe;
}

@property (nonatomic, retain) IBOutlet UIButton *pressMe;

@end
```

Next, synthesize the pressMe button in the .m file and make your viewDidLoad method look like this:

```
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    self.title = @"Root Level";
    UIImage *buttonImageNormal = [UIImage imageNamed:@"whiteButton.png"];
    UIImage *stretchableButtonImageNormal = [buttonImageNormal stretchableImageWithLeftCapWidth:12
                                                                                   topCapHeight:0];
    [pressMe setBackgroundImage:stretchableButtonImageNormal
                       forState:UIControlStateNormal];

    UIImage *buttonImagePressed = [UIImage imageNamed:@"blueButton.png"];
    UIImage *stretchableButtonImagePressed = [buttonImagePressed stretchableImageWithLeftCapWidth:12
                                                                                     topCapHeight:0];
    [pressMe setBackgroundImage:stretchableButtonImagePressed
                       forState:UIControlStateHighlighted];

    [super viewDidLoad];
}
```

You added two buttons to your project, one for its normal state, and one for its pressed (highlighted) state. The code should be self-explanatory. The left end cap is used to accurately present the button's image (the curvature) regardless of how large or small the button is made on the view. That being said, if the button is not changed in IB, it will be displayed as it is now.

Double-click to bring up RootView.xib and select the button. Next, click the Attributes tab in the Inspector and change the button type from Rounded Rect to Custom. You will notice the button dramatically change in IB, but do not fret, this behavior is exactly what is expected. Next, in the RootView.xib window itself, control-drag from File's Owner to the

the button and connect it to the pressMe outlet. That way, IB knows about the existence of the button and we can then create the method to bring up the FirstViewController class.

Close RootView.xib and click to bring up RootViewController.h. Make it look like this:

```
//
//  RootViewController.h
//  Navi1
//
//  Created by William Jones on 6/14/09.
//  Copyright 2009 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface RootViewController : UIViewController {
    IBOutlet UIButton *pressMe;
}

@property (nonatomic, retain) IBOutlet UIButton *pressMe;

- (IBAction)pressMe:(id)sender;

@end
```

An IBAction is a method called by an IBOutlet. This one is being called by our button of the same name. Now, switch to RootViewController.m, import FirstViewController.h, and add this method to the top:

```
//
//  RootViewController.m
//  Navi1
//
//  Created by William Jones on 6/14/09.
//  Copyright 2009 __MyCompanyName__. All rights reserved.
//

#import "RootViewController.h"
#import "FirstViewController.h"

@implementation RootViewController

@synthesize pressMe;

- (IBAction)pressMe:(id)sender
{
    FirstViewController *firstView = [[FirstViewController alloc] initWithNibName:@"FirstView"
                                                                            bundle:nil];
    [self.navigationController pushViewController:firstView
                                         animated:YES];
    [firstView release];
}
```

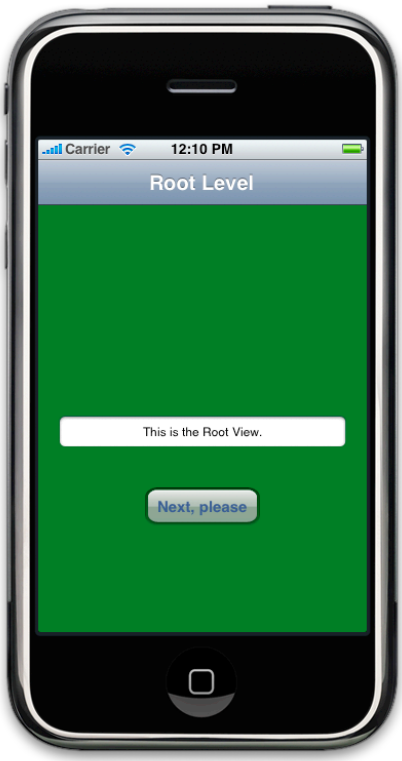
You are almost done. Finally, double-click to bring up RootView.xib, single-click to select the button, then click the Connections tab in the Inspector (the second one from the left),

drag from the circle to the right of Touch Up Inside to File's Owner, and connect it to the pressMe method. Click Build and Go to see your progress. This is what I have thus far:

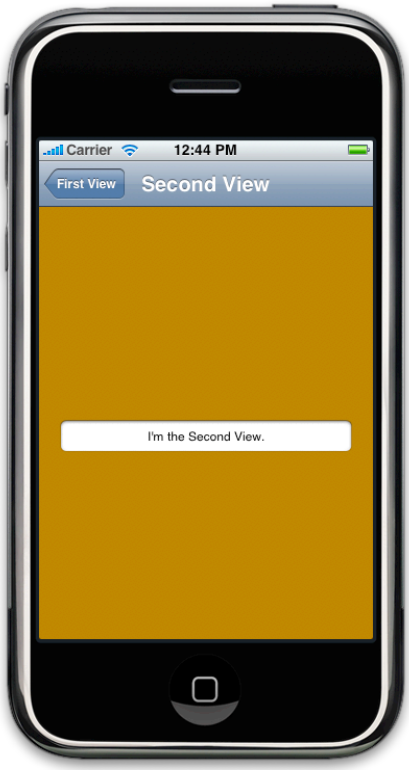

You will notice that FirstView does not have a title. Do the same thing as with RootView to change that. You will also notice that when FirstView is called up, the back arrow contains the title of the previous view, Root Level. Where did we enter the code for that? We did not have to enter anything, as Apple has already done that in the background for us. Part of the frameworks we are using adds this functionality. Of course, if you wish to have different text in the back arrow, that can be added programatically. To change the tex of the back arrow, simply enter the following into the viewDidLoad method of RootViewController.m as follows:

```
UIBarButtonItem *backButton = [[UIBarButtonItem alloc] initWithTitle:@"Root View"
                                                               style:UIBarButtonItemStyleBordered
                                                              target:nil
                                                              action: nil];
self.navigationItem.backBarButtonItem = backButton;
[backButton release];
```

Now, when you go to FirstView, you will see "Root View" in the back arrow. Ok, that seemed like a bit. Do the same thing again to add SecondView and test out your work. You should get something like this:


Carrier
12:10 PM
Root Level
This is the Root View.
Next, please


Carrier
12:44 PM
Root View
FirstView
I'm the First View.
Next, please


Carrier
12:44 PM
First View
Second View
I'm the Second View.

Part 3. Combining a Tab Bar Controller with a Navigation Controller

If you thought the first two parts were tricky, try this one on for size: we are now going to combine a tab bar controller with a navigation controller. As per Apple's documentation, you must embed the navigation controller within the tab bar controller, and not the other way around. To that end, in our next project, we must create the tab bar controller first, then create the navigation controller. The easiest way to do this is to use the tab bar project from part 1, but if you wish to re-create it (so as not to destroy the first project), feel free to do so. In this project, we will be creating three different view controllers, one each for the two tab bar controllers, and the third to demonstrate the functionality of the navigation controller.

Strap yourself in - this may be a bit of a bumpy ride. Create a new Window-based project in XCode and call it TabiNavi (a joke on the two types of controllers we will use for this project). Create three UIViewController subclasses named FirstViewController, SecondViewController, and ThirdViewController, then create their accompanying .xib files and add desired content to each of them in turn.

Next, double-click to open MainWindow.xib and drag a tab bar controller from the Library to the MainWindow.xib window. You can close the plain View window as you will not need it again.

Add a UIButton to FirstView (this will be used to call up Third View) and add the text field to announce that it is First View. Add the text fields to SecondView and ThirdView as well. If you wish, change the background color as well, to more easily differentiate between the three views. Do not forget to connect each File's Owner object to the correct class and then their View object to the view outlet.

Next, in NaviTabiAppController.h, add a pointer to an instance of a UITabBarController and call it rootController, as in Part 1. Add the appropriate property statement and synthesize it in NaviTabiAppController.m. The next two steps are to be done in IB.

Double-click to open MainWindow.xib. Now, connect the NaviTabi App Delegate object to the rootController outlet by control-dragging to the Tab Bar Controller object. Now, for each of the tab bar buttons, set their class and nibs to FirstViewController/ FirstView and SecondViewController/SecondView, just as you did in Part 1. At this point, you should be able to Build and Go and be able to switch between the two views.

Here's the slightly trickier part - adding the navigation controller. Pick one of the two tab bar buttons, or if you are feeling especially frisky, implement a navigation controller on both buttons. If you decide to do that, you will need to create another UIViewController subclass (name it FourthViewController) to call up with the second tab bar button.

Now we need to edit NaviTabiAppDelegate.h to add the first navigation controller. As in Part 2, add the IBOutlet state and property statement for a pointer to an instance of UINavigationController named navController, then add the synthesize statement to the .m file. Now, let us say that you wanted to implement the navigation controller in FirstViewController. Switch to FirstViewController.h and add the IBOutlet for the button needed to call up ThirdView.

Now, as per b00giZm's blogpost, we will need to add at least one additional UIViewController subclass (which will get re-subclassed as a UINavigationController subclass) to enable the navigation controller functionality. Create a UIViewController

subclass called MyNavigationController, then change the interface statement such that it is a subclass of UINavigationController. Now, double-click to open MainWindow.xib and follow the instructions in b00giZm's post. All you need do now is wire up the button the same way we did in Part 2 and everything should work as expected.

Now, for the SecondViewController, do exactly the same thing as with the FirstViewController and you will now have to separate navigation controllers, each capable of pulling up their independent views, but both of which are hosted in two separate views contained within the tab bar controller. These are the screen shots I got after running NaviTabi: