iPhone App Dev

Lesson 8

Source Codes

https://github.com/makzan/ios-dev-course-example

Contact

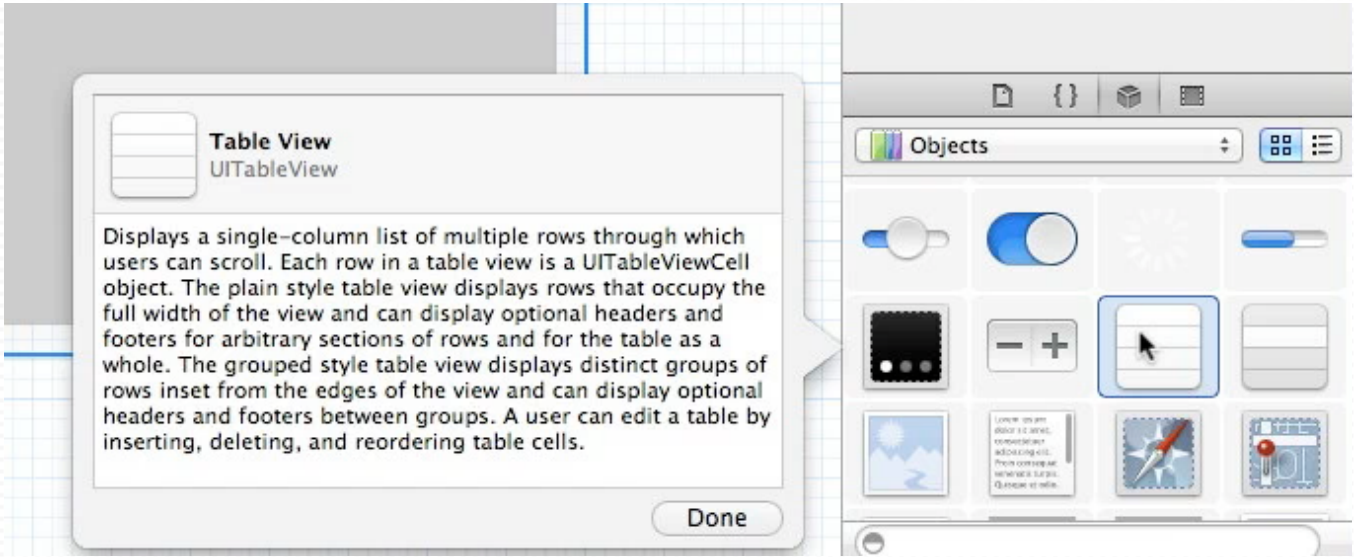
makzan@42games.net

Summary

- Introducing the UINavigationController
- Setting the root view controller
- Traveling between the view hierarchy
- Saving and reading data with NSUserDefault
- Bar Button Item
- Presenting Modal view controller
- Debugging with exceptional break point

Introducing the UINavigationController

Introducing the UINavigation Controller



First, let's identify the table view in interface builder.

Setting the Root View Controller

We'll use the table view demo from last lesson on the UINavigationController demo.

Setting the Root View Controller

AppDelegate.h

```
#import <UIKit/UIKit.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@property (strong, nonatomic) UINavigationController *viewController;

@end
```

We will need to change the AppDelegate.h

Setting the Root View Controller

AppDelegate.m

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
    launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];
    // Override point for customization after application launch.

    ViewController *viewController = [[[ViewController alloc] initWithNibName:@"ViewController" bundle:
        nil] autorelease];

    self.navController = [[UINavigationController alloc] initWithRootViewController:viewController];

    self.window.rootViewController = self.navController;


    [self.window makeKeyAndVisible];
    return YES;
}
```

When the app launches, we create the UINavigationController and set it as Root

Setting the Root View Controller

The result with an empty navigation bar on top

Traveling between the view hierarchy

Traveling between View Hierarchy

- Push a view controller
- Pop a view controller
- Back to root view controller
- Pop to a specific view controller
- Pop to a specific step

Traveling between View Hierarchy

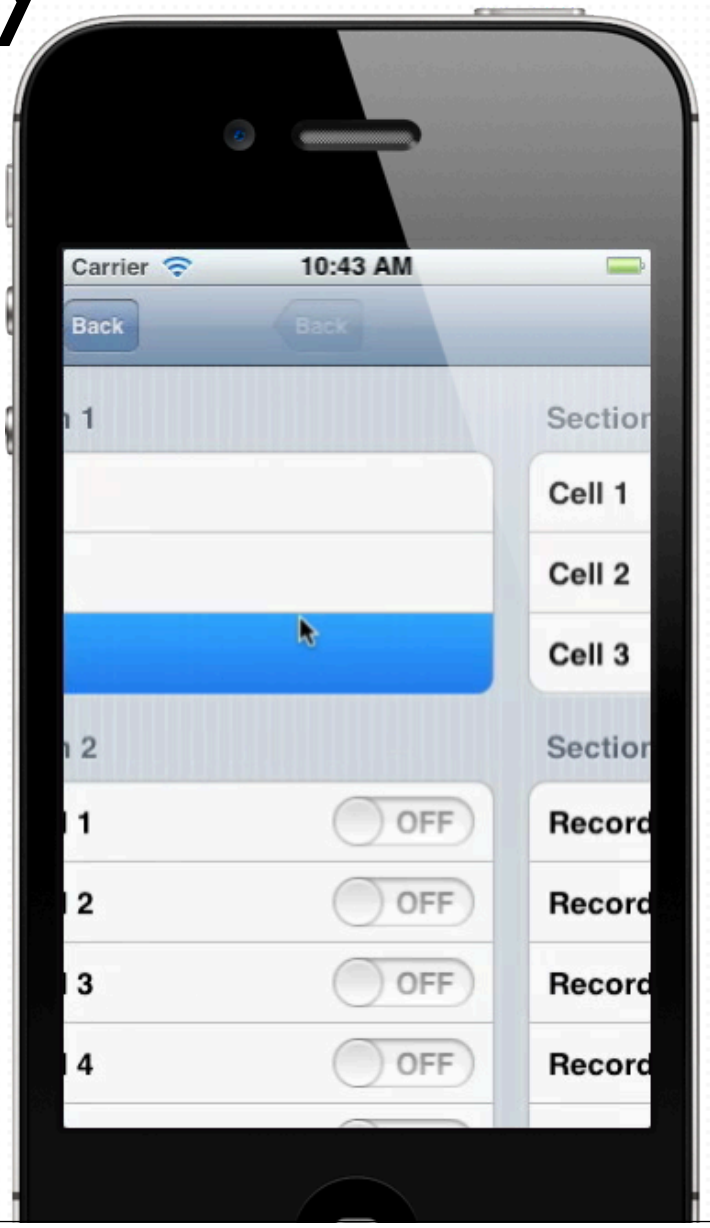
ViewController.m

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    if (indexPath.section == 0)
    {
        // Contsruct another view controller here.
        ViewController *viewController = [[[ViewController alloc] init] autorelease];
        [self.navigationController pushViewController:viewController animated:YES];
    }
}
```

When we tap on a cell, we programmatically construct another view controller, and push it into the navigation hierarchy

Traveling between View Hierarchy

The animation when pushing view controller into hierarchy



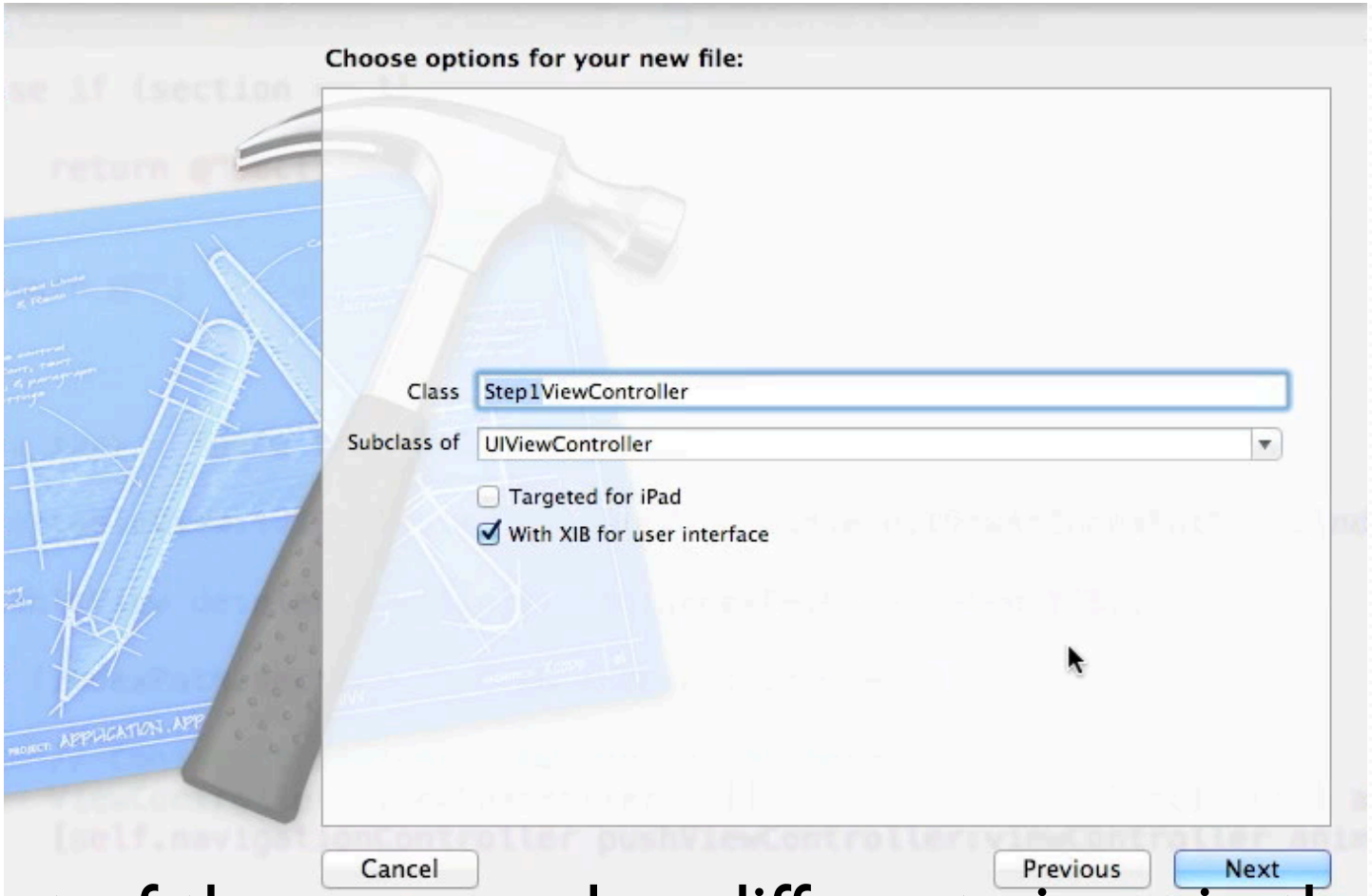
Traveling between View Hierarchy

ViewController.m

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    if (indexPath.section == 0 && indexPath.row == 0)
    {
        // Contsruct another view controller here.
        ViewController *viewController = [[[ViewController alloc] init] autorelease];
        [self.navigationController pushViewController:viewController animated:YES];
    }
    else if (indexPath.section == 0 && indexPath.row == 1)
    {
        [self.navigationController popToViewController:[self.navigationController.viewControl
            objectAtIndex:1] animated:YES];
    }
    else if (indexPath.section == 0 && indexPath.row == 2)
    {
        [self.navigationController popToRootViewControllerAnimated:YES];
    }
}
```

We may also explore the pop methods

Traveling between View Hierarchy



In most of the case, we has different views in the view hierarchy. Let's create one.

Title and Color of Nav Bar

```
self.navigationItem.title = @"Home Screen";


// Configure the navigation bar background color
self.navigationController.navigationBar.tintColor = [UIColor blackColor];
```

A quick note, we can set the title and tint color of the navigation controller.

Traveling between View Hierarchy

Step1ViewController.m

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.

    self.dataArray = [NSArray arrayWithObjects:@"澳門", @"氹仔", @"路環", nil];

    self.navigationItem.title = @"Choose a place";
}
```

In the Step1ViewController, we setup the similar table view and data source. In this example, we prepare 3 places for the cell.

Traveling between View Hierarchy

Step 1 ViewController.m

```
#pragma mark - Table View Delegates

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    NSString *selectedPlace = [self.dataArray objectAtIndex:indexPath.row];
    NSLog(@"Selected Place: %@", selectedPlace);
}
```

When selecting any row, we display the selected value.
We'll come back to this code later.

Traveling between View Hierarchy

ViewController.m

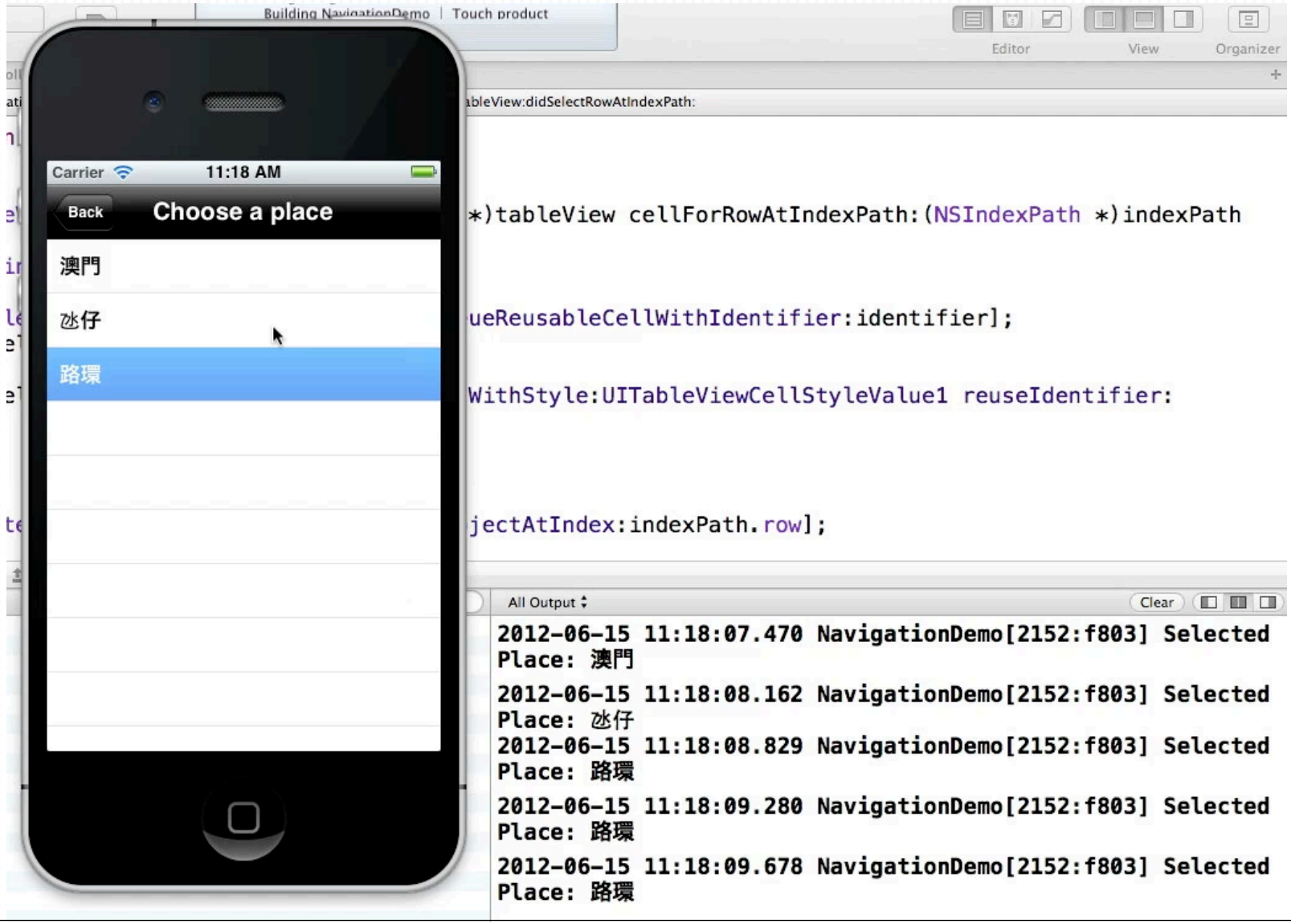
```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    if (indexPath.section == 0 && indexPath.row == 0)
    {
        // Contsruct another view controller here.
        Step1ViewController *viewController = [[[Step1ViewController alloc] init] autorelease];
        [self.navigationController pushViewController:viewController animated:YES];
    }
    else if (indexPath.section == 0 && indexPath.row == 2)
    {
        [self.navigationController popToRootViewControllerAnimated:YES];
    }
}
```

Afterwards, we can change to push the Step1ViewController.

Traveling between View Hierarchy

The result of pushing to a different view controller.

Traveling between View Hierarchy



Traveling between View Hierarchy

Step1ViewController.m

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    NSString *selectedPlace = [self.dataArray objectAtIndex:indexPath.row];
    NSLog(@"Selected Place: %@", selectedPlace);

    [self.navigationController popViewControllerAnimated:YES];
}
```

Let's come back to the Step1ViewController.
We can use pop the view controller after we choose any place option.

Saving and Reading data with NSUserDefault

Saving and Reading Data with NSUserDefault

- NSUserDefault stores object in file in key/value pair
- It is actually a XML plist file
- It is prefect for saving settings, NSDictionary based data, NSArray data.

Saving and Reading Data with NSUserDefault

Step1ViewController.m

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    NSString *selectedPlace = [self.dataArray objectAtIndex:indexPath.row];
    NSLog(@"Selected Place: %@", selectedPlace);

    // save the selection
    [[NSUserDefaults standardUserDefaults] setValue:selectedPlace forKey:@"SelectedPlace"];

    // and force the OS to save the changes to disk.
    [[NSUserDefaults standardUserDefaults] synchronize];

    [self.navigationController popViewControllerAnimated:YES];
}
```

Back to the Step1ViewController, we save the NSString of selected place into NSUserDefault

Saving and Reading Data with NSUserDefault

[NSUserDefaults sharedUserDefaults] synchronize]

This force the iPhone to actually write the data into the file instead of putting them in RAM. Calling this can ensure that we can restore the data from the file later.

Saving and Reading Data with NSUserDefault

ViewController.m

```
// reset the detail text first.
cell.detailTextLabel.text = @"";

if (indexPath.section == 0)
{
    if (indexPath.row == 0)
    {
        cell.textLabel.text = @"選擇地點";
        cell.detailTextLabel.text = [[NSUserDefaults standardUserDefaults] objectForKey:
            @"SelectedPlace"];
    }
    else if (indexPath.row == 1)
    {
        cell.textLabel.text = @"Cell 2";
    }
    else if (indexPath.row == 2)
    {
        cell.textLabel.text = @"Back to Home";
    }
}
```

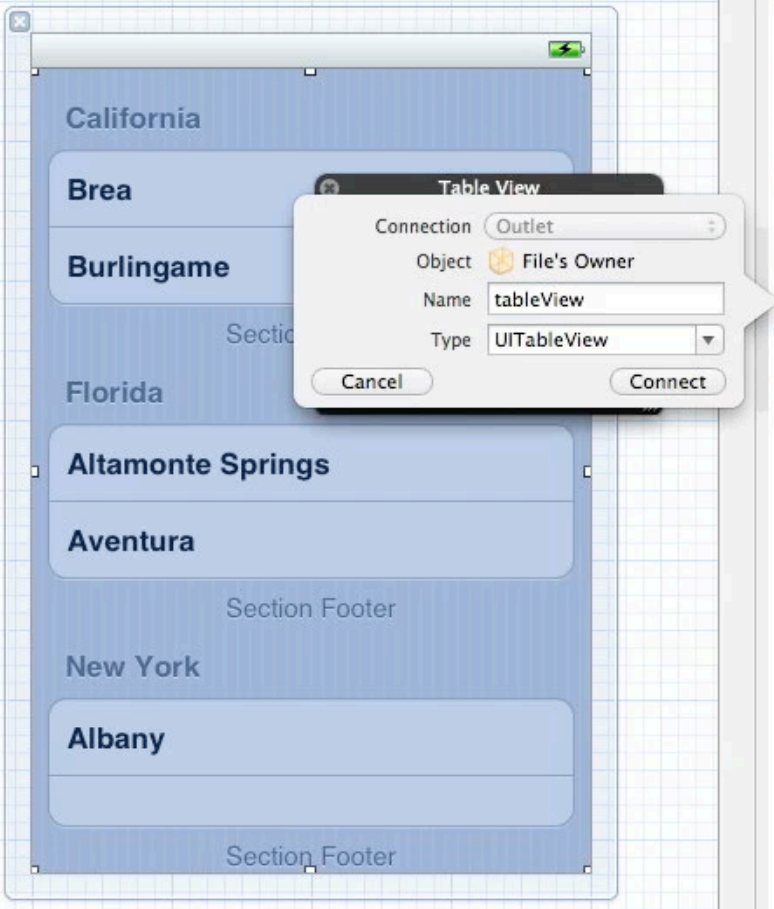
When we are setting up the main view, we can display the saved place choice.

Saving and Reading Data with NSUserDefault

- But we are not done yet, we need to reload the table every time when we chose a place and back to the main view.
- We can call [tableView reloadData] to redraw the table cell. (drawing table cell is under DataSource protocol)
- So we need the IBOutlet of table view to access it within the class.

Saving and Reading Data with NSUserDefault

ViewController.xib




```
//
//  ViewController.h
//  TableViewDemo
//
//  Created by Freshman on 6/8/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController <
    UITableViewDelegate, UITableViewDataSource>
@property (retain, nonatomic) IBOutlet UITableView *tableView;

@end
```

Link the table view as IBOutlet.

Saving and Reading Data with NSUserDefault

ViewController.m

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    // we need to call this every time when the view shows.
    self.navigationController.navigationBarHidden = NO;

    // reload the table view every time we back from navigation controller
    [self.tableView reloadData];
}
```

When the view controller appears again, we reload the data of table view.

Saving and Reading Data with NSUserDefault

Try quitting the app and launching it again. The place choice restores.

Bar Button Item

Bar Button Item

ViewController.m

```
// setup the top right button
self.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc] initWithTitle:@"About" style:
    UIBarButtonItemStyleBordered target:self action:@selector(tappedAbout)];
```

The UIBarButtonItem is a special button that only available on navigation toolbar.

Bar Button Item

ViewController.m

```
// setup the top right button
UIBarButtonItem *item1 = [[UIBarButtonItem alloc] initWithTitle:@"About" style:
    UIBarButtonItemStyleBordered target:self action:@selector(tappedAbout)];
UIBarButtonItem *item2 = [[UIBarButtonItem alloc] initWithTitle:@"About" style:
    UIBarButtonItemStyleBordered target:self action:@selector(tappedAbout)];

self.navigationItem.rightBarButtonItems = [NSArray arrayWithObjects:item1, item2, nil];
```

We can put more than one item on the navigation bar

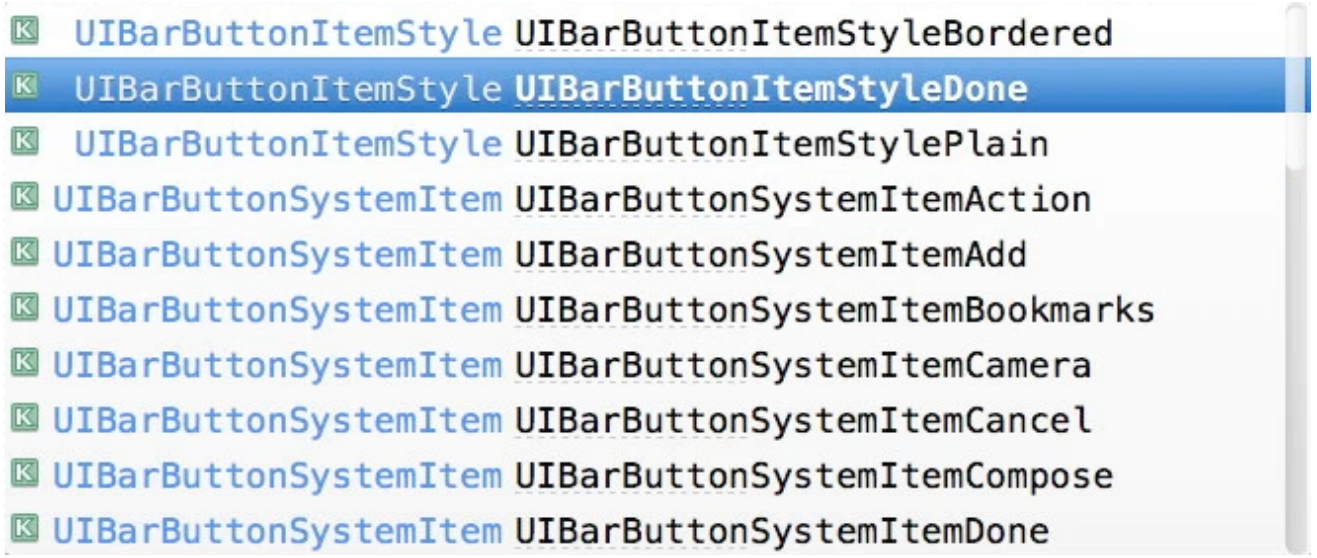
Bar Button Item

The result with two bar item buttons putting together.

Bar Button Item

And we can fill up the navigation bar with buttons, if you need them.

Bar Button Item



There are different predefined item style to choose from

Bar Button Item

For example, we can use a system camera icon.

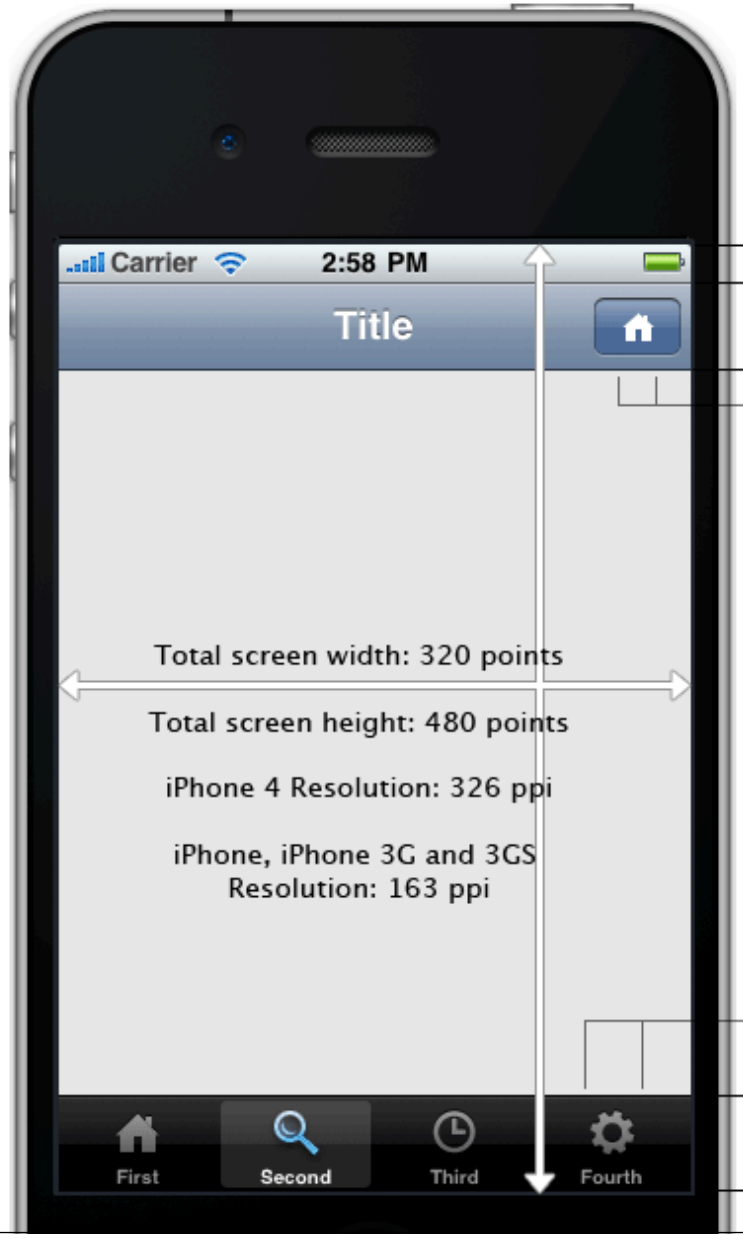
Bar Button Item

And a system trash icon

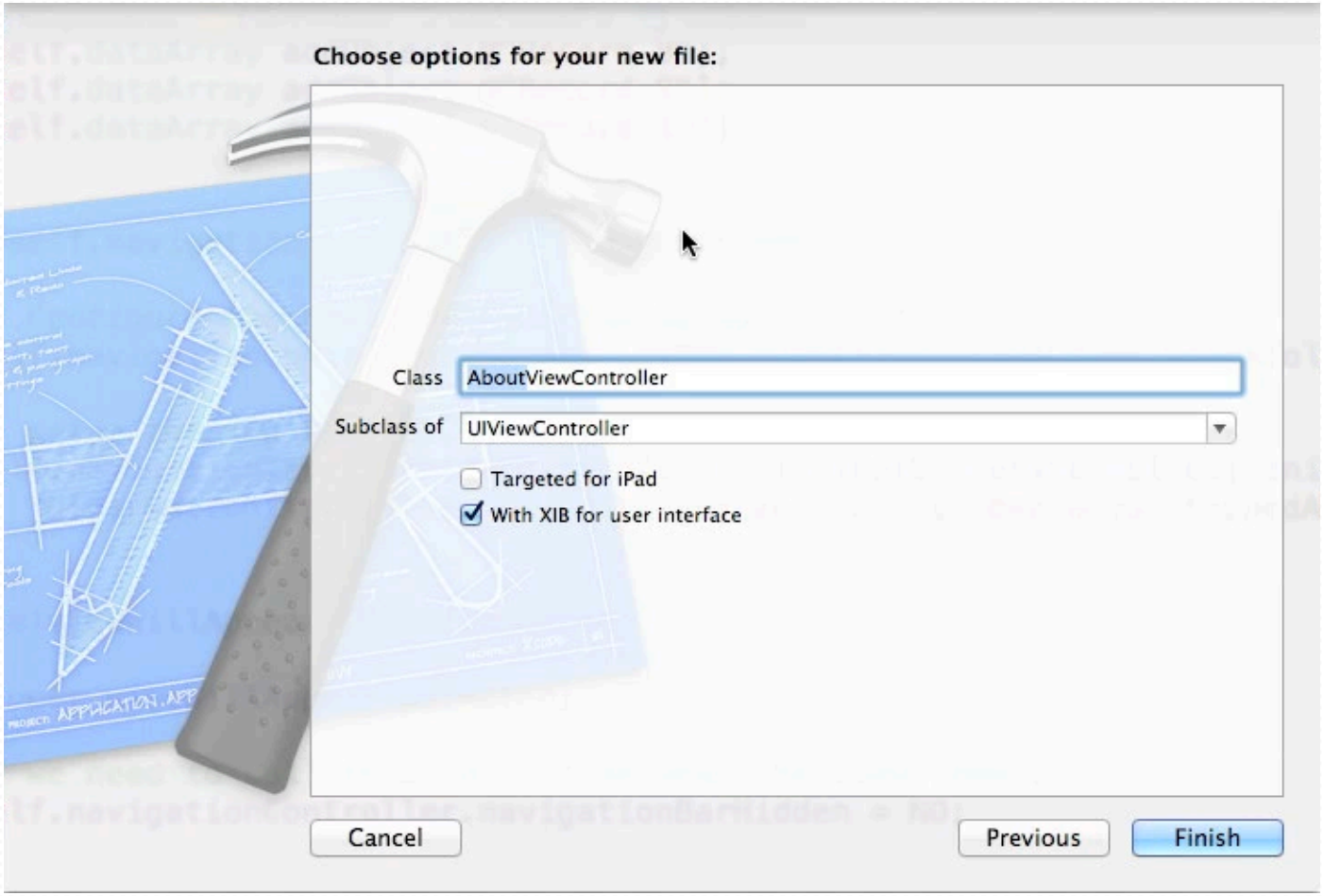
Bar Button Item

The navigation bar gets narrower in landscape view

Bar Button Item

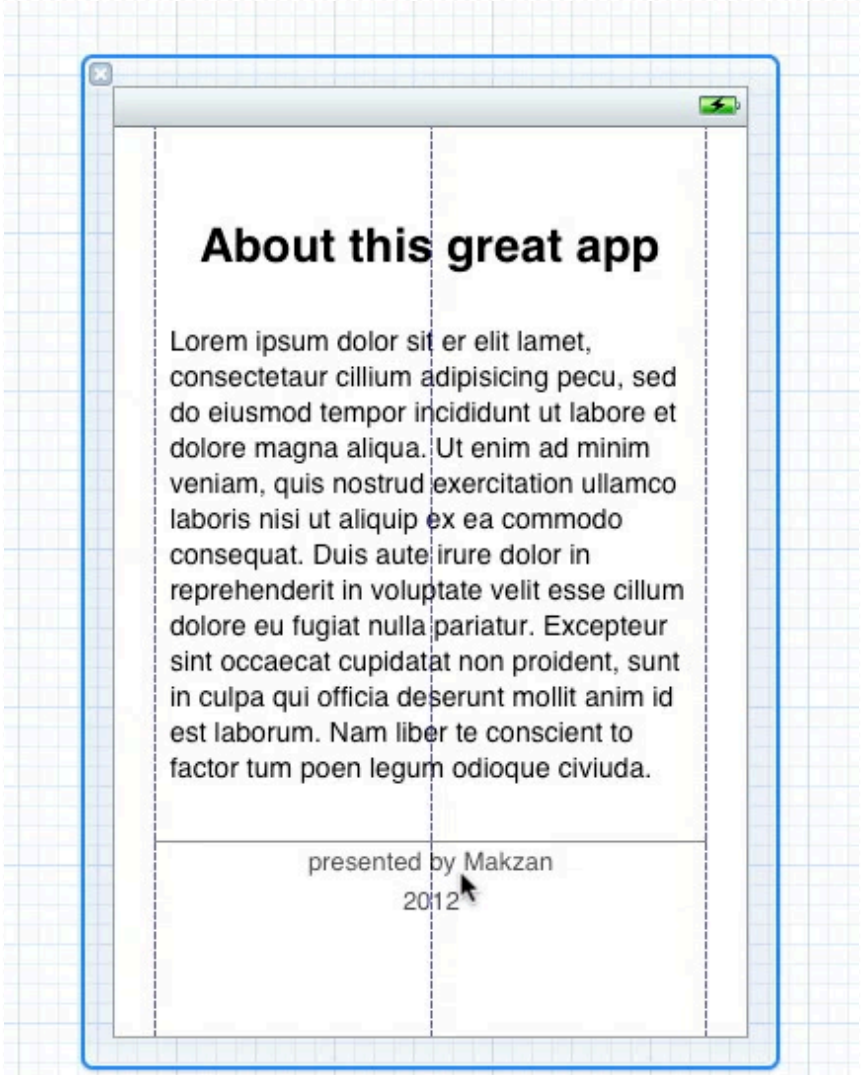


Bar Button Item



Let's create an About view for the about bar item.

Bar Button Item



Put some content in the About view

Bar Button Item

ViewController.m

```
#import "ViewController.h"
#import "Step1ViewController.h"
#import "AboutViewController.h
@inter
@prope                                          tableArray *dataArray;
@end

@implementation ViewController
@synthesize dataArray = _dataArray;

- (void)viewDidLoad
{
```

AboutViewController.h
AboutViewController.m
AboutViewController.xib

We need to import it before using the AboutViewController

Bar Button Item

ViewController.m

```
#pragma mark - Bar Buttons Actions

- (void)tappedAbout {
    AboutViewController *aboutViewController = [[[AboutViewController alloc] init] autorelease];
    [self.navigationController pushViewController:aboutViewController animated:YES];
}
```

When the about icon is tapped, we push the newly created AboutViewController

Bar Button Item

It works now.

How about now we want to show it from bottom to top instead of showing from right to left?

Presenting Modal View Controller

Presenting Modal View Controller

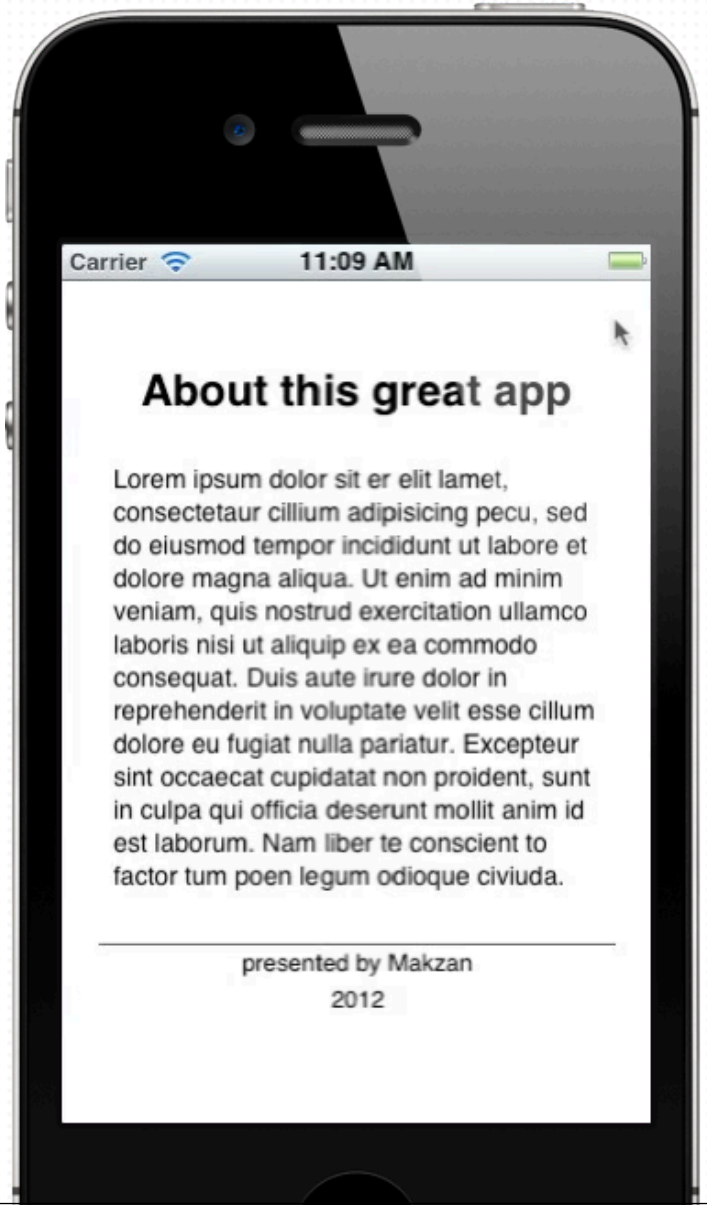
ViewController.m

```
- (void)tappedAbout {
    AboutViewController *aboutViewController = [[[AboutViewController alloc] init] autorelease

    // show the about as modal view
    [self presentModalViewController:aboutViewController animated:YES];
}
```

Let's come back to the tappedAbout method. This time we are not pushing it into view hierarchy. Instead, we present it as a modal view.

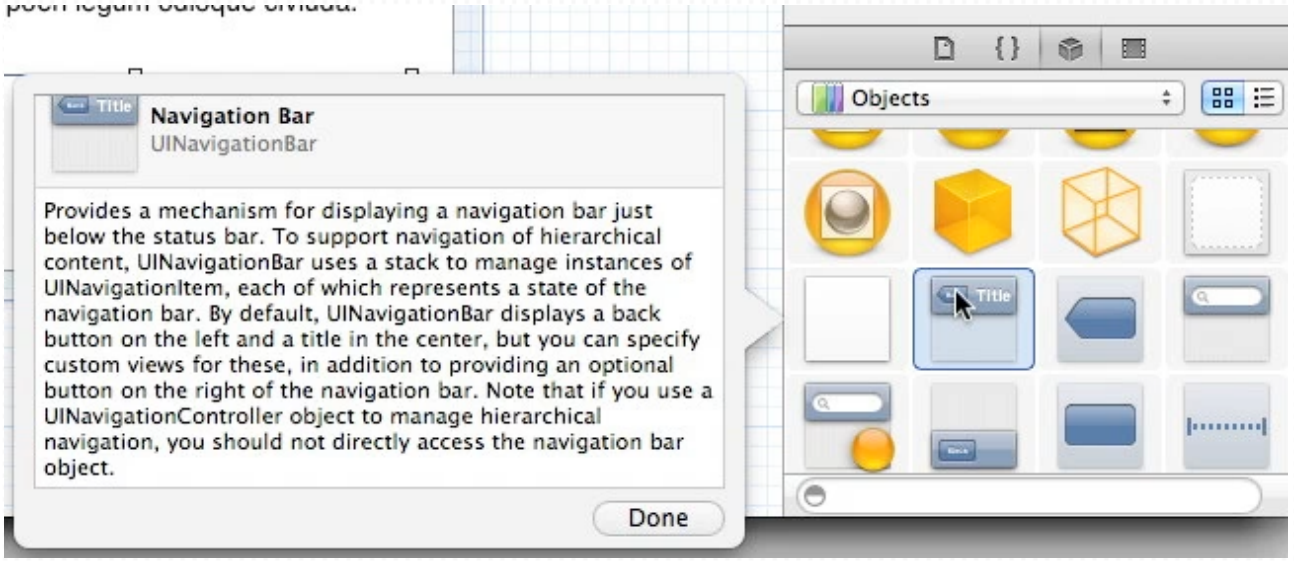
Presenting Modal View Controller

The result looks nice but we do not have anyway to exit this view.



Presenting Modal View Controller

AboutViewController.xib



Sometimes, we can add the Navigation Bar directly in xib interface.

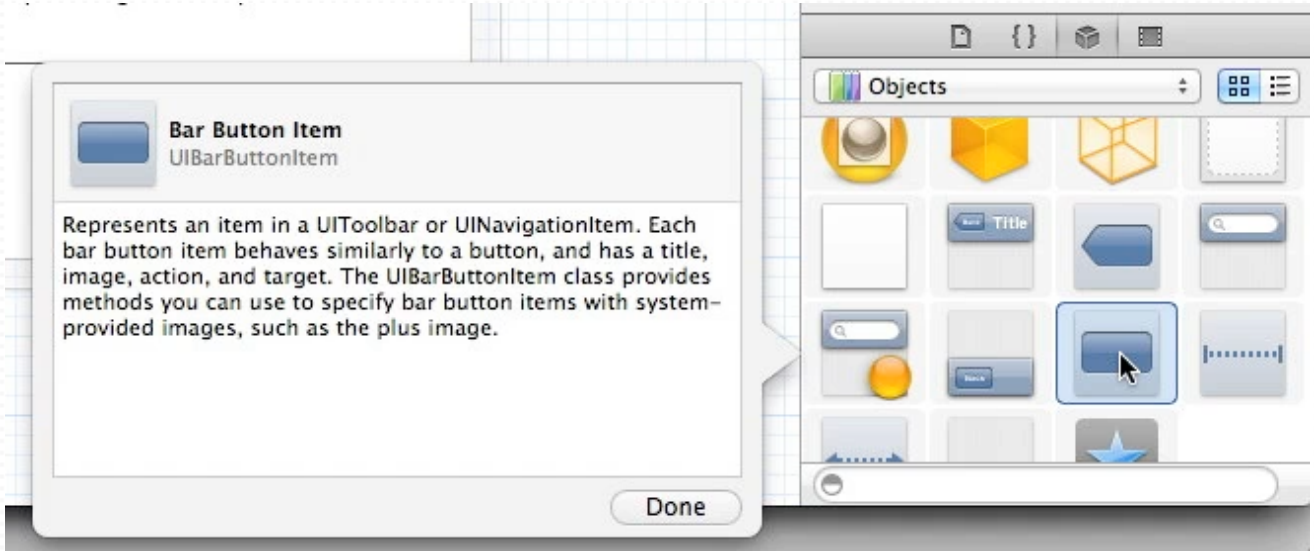
Presenting Modal View Controller

AboutViewController.xib

Drag the Navigation Bar to the top of the view

Presenting Modal View Controller

AboutViewController.xib



Next, we need to identify the Bar Button Item

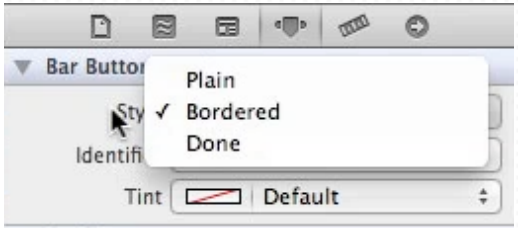
Presenting Modal View Controller

AboutViewController.xib

And drag to add it in either left or right side.

Presenting Modal View Controller

AboutViewController.xib

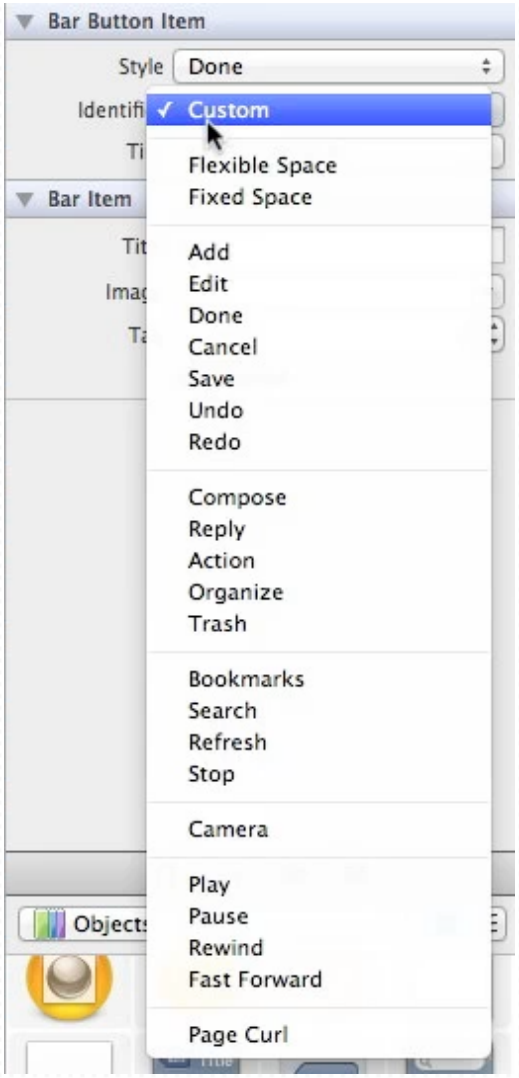


There are few styles of the button that we can choose

Presenting Modal View Controller

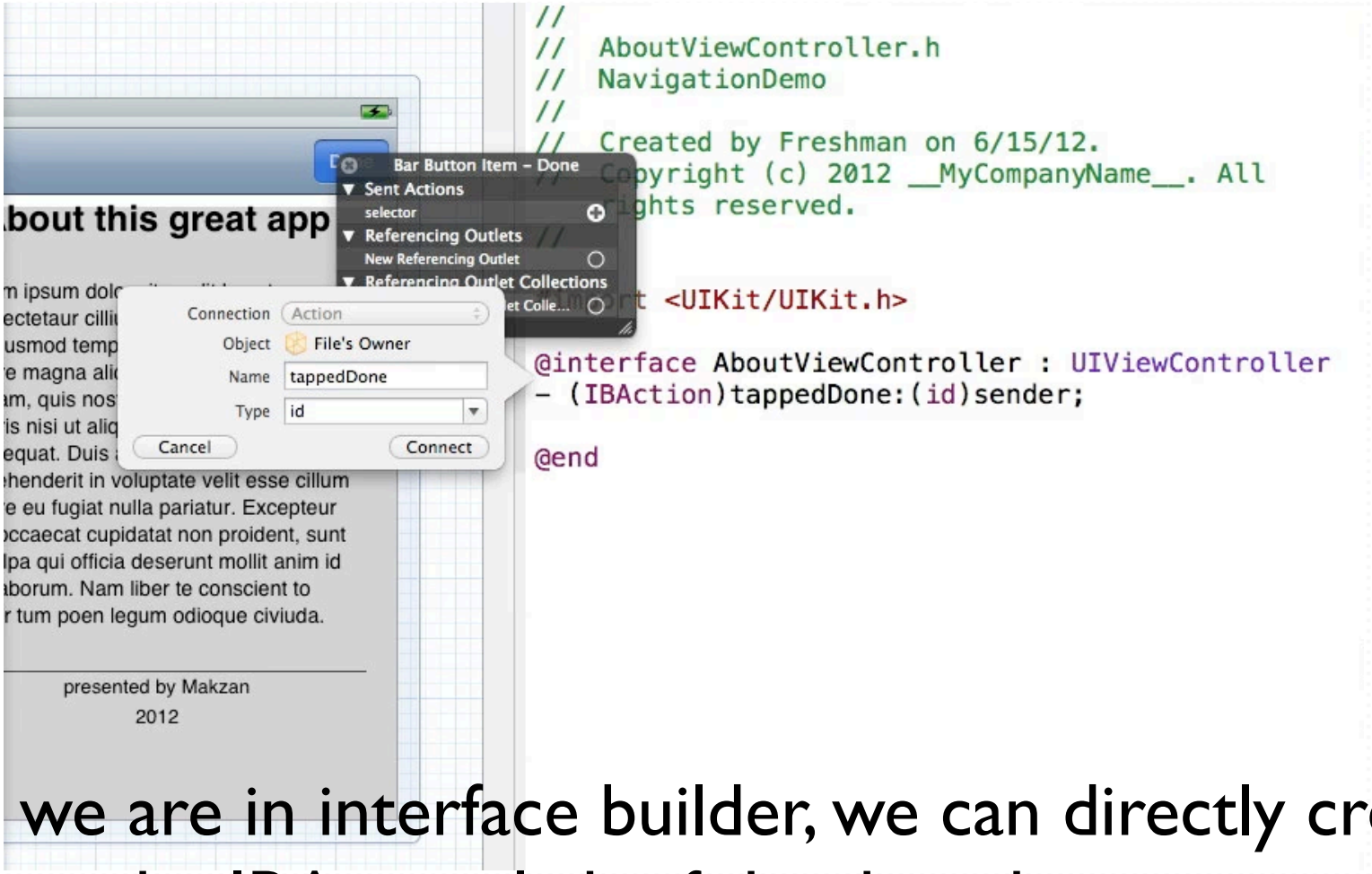
AboutViewController.xib

There are many predefined text and icons that we can use



Presenting Modal View Controller

AboutViewController.xib




```
//
//  AboutViewController.h
//  NavigationDemo
//
//  Created by Freshman on 6/15/12.
//  Copyright (c) 2012 __MyCompanyName__. All
//  rights reserved.
//

#import <UIKit/UIKit.h>

@interface AboutViewController : UIViewController
- (IBAction)tappedDone:(id)sender;

@end
```

Since we are in interface builder, we can directly create the IBAction link of the done button.

Presenting Modal View Controller

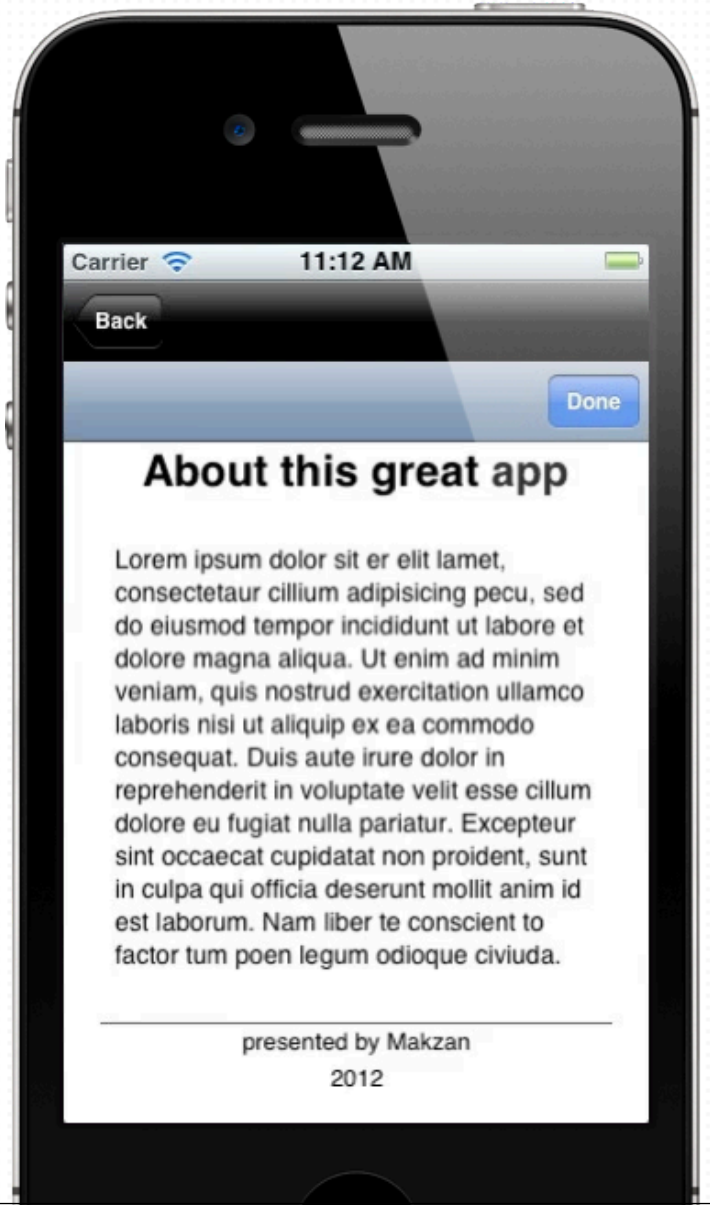
AboutViewController.m

```
- (IBAction)tappedDone:(id)sender {
    [self dismissModalViewControllerAnimated:YES];
}
```

When the done button is tapped, we dismiss the modal view and return to what we did before presenting the modal view.

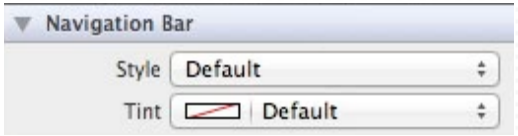
Presenting Modal View Controller

Remember that when we have a navigation bar in xib, it can be strange when pushing into view hierarchy.



Presenting Modal View Controller

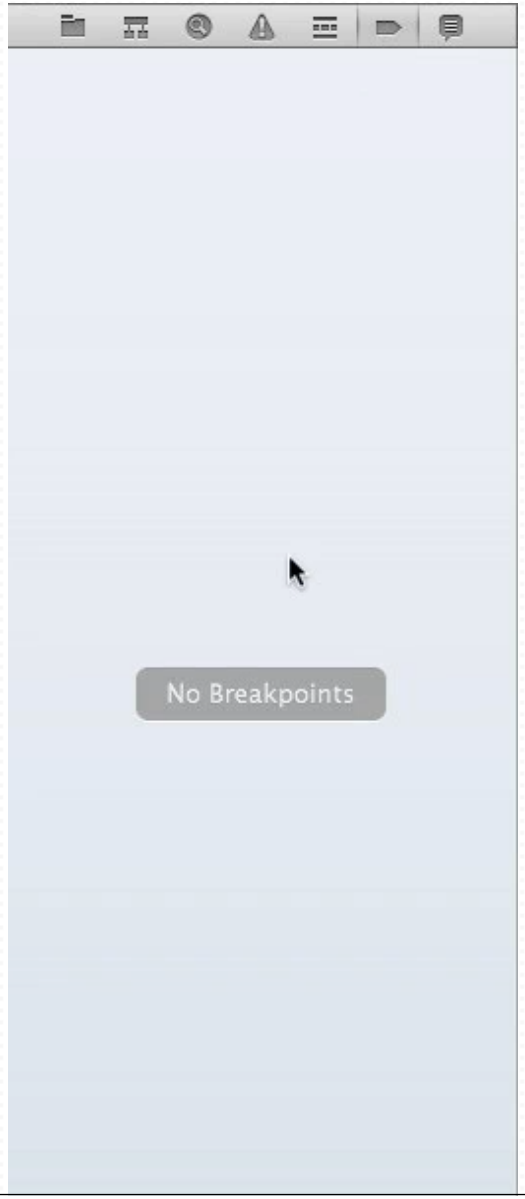
AboutViewController.xib



And we forgot the change the tint color in the About view. In navigation controller, we can choose the tint color directly.

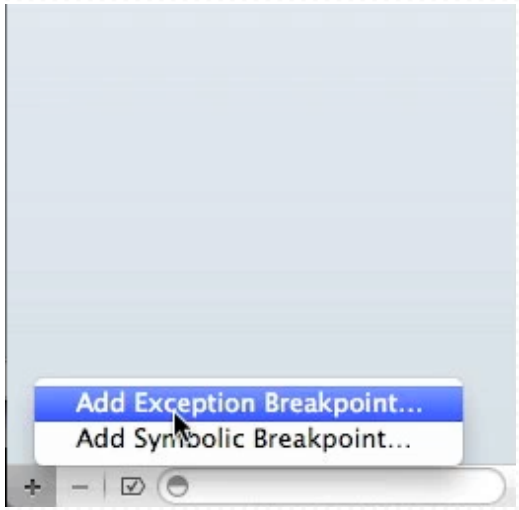
Debugging with Exceptional Break Point

Debugging with Exceptional Break Point



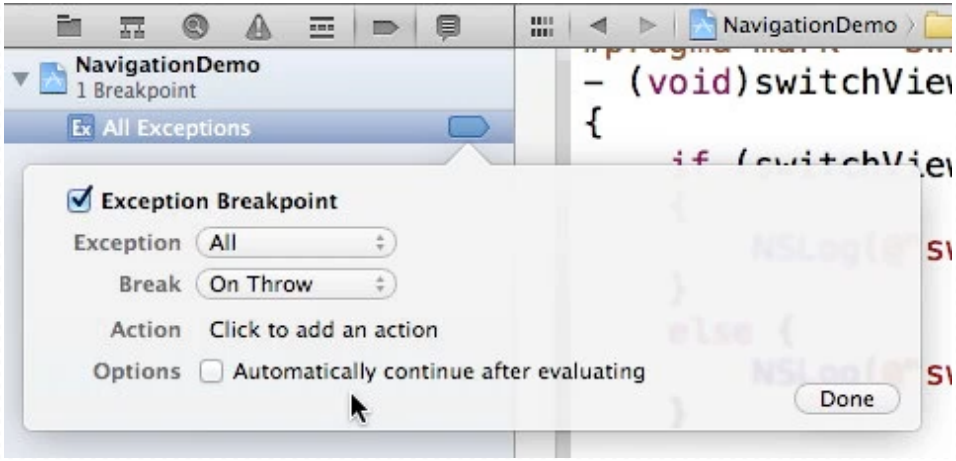
Choose the 6th tab on the left side pane

Debugging with Exceptional Break Point



At the bottom left, Choose Add Exception Breakpoint

Debugging with Exceptional Break Point



Just use the default setting and done.

Debugging with Exceptional Break Point

- Sometimes, when an error crashes the app, the XCode points the issue location to main.m file instead of relevant code.
- This exception break point is useful to help XCode pointing the issue to the correct place that causes the issue.

Conclusion

- Introduced the UINavigationController
- Traveling between the view hierarchy
- Saving and reading data with NSUserDefault
- Bar Button Item
- Presenting Modal view controller
- Debugging with exceptional break point