

# **Chapter III**

## **Mastering Standard iOS UI Elements**



## **IV.Mastering Standard iOS UI Element**

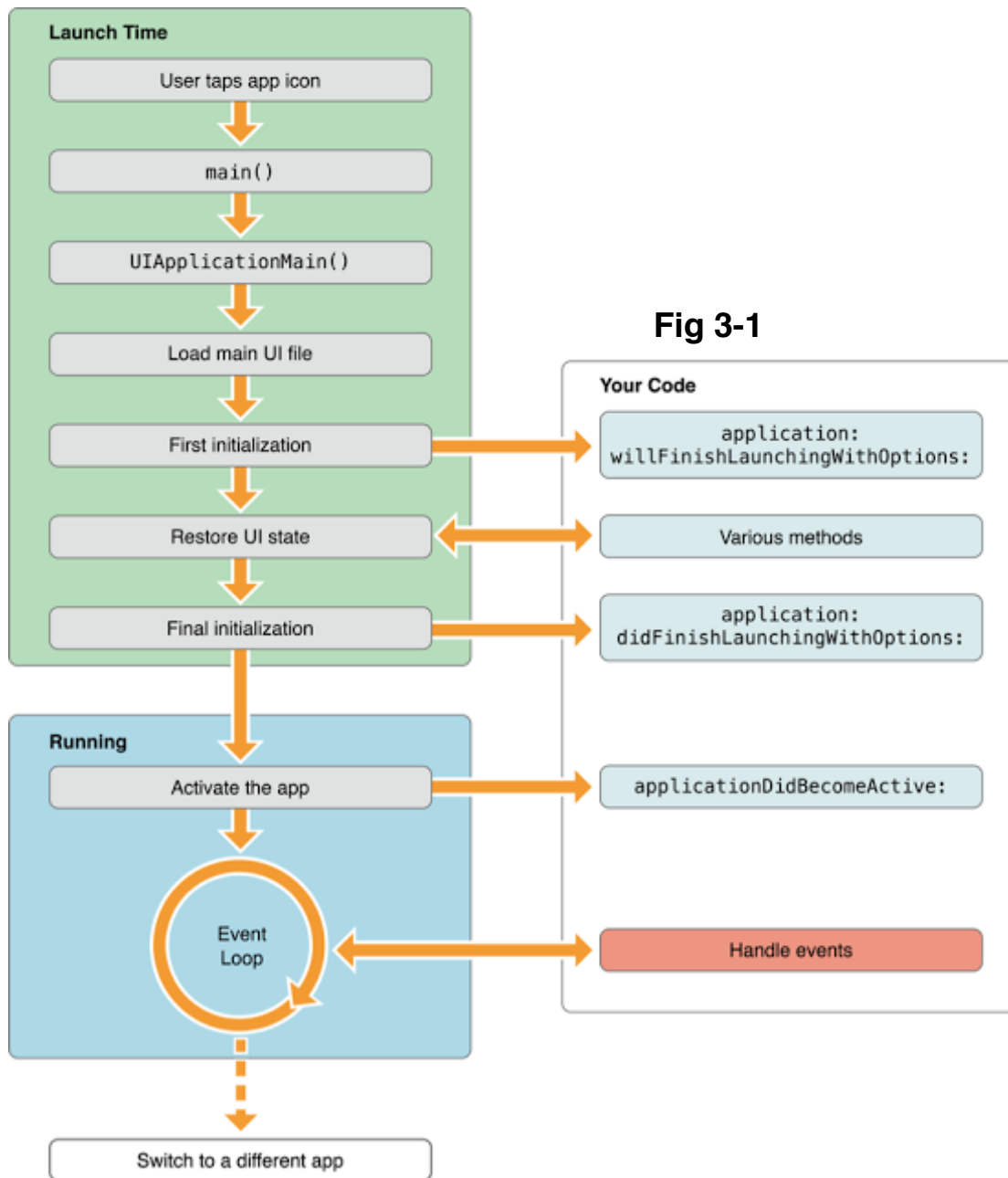
### **1. iOS App Application Life Circle**

The iOS operating system manages the application states, through out its life circle from start running to till termination.And the app is responsible for handling user-experience through these state transitions.

- Not Running — Either the application has not started yet or was running and has been terminated by the system.
- Inactive— An application is running in the Foreground but is not receiving any events. This could happen in case a Call or Message is received. An application could also stay in this state while in transition to a different state. In this State, we can not interact with app's UI.
- Active— An application is running in the Foreground and receiving the events. This is the normal mode for the Foreground apps. The only way to go to or from the Active state is through the Inactive state. User normally interacts with UI, and can see the response/result for user actions.
- Background — An application is running in the background and executing the code. Freshly launching apps directly enter into In-Active state and then to Active state. Apps that are suspended, will come back to this background state, and then transition to In-Active → Active states. In addition, an application being launched directly into the background enters this state instead of the inactive state.
- Suspended — An application is in the background but is not executing the code. The system moves the application to this state automatically and does not notify. In case of low memory, the system may purge suspended application without notice to make free space for the foreground application. Usually after 5 secs spent in the background, apps will transition to Suspend state, but we can extend the time if app needs.

iOS App Application life circle composes two portions:

- 1) The App Launch Cycle
- 2) The View Controller Life Cycle



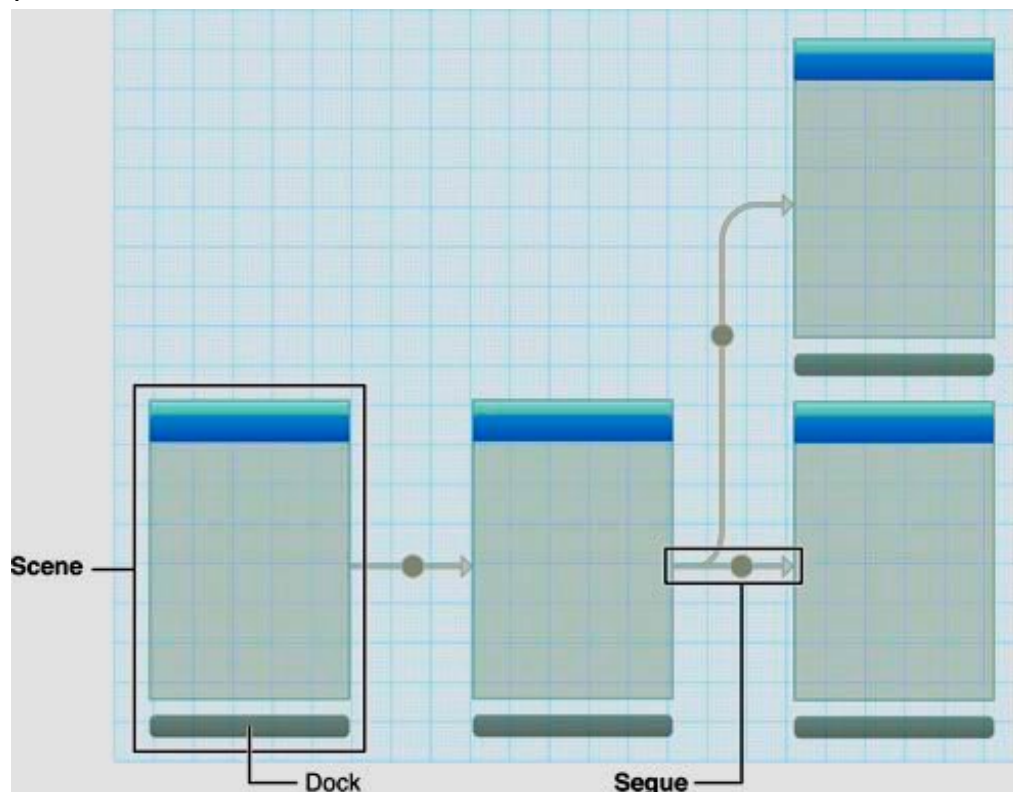
During the app launch cycle, the iOS system creates a process and main thread for your app and calls your app's main function on that main thread. The default main function that comes with your Xcode project promptly hands control over to the UIKit framework, it will initialise your app and prepares it to run as demonstrate in the fig 2.2. During View Controller Life Cycle, the view controllers will fire all these events:

- `viewDidLoad`
- `viewWillAppear.`
- `viewDidAppear`
- `viewWillDisappear`
- `viewDidDisappear`

## 2. Creating Storyboard & Scenes

A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views; scenes are connected by segue objects, which represent a transition between two view controllers. (Fig 1-4)

Xcode provides a visual editor for storyboards, where you can lay out and design the user interface of your application by adding views such as buttons, table views, and text views onto scenes. In addition, a storyboard enables you to connect a view to its controller object, and to manage the transfer of data between view controllers. Using storyboards enable you to visualize the appearance and flow of your user interface on one canvas.



**(Fig 3-2)**

**Note:** A Scene Corresponds to a Single View Controller and Its Views .A Segue Manages the Transition Between Two Scenes.

### 3. Containers : Windows & Views

A window handles the overall presentation of your app's user interface. Windows work with views and a window is just a blank container for one or more views to sit in. windows do not have title bars, close boxes, or any other visual adornments in iOS. Users don't see, close, or move the window of an iOS app. And instead of opening another window to display new content and instead an iOS app changes the views inside its window. It has several responsibilities:

- It contains your application's visible content.
- It plays a key role in the delivery of touch events to your views and other application objects.
- It works with your application's view controllers to facilitate orientation changes.

When you base a new iOS app project on one of the Xcode templates and use storyboards to design your user interface, you don't have to explicitly create, configure, or load your app's window.

When you create a main storyboard file for your app—and identify it as the main storyboard in your information property list file—iOS performs several setup tasks for you. Specifically, at launch time iOS:

- Instantiates a window.
- Loads the main storyboard and instantiates its initial view controller.
- Assigns the new view controller to the window's rootViewController property and then makes the window visible.

Before the initial view controller is displayed, your app delegate is called to give you a chance to configure the view controller.

Views can include others user interface elements and responsible to deliver touch event to its containers and others related tasks. and its can be identify by tag for futures reference.

## 4. Understanding UIView and its inheritance UI Elements

UIView Class Hierarchy for information

NSObject\UIResponder\**UIView**

- UIImageView
- UIInputView
- UILabel
- UINavigationBar
- UIPickerView
- UIPopoverBackgroundView
- UIProgressView
- UIScrollView
- UISearchBar
- UIStackView
- UITabBar
- UITableViewCell
- UITableViewHeaderFooterView
- UIToolbar
- UIVisualEffectView
- UIWebView
- UIWindow
- WKWebView
- ADBannerView
- CAInterAppAudioSwitcherView
- CAInterAppAudioTransportView
- GLKView
- HKActivityRingView
- MKAnnotationView
- MKMapView
- MKOverlayView
- MPVolumeView
- MTKView
- PHLivePhotoView
- SCNView
- SKView
- UIActionSheet
- UIActivityIndicatorView
- UIAlertView
- UICollectionReusableView
- UIControl

### 1. Views

An object that manages the content for a rectangular area on the screen. Views are the fundamental building blocks of your app's user interface, and the `UIView` class defines the behaviors that are common to all views. A view object renders content within its bounds rectangle and handles any interactions with that content.

**Ref:** <https://developer.apple.com/documentation/uikit/uiview>

### UIView by Code

```
let view1 = UIView(frame: CGRect(x:0,y:0,width:width * 2 / 3,height:height * 2/3))

view1.backgroundColor = UIColor.green

view1.tag = 100

view1.alpha = 0.4

view.addSubview(view1)
```

Eg. **0302UITest\_kitchensink**

Eg. **0303ViewDemo1**

## 2. UIImageView

An object that displays a single image or a sequence of animated images in your interface.

Ref: <https://developer.apple.com/documentation/uikit/uiimageView>

### Image View by Code

```
let imageView = UIImageView()  
  
imageView.frame.size = view.frame.size  
  
imageView.image = UIImage(named: "yourimagefilename")  
  
imageView.contentMode = .scaleAspectFill  
  
imageView.clipsToBounds = true  
  
view.addSubview(imageView)
```

Eg. **0304basicUIs\_UIImageViewSample**

**0305basicUIs\_UIImageViewSampleByCode**

## 3. Label

A label displays static text and Displays any amount of static text  
Doesn't allow user interaction except, potentially, to copy the text  
Use a label to name or describe parts of your UI or to provide short messages to the user. A label is best suited for displaying a relatively small amount of text.

## Label by Code

```
let label = UILabel()  
  
label.frame = CGRect(x:10,y:10, width:200, height:80 )  
  
label.text = "This is label "  
  
label.textColor = UIColor.green  
  
view.addSubview( label )
```

**Eg:0306Labeldemo1**

## 4. Buttons (System Button)

A system button performs an app-specific action.

Button by Code

```
let button1 = UIButton(frame: CGRect(x: margin, y: topMargin, width: 200, height: 100))  
  
button1.setTitle("Click Me", for: .normal)  
  
button1.backgroundColor = UIColor.green  
  
view.addSubview(button1)  
  
button1.setTitle("Clicked", for: .highlighted)
```

**Eg:0307Buttondemo1**

## 5. Switch

A switch presents two mutually exclusive choices or states. and it indicates the binary state of an item and it is used specifically in table views only.

Fill in the blank



```
let sw1 = UISwitch()  
  
sw1.center = CGPoint(x: 120, y: 150)
```

**Eg: 0308SwitchDemo1**

## 6. Stepper

A stepper increases or decreases a value by a constant amount. and supports custom images. But use a stepper when users might need to make small adjustments to a value and avoid using a stepper when users are likely to make large changes to a value.

### Stepper by code

```
let stepper1 = UIStepper()  
  
stepper1.center = CGPoint(x: view.frame.size.width / 2, y: 80)  
view.addSubview(stepper1)
```

**Eg: 0309StepperDemo1**

## 7. Segmented control

A segmented control is a linear set of segments, each of which functions as a button that can display a different view. Consists of two or more segments whose widths are proportional, based on the total number of segments and can display text or images

Use a segmented control to offer choices that are closely related

## Segment View by Code

```
let seg1 = UISegmentedControl(items:
                                ["One", "Two", "Three", "Four"])
seg1.frame = CGRect(x: 40, y: 40, width:
view.frame.size.width - 80, height: 60)

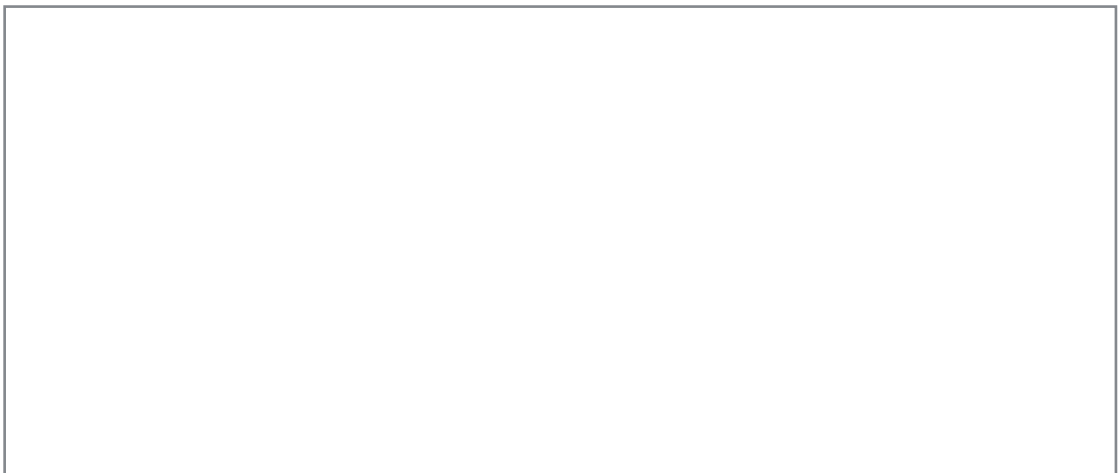
view.addSubview(seg1)
```

### Eg.0310Segmentdemo1

## 8. Slider Control

A slider allows users to make adjustments to a value or process throughout a range of allowed values. Use a slider to give users fine-grained control over values they can choose or over the operation of the current process.

Fill in the blank



## 9. Progress View

A progress view shows the progress of a task or process that has a known duration. Use a progress view to give feedback on a task that has a well-

defined duration, especially when it's important to indicate approximately how long the task will take.

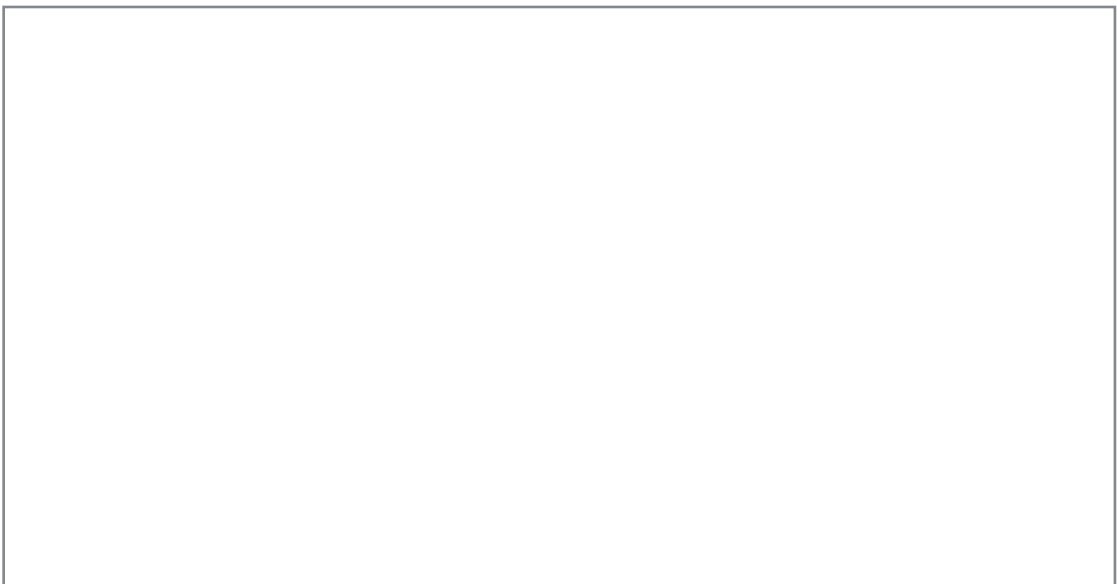
Fill in the blank



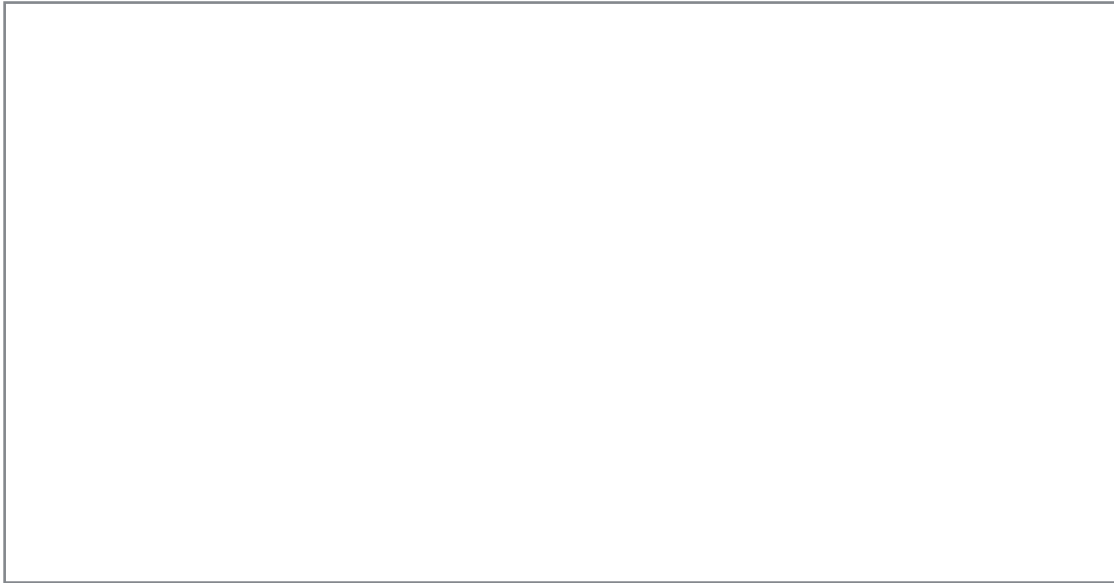
## **10. Text Field**

A text field accepts a single line of user input

Fill in the blank



## 11. Toolbar

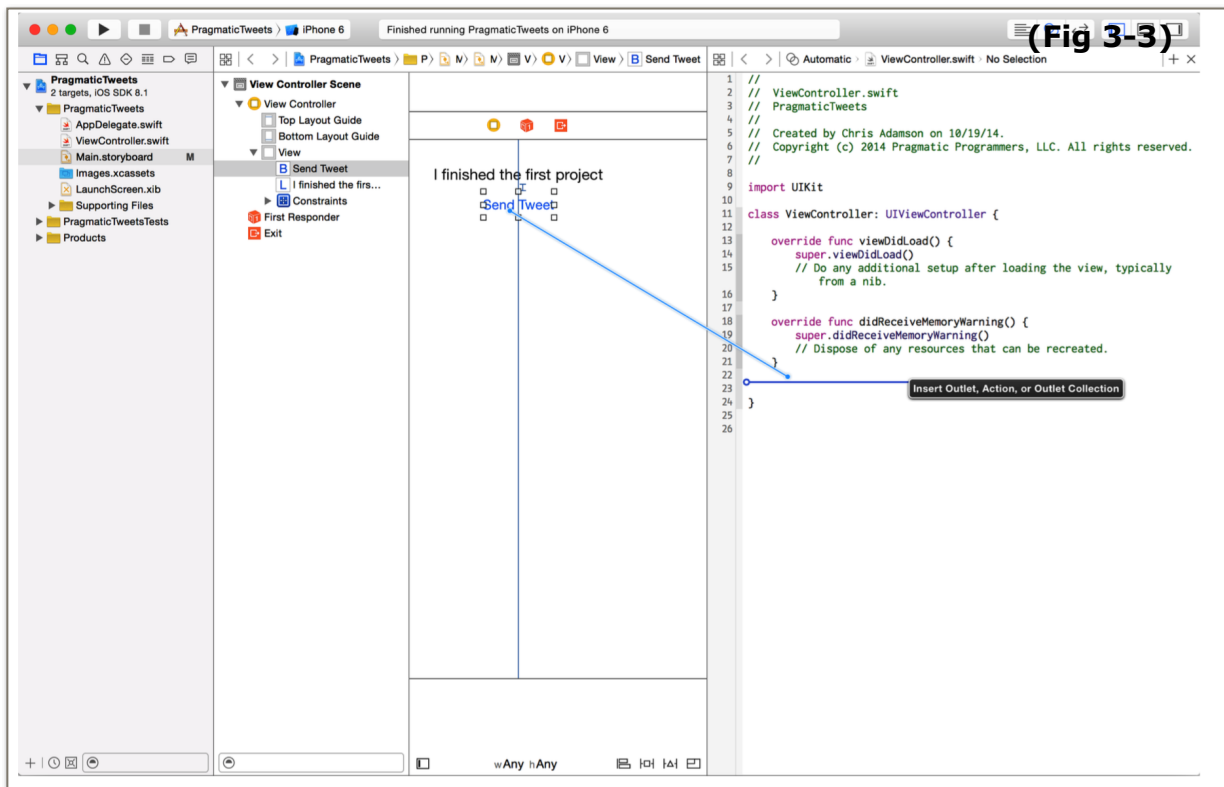


## 5. IBOutlet & IBActions

In iOS, we use Interface Builder connections to tie the user interface to our code. Using Xcode, we can create two kinds of connections:

- **IBOutlet** : An outlet connects a variable or property in code to an object in a storyboard. This lets us read and write the object's properties, like reading the value of a slider or setting the initial contents of a text field.

- **IBAction**: An action connects an event generated by a storyboard object to a method in our code. This lets us respond to a button being tapped or a slider's value changing. (See Fig-1-3)



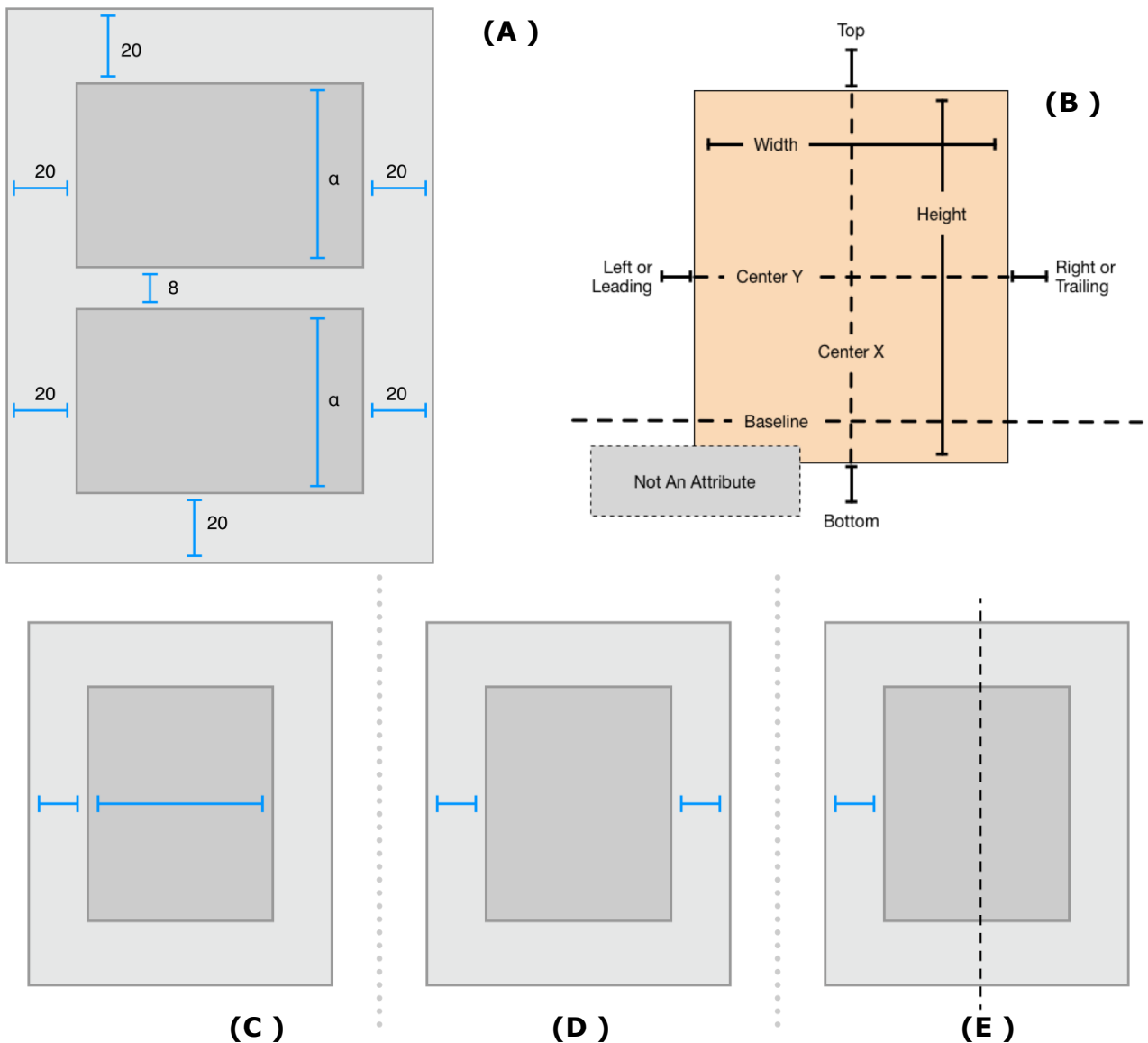
**Eg. 0312BasicUIElement\_IBOutWithFunctionCoffeeshop**  
**0313IBActionArray\_Picturegallery**  
**0314IBoutlet\_spaceX**

## 6. Autolayout & Constrain

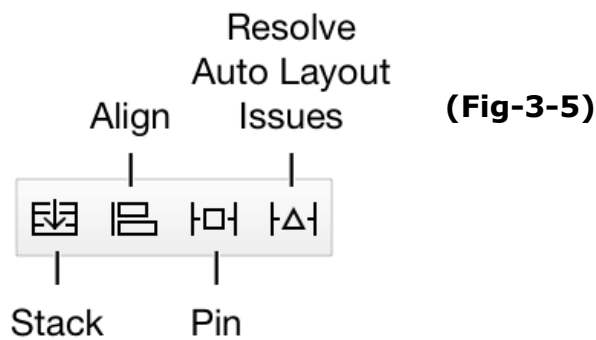
### 1. Autolayout in Storyboard

Auto Layout dynamically calculates the size and position of all the views in your view hierarchy, based on constraints placed on those views. This constraint-based approach to design allows you to build user interfaces that dynamically respond to both internal and external changes.

Auto Layout defines your user interface using a series of constraints. Constraints typically represent a relationship between two views. Auto Layout then calculates the size and location of each view based on these constraints. This produces layouts that dynamically respond to both internal and external changes.



(Fig-3-4)



There are two basic types of attributes. Size and Location

Size attributes (for example, Height and Width) and location attributes (for example, Leading, Left, and Top). Size attributes are used to specify how large an item is, without any indication of its location. Location attributes are used to specify the location of an item relative to something else. However, they carry no indication of the item's size.

In general, the constraints must define both the size and the position of each view. satisfiable layout requires two constraints per view per dimension (Fig 1-6 C)

## 2. Autolayout by Code

Laying out a view or its sibling by code is followed by :

1) add the required component to its parent views

2) Set the component's property :

**translatesAutoresizingMaskIntoConstraints**  to false

3) Then apply the autolayout rules as following techniques.

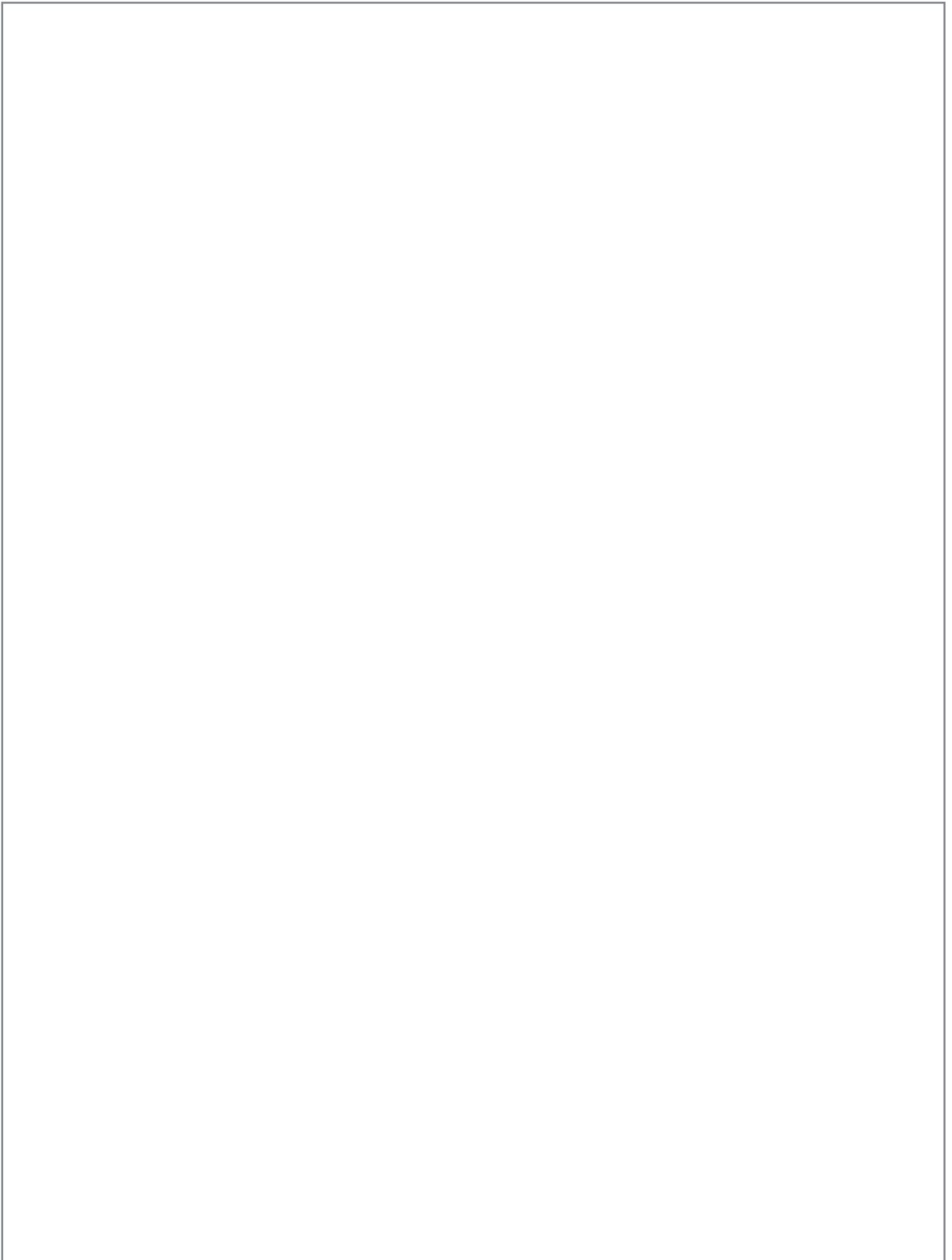
**Type 1: Setting Leading , Trailing , Top & Bottom**

**Type 2: Setting Center with fixed width and Height**

**Type 3: Setting Promotional Width and Height**

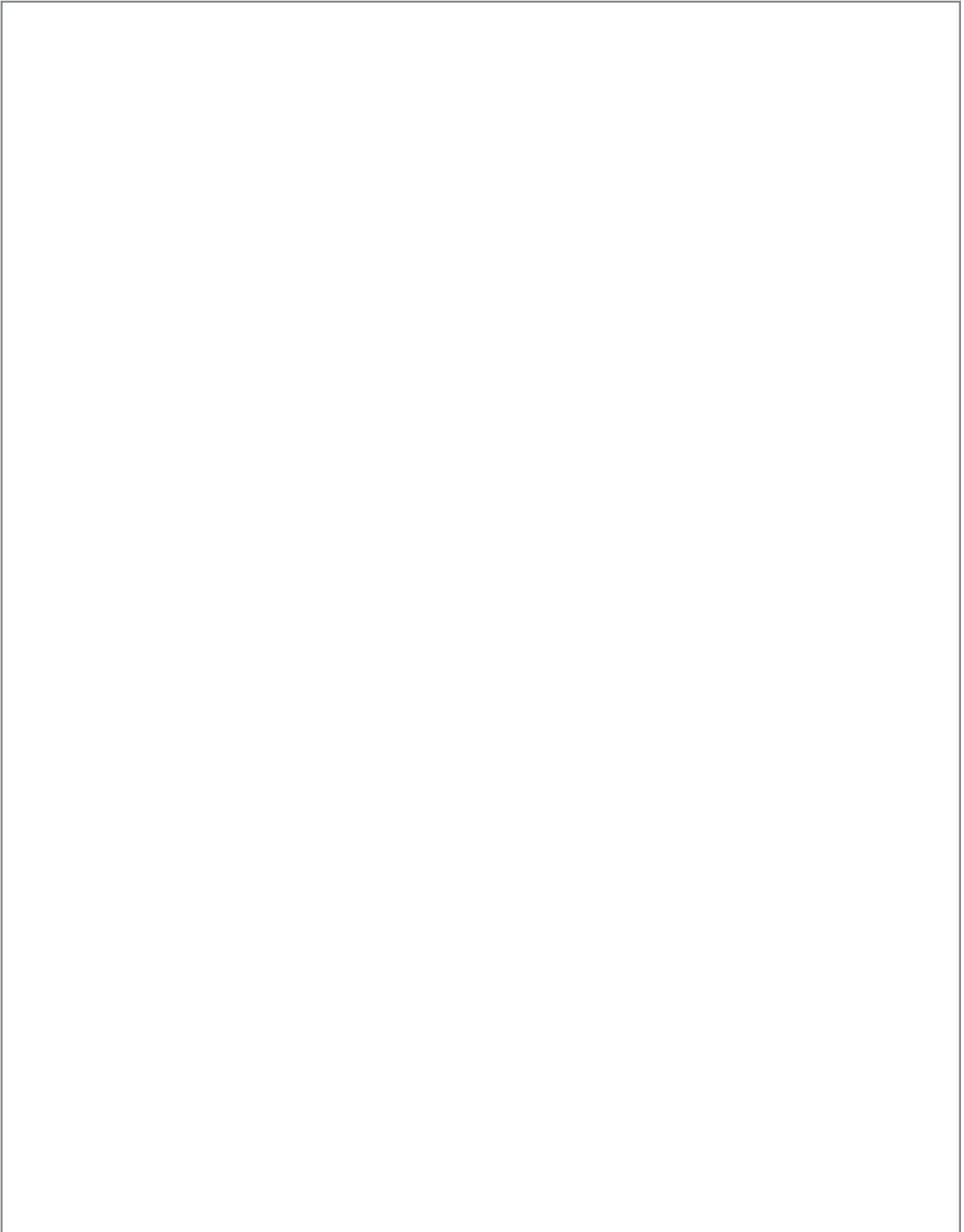
**Type 4: Setting reference to other UI**

## Type1: Setting Leading, Trailing , Top & Bottom

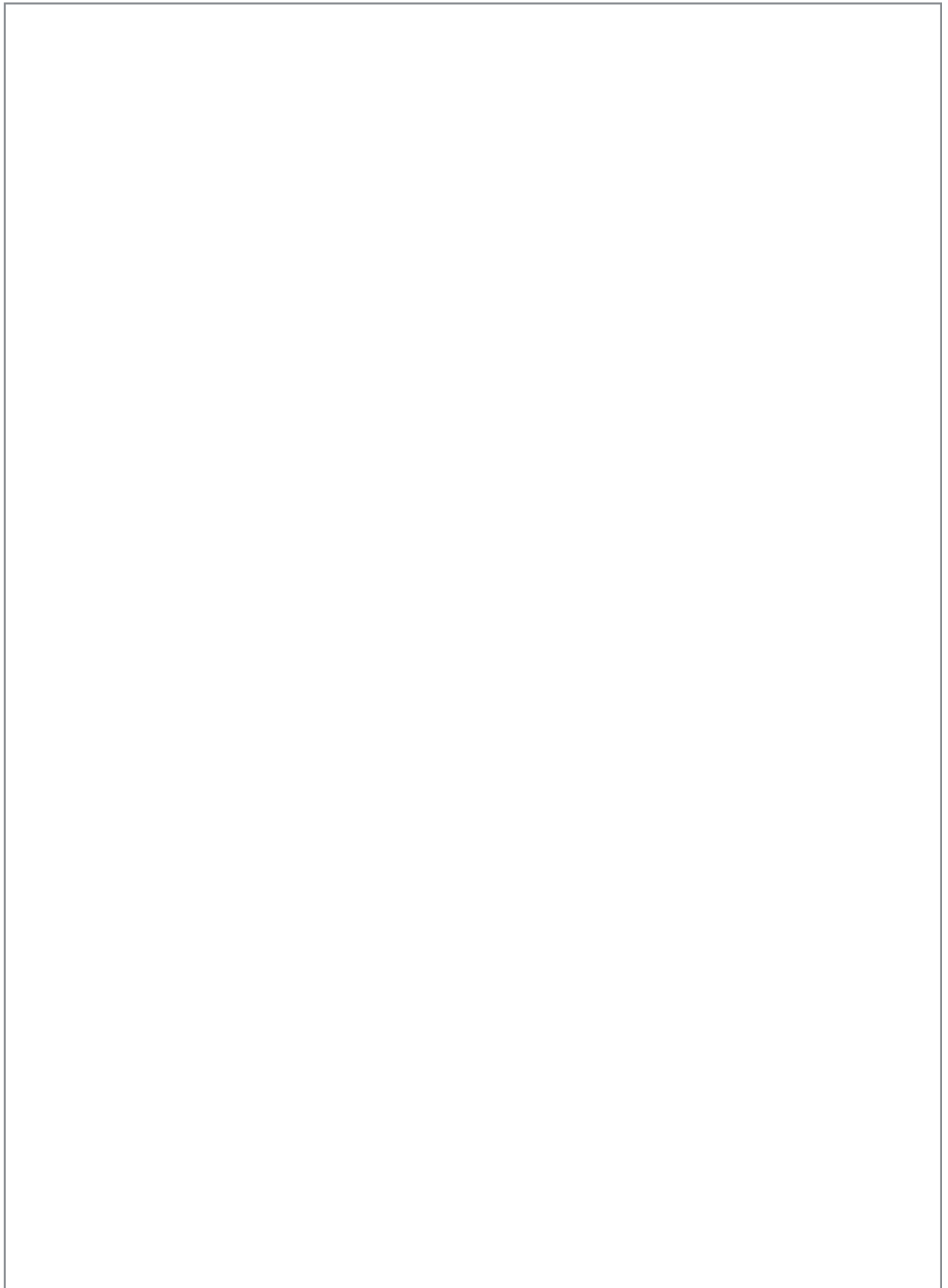




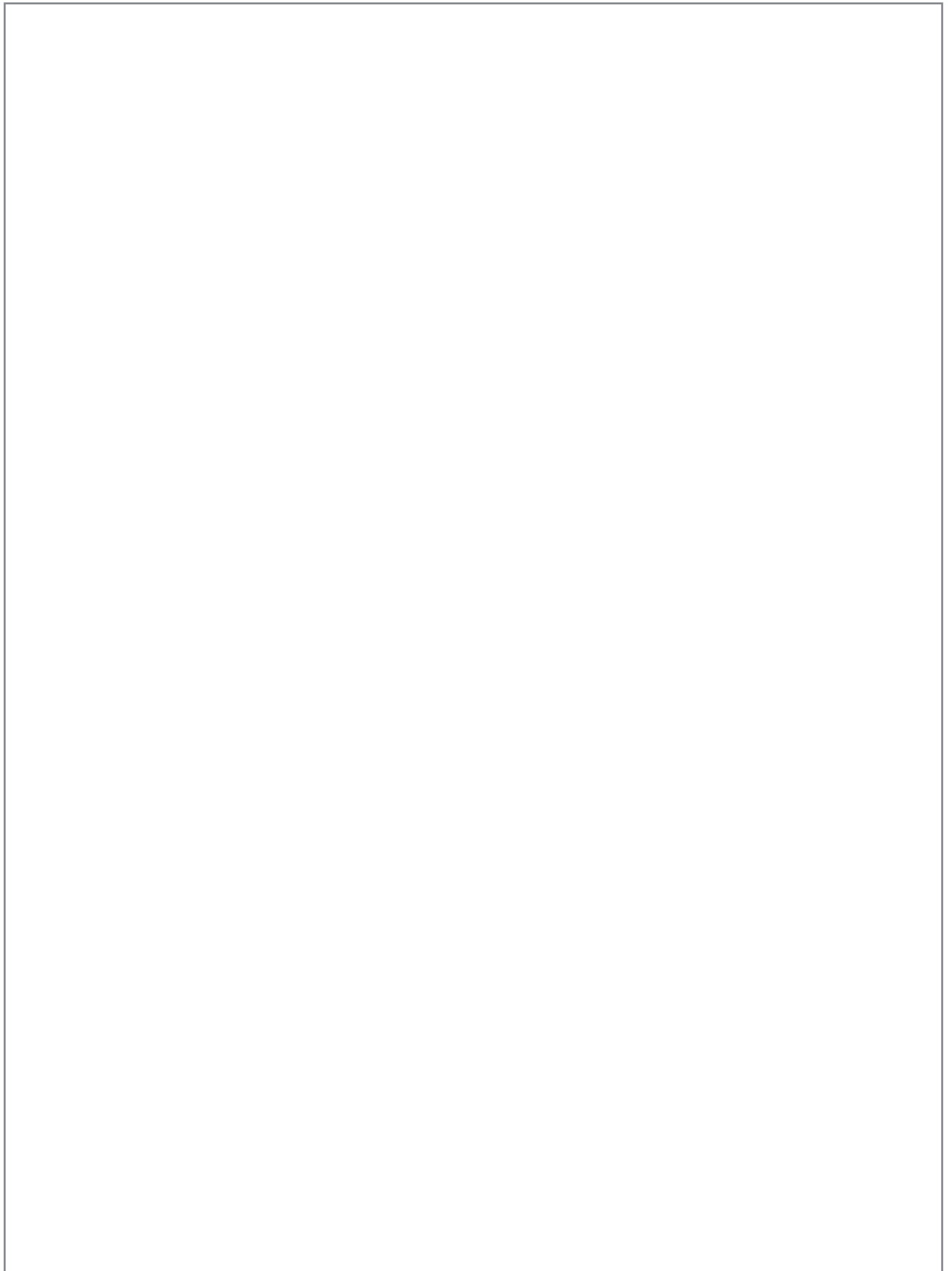
## Type2: Center with fixed width and Height



### Type 3: Setting Proportional Width and Height

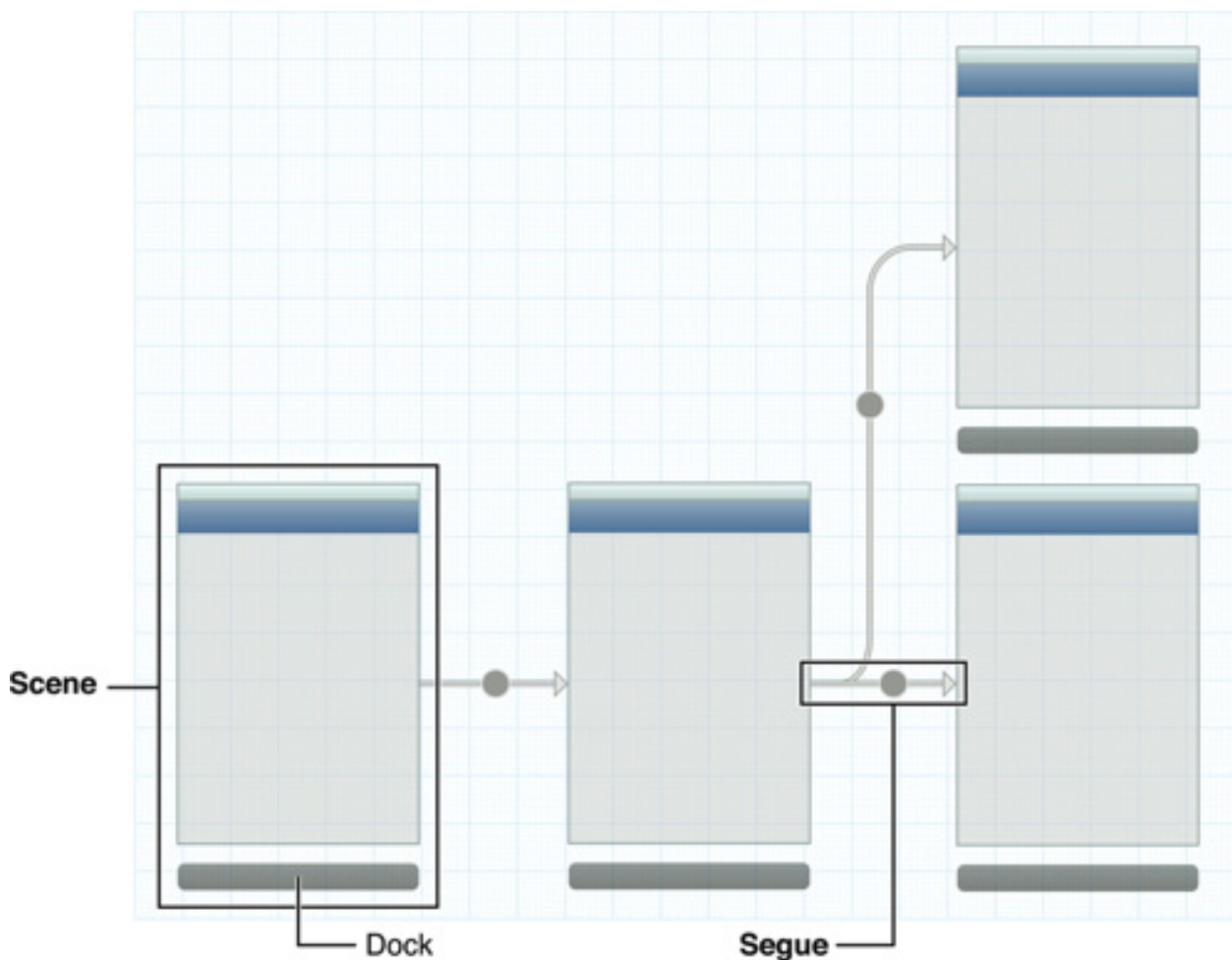


#### Type 4: Setting reference to other UI



## Chapter 3: Exercise

1. Create New iOS App Project using Single View Application Template and run it in iOS 10: iPhone 6, iPhone 6s, iPad Air 2 Simulator.
2. Create New iOS App Project using Master Detail View Application Template and run it as in exercise 1
3. Test the Xcode debugging features with Break point and using print function.
4. Create new iOS App using Single View Application Template and add all basic UI Elements stated in Article 1.5.2
5. Create new iOS App layout as shown.



2. Create new iOS App using Single View Application Template and add Two more Viewcontroller and then create transition between each view controller with segue.(Hint: use button to trigger segue to jump to another screen)
3. Create and set the following two views' constrain to make looking good in portrait and landscape as shown

