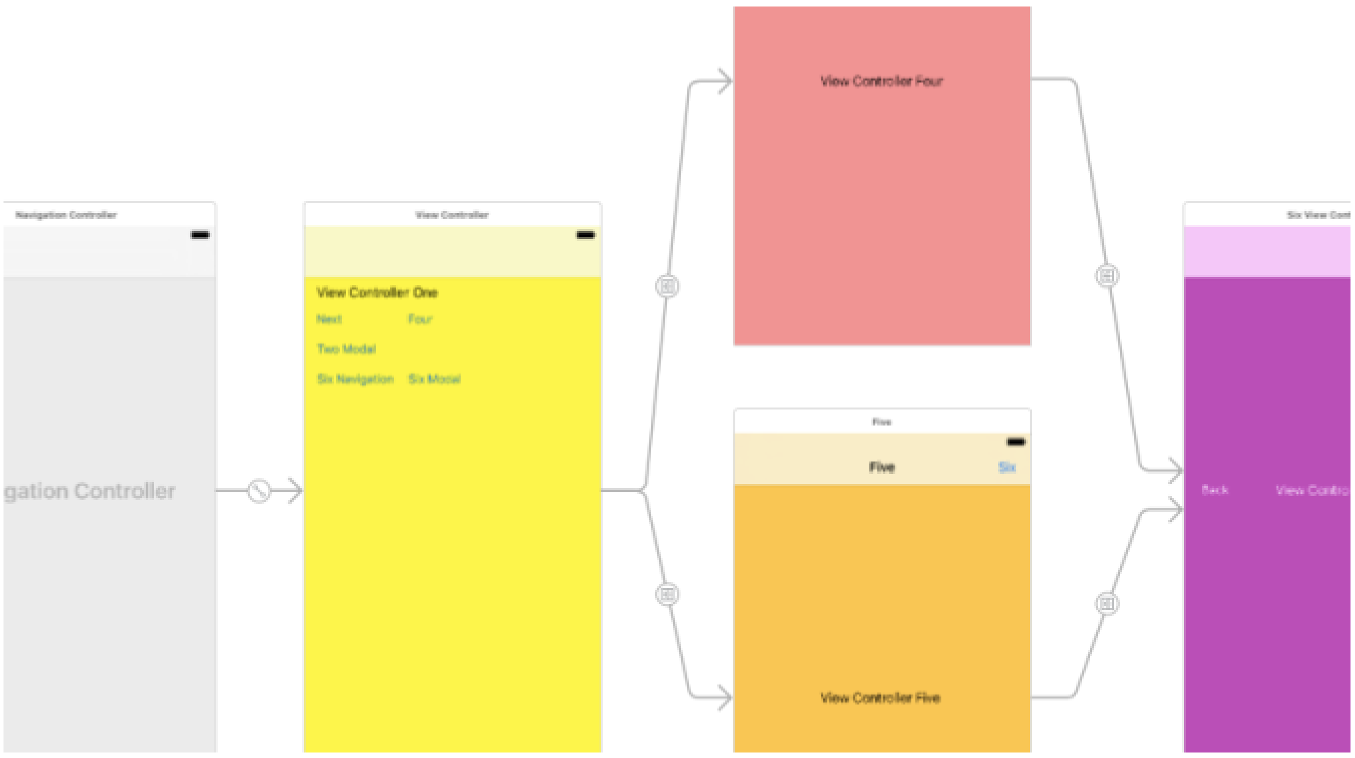
Making App Pie

Tag Archives: UINavigationController in Swift



TUTORIAL

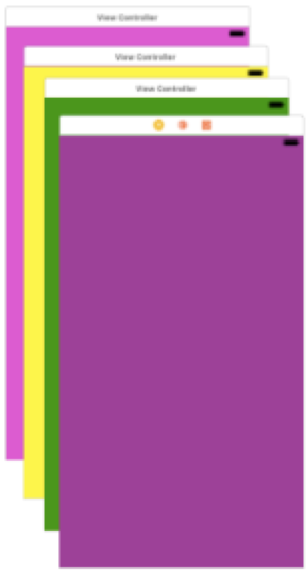
Programmatic Navigation View Controllers in Swift 3.0

JULY 11, 2016 | STEVEN LIPTON | LEAVE A COMMENT

Navigation controllers are the workhorse of organizing view controllers. I've covered much of their use in other posts about MVC (https://makeapppie.com/2014/08/04/the-swift-swift-tutorial-why-do-we-need-delegates/), segues and delegates (https://makeapppie.com/2016/06/27/using-segues-and-delegates-for-navigation-controllers-in-swift-3-0/). In this chapter, we'll go through some of the Swift code for the Navigation controller.

The View Controller Stack

Navigation view controllers are stack based. The newest view controller is the visible one. It is on top of the last one we saw.
If you are not familiar with the stack data structure, it is useful to understand stacks and their nomenclature a little better. Stacks work a lot like a deck of cards that are face up.



(https://apppie.files.wordpress.com/2016/07/2016-07-06_06-20-22.png)

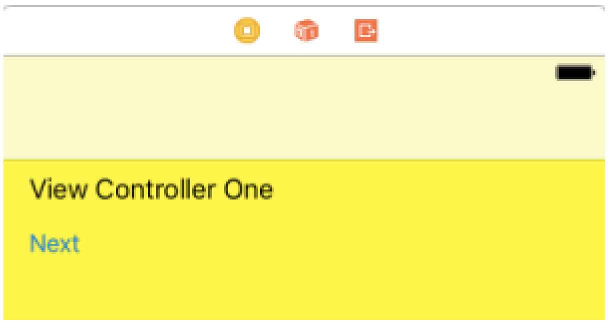
You can only see the top card. When you place a card on top of the stack, you *push* a card on the stack. When you remove a card from the stack and show the card below it, you *pop* it off the stack. We use the terms push and pop a lot to talk about stacks. The two major methods we'll be talking about `popViewController` and `pushViewController` use this nomenclature to explain what they do.

Opening a View Controller in a Xib

Let's look at a few ways to programmatically move through view controllers by pushing and popping to the navigation stack directly.
Start a new single view project in Swift called **SwiftProgNavControllerDemo**. Go into the storyboard. Make sure you have a **iPhone 6s** selected for **View as**: in the lower left of the storyboard"

(https://apppie.files.wordpress.com/2016/07/2016-07-05_06-18-15.png)

Select the blank view controller. Be sure to select the controller by clicking the (https://apppie.files.wordpress.com/2016/07/view-controller-icon.png) icon and not the view.
From the drop down menu select **Edit>Embed in > Navigation Controller**.
In the view controller, add a label and a button so your code looks like the diagram below. If you wish you can also color the background:



(https://apppie.files.wordpress.com/2016/07/2016-07-06_07-49-58.png)

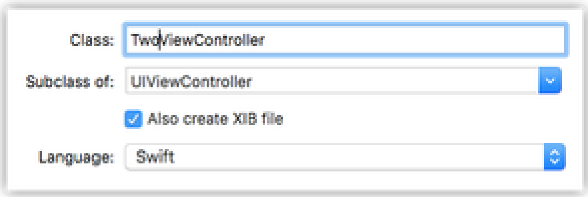
Open the assistant editor. Control-drag the button and make an action for a `UIButton` called `nextButton`. Remove everything else from the class for now. Add the following two lines to the `nextButton`

```
let vc = TwoViewController(
     nibName: "TwoViewController",
     bundle: nil)
navigationController?.pushViewController(vc,
     animated: true)
```

Line 1 creates a view controller of class `TwoViewController`, using the XIB of the same name. Line 2 pushed the view controller on the navigation controller stack maintained by `ViewController`. Your code should look like this when done:

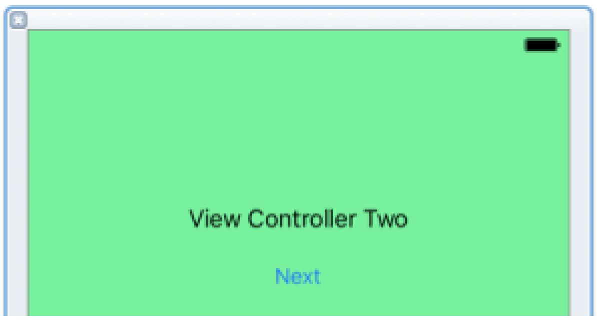
```
class ViewController: UIViewController {
    @IBAction func nextButton(_ sender: UIButton){
    let vc = TwoViewController(
        nibName: "TwoViewController",
        bundle: nil)
     navigationController?.pushViewController(vc,
        animated: true)
    }
}
```

We need another view controller as a destination. Press **Command-N** or click **File>New>File…** Choose a **iOS source** template of **Cocoa Touch** Class. Make the new file a subclass of `UIViewController` and name the file **TwoViewController**. We'll work with a xib for our destination controller. Check on the option **Also create XIB file**.


Class: TwoViewController
Subclass of: UIViewController
☑ Also create XIB file
Language: Swift

(https://apppie.files.wordpress.com/2016/07/2016-07-06_07-45-10.png)

You will find a new xib in interface builder. Set it up to look like the illustration below.



(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-44-23.png)

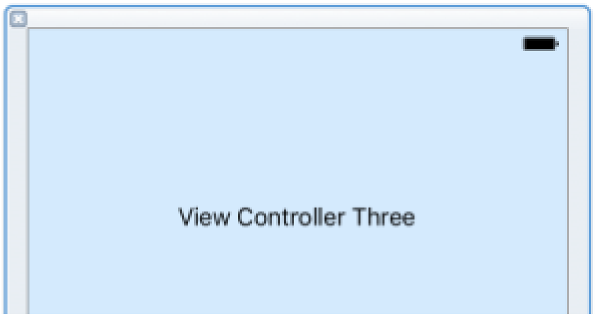
In the assistant editor, remove everything to have an empty class. Control-drag the **Next** button inside the `TwoViewController` class. Make an `@IBAction` method named `nextButton` as an `UIButton`. Add the following code to the `nextButton()` method:

```
let vc = ThreeViewController(
            nibName: "ThreeViewController",
            bundle: nil)
        navigationController?.pushViewController(vc,
                animated: true )
```

Your code should look like this:

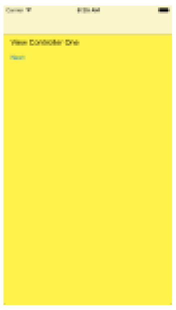
```
class TwoViewController: UIViewController {

    @IBAction func nextButton(_ sender: UIButton) {
        let vc = ThreeViewController(
            nibName: "ThreeViewController",
            bundle: nil)
        navigationController?.pushViewController(vc,
            animated: true )
    }
}
```

Let’s do this one more time so we end up with three view controllers to push onto the view controller stack. Follow the same procedure as you did for `TwoViewController`, but name it `ThreeViewController.` Set the view to look like this:

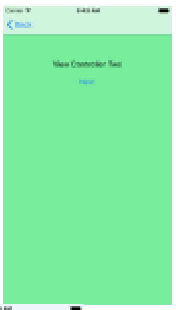


(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-44-43.png)

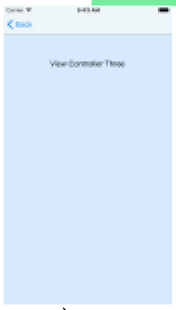
There’s no buttons here, so you have no actions to set. Change your simulator to **iPhone 6s**. Build and run. Tap the **Next** button and move between the three view controllers.



(https://apppie.files.wordpress.com/2016/07/2016-07-06_08-25-30.png)



(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-43-07.png)



(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-43-43.png)

Pushing a view controller from a xib is two lines of code:

```
let vc = ViewControllerName(nibName: "nameOfNib", bundle: nil)
navigationController?.pushViewController(vc, animated: true )
```

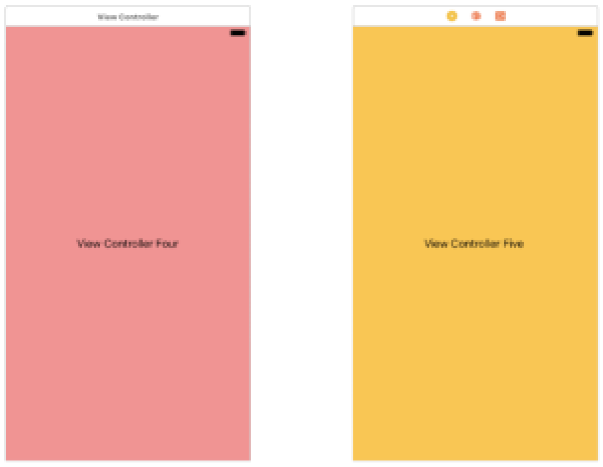
The first line creates the view controller. I tend to keep it simple and use `vc`, though if I had more than one, I'd be more descriptive.
Xibs are probably one of the most common uses for pushing a view controller. There are situations where you cannot use the storyboard and this is a good traditional alternative. Often self-contained reusable modules will use a xib instead of a storyboard.

Programmatic Segues to View Controllers

For most uses, I prefer the storyboard over xibs for two reasons: first it is better documentation of the user interface. Secondly, I prefer to let the system do as much of the background work as possible by using the storyboard. The deeper into code you go, the more you have to worry about unexpected bugs.

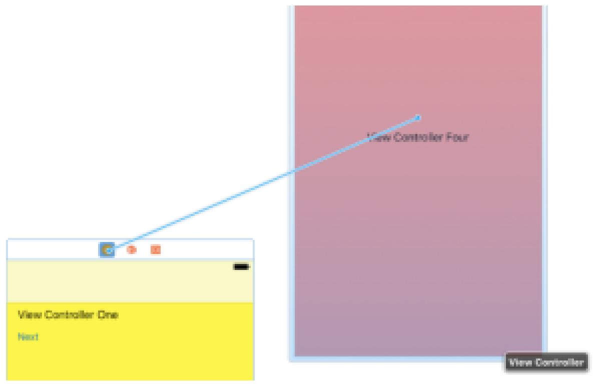
We can programmatically push a view controller from a storyboard in two ways: segues or storyboard identifiers. We'll start with segues. One of the first ways anyone learns to use storyboards is direct segues, connecting up a button directly to a view. You can also do segues programmatically, which is useful when you conditionally go to a view.

Go to the storyboard. Add two more view controllers to the storyboard. Label one **View Controller Four** and the other **View Controller five**.



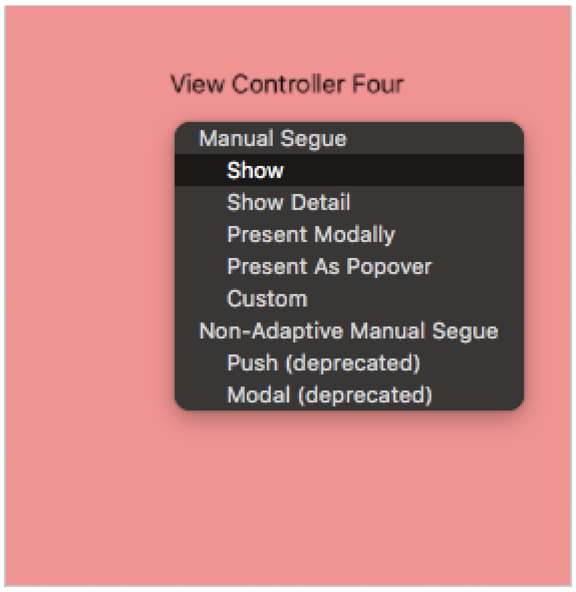
(https://apppie.files.wordpress.com/2016/07/2016-07-07_05-47-25.png)

Click on the `ViewController` scene title in the storyboard. From the view controller Icon (https://apppie.files.wordpress.com/2016/07/view-controller-icon.png)on `ViewController`, control-drag from `ViewController` to somewhere on View Controller Four's content so it highlights.



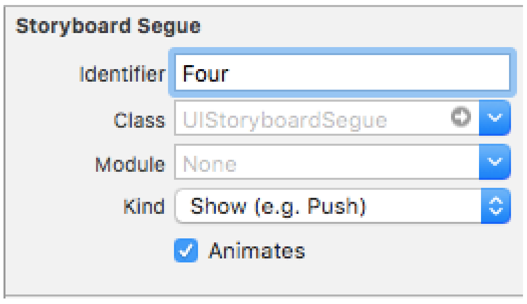
(https://apppie.files.wordpress.com/2016/07/2016-07-07_05-55-32.png)

Release the mouse button. In the menu that appears, select **Show**.



(https://apppie.files.wordpress.com/2016/07/2016-07-07_05-59-44.png)

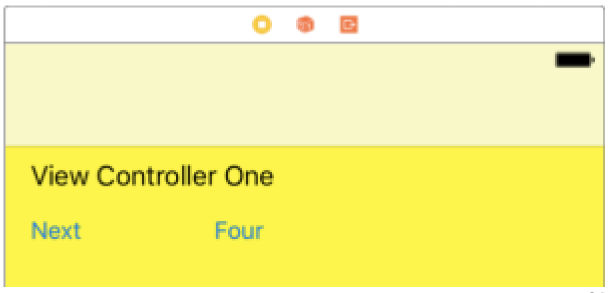
Click on the Show segue icon (https://apppie.files.wordpress.com/2016/07/show-segue-icon.png) to select the segue. Go into the properties inspector and set the segue's **Identifier** to **Four**.



Storyboard Segue
Identifier: Four
Class: UIStoryboardSegue
Module: None
Kind: Show (e.g. Push)
Animates ✓

(https://apppie.files.wordpress.com/2016/07/2016-07-07_06-03-18.png)

Drag another button out to the View Controller scene and make the title **Four**.



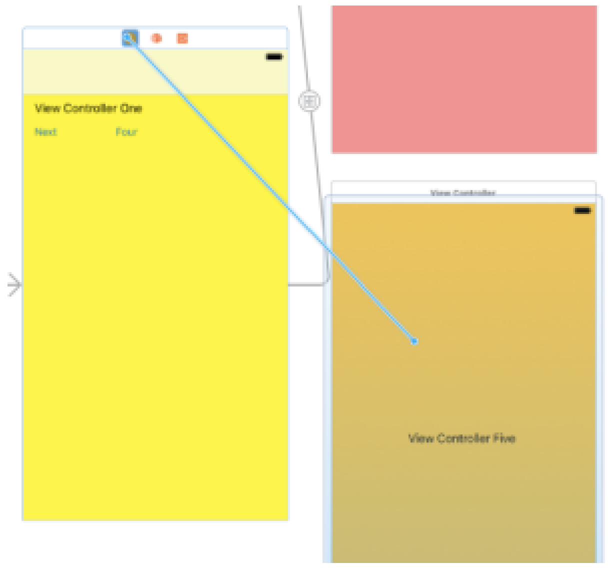
(https://apppie.files.wordpress.com/2016/07/2016-07-07_06-05-28.png)

Go to `ViewController` class and add the following method:

```
@IBAction func fourFiveToggleButton(_ sender: UIButton){
    performSegue(withIdentifier: "Four",
        sender: self)
}
```

Open the assistant editor and drag from the circle next to the `fourButton()` method over the **Four** button and release.
The code above is a mere one line: it runs the segue. If you set the segue identifier correctly, that is all you need to do.

One use is conditional cases. Conditions in the current view controller or model might change. The app may display different view controllers based on those conditions. Let's try a simple example. Just as we did with **View Controller Four**, make a segue with an identifier **Five** by control-dragging from the view controller icon (https://apppie.files.wordpress.com/2016/07/view-controller-icon.png)on **View Controller One** to the view of **View Controller Five**.



(https://apppie.files.wordpress.com/2016/07/2016-07-07_06-18-05.png)

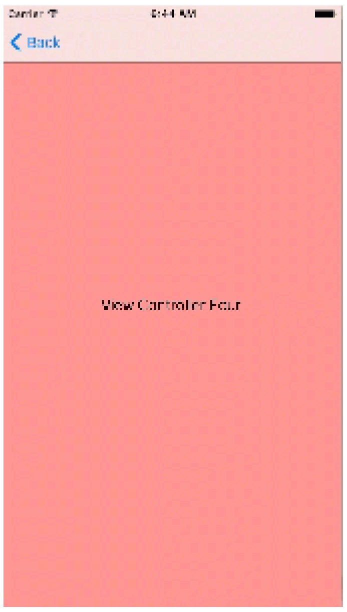
Select a **Show** segue. Select the segue by clicking the show segue icon (https://apppie.files.wordpress.com/2016/07/show-segue-icon.png). In the attributes inspector change the Identifier to **Five**.

Change the code for `fourFiveToggleButton` to this

```
@IBAction func fourFiveToggleButton(_ sender: UIButton){
    let normal = UIControlState(rawValue: 0) //beta 1 has no .normal
    if sender.titleLabel?.text == "Four"{
        performSegue(withIdentifier: "Four",
                     sender: self)
        sender.setTitle("Five", for: normal)
    } else{
        performSegue(withIdentifier: "Five",
                     sender: self)
        sender.setTitle("Four", for: normal)
    }
}
```

The code checks the contents of the title label. If the label is **Four** it goes to **View Controller Four** and changes the title label. If the label is **Five**, it goes to **View Controller Five** and toggles the label back to **Four**

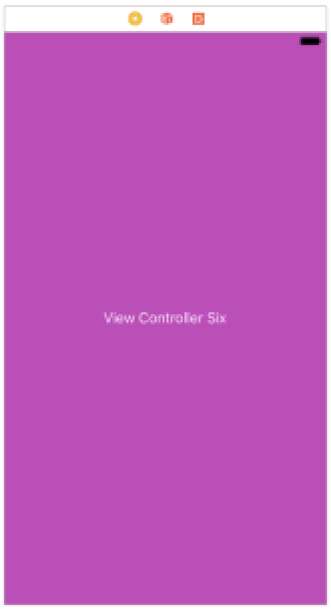
Build and run. we can toggle between the two views.



(https://apppie.files.wordpress.com/2016/07/2016-07-07_06-44-06.gif)

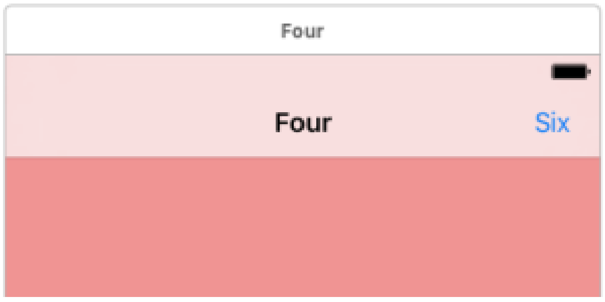
An Interesting Stack Demonstration

Stacks are linear collections. How we compose that collection on the storyboard might be nonlinear. For example, View controllers four and five might both segue into view controller six. On the storyboard drag out another view controller. Set a background color for it. Label it **View Controller Six**



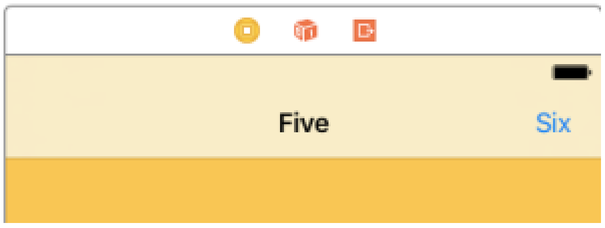
(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-55-47.png)

For this example, we'll use the storyboard directly. On **View Controller Four,** drag a **Navigation Item**. Title the navigation Item **Four**. Drag a bar button item to the navigation controller. Title it **Six**



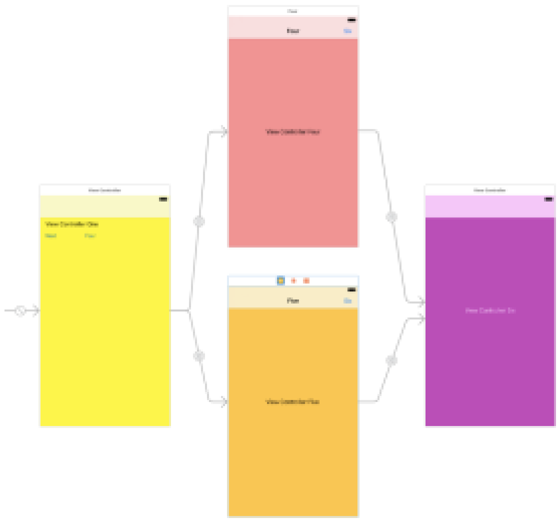
(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-57-26.png)

Do the same for **View Controller Five** so the navigation bar looks like this:



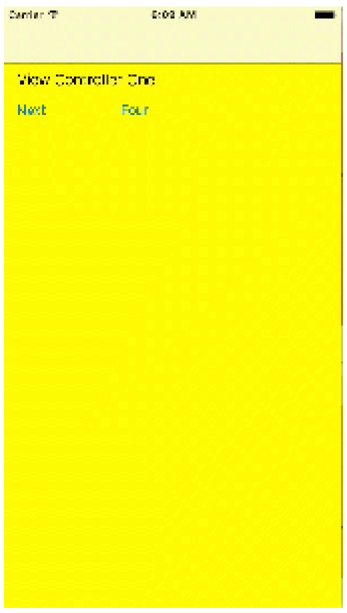
(https://apppie.files.wordpress.com/2016/07/2016-07-08_05-57-47.png)

Control drag from View controller Four's **Six** button to the **Six View Controller**. Select a **Show** Segue. Repeat for the Five View Controller. Control drag from the **Six** button to the **Six View Controller**. Select a Show Segue. Your storyboard now has this:



(https://apppie.files.wordpress.com/2016/07/2016-07-08_06-06-24.png)

We've set a storyboard where we go to **Five** or **Four**, and then go to six. Build and run. We can go to **Six** from both view controllers.

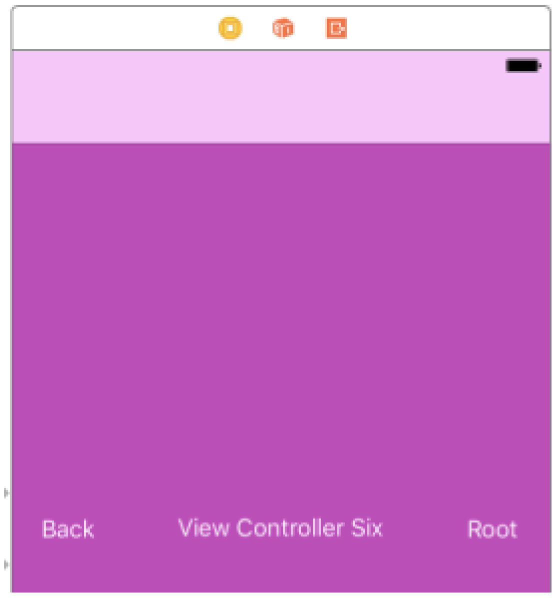


(https://apppie.files.wordpress.com/2016/07/2016-07-08_06-09-46.gif)

Closing a View Controller

Up to now, we've relied on the navigation controller's **Back** button. Dismissal of view controllers with `popViewController()` are common in an application. Almost every delegate between view controllers in a navigation stack will use it. There are several versions of the popping off controller for different uses.

Add two buttons to View Controller Six, titled **Back** and **Root**.



(https://apppie.files.wordpress.com/2016/07/2016-07-08_06-14-23.png)

Press **Command-N** to make a new class called **SixViewController**, subclassing `UIViewController`. Remove all the methods in the class. In the `SixViewController` class, create an action `backButton`:

```
@IBAction func backButton(_ sender: UIButton) {
       navigationController?.popViewController(animated:true)
   }
```

You will get a warning `Expression of type 'UIViewController?' is unused.` Ignore it for now, we'll discuss it later.

Also in `SixViewController,` add to the `rootButton()` method:

```
@IBAction func rootButton(_ sender: UIButton) {
  navigationController?.popToRootViewController(animated:true)
    }
```

Go to the story board and open the assistant editor. Drag from the circle next to `backButton` to the **Back** button. Drag from the circle next to the `rootButton` to the **Root** Button. Build and Run. Go to **Six**. Press the new **Back** button. You go back to Four. Go to **Six** again and press the root button. You go back to **One**.

There are three versions of pop: `popViewController()`, `popToRootController()`, and `popToViewController()` The most Common is `popViewController()` which removes the top view controller from the stack. `popToRootViewController()` and `popToViewController()` pops everything or everything up to a specific view controller off the stack, returning what it popped off.

Because `popViewController` returns a value, we are getting the two warnings. We have to do something with the return value. You'll notice in the error message that we get an optional returned. If `nil`, there were no view controllers to pop off. Use this to make sure you do have a navigation stack under the current controller. Change the class to this:

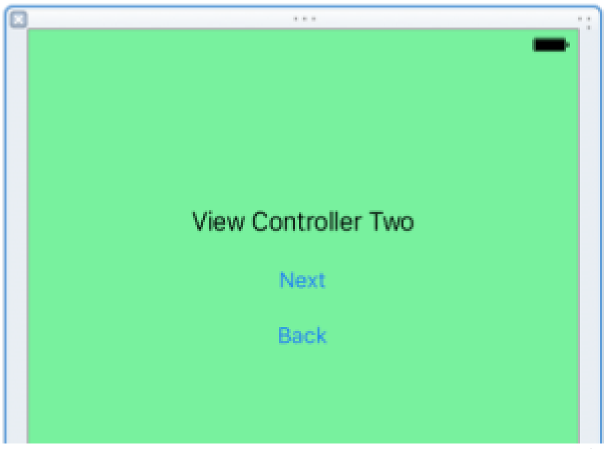
```
class SixViewController: UIViewController {
    @IBAction func backButton(_ sender: UIButton) {
        guard (navigationController?.popViewController(animated:true)
            else {
                print("No Navigation Controller")
                return
            }
    }
    @IBAction func rootButton(_ sender: UIButton) {
        guard navigationController?.popToRootViewController(animated
        else {
            print("No Navigation Controller")
            return
        }
    }

    }
```

By using `guard` and checking for `nil` the application checks to make sure there is a navigation stack. If not, code handles the error. If there is a segue set to **Present modally** by mistake and tries popping off the controller, the error gets handled. This is especially important in two situations: when you use a xib in a navigation controller and when you use a Storyboard ID. Both cases are independent of segues. Both can be used as a modal controller and a navigation controller. It's likely you have modules set up for use in different applications, and sometime they are modal and sometimes navigation controllers. For example, go to the code for `TwoViewController`. Add the following action:

```
@IBAction func backButton(_ sender:UIButton){
       guard navigationController?.popViewController(animated: true)
           print("Not a navigation Controller")
           dismiss(animated: true, completion: nil)
           return
       }
   }
```

We expanded the `guard` clause slightly here from the previous example. If there is no navigation controller, we must be in a modal. Instead of popping the controller, we dismiss it.

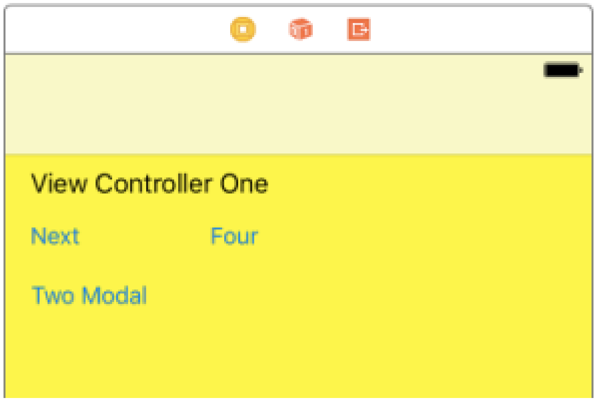
Add a **Back** button to the Two View Controller:



(https://apppie.files.wordpress.com/2016/07/2016-07-08_06-58-13.png)

With the assistant editor open, Control-drag from the **Back** button we created to the `backButton` code.

Add another button to **View Controller One** titled **Two Modal**.

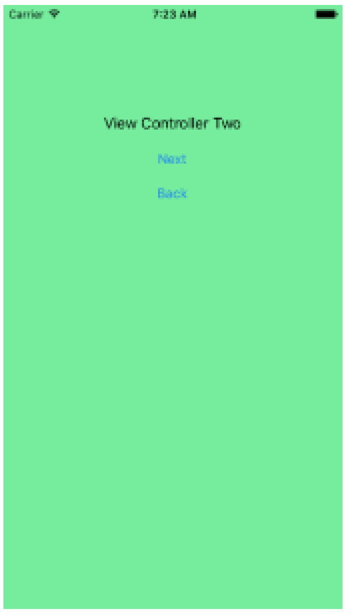


(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-00-43.png)

Open the assistant editor. Control drag the **Two Modal** Button to make a new action named **modalTwoButton**. Add the following code to the new action to present a modal view:

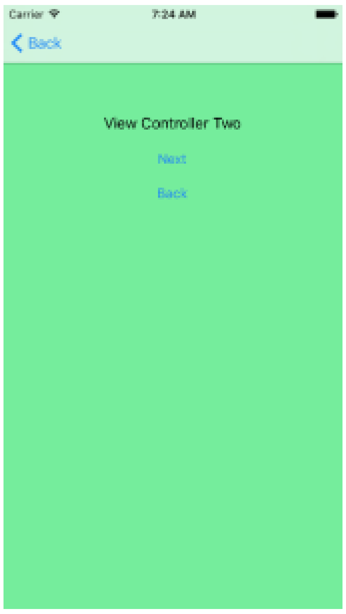
```
@IBAction func modalTwoButton(_ sender: UIButton) {
    let vc = TwoViewController(
        nibName: "TwoViewController",
        bundle: nil)
    present(vc,
        animated: true,
        completion: nil)
    }
```

Build and run. Tap the **TwoModal** Button, and the modal view slides up from the bottom.



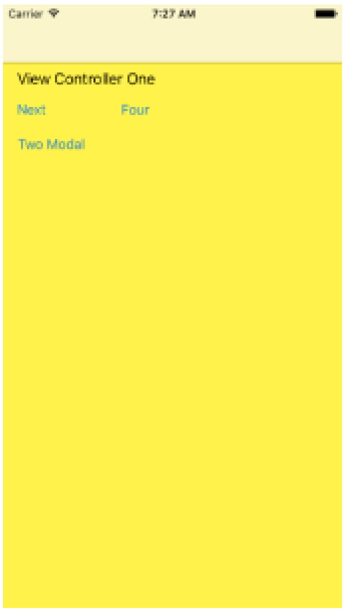
(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-23-36.png)

Tap the **Back** button and it goes back to View Controller one. Tap the **Next** button and you slide sideways into a navigation view.



(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-24-36.png)

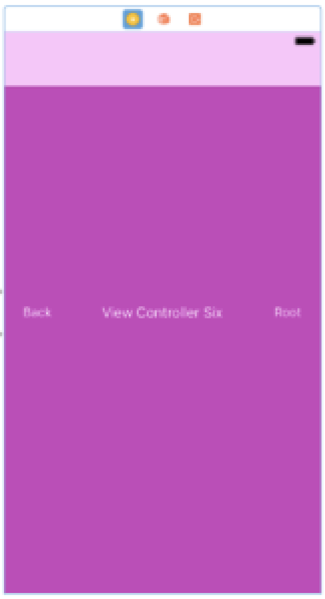
Tap **Back** and you are back in View Controller One



(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-27-12.png)

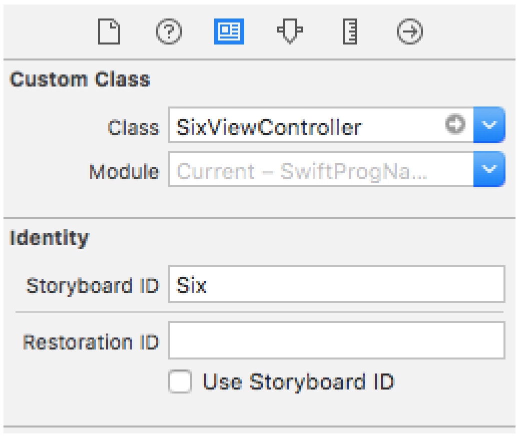
Using Storyboard ID's With Navigation Controllers

In between Xibs and the Storyboard are storyboard ID's. When you want all of your view controllers on one storyboard, but also want to call the view controller from several different controllers you might want to use a storyboard ID. Storyboard ID's can programmatically called both by modals and navigation controllers. Some view controllers might have a segue at one place and called by a Storyboard ID in another. On the storyboard find **View Controller Six**



(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-32-27.png)

In the Identity inspector, set the **Storyboard ID** to **Six**



Custom Class

Class SixViewController

Module Current – SwiftProgNa...

Identity

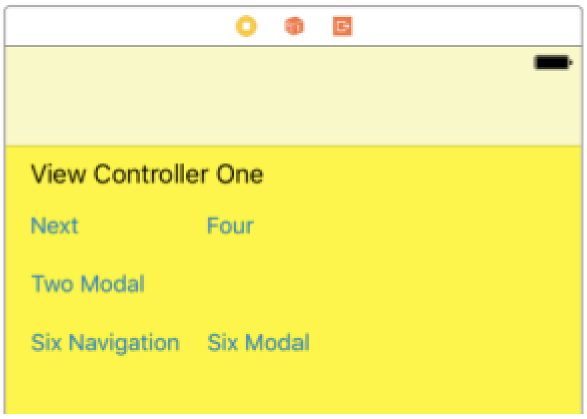
Storyboard ID Six

Restoration ID

Use Storyboard ID

(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-34-45.png)

On **View Controller One** add two more Buttons labeled **Six Navigation** and **Six Modal.**



(https://apppie.files.wordpress.com/2016/07/2016-07-08_07-38-37.png)

Control-Drag the **Six Navigation** Button into the assistant editor set to **Automatic**. Make an action **sixNavigationButton**. Now do the same with the modal button. Control-Drag the **Six Modal** Button into the assistant editor. Make an action **sixModalButton**.

The two actions are very similar. They will get a view controller from the method

```
storyboard?.instantiateViewController(withIdentifier:String)
```

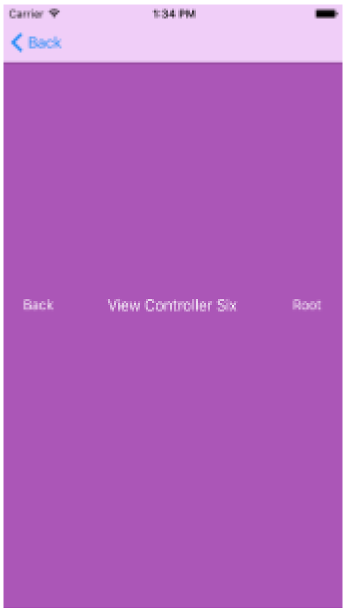
then present it for a modal or push it for a navigation controller. Add this code to the two actions:

```
@IBAction func sixNavigationButton(_ sender: UIButton) {
        guard let vc = storyboard?.instantiateViewController(withIde
            print("View controller Six not found")
            return
        }
        navigationController?.pushViewController(vc, animated: true)
    }

    @IBAction func sixModalButton(_ sender: UIButton) {
        guard let vc = storyboard?.instantiateViewController(withIde
            print("View controller Six not found")
            return
        }
        present(vc, animated: true, completion: nil)
    }
```

Go over to the `SixViewController` class. Add the `dismiss` method to dismiss the modal in the `backButton` action :

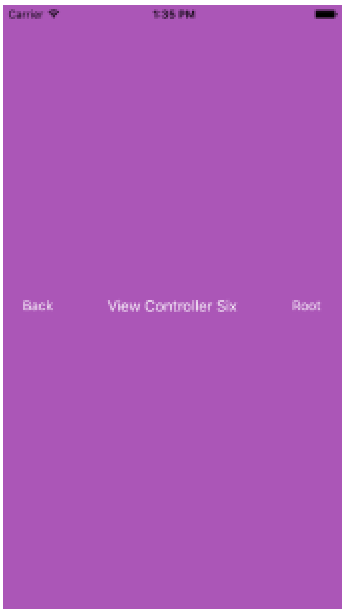
```
@IBAction func backButton(_ sender: UIButton) {
    guard (navigationController?.popViewController(animated:true)) !=
        else {
            dismiss(animated: true, completion: nil)
            return
    }
}
```

Build and run. Tap **Six Navigation**, and you get to the navigation



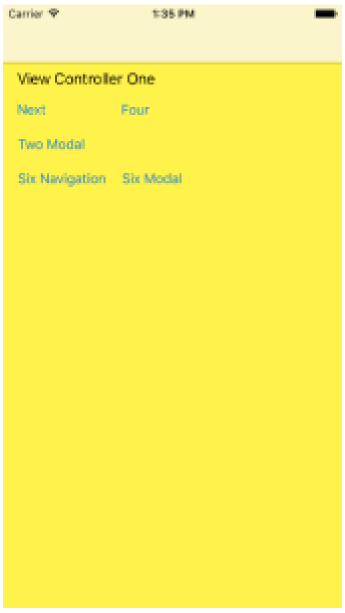
(https://apppie.files.wordpress.com/2016/07/2016-07-10_13-35-01.png)

Tap back and you get back to **One** again. Tap six modal and you get the modal



(https://apppie.files.wordpress.com/2016/07/2016-07-10_13-35-24.png)

Tap back and you get back to **One** again.

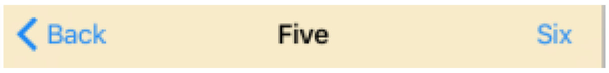


(https://apppie.files.wordpress.com/2016/07/2016-07-10_13-35-40.png)

You'll notice I left the **Root** button doing nothing for a modal since it has no meaning for modal.

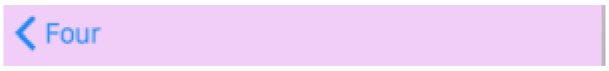
One More Place to Explore: The Navigation Back Button.

For most cases we get a **Back** button like this on the navigation bar:

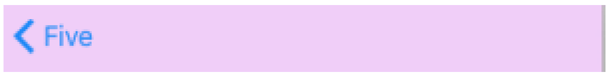


(https://apppie.files.wordpress.com/2016/07/2016-07-08_08-08-37.png)

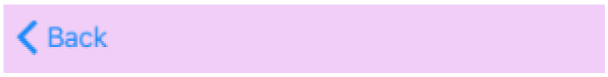
But you may notice that the **Six** controller does this, depending where it is pushed from:



(https://apppie.files.wordpress.com/2016/07/2016-07-08_08-07-53.png)



(https://apppie.files.wordpress.com/2016/07/2016-07-08_08-08-16.png)



(https://apppie.files.wordpress.com/2016/07/2016-07-08_08-11-38.png)

The title for the **Back** button comes from the view controller below it on the stack. When I push **Six** from **Five**, **Five** shows up as the title in the back button. This is another exploration you might want to take about navigation controllers, which you can find in the post Using the Navigation Bar Title and Back Button (https://makeapppie.com/2016/06/22/using-the-navigation-bar-title-and-back-button-in-swift-3-0/)

The Whole Code

```
<h2>ViewController.swift</h2>
//
//  ViewController.swift
//  SwiftProgNavControllerDemo
//
//  Created by Steven Lipton on 7/6/16.
//  Copyright © 2016 Steven Lipton. All rights reserved.
//

import UIKit

class ViewController: UIViewController {
//
// Action for using a story board id for navigation controller
//  let vc = storyboard?.instantiateViewController(withIdentifier: '
//
    @IBAction func sixNavigationButton(_ sender: UIButton) {
        guard let vc = storyboard?.instantiateViewController(withIde
            print("View controller Six not found")
            return
        }
        navigationController?.pushViewController(vc, animated: true)
    }

//
// Action for using a story board id for modal controller
//  let vc = storyboard?.instantiateViewController(withIdentifier: '
//
    @IBAction func sixModalButton(_ sender: UIButton) {
        guard let vc = storyboard?.instantiateViewController(withIde
            print("View controller Six not found")
            return
        }
        present(vc, animated: true, completion: nil)
    }

//
// modal example for use with dismissal see TwoViewController
//
    @IBAction func modalTwoButton(_ sender: UIButton) {
        let vc = TwoViewController(
            nibName: "TwoViewController",
            bundle: nil)
        present(vc, animated: true, completion: nil)
    }

//
// Example of pushing a view controller
//navigationController?.pushViewController(vc, animated: true)
//
    @IBAction func nextButton(_ sender: UIButton) {
        let vc = TwoViewController(
            nibName: "TwoViewController",
            bundle: nil)
        navigationController?.pushViewController(vc, animated: true)
    }

//
// Example of Logically pushing a view controller from a segue
// performSegue(withIdentifier: "Four",sender: self)
//
//

    @IBAction func fourFiveToggleButton(_ sender: UIButton){
        let normal = UIControlState(rawValue: 0) //beta 1 has no .nc
        if sender.titleLabel?.text == "Four"{
            performSegue(withIdentifier: "Four",
                         sender: self)
            sender.setTitle("Five", for: normal)
        } else{
            performSegue(withIdentifier: "Five",
                         sender: self)
            sender.setTitle("Four", for: normal)
        }
    }

}




<h2>TwoViewController.swift</h2>




//
//  TwoViewController.swift
//  SwiftProgNavControllerDemo
//
//  Created by Steven Lipton on 7/6/16.
//  Copyright © 2016 Steven Lipton. All rights reserved.
//

import UIKit

class TwoViewController: UIViewController {

//
// A back button for dismissal of both modals and navigation contro
//
```

```
//
    @IBAction func backButton(_ sender:UIButton){
        guard navigationController?.popViewController(animated: true
            print("Not a navigation Controller")
            dismiss(animated: true, completion: nil)
            return
        }
    }
}




<h2>ThreeViewController.swift</h2>




//
//  ThreeViewController.swift
//  SwiftProgNavControllerDemo
//
//  Created by Steven Lipton on 7/6/16.
//  Copyright © 2016 Steven Lipton. All rights reserved.
//

import UIKit

class ThreeViewController: UIViewController {
    }





<h2>SixViewController.swift</h2>




//
//  SixViewController.swift
//  SwiftProgNavControllerDemo
//
//  Created by Steven Lipton on 7/8/16.
//  Copyright © 2016 Steven Lipton. All rights reserved.
//

import UIKit

class SixViewController: UIViewController {

//
// Another back button for dismissal of both modals and navigation
//
//
    @IBAction func backButton(_ sender: UIButton) {
        guard (navigationController?.popViewController(animated:true
            else {
                dismiss(animated: true, completion: nil)
                return
        }
    }

//
// A pop to root of the navigation controller example
//  navigationController?.popToRootViewController(animated: true)
//
    @IBAction func rootButton(_ sender: UIButton) {
        guard navigationController?.popToRootViewController(animatec
        else {
            print("No Navigation Controller")
            return
        }
    }

    }
```

BACK BUTTON **DISMISS NAVIGATION** **DISMISSING MODAL** **DISMISSING XIB** **NAVIGATION CONTROLLLER** **NAVIGATION TITLE BAR** **POP VIEW CONTROLLER** **SEGUES** **SEGUES IN SWIFT** **SWIFT** **SWIFT PROGRAMMING** **UINAVIGATION CONTROLLER** **UINAVIGATIONCONTROLLER IN SWIFT** **XIB AS MODAL AND NAVIGATION**
SWIFT SWIFT, **TUTORIAL**

Swift Swift: Programmatic Navigation View Controllers in Swift

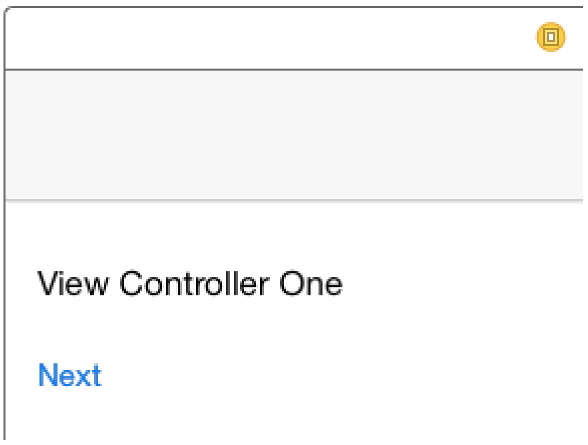
SEPTEMBER 15, 2014 | STEVEN LIPTON | 25 COMMENTS
[Converted to Swift 2.0 — SJL 9/17/15]
Navigation controllers are the workhorse of organizing view controllers. I've covered much of their use in other posts about MVC (https://makeapppie.com/2014/08/04/the-swift-swift-tutorial-why-do-we-need-delegates/), segues and delegates (https://makeapppie.com/2014/07/01/swift-swift-using-segues-and-delegates-in-navigation-controllers-part-1-the-template/). In this chapter, We'll go through some of the Swift code for the Navigation controller.

The View Controller Stack

Navigation view controllers are stack based. The newest view controller is the visible one. It is on top of the last one we saw.
If you are not familiar with the stack data structure, it is useful to understand stacks and their nomenclature a little better. Stacks work a lot like a deck of cards that are face up. You can only see the top card. When you place a card on top of the stack, you *push* a card on the stack. When you remove a card from the stack and show the card below it, you *pop* it off the stack. We use the terms push and pop a lot to talk about stacks, and you will find they describes methods often.

Opening a View Controller in a Xib

I've shown elsewhere how to move to navigation controllers through segues. Let's look at a few ways to do so programmatically by pushing and popping to the navigation stack directly.
Start a new single view project in Swift called `SwiftProgNavControllerDemo` . Go into the storyboard and select the blank view controller. Be sure to select the controller and not the view. From the drop down menu select **Edit>Embed in > Navigation Controller**.
In the view controller, Add a label and a button so your code looks like the diagram:



(https://apppie.files.wordpress.com/2014/09/screenshot-2014-09-13-17-05-30.png)
Open the assistant editor. Control-drag the button and make an action for a `UIButton` called `nextButton`. Remove the commented out method except the `viewDidLoad()`. Add the following two lines to the `nextButton`

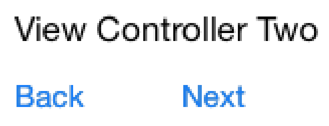
```
1  let vc = TwoViewController(nibName: "TwoViewController", bundle: nil)
2  navigationController?.pushViewController(vc, animated: true)
```

Line 1 creates a view controller of class `TwoViewController`, using the XIB of the same name. Line 2 pushed the view controller on the navigation controller stack maintained by `ViewController`. Your code should look like this when done:

```
class ViewController: UIViewController {

    @IBAction func nextButton(sender: UIButton) {
        let vc = TwoViewController(nibName: "TwoViewController", bun
        navigationController?.pushViewController(vc, animated: true)
    }
    override func viewDidLoad() {
       super.viewDidLoad()
       // Do any additional setup after loading the view,
       // typically from a nib.
    }
}
```

Close the assistant editor.
We need another view controller as a destination. Press **Command-n** or click **File>New>File…** Choose a **iOS source** template of **Cocoa Touch** Class. Make the new file a subclass of

`UIViewController` and name the file **TwoViewController**. To prove we are doing nothing with the storyboard, Click the option **Also create XIB file** to yes.
You will find a new XIB in interface builder. Set it up to look like the illustration below.



(https://apppie.files.wordpress.com/2014/09/screenshot-2014-09-13-17-04-18.png)

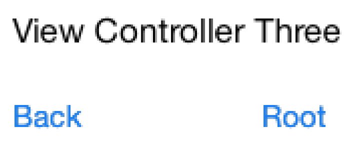
Open the assistant editor and control-drag the back button inside the `TwoViewController` class. Make an `@IBAction` method named `backButton` as an `UIButton`. Do the same for the **Next** button, but label the action method `nextButton` as we did in the last view controller. Add the following code to the `nextButton()` method:

```
let vc = ThreeViewController(nibName: "ThreeViewController", bundle: 
    navigationController?.pushViewController(vc, animated: true )
```

Your code should look like this:

```
class TwoViewController: UIViewController {

    @IBAction func nextButton(sender: UIButton) {
        let vc = ThreeViewController(nibName: "ThreeViewController",
        navigationController?.pushViewController(vc, animated: true 
    }
    @IBAction func backButton(sender: AnyObject) {
    }
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup
        // after loading the view.
    }
}
```

Let’s do this one more time so we end up with three view controllers to push onto the view controller stack. Follow the same procedure as you did for `TwoViewController`, but name it `ThreeViewController`, Instead of a **Next** button make a button **Root** like the diagram shows.



(https://apppie.files.wordpress.com/2014/09/screenshot-2014-09-13-17-04-08.png)

Control-drag the buttons to make two @`IBAction` methods named `backButton` and `rootButton` . Once again, remove the extra code except `viewDidLoad()`. Your `ThreeViewController` class should look like this:

```
class ThreeViewController: UIViewController {
    @IBAction func rootButton(sender: UIButton) {
    }
    @IBAction func backButton(sender: UIButton) {
    }
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

}
```

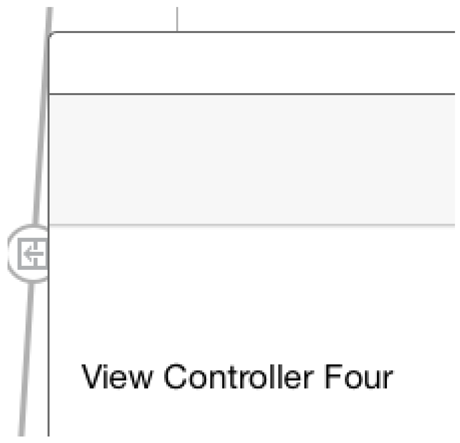
Now that we have set up the views, build and run. Tap the **Next** button and move between the three view controllers. Pushing a view controller in these controllers is a mere two lines of code:

```
let vc = ViewControllerName(nibName: "nameOfNib", bundle: nil)
navigationController?.pushViewController(vc, animated: true )
```

The first line creates the view controller, which of course we can name whatever we want. I tend to keep it simple and use `vc`, though if I had more than one, I'd be more descriptive. We used xibs here, which work a lot better than trying to connect to views on a storyboard. There's a bunch of weak variables you have to dance over in that case, so often it just doesn't work.
I rarely push view controllers. there are situations where you cannot use the storyboard and this is the alternative. I prefer the storyboard for two reasons: first it is better documentation of my user interface. Secondly, I prefer to let the system do as much of the background work as possible. The deeper into code you go, the more you have to worry about unexpected bugs. Swift and ARC Together often set `nil` when you don't expect or want it. Crashing is a good way to catch those in early development, so you don't have bigger more subtle bugs later.

Manual Segues to View Controllers

In other discussions of Segues, we talked about direct segues, connecting up a button directly to a view. You can do segues programmatically, which is useful when you conditionally go to a view.
Now add another subclass of `UIViewController` called `FourViewController.` This time you do not need a XIB. Go into the storyboard and drag a view controller onto the storyboard. Click on its view controller icon. In the identity inspector make the custom class `FourViewController`. Add a label to the view just so you know it is there.

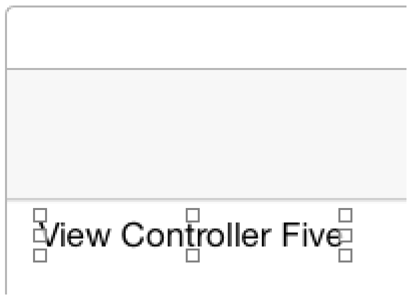


(https://apppie.files.wordpress.com/2014/09/screenshot-2014-09-13-17-04-53.png)

Click on the `ViewController` scene title in the storyboard. From the view controller Icon on `ViewController`, control-drag from `ViewController` to somewhere on `FourViewController`'s content, and `release` the mouse button. In the popup, select **Show**. Go into the properties inspector and set the segue's identifier to **four**.
Drag another button out to the View Controller scene and make the title `Four`. Go to `ViewController` class and add the following method:

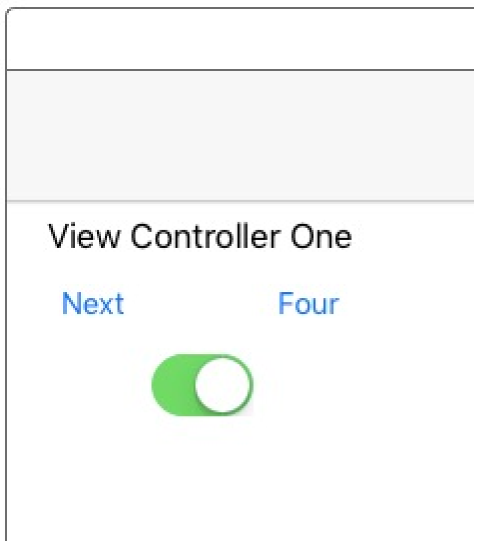
```
@IBAction func fourButton(sender: UIButton){
        performSegueWithIdentifier("four", sender: self)
    }
```

Open the assistant editor and drag from the circle next to the `fourButton()` method over the **Four** button and release.
The code above is a mere one line: it runs the segue. If you set the segue identifier correctly, that is all you need to do.
Where this is useful is conditional cases. We may want to go to different view controllers based on conditions in the current view controller or model. Let's try an example: Drag another view controller out to the storyboard. Set it up just like `FourViewController` except call it `FiveViewController` .



(https://apppie.files.wordpress.com/2014/09/screenshot-2014-09-13-17-04-42.png)

Add a label with the text **View Controller Five** in the new scene. Make a segue with an identifier **five** by control-dragging from `ViewController` to `FiveViewController`. Add a switch to the content view as in the illustration,



(https://apppie.files.wordpress.com/2014/09/2015-09-17_06-11-17.jpg)

Hook up an outlet by control dragging from the switch to the `ViewController` class in the assistant editor.

```
@IBOutlet weak var fourFiveSwitch: UISwitch!
Change the code for the fourButton() method to the following:
@IBAction func fourButton(sender: UIButton){
        if fourFiveSwitch.on{
            performSegueWithIdentifier("four", sender: self)
        }else{
            performSegueWithIdentifier("five", sender: self)
        }
    }
```

Build and run. Tap the **four** button and then hit the back button in the navigation bar. Now change the switch setting and tap the **Four** again.

Closing a View Controller

Up to now, we've relied on the navigation controller's back button. While I may not use `pushViewController()` a lot, I do use `popViewController()` often. Almost every delegate between view controllers in a navigation stack will use it. There are several versions of the controller and I wanted to explore them with you.
In the `TwoViewController` class, change the `backButton` code to read:

```
@IBAction func backButton(sender: UIButton) {
       navigationController?.popViewControllerAnimated(true)
   }
```

Now do the same in `ThreeViewController:`

```
@IBAction func backButton(sender: UIButton) {
        navigationController?.popViewControllerAnimated(true)
    }
```

Also in `ThreeViewController`, Let's add to the `rootButton()` method the following.

```
@IBAction func rootButton(sender: UIButton) {
        navigationController?.popToRootViewControllerAnimated(true)
    }
```

Build and Run. you will find the **Next** and **Root** button now take you around the application.
There are three versions of pop: `popViewController()`, `popToRootController()`, and `popToViewController()` The most Common is `popViewController()` which removes the top view controller from the stack. `popToRootViewController()` and `popToViewController()` pops everything or everything up to a specific View Controller off the stack, returning what it popped off.

Moving Values

So far we have not moved values between view controllers. If using a storyboard and segues with a `performSegueWithIdentifier()`, It is the same way we have already talked about for segues and storyboard. If we don't use segues, it is rather simple, Be careful how you use it, since it is easy to break MVC.
Let's start by changing the view controller class we have written to this:

```
 1  class ViewController: UIViewController {
 2      var vcCount = 0
 3      @IBAction func nextButton(sender: UIButton) {
 4          let vc = TwoViewController(nibName: "TwoViewController", bun
 5          vc.vcCount = vcCount++
 6          navigationController?.pushViewController(vc, animated: true)
 7      }
 8      @IBOutlet weak var fourFiveSwitch: UISwitch!
 9      @IBAction func fourButton(sender: UIButton){
10          if fourFiveSwitch.on{
11              performSegueWithIdentifier("four", sender: self)
12          }else{
13              performSegueWithIdentifier("five", sender: self)
14          }
15      }
16  }
```

We assign an integer property `vcCount` a value of zero in line 2. Line 5 increments the count, then sends that to the property of the `TwoViewController` instance we created in line 4. Change `TwoViewController` to this:

```
 1  class TwoViewController: UIViewController {
 2      var vcCount:Int = 0
 3      @IBAction func nextButton(sender: UIButton) {
 4          let vc = ThreeViewController(nibName: "ThreeViewController",
 5          navigationController?.pushViewController(vc, animated: true
 6      }
 7      @IBAction func backButton(sender: UIButton) {
 8          navigationController?.popViewControllerAnimated(true)
 9      }
10
11      override func viewDidLoad() {
12          super.viewDidLoad()
13          print("\(vcCount) ")
14      }
15  }
```

On line 2, we added a integer property `vcCount`. In `viewDidLoad()`, we printed the property to the console.

Using the Delegate

We can send data to other controllers now, but we need delegates to return it from a popped controller to the next visible controller. The process is the same as discussed in my delegates post, except there is no `prepareForSegue()` In the `ViewTwoController.swift` file, make a protocol

```
protocol TwoVCDelegate{
    func didFinishTwoVC(controller:TwoViewController)
}
```

We will need to add the delegate to **`TwoViewController`**:

```
var delegate:TwoVCDelegate! = nil
```

We'll use the protocol by changing our **`backButton`** method in **`TwoViewController`** to:

```
@IBAction func backButton(sender: UIButton) {
        //navigationController?.popViewControllerAnimated(true)
        delegate.didFinishTwoVC(self)
    }
```

Our final steps are to adopt the protocol in **`ViewController`**, by changing the code like this:

```
 1  class ViewController: UIViewController,TwoVCDelegate {
 2      var vcCount:Int = 0
 3      func didFinishTwoVC(controller: TwoViewController) {
 4          vcCount = controller.vcCount + 1
 5          controller.navigationController?.popViewControllerAnimated(t
 6      }
 7      @IBAction func nextButton(sender: UIButton) {
 8          let vc = TwoViewController(nibName: "TwoViewController", bun
 9         vc.vcCount = vcCount
10         vc.delegate = self
11          navigationController?.pushViewController(vc, animated: true)
12      }
13      @IBOutlet weak var fourFiveSwitch: UISwitch!
14      @IBAction func fourButton(sender: UIButton){
15          if fourFiveSwitch.on{
16              performSegueWithIdentifier("four", sender: self)
17          }else{
18              performSegueWithIdentifier("five", sender: self)
19          }
20      }
21  }
```

Line 1 adopts the delegate, and lines 3-6 is the required method **`didFinishTwoVC`**. In that method we take the current value in the **`vcCount`** property in **`TwoViewController`** and increment it, then pop the controller off the stack. In line 10, we added the assignment of the **`TwoViewController`**'s delegate to our current view controller.

One More Thing: Setting the Navigation Title Bar

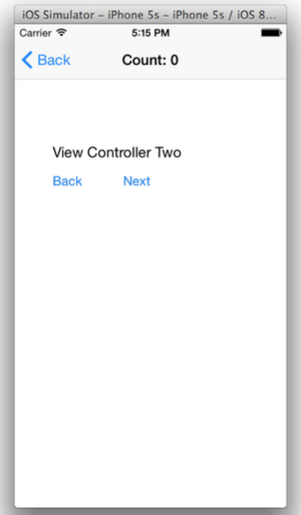
We have been using the console to tell us the value of **`vcCount`** in the view controllers. We can also place it in the title of the navigation bar. Change the **`viewDidLoad()`** in **`TwoViewController`** to this:

```
override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Count: \(vcCount)"
    }
```

All you need to do to change the navigation bar title is assign a string to `navigationItem.title`. The `navigationItem` is a property of type `UINavigationItem`, and has a property to quickly change the title of the bar. There is a lot you can change in the `UINavigationItem`, but that is a topic for another post.
Also change the `vcCount` declaration to this in `ViewController`:

```
var vcCount:Int = 0{
        didSet{
            navigationItem.title = "Count: \(vcCount)"
        }
    }
```

We cannot use `viewDidload()` in this case since the root view controller will load only once. We could have used `viewWillAppear()` instead, but we can also use the property observer feature of Swift. Using `didSet`, any time `vcCount` changes, the title changes with it. Build and run. Go back and forth from view one to view two.



(https://apppie.files.wordpress.com/2014/09/screenshot-2014-09-13-17-15-04.png)

The Whole Code

```
//
//  ViewController.swift
//  SwiftNavControllerDemo
//
//  Created by Steven Lipton on 9/10/14.
//  Copyright (c) 2014 MakeAppPie.Com. All rights reserved.
//

import UIKit

class ViewController: UIViewController,TwoVCDelegate {
    var vcCount:Int = 0{
        didSet{
        navigationItem.title = "Count: \(vcCount)"
        }
    }
    func didFinishTwoVC(controller: TwoViewController) {
        vcCount = controller.vcCount + 1
        controller.navigationController?.popViewControllerAnimated(t
    }
    @IBAction func nextButton(sender: UIButton) {
        let vc = TwoViewController(nibName: "TwoViewController", bun
        vc.vcCount = vcCount
        vc.delegate = self

        navigationController?.pushViewController(vc, animated: true)
    }
    @IBOutlet weak var fourFiveSwitch: UISwitch!
    @IBAction func fourButton(sender: UIButton){
        if fourFiveSwitch.on{
            performSegueWithIdentifier("four", sender: self)
        }else{
            performSegueWithIdentifier("five", sender: self)
        }
    }
}

//
//  TwoViewController.swift
//  SwiftNavControllerDemo
//
//  Created by Steven Lipton on 9/10/14.
//  Copyright (c) 2014 MakeAppPie.Com. All rights reserved.
//

import UIKit
protocol TwoVCDelegate{
    func didFinishTwoVC(controller:TwoViewController)
}

class TwoViewController: UIViewController {
    var vcCount:Int = 0
    @IBAction func nextButton(sender: UIButton) {
        let vc = ThreeViewController(nibName: "ThreeViewController",
        navigationController?.pushViewController(vc, animated: true
    }
    var delegate:TwoVCDelegate!=nil
    @IBAction func backButton(sender: UIButton) {
        //navigationController?.popViewControllerAnimated(true)
        delegate.didFinishTwoVC(self)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Count: \(vcCount)"
    }

}

//
//  ThreeViewController.swift
//  SwiftNavControllerDemo
//
//  Created by Steven Lipton on 9/11/14.
//  Copyright (c) 2014 MakeAppPie.Com. All rights reserved.
//
```

```

import UIKit

class ThreeViewController: UIViewController {

    @IBAction func rootButton(sender: UIButton) {
        navigationController?.popToRootViewControllerAnimated(true)
    }
    @IBAction func backButton(sender: UIButton) {
        navigationController?.popViewControllerAnimated(true)
    }
    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Three"
        // Do any additional setup after loading the view.
    }

   }
//
//  FourViewController.swift
//  SwiftNavControllerDemo
//
//  Created by Steven Lipton on 9/11/14.
//  Copyright (c) 2014 MakeAppPie.Com. All rights reserved.
//

import UIKit

class FourViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Four" //another example of a bar ti
    }

}
//
//  FiveViewController.swift
//  SwiftNavControllerDemo
//
//  Created by Steven Lipton on 9/11/14.
//  Copyright (c) 2014 MakeAppPie.Com. All rights reserved.
//

import UIKit

class FiveViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Five" //another example of a bar ti
    }
}
```

BACK BUTTON **NAVIGATION CONTROLLLER** **NAVIGATION TITLE BAR** **POP VIEW CONTROLLER** **SEGUES** **SEGUES IN SWIFT** **SWIFT** **SWIFT PROGRAMMING** **UINAVIGATION CONTROLLER** **UINAVIGATIONCONTROLLER IN SWIFT**