How to Create Your Own Slide-Out Navigation Panel in Swift

James Frost on October 14, 2014

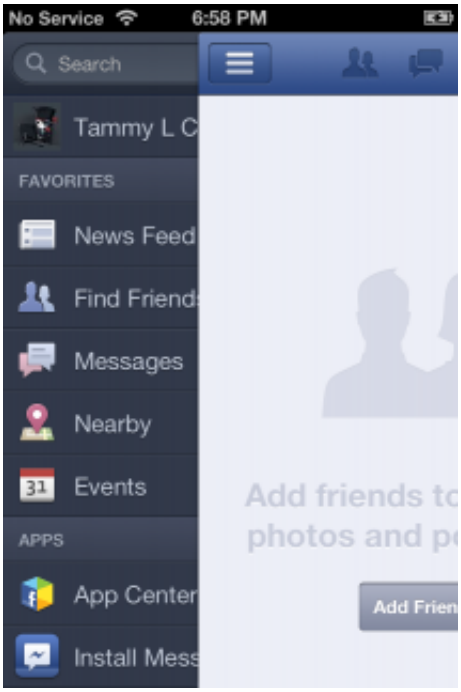
Note from Ray: This is a Swift update to a popular Objective-C tutorial on our site, released as part of the iOS 8 Feast. Updated by James Frost for iOS 8, Xcode 6.1, and Swift. Original post by Tutorial Team member Tammy Coron. Enjoy!

This tutorial will show you how to build a slide-out navigation panel similar to the ones available in the Facebook and Path iOS apps.

The slide-out navigation panel design pattern lets developers add permanent navigation to their apps without taking up valuable screen real estate. The user can choose to reveal the navigation at any time, while still seeing their current context.

These days, there are many pre-built solutions out there you can use, such as John-Lluch's excellent SWRevealViewController or Mutual Mobile's MMDrawerController. If you're looking for the quickest and easiest way, using library might be a good way to go.

However, in this tutorial you'll see that it's really not as complicated as you might think. You'll take a less-is-more approach and skip all of the complicated code that's not really required, so that you can apply the slide-out navigation technique to your own applications with relative ease.



Slide to the right!

Getting Started

You're going to build slide-out navigation into a cute kitten and puppy photo browser. To get started, download the starter project for this tutorial. It's a zip file, so save it to a convenient location and then extract it to get the project.

Next open the project in Xcode and take a look at how it's organized. The **Resources** folder contains all of the kitten and puppy images that'll be displayed by the app. Notice too that are three main view controllers. When it comes time to adapt this tutorial to your own projects, here's what you should keep in mind:

- **ContainerViewController**: This is where the magic happens! This contains the views of the left, center, and right view controllers and handles things like animations and swiping. In this project, it's created and added to the window in `applicationDidFinishLaunching:` in **AppDelegate.swift**
- **CenterViewController**: The center panel. This can be replaced with your own view controller (make sure you copy the button actions).
- **SidePanelViewController**: Used for the left and right side panels. This could be replaced with your own view controller.

The views for the center, left, and right view controllers are all defined within **Main.storyboard**, so feel free to take a quick look at that to get an idea of how the app will look.

Now that you're familiar with the structure of the project, it's time to start at square one: the center panel.

Finding Your Center

In this section, you are going to place the `CenterViewController` inside the `ContainerViewController`, as a child view controller.

> **Note:** This section uses a concept called View Controller Containment introduced in iOS 5. If you are new to this concept, check out Chapter 22 in iOS 5 by Tutorials, "UIViewController Containment."

Open **ContainerViewController.swift**. Notice at the bottom of the file that there is a small extension for `UIStoryboard`. This adds a few class methods which make it a bit more concise to load specific view controllers from the app's storyboard. You'll make use of these methods soon.

Add a couple of properties for the `CenterViewController` and for a `UINavigationController`, above `ContainerViewController`'s `init` method:

```
var centerNavigationController: UINavigationController!
var centerViewController: CenterViewController!
```

> **Note:** These are *implicitly-unwrapped optionals* (as denoted by the `!`). They have to be optional because their values won't be initialized until after `init` has been called, but they can be automatically unwrapped because once they're created you know they will always have values.

Find `viewDidLoad` and add the following block of code to it, beneath the call to super:

```
centerViewController = UIStoryboard.centerViewController()
centerViewController.delegate = self

// wrap the centerViewController in a navigation controller, so we can push views to it
// and display bar button items in the navigation bar
centerNavigationController = UINavigationController(rootViewController:
centerViewController)
view.addSubview(centerNavigationController.view)
addChildViewController(centerNavigationController)

centerNavigationController.didMoveToParentViewController(self)
```

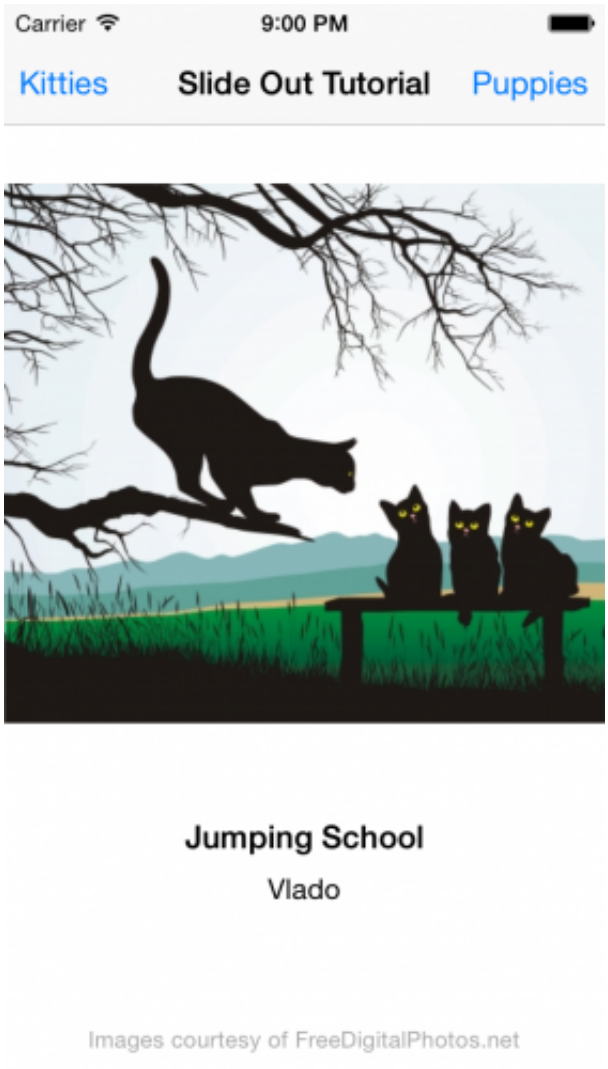
The code above creates a new `CenterViewController` and assigns it to the `centerViewController` property that you just created. It also creates a `UINavigationController` to contain the `CenterViewController`. It then adds the navigation controller's view to `ContainerViewController`'s view and sets up the parent-child relationship using `addSubview`, `addChildViewContoller` and `didMoveToParentViewController`.

This code also sets the current view controller as the center view controllers's delegate. This will be used by the center view controller to tell its container when to show and hide the left and right side panels.

You need to modify the interface of this class so that it will adhere to the protocol requirements of `CenterViewControllerDelegate`. Do this by updating the class declaration for `ContainerViewController`, near the top of the file:

```
class ContainerViewController: UIViewController, CenterViewControllerDelegate {
```

Now is a good time to check your progress. Build and run the project. If all went well, you should see something similar to the screen below:



Yes, those buttons at the top will eventually bring you kitties and puppies. What better reason could there be for creating sliding navigation panels? But to get your cuteness fix, you've got to start sliding. First, to the left!

Kittens to the Left of Me...

You've created your center panel, but adding the left view controller requires a different set of steps. There's quite a bit of set up to get through here, so bear with it. Think of the kittens!

To expand the left menu, the user will tap on the **Kitties** button in the navigation bar. So head on over to **CenterViewController.swift**.

In the interests of keeping this tutorial focused on the important stuff, the IBActions and IBOutlets have been pre-connected for you in the storyboard. However, to implement your DIY slide-out navigation, you need to understand how the buttons are configured.

Notice that there are already two IBAction methods, one for each of the buttons. Find `kittiesTapped` and add the following implementation to it:

```
delegate?.toggleLeftPanel?()
```

As previously mentioned, the method is already hooked up to the **Kitties** button.

This uses *optional chaining* to only call **toggleLeftPanel** if **delegate** has a value, and if **delegate** has implemented the method.

You can see the definition of the delegate protocol at the top of **CenterViewController.swift**. As you'll see, there are optional methods **toggleLeftPanel** and **toggleRightPanel**. If you remember, when you set up the center view controller instance earlier, you set its delegate as the container view controller. Time to go and implement **toggleLeftPanel**.

> **Note:** For more information on delegate methods and how to implement them, please refer to Apple's Developer Documentation.

Return to **ContainerViewController.swift**. First add an **enum** to the top of the file, below the **import** statements:

```
enum SlideOutState {
  case BothCollapsed
  case LeftPanelExpanded
  case RightPanelExpanded
}
```

This will let you keep track of the current state of the side panels, so you can tell whether neither panel is visible, or one of the left or right panels are visible.

Now, below your existing **centerViewController** property, add two more properties:

```
var currentState: SlideOutState = .BothCollapsed
var leftViewController: SidePanelViewController?
```

These will hold the current state, and the left side panel view controller itself:

The current state is initialized to be **.BothCollapsed** – that is, neither of the side panels are visible when the app first loads. The **leftViewController** property is an **optional**, because you'll be adding and removing the view controller at various times, so it might not always have a value.

Now add an implementation for the **toggleLeftPanel** delegate method:

```
let notAlreadyExpanded = (currentState != .LeftPanelExpanded)

if notAlreadyExpanded {
  addLeftPanelViewController()
}

animateLeftPanel(shouldExpand: notAlreadyExpanded)
```

First, this method checks whether the left side panel is already expanded or not. If it's not already visible, then the panel is added and animated to its 'open' position. If the panel *is* already visible, then it will be animated to its 'closed' position.

What does it mean to 'add' the left panel? Locate **addLeftPanelViewController**, and add the following code inside it:

```
if (leftViewController == nil) {
  leftViewController = UIStoryboard.leftViewController()
  leftViewController!.animals = Animal.allCats()

  addChildSidePanelController(leftViewController!)
}
```

The code above first checks to see if the **leftViewController** property is nil. If it is, then the code creates a new **SidePanelViewController**, and assigns it a list of animals to display – in this case, cats!

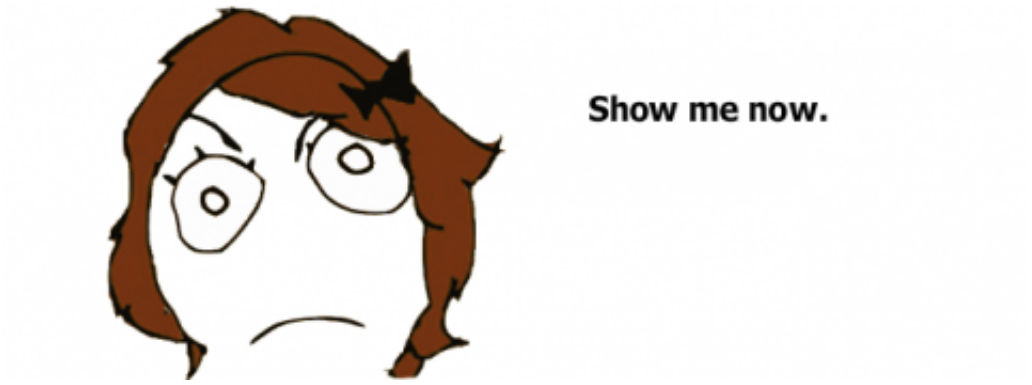
Next, add the implementation for `addChildSidePanelController` below `addLeftPanelViewController`:

```
func addChildSidePanelController(sidePanelController: SidePanelViewController) {
  view.insertSubview(sidePanelController.view, atIndex: 0)

  addChildViewController(sidePanelController)
  sidePanelController.didMoveToParentViewController(self)
}
```

This method inserts the child view into the container view controller. This is much the same as adding the center view controller earlier. It simply inserts its view (in this case it's inserted at z-index 0, which means that it will be *below* the center view controller) and adds it as a child view controller.

It's almost time to try the project out again, but there's one more thing to do: add some animation! It won't take long!



And sliiiiiiide!

First, add a constant below your other properties in `ContainerViewController.swift`:

```
let centerPanelExpandedOffset: CGFloat = 60
```

This value is the width, in points, of the center view controller that will be left visible once it has animated offscreen. 60 points should do it.

Locate the method stub for `animateLeftPanel` and add the following block of code to it:

```
if (shouldExpand) {
  currentState = .LeftPanelExpanded

  animateCenterPanelXPosition(targetPosition:
CGRectGetWidth(centerNavigationController.view.frame) - centerPanelExpandedOffset)
} else {
  animateCenterPanelXPosition(targetPosition: 0) { finished in
    self.currentState = .BothCollapsed

    self.leftViewController!.view.removeFromSuperview()
    self.leftViewController = nil;
  }
}
```

This method simply checks whether it's been told to expand or collapse the side panel. If it should expand, then it sets the current state to indicate that the left panel is expanded, and then animates the center panel so it's open. Otherwise, it animates the center panel closed and then removes its view and sets the current state to indicate that it's closed.

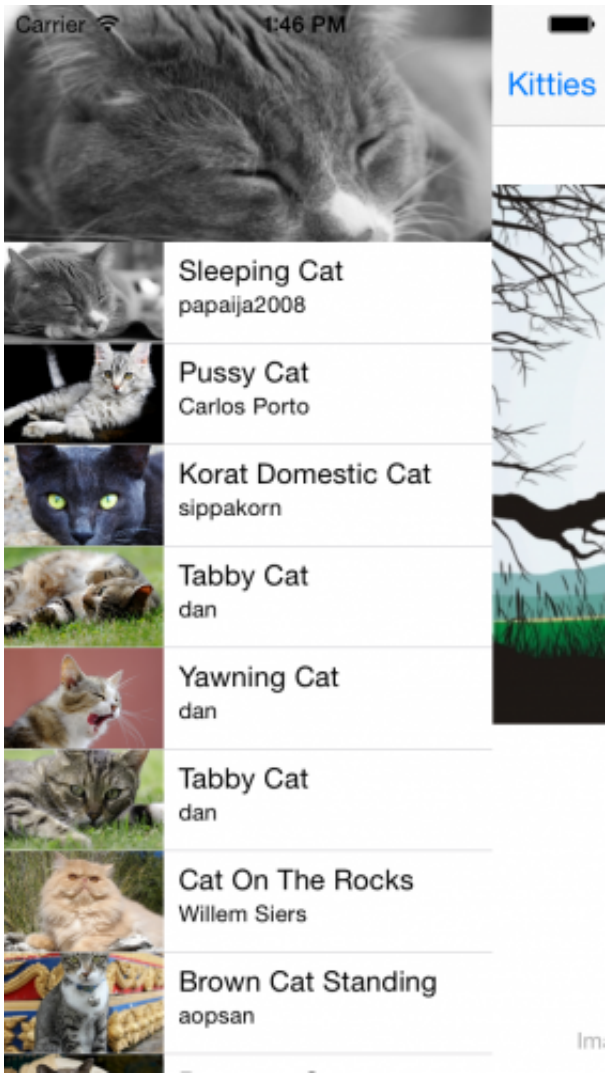
Finally, add `animateCenterPanelXPosition` underneath that:

```
func animateCenterPanelXPosition(#targetPosition: CGFloat, completion: ((Bool) ->
Void)! = nil) {
  UIView.animateWithDuration(0.5, delay: 0, usingSpringWithDamping: 0.8,
initialSpringVelocity: 0, options: .CurveEaseInOut, animations: {
    self.centerNavigationController.view.frame.origin.x = targetPosition
  }, completion: completion)
}
```

This is where the actual animation happens. The center view controller's view is animated to the specified position, with a nice spring animation. The method also takes an optional completion closure, which it passes on to the **UIView** animation. You can try tweaking the duration and spring damping parameters if you want to change the appearance of the animation.

OK… It's taken a little while to get everything in place, but now is a great time to build and run the project. So do it!

When you've run the project, try tapping on the **Kitties** button in the navigation bar. The center view controller should slide over – whoosh! – and reveal the Kitties menu underneath. D'aww, look how cute they all are.



But too much cuteness can be a dangerous thing! Tap the **Kitties** button again to hide them!

Me and my shadow

When the left panel is open, notice how it's right up against the center view controller. It would be nice if there were a bit more of a distinction between them. How about adding a shadow?

Add the following method below your animation methods:

```
func showShadowForCenterViewController(shouldShowShadow: Bool) {
  if (shouldShowShadow) {
    centerNavigationController.view.layer.shadowOpacity = 0.8
  } else {
    centerNavigationController.view.layer.shadowOpacity = 0.0
  }
}
```

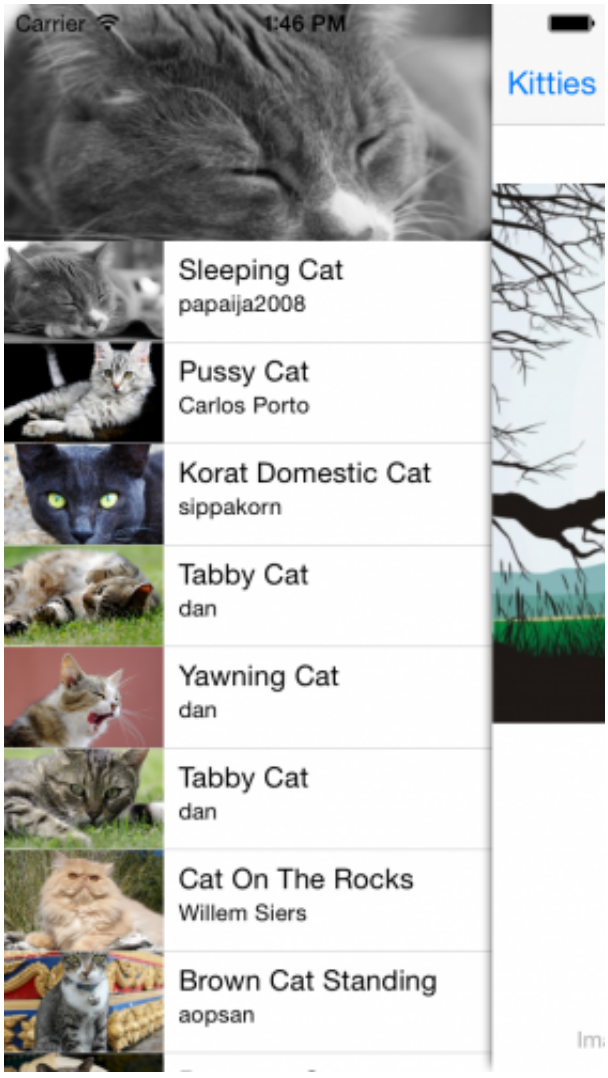
This adjusts the opacity of the navigation controller's shadow to make it visible or hidden. You can implement a `didSet` observer to add or remove the shadow whenever the `currentState` property changes.

Scroll to the top of **ContainerViewController.swift** and change the `currentState` declaration to:

```
var currentState: SlideOutState = .BothCollapsed {
  didSet {
    let shouldShowShadow = currentState != .BothCollapsed
    showShadowForCenterViewController(shouldShowShadow)
  }
}
```

The `didSet` closure will be called whenever the property's value changes. If either of the panels are expanded, then the shadow will be displayed.

Build and run the project again. This time when you tap the kitties button, check out the sweet new shadow! Looks better, huh?



Up next, adding the same functionality but for the right side, which means… puppies!

Puppies to the Right…

To add the right panel view controller, simply repeat the steps for adding the left view controller.

In **ContainerViewController.swift**, add the following property below the `leftViewController` property:

```
var rightViewController: SidePanelViewController?
```

Now locate `toggleRightPanel`, and add the following:

```
let notAlreadyExpanded = (currentState != .RightPanelExpanded)

if notAlreadyExpanded {
  addRightPanelViewController()
}

animateRightPanel(shouldExpand: notAlreadyExpanded)
```

Then add implementations for the corresponding `addRightPanelViewController` and `animateRightPanelViewController` methods below their left view controller counterparts:

```
func addRightPanelViewController() {
  if (rightViewController == nil) {
    rightViewController = UIStoryboard.rightViewController()
    rightViewController!.animals = Animal.allDogs()

    addChildSidePanelController(rightViewController!)
  }
}

func animateRightPanel(#shouldExpand: Bool) {
  if (shouldExpand) {
    currentState = .RightPanelExpanded

    animateCenterPanelXPosition(targetPosition: -
CGRectGetWidth(centerNavigationController.view.frame) + centerPanelExpandedOffset)
  } else {
    animateCenterPanelXPosition(targetPosition: 0) { _ in
      self.currentState = .BothCollapsed

      self.rightViewController!.view.removeFromSuperview()
      self.rightViewController = nil;
    }
  }
}
```

The code above is almost an exact duplicate of the code for the left panel, except of course for the differences in method and property names and the direction. If you have any questions about it, review the explanation from the previous section.

Just as before, the IBActions and IBOutlets have been connected in the storyboard for you. Similar to the **Kitties** button, the **Puppies** button is hooked up to an IBAction method named `puppiesTapped`. This button controls the sliding of the center panel to reveal the right-side panel.

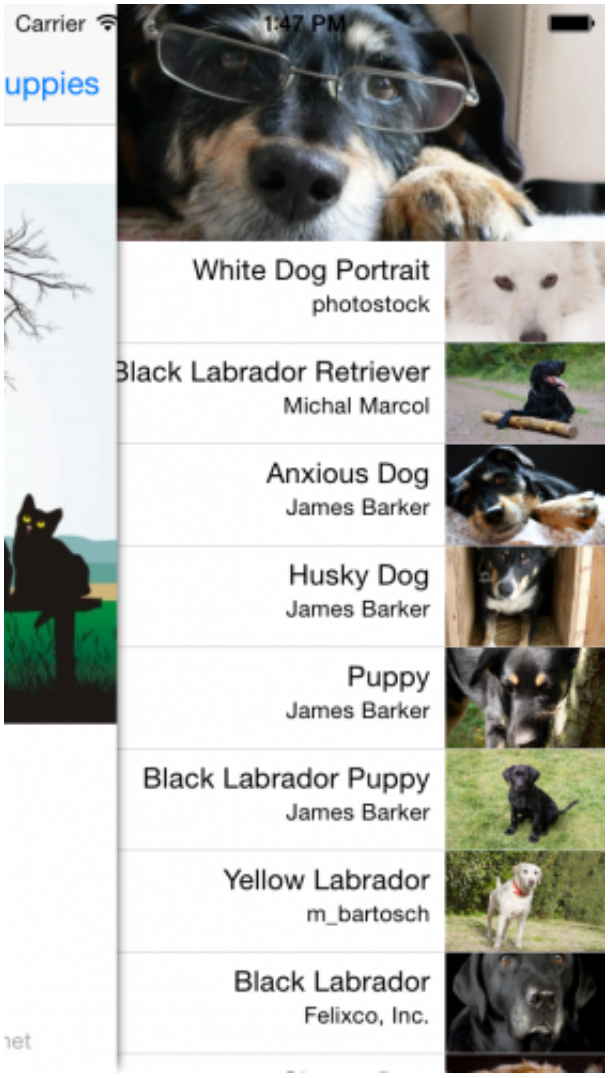
Finally, switch to **CenterViewController.swift** and add the following snippet to `puppiesTapped`:

```
delegate?.toggleRightPanel?()
```

Again, this is the same as `kittiesTapped`, except it's toggling the right panel instead of the left.

Time to see some puppies!

Build and run the program again to make sure everything is working. Tap on the **Puppies** button. Your screen should look like this:



Looking good, right? But remember, you don't want to expose yourself to the cuteness of puppies for too long, so tap that button again to hide them away.

You can now view both kitties and puppies, but it would be great to be able to view a bigger picture of each one, wouldn't it? MORE CUTENESS :]

Pick An Animal, Any Animal

The kitties and puppies are listed within the left and right panels. These are both instances of **SidePanelViewController**, which essentially just contain table views.

Head over to **SidePanelViewController.swift** to handle taps on the rows. Take a look at the `SidePanelViewControllerDelegate` definition at the top of the file. A side panel's delegate can be notified via this method whenever an animal is tapped. Let's use it!

Still in **SidePanelViewController.swift**, first add an optional delegate property at the top of the class, underneath the table view IBOutlet:

```
var delegate: SidePanelViewControllerDelegate?
```

Then fill in the implementation for **tableView(_:didSelectRowAtIndexPath:)**:

```
func tableView(tableView: UITableView!, didSelectRowAtIndexPath indexPath:
NSIndexPath!) {
  let selectedAnimal = animals[indexPath.row]
  delegate?.animalSelected(selectedAnimal)
}
```

If there is a delegate set, this will tell it that an animal has been selected. But there is no delegate yet! It would make sense for **CenterViewController** to be the side panel's delegate, as it can then display the selected animal photo and title.

Open up **CenterViewController.swift** to implement the delegate protocol. Change the class declaration to the following:

```
class CenterViewController: UIViewController, SidePanelViewControllerDelegate {
```

And then add the implementation for **animalSelected**:

```
func animalSelected(animal: Animal) {
  imageView.image = animal.image
  titleLabel.text = animal.title
  creatorLabel.text = animal.creator

  delegate?.collapseSidePanels?()
}
```

This method simply populates the image view and labels in the center view controller with the animal's image, title, and creator. Then, if the center view controller has a delegate of its own, you can tell it to collapse the side panel away so you can focus on the selected item.

collapseSidePanels hasn't been implemented yet, so switch over to **ContainerViewController.swift** to finish things up.

Add the method below **toggleRightPanel**:

```
func collapseSidePanels() {
  switch (currentState) {
    case .RightPanelExpanded:
      toggleRightPanel()
    case .LeftPanelExpanded:
      toggleLeftPanel()
    default:
        break
  }
}
```

The switch statement in this method simply checks the current state of the side panels, and collapses whichever one is open (if any!).

Finally, update **addChildSidePanelViewController** to the following implementation:

```
func addChildSidePanelController(sidePanelController: SidePanelViewController) {
  sidePanelController.delegate = centerViewController

  view.insertSubview(sidePanelController.view, atIndex: 0)

  addChildViewController(sidePanelController)
  sidePanelController.didMoveToParentViewController(self)
}
```

In addition to what it was doing before, the method will now set the center view controller as the side panels'

delegate.

That should do it! Build and run the project again. View kitties or puppies, and tap on one of the cute little critters. The side panel should collapse itself again and you should see the details of the animal you chose.

Move Your Hands Back and Forth

The navigation bar buttons are great, but most apps also allow you to “swipe” to open the side panels. Adding gestures to your app is surprisingly simple. Don’t be intimated; you’ll do fine!

Open **ContainerViewController.swift** and locate `viewDidLoad`. Add the following to the end:

```
let panGestureRecognizer = UIPanGestureRecognizer(target: self, action:
"handlePanGesture:")
centerNavigationController.view.addGestureRecognizer(panGestureRecognizer)
```

The above code defines a **UIPanGestureRecognizer** and assigns **handlePanGesture()** to it to handle any detected pan gestures. (You will write the code for that method soon.)

By default, a pan gesture recognizer detects a single touch with a single finger, so it doesn't need any extra configuration. All that's required is to add the newly created gesture recognizer to **centerNavigationController.view**.

> **Note:** Refer to our Using UIGestureRecognizer with Swift Tutorial for more information about gesture recognizers in iOS.

Next make this class a **UIGestureRecognizerDelegate** by changing the class declaration at the top:

```
class ContainerViewController: UIViewController, CenterViewControllerDelegate,
UIGestureRecognizerDelegate {
```

Didn't I tell you that'd be simple? There's only one move remaining in your slide-out routine.

Now Move That View!

The gesture recognizer calls **handlePanGesture:**code> when it detects a gesture. So your last task for this tutorial is to implement that method.

Locate the stub for **handlePanGesture:** and add the following block of code to it (it's a big one!):

```
let gestureIsDraggingFromLeftToRight = (recognizer.velocityInView(view).x > 0)

switch(recognizer.state) {
  case .Began:
    if (currentState == .BothCollapsed) {
      if (gestureIsDraggingFromLeftToRight) {
        addLeftPanelViewController()
      } else {
        addRightPanelViewController()
      }

      showShadowForCenterViewController(true)
    }
  case .Changed:
    recognizer.view!.center.x = recognizer.view!.center.x +
recognizer.translationInView(view).x
    recognizer.setTranslation(CGPointZero, inView: view)
  case .Ended:
    if (leftViewController != nil) {
      // animate the side panel open or closed based on whether the view has moved more
or less than halfway
      let hasMovedGreaterThanHalfway = recognizer.view!.center.x >
view.bounds.size.width
      animateLeftPanel(shouldExpand: hasMovedGreaterThanHalfway)
    } else if (rightViewController != nil) {
      let hasMovedGreaterThanHalfway = recognizer.view!.center.x < 0
      animateRightPanel(shouldExpand: hasMovedGreaterThanHalfway)
    }
  default:
        break
```

```
}
```

The pan gesture recognizer detects pans in any direction, but you're only interested in horizontal movement. First, you set up the `gestureIsDraggingFromLeftToRight` Boolean to check for this.

There are three states that need to be tracked: `UIGestureRecognizerState.Began`, `UIGestureRecognizerState.Changed`, and `UIGestureRecognizerState.Ended`:

- `Began`: If the user starts panning, and neither panel is visible then show the correct panel based on the pan direction.
- `Changed`: If the user is already panning, translate the center view controller's view by the amount that the user has panned
- `Ended`: When the pan ends, check whether the left or right view controller is visible. Depending on which one is visible and how far the pan has gone, perform the animation.

You can move the center view around, and show and hide the left and right views using a combination of these three states, as well as the location and velocity / direction of the pan gesture.

For example, if the gesture direction is right, then show the left panel. If the direction is left, then show the right panel.

Build and run the program again. At this point, you should be able to slide the center panel left and right, revealing the panels underneath. If everything is working… you're good to go!

Where to Go from Here?

Congratulations! If you made it all the way through, you're a slide-out navigation panel ninja!

I hope you enjoyed this tutorial. Feel free to download the completed project file. I'm sure you'll enjoy being stuck in the middle of kitties and puppies!

As I mentioned earlier, if you prefer a pre-built library over the DIY solution, be sure to check out MMDrawerController or SWRevealViewController. For an in-depth look (and pretty pictures), check out iOS developer and designer Ken Yarmosh's post New iOS Design Pattern: Slide-Out Navigation. He does a great job of explaining the benefits of using this design pattern and showing common uses in the wild.

Leave a comment in the forums below to share your slide-out moves and grooves!

James Frost

James is an iOS developer at Mubaloo Ltd, most recently spending a lot of time getting acquainted with iBeacons. He started writing code in the early 90s on his family's BBC B, and hasn't stopped.

You can find him on Twitter, his personal website, and Stack Overflow.