***Passing Data Between View Controllers in iOS***

**The five ways are:**

1. **Segues**
2. **Delegate design pattern**
3. **Singleton design pattern**
4. **Closures**
5. **Notification Center**

## *Contents*

[[](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section1)](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section1)

###### SECTION 1

The difference between best and weak practices.

[[](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section2)](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section2)

###### SECTION 2

Passing data forward

[[](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section3)](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section3)

###### SECTION 3

Passing data backward

[[](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section4)](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section4)

###### SECTION 4

Advanced techniques

[[](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section5)](https://matteomanferdini.com/how-ios-view-controllers-communicate-with-each-other/#section5)

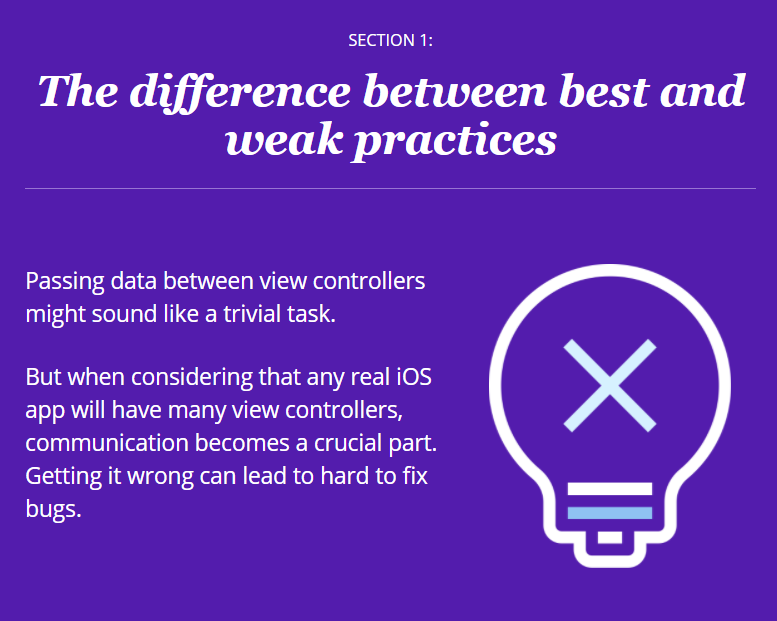
###### SECTION 5

Wrong techniques

## ***Don’t be a copy and paste developer***

**What is the difference between the few developers that become highly skilled and all the other copy-and-paste developers?**

* **Good developers know they lack knowledge and take the time to grow it.**
* **Bad developers never bother. They go on with the little they know, trying to hack together something and calling it a day.**



Here are all the ways in which you can pass data between view controllers:

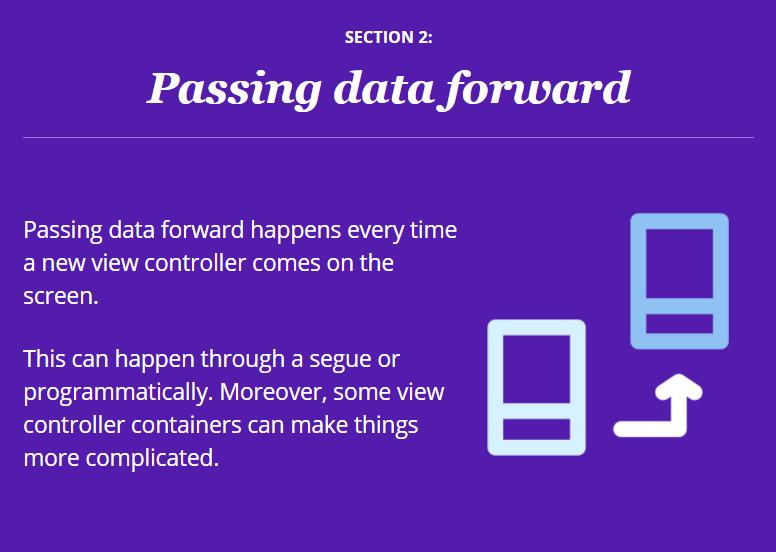
* when a segue is performed
* triggering transitions programmatically in your code
* through the state of the app
* using singletons
* using the app delegate
* assigning a delegate to a view controller
* through unwind segues
* referencing a view controller directly
* using the user defaults
* through notifications
* with Swift closures

But only a few of them are the best practices.

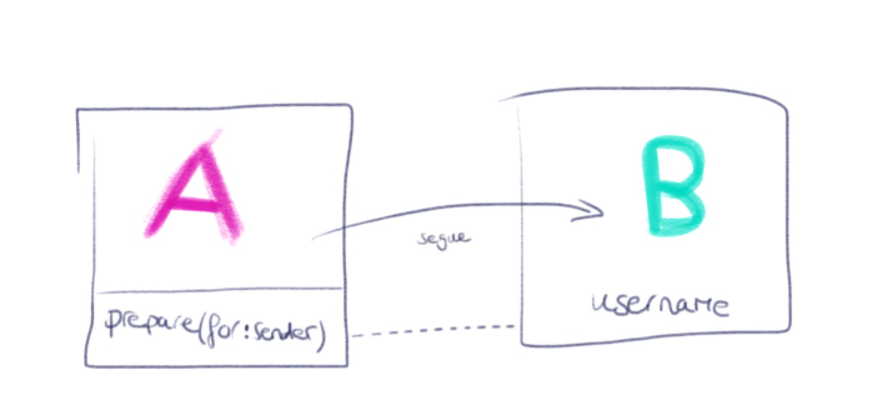
**Two directions in which data can flow.**

* **forward**, to any new view controller that comes on the screen; and
* **backwards**, to a view controller that was previously on display and to which the user goes back at some point.

The two are not equivalent. Each one has its techniques.



# Segues



**Syntax: -**

On the first ViewController class, you need to override the prepare(for segue) method:

**override func prepare(for segue: UIStoryboardSegue, sender: Any?) {**

// Create a variable to store the name the user entered on textField

**let nickName = textField.text ?? ""**

// Create a new variable to store the instance of the SecondViewController

// set the variable from the SecondViewController that will receive the data

if **let destinationVC = segue.destination as! SecondViewController {**

**destinationVC.nickName = nickname**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//**

**@IBAction func nextTapped(\_ sender: Any) {**

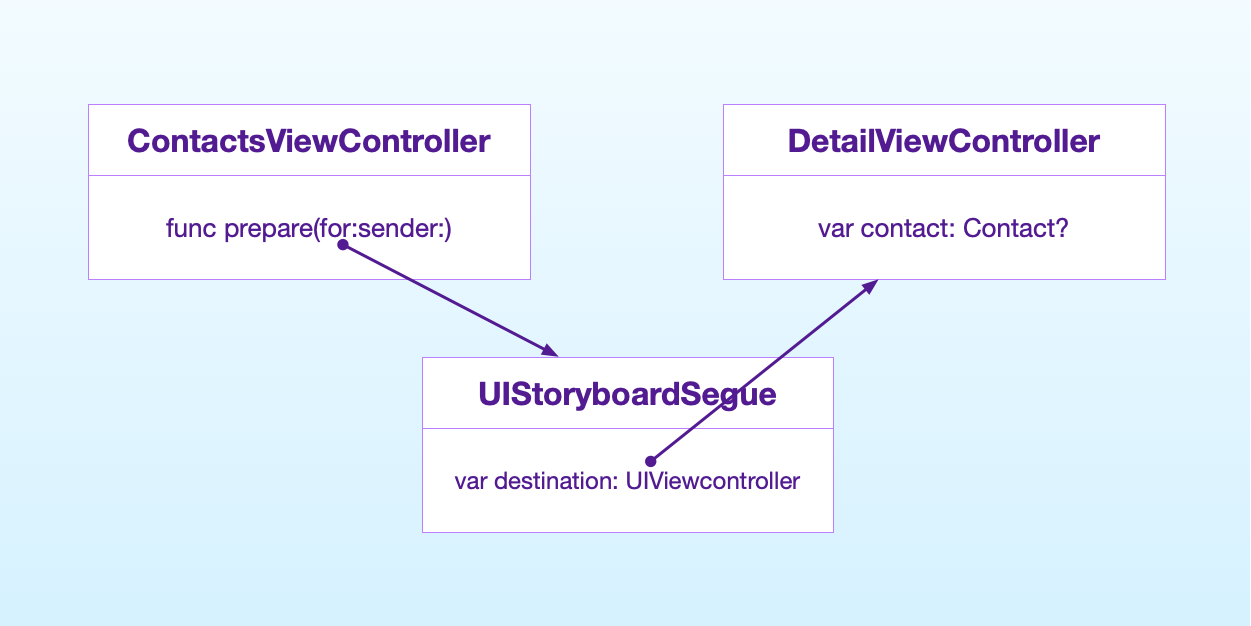
**self.performSegue(withIdentifier: "InputVCToDisplayVC", sender: self)**

**}**

The most common situation in which you need to pass data forward is when a transition happens through a storyboard segue.

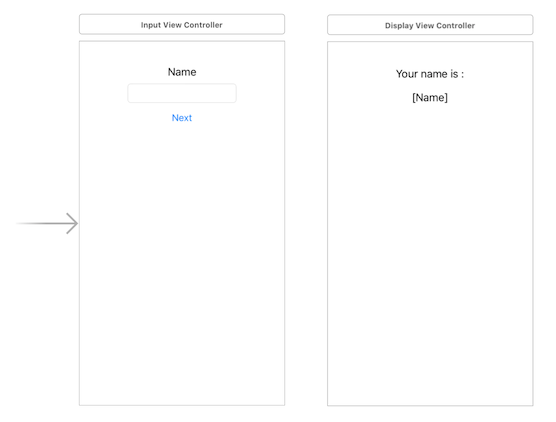
There are two different types of forward segues. The first one we will explore is a show segue in a navigation controller.

## ***Passing data forward through a show segue***

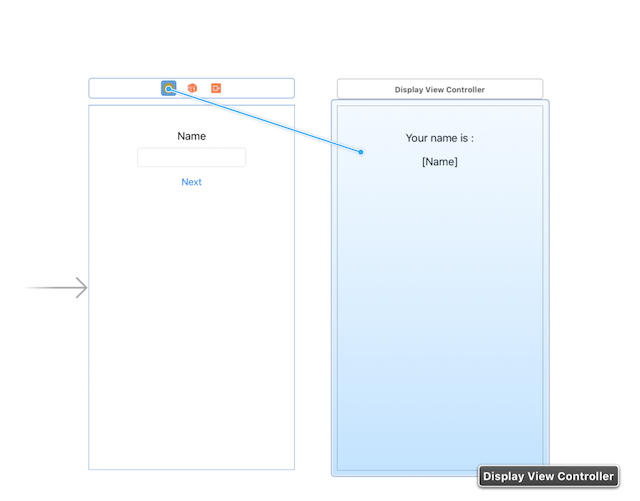


**Example: -**

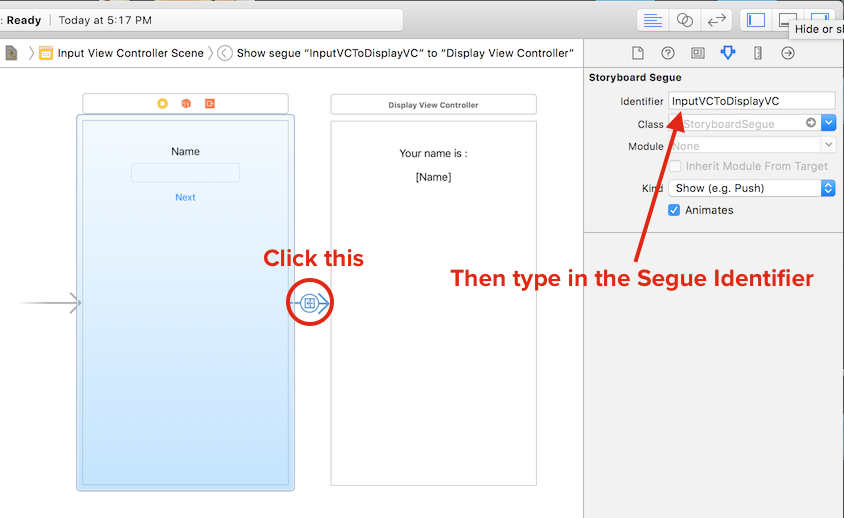
Assume you have two view controllers set up in storyboard like this,



Link the InputViewController to DisplayViewController by clicking the yellow round icon on top of InputViewController, then press and hold control and drag it to DisplayViewController. After releasing the mouse click, select '**Show**' inside Manual Segue.



Then select the segue between view controller, and set an identifier for it, we will use this identifier on the next step. Usually my naming convention will be "sourceVCtoDestinationVC".



When the **Next** button is tapped, we will tell the InputViewController to perform the segue. The segue identifier is used here. This method tells the InputViewController to perform a segue to DisplayViewController.

The **sender** parameter refer to the object which initiated this segue, we put 'self' to indicate that this segue is initiated by the InputViewController.

**class InputViewController: UIViewController {**

**...**

**@IBAction func nextTapped(\_ sender: Any) {**

**self.performSegue(withIdentifier: "InputVCToDisplayVC", sender: self)**

**}**

**...**

**}**

You can now build the app and tap **Next**, it will show the display view controller, but the data is not passed yet.

In InputViewController, if there is a MARK: - Navigation , uncomment the code below it. Else add the following code :

**override func prepare(for segue: UIStoryboardSegue, sender: Any?) {**

**}**

The prepare(for segue:, sender:) method will be called by the view controller just before a segue is performed. You can pass the data to the next view controller here.

Lets put a name string variable in Display View Controller to hold the name input data.

**class DisplayViewController: UIViewController {**

**...**

**var name: String?**

**...**

**}**

In the prepareForSegue method in InputViewController, add the following code :

**override func prepare(for segue: UIStoryboardSegue, sender: Any?) {**

**if(segue.identifier == "InputVCToDisplayVC"){**

**let displayVC = segue.destination as! DisplayViewController**

**displayVC.name = nameTextField.text**

**}**

**}**

This will set the **name** variable of the DisplayViewController before it is displayed. Since we already know the destination of the segue (the view controller to be presented) is a DisplayViewController, we can safely downcast it to DisplayViewController.

Inside the DisplayViewController **viewDidLoad** method, you can set nameLabel.text = name . Now you have successfully pass data forward from one controller to another using prepareForSegue method.

# 2

# 6

# 3

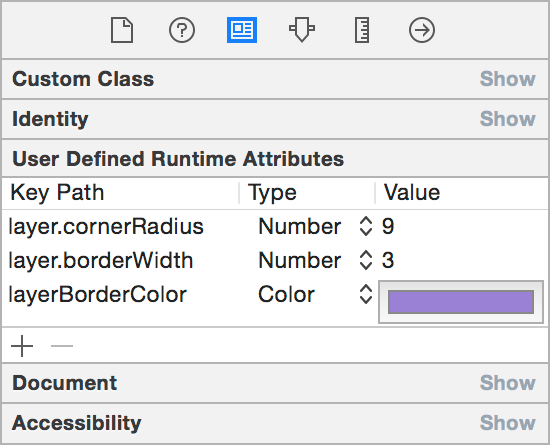
# 3

# [IBInspectable / IBDesignable](https://nshipster.com/ibinspectable-ibdesignable/)

With IBInspectable and IBDesignable, it’s possible to build a custom interface for configuring your custom controls and have them rendered in real-time while designing your project.

## **IBInspectable**

IBInspectable properties provide new access to an old feature: user-defined runtime attributes. Currently accessible from the identity inspector, these attributes have been available since before Interface Builder was integrated into Xcode. They provide a powerful mechanism for configuring any key-value coded property of an instance in a NIB, XIB, or storyboard:



While powerful, runtime attributes can be cumbersome to work with. The key path, type, and value of an attribute need to be set on each instance, without any autocompletion or type hinting, which requires trips to the documentation or a custom subclass’s source code to double-check the settings. IBInspectable properties solve this problem outright: in Xcode 6 you can now specify any property as inspectable and get a user interface built just for your custom class.

For example, these properties in a UIView subclass update the backing layer with their values:

**@IBInspectable var cornerRadius: CGFloat = 0 {**

**didSet {**

**layer.cornerRadius = cornerRadius**

**layer.masksToBounds = cornerRadius > 0**

**}**

**}**

**@IBInspectable var borderWidth: CGFloat = 0 {**

**didSet {**

**layer.borderWidth = borderWidth**

**}**

**}**

**@IBInspectable var borderColor: UIColor? {**

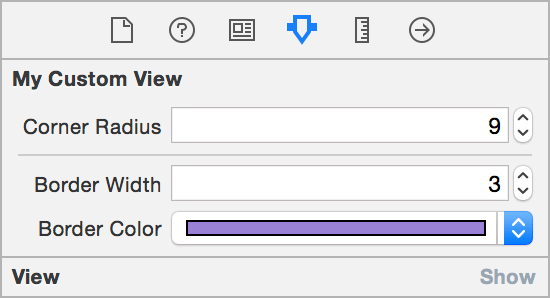
**didSet {**

**layer.borderColor = borderColor?.CGColor**

**}**

**}**

Marked with @IBInspectable (or IBInspectable in Objective-C), they are easily editable in Interface Builder’s inspector panel. Note that Xcode goes the extra mile here—property names are converted from camel- to title-case and related names are grouped together:



Since inspectable properties are simply an interface on top of user-defined runtime attributes, the same list of types is supported: booleans, strings, and numbers (i.e., NSNumber or any of the numeric value types), as well as CGPoint, CGSize, CGRect, UIColor, and NSRange, adding UIImage for good measure.

Those already familiar with runtime attributes will have noticed a bit of trickery in the example above. UIColor is the only color type supported, not the CGColor native to a view’s backing CALayer. The borderColor computed property maps the UIColor (set via runtime attribute) to the layer’s required CGColor.

### **Making Existing Types Inspectable**

Built-in Cocoa types can also be extended to have inspectable properties beyond the ones already in Interface Builder’s attribute inspector. If you like rounded corners, you’ll love this UIView extension:

**extension UIView {**

**@IBInspectable var cornerRadius: CGFloat {**

**get {**

**return layer.cornerRadius**

**}**

**set {**

**layer.cornerRadius = newValue**

**layer.masksToBounds = newValue > 0**

**}**

**}**

**}**

Presto! A configurable border radius on any UIView you create.

## **IBDesignable**

As if that weren’t enough, IBDesignable custom views also debut in Xcode 6. When applied to a UIView or NSView subclass, the @IBDesignable designation lets Interface Builder know that it should render the view directly in the canvas. This allows seeing how your custom views will appear without building and running your app after each change.

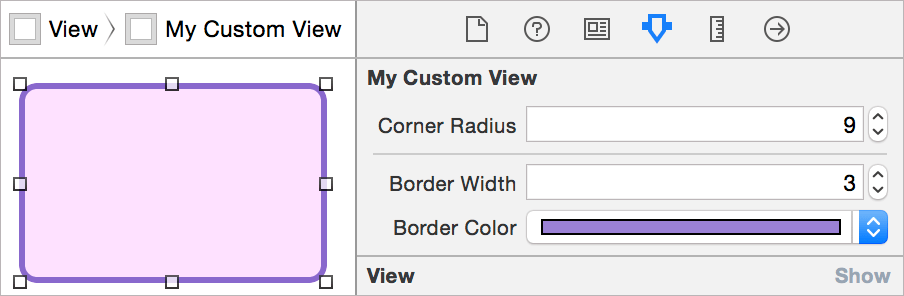
To mark a custom view as IBDesignable, prefix the class name with @IBDesignable (or the IB\_DESIGNABLE macro in Objective-C). Your initializers, layout, and drawing methods will be used to render your custom view right on the canvas:

**@IBDesignable**

**class MyCustomView: UIView {**

**...**

**}**



The time-savings from this feature can’t be overstated. Combined with IBInspectable properties, a designer or developer can easily tweak the rendering of a custom control to get the exact result she wants. Any changes, whether made in code or the attribute inspector, are immediately rendered on the canvas.

Moreover, any problems can be debugged without compiling and running the whole project. To kick off a debugging session right in place, simply set a breakpoint in your code, select the view in Interface Builder, and choose **Editor** ➔ **Debug Selected Views**.

Since the custom view won’t have the full context of your app when rendered in Interface Builder, you may need to generate mock data for display, such as a default user profile image or generic weather data. There are two ways to add code for this special context:

* prepareForInterfaceBuilder(): This method compiles with the rest of your code but is only executed when your view is being prepared for display in Interface Builder.
* TARGET\_INTERFACE\_BUILDER: The #if TARGET\_INTERFACE\_BUILDER preprocessor macro will work in either Objective-C or Swift to conditionally compile the right code for the situation:

#if !TARGET\_INTERFACE\_BUILDER

// this code will run in the app itself

#else

// this code will execute only in IB

#endif

**Example for UIView**

**@IBInspectable var cornerRadius: Double {  
 get {  
 return Double(self.layer.cornerRadius)  
 }set {  
 self.layer.cornerRadius = CGFloat(newValue)  
 }  
}**

**@IBInspectable var borderWidth: Double {  
 get {  
 return Double(self.layer.borderWidth)  
 }  
 set {  
 self.layer.borderWidth = CGFloat(newValue)  
 }  
}**

**@IBInspectable var borderColor: UIColor? {  
 get {  
 return UIColor(cgColor: self.layer.borderColor!)  
 }  
 set {  
 self.layer.borderColor = newValue?.cgColor  
 }  
}**

**@IBInspectable var shadowColor: UIColor? {  
 get {  
 return UIColor(cgColor: self.layer.shadowColor!)  
 }  
 set {  
 self.layer.shadowColor = newValue?.cgColor  
 }  
}**

**@IBInspectable var shadowOpacity: Float {  
 get {  
 return self.layer.shadowOpacity  
 }  
 set {  
 self.layer.shadowOpacity = newValue  
 }  
}**

## **Also we can add code for animation Effects for our View inside custom class -**

private func **provideAnimation**(**animationDuration**:TimeInterval, **deleyTime**:TimeInterval, **springDamping**:CGFloat, **springVelocity**:CGFloat)**{**  
 self.transform = CGAffineTransform(scaleX: 0.1, y: 0.1)  
 **UIView**.animate(**withDuration**: animationDuration,  
 **delay**: deleyTime,  
 **usingSpringWithDamping**: springDamping,  
 **initialSpringVelocity**: springVelocity,  
 **options**: .allowUserInteraction,  
 **animations**: **{**  
 self.transform = CGAffineTransform.identity  
 **})**  
 **}**