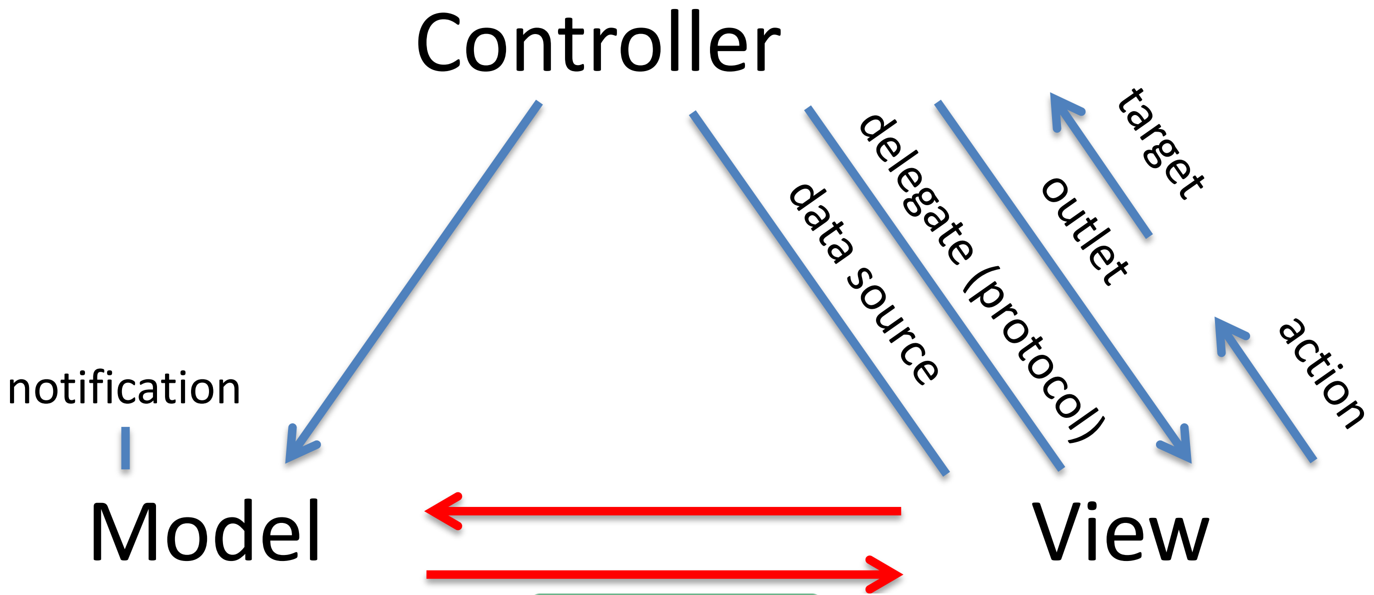
iOS

Some first elements

Model View Controller



Controller

notification

Model

View

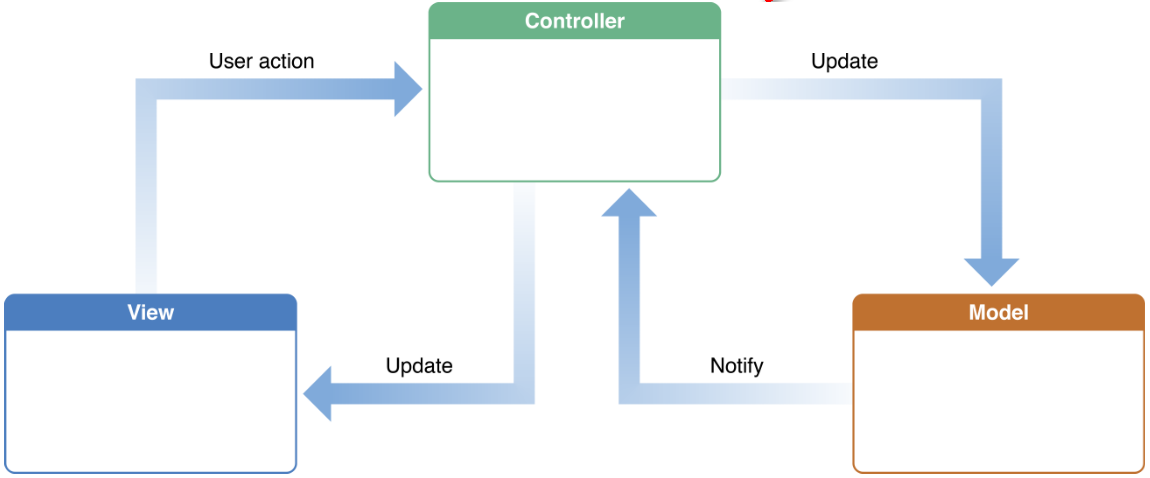
data source

delegate (protocol)

outlet

target

action



Controller

User action

Update

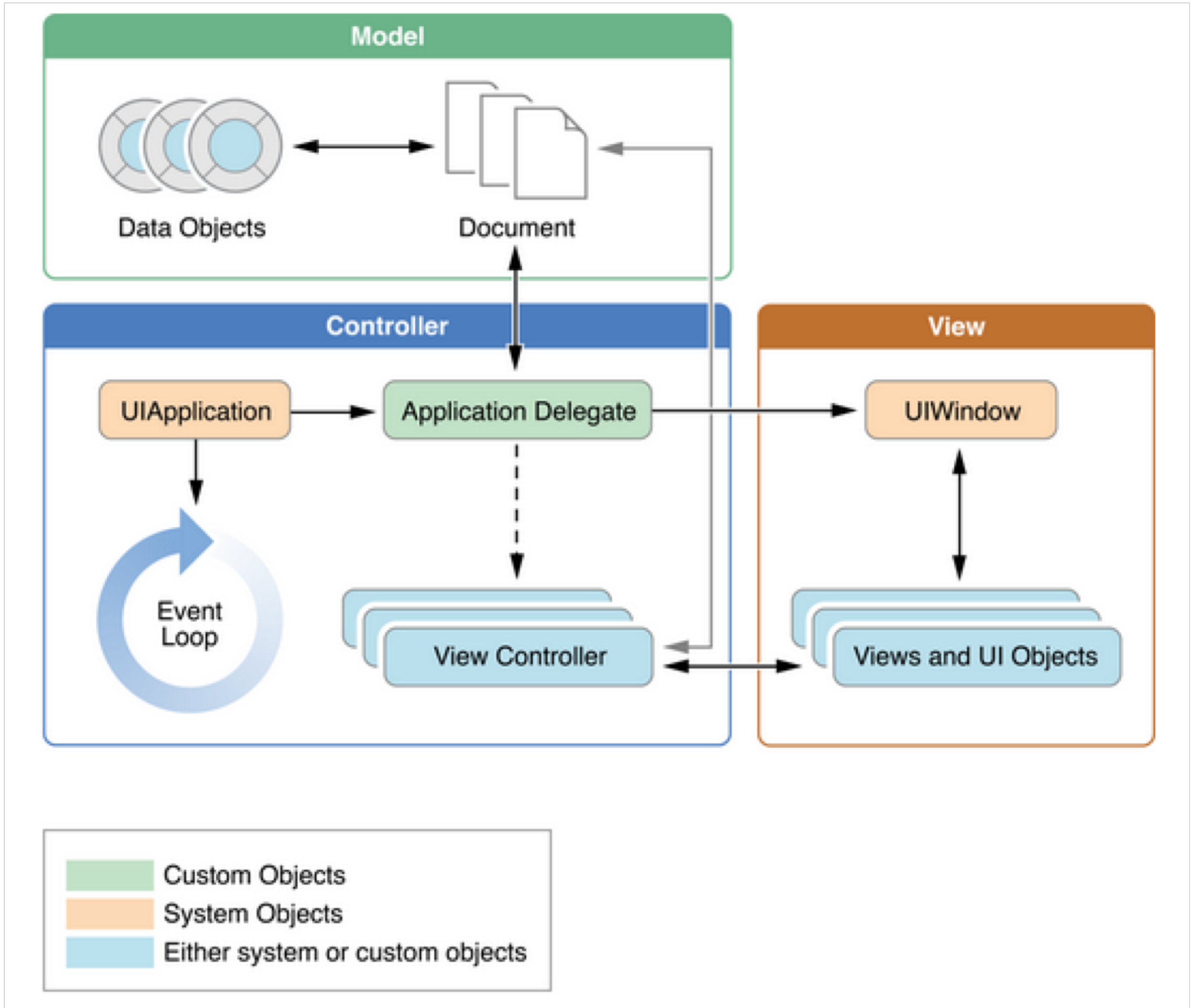
View

Update

Notify

Model

App structure



ApplicationDelegate

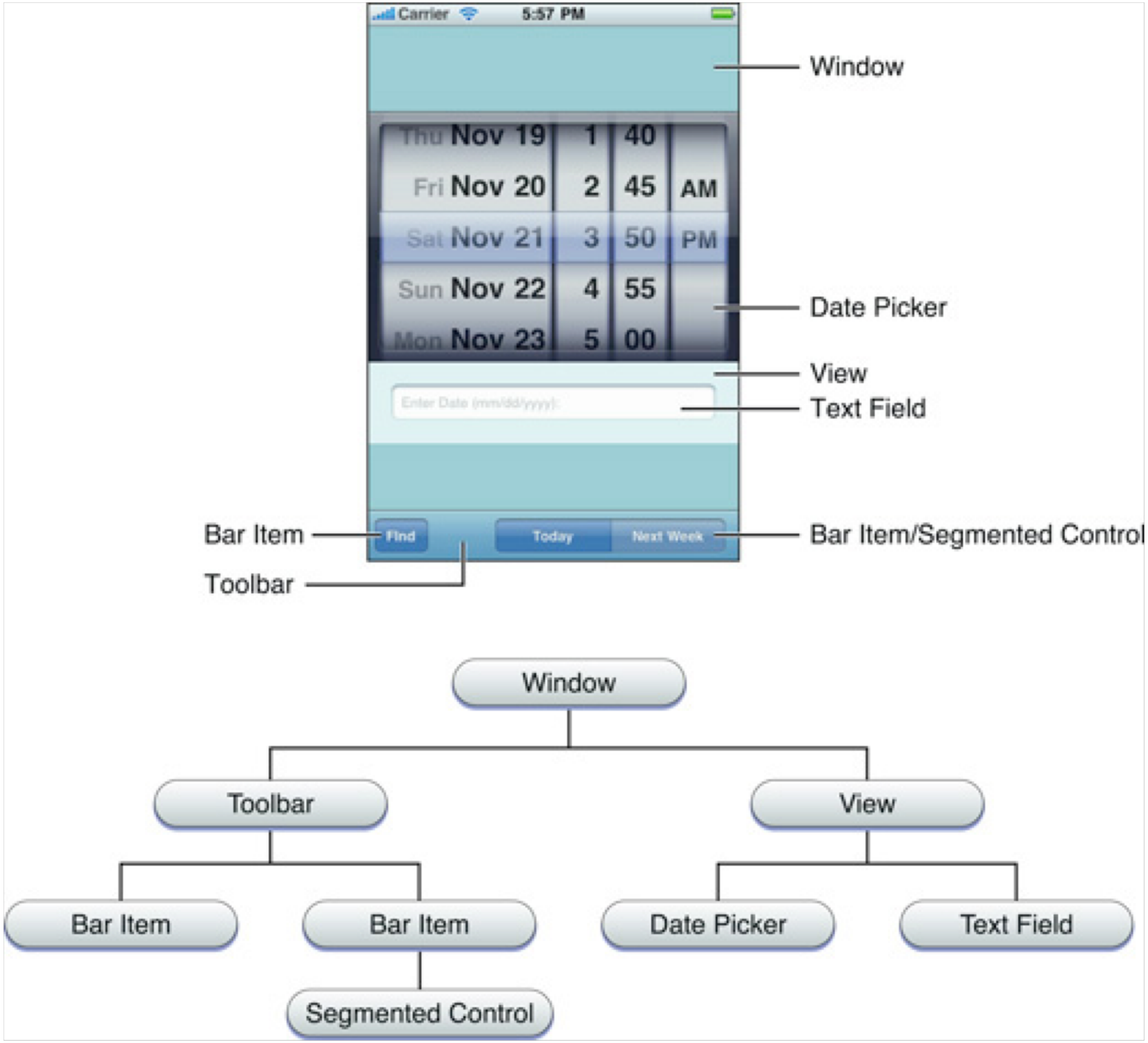
- A single window, where all of our app content is drawn
- "Skeletons" of important methods that allow the application object to talk to the app delegate
  - During runtime events (e.g., app launch, low-memory warnings, and app termination) the application object calls the corresponding method in the app delegate, giving it an opportunity to respond appropriately
- Delegate design pattern

AppDelegate.swift

```swift
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

   var window: UIWindow?

   func application(application: UIApplication, didFinishLaunchingWithOptions
                 launchOptions: [NSObject: AnyObject]?) -> Bool {
   // Override point for customization after application launch.
   return true}

   func applicationWillResignActive(application: UIApplication) {
   // Sent when the application is about to move from active
   // to inactive state}

   func applicationDidEnterBackground(application: UIApplication) {
   // Use this method to release shared resources, save user data,
   // invalidate timers, and store application state}

   func applicationWillEnterForeground(application: UIApplication) {
   // Called as part of the transition from the background to the inactive
   // state}

   func applicationDidBecomeActive(application: UIApplication) {
   // Restart any tasks that were paused while the application was inactive}

   func applicationWillTerminate(application: UIApplication) {
   // Called when the application is about to terminate}
}
```

Views

- Represent user interface elements that display contents or respond to user events
  - Can be nested in a view hierarchy
  - Can animate their property values
- Views do not know the role they play
  - For example, clicking a button is always the same, it does not know what it controls


Carrier
5:57 PM
Thu Nov 19
Fri Nov 20
Sat Nov 21
Sun Nov 22
Mon Nov 23
1
2
3
4
5
40
45
50
55
00
AM
PM
Enter Date (mm/dd/yyyy):
Find
Today
Next Week
Window
Date Picker
View
Text Field
Bar Item
Bar Item/Segmented Control
Toolbar
Window
Toolbar
View
Bar Item
Bar Item
Date Picker
Text Field
Segmented Control

View Controllers

- Provide the infrastructure for managing content and for coordinating the showing and hiding of it
  - Manage the views used to display content
  - Communicate and coordinate with other view controllers when transitions occur
- Different view controllers can control separate portions of your user interface
- May also communicate with other controllers, such as data controllers or document objects

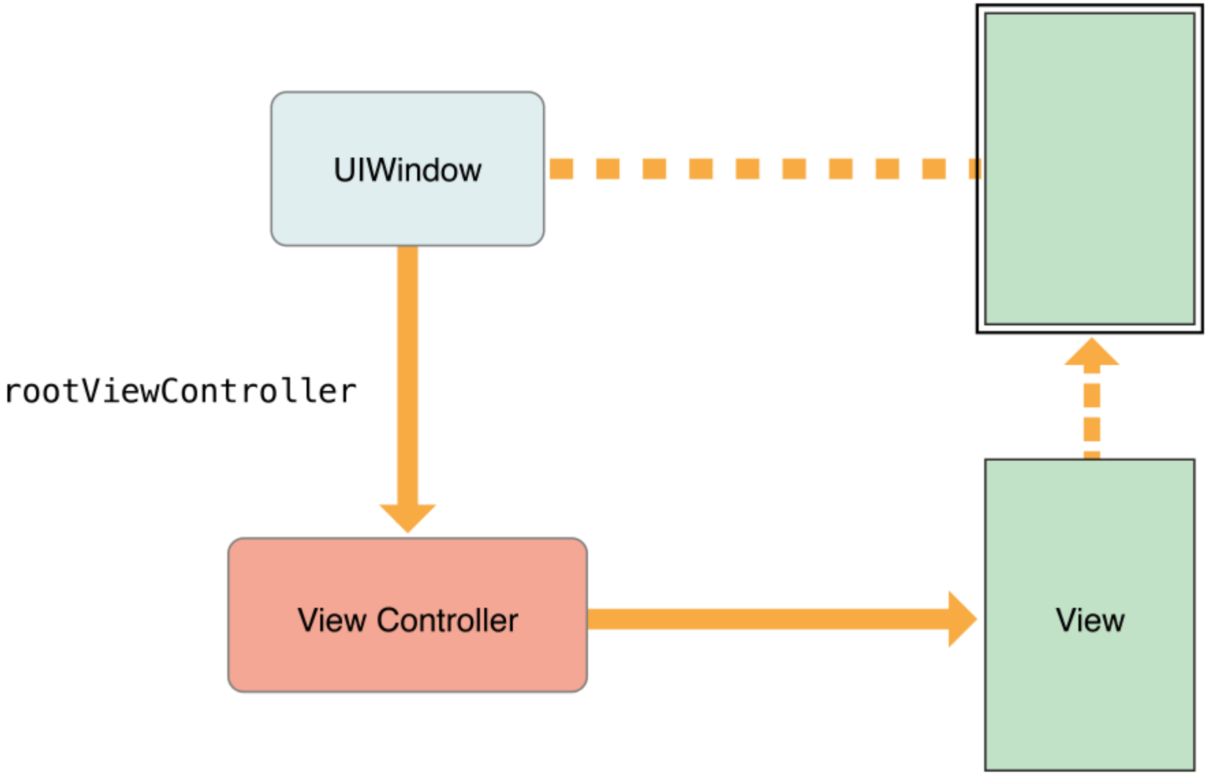
View Controllers

- You use custom subclasses of UIViewController to present your app’s content
  - Content view controllers
    - UIViewController, UITableViewController, UICollectionViewController
  - Container view controllers
    - UINavigationController, UITabBarController, UISplitViewController

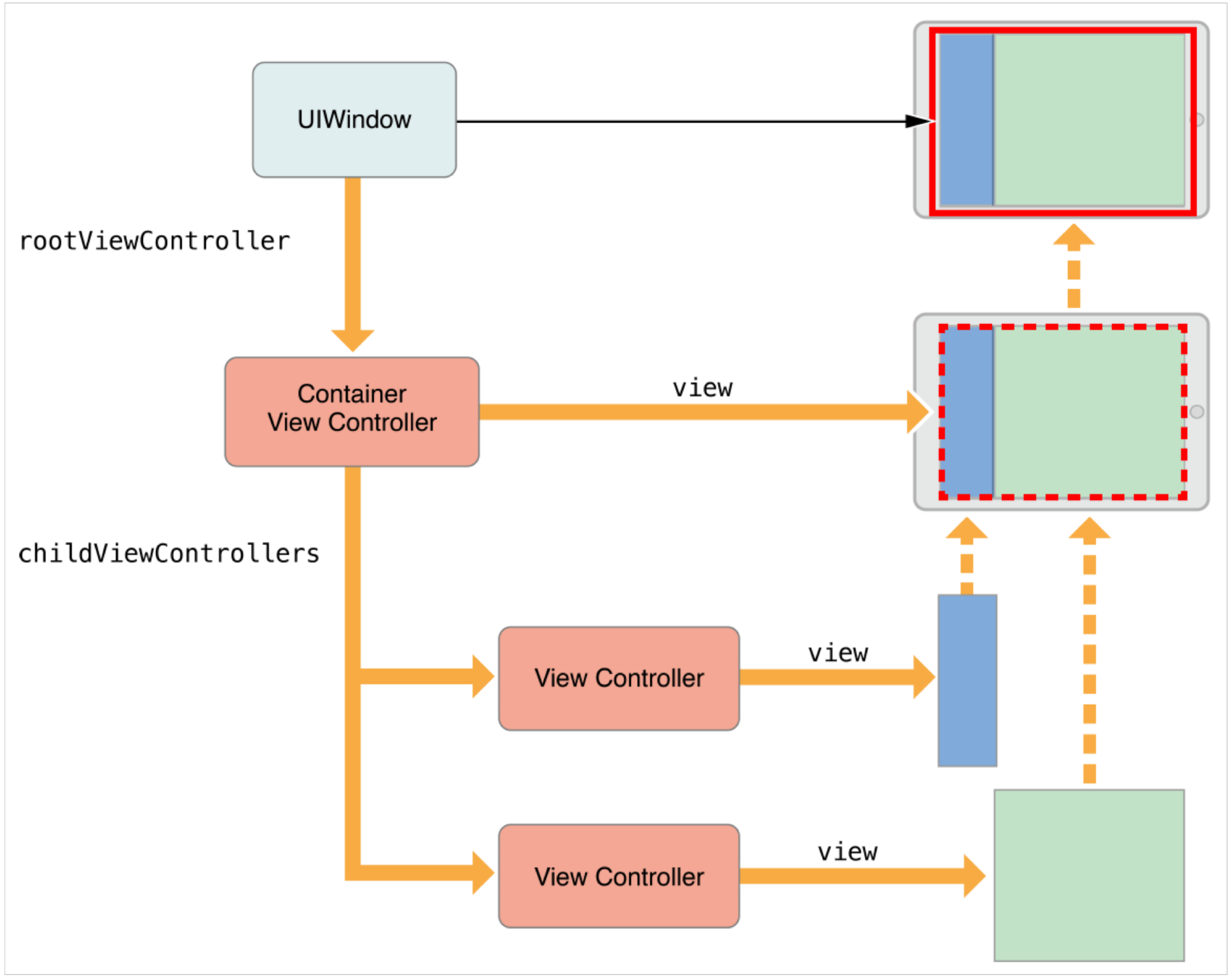
Views and View Controllers

- Every view is controlled by only one view controller
  - When a view is assigned to the view controller’s view property, the view controller owns it
- If the view is a subview, it might be controlled by the same view controller or a different view controller

Graphical elements (I)

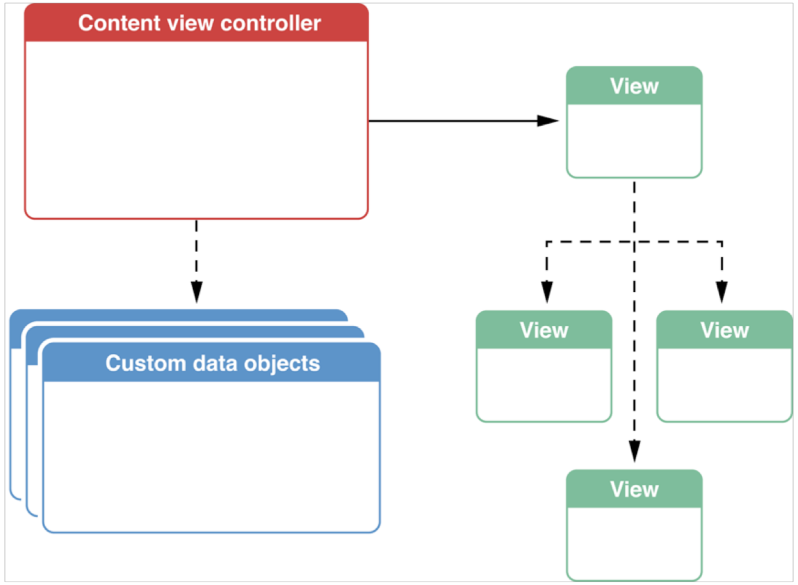


View Controller Hierarchy (II)



Content View Controllers

- Present content on the screen using a view or a group of views organized into a view hierarchy
  - Each controller is responsible for managing all the views in a single view hierarchy
  - A single controller should never manage multiple screens



UITableViewController

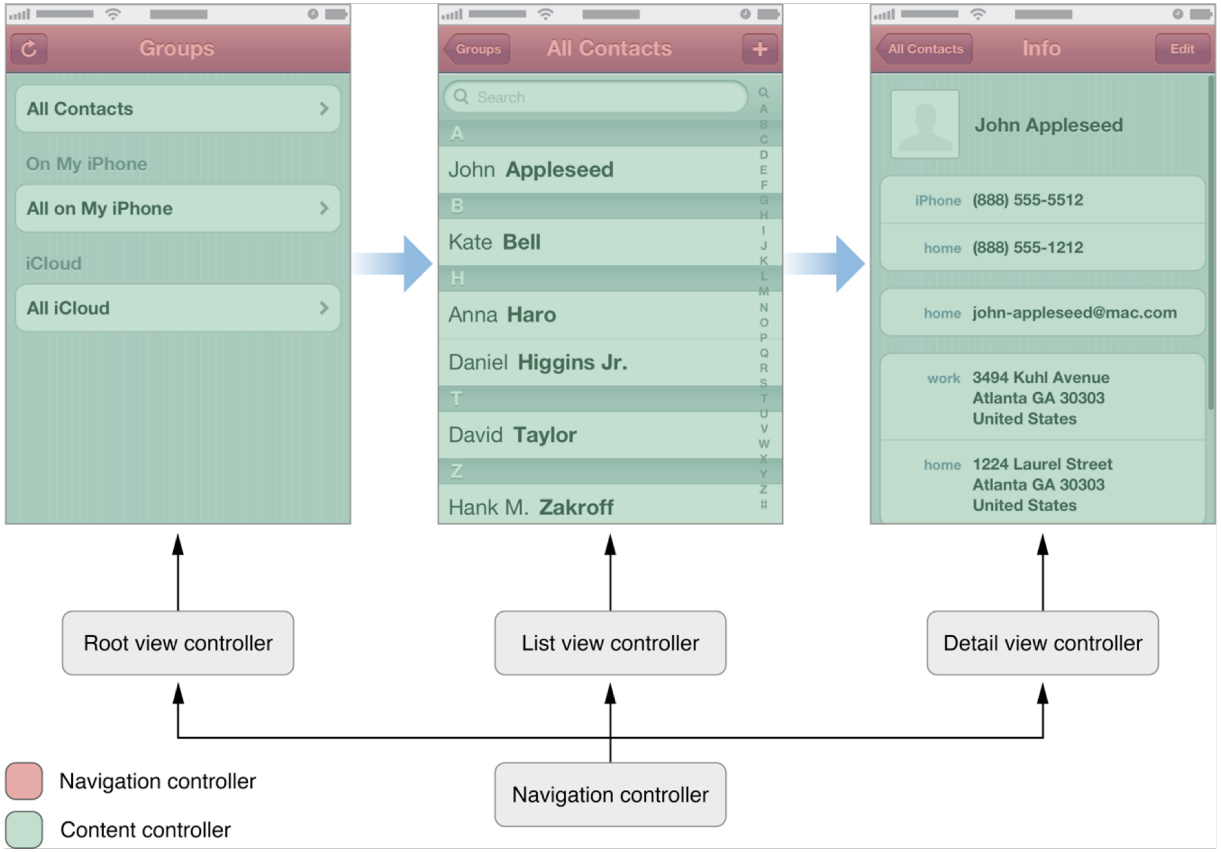
- A built-in controller designed for managing tabular data
  - Manages a table view and adds support for many standard table-related behaviors
    - A table view presents data in a single-column list of multiple rows and is a means for displaying and editing hierarchical lists of information
  - Has a pointer to the root view of the interface, but it also has a separate pointer to the table view

UICollectionViewController

- Represents a view controller whose content consists of a collection view
  - Displays an ordered collection of data
- Similar to a table view displays data using a combination of cell, layout, and supplementary views
  - can display items in a grid or in a custom layout that you design
  - Each cell must be an instance of UICollectionViewCell

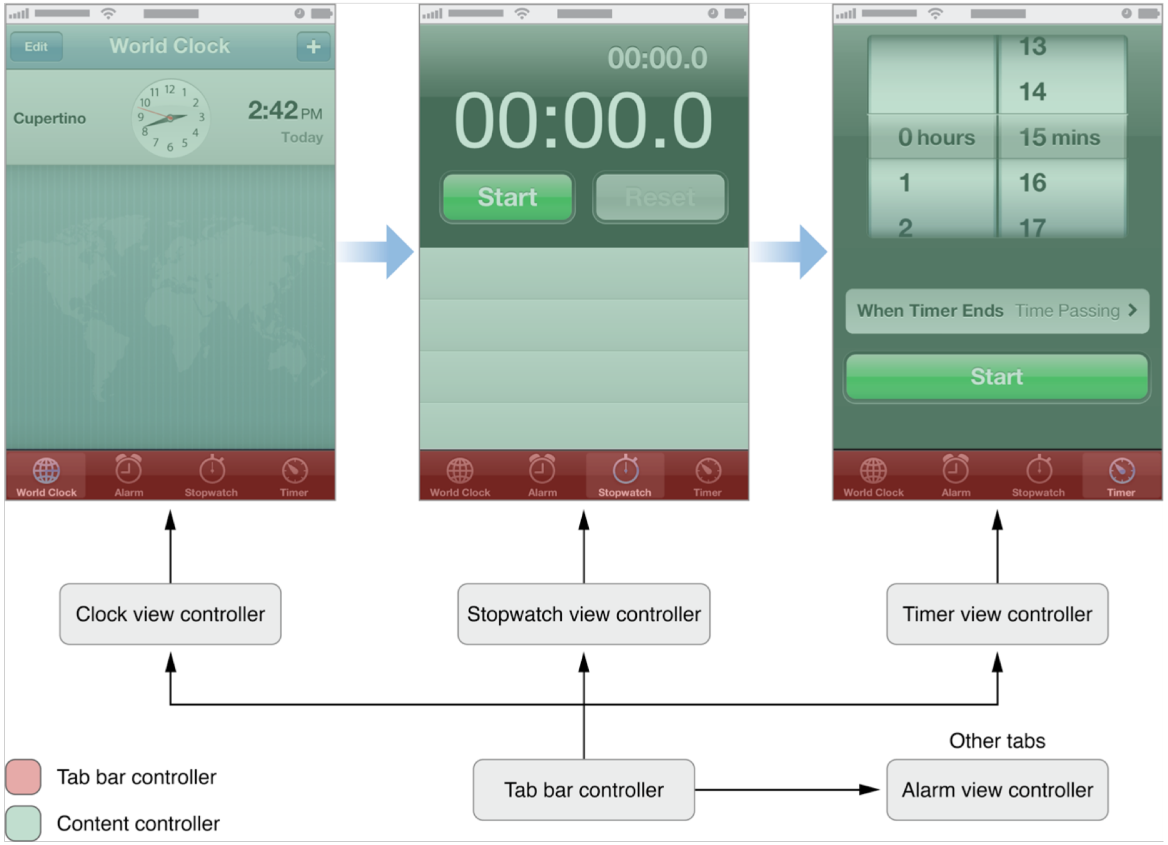
UINavigationController

- Presents data organized hierarchically
- Provides methods for managing a stack-based collection of content view controllers



UITabBarController

- Used to divide your app into distinct modes of operation
- The tab bar has multiple tabs, each represented by a child view controller



Clock view controller

Stopwatch view controller

Timer view controller

Other tabs

Tab bar controller

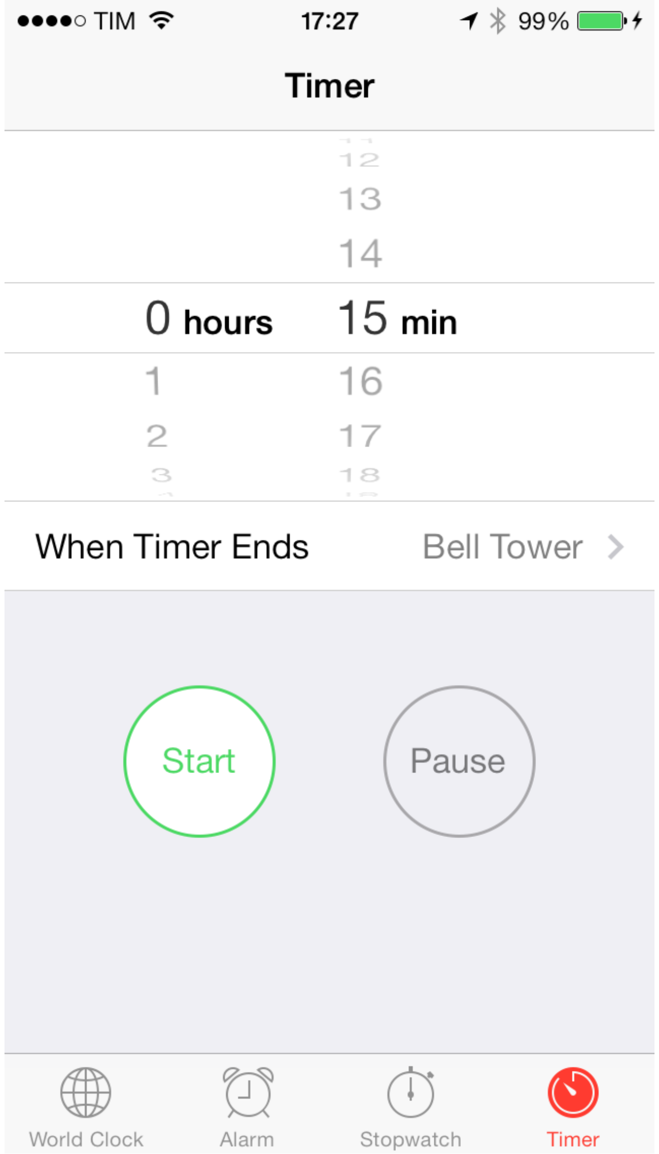
Alarm view controller

🟥 Tab bar controller

🟩 Content controller

UITabBarController

- What if we had more than 4 View Controllers?
  - A More button appears
- Everything happens automatically



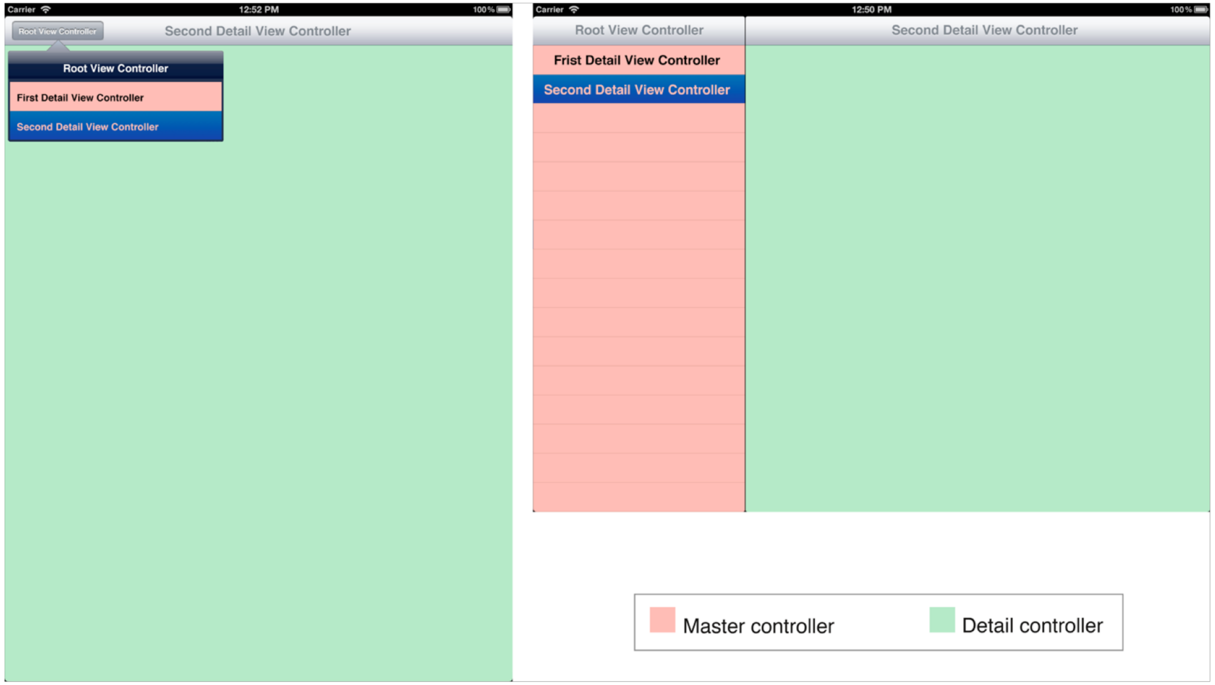
Timer

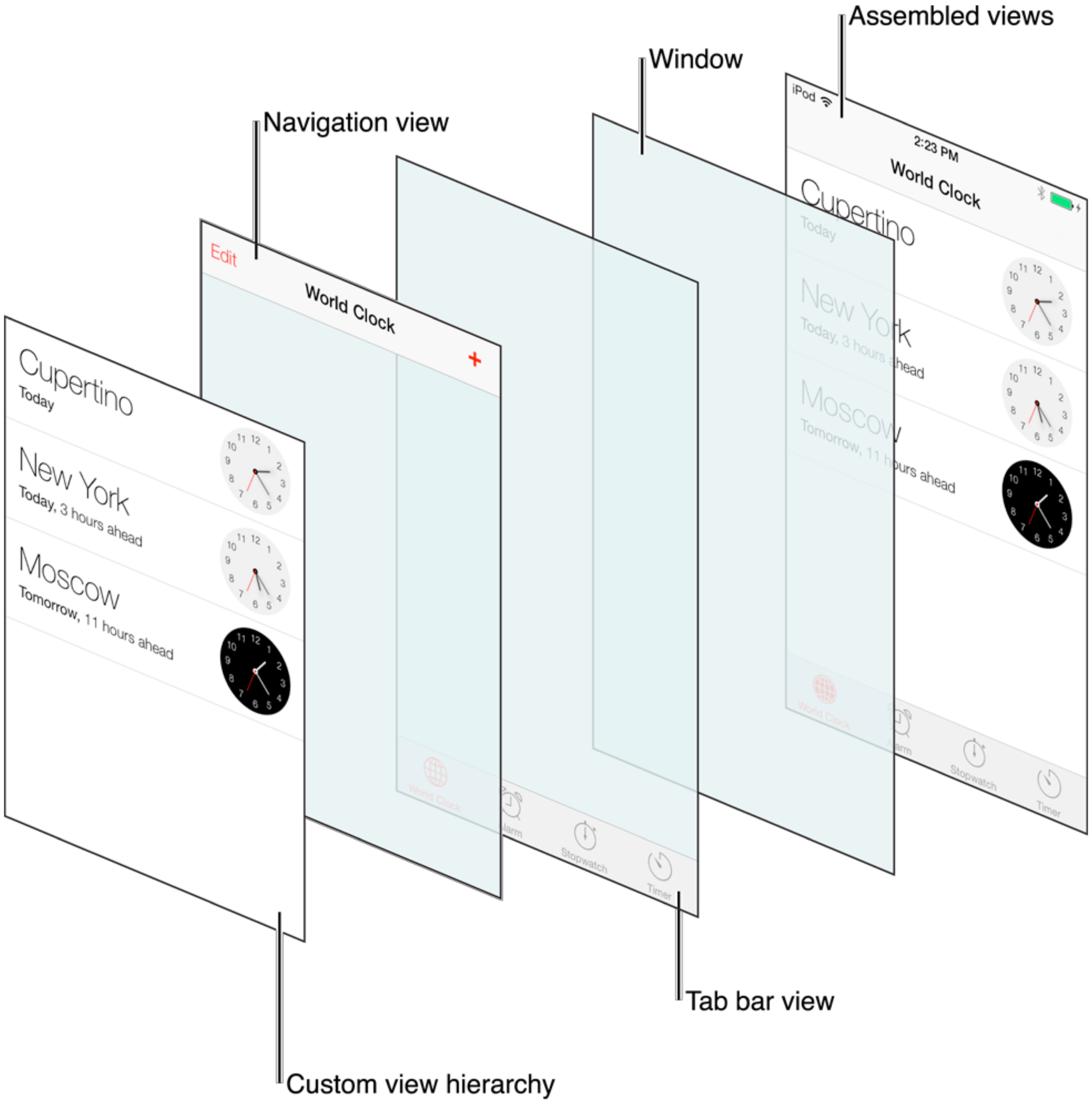
0 hours 15 min

When Timer Ends Bell Tower

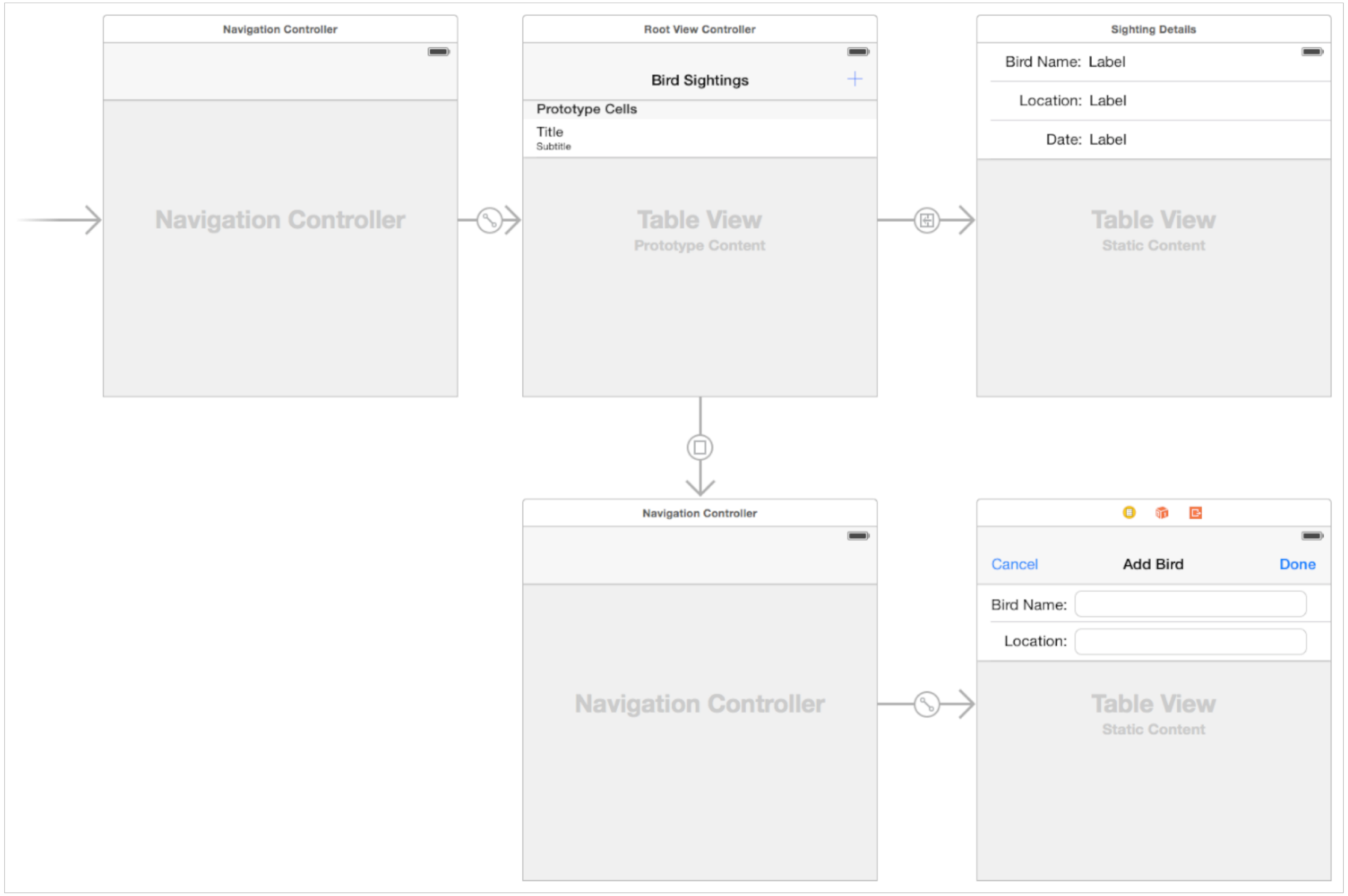
UISplitViewController

- It presents a master-detail interface
  - Changes in the primary view controller (the master) drive changes in a secondary view controller (the detail)


Carrier
12:52 PM
100%
Root View Controller
Second Detail View Controller
Root View Controller
First Detail View Controller
Second Detail View Controller
Carrier
12:50 PM
100%
Root View Controller
Second Detail View Controller
Frist Detail View Controller
Second Detail View Controller
🟥 Master controller
🟩 Detail controller


Assembled views
Window
Navigation view
Tab bar view
Custom view hierarchy
Edit
World Clock
+
Cupertino
Today
New York
Today, 3 hours ahead
Moscow
Tomorrow, 11 hours ahead
World Clock
Alarm
Stopwatch
Timer
iPod
2:23 PM

Overall organization (storyboard)

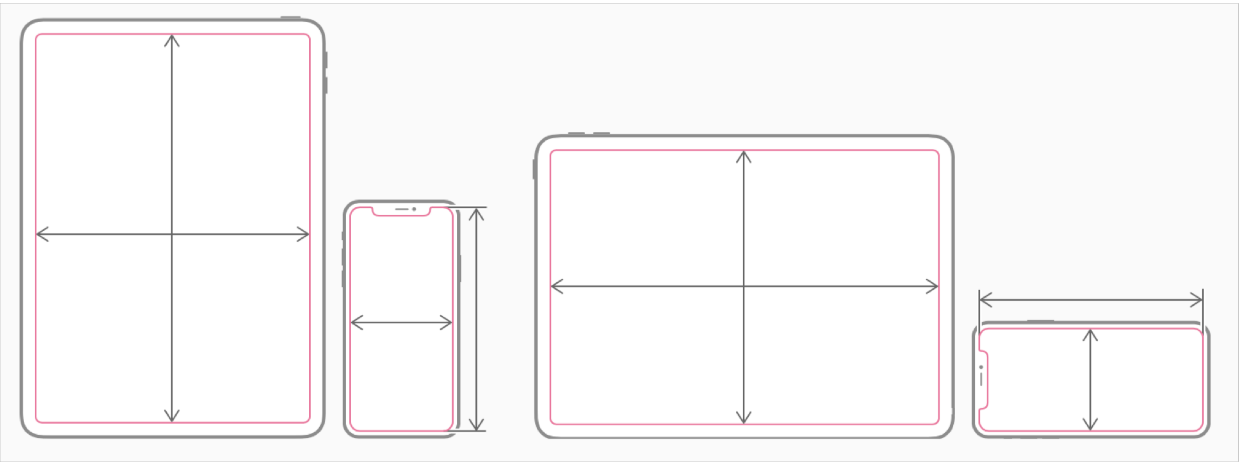


Unified Storyboards for Universal Apps

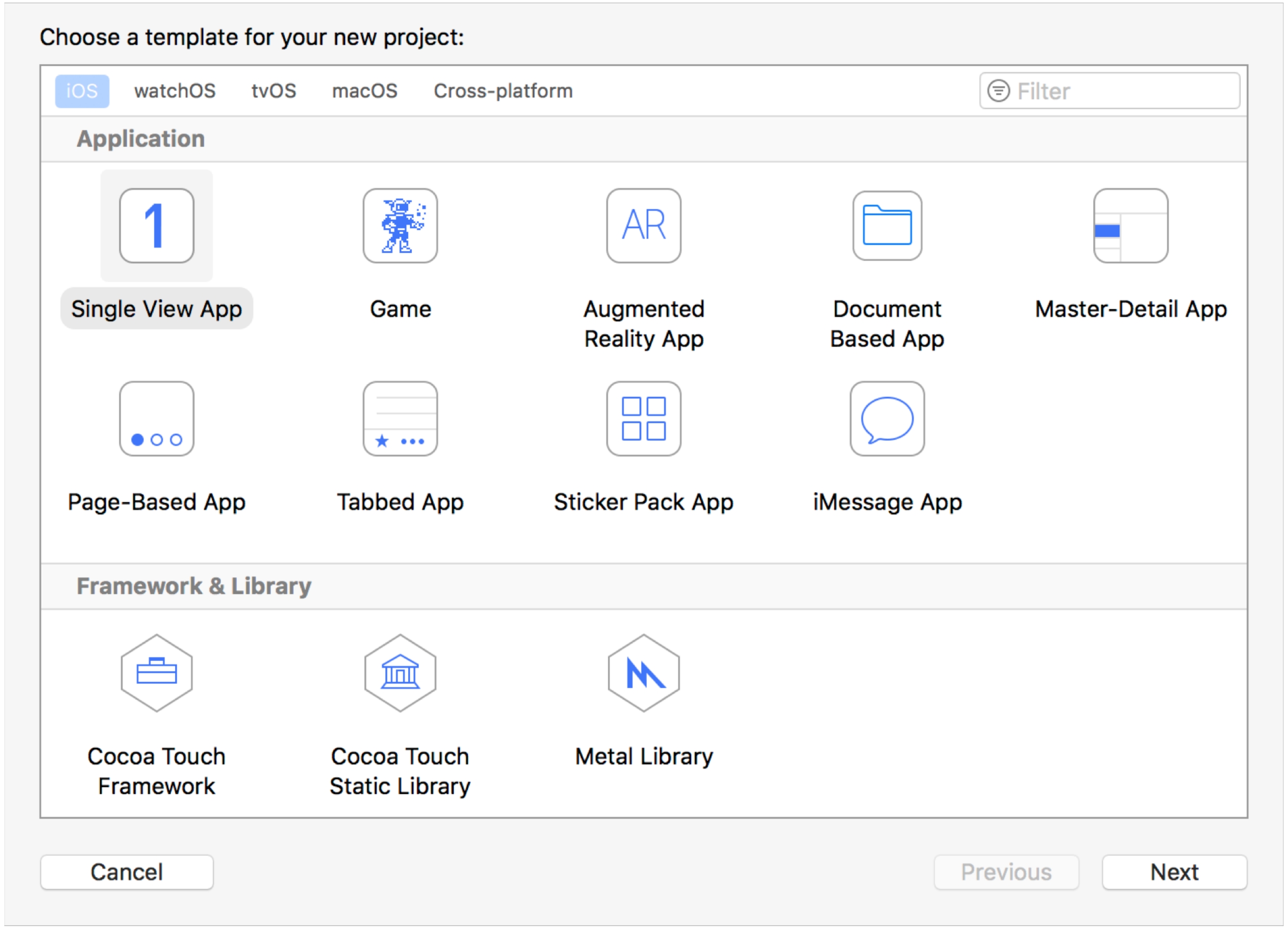
- Create a single interface for your app that works well on both iPad and iPhone, adjusting to orientation changes and different screen sizes as needed
- Design apps with a common interface and then customize them for different size classes

Size Classes

- Size classes are traits that are automatically assigned to content areas based on their size
- A view may possess any combination of size classes
  - Regular width, regular height
  - Compact width, compact height
  - Regular width, compact height
  - Compact width, regular height

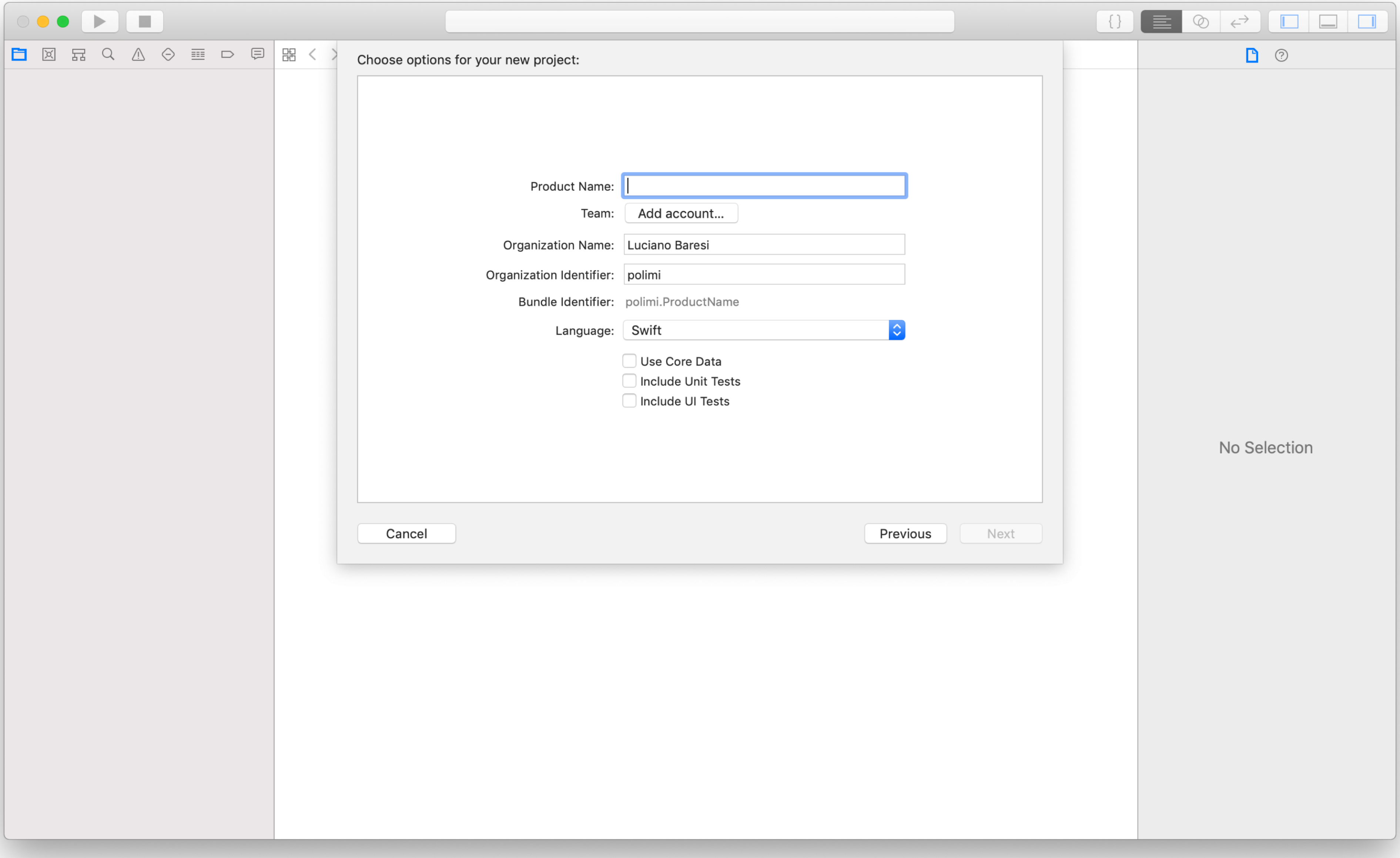


How to start



Choose a template for your new project:

Two choices



Choose options for your new project:

Product Name: |

Team: Add account...

Organization Name: Luciano Baresi

Organization Identifier: polimi

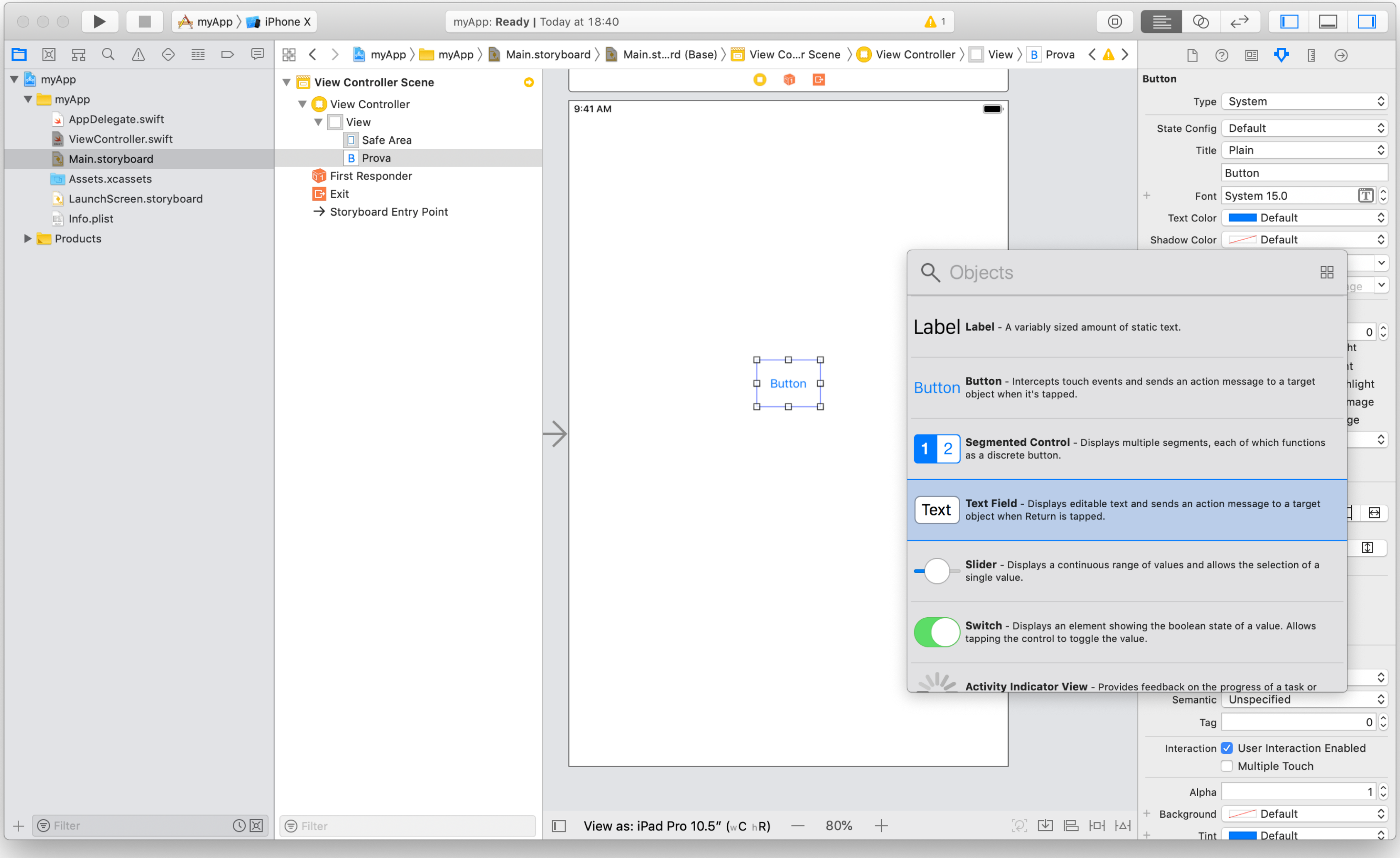
Bundle Identifier: polimi.ProductName

Language: Swift

□ Use Core Data

□ Include Unit Tests

□ Include UI Tests



Button

Type: System

State Config: Default

Title: Plain

Button

Font: System 15.0

Text Color: Default

Shadow Color: Default

Semantic: Unspecified

Tag: 0

Interaction: ☑ User Interaction Enabled

□ Multiple Touch

Alpha: 1

Background: Default

Tint: Default

Label **Label** - A variably sized amount of static text.

Button **Button** - Intercepts touch events and sends an action message to a target object when it's tapped.

1 2 **Segmented Control** - Displays multiple segments, each of which functions as a discrete button.

Text **Text Field** - Displays editable text and sends an action message to a target object when Return is tapped.

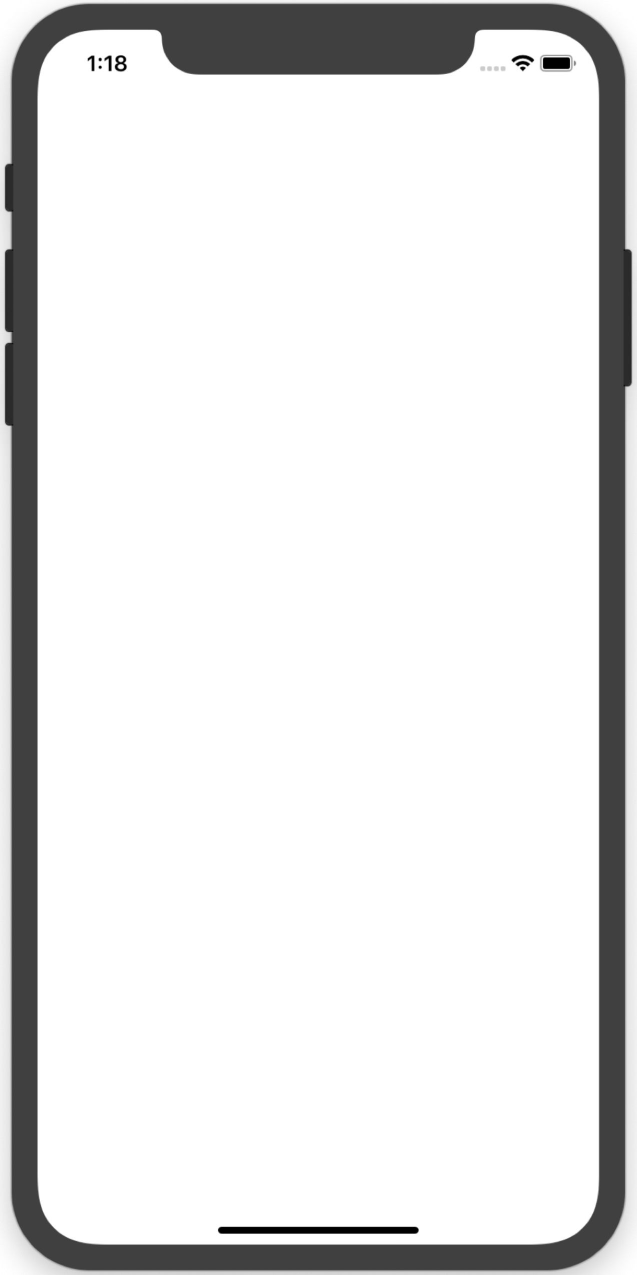
Slider - Displays a continuous range of values and allows the selection of a single value.

Switch - Displays an element showing the boolean state of a value. Allows tapping the control to toggle the value.

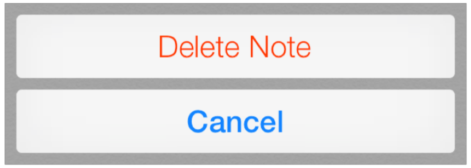
Activity Indicator View - Provides feedback on the progress of a task or

iOS Simulator

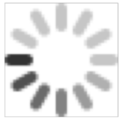
- Easy to start and try apps
- Known problems
  - Different devices
  - Different orientations
- Better positioning of elements



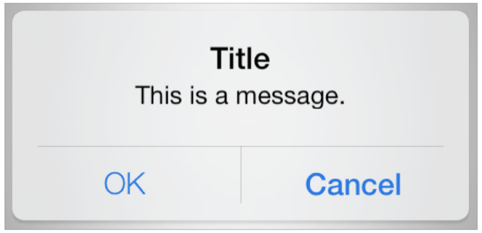
Views (I)



Action sheet



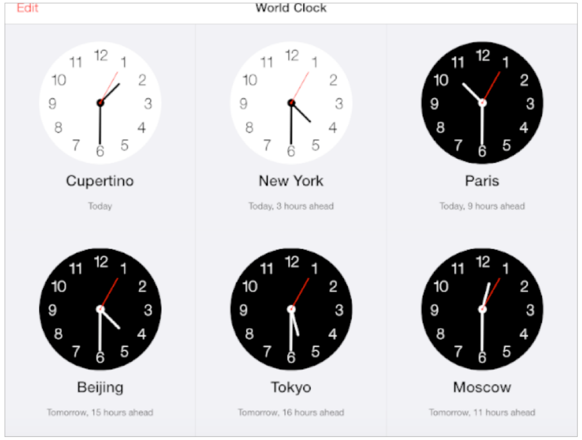
Activity indicator



Title

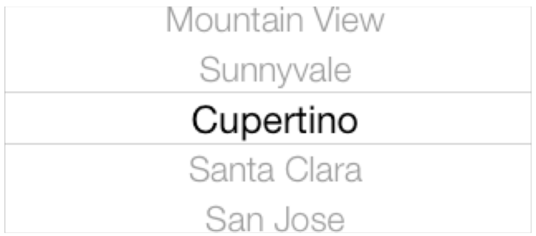
This is a message.

Alert view

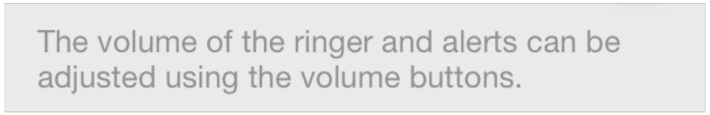


Collection view

Image view

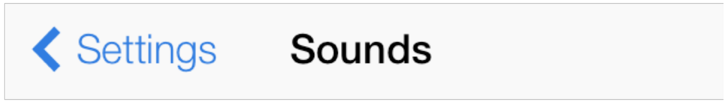


Picker view



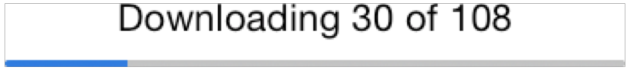
The volume of the ringer and alerts can be adjusted using the volume buttons.

Label



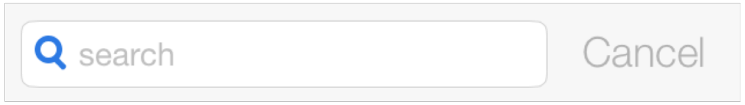
Navigation bar and items

Views (II)

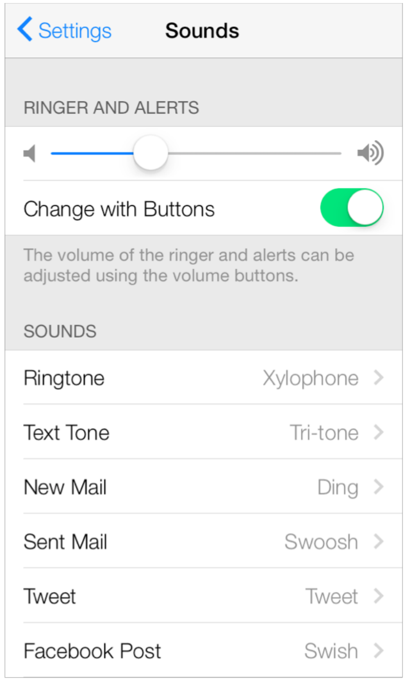


Downloading 30 of 108

Progress view



Search bar



RINGER AND ALERTS

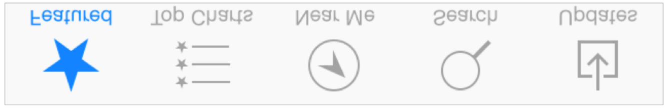
Change with Buttons

The volume of the ringer and alerts can be adjusted using the volume buttons.

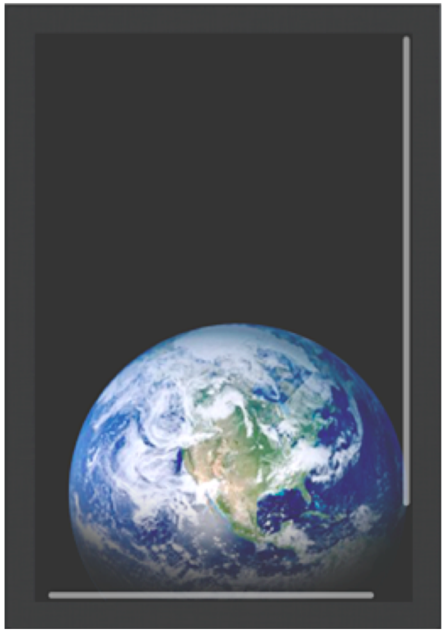
SOUNDS

Ringtone	Xylophone
Text Tone	Tri-tone
New Mail	Ding
Sent Mail	Swoosh
Tweet	Tweet
Facebook Post	Swish

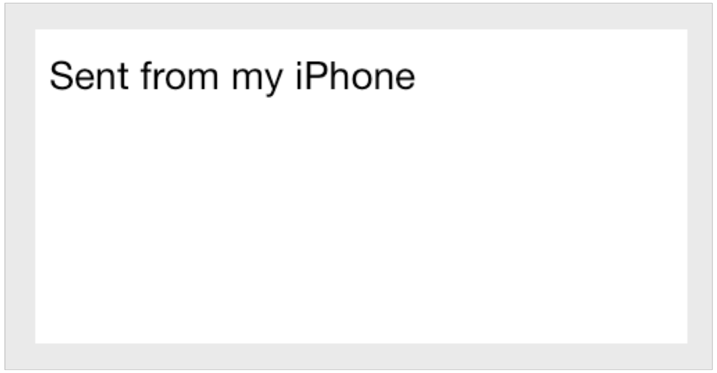
Table view



Tab bar

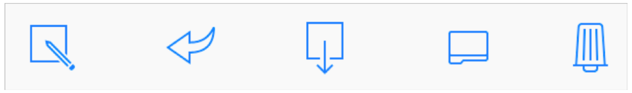


Scroll view

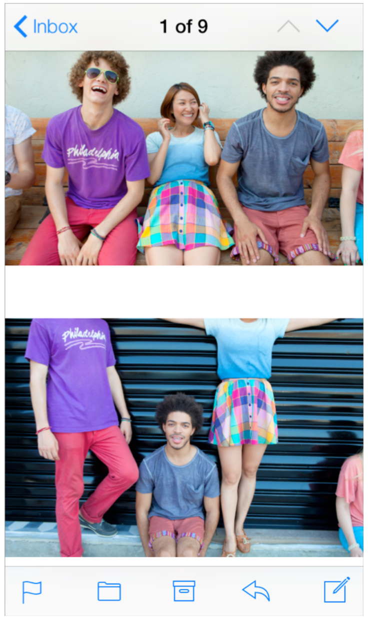


Sent from my iPhone

Text view



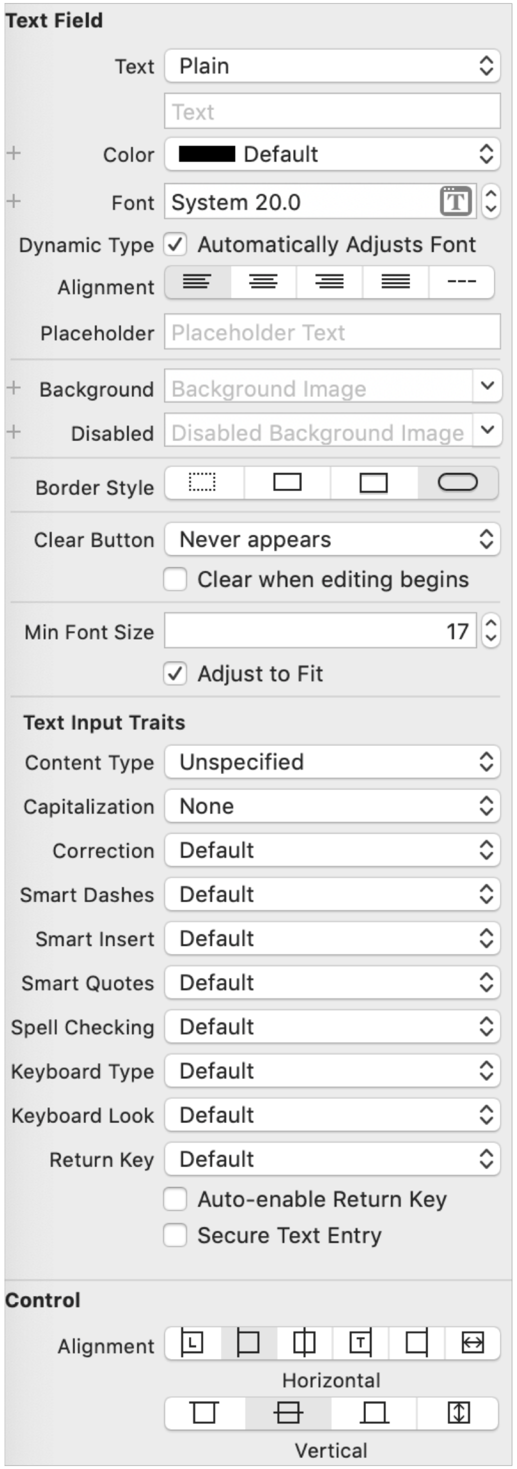
Tool bar



Web view

UITextView

- Implements the behavior of a scrollable, multiline text region
- Supports the display of text using custom style information and also supports text editing
- The appearance of the keyboard itself can be customized using its properties



Text Field

Text: Plain

Text

Color: Default

Font: System 20.0

Dynamic Type: ☑ Automatically Adjusts Font

Alignment

Placeholder: Placeholder Text

Background: Background Image

Disabled: Disabled Background Image

Border Style

Clear Button: Never appears

□ Clear when editing begins

Min Font Size: 17

☑ Adjust to Fit

Text Input Traits

Content Type: Unspecified

Capitalization: None

Correction: Default

Smart Dashes: Default

Smart Insert: Default

Smart Quotes: Default

Spell Checking: Default

Keyboard Type: Default

Keyboard Look: Default

Return Key: Default

□ Auto-enable Return Key

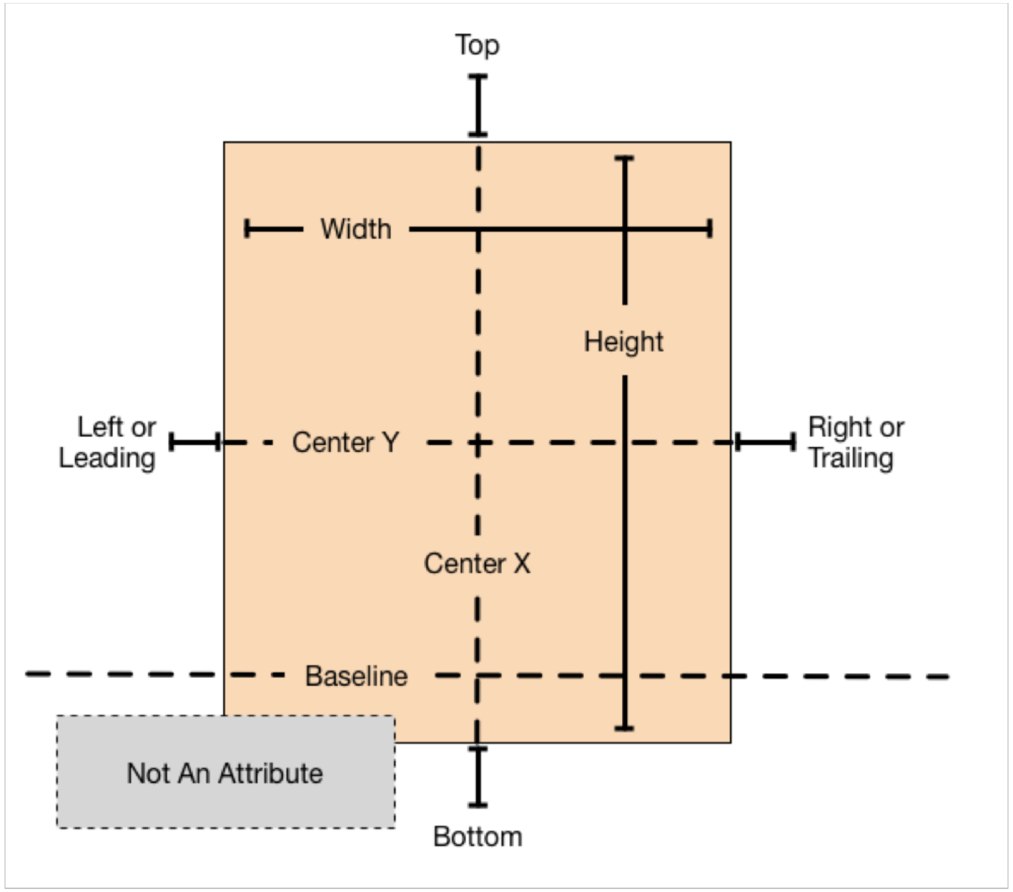
□ Secure Text Entry

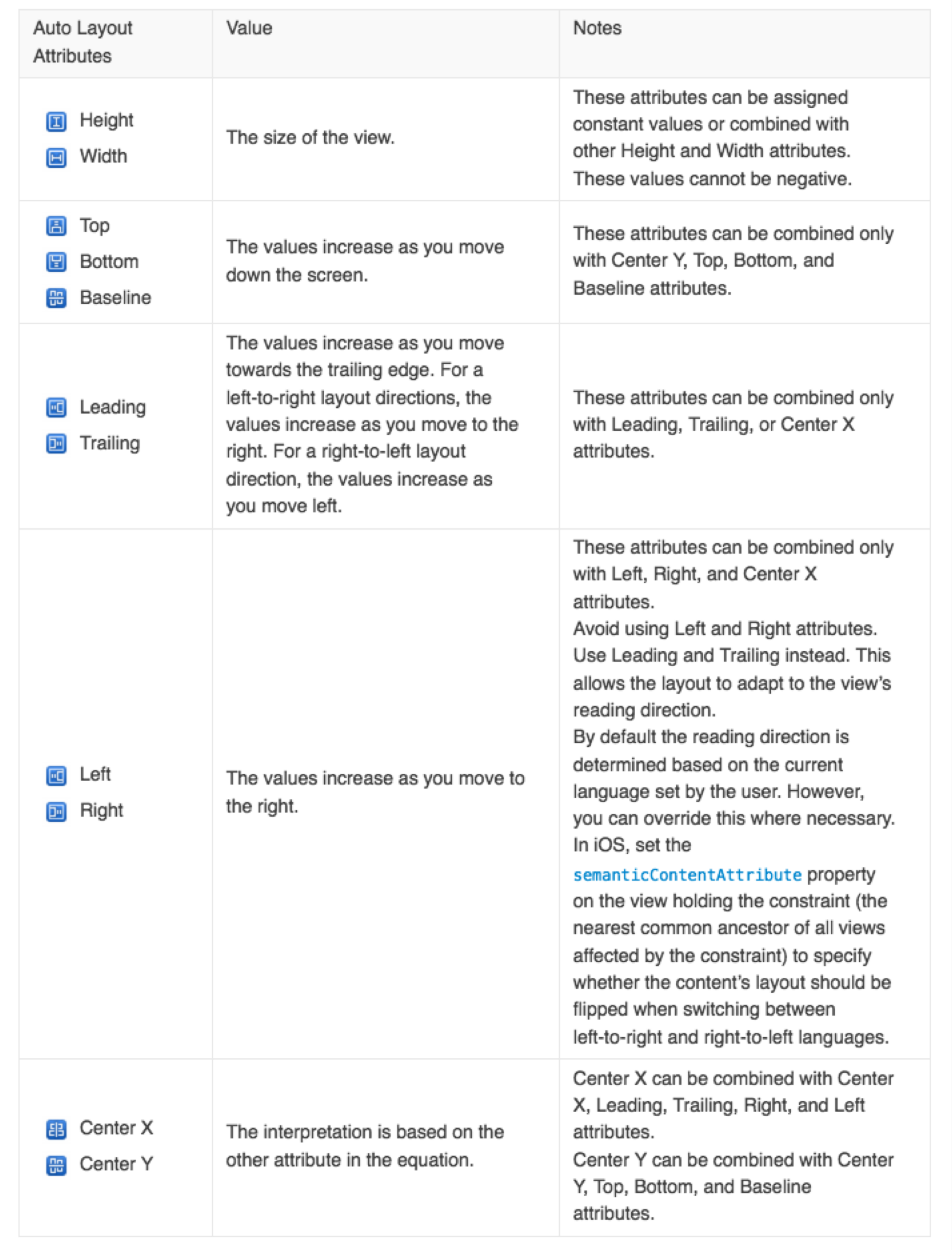
Control

Alignment

Autolayout

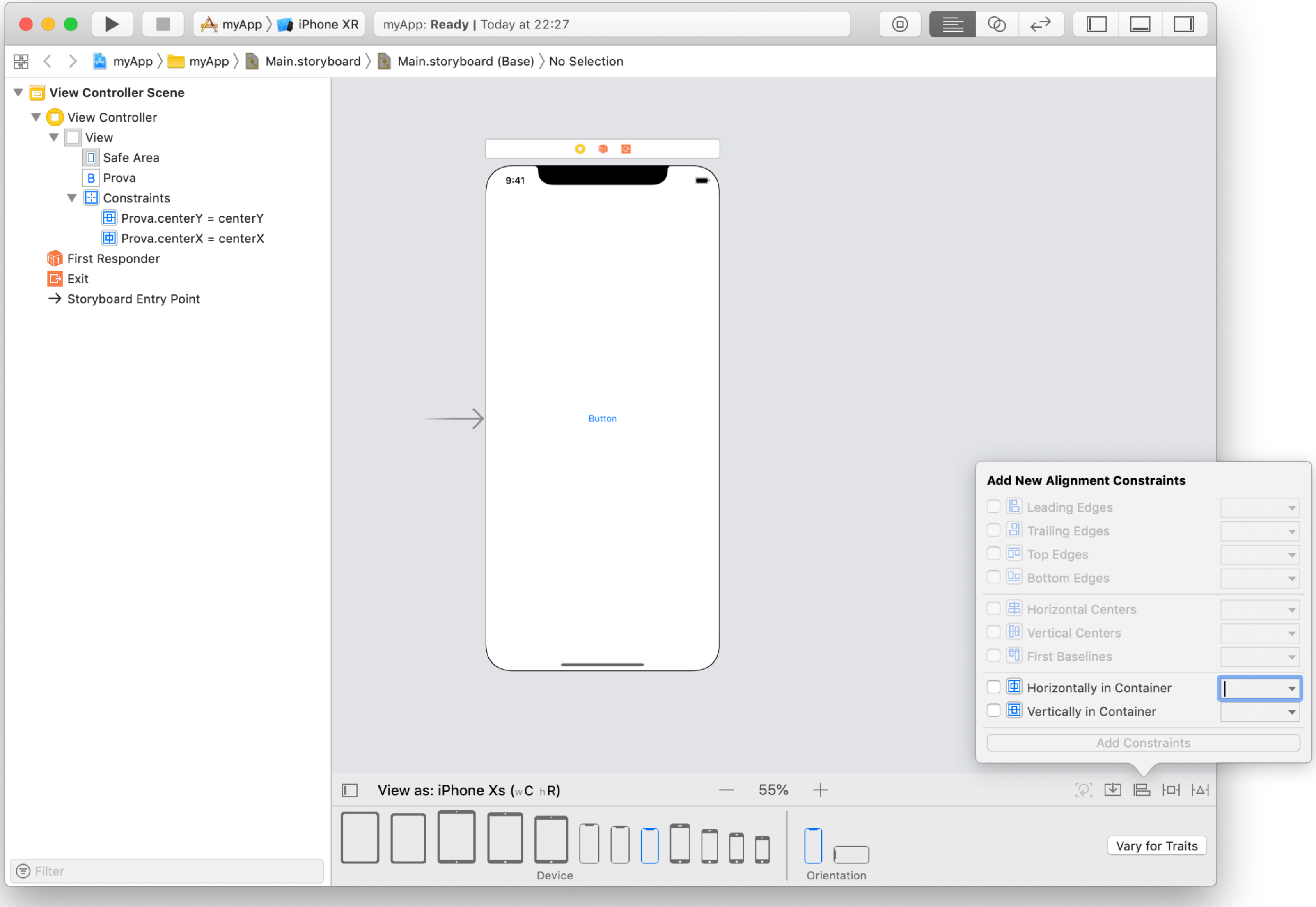
- Determines where objects should go and how big they should be based on constraints we set on them
  - This allows interfaces to adapt to being rotated between portrait and landscape, and to handle differing screen sizes
- Constraints allow us to express what matters to us and to let other factors vary as needed
  - We can specify the size of components, their alignment with or distance from other components, etc.



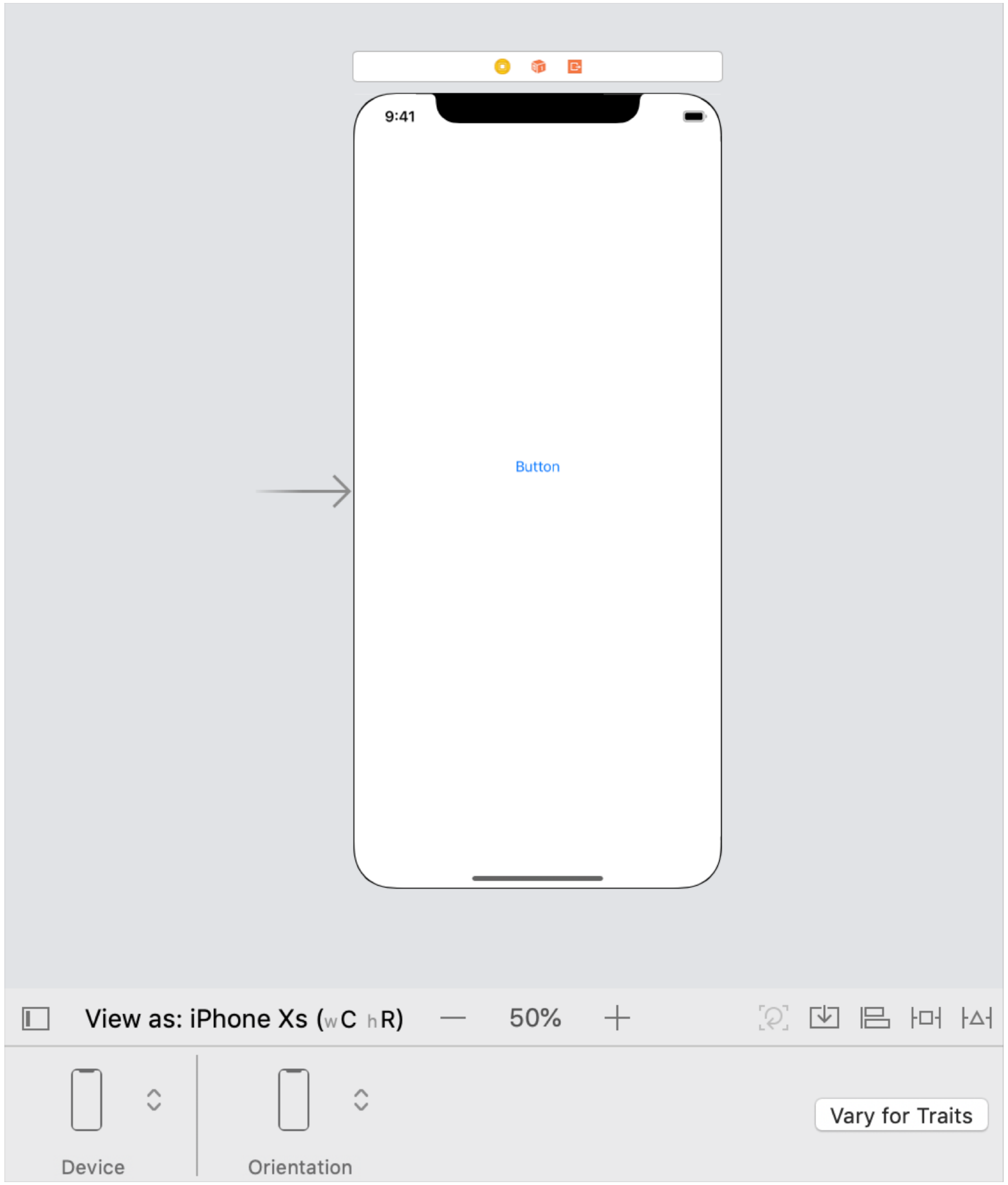


Auto Layout Attributes	Value	Notes
Height Width	The size of the view.	These attributes can be assigned constant values or combined with other Height and Width attributes. These values cannot be negative.
Top Bottom Baseline	The values increase as you move down the screen.	These attributes can be combined only with Center Y, Top, Bottom, and Baseline attributes.
Leading Trailing	The values increase as you move towards the trailing edge. For a left-to-right layout directions, the values increase as you move to the right. For a right-to-left layout direction, the values increase as you move left.	These attributes can be combined only with Leading, Trailing, or Center X attributes.
Left Right	The values increase as you move to the right.	These attributes can be combined only with Left, Right, and Center X attributes. Avoid using Left and Right attributes. Use Leading and Trailing instead. This allows the layout to adapt to the view's reading direction. By default the reading direction is determined based on the current language set by the user. However, you can override this where necessary. In iOS, set the `semanticContentAttribute` property on the view holding the constraint (the nearest common ancestor of all views affected by the constraint) to specify whether the content's layout should be flipped when switching between left-to-right and right-to-left languages.
Center X Center Y	The interpretation is based on the other attribute in the equation.	Center X can be combined with Center X, Leading, Trailing, Right, and Left attributes. Center Y can be combined with Center Y, Top, Bottom, and Baseline attributes.

Example


myApp 〉 iPhone XR
myApp: Ready | Today at 22:27
myApp 〉 myApp 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection
View Controller Scene
View Controller
View
Safe Area
Prova
Constraints
Prova.centerY = centerY
Prova.centerX = centerX
First Responder
Exit
Storyboard Entry Point
Filter
9:41
Button
Add New Alignment Constraints
□ Leading Edges
□ Trailing Edges
□ Top Edges
□ Bottom Edges
□ Horizontal Centers
□ Vertical Centers
□ First Baselines
□ Horizontally in Container
□ Vertically in Container
Add Constraints
View as: iPhone Xs (wC hR)
55%
Device
Orientation
Vary for Traits

Different devices



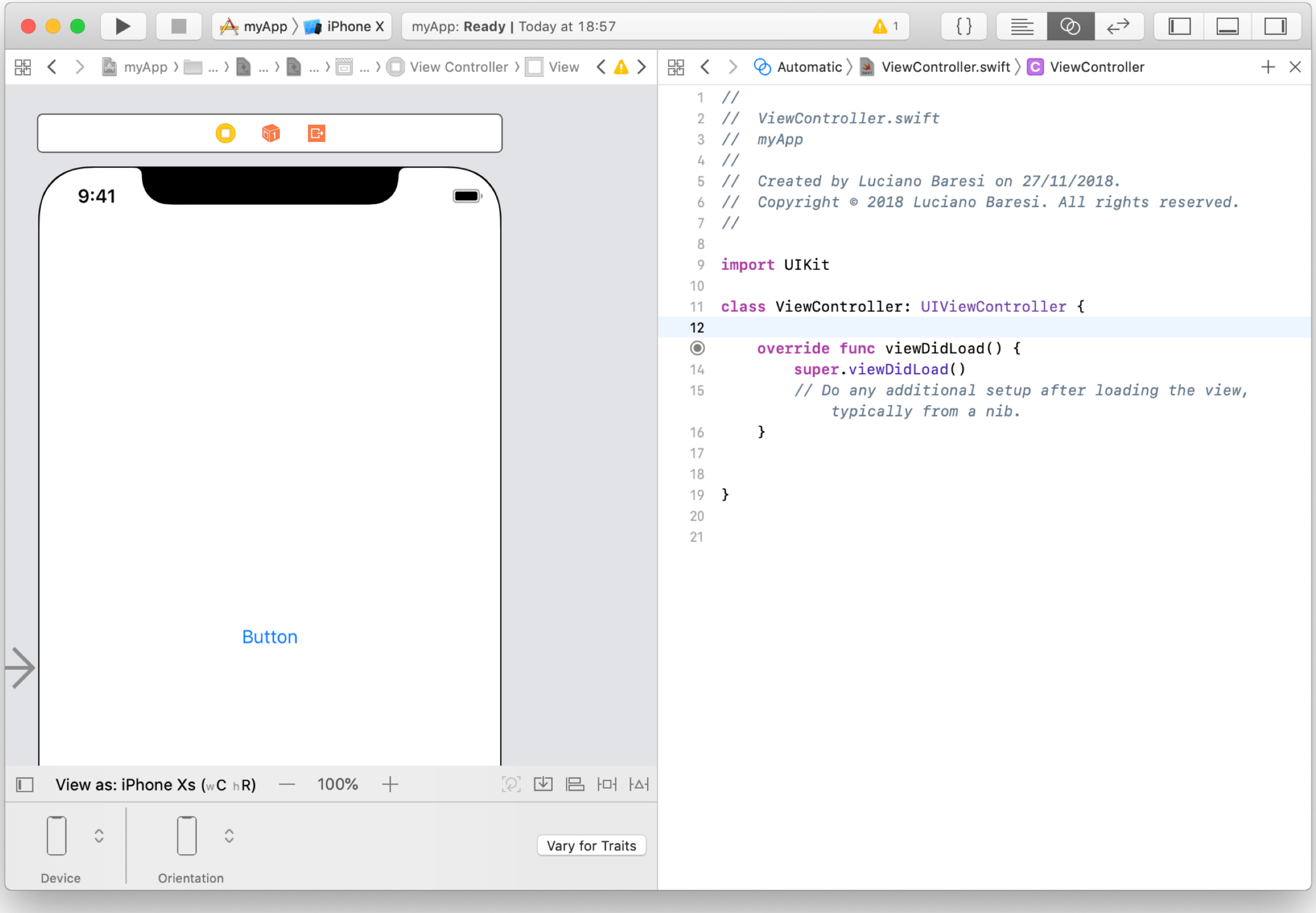
App content

- Xcode provides a library of objects
  - Some of these are user interface elements that belong to a view, such as buttons and text fields
  - Others define the behavior of our app, such as view controllers and gesture recognizers
- A view controller manages a corresponding view and its sub-views

Interface Builder and code

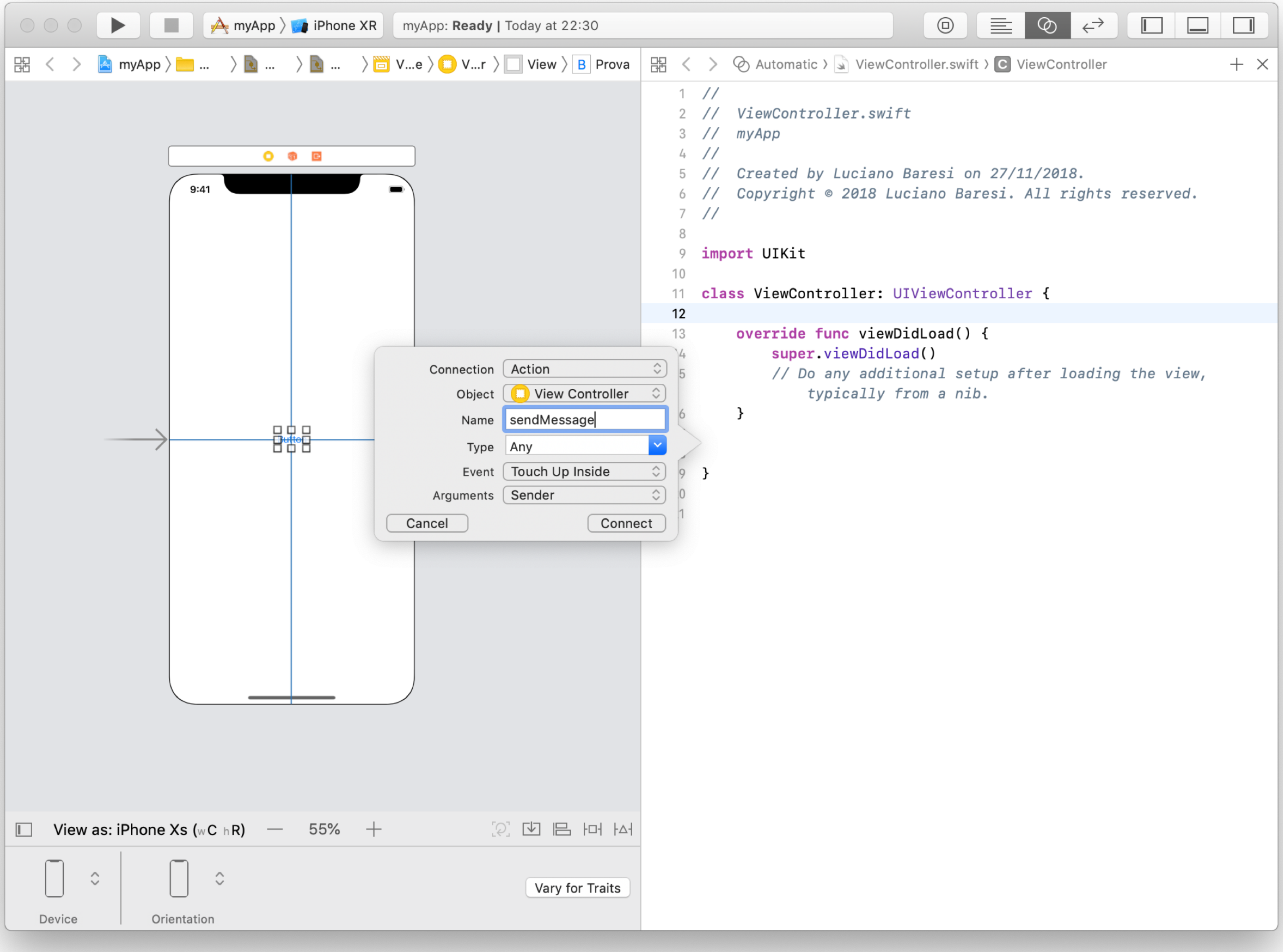
- An IBOutlet connects a variable or property in code to an object in a storyboard
  - This lets us read and write objects' properties, like reading the value of a slider or setting the initial contents of a text field
- An IBAction connects an event generated by a storyboard object to a method in the code
  - This lets us respond to a button being tapped or a slider's value changing

Generated code



```swift
//
//  ViewController.swift
//  myApp
//
//  Created by Luciano Baresi on 27/11/2018.
//  Copyright © 2018 Luciano Baresi. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
            typically from a nib.
    }


}

```

Control-click, drag



myApp 〉 iPhone XR
myApp: Ready | Today at 22:30
myApp 〉 ... 〉 ... 〉 ... 〉 V...e 〉 V...r 〉 View 〉 Prova
Automatic 〉 ViewController.swift 〉 ViewController
9:41
View as: iPhone Xs (wC hR)
55%
Device
Orientation
Vary for Traits
Cancel
Connect

Connection: Action

Object: View Controller

Name: sendMessage

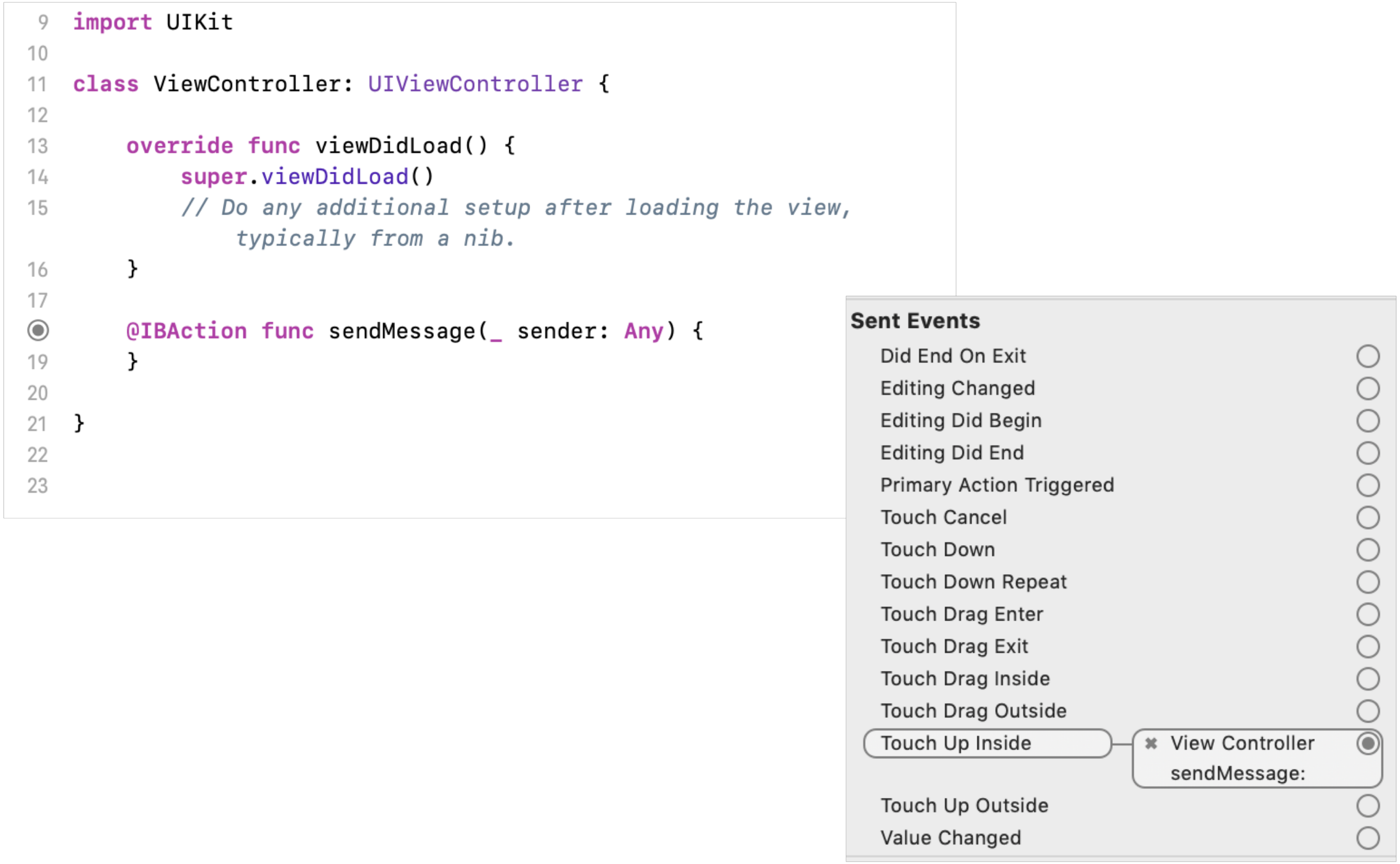
Type: Any

Event: Touch Up Inside

Arguments: Sender

```
//
//  ViewController.swift
//  myApp
//
//  Created by Luciano Baresi on 27/11/2018.
//  Copyright © 2018 Luciano Baresi. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
            typically from a nib.
    }

}
```

. . . and the result is



```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
            typically from a nib.
    }

    @IBAction func sendMessage(_ sender: Any) {
    }

}
```

Sent Events

- Did End On Exit ○
- Editing Changed ○
- Editing Did Begin ○
- Editing Did End ○
- Primary Action Triggered ○
- Touch Cancel ○
- Touch Down ○
- Touch Down Repeat ○
- Touch Drag Enter ○
- Touch Drag Exit ○
- Touch Drag Inside ○
- Touch Drag Outside ○
- Touch Up Inside — ✖ View Controller sendMessage: ●
- Touch Up Outside ○
- Value Changed ○

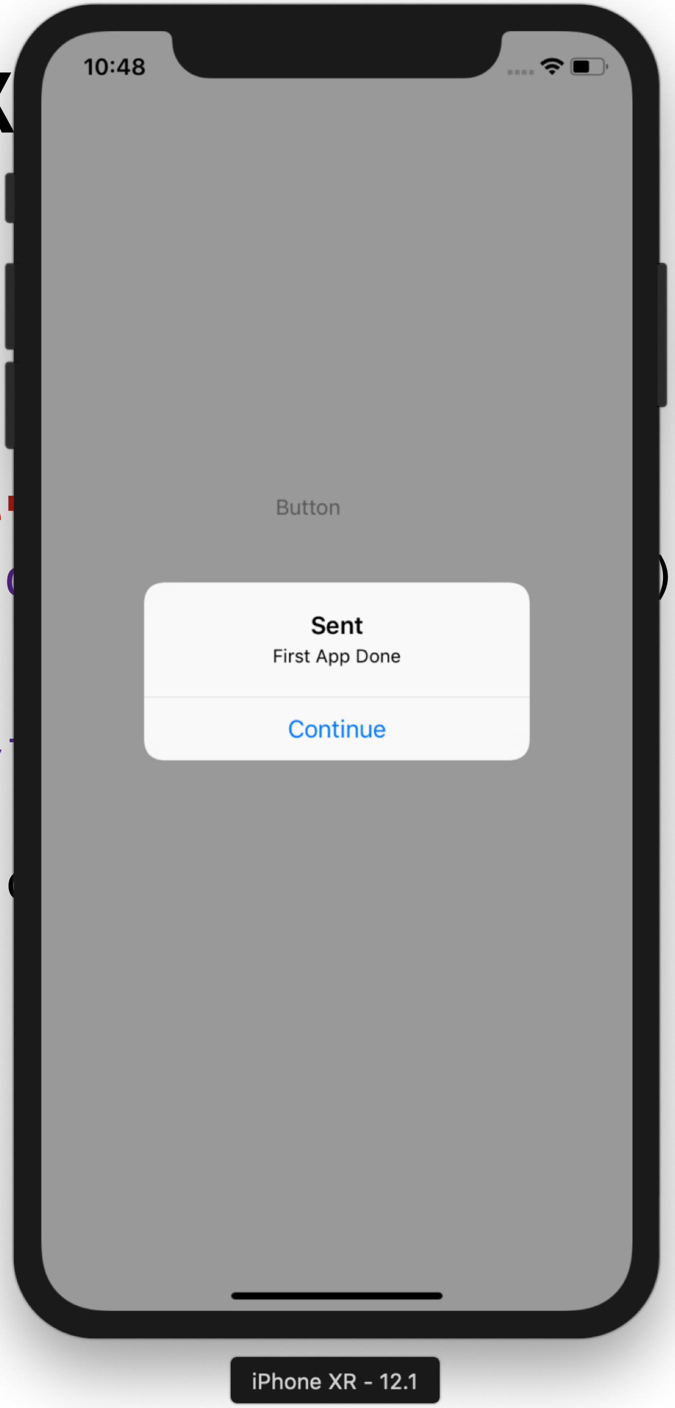
Logging

- On iOS, we can use function print() to write a string out to the system’s log file
  - For example, we can implement our action to just log a message every time the button is tapped

```
@IBAction func sendMessage(_ sender: UIButton) {
   print ("Button pressed")
}
```

A first complete ex

```
@IBAction func press(_ sender: Any) {
   let alert = UIAlertController(title:
               message: "First App Done"
               preferredStyle: UIAlertC

   alert.addAction(UIAlertAction(title:
               style: UIAlertAction.Sty
               handler: nil))
   self.present(alert, animated: true, 
}
```



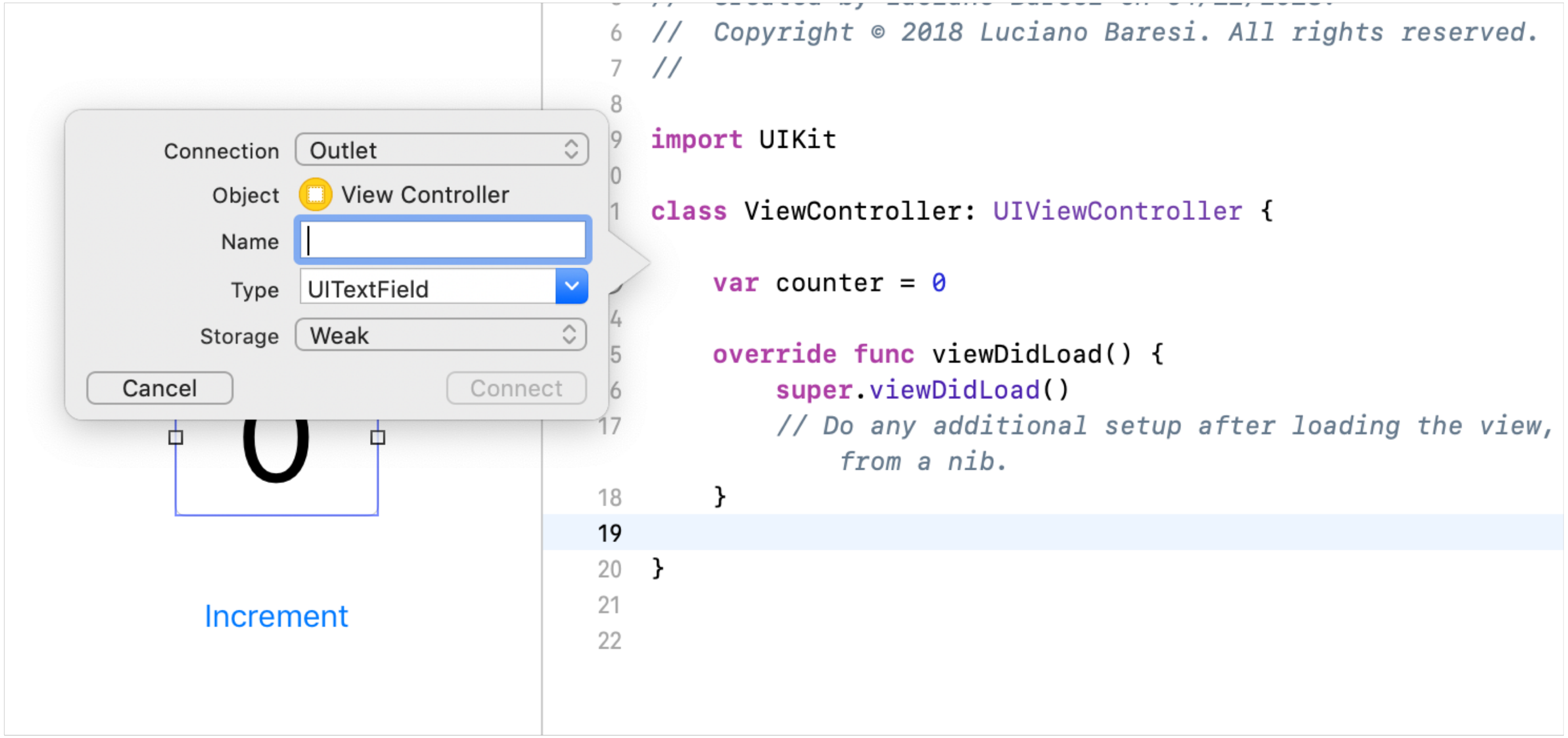
Sent

First App Done

Outlets

- Preceding a property with the @IBOutlet modifier tells Interface Builder that a property can serve as an outlet

- A stored property implies a variable to store the value
- A computed property does not imply a backing variable



Connection: Outlet

Object: View Controller

Name:

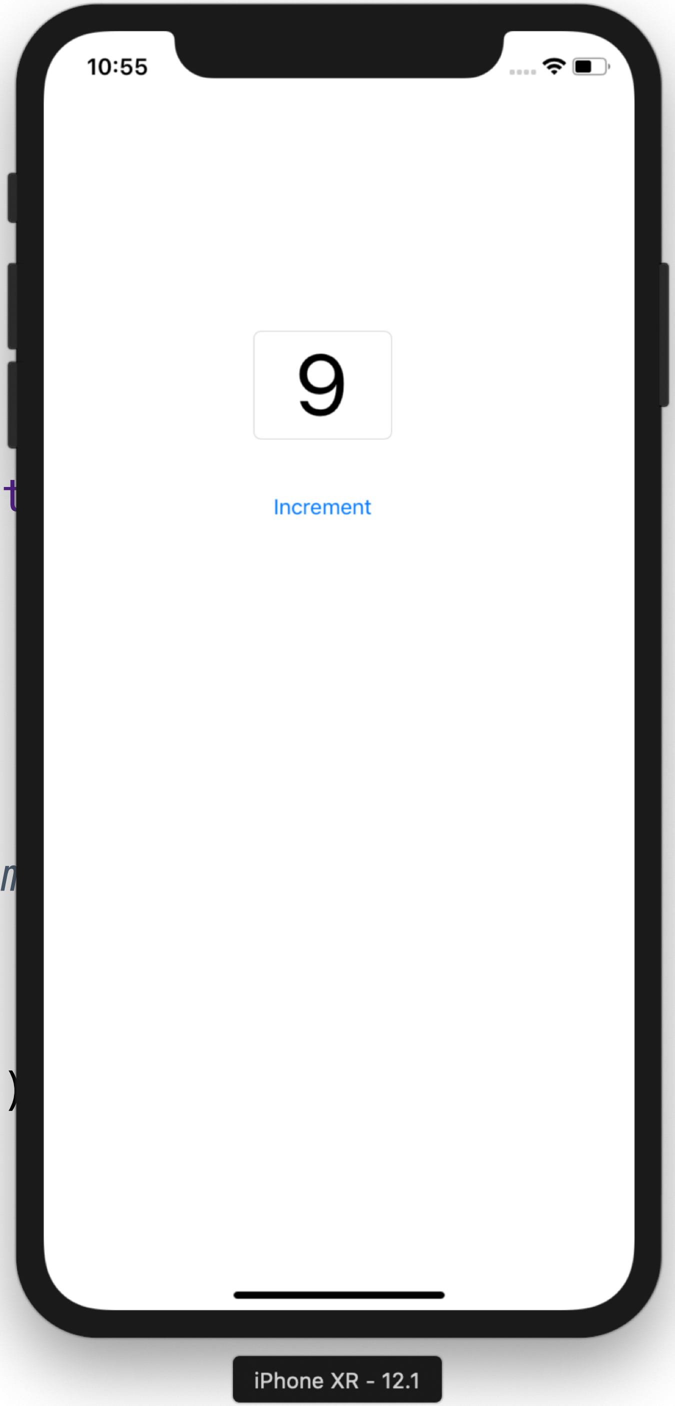
Type: UITextField

Storage: Weak

```
//  Copyright © 2018 Luciano Baresi. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    var counter = 0

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, 
            from a nib.
    }

}
```

Final result

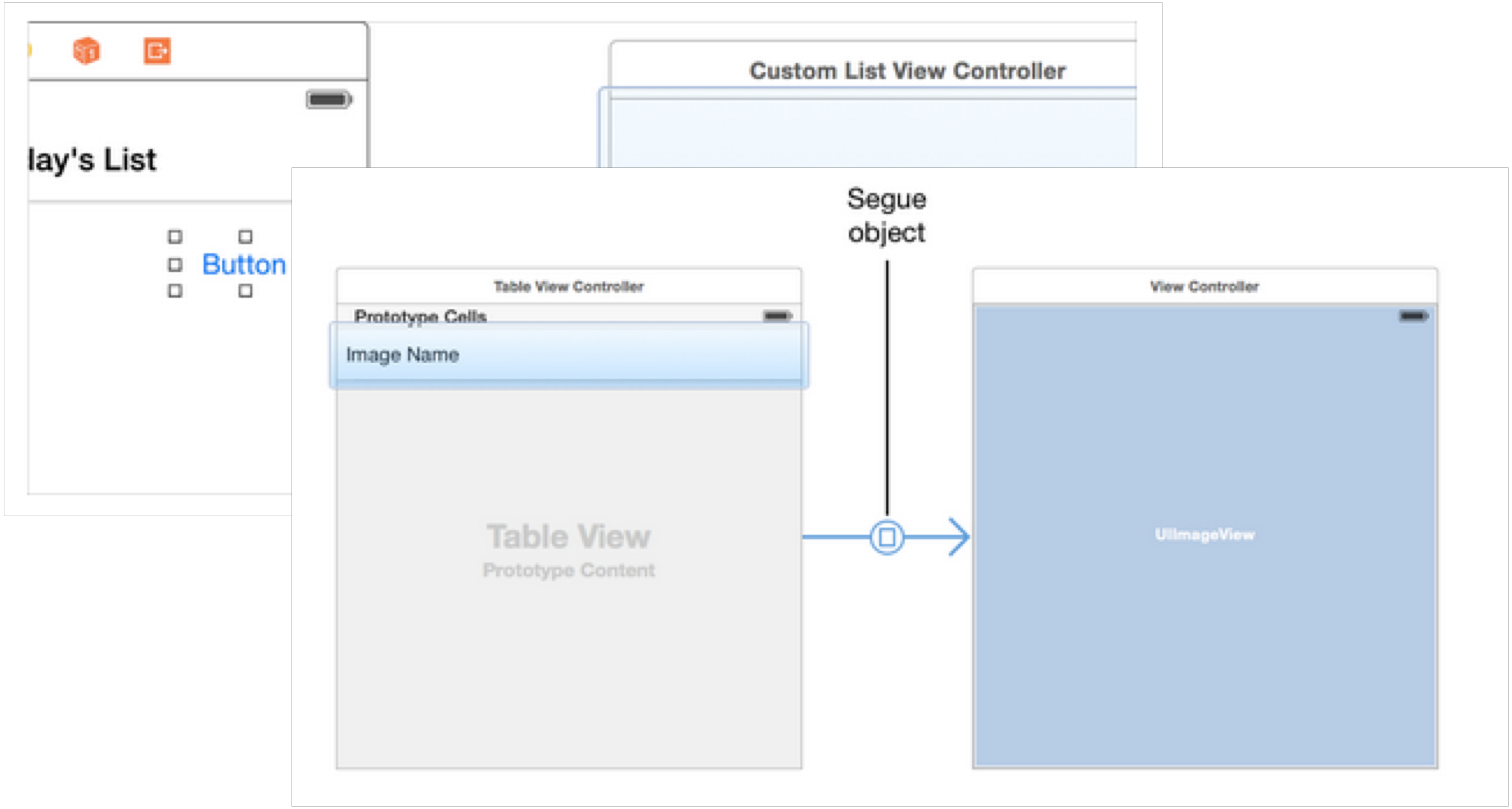
```
import UIKit


class ViewController: UIViewController {

    @IBOutlet weak var counterlabel: UIText

    var counter = 0

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after
        // loading the view, typically from
    }


    @IBAction func increment(_ sender: Any)
        counter += 1
        counterlabel.text = "\(counter)"
    }
}
```



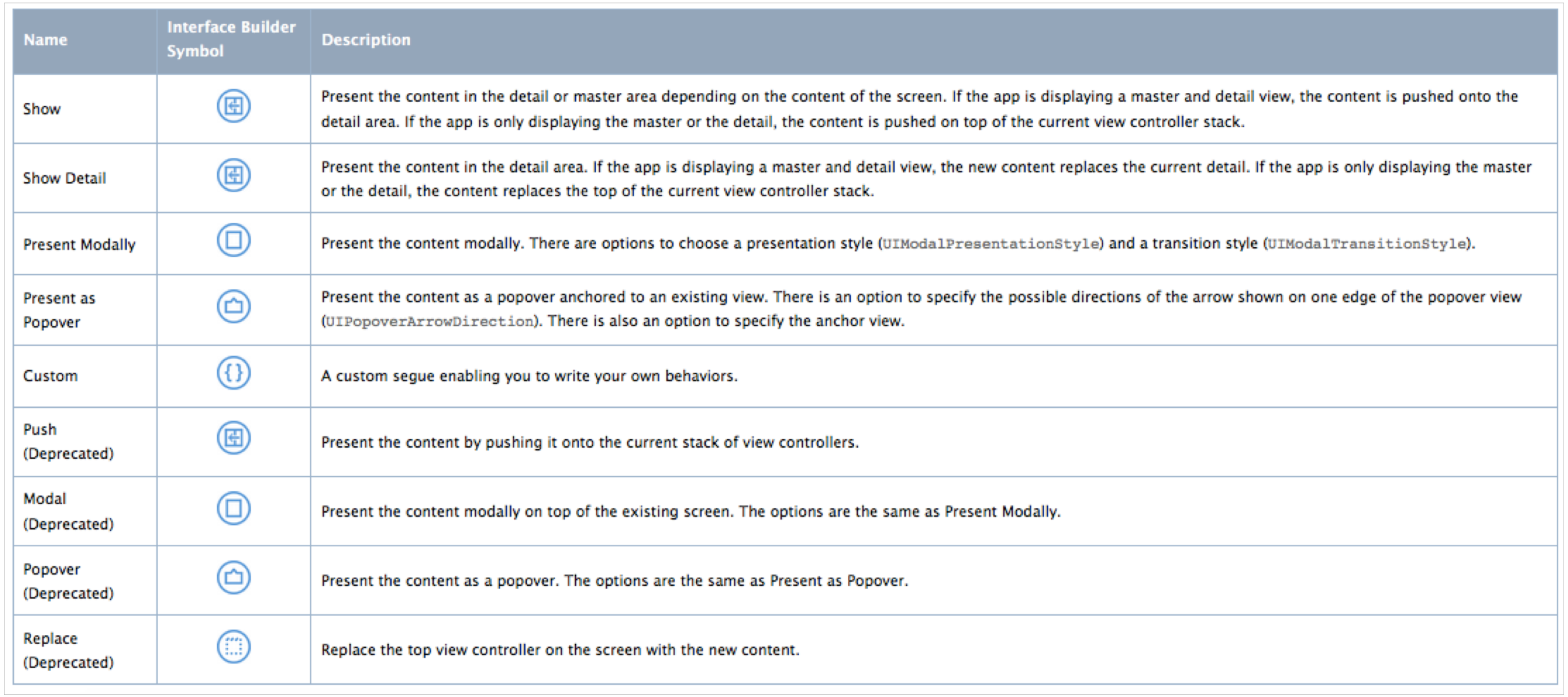
Weak attribute

- Automatic Reference Counting (ARC) solves almost all memory problems, but it cannot solve retain cycles
- Our ViewController knows about the UILabel, so ARC cannot free the label from memory as long as the view controller exists
- But if the UILabel also requires the ViewController, then neither can ever be freed from memory (cycle)
- The way to break this is to declare one side of the arrangement as weak
- The rule of thumb is that
  - Only "top-level" objects in a storyboard scene (like the view) need strong references, and everything else can be weak
  - Xcode defaults to this behavior when we made the connection

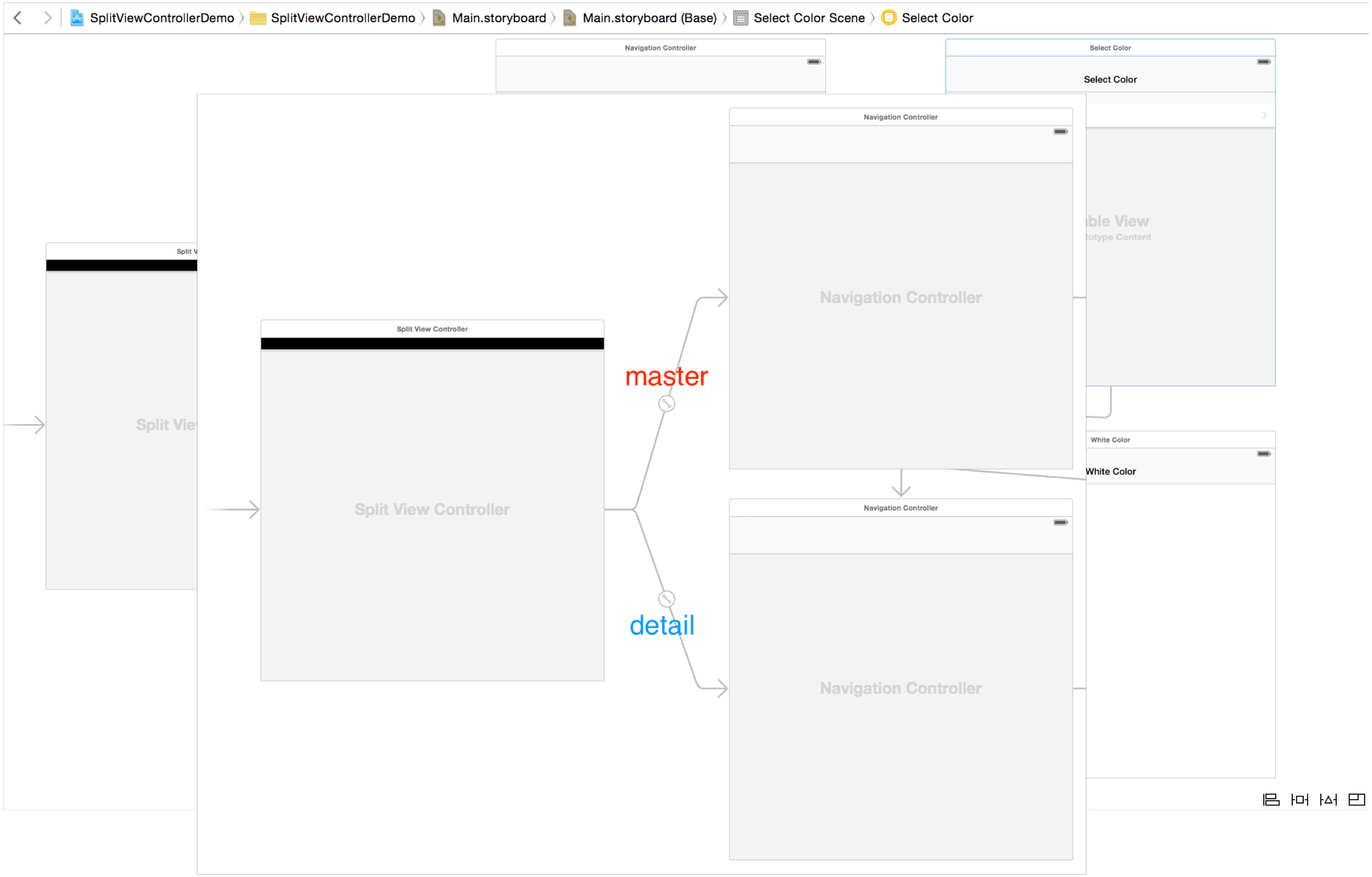
More scenes

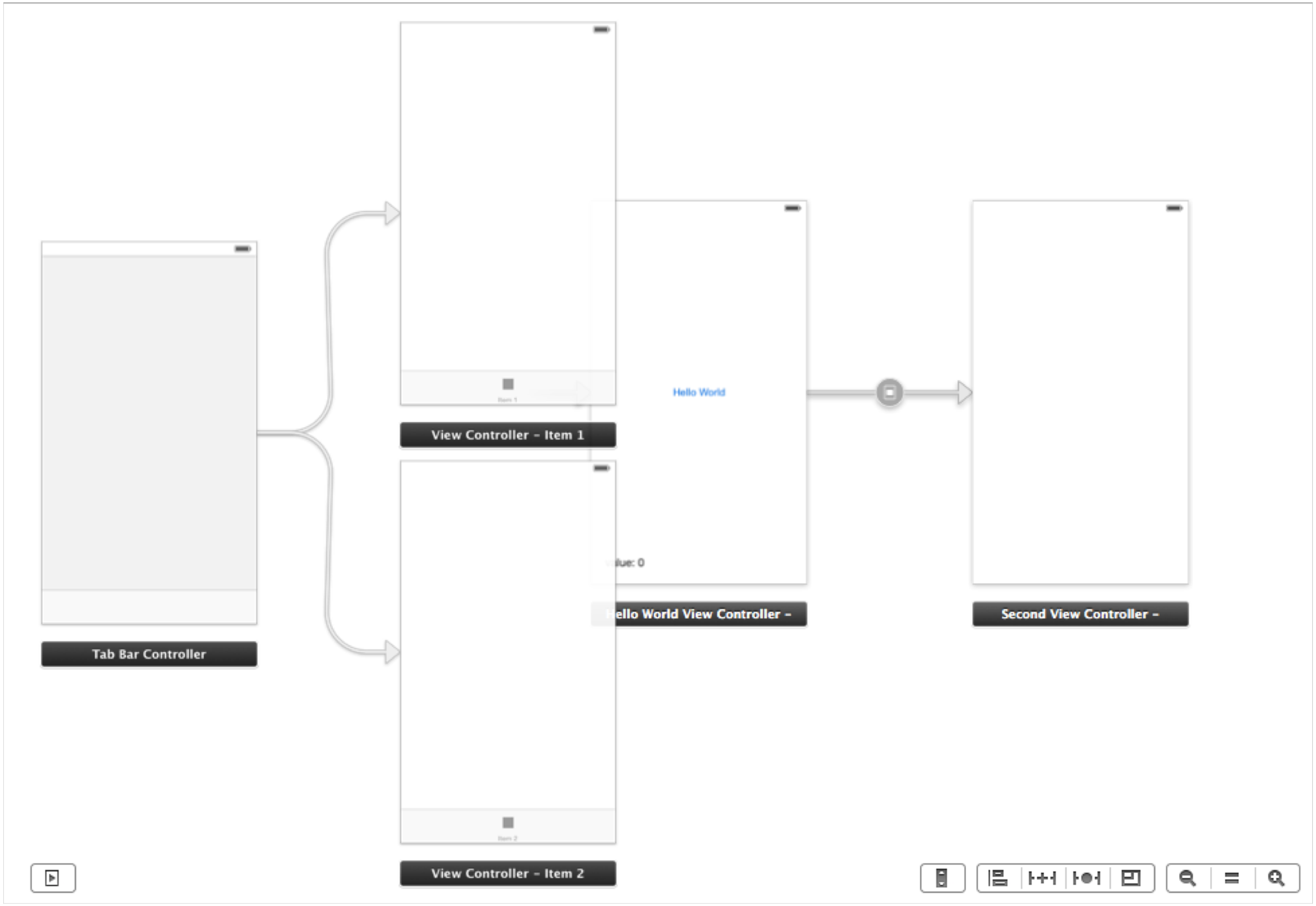

lay's List
Button
Custom List View Controller
Segue object
Table View Controller
Prototype Cells
Image Name
Table View
Prototype Content
View Controller
UIImageView

Segue types



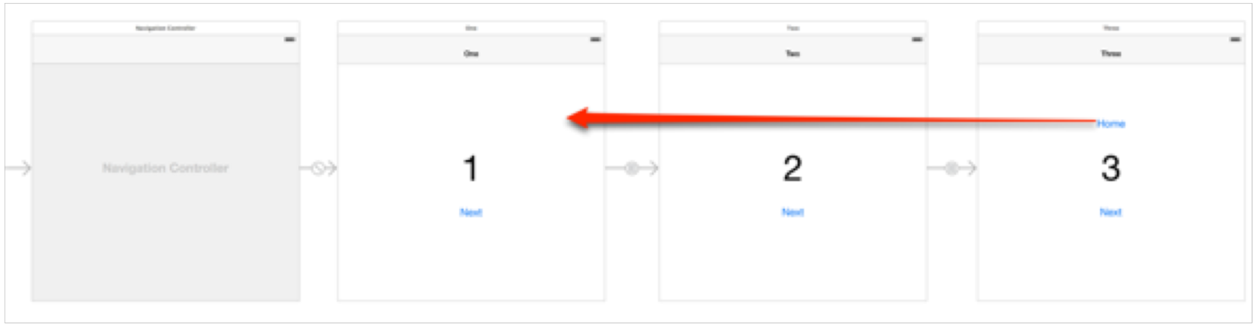
Name	Interface Builder Symbol	Description
Show	⊞	Present the content in the detail or master area depending on the content of the screen. If the app is displaying a master and detail view, the content is pushed onto the detail area. If the app is only displaying the master or the detail, the content is pushed on top of the current view controller stack.
Show Detail	⊞	Present the content in the detail area. If the app is displaying a master and detail view, the new content replaces the current detail. If the app is only displaying the master or the detail, the content replaces the top of the current view controller stack.
Present Modally	□	Present the content modally. There are options to choose a presentation style (`UIModalPresentationStyle`) and a transition style (`UIModalTransitionStyle`).
Present as Popover	⌂	Present the content as a popover anchored to an existing view. There is an option to specify the possible directions of the arrow shown on one edge of the popover view (`UIPopoverArrowDirection`). There is also an option to specify the anchor view.
Custom	{}	A custom segue enabling you to write your own behaviors.
Push (Deprecated)	⊞	Present the content by pushing it onto the current stack of view controllers.
Modal (Deprecated)	□	Present the content modally on top of the existing screen. The options are the same as Present Modally.
Popover (Deprecated)	⌂	Present the content as a popover. The options are the same as Present as Popover.
Replace (Deprecated)	⬚	Replace the top view controller on the screen with the new content.


SplitViewControllerDemo
SplitViewControllerDemo
Main.storyboard
Main.storyboard (Base)
Select Color Scene
Select Color
Navigation Controller
Select Color
Select Color
ble View
totype Content
Split V
Split Vie
Navigation Controller
Navigation Controller
Split View Controller
Split View Controller
master
White Color
White Color
Navigation Controller
Navigation Controller
detail


Tab Bar Controller
Item 1
View Controller – Item 1
Item 2
View Controller – Item 2
Hello World
lue: 0
ello World View Controller –
Second View Controller –

Unwind Segue

- Can be used to “unwind” the navigation stack and specify a destination to go back to
- Unwind segues always segue from the source or current view controller to an existing view controller, a view controller that is already present in the navigation hierarchy

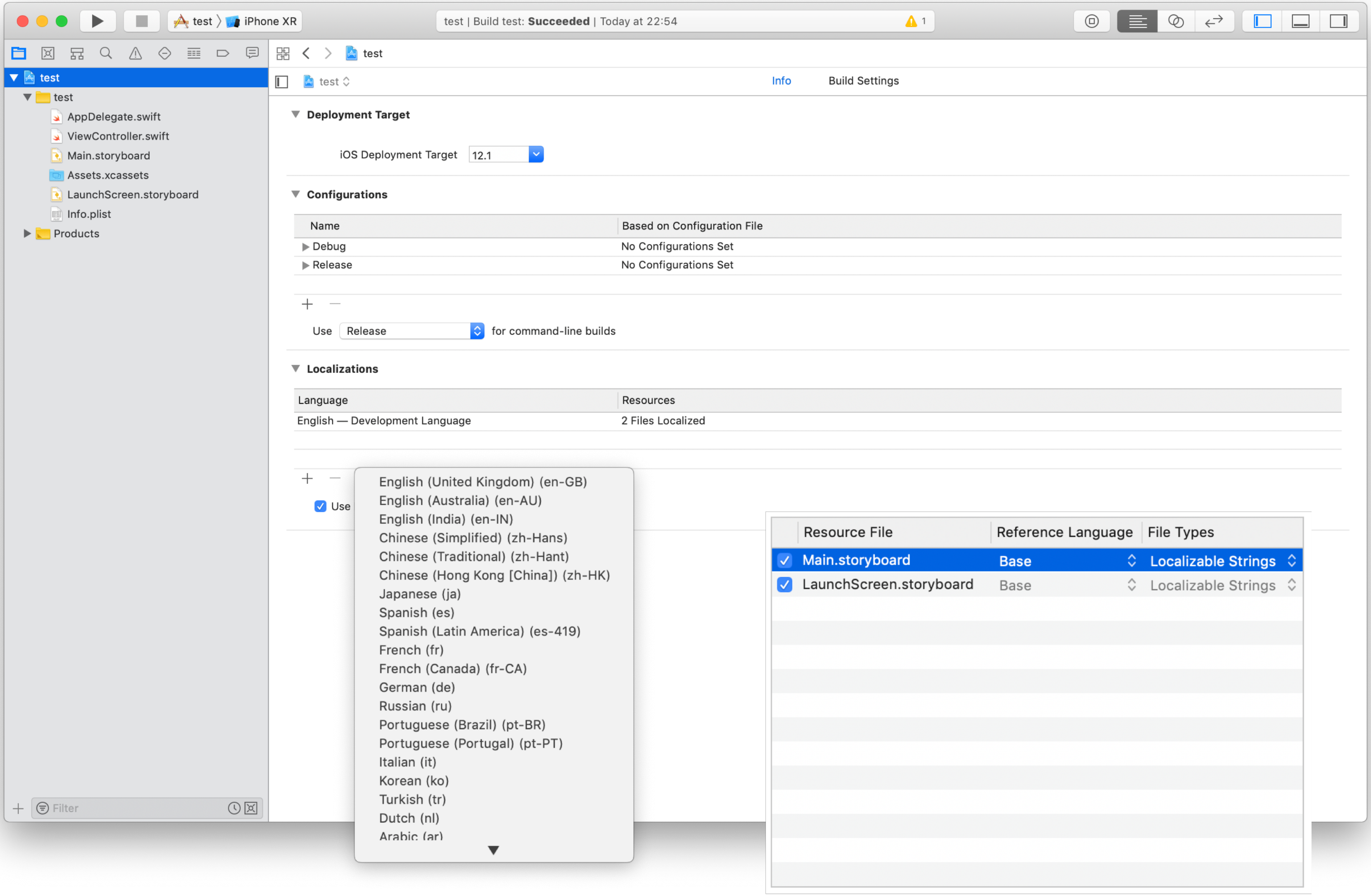


UIGestureRecognizer

- We can get notified of the raw touch events or we can react to certain, predefined “gestures”
- Gestures are recognized by class UIGestureRecognizer (abstract)
  - TapGestureRecognizer, UIPinchGestureRecognizer, UIRotationGestureRecognizer, UISwipeGestureRecognizer, UIPanGestureRecognizer, UIScreenEdgePanGestureRecognizer, UILongPressGestureRecognizer
- There are two sides to using a gesture recognizer
  - Adding a gesture recognizer to a UIView to ask it to recognize that gesture
  - Providing the implementation of a method to “handle” that gesture when it happens

Internationalization

- The ability of code to adapt to local conventions in different parts of the world
  - This includes things like language, time and date formatting, and currency symbols and separators
- We must create a localization for each locale we want to support
  - A localization is a collection of strings, currency formats, graphics, sounds, and other resources that are specific to one locale
  - We declare supported localizations at the project level



test | Build test: **Succeeded** | Today at 22:54

Deployment Target

iOS Deployment Target 12.1

Configurations

Name	Based on Configuration File
▶ Debug	No Configurations Set
▶ Release	No Configurations Set

Use Release for command-line builds

Localizations

Language	Resources
English — Development Language	2 Files Localized

☑ Use

	Resource File	Reference Language	File Types
☑	Main.storyboard	Base	Localizable Strings
☑	LaunchScreen.storyboard	Base	Localizable Strings

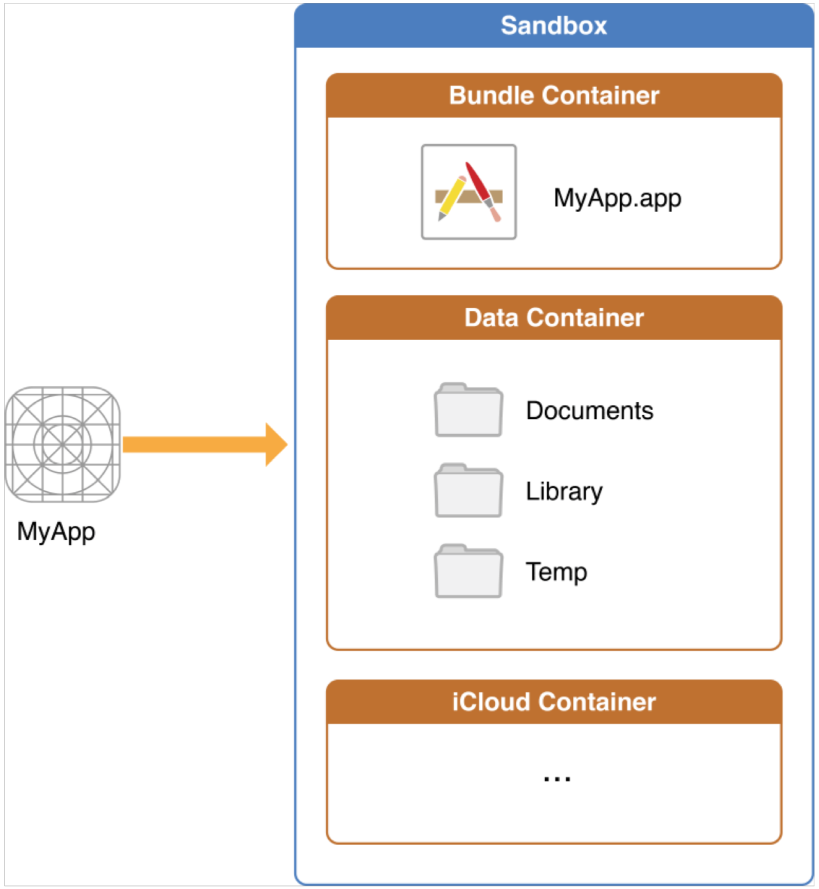
Data Management

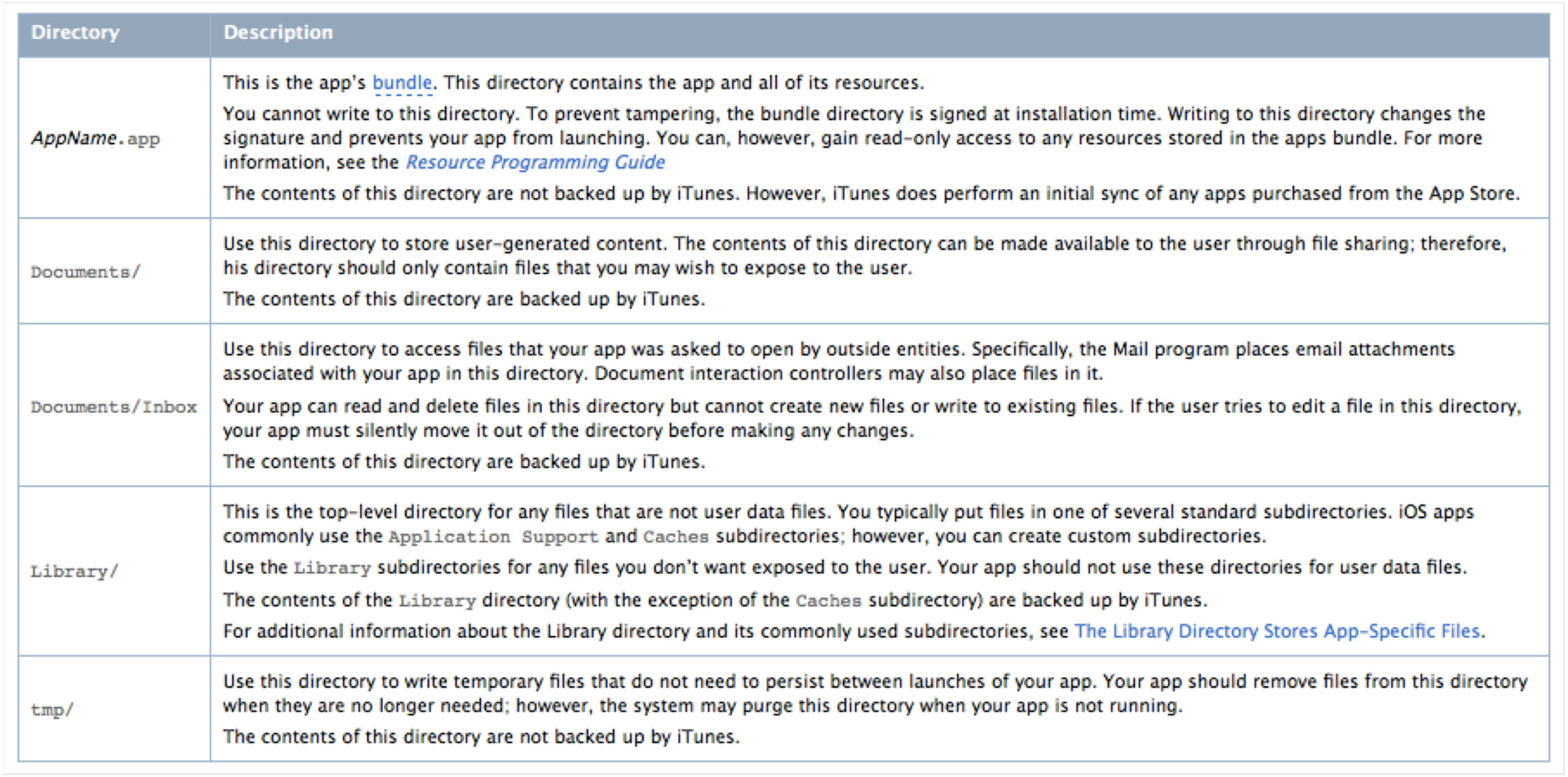
Three options

- File System
  - Based on the UNIX file system
- SQLite
  - Embedded DBMS (like in Android)
- Core Data
  - Object-oriented database
  - Powerful framework in iOS

File system

- Interactions with the file system are limited to the directories inside the app's sandbox
  - Exception: when an app uses public system interfaces to access things such as the user's contacts or music
- During installation of a new app, the installer creates a number of containers for the app
  - Each container has a specific role





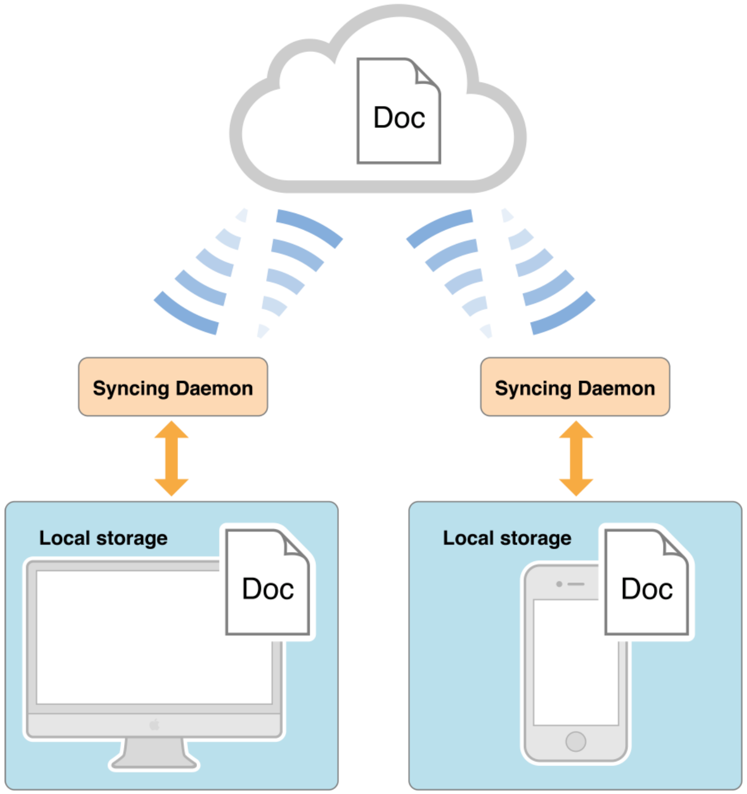
Directory	Description
AppName`.app`	This is the app's bundle. This directory contains the app and all of its resources. You cannot write to this directory. To prevent tampering, the bundle directory is signed at installation time. Writing to this directory changes the signature and prevents your app from launching. You can, however, gain read-only access to any resources stored in the apps bundle. For more information, see the *Resource Programming Guide* The contents of this directory are not backed up by iTunes. However, iTunes does perform an initial sync of any apps purchased from the App Store.
`Documents/`	Use this directory to store user-generated content. The contents of this directory can be made available to the user through file sharing; therefore, his directory should only contain files that you may wish to expose to the user. The contents of this directory are backed up by iTunes.
`Documents/Inbox`	Use this directory to access files that your app was asked to open by outside entities. Specifically, the Mail program places email attachments associated with your app in this directory. Document interaction controllers may also place files in it. Your app can read and delete files in this directory but cannot create new files or write to existing files. If the user tries to edit a file in this directory, your app must silently move it out of the directory before making any changes. The contents of this directory are backed up by iTunes.
`Library/`	This is the top-level directory for any files that are not user data files. You typically put files in one of several standard subdirectories. iOS apps commonly use the `Application Support` and `Caches` subdirectories; however, you can create custom subdirectories. Use the `Library` subdirectories for any files you don't want exposed to the user. Your app should not use these directories for user data files. The contents of the `Library` directory (with the exception of the `Caches` subdirectory) are backed up by iTunes. For additional information about the Library directory and its commonly used subdirectories, see The Library Directory Stores App-Specific Files.
`tmp/`	Use this directory to write temporary files that do not need to persist between launches of your app. Your app should remove files from this directory when they are no longer needed; however, the system may purge this directory when your app is not running. The contents of this directory are not backed up by iTunes.

Files and directories

- Directories
  - You must use the methods of FileManager
  - A process can create directories anywhere it has permission to do so
- Files
  - When specifying the location of files, you can use either NSURL or NSString objects
    - The use of the URL class is generally preferred
  - Two parts: creation of a record for the file in the file system and filling the file with content
- To copy items around the file system, you use class FileManager
  - The file manager asks its delegate whether the operation should begin at all and whether it should proceed when an error occurs

iCloud Storage API

- Manage files and key-value data that are automatically synchronized among a user's iCloud devices



How it works

- A document is not moved to iCloud immediately
  - First, it is moved from its current location in the file system to a local system-managed directory where it can be monitored by the iCloud service
  - After that transfer, the file is transferred to iCloud and to the user's other devices as soon as possible
- Apps are expected to use file coordinator objects to perform all changes
  - File coordinators mediate changes between your app and the daemon that facilitates the transfer of the document to and from iCloud
  - The file coordinator acts like a locking mechanism for the document
- Class Document helps manage documents in iCloud

File Coordinators and File Presenters

- NSFileCoordinator coordinates the reads and writes performed by our app and the sync daemon on the same document
  - We use presenters in conjunction with a file coordinator to coordinate access to a file or directory among the objects of our application and between our application and other processes
  - Instances of NSFileCoordinator are meant to be used on a per-file-operation basis
- The FilePresenter protocol should be implemented by objects that allow the user to view or edit the content of files or directories
  - The job of a file presenter is to protect the integrity of its own data structures
  - Class Document is an example of a file presenter that tracks changes to its underlying file or file package

What apps should do to work with iCloud?

- Manage each document in iCloud using a file presenter
  - After creating a file presenter, register it
  - Before deleting a file presenter, unregister it
- All file-related operations must be performed through a file coordinator object
  - Create an instance of class NSFileCoordinator and initialize it with the file presenter object that is about to perform the file operation
  - Use the methods of the NSFileCoordinator object to read/write the file
  - When we are done with the operations, release the file coordinator object

SQLite

Different wrappers available (on GitHub)

SQLite.swift

- Swift interface to SQLLite
- Class Connection helps establish Database connections
  - We can create a writable database in our app's Documents directory
  - If we omit the path, SQLite.swift will provision an in-memory database
  - SQLite will attempt to create the database file if it does not already exist
  - We can also bundle a database with our app, and then we can establish a read-only connection to it

Core Data

Core Data

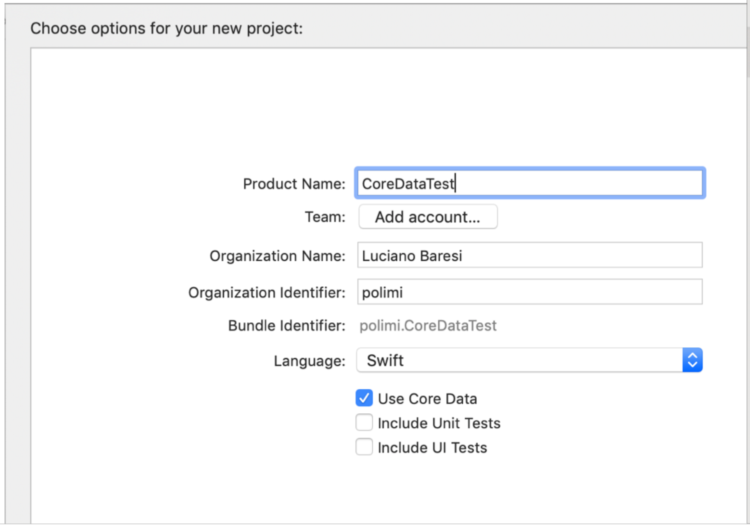
- Is a framework that we use to manage the objects in the model layer of our applications
- Provides generalized and automated solutions to common tasks associated with lifecycle and graph management of objects, including persistence
- Core Data can only do its magic because it keeps the object graph it manages in memory

Key characteristics

- Change tracking and built-in management of undo and redo beyond basic text editing
- Maintenance of change propagation, including maintaining the consistency of relationships among objects
- Lazy loading of objects, partially materialized futures (faulting), and copy-on-write data sharing to reduce overhead
- Automatic validation of property values. Managed objects extend the standard key-value coding validation methods to ensure that individual values lie within acceptable ranges, so that combinations of values make sense
- Schema migration tools that simplify schema changes and allow you to perform efficient in-place schema migration
- Optional integration with the application's controller layer to support user interface synchronization
- Grouping, filtering, and organizing of data in memory and in the user interface
- Automatic support for storing objects in external data repositories
- Sophisticated query compilation. Instead of writing SQL, you can create complex queries by associating an NSPredicate object with a fetch request
- Version tracking and optimistic locking to support automatic multiwriter conflict resolution

Let’s start

- AppDelegate.swift file
  - quite a significant amount of new code (Core Data stack)
  - Most of these methods are setting up the Core Data stack and the defaults are fine for now



Choose options for your new project:

Product Name: CoreDataTest

Team: Add account...

Organization Name: Luciano Baresi

Organization Identifier: polimi

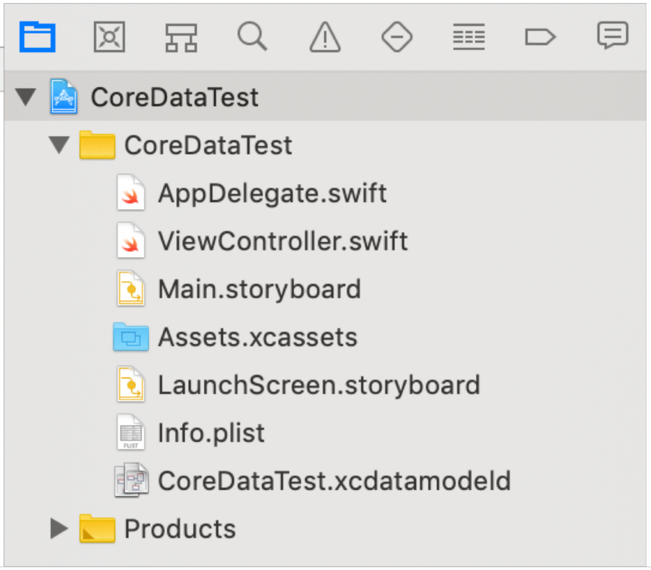
Bundle Identifier: polimi.CoreDataTest

Language: Swift

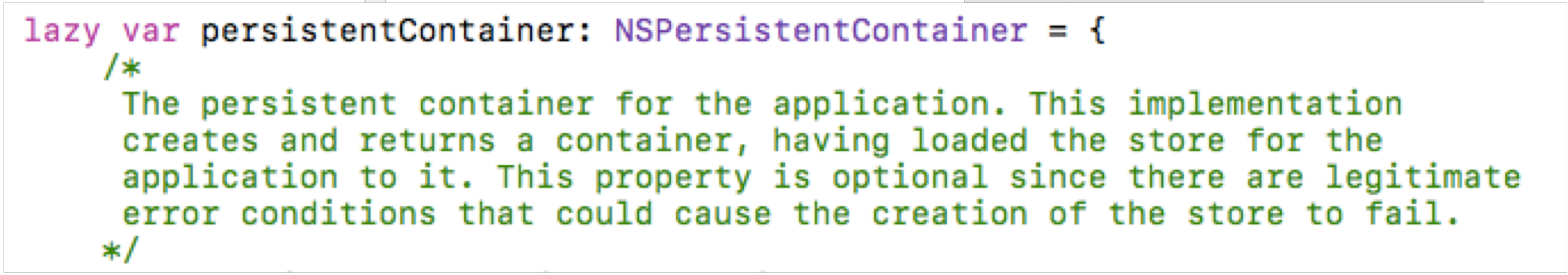
☑ Use Core Data

□ Include Unit Tests

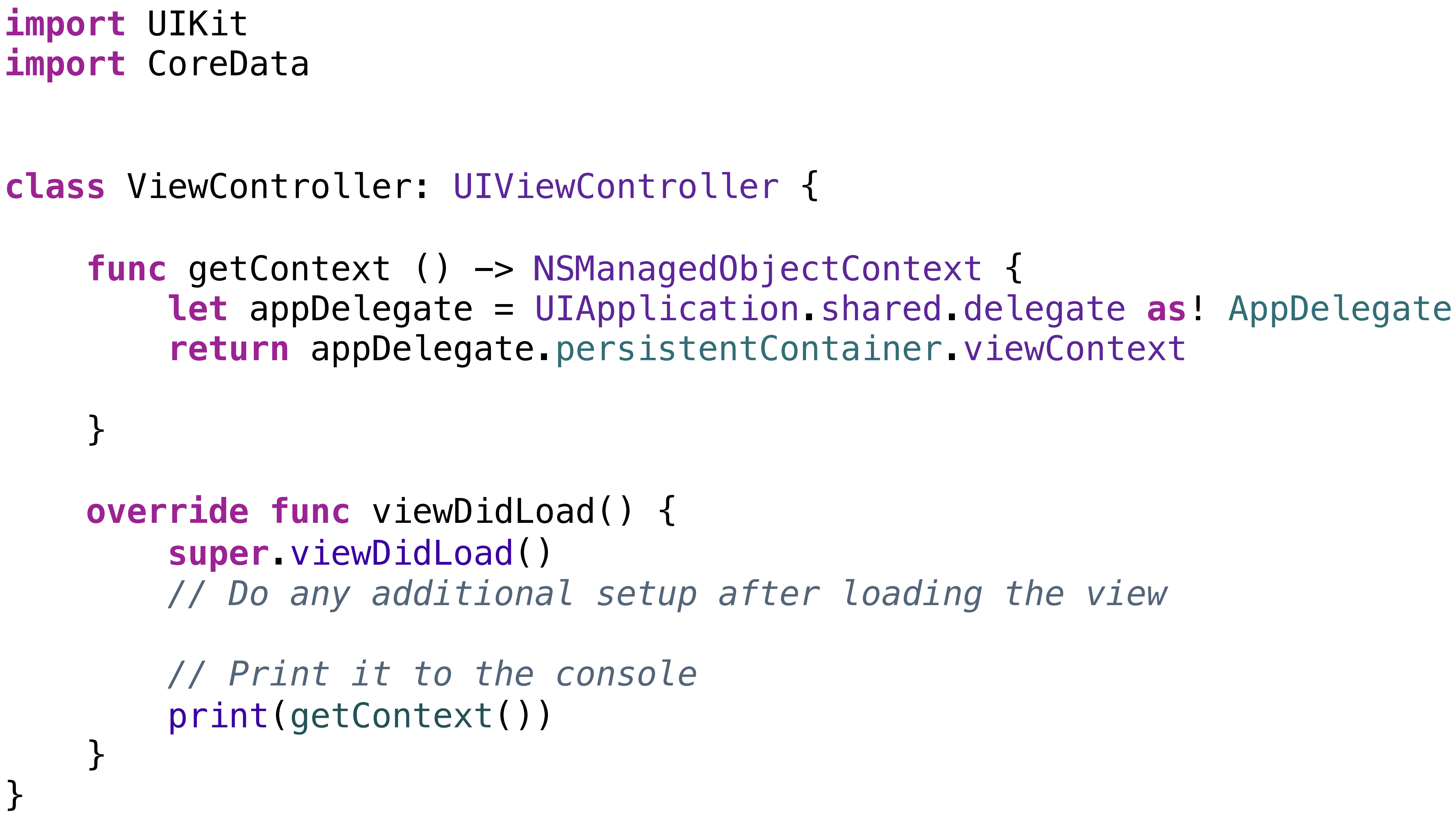
□ Include UI Tests



- CoreDataTest
  - CoreDataTest
    - AppDelegate.swift
    - ViewController.swift
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - CoreDataTest.xcdatamodeld
  - Products



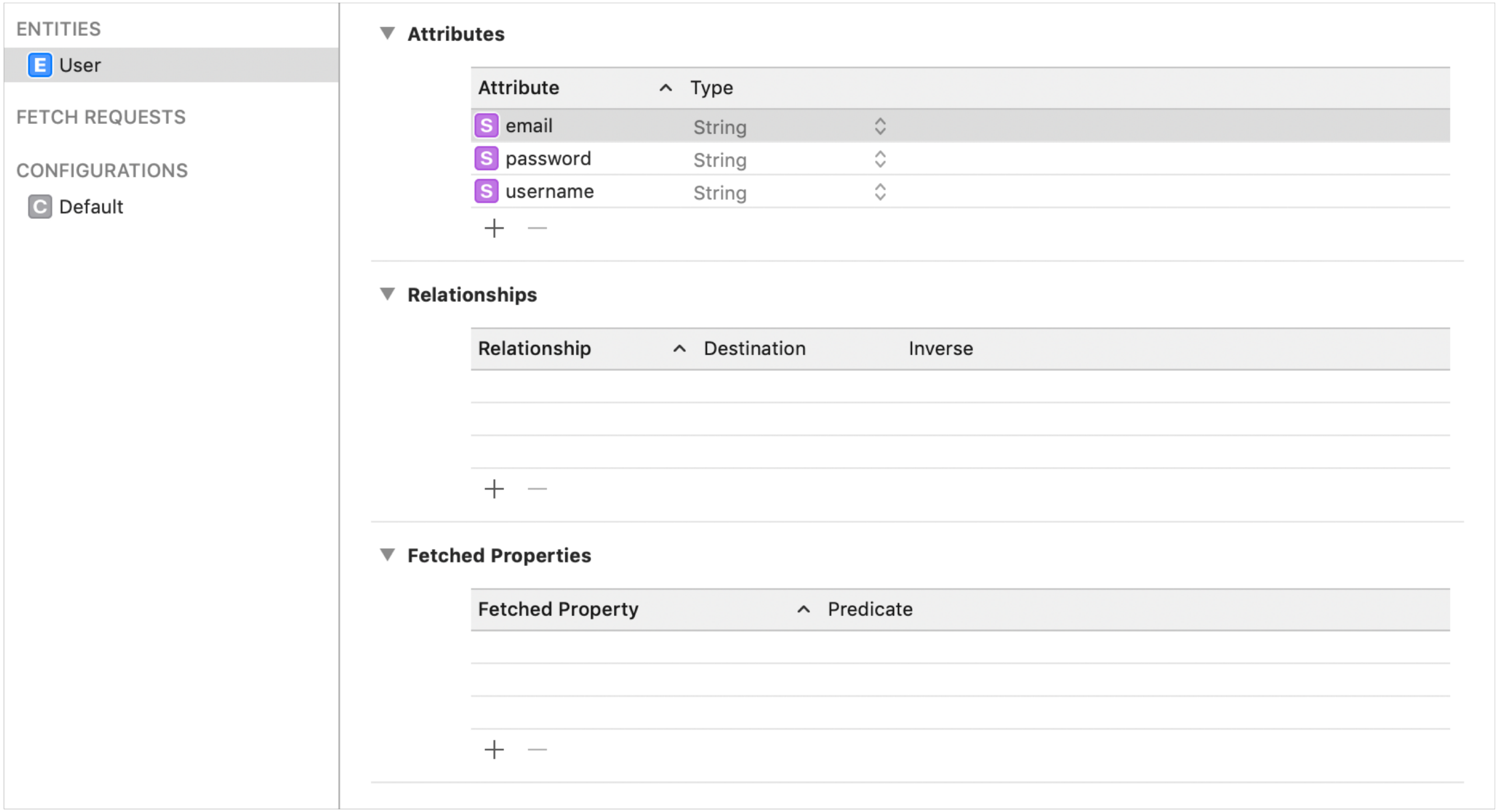
```
lazy var persistentContainer: NSPersistentContainer = {
    /*
     The persistent container for the application. This implementation
     creates and returns a container, having loaded the store for the
     application to it. This property is optional since there are legitimate
     error conditions that could cause the creation of the store to fail.
    */
```



```swift
import UIKit
import CoreData


class ViewController: UIViewController {

    func getContext () -> NSManagedObjectContext {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        return appDelegate.persistentContainer.viewContext

    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view

        // Print it to the console
        print(getContext())
    }
}
```

Entities

- An Entity in the code becomes an NSManagedObject



ENTITIES

E User

FETCH REQUESTS

CONFIGURATIONS

C Default

Attributes

Attribute	Type
S email	String
S password	String
S username	String

Relationships

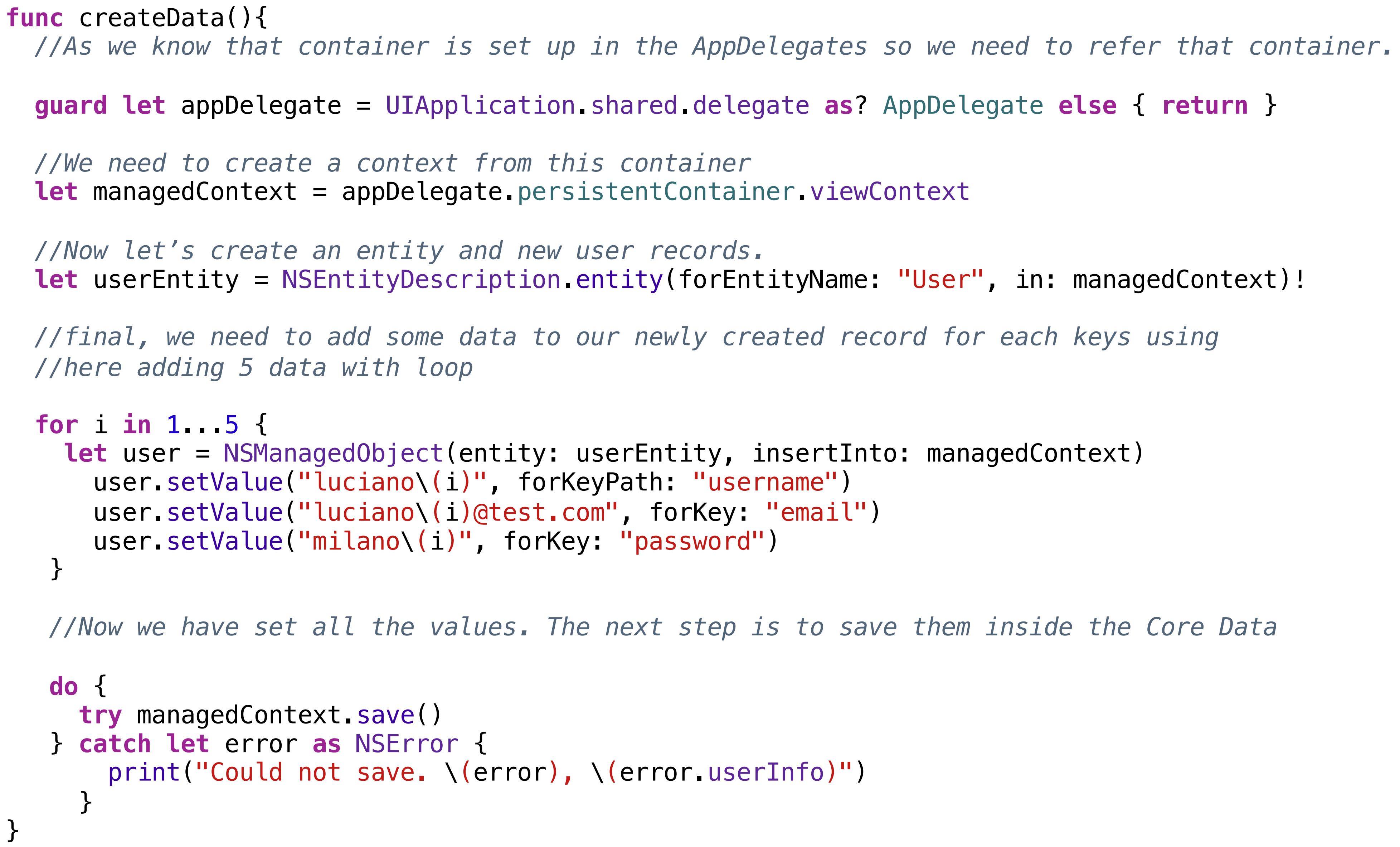
Relationship	Destination	Inverse

Fetched Properties

Fetched Property	Predicate

Entities

- All attributes are objects
- Attributes can be accessed easily
  - NSManagedObject offers valueForKey and setValue



```
func createData(){
  //As we know that container is set up in the AppDelegates so we need to refer that container.

  guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

  //We need to create a context from this container
  let managedContext = appDelegate.persistentContainer.viewContext

  //Now let’s create an entity and new user records.
  let userEntity = NSEntityDescription.entity(forEntityName: "User", in: managedContext)!

  //final, we need to add some data to our newly created record for each keys using
  //here adding 5 data with loop

  for i in 1...5 {
    let user = NSManagedObject(entity: userEntity, insertInto: managedContext)
      user.setValue("luciano\(i)", forKeyPath: "username")
      user.setValue("luciano\(i)@test.com", forKey: "email")
      user.setValue("milano\(i)", forKey: "password")
   }

   //Now we have set all the values. The next step is to save them inside the Core Data

   do {
     try managedContext.save()
   } catch let error as NSError {
       print("Could not save. \(error), \(error.userInfo)")
     }
}
```

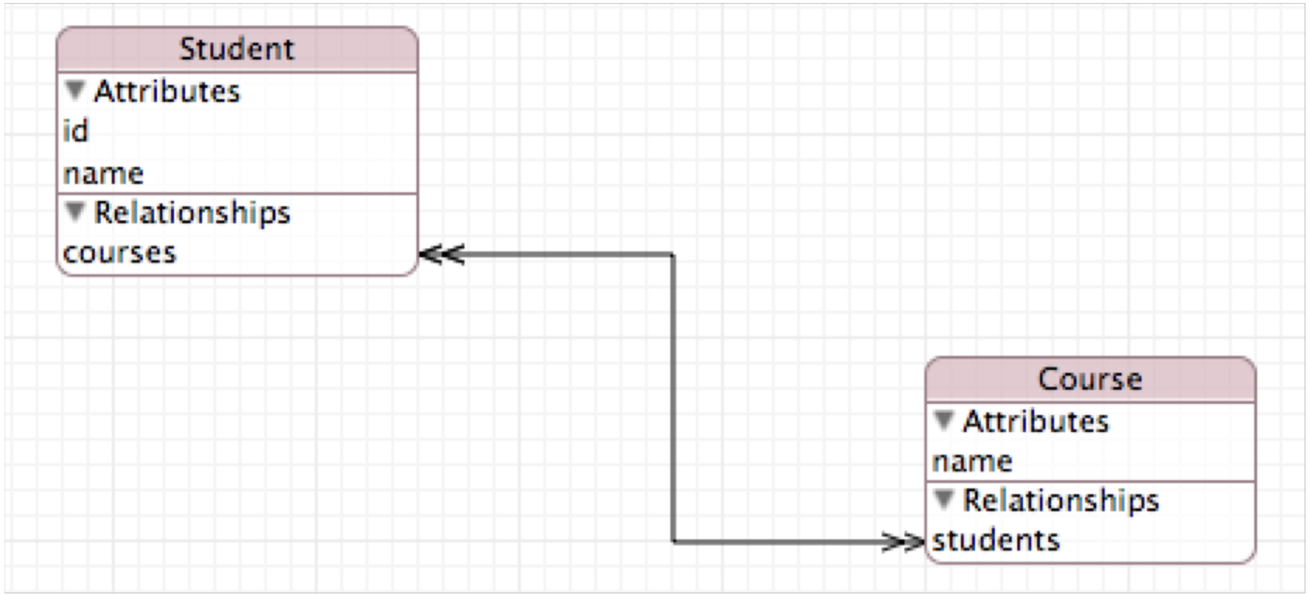
```swift
func retrieveData() {

  //As we know that container is set up in the AppDelegates so we need to refer that container.
  guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

  //We need to create a context from this container
  let managedContext = appDelegate.persistentContainer.viewContext

  //Prepare the request of type NSFetchRequest  for the entity
  let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "User")

  do {
    let result = try managedContext.fetch(fetchRequest)
    for data in result as! [NSManagedObject] {
      print(data.value(forKey: "username") as! String)
    }

  } catch {
    print("Failed")
    }
}
```

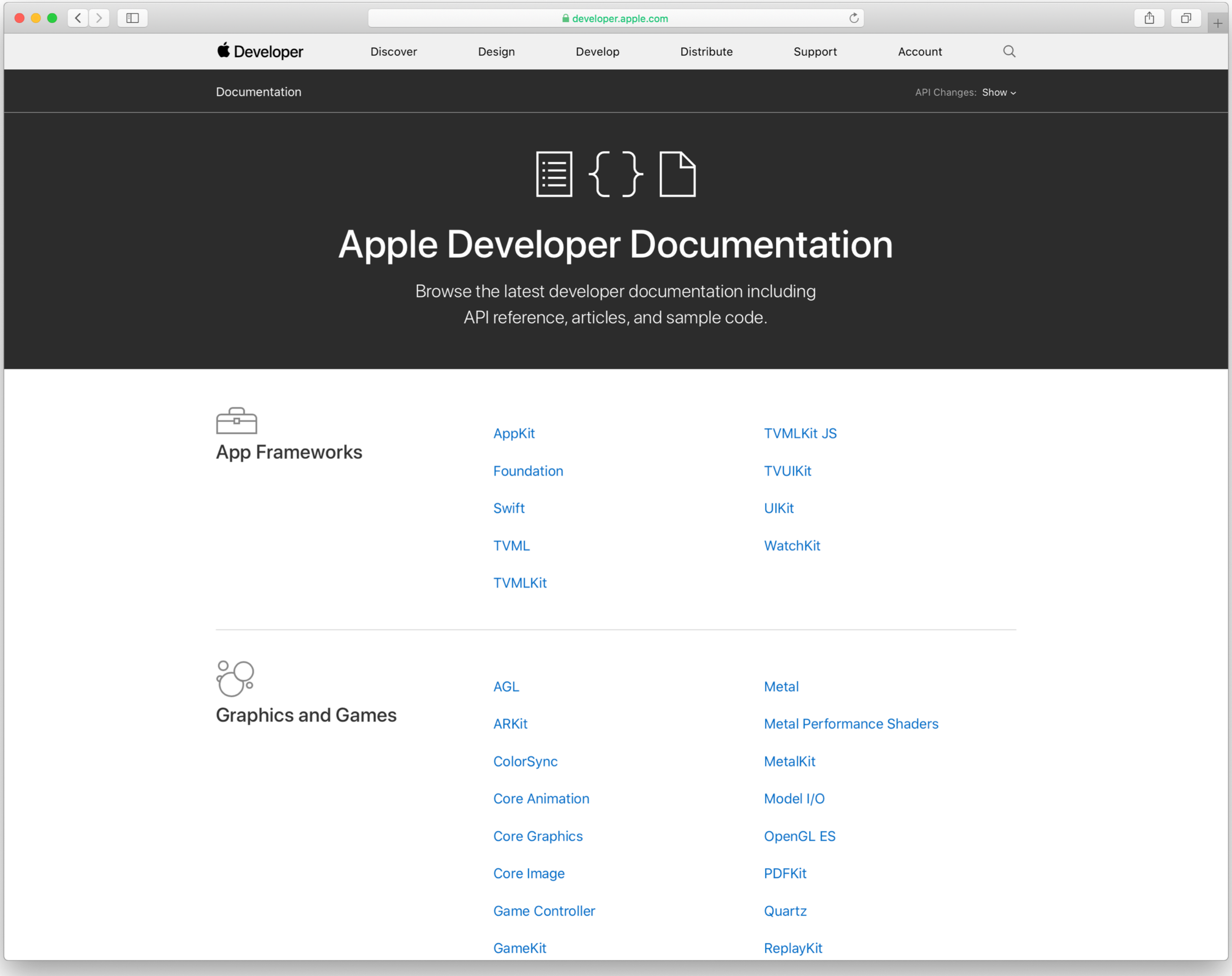
```swift
func updateData(){
  //As we know that container is set up in the AppDelegates so we need to refer that container.
  guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

  //We need to create a context from this container
  let managedContext = appDelegate.persistentContainer.viewContext

  let fetchRequest:NSFetchRequest<NSFetchRequestResult> = NSFetchRequest.init(entityName: "User")
  fetchRequest.predicate = NSPredicate(format: "username = %@", "luciano1")

  do {
    let test = try managedContext.fetch(fetchRequest)

    let objectUpdate = test[0] as! NSManagedObject
    objectUpdate.setValue("newName", forKey: "username")
    objectUpdate.setValue("newmail", forKey: "email")
    objectUpdate.setValue("newpassword", forKey: "password")
    do {
      try managedContext.save()
    }
    catch {
      print(error)
    }
  }
  catch {
    print(error)
  }
}
```

```
func deleteData(){
  //As we know that container is set up in the AppDelegates so we need to refer that container.
  guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

  //We need to create a context from this container
  let managedContext = appDelegate.persistentContainer.viewContext

  let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "User")
  fetchRequest.predicate = NSPredicate(format: "username = %@", "luciano3")

  do {
    let test = try managedContext.fetch(fetchRequest)

    let objectToDelete = test[0] as! NSManagedObject
    managedContext.delete(objectToDelete)

    do {
      try managedContext.save()
    }
    catch {
      print(error)
    }
  }
  catch {
    print(error)
  }
}
```

Relationships

- The type of relationships can be either toOne or to Many





Apple Developer Documentation

Browse the latest developer documentation including
API reference, articles, and sample code.

App Frameworks

- AppKit
- Foundation
- Swift
- TVML
- TVMLKit
- TVMLKit JS
- TVUIKit
- UIKit
- WatchKit

Graphics and Games

- AGL
- ARKit
- ColorSync
- Core Animation
- Core Graphics
- Core Image
- Game Controller
- GameKit
- Metal
- Metal Performance Shaders
- MetalKit
- Model I/O
- OpenGL ES
- PDFKit
- Quartz
- ReplayKit