iOS - Quick Guide

Previous Page Next Pag

iOS - Getting Started

General Overview

iOS, which was previously called iPhone OS, is a mobile operating system developed by Apple Inc. Its first release was in 2007, which included iPhone and iPod Touch. iPad (1st Generation) was released in April 2010 and iPad Mini was released in November 2012.

The iOS devices get evolved quite frequently and from experience, we find that at least one version of iPhone and iPad is launched every year. Now, we have iphone5 launched which has its predecessors starting from iPhone, iPhone 3gs, iPhone 4, iPhone 4s. Similarly, iPad has evolved from iPad (1st Generation) to iPad (4th Generation) and an additional iPad Mini version.

The iOS SDK has evolved from 1.0 to 6.0. iOS 6.0, the latest SDK is the only officially supported version in Xcode 4.5 and higher. We have a rich Apple documentation and we can find which methods and libraries can be used based on our deployment target. In the current version of Xcode, we'll be able to choose between deployment targets of iOS 4.3, 5.0 and 6.0.

The power of iOS can be felt with some of the following features provided as a part of the device.

- Maps
- Siri
- Facebook and Twitter
- Multi-Touch
- Accelerometer
- GPS
- High end processor
- Camera
- Safari
- Powerful APIs

Game center

In-App Purchase

Reminders

Wide Range of gestures

The number of users using iPhone/iPad has increased a great deal. This creates the opportunity for developers to make money by creating applications for iPhone and iPad the Apple's App Store.

For some one new to iOS, Apple has designed an application store where the user can buy apps developed for their iOS devices. A developer can create both free and paid apps to App Store. To develop applications and distribute to the store, the developer will require to register with iOS developer program which costs $99 a year and a Mac with Mountain Lion or higher for its development with latest Xcode.

Registering as an Apple Developer

An Apple ID is most necessary if you are having any Apple device and being a developer, you definitely need it. It's free and hence, no issues in having one. The benefits of having an Apple account are as follows −

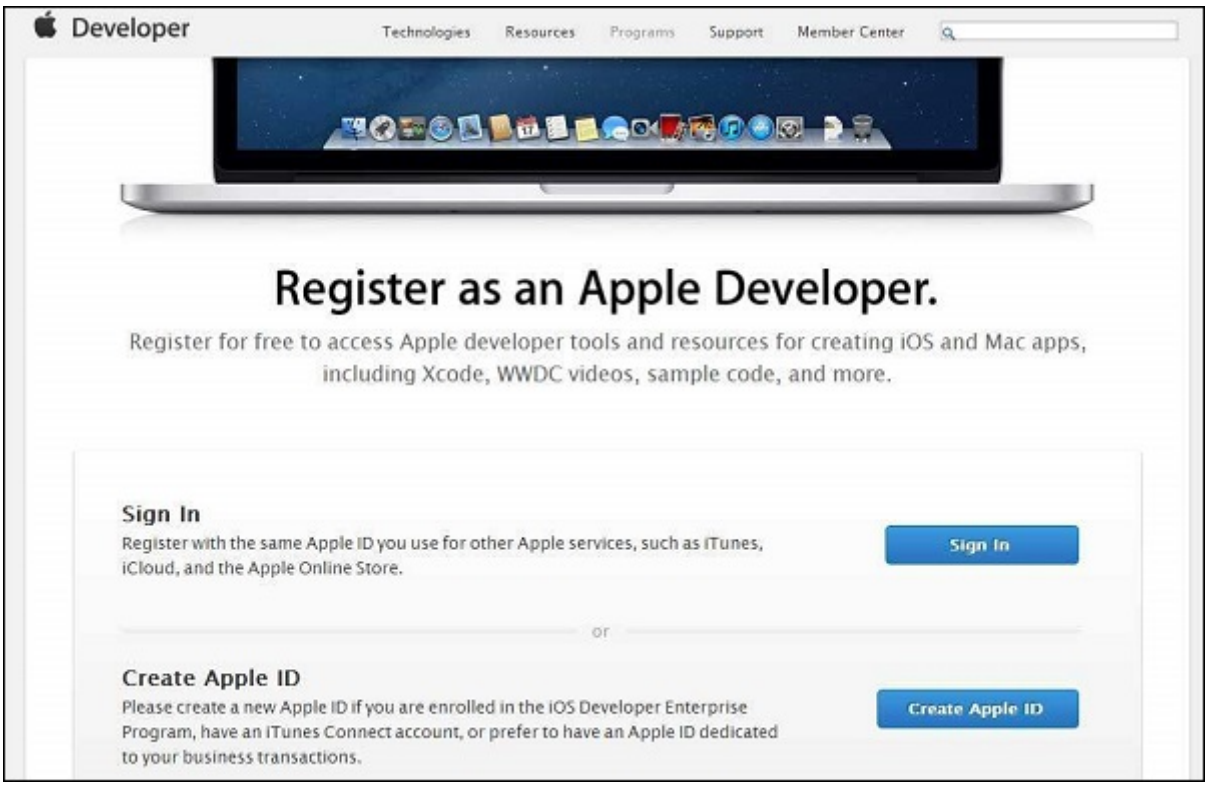
Access to development tools.

Worldwide Developers Conference (WWDC) videos.

Can join iOS developer program teams when invited.

To register an Apple account, follow the steps given below −

Step 1 − Click the link https://developer.apple.com/programs/register/ and select "Create Apple ID"



Step 2 − Provide the necessary information, which is self explanatory as given in the page.

Step 3 − Verify your account with your email verification and the account becomes active.

Step 4 − Now you will be able to download the developer tools like Xcode, which is packaged with iOS simulator and iOS SDK, and other developer resources.

Apple iOS Developer Program

The first question that would arise to a new developer is – Why should I register for an iOS developer program? The answer is quite simple; Apple always focuses on providing quality applications to its user. If there was no registration fee, there could be a possibility of junk apps being uploaded that could cause problems for the app review team of Apple.

The benefits of joining the iOS developer program are as follows −

Run the apps you develop on the real iOS device.

Distribute the apps to the app store.

Get access to the developer previews.

The steps to join the iOS developer program are as follows −

Step 1 − To register, click on the link − (https://developer.apple.com/programs/ios/).

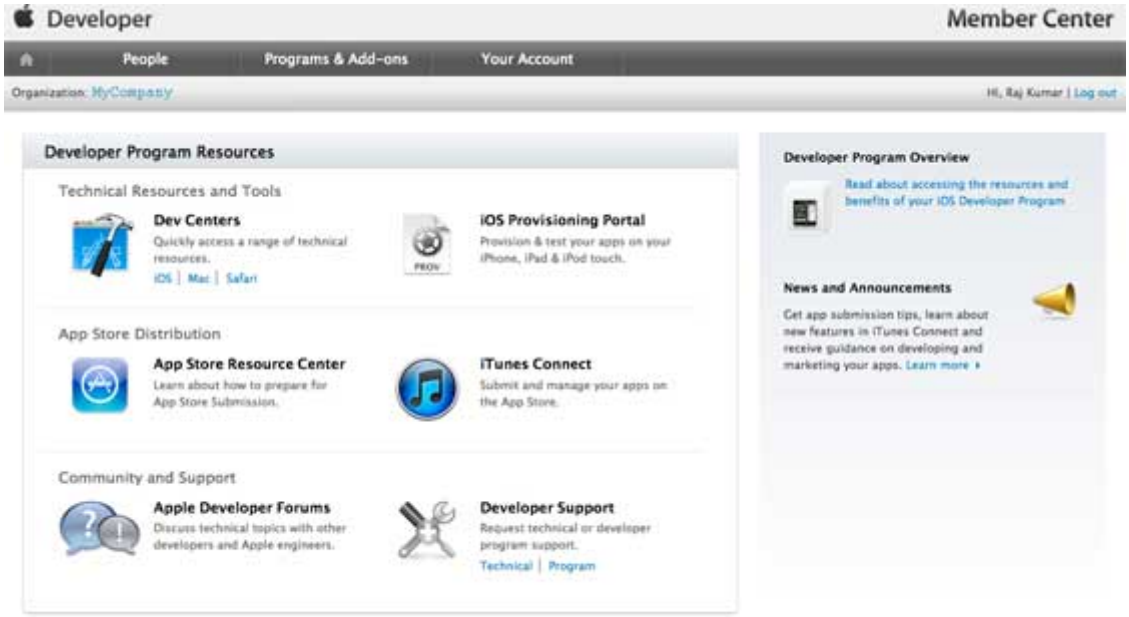
Step 2 − Click on Enroll Now in the page that is displayed.

Step 3 − You can either sign in to your existing apple account (if you have one) or create a new Apple ID.

Step 4 − Thereafter, you have to select between Individual and Company accounts. Use company account if there will be more than one developer in your team. In individual account, you can't add members.

Step 5 − After entering the personal information (for those who newly registers), you can purchase and activate the program by paying with the help of your credit card (only accepted mode of payment).

Step 6 − Now you will get access to developer resources by selecting the member center option in the page.



Step 7 − Here you will be able to do the following −

Create provisioning profiles.

Manage your team and devices.

Managing application to app store through iTunes Connect.

Get forum and technical support.

iOS - Environment Setup

iOS - Xcode Installation

Step 1 − Download the latest version of Xcode from https://developer.apple.com/downloads/

Step 2 − Double click the Xcode dmg file.

Step 3 − You will find a device mounted and opened.

Step 4 − There will be two items in the window that's displayed namely, Xcode application and the Application folder's shortcut.

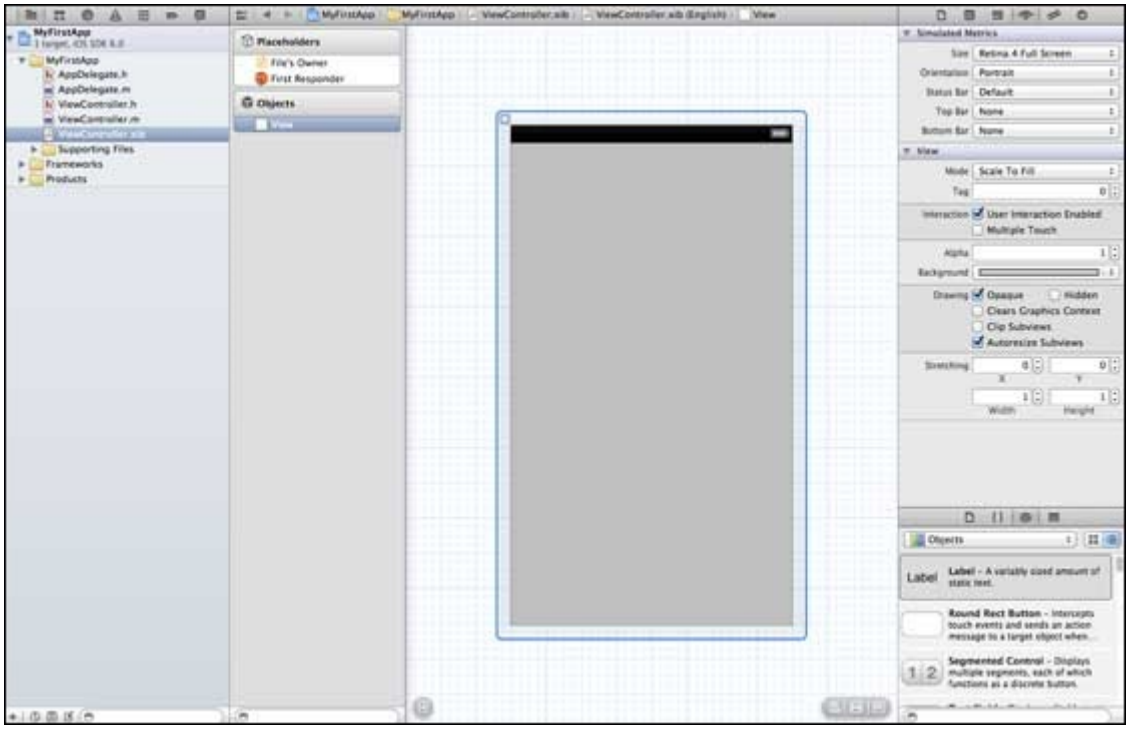
Step 5 − Drag the Xcode to application and it will be copied to your applications.

Step 6 − Now Xcode will be available as a part of other applications from which you can select and run.

You also have another option of downloading Xcode from the Mac App store and then install following the step-by-step procedure given on the screen.

Interface Builder

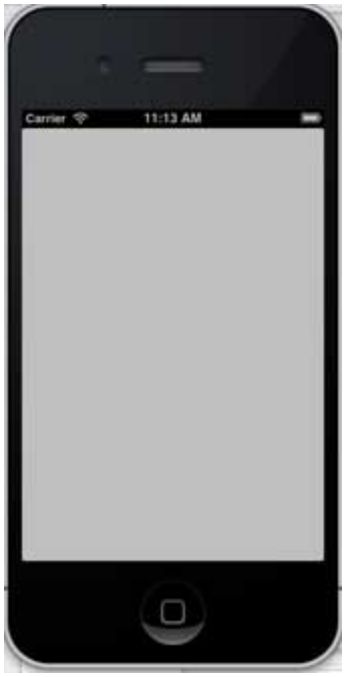
Interface builder is the tool that enables easy creation of UI interface. You have a rich set of UI elements that is developed for use. You just have to drag and drop into your UI view. We'll learn about adding UI elements, creating outlets and actions for the UI elements in the upcoming pages.



You have objects library at the right bottom that consists the entire necessary UI element. The user interface is often referred as **xibs**, which is its file extension. Each of the xibs is linked to a corresponding view controller.

iOS Simulator

An iOS simulator actually consists of two types of devices, namely iPhone and iPad with their different versions. iPhone versions include iPhone (normal), iPhone Retina, iPhone 5. iPad has iPad and iPad Retina. A screenshot of an iPhone simulator is displayed below.



You can simulate location in an iOS simulator for playing around with latitude and longitude effects of the app. You can also simulate memory warning and in-call status in the simulator. You can use the simulator for most purposes, however you cannot test device features like accelerometer. So, you might always need an iOS device to test all the scenarios of an application thoroughly.

iOS - Objective C

The language used in iOS development is objective C. It is an object-oriented language and hence, it would be easy for those who have some background in object-oriented programming languages.

Interface and Implementation

In Objective C, the file where the declaration of class is done is called the **interface file** and the file where the class is defined is called the **implementation file**.

A simple interface file **MyClass.h** would look like the following −

```
@interface MyClass:NSObject {
   // class variable declared here
}

// class properties declared here
// class methods and instance methods declared here
@end
```

The implementation file **MyClass.m** would be as follows −

```
@implementation MyClass
   // class methods defined here
@end
```

Object Creation

Object creation is done as follows −

```
MyClass  *objectName = [[MyClass alloc]init] ;
```

Methods

Method is declared in Objective C as follows −

```
-(returnType)methodName:(typeName) variable1 :(typeName)variable2;
```

An example is shown below.

```
-(void)calculateAreaForRectangleWithLength:(CGfloat)length
andBreadth:(CGfloat)breadth;
```

You might be wondering what the **andBreadth** string is for; actually it's an optional string, which helps us read and understand the method easily, especially at the time of calling. To call this method in the same class, we use the following statement −

```
[self calculateAreaForRectangleWithLength:30 andBreadth:20];
```

As said above, the use of andBreadth helps us understand that breadth is 20. Self is used to specify that it's a class method.

Class Methods

Class methods can be accessed directly without creating objects for the class. They don't have any variables and objects associated with it. An example is shown below.

```
+(void)simpleClassMethod;
```

It can be accessed by using the class name (let's assume the class name as MyClass) as follows −

```
[MyClass simpleClassMethod];
```

Instance Methods

Instance methods can be accessed only after creating an object for the class. Memory is allocated to the instance variables. An example instance method is shown below.

```
-(void)simpleInstanceMethod;
```

It can be accessed after creating an object for the class as follows −

```
MyClass  *objectName = [[MyClass alloc]init] ;
[objectName simpleInstanceMethod];
```

Important Data Types in Objective C

Sr.No.	Data Type
1	**NSString** It is used for representing a string.
2	**CGfloat** It is used for representing a floating point value (normal float is also allowed but it's better to use CGfloat).
3	**NSInteger** It is used for representing integer.
4	**BOOL** It is used for representing Boolean (YES or NO are BOOL types allowed).

Printing Logs

NSLog - used for printing a statement. It will be printed in the device logs and debug console in release and debug modes respectively. For example,

```
NSlog(@"");
```

Control Structures

Most of the control structures are same as in C and C++, except for a few additions like for in statement.

Properties

For an external class to access the class, variable properties are used. For example,

```
@property(nonatomic , strong) NSString *myString;
```

Accessing Properties

You can use dot operator to access properties. To access the above property, we will do the following.

```
self.myString = @"Test";
```

You can also use the set method as follows −

```
[self setMyString:@"Test"];
```

Categories

Categories are used to add methods to the existing classes. By this way, we can add method to classes for which we don't have even implementation files where the actual class is defined. A sample category for our class is as follows −

```
@interface MyClass(customAdditions)
- (void)sampleCategoryMethod;
@end

@implementation MyClass(categoryAdditions)

-(void)sampleCategoryMethod {
   NSLog(@"Just a test category");
}
```

Arrays

NSMutableArray and NSArray are the array classes used in objective C. As the name suggests, the former is mutable and the latter is immutable. An example is shown below.

```
NSMutableArray *aMutableArray = [[NSMutableArray alloc]init];
[anArray addObject:@"firstobject"];
NSArray *aImmutableArray = [[NSArray alloc]
initWithObjects:@"firstObject",nil];
```

Dictionary

NSMutableDictionary and NSDictionary are the dictionary classes used in objective C. As the name suggests, the former is mutable and the latter is immutable. An example is shown below.

```
NSMutableDictionary *aMutableDictionary = [[NSMutableArray alloc]init];
[aMutableDictionary setObject:@"firstobject" forKey:@"aKey"];
NSDictionary*aImmutableDictionary= [[NSDictionary alloc]initWithObjects:[NSArray array
@"firstObject",nil] forKeys:[ NSArray arrayWithObjects:@"aKey"]];
```

iOS - First iPhone Application

Creating the First App

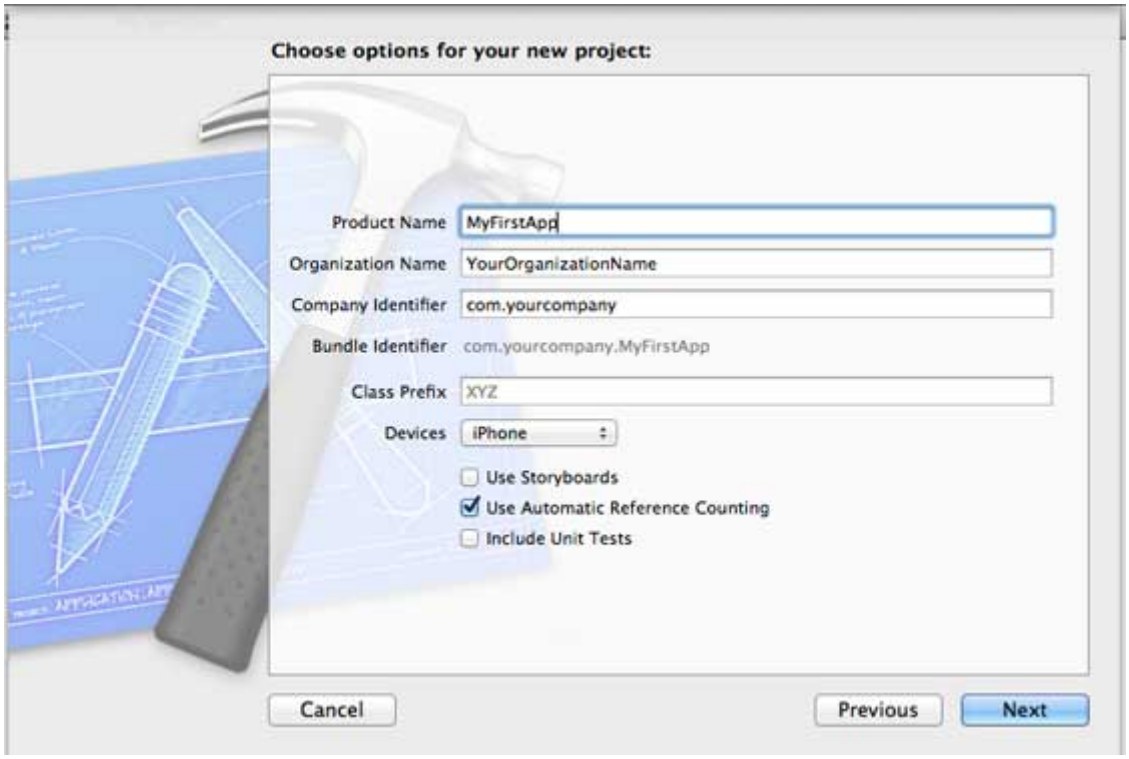
Now we are going to create a simple single view application (a blank app) that will run on the iOS simulator.

The steps are as follows.

Step 1 − Open Xcode and select **Create a new Xcode project**.

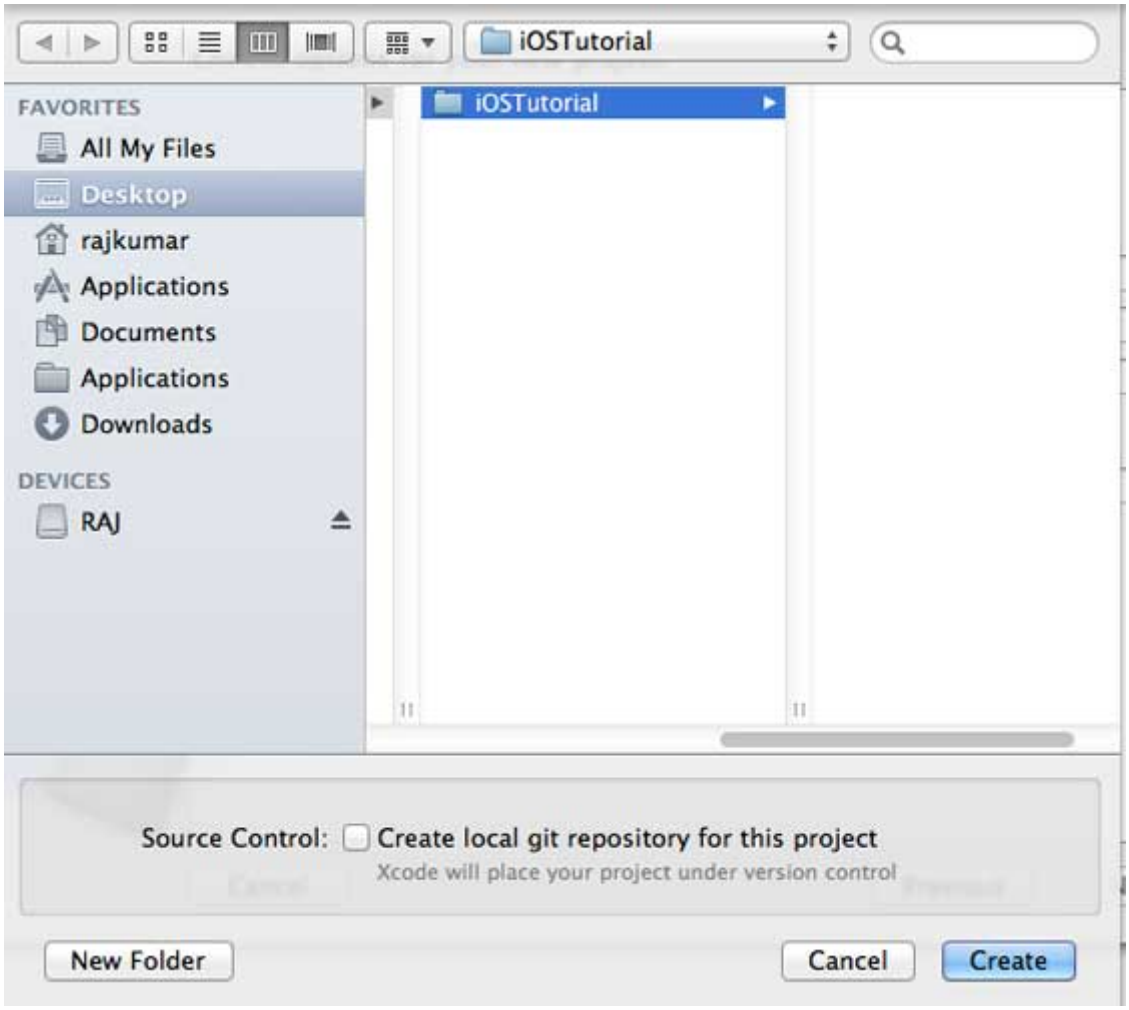
Step 2 − Select **Single View Application**.

Step 3 − Enter the product name, i.e., the name of the application, organization name, and then the company identifier.

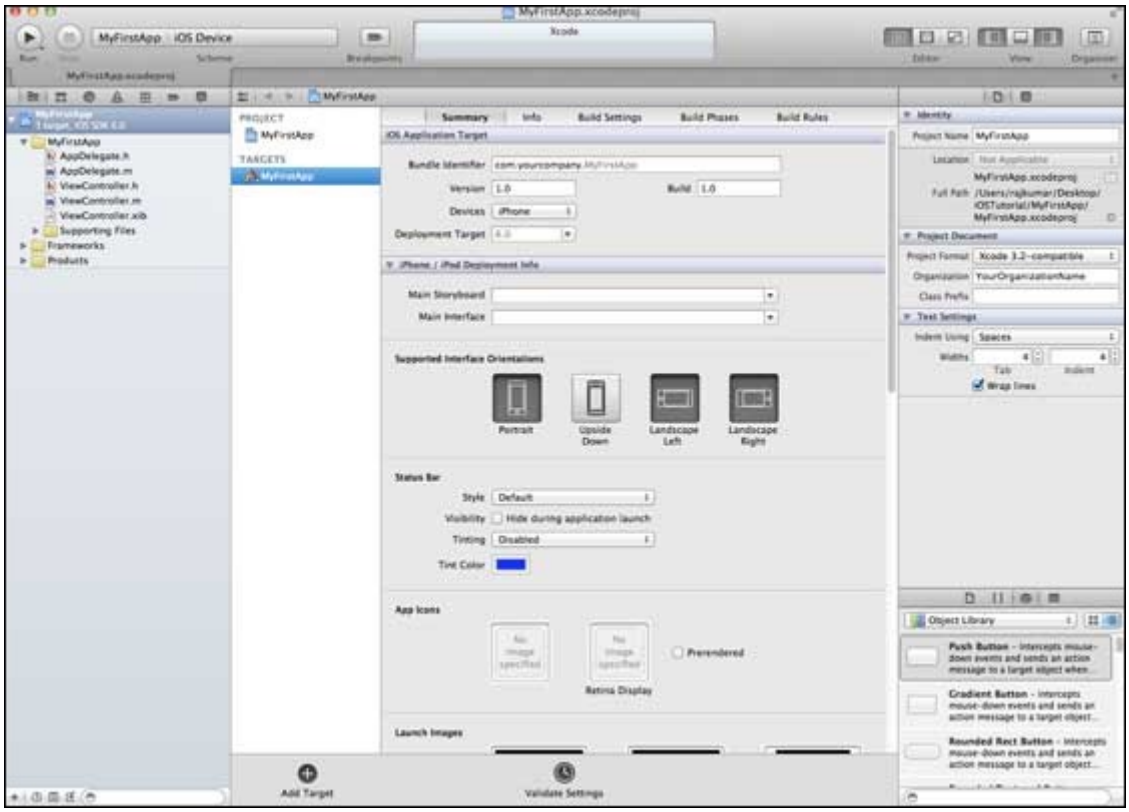


Step 4 − Ensure that **Use Automatic Reference Counting** is selected in order to automatically release the resources allocated once it goes out of scope. Click Next.

Step 5 − Select the directory for the project and select create.



Step 6 − You will see a screen as follows −

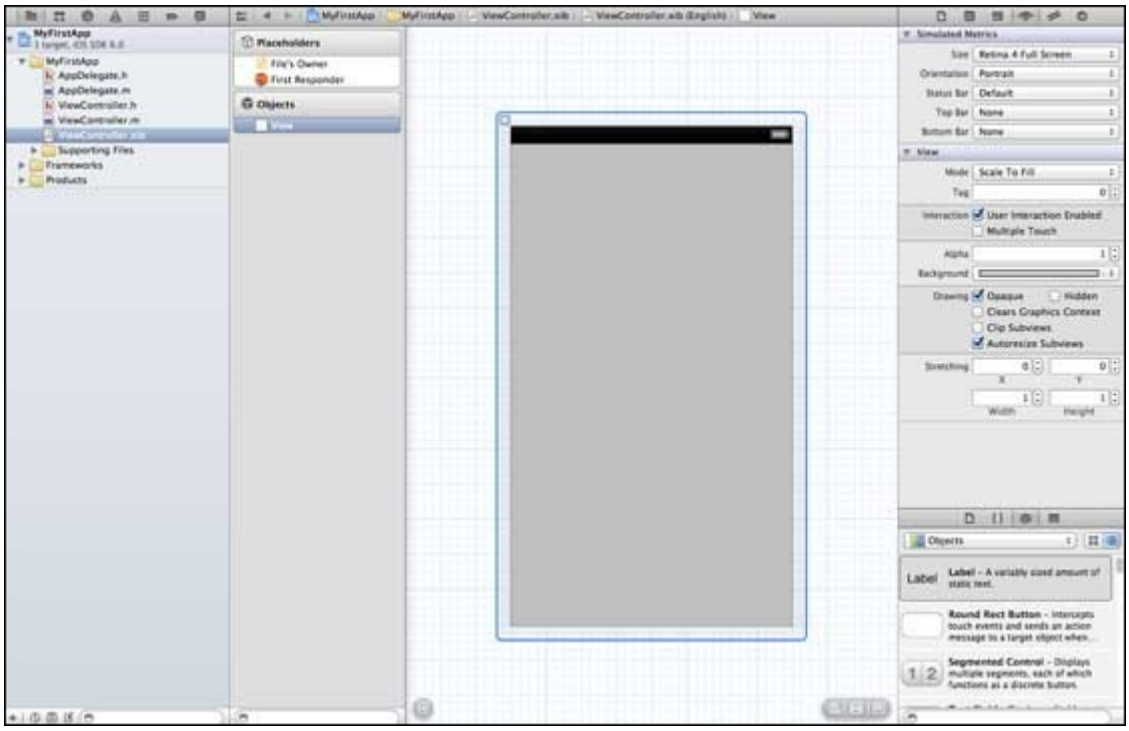


In the screen above, you will be able to select the supported orientations, build and release settings. There is a field deployment target, the device version from which we want to support, lets select 4.3, which is the minimum deployment target allowed now. For now, these are not required and let's focus on running the application.

Step 7 − Now, select iPhone simulator in the drop down near Run button and select run.

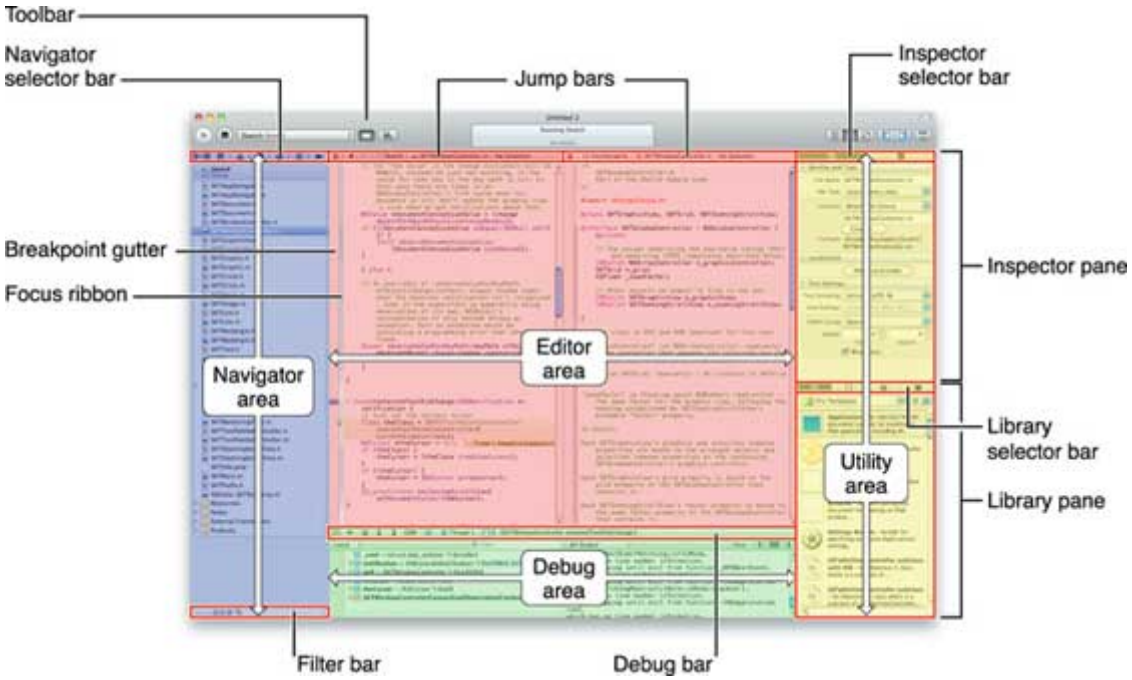
Step 8 − That's it; you have successfully run your first application. You will get an output as follows −

Now let's change the background color, just to have a start with the interface builder. Select ViewController.xib. Select background option in the right side, change the color and run.



In the above project, by default, the deployment target would have been set to iOS 6.0 and auto-layout will be enabled. To ensure that our application runs on devices that are on iOS 4.3 onwards, we have already modified the deployment target at the start of creation of this application, but we didn't disable auto-layout.

To disable auto-layout, we need to deselect the auto-layout checkbox in the file inspector of each nib, i.e., the xib files. The various sections of Xcode project IDE are given in the following figure (Courtesy: Apple Xcode 4 User documentation).



File inspector is found in the inspector selector bar as shown above and auto layout can be unchecked there. Auto layout can be used when you want to target only iOS 6 devices. Also, you'll be able to use many new features like passbook if you raise the deployment target to iOS 6. For now, let's stick to iOS 4.3 as the deployment target.

Code of the First iOS Application

You will find five different files that would have been generated for your application. They are listed as follows −

AppDelegate.h

AppDelegate.m

ViewController.h

ViewController.m

ViewController.xib

AppDelegate.h

```
// Header File that provides all UI related items.
#import <UIKit/UIKit.h>

// Forward declaration (Used when class will be defined /imported in future)
@class ViewController;

// Interface for Appdelegate
@interface AppDelegate : UIResponder <UIApplicationDelegate>

// Property window
@property (strong, nonatomic) UIWindow *window;
```

```
// Property Viewcontroller

 @property (strong, nonatomic) ViewController *viewController;
//this marks end of interface
@end
```

Important items in code −

AppDelegate inherits from UIResponder that handles iOS events.

Implements the delegate methods of UIApplicationDelegate, which provides key application events like finished launching, about to terminate and so on.

UIWindow object to manage and co-ordinate the various views on the iOS device screen. It's like the base view over which all other views are loaded. Generally there is only one window for an application.

UIViewController to handle the screen flow.

AppDelegate.m

```
// Imports the class Appdelegate's interface
import "AppDelegate.h"

// Imports the viewcontroller to be loaded
#import "ViewController.h"

// Class definition starts here
@implementation AppDelegate


// Method to intimate us that the application launched successfully
- (BOOL)application:(UIApplication *)application
 didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
   self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

   // Override point for customization after application launch.
   self.viewController = [[ViewController alloc]
   initWithNibName:@"ViewController" bundle:nil];
   self.window.rootViewController = self.viewController;
   [self.window makeKeyAndVisible];
   return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application {
   /* Use this method to release shared resources, save user data,
   invalidate timers, and store enough application state information
   to restore your application to its current state in case it is
   terminated later. If your application supports background
   execution, this method is called instead of
   applicationWillTerminate: when the user quits.*/
}

- (void)applicationWillEnterForeground:(UIApplication *)application {
   /* Called as part of the transition from the background to the
   inactive state. Here you can undo many of the changes made on
   entering the background.*/
}
```

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
   /* Restart any tasks that were paused (or not yet started) while
   the application was inactive. If the application was previously in
   the background, optionally refresh the user interface.*/
}

- (void)applicationWillTerminate:(UIApplication *)application {
   /* Called when the application is about to terminate. Save data if
   appropriate. See also applicationDidEnterBackground:. */
}

- (void)applicationWillTerminate:(UIApplication *)application {
   /* Called when the application is about to terminate. Save data if appropriate.
   See also applicationDidEnterBackground:. */
}
@end
```

Important items in code −

UIApplication delegates are defined here. All the methods defined above are UI application delegates and contains no user defined methods.

UIWindow object is allocated to hold the application allocated.

UIViewController is allocated as the window's initial view controller.

To make the window visible, makeKeyAndVisible method is called.

ViewController.h

```
#import <UIKit/UIKit.h>

// Interface for class ViewController
@interface ViewController : UIViewController

@end
```

Important items in code −

The ViewController class inherits the UIViewController, which provides the fundamental view management model for the iOS applications.

ViewController.m

```
#import "ViewController.h"

// Category, an extension of ViewController class
@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   // Do any additional setup after loading the view, typically from a nib.
}
```

```
- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}
@end
```

Important items in code −

Two methods implemented here are defined in the base class UIViewController.

Do initial setup in viewDidLoad which is called after the view loads.

didReceiveMemoryWarning method is called in case of memory warning.

iOS - Actions and Outlets

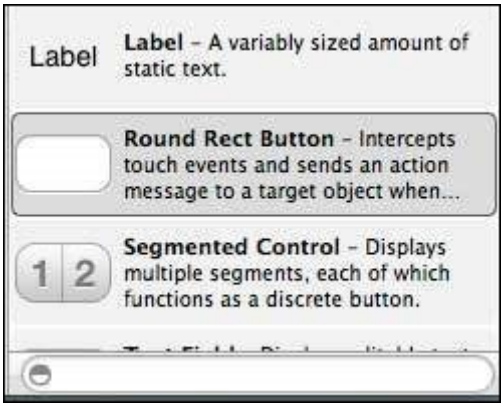
Actions and outlets in iOS are referred to as **ibActions** and **ibOutlets** respectively, where **ib** stands for interface builder. These are related to the UI elements and we will explore them after knowing visually how to implement them.

Actions and Outlets – Steps Involved

Step 1 − Let's use our First iPhone Application.

Step 2 − Select the ViewController.xib file from the files in the navigator section.

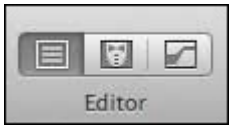
Step 3 − Now, you can select the UI elements from the library pane in the right hand side of our window, which is shown below.



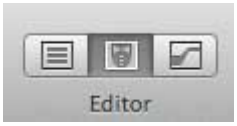
Step 4 − You can drag and drop the UI elements to our view in our interface builder.

Step 5 − Let us add a Label and Round Rect Button to our view.

Step 6 − From the Editor Selector button in the workspace toolbar found on the top right corner as shown below.

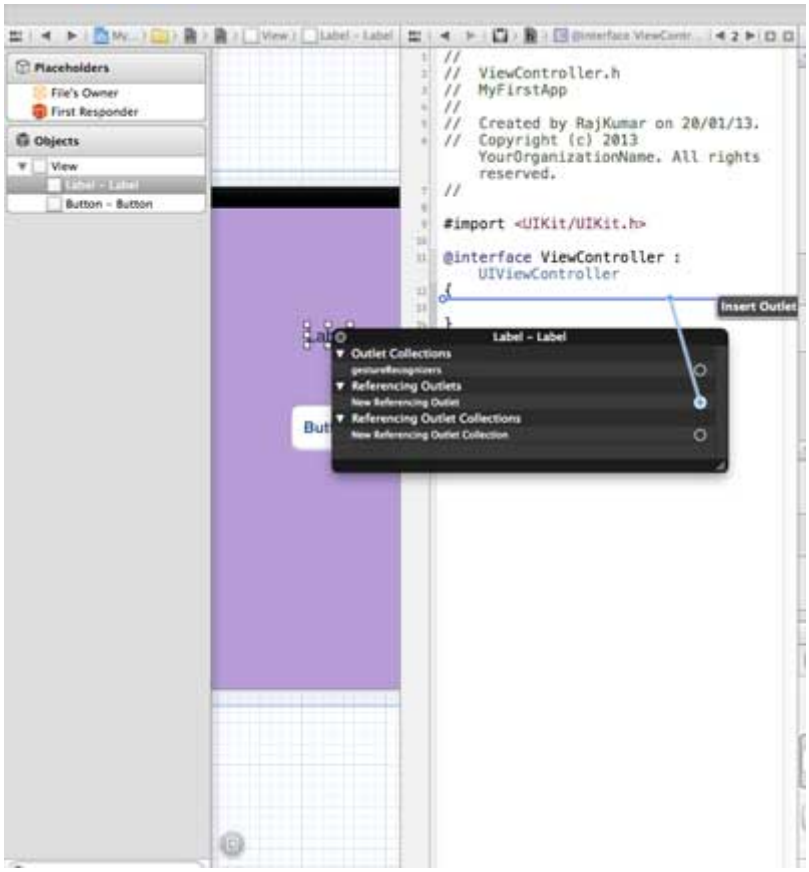


Select Assistant editor button.

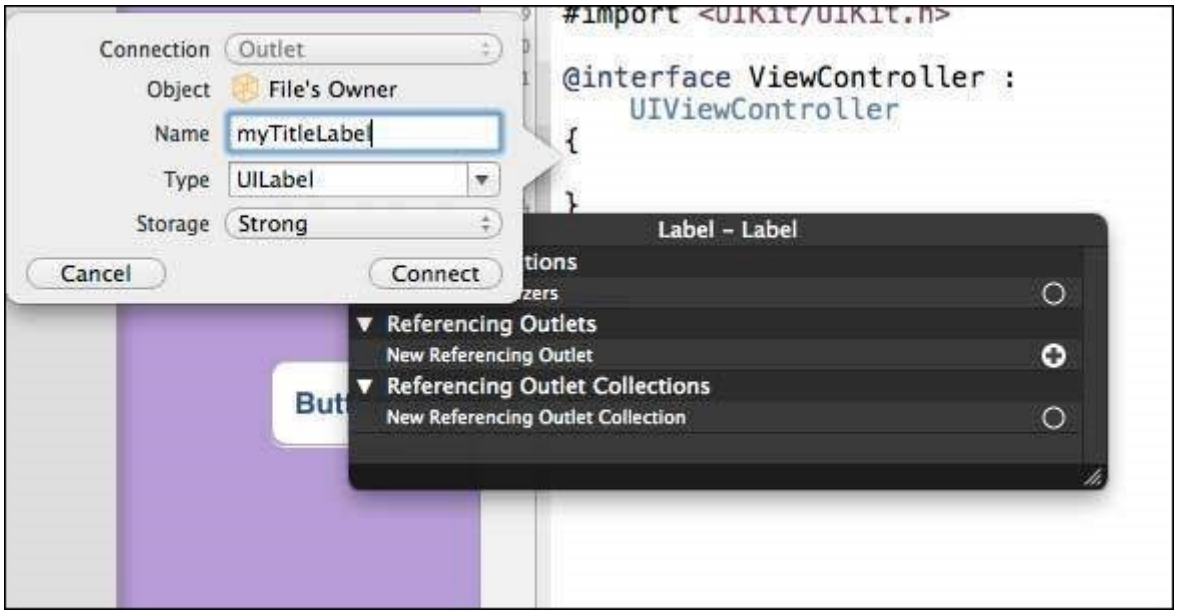


Step 7 − We will see two windows in our editor area in the center, one is ViewController.xib file and the other is ViewController.h.

Step 8 − Now, right click on the label and select, hold and drag the new referencing outlet as shown below.

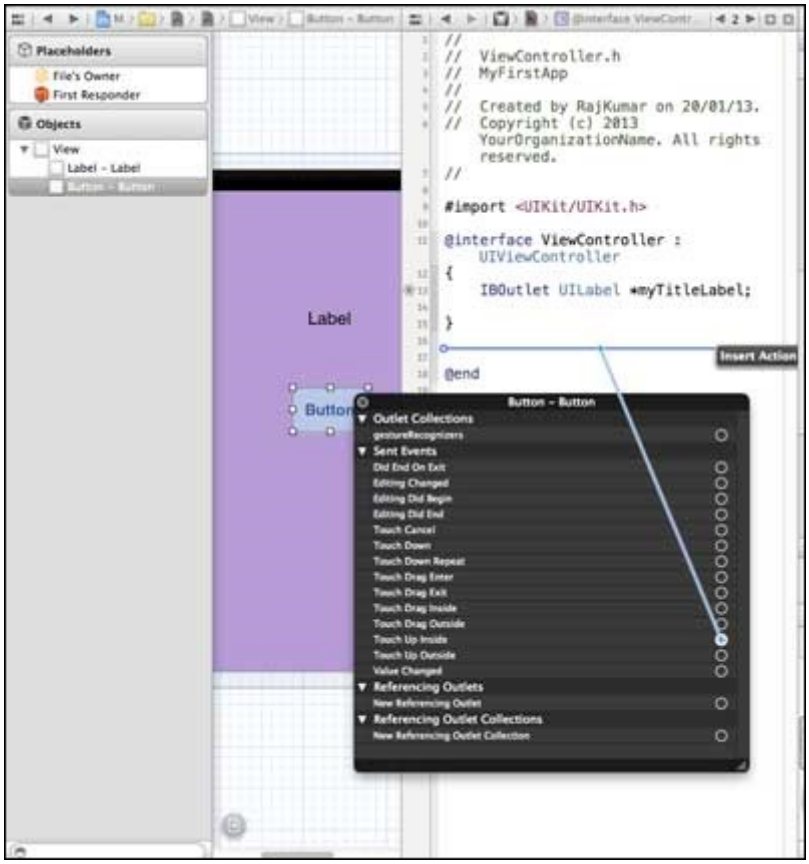


Step 9 − Drop in the ViewController.h in between the curly braces. In case there are no curly braces in the file, add the ViewController before doing this. You will find a pop-up as shown below.

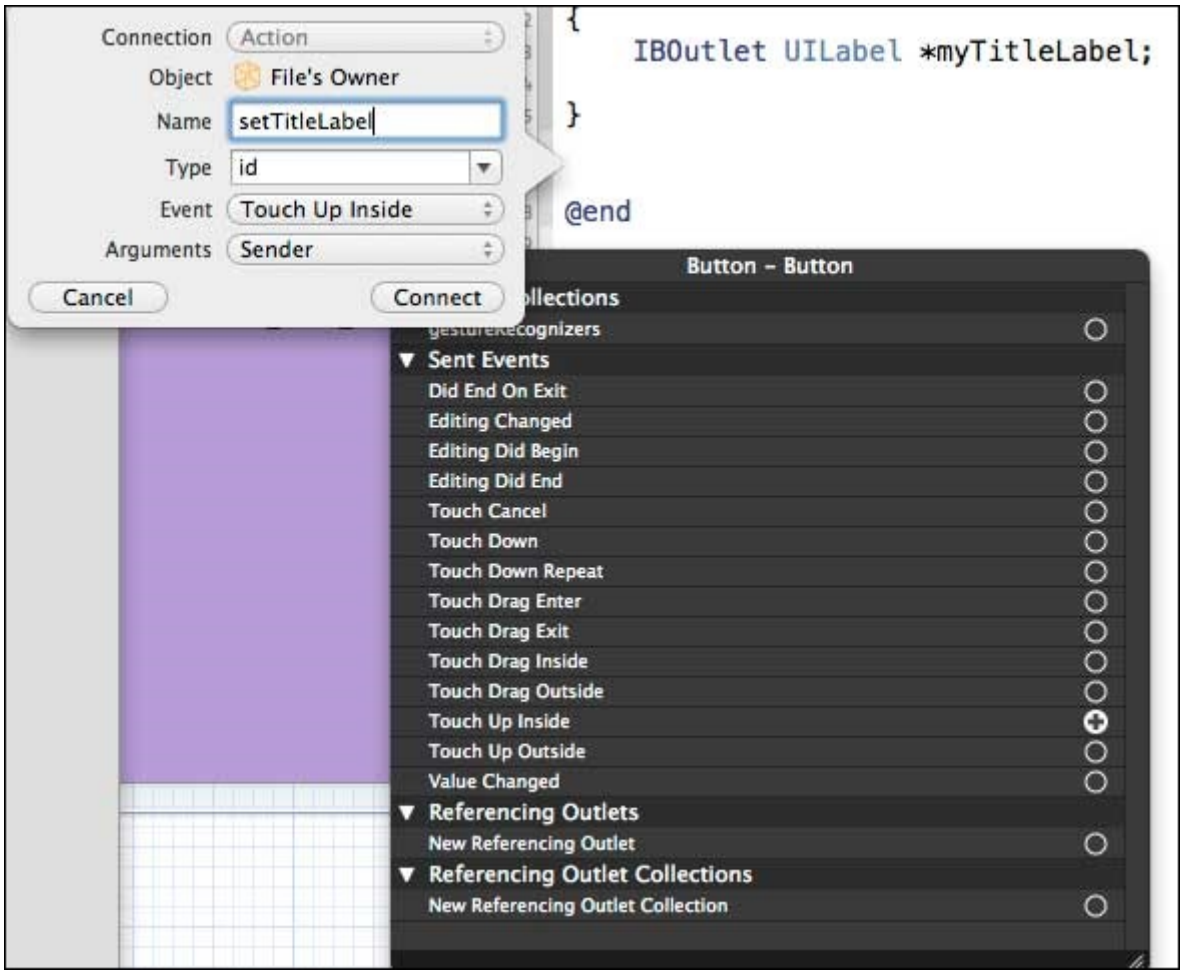


Step 10 − Type the label name for the outlet, here we have used the label myTitleLabel. Click connect and the ibOutlet will be complete.

Step 11 − Similarly, to add an action, right click the Round rect button, select touch up inside and drag it below the curly braces.



Step 12 − Drop it and name it setTitleLabel.



Step 13 − Select ViewController.m file, you'll find a method as shown below.

```
-(IBAction) setTitleLabel:(id)sender {
}
```

Step 14 − Add a statement as shown below inside the above method.

```
[myTitleLabel setText:@"Hello"];
```

Step 15 − Let us now run the program by selecting the run button. You will see the following output.

Step 16 − Now click the button.

Step 17 − The label that we created have been changed by the action on the button.

Step 18 − From the above example, we can conclude that IBOutlet creates a reference to the UIElement (here for the UILabel). Similarly, the IBAction links the UIButton with a method, which is called on the event touch up inside.

Step 19 − You can play around with actions by selecting different events while creating the action.

iOS - Delegates

Example for Delegate

Let's assume an object A calls an object B to perform an action. Once the action is complete, object A should know that B has completed the task and take necessary action. This is achieved with the help of delegates.

The key concepts in the above example are −

A is a delegate object of B.

B will have a reference of A.

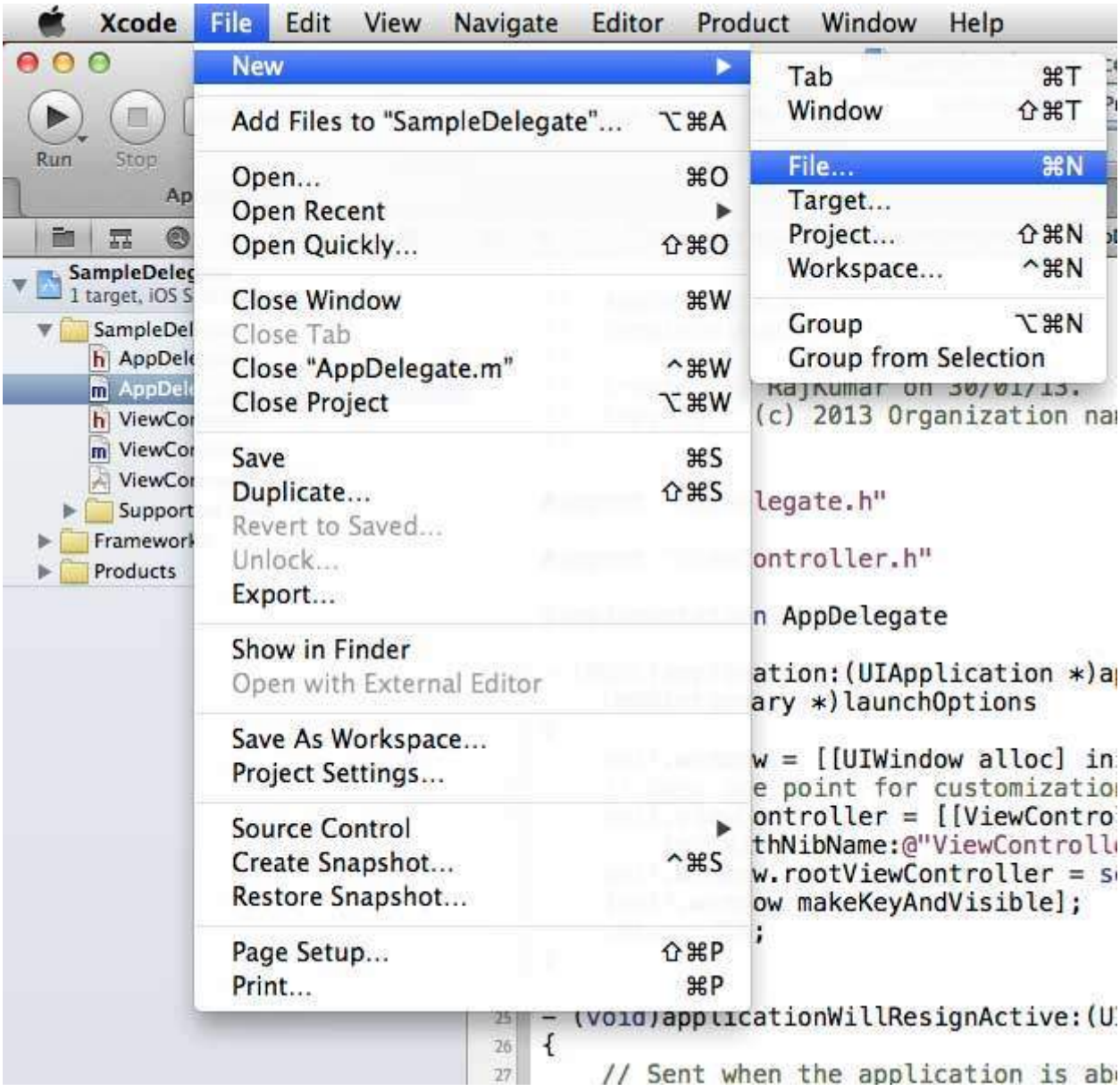
A will implement the delegate methods of B.

B will notify A through the delegate methods.

Steps in Creating a Delegate

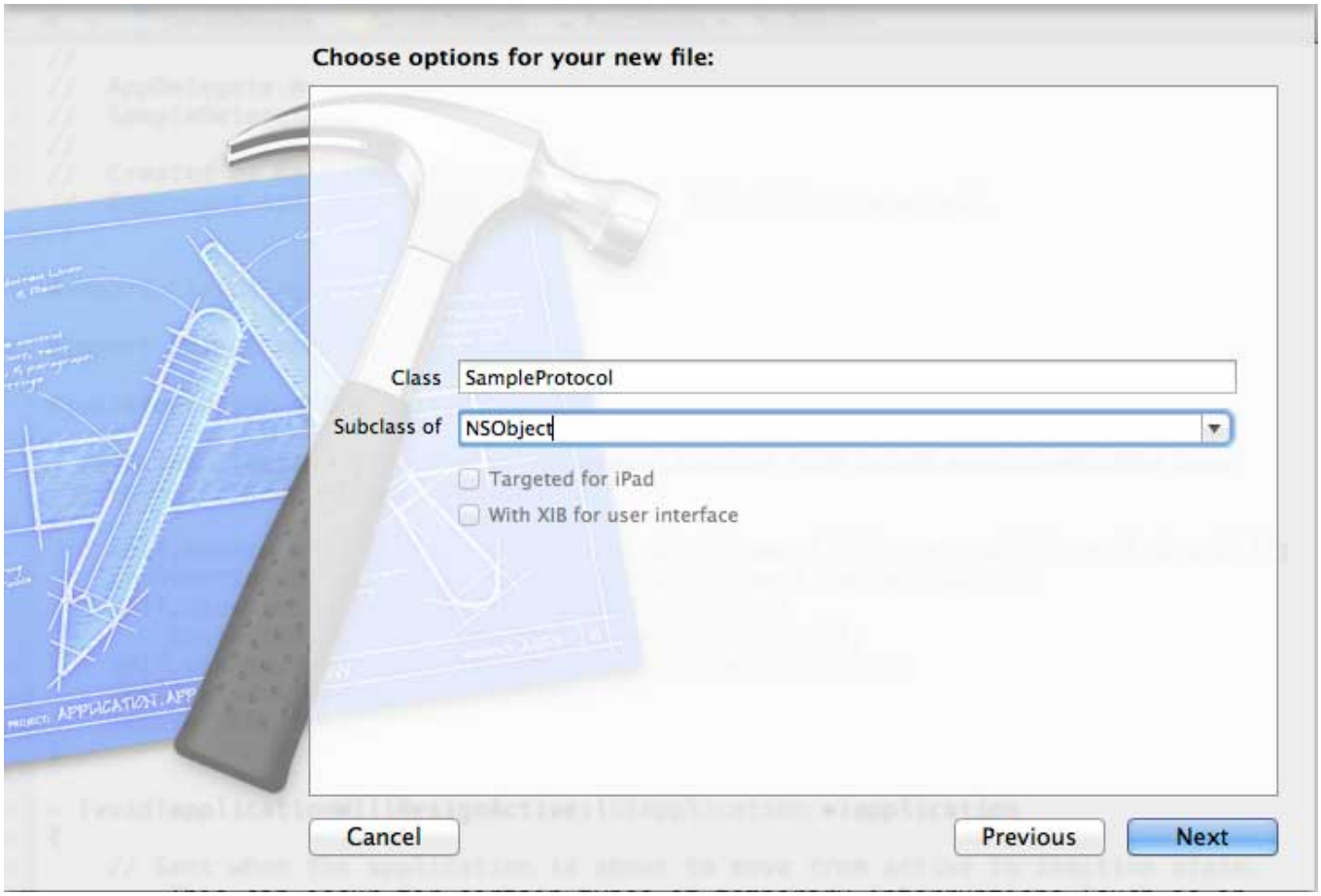
Step 1 − First, create a single view application.

Step 2 − Then select File → New → File...



Step 3 − Then select Objective C Class and click Next.

Step 4 − Give a name to the class, say, SampleProtocol with subclass as NSObject as shown below.



Step 5 − Then select create.

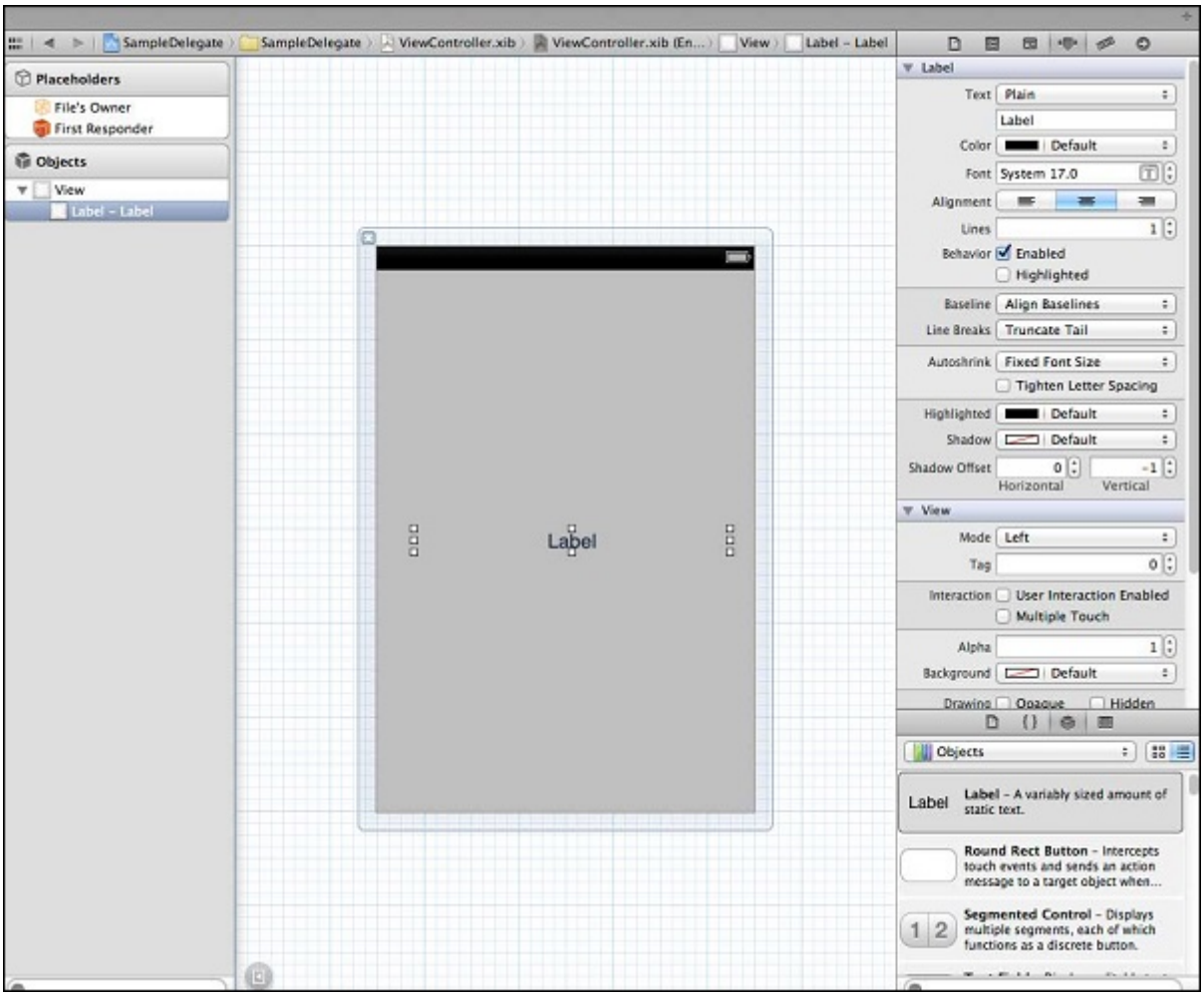
Step 6 − Add a protocol to the SampleProtocol.h file and the updated code is as follows −

```
#import <Foundation/Foundation.h>
// Protocol definition starts here
@protocol SampleProtocolDelegate <NSObject>
@required
- (void) processCompleted;
@end
// Protocol Definition ends here
@interface SampleProtocol : NSObject {
   // Delegate to respond back
   id <SampleProtocolDelegate> _delegate;
}
@property (nonatomic,strong) id delegate;

-(void)startSampleProcess; // Instance method
@end
```

Step 7 − Implement the instance method by updating the SampleProtocol.m file as shown below.

```
#import "SampleProtocol.h"

@implementation SampleProtocol

-(void)startSampleProcess {
   [NSTimer scheduledTimerWithTimeInterval:3.0 target:self.delegate
        selector:@selector(processCompleted) userInfo:nil repeats:NO];
}
@end
```

Step 8 − Add a UILabel in the ViewController.xib by dragging the label from the object library to UIView as shown below.



Step 9 − Create an IBOutlet for the label and name it as myLabel and update the code as follows to adopt SampleProtocolDelegate in ViewController.h.

```
#import <UIKit/UIKit.h>
#import "SampleProtocol.h"

@interface ViewController : UIViewController<SampleProtocolDelegate> {
   IBOutlet UILabel *myLabel;
}
@end
```

Step 10 Implement the delegate method, create object for SampleProtocol and call the startSampleProcess method. The Updated ViewController.m file is as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   SampleProtocol *sampleProtocol = [[SampleProtocol alloc]init];
   sampleProtocol.delegate = self;
```

```
   [myLabel setText:@"Processing..."];
   [sampleProtocol startSampleProcess];
   // Do any additional setup after loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

#pragma mark - Sample protocol delegate
-(void)processCompleted {
   [myLabel setText:@"Process Completed"];
}
@end
```

Step 11 We will see an output as follows. Initially the label displays "processing...", which gets updated once the delegate method is called by the SampleProtocol object.

iOS - UI Elements

What UI Elements are?

UI elements are the visual elements that we can see in our applications. Some of these elements respond to user interactions such as buttons, text fields and others are informative such as images, labels.

How to Add UI Elements?

We can add UI elements both in code and with the help of interface builder. Depending on the need we can use either one of them.

Our Focus

We'll be focussing more on adding UI elements through code in our applications. Using interface builder is simple and straight forward, we just need to drag and drop the UI elements.

Our Approach

We will create a simple iOS application and use it for explaining some of the UI elements.

Step 1 − Create a Viewbased application as we did in our First iOS application.

Step 2 − We will be only updating the ViewController.h and ViewController.m files.

Step 3 − Then we add a method to our ViewController.m file specific for creating the UI element.

Step 4 − We will call this method in our viewDidLoad method.

Step 5 − The important lines of code have been explained in the code with single line comment above those lines.

List of UI Elements

UI specific elements and their related functionalities are explained below −

Sr.No.	UI Specific Elements
1	**Text Fields** It is an UI element that enables the app to get user input.
2	**Input types - TextFields** We can set the type of input that user can give by using the keyboard property of UITextField.
3	**Buttons** It is used for handling user actions.
4	**Label** It is used for displaying static content.
5	**Toolbar** It is used if we want to manipulate something based on our current view.
6	**Status Bar** It displays the key information of device.
7	**Navigation Bar** It contains the navigation buttons of a navigation controller, which is a stack of view controllers which can be pushed and popped.
8	**Tab bar** It is generally used to switch between various subtasks, views or models within the same view.
9	**Image View** It is used to display a simple image or sequence of images.
10	**Scroll View** It is used to display content that is more than the area of screen.
11	**Table View** It is used for displaying scrollable list of data in multiple rows and sections.
12	**Split View** It is used for displaying two panes with master pane controlling the information on detail pane.
13	**Text View**

	It is used for diplaying scrollable list of text information that is optionally editable.
14	**View Transition** It explains the various view transitions between views.
15	**Pickers** It is used for displaying for selecting a specific data from a list.
16	**Switches** It is used as disable and enable for actions.
17	**Sliders** It is used to allow users to make adjustments to a value or process throughout a range of allowed values.
18	**Alerts** It is used to give important information to users.
19	**Icons** It is an image representation used for an action or depict something related to the application.

iOS - Accelerometer

Accelerometer is used for detecting the changes in the position of the device in the three directions x, y and z. We can know the current position of the device relative to the ground. For testing this example, you'll need to run it on a **device** and to doesn't work on simulator.

Accelerometer – Steps Involved

Step 1 − Create a simple **View based application**.

Step 2 − Add three labels in **ViewController.xib** and create ibOutlets naming them as xlabel, ylabel, and zlabel.

Step 3 − Update ViewController.h as follows −

```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController<UIAccelerometerDelegate> {
   IBOutlet UILabel *xlabel;
   IBOutlet UILabel *ylabel;
   IBOutlet UILabel *zlabel;
}
@end
```

Step 4 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   [[UIAccelerometer sharedAccelerometer]setDelegate:self];
   //Do any additional setup after loading the view,typically from a nib
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:
   (UIAcceleration *)acceleration {
   [xlabel setText:[NSString stringWithFormat:@"%f",acceleration.x]];
   [ylabel setText:[NSString stringWithFormat:@"%f",acceleration.y]];
   [zlabel setText:[NSString stringWithFormat:@"%f",acceleration.z]];
}
@end
```

Output

When we run the application in **iPhone** device, we'll get the following output −

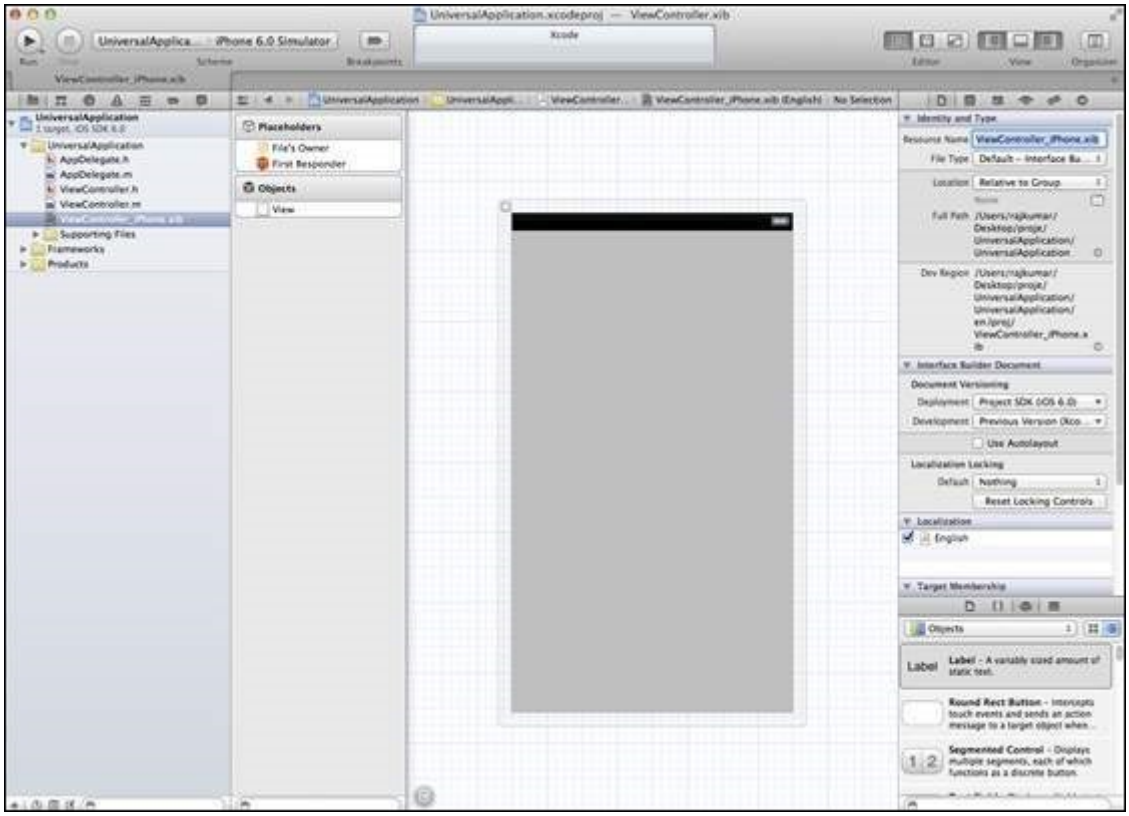
iOS - Universal Applications

A universal application is an application that is designed for both iPhone and iPad in a single binary. A universal application allows code reuse and fast updates.

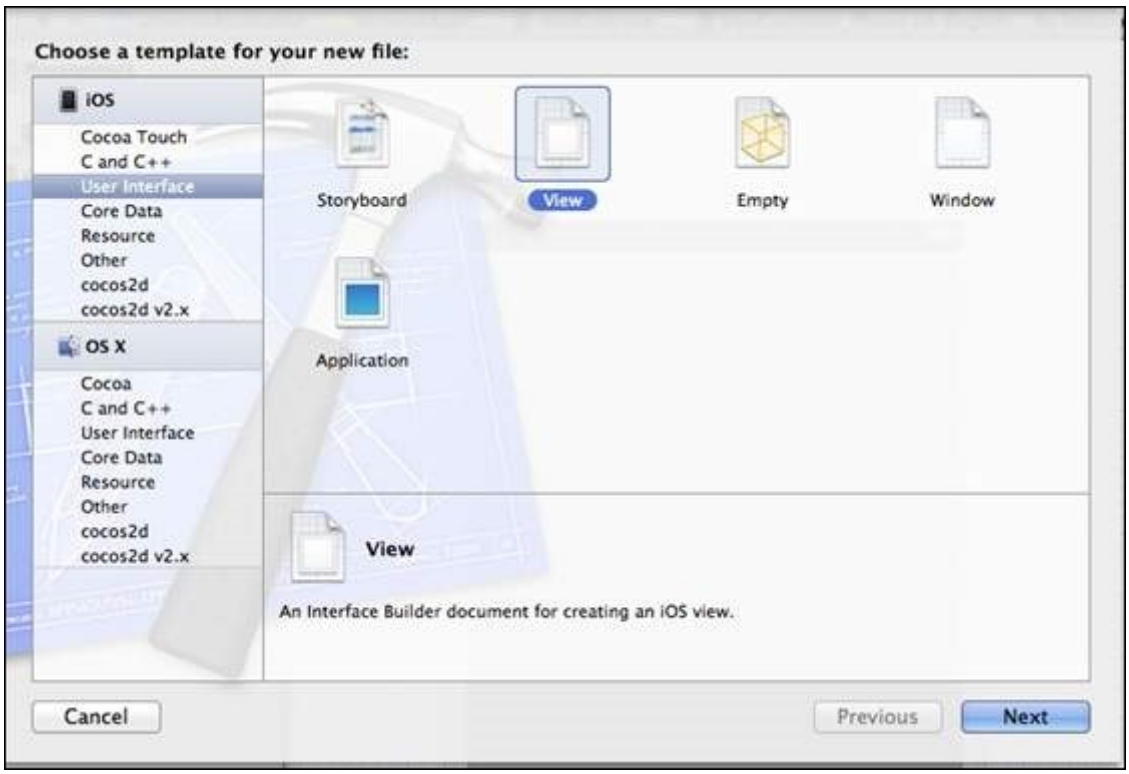
Universal Application – Steps Involved

Step 1 − Create a simple **View based application**.

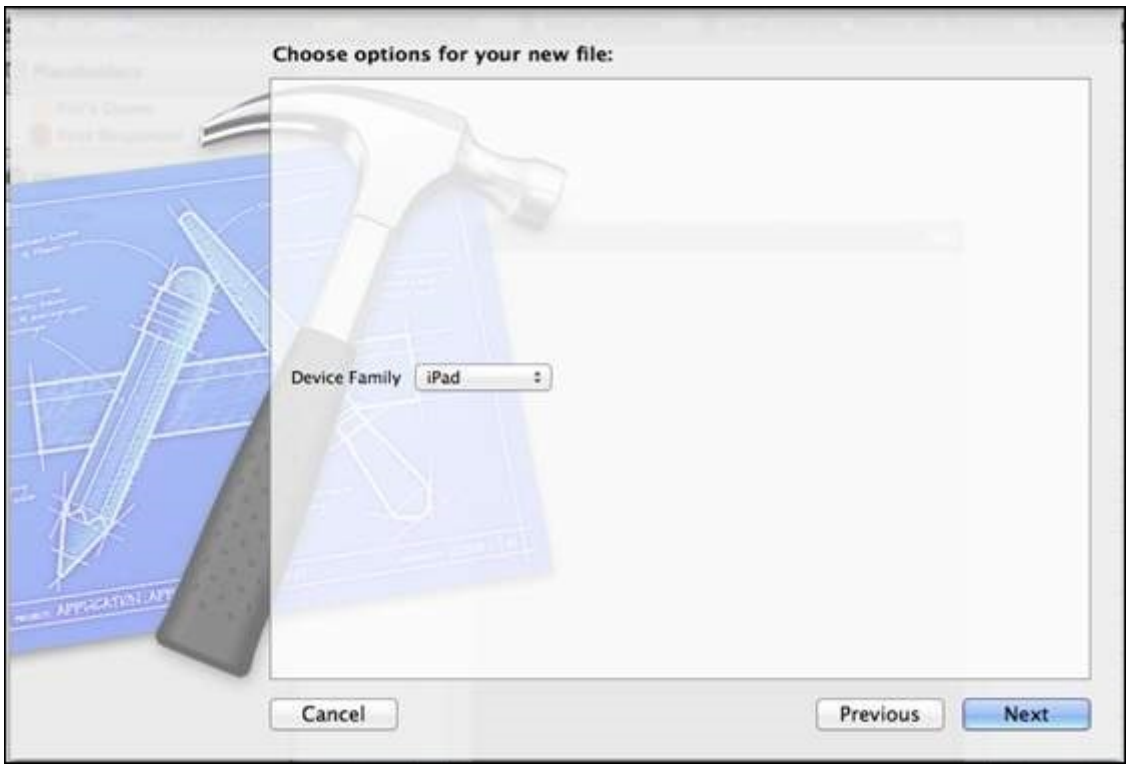
Step 2 − Change the File name **ViewController.xib** file to **ViewController_iPhone.xib** as shown below in the file inspector in the right hand side.



Step 3 − Select File → New → File... then select the subsection **"User Interface"** and select **View**. Click Next.



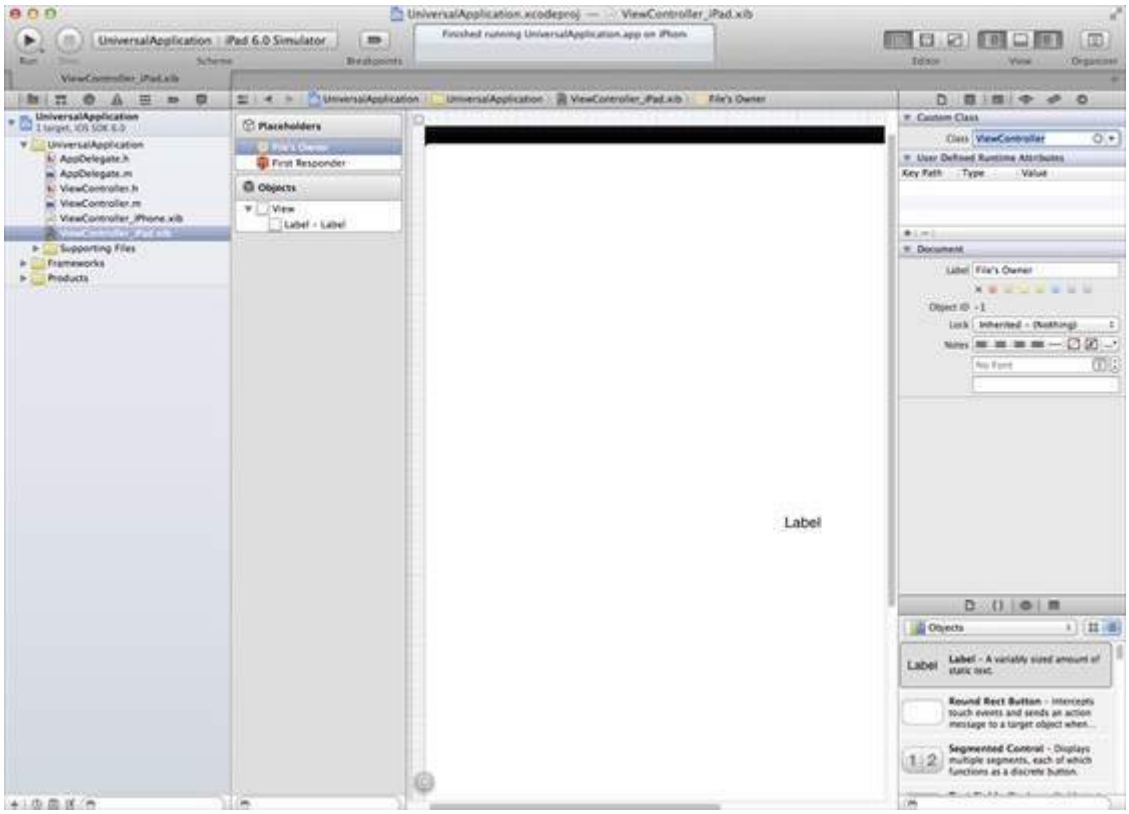
Step 4 − Select the device family as **iPad** and click next.



Step 5 − Save the file as **ViewController_iPad.xib** and select Create.

Step 6 − Add a label in the center of the screen in both **ViewController_iPhone.xib** and **ViewController_iPad.xib**.

Step 7 − In **ViewController_iPad.xib**, select the **identity inspector** and set the custom class as **ViewController**.

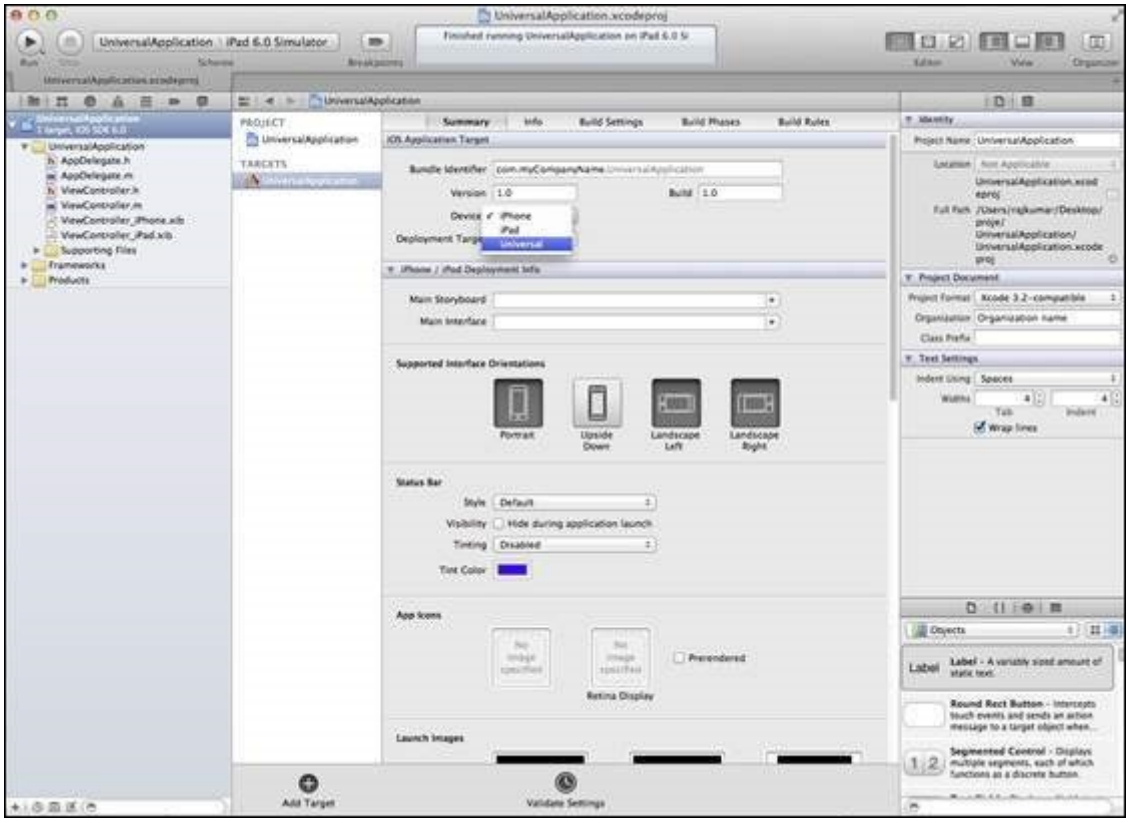


Step 8 − Update the application:DidFinishLaunching:withOptions method in AppDelegate.m as follows −

```
- (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
```

```
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen
    mainScreen] bounds]];

    // Override point for customization after application launch.
    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone) {
        self.viewController = [[ViewController alloc]
        initWithNibName:@"ViewController_iPhone" bundle:nil];
    } else {
        self.viewController = [[ViewController alloc] initWithNibName:
        @"ViewController_iPad" bundle:nil];
    }
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

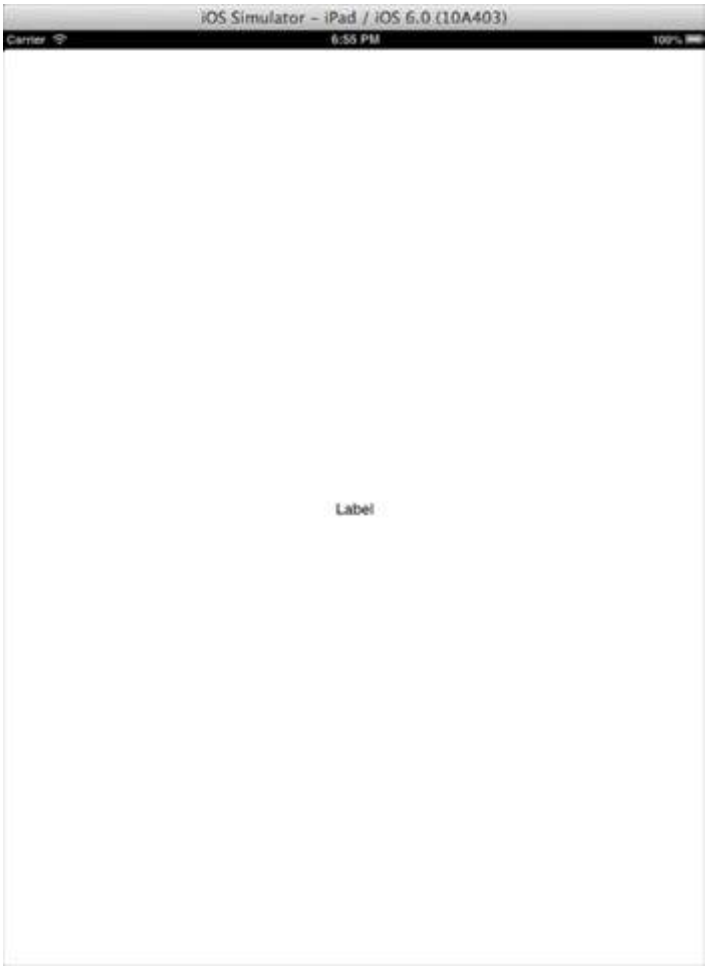
Step 9 − Update the devices in project summary to **Universal** as shown below −



Output

When we run the application, we'll get the following output −

When we run the application in iPad simulator, we'll get the following output −



iOS - Camera Management

Camera is one of the common features in a mobile device. It is possible for us to take pictures with the camera and use it in our application and it is quite simple too.

Camera Management – Steps Involved

Step 1 − Create a simple **View based application**.

Step 2 − Add a **button** in **ViewController.xib** and create IBAction for the button.

Step 3 − Add an **image view** and create IBOutlet naming it as imageView.

Step 4 − Update **ViewController.h** as follows −

```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController<UIImagePickerControllerDelegate> {
   UIImagePickerController *imagePicker;
   IBOutlet UIImageView *imageView;
}

- (IBAction)showCamera:(id)sender;
@end
```

Step 5 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

- (IBAction)showCamera:(id)sender {
   imagePicker.allowsEditing = YES;
   if ([UIImagePickerController isSourceTypeAvailable:
   UIImagePickerControllerSourceTypeCamera]) {
      imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
   } else {
      imagePicker.sourceType =
      UIImagePickerControllerSourceTypePhotoLibrary;
   }
   [self presentModalViewController:imagePicker animated:YES];
}

-(void)imagePickerController:(UIImagePickerController *)picker
   didFinishPickingMediaWithInfo:(NSDictionary *)info {
      UIImage *image = [info objectForKey:UIImagePickerControllerEditedImage];
```

```
      if (image == nil) {
         image = [info objectForKey:UIImagePickerControllerOriginalImage];
      }
   imageView.image = image;
}

-(void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {
   [self dismissModalViewControllerAnimated:YES];
}
@end
```

Output

When we run the application and click show camera button, we'll get the following output −

Once we take a picture, we can edit the picture, i.e., move and scale as shown below −

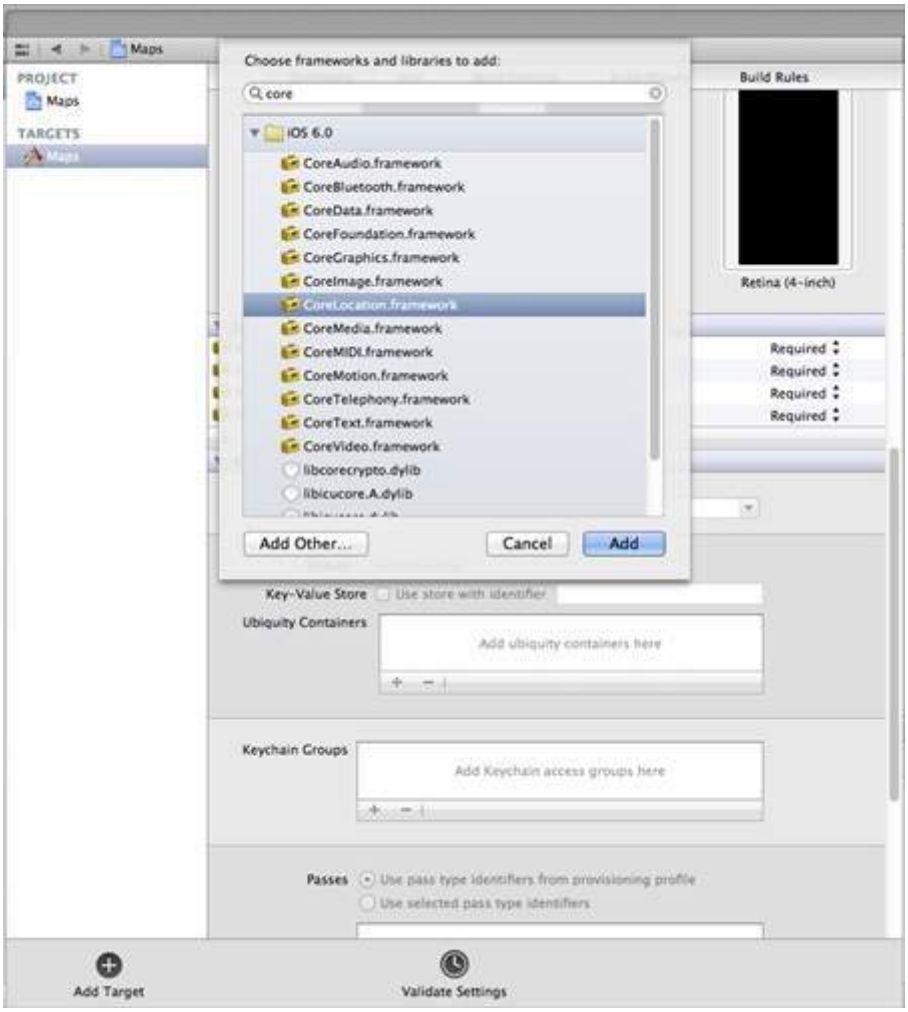
iOS - Location Handling

We can easily locate the user's current location in iOS, provided the user allows the application to access the information with the help of the core location framework.

Location Handling – Steps Involved

Step 1 − Create a simple View based application.

Step 2 − Select your project file, then select targets and then add CoreLocation.framework as shown below −



Step 3 − Add two labels in **ViewController.xib** and create ibOutlets naming the labels as **latitudeLabel** and **longitudeLabel** respectively.

Step 4 − Create a new file by selecting File → New → File... → select **Objective C class** and click next.

Step 5 − Name the class as **LocationHandler** with **"sub class of"** as NSObject.

Step 6 − Select create.

Step 7 − Update **LocationHandler.h** as follows −

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>

@protocol LocationHandlerDelegate <NSObject>

@required
-(void) didUpdateToLocation:(CLLocation*)newLocation
 fromLocation:(CLLocation*)oldLocation;
@end

@interface LocationHandler : NSObject<CLLocationManagerDelegate> {
   CLLocationManager *locationManager;
}
@property(nonatomic,strong) id<LocationHandlerDelegate> delegate;

+(id)getSharedInstance;
-(void)startUpdating;
-(void) stopUpdating;
```

```
@end
```

Step 8 − Update **LocationHandler.m** as follows −

```
#import "LocationHandler.h"
static LocationHandler *DefaultManager = nil;

@interface LocationHandler()

-(void)initiate;

@end

@implementation LocationHandler

+(id)getSharedInstance{
   if (!DefaultManager) {
      DefaultManager = [[self allocWithZone:NULL]init];
      [DefaultManager initiate];
   }
   return DefaultManager;
}

-(void)initiate {
   locationManager = [[CLLocationManager alloc]init];
   locationManager.delegate = self;
}

-(void)startUpdating{
    [locationManager startUpdatingLocation];
}

-(void) stopUpdating {
   [locationManager stopUpdatingLocation];
}

-(void)locationManager:(CLLocationManager *)manager didUpdateToLocation:
   (CLLocation *)newLocation fromLocation:(CLLocation *)oldLocation {
   if ([self.delegate respondsToSelector:@selector
   (didUpdateToLocation:fromLocation:)]) {
      [self.delegate didUpdateToLocation:oldLocation
      fromLocation:newLocation];
   }
}
@end
```

Step 9 − Update **ViewController.h** as follows where we have implemented the **LocationHandler delegate** and create two ibOutlets −

```
#import <UIKit/UIKit.h>
#import "LocationHandler.h"
@interface ViewController : UIViewController<LocationHandlerDelegate> {
   IBOutlet UILabel *latitudeLabel;
   IBOutlet UILabel *longitudeLabel;
}
@end
```

Step 10 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   [[LocationHandler getSharedInstance]setDelegate:self];
   [[LocationHandler getSharedInstance]startUpdating];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(void)didUpdateToLocation:(CLLocation *)newLocation
 fromLocation:(CLLocation *)oldLocation {
   [latitudeLabel setText:[NSString stringWithFormat:
   @"Latitude: %f",newLocation.coordinate.latitude]];
   [longitudeLabel setText:[NSString stringWithFormat:
   @"Longitude: %f",newLocation.coordinate.longitude]];

}
@end
```

Output

When we run the application, we'll get the following output −

iOS - SQLite Database

SQLite can be used in iOS for handling data. It uses sqlite queries, which makes it easier for those who know SQL.

Steps Involved

Step 1 − Create a simple **View based application**.

Step 2 − Select your project file, then select targets and then add **libsqlite3.dylib** library in choose frameworks.

Step 3 − Create a new file by selecting File→ New → File... → select **Objective C class** and click next.

Step 4 − Name the class as **DBManager** with **"sub class of"** as NSObject.

Step 5 − Select create.

Step 6 − Update **DBManager.h** as follows −

```
#import <Foundation/Foundation.h>
#import <sqlite3.h>

@interface DBManager : NSObject {
   NSString *databasePath;
}
```

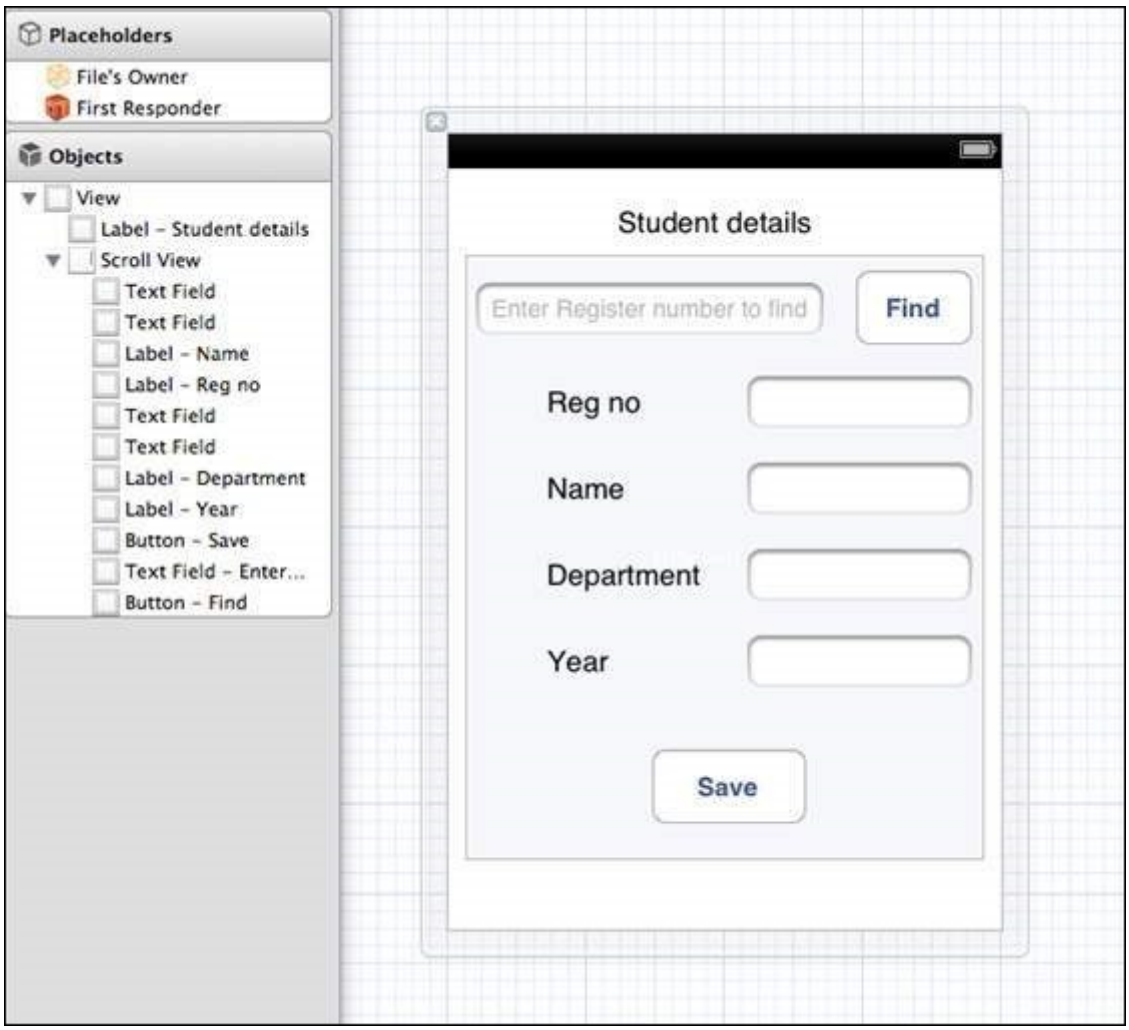
```
+(DBManager*)getSharedInstance;
-(BOOL)createDB;
-(BOOL) saveData:(NSString*)registerNumber name:(NSString*)name
   department:(NSString*)department year:(NSString*)year;
-(NSArray*) findByRegisterNumber:(NSString*)registerNumber;

@end
```

Step 7 − Update **DBManager.m** as follows −

```
#import "DBManager.h"
static DBManager *sharedInstance = nil;
static sqlite3 *database = nil;
static sqlite3_stmt *statement = nil;

@implementation DBManager

+(DBManager*)getSharedInstance {
   if (!sharedInstance) {
      sharedInstance = [[super allocWithZone:NULL]init];
      [sharedInstance createDB];
   }
   return sharedInstance;
}

-(BOOL)createDB {
   NSString *docsDir;
   NSArray *dirPaths;

   // Get the documents directory
   dirPaths = NSSearchPathForDirectoriesInDomains
   (NSDocumentDirectory, NSUserDomainMask, YES);
   docsDir = dirPaths[0];

   // Build the path to the database file
   databasePath = [[NSString alloc] initWithString:
   [docsDir stringByAppendingPathComponent: @"student.db"]];
   BOOL isSuccess = YES;
   NSFileManager *filemgr = [NSFileManager defaultManager];
   if ([filemgr fileExistsAtPath: databasePath ] == NO) {
      const char *dbpath = [databasePath UTF8String];
      if (sqlite3_open(dbpath, &database) == SQLITE_OK) {
         char *errMsg;
         const char *sql_stmt =
         "create table if not exists studentsDetail (regno integer
         primary key, name text, department text, year text)";

         if (sqlite3_exec(database, sql_stmt, NULL, NULL, &errMsg) != SQLITE_OK) {
            isSuccess = NO;
            NSLog(@"Failed to create table");
         }
         sqlite3_close(database);
         return  isSuccess;
      } else {
         isSuccess = NO;
         NSLog(@"Failed to open/create database");
      }
   }
   return isSuccess;
}

- (BOOL) saveData:(NSString*)registerNumber name:(NSString*)name
```

```
   department:(NSString*)department year:(NSString*)year; {
   const char *dbpath = [databasePath UTF8String];

   if (sqlite3_open(dbpath, &database) == SQLITE_OK) {
      NSString *insertSQL = [NSString stringWithFormat:@"insert into
      studentsDetail (regno,name, department, year) values
      (\"%d\",\"%@\", \"%@\", \"%@\")",[registerNumber integerValue],
      name, department, year];
      const char *insert_stmt = [insertSQL UTF8String];
      sqlite3_prepare_v2(database, insert_stmt,-1, &statement, NULL);

      if (sqlite3_step(statement) == SQLITE_DONE) {
         return YES;
      } else {
         return NO;
      }
      sqlite3_reset(statement);
   }
   return NO;
}

- (NSArray*) findByRegisterNumber:(NSString*)registerNumber {
   const char *dbpath = [databasePath UTF8String];

   if (sqlite3_open(dbpath, &database) == SQLITE_OK) {
      NSString *querySQL = [NSString stringWithFormat:
      @"select name, department, year from studentsDetail where
      regno=\"%@\"",registerNumber];
      const char *query_stmt = [querySQL UTF8String];
      NSMutableArray *resultArray = [[NSMutableArray alloc]init];

      if (sqlite3_prepare_v2(database,
         query_stmt, -1, &statement, NULL) == SQLITE_OK) {

         if (sqlite3_step(statement) == SQLITE_ROW) {
            NSString *name = [[NSString alloc] initWithUTF8String:
            (const char *) sqlite3_column_text(statement, 0)];
            [resultArray addObject:name];
            NSString *department = [[NSString alloc] initWithUTF8String:
            (const char *) sqlite3_column_text(statement, 1)];
            [resultArray addObject:department];
            NSString *year = [[NSString alloc]initWithUTF8String:
            (const char *) sqlite3_column_text(statement, 2)];
            [resultArray addObject:year];
            return resultArray;
         } else {
            NSLog(@"Not found");
            return nil;
         }
         sqlite3_reset(statement);
      }
   }
   return nil;
}
```

Step 8 − Update **ViewController.xib** file as follows −



Step 9 − Create IBOutlets for the above text fields.

Step 10 − Create IBAction for the above buttons.

Step 11 − Update **ViewController.h** as follows −

```
#import <UIKit/UIKit.h>
#import "DBManager.h"

@interface ViewController : UIViewController<UITextFieldDelegate> {
   IBOutlet UITextField *regNoTextField;
   IBOutlet UITextField *nameTextField;
   IBOutlet UITextField *departmentTextField;
   IBOutlet UITextField *yearTextField;
   IBOutlet UITextField *findByRegisterNumberTextField;
   IBOutlet UIScrollView *myScrollView;
}

-(IBAction)saveData:(id)sender;
-(IBAction)findData:(id)sender;
@end
```

Step 12 − Update **ViewController.m** as follows −

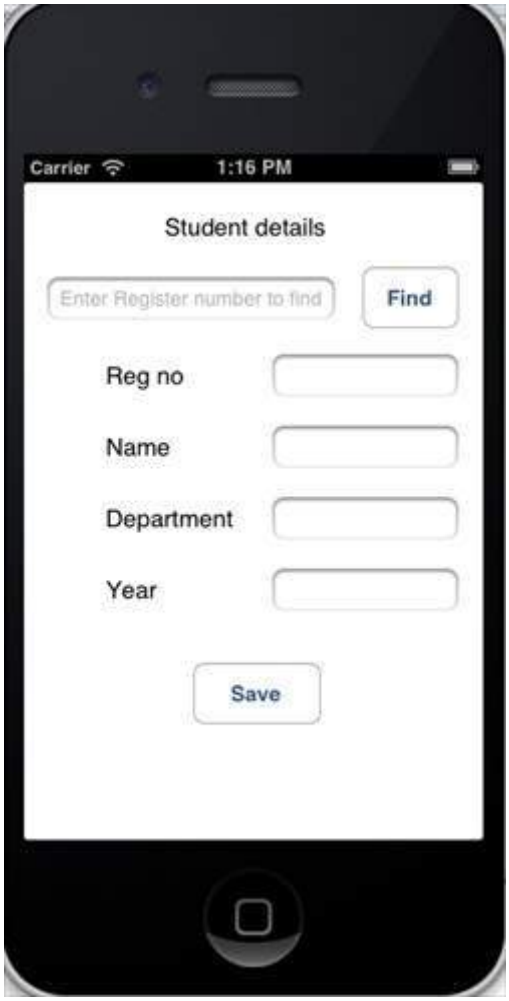
```
#import "ViewController.h"

@interface ViewController ()

@end
```

```
@implementation ViewController

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)
   nibBundleOrNil {
   self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];

   if (self) {
      // Custom initialization
   }
   return self;
}

- (void)viewDidLoad {
   [super viewDidLoad];
   // Do any additional setup after loading the view from its nib.
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(IBAction)saveData:(id)sender {
   BOOL success = NO;
   NSString *alertString = @"Data Insertion failed";

   if (regNoTextField.text.length>0 &&nameTextField.text.length>0 &&
      departmentTextField.text.length>0 &&yearTextField.text.length>0 ) {
      success = [[DBManager getSharedInstance]saveData:
      regNoTextField.text name:nameTextField.text department:
      departmentTextField.text year:yearTextField.text];
   } else {
      alertString = @"Enter all fields";
   }

   if (success == NO) {
      UIAlertView *alert = [[UIAlertView alloc]initWithTitle:
      alertString message:nil
      delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
      [alert show];
   }
}

-(IBAction)findData:(id)sender {
   NSArray *data = [[DBManager getSharedInstance]findByRegisterNumber:
   findByRegisterNumberTextField.text];

   if (data == nil) {
      UIAlertView *alert = [[UIAlertView alloc]initWithTitle:
      @"Data not found" message:nil delegate:nil cancelButtonTitle:
      @"OK" otherButtonTitles:nil];
      [alert show];
      regNoTextField.text = @"";
      nameTextField.text =@"";
      departmentTextField.text = @"";
      yearTextField.text =@"";
   } else {
      regNoTextField.text = findByRegisterNumberTextField.text;
      nameTextField.text =[data objectAtIndex:0];
      departmentTextField.text = [data objectAtIndex:1];
      yearTextField.text =[data objectAtIndex:2];
   }
}
```

```
#pragma mark - Text field delegate
-(void)textFieldDidBeginEditing:(UITextField *)textField {
   [myScrollView setFrame:CGRectMake(10, 50, 300, 200)];
   [myScrollView setContentSize:CGSizeMake(300, 350)];
}

-(void)textFieldDidEndEditing:(UITextField *)textField {
   [myScrollView setFrame:CGRectMake(10, 50, 300, 350)];

}

-(BOOL) textFieldShouldReturn:(UITextField *)textField {
   [textField resignFirstResponder];
   return YES;
}
@end
```

Output

When we run the application, we'll get the following output where we can add and find the student details −



iOS - Sending Email

We can send emails using the Email application of iOS device.

Steps Involved

Step 1 − Create a simple **View based application**.

Step 2 − Select your project file, then select targets and then add **MessageUI.framework**.

Step 3 − Add a button in **ViewController.xib** and create an action for sending email.

Step 4 − Update **ViewController.h** as follows −

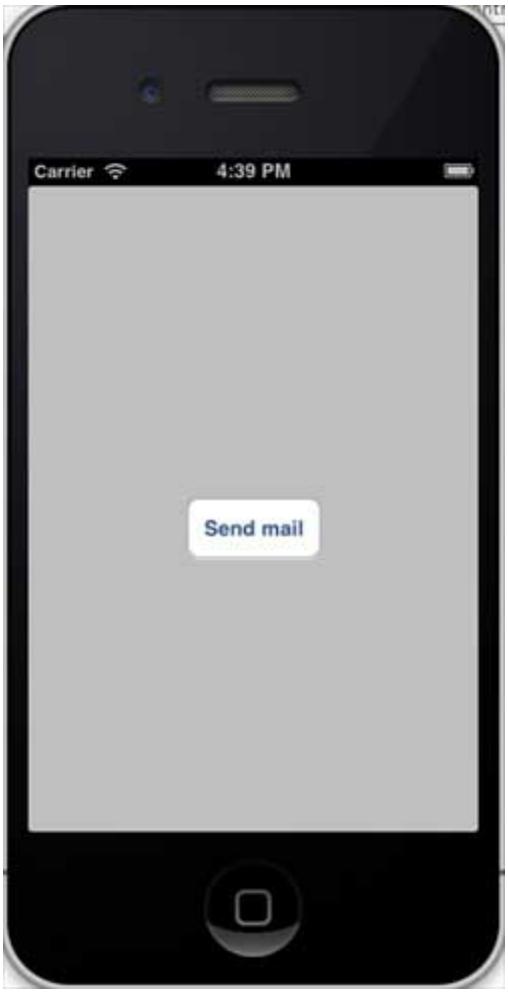
```
#import <UIKit/UIKit.h>
#import <MessageUI/MessageUI.h>

@interface ViewController : UIViewController<MFMailComposeViewControllerDelegate> {
   MFMailComposeViewController *mailComposer;
}

-(IBAction)sendMail:(id)sender;

@end
```

Step 5 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(void)sendMail:(id)sender {
   mailComposer = [[MFMailComposeViewController alloc]init];
   mailComposer.mailComposeDelegate = self;
   [mailComposer setSubject:@"Test mail"];
   [mailComposer setMessageBody:@"Testing message
   for the test mail" isHTML:NO];
   [self presentModalViewController:mailComposer animated:YES];
}

#pragma mark - mail compose delegate
-(void)mailComposeController:(MFMailComposeViewController *)controller
 didFinishWithResult:(MFMailComposeResult)result error:(NSError *)error{
   if (result) {
      NSLog(@"Result : %d",result);
   }

   if (error) {
      NSLog(@"Error : %@",error);
   }

   [self dismissModalViewControllerAnimated:YES];
}
@end
```

Output

When we run the application, we'll get the following output −



On clicking Send Email, we will get the following output −

iOS - Audio and Video

Audio and video is quite common in the latest devices. It is supported in iOS with the help of **AVFoundation.framework** and **MediaPlayer.framework** respectively.

Steps Involved

Step 1 − Create a simple **View based application**.

Step 2 − Select your project file, select targets, and then we should add **AVFoundation.framework** and **MediaPlayer.framework**.

Step 3 − Add two buttons in ViewController.xib and create an action for playing audio and video respectively.

Step 4 − Update **ViewController.h** as follows −

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import <MediaPlayer/MediaPlayer.h>

@interface ViewController : UIViewController {
   AVAudioPlayer *audioPlayer;
   MPMoviePlayerViewController *moviePlayer;
}
-(IBAction)playAudio:(id)sender;
-(IBAction)playVideo:(id)sender;
@end
```

Step 5 − Update **ViewController.m** as follows −

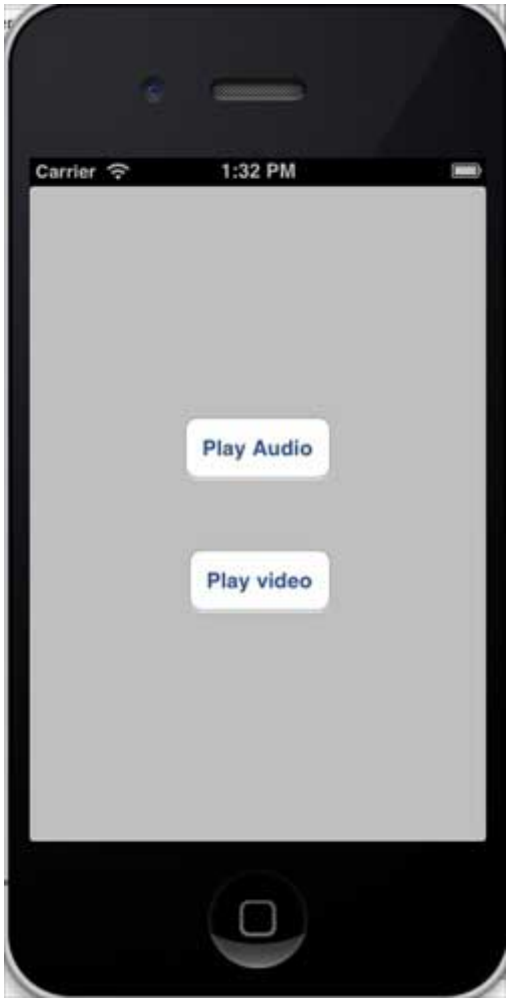
```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(IBAction)playAudio:(id)sender {
   NSString *path = [[NSBundle mainBundle]
   pathForResource:@"audioTest" ofType:@"mp3"];
   audioPlayer = [[AVAudioPlayer alloc]initWithContentsOfURL:
   [NSURL fileURLWithPath:path] error:NULL];
   [audioPlayer play];
}

-(IBAction)playVideo:(id)sender {
   NSString *path = [[NSBundle mainBundle]pathForResource:
   @"videoTest" ofType:@"mov"];
   moviePlayer = [[MPMoviePlayerViewController
   alloc]initWithContentURL:[NSURL fileURLWithPath:path]];
   [self presentModalViewController:moviePlayer animated:NO];
}
@end
```

Note

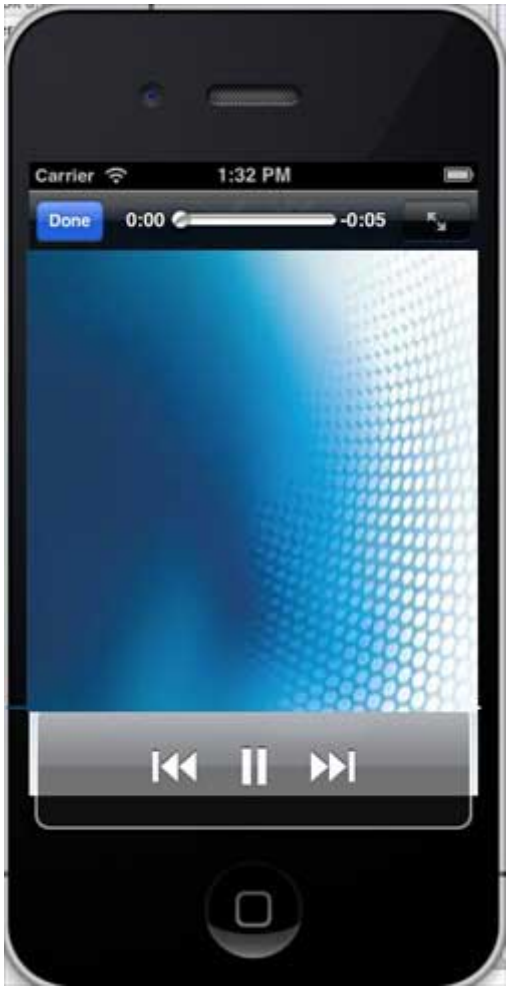
We need to add audio and video files for ensuring that we get the expected output.

Output

When we run the application, we'll get the following output −



When we click on play video, we will get an output as shown below −



When we click play audio, you will hear the audio.

iOS - File Handling

File handling cannot be explained visually with the application and hence the key methods that are used for handling files are explained below. Note that the application bundle only has read permission and we won't be able to modify the files. You can anyway modify the documents directory of your application.

Methods used in File Handling

The methods used for **accessing** and **manipulating** the files are discussed below. Here we have to replace FilePath1, FilePath2 and FilePath strings to our required full file paths to get the desired action.

Check if a File Exists at a Path

```
NSFileManager *fileManager = [NSFileManager defaultManager];
//Get documents directory
NSArray *directoryPaths = NSSearchPathForDirectoriesInDomains
(NSDocumentDirectory, NSUserDomainMask, YES);
NSString *documentsDirectoryPath = [directoryPaths objectAtIndex:0];

if ([fileManager fileExistsAtPath:@""]==YES) {
   NSLog(@"File exists");
}
```

Comparing Two File Contents

```
if ([fileManager contentsEqualAtPath:@"FilePath1" andPath:@" FilePath2"]) {
   NSLog(@"Same content");
}
```

Check if Writable, Readable, and Executable

```
if ([fileManager isWritableFileAtPath:@"FilePath"]) {
   NSLog(@"isWritable");
}

if ([fileManager isReadableFileAtPath:@"FilePath"]) {
   NSLog(@"isReadable");
}

if ( [fileManager isExecutableFileAtPath:@"FilePath"]) {
   NSLog(@"is Executable");
}
```

Move File

```
if([fileManager moveItemAtPath:@"FilePath1"
   toPath:@"FilePath2" error:NULL]) {
```

```
   NSLog(@"Moved successfully");
}
```

Copy File

```
if ([fileManager copyItemAtPath:@"FilePath1"
   toPath:@"FilePath2"  error:NULL]) {
   NSLog(@"Copied successfully");
}
```

Remove File

```
if ([fileManager removeItemAtPath:@"FilePath" error:NULL]) {
   NSLog(@"Removed successfully");
}
```

Read File

```
NSData *data = [fileManager contentsAtPath:@"Path"];
```

Write File

```
[fileManager createFileAtPath:@"" contents:data attributes:nil];
```

iOS - Accessing Maps

Maps are always helpful for us to locate places. Maps are integrated in iOS using the MapKit framework.

Steps Involved

Step 1 − Create a simple view-based application.

Step 2 − Select your project file, then select targets and then add MapKit.framework.

Step 3 − We should also add Corelocation.framework.

Step 4 − Add a MapView to ViewController.xib and create an ibOutlet and name it as mapView.

Step 5 − Create a new file by selecting File → New → File... → select Objective C class and click next.

Step 6 − Name the class as MapAnnotation with "sub class of" as NSObject.

Step 7 − Select create.

Step 8 − Update MapAnnotation.h as follows:

```
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

@interface MapAnnotation : NSObject<MKAnnotation>
@property (nonatomic, strong) NSString *title;
```

```
@property (nonatomic, readwrite) CLLocationCoordinate2D coordinate;

- (id)initWithTitle:(NSString *)title andCoordinate:
   (CLLocationCoordinate2D)coordinate2d;

@end
```

Step 9 − Update **MapAnnotation.m** as follows −

```
#import "MapAnnotation.h"

@implementation MapAnnotation
-(id)initWithTitle:(NSString *)title andCoordinate:
 (CLLocationCoordinate2D)coordinate2d {
   self.title = title;
   self.coordinate =coordinate2d;
   return self;
}
@end
```

Step 10 − Update **ViewController.h** as follows −

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import <CoreLocation/CoreLocation.h>
@interface ViewController : UIViewController<MKMapViewDelegate> {
   MKMapView *mapView;
}
@end
```

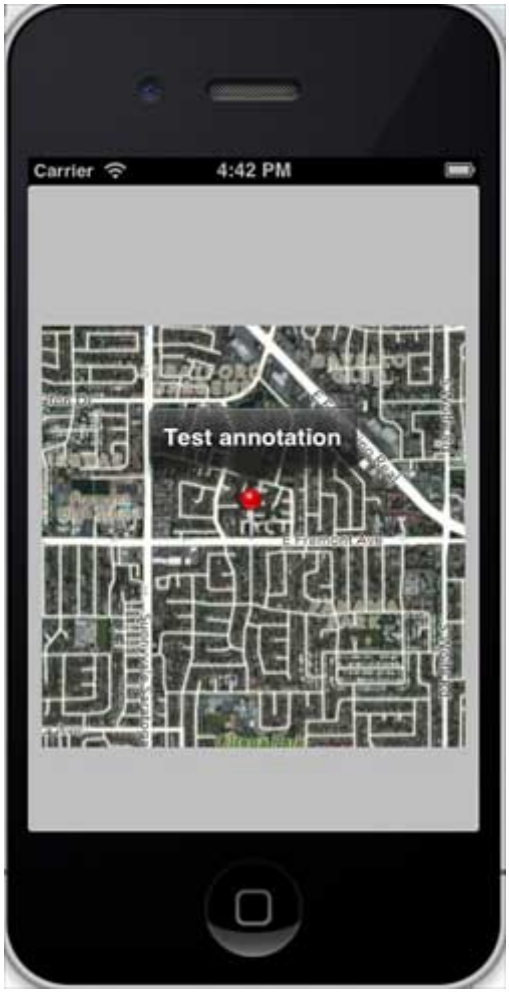
Step 11 − Update **ViewController.m** as follows −

```
#import "ViewController.h"
#import "MapAnnotation.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   mapView = [[MKMapView alloc]initWithFrame:
   CGRectMake(10, 100, 300, 300)];
   mapView.delegate = self;
   mapView.centerCoordinate = CLLocationCoordinate2DMake(37.32, -122.03);
   mapView.mapType = MKMapTypeHybrid;
   CLLocationCoordinate2D location;
   location.latitude = (double) 37.332768;
   location.longitude = (double) -122.030039;

   // Add the annotation to our map view
   MapAnnotation *newAnnotation = [[MapAnnotation alloc]
   initWithTitle:@"Apple Head quaters" andCoordinate:location];
   [mapView addAnnotation:newAnnotation];
   CLLocationCoordinate2D location2;
   location2.latitude = (double) 37.35239;
   location2.longitude = (double) -122.025919;
   MapAnnotation *newAnnotation2 = [[MapAnnotation alloc]
   initWithTitle:@"Test annotation" andCoordinate:location2];
```

```
    [mapView addAnnotation:newAnnotation2];
    [self.view addSubview:mapView];
}

// When a map annotation point is added, zoom to it (1500 range)
- (void)mapView:(MKMapView *)mv didAddAnnotationViews:(NSArray *)views {
    MKAnnotationView *annotationView = [views objectAtIndex:0];
    id <MKAnnotation> mp = [annotationView annotation];
    MKCoordinateRegion region = MKCoordinateRegionMakeWithDistance
    ([mp coordinate], 1500, 1500);
    [mv setRegion:region animated:YES];
    [mv selectAnnotation:mp animated:YES];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
@end
```

Output

When we run the application, we'll get the output as shown below −



When we scroll the map up, we will get the output as shown below −

iOS - In-App Purchase

In-App purchase is used to purchase additional content or upgrade features with respect to an application.

Steps Involved

Step 1 − In iTunes connect, ensure that you have a **unique App ID** and when we create the application update with the **bundle ID** and code signing in Xcode with corresponding provisioning profile.

Step 2 − Create a new application and update application information. You can know more about this in apple's **Add new apps** documentation.

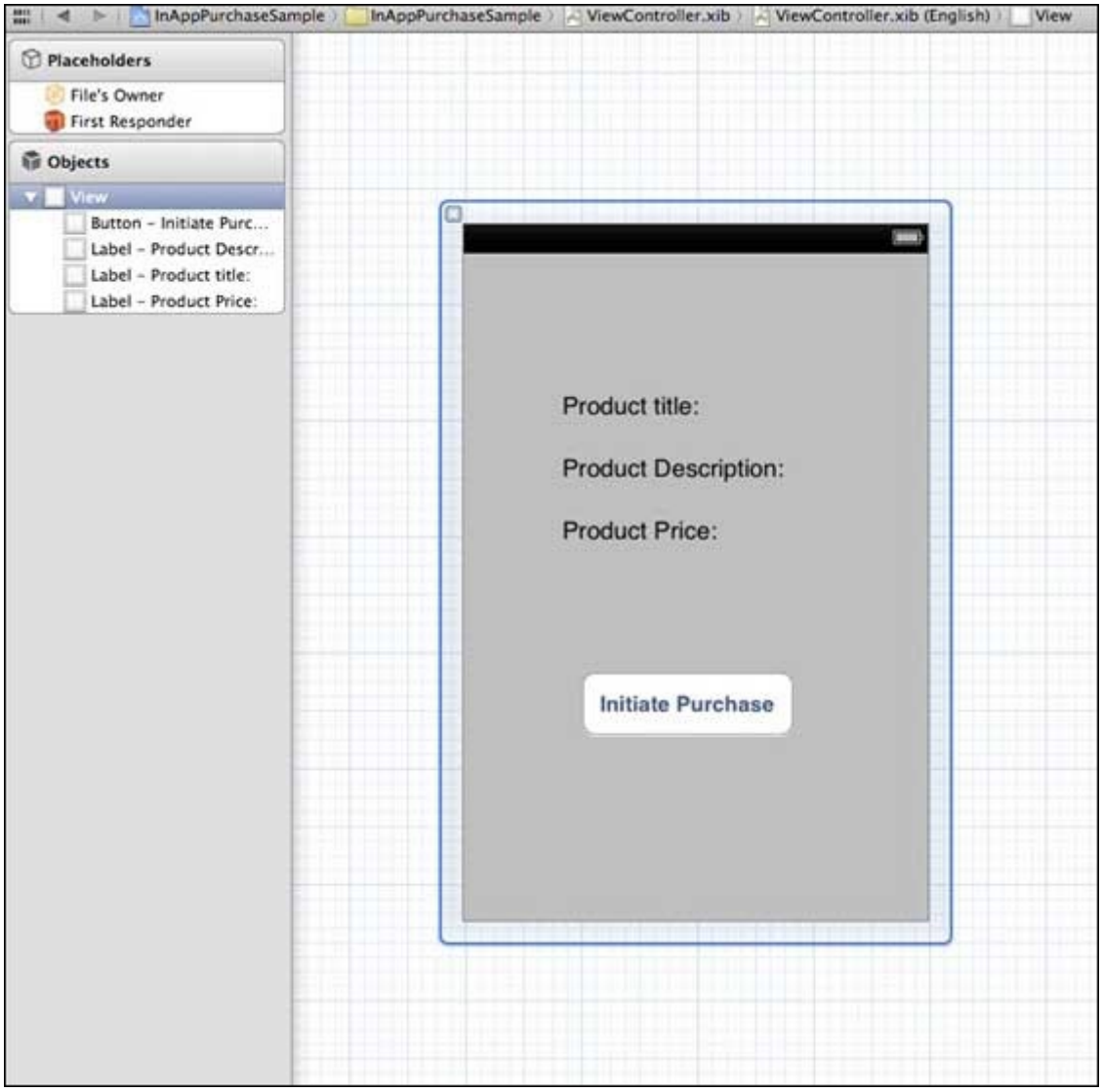
Step 3 − Add a new product for in-app purchase in **Manage In-App Purchase** of your application's page.

Step 4 − Ensure you setup the bank details for your application. This needs to be setup for **In-App purchase** to work. Also, create a test user account using **Manage Users** option in iTunes connect page of your app.

Step 5 − The next steps are related to handling code and creating UI for our In-App purchase.

Step 6 − Create a **single view application** and enter the bundle identifier is the identifier specified in iTunes connect.

Step 7 − Update the **ViewController.xib** as shown below −



Step 8 − Create **IBOutlets** for the three labels and the button naming them as productTitleLabel, productDescriptionLabel, productPriceLabel and purchaseButton respectively.

Step 9 − Select your project file, then select targets and then add **StoreKit.framework**.

Step 10 − Update **ViewController.h** as follows −

```
#import <UIKit/UIKit.h>
#import <StoreKit/StoreKit.h>

@interface ViewController : UIViewController<
SKProductsRequestDelegate,SKPaymentTransactionObserver> {
   SKProductsRequest *productsRequest;
   NSArray *validProducts;
   UIActivityIndicatorView *activityIndicatorView;
   IBOutlet UILabel *productTitleLabel;
   IBOutlet UILabel *productDescriptionLabel;
   IBOutlet UILabel *productPriceLabel;
   IBOutlet UIButton *purchaseButton;
}

- (void)fetchAvailableProducts;
- (BOOL)canMakePurchases;
- (void)purchaseMyProduct:(SKProduct*)product;
- (IBAction)purchase:(id)sender;
```

```
@end
```

Step 11 − Update **ViewController.m** as follows −

```
#import "ViewController.h"
#define kTutorialPointProductID
@"com.tutorialPoints.testApp.testProduct"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];

   // Adding activity indicator
   activityIndicatorView = [[UIActivityIndicatorView alloc]
   initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhiteLarge];
   activityIndicatorView.center = self.view.center;
   [activityIndicatorView hidesWhenStopped];
   [self.view addSubview:activityIndicatorView];
   [activityIndicatorView startAnimating];

   //Hide purchase button initially
   purchaseButton.hidden = YES;
   [self fetchAvailableProducts];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(void)fetchAvailableProducts {
   NSSet *productIdentifiers = [NSSet
   setWithObjects:kTutorialPointProductID,nil];
   productsRequest = [[SKProductsRequest alloc]
   initWithProductIdentifiers:productIdentifiers];
   productsRequest.delegate = self;
   [productsRequest start];
}

- (BOOL)canMakePurchases {
   return [SKPaymentQueue canMakePayments];
}

- (void)purchaseMyProduct:(SKProduct*)product {
   if ([self canMakePurchases]) {
      SKPayment *payment = [SKPayment paymentWithProduct:product];
      [[SKPaymentQueue defaultQueue] addTransactionObserver:self];
      [[SKPaymentQueue defaultQueue] addPayment:payment];
   } else {
      UIAlertView *alertView = [[UIAlertView alloc]initWithTitle:
      @"Purchases are disabled in your device" message:nil delegate:
      self cancelButtonTitle:@"Ok" otherButtonTitles: nil];
      [alertView show];
   }
}
-(IBAction)purchase:(id)sender {
```

```
   [self purchaseMyProduct:[validProducts objectAtIndex:0]];
   purchaseButton.enabled = NO;
}

#pragma mark StoreKit Delegate

-(void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions {
   for (SKPaymentTransaction *transaction in transactions) {
      switch (transaction.transactionState) {
         case SKPaymentTransactionStatePurchasing:
            NSLog(@"Purchasing");
         break;

         case SKPaymentTransactionStatePurchased:
            if ([transaction.payment.productIdentifier
            isEqualToString:kTutorialPointProductID]) {
               NSLog(@"Purchased ");
               UIAlertView *alertView = [[UIAlertView alloc]initWithTitle:
               @"Purchase is completed succesfully" message:nil delegate:
               self cancelButtonTitle:@"Ok" otherButtonTitles: nil];
               [alertView show];
            }
            [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
         break;

         case SKPaymentTransactionStateRestored:
            NSLog(@"Restored ");
            [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
         break;

         case SKPaymentTransactionStateFailed:
            NSLog(@"Purchase failed ");
         break
         default:
         break;
      }
   }
}

-(void)productsRequest:(SKProductsRequest *)request
 didReceiveResponse:(SKProductsResponse *)response {
   SKProduct *validProduct = nil;
   int count = [response.products count];

   if (count>0) {
      validProducts = response.products;
      validProduct = [response.products objectAtIndex:0];

      if ([validProduct.productIdentifier
         isEqualToString:kTutorialPointProductID]) {
         [productTitleLabel setText:[NSString stringWithFormat:
            @"Product Title: %@",validProduct.localizedTitle]];
         [productDescriptionLabel setText:[NSString stringWithFormat:
            @"Product Desc: %@",validProduct.localizedDescription]];
         [productPriceLabel setText:[NSString stringWithFormat:
            @"Product Price: %@",validProduct.price]];
      }
   } else {
      UIAlertView *tmp = [[UIAlertView alloc]
         initWithTitle:@"Not Available"
         message:@"No products to purchase"
         delegate:self
```

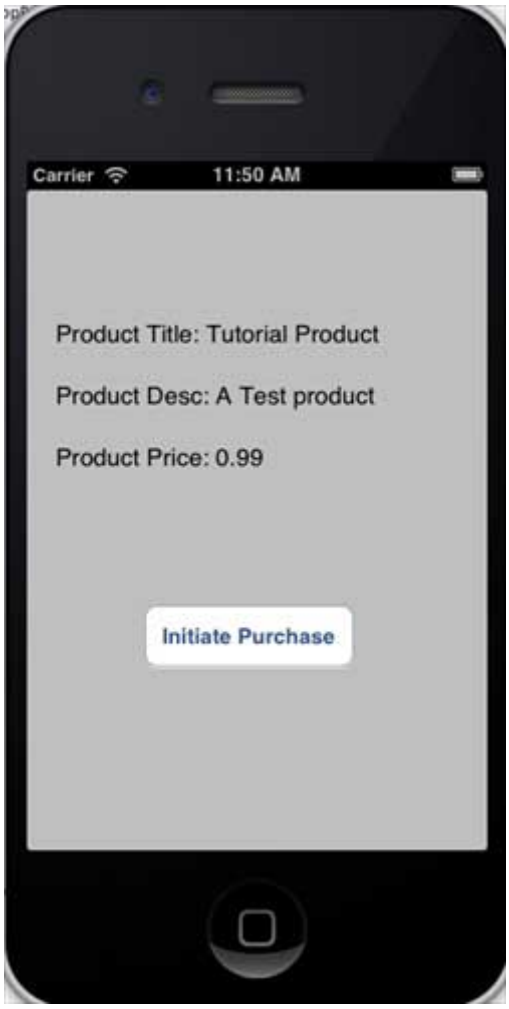
```
            cancelButtonTitle:nil
            otherButtonTitles:@"Ok", nil];
            [tmp show];
    }

    [activityIndicatorView stopAnimating];
    purchaseButton.hidden = NO;
}
@end
```

Note

You have to update kTutorialPointProductID to the productID you have created for your In-App Purchase. You can add more than one product by updating the productIdentifiers's NSSet in fetchAvailableProducts. Similary, handle the purchase related actions for product IDs you add.

Output

When we run the application, we'll get the following output −



Ensure you had logged out of your account in the settings screen. On clicking the Initiate Purchase, select Use Existing Apple ID. Enter your valid test account username and password. You will be shown the following alert in a few seconds.

Once your product is purchased successfully, you will get the following alert. You can see relevant code for updating the application features where we show this alert.

iOS - iAd Integration

iAd is used to display ads, served by the apple server. iAd helps us in earning revenue from an iOS application.

iAd Integration – Steps Involved

Step 1 − Create a simple view-based application.

Step 2 − Select your project file, then select targets and then add iAd.framework in choose frameworks.

Step 3 − Update ViewController.h as follows −

```
#import <UIKit/UIKit.h>
#import <iAd/iAd.h>
@interface ViewController : UIViewController<ADBannerViewDelegate> {
   ADBannerView *bannerView;
}
@end
```

Step 4 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end
```

```
@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   bannerView = [[ADBannerView alloc]initWithFrame:
   CGRectMake(0, 0, 320, 50)];

   // Optional to set background color to clear color
   [bannerView setBackgroundColor:[UIColor clearColor]];
   [self.view addSubview: bannerView];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

#pragma mark - AdViewDelegates

-(void)bannerView:(ADBannerView *)banner
   didFailToReceiveAdWithError:(NSError *)error {
   NSLog(@"Error loading");
}

-(void)bannerViewDidLoadAd:(ADBannerView *)banner {
   NSLog(@"Ad loaded");
}

-(void)bannerViewWillLoadAd:(ADBannerView *)banner {
   NSLog(@"Ad will load");
}

-(void)bannerViewActionDidFinish:(ADBannerView *)banner {
   NSLog(@"Ad did finish");

}
@end
```

Output

When we run the application, we'll get the following output −

iOS - GameKit

Gamekit is a framework that provides leader board, achievements, and more features to an iOS application. In this tutorial, we will be explaining the steps involved in adding a leader board and updating the score.

Steps Involved

Step 1 − In iTunes connect, ensure that you have a **unique App ID** and when we create the application update with the **bundle ID** and code signing in Xcode with corresponding provisioning profile.

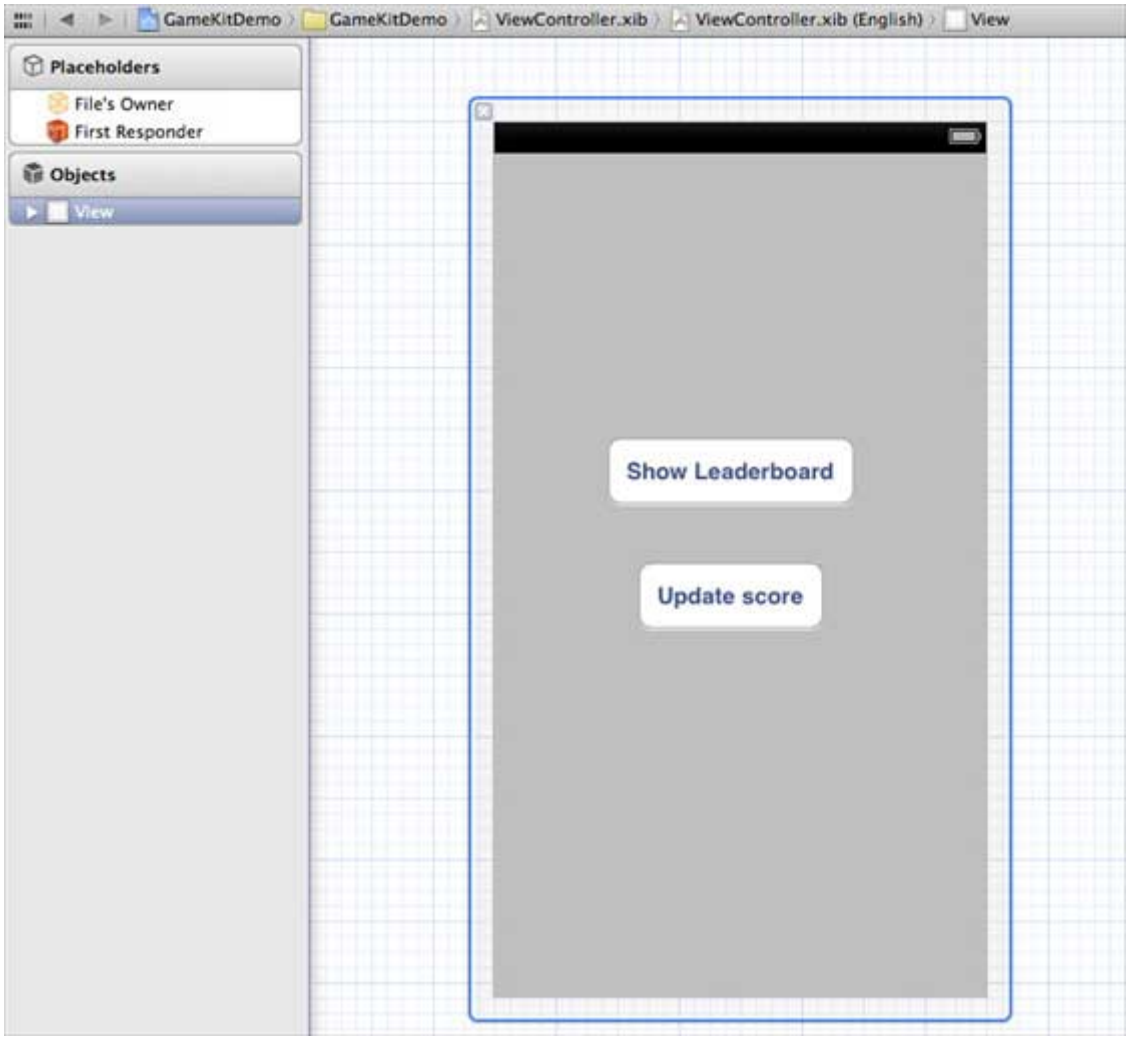
Step 2 − Create a new application and update application information. You can know more about this in apple-add new apps documentation.

Step 3 − Setup a leader board in **Manage Game Center** of your application's page where add a single leaderboard and give **leaderboard ID** and score Type. Here we give leader board ID as tutorialsPoint.

Step 4 − The next steps are related to handling code and creating UI for our application.

Step 5 − Create a **single view application** and enter the **bundle identifier** is the identifier specified in **iTunes connect**.

Step 6 − Update the ViewController.xib as shown below −



Step 7 − Select your project file, then select **targets** and then add **GameKit.framework**.

Step 8 − Create **IBActions** for the buttons we have added.

Step 9 − Update the **ViewController.h** file as follows −

```
#import <UIKit/UIKit.h>
#import <GameKit/GameKit.h>

@interface ViewController : UIViewController
<GKLeaderboardViewControllerDelegate>

-(IBAction)updateScore:(id)sender;
-(IBAction)showLeaderBoard:(id)sender;

@end
```

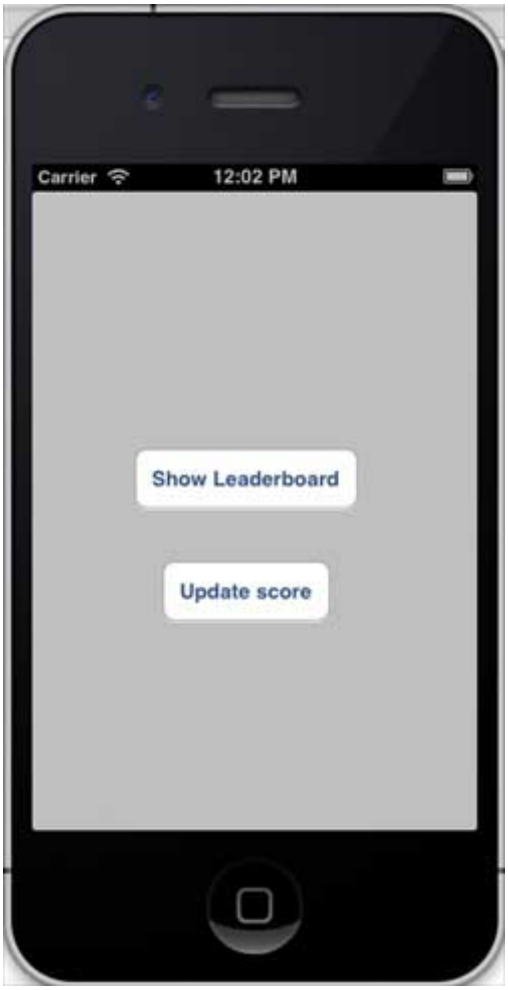
Step 10 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   if([GKLocalPlayer localPlayer].authenticated == NO) {
```

```
      [[GKLocalPlayer localPlayer]
      authenticateWithCompletionHandler:^(NSError *error) {
         NSLog(@"Error%@",error);
      }];
   }
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

- (void) updateScore: (int64_t) score
   forLeaderboardID: (NSString*) category {
   GKScore *scoreObj = [[GKScore alloc]
   initWithCategory:category];
   scoreObj.value = score;
   scoreObj.context = 0;
   [scoreObj reportScoreWithCompletionHandler:^(NSError *error) {
      // Completion code can be added here
      UIAlertView *alert = [[UIAlertView alloc]
      initWithTitle:nil message:@"Score Updated Succesfully"
      delegate:self cancelButtonTitle:@"Ok" otherButtonTitles: nil];
      [alert show];
   }];
}

-(IBAction)updateScore:(id)sender {
   [self updateScore:200 forLeaderboardID:@"tutorialsPoint"];
}

-(IBAction)showLeaderBoard:(id)sender {
   GKLeaderboardViewController *leaderboardViewController =
   [[GKLeaderboardViewController alloc] init];
   leaderboardViewController.leaderboardDelegate = self;
   [self presentModalViewController:
   leaderboardViewController animated:YES];
}

#pragma mark - Gamekit delegates
- (void)leaderboardViewControllerDidFinish:
(GKLeaderboardViewController *)viewController {
   [self dismissModalViewControllerAnimated:YES];
}
@end
```

Output

When we run the application, we'll get the following output −



When we click "show leader board", we would get a screen similar to the following −

When we click "update score", the score will be updated to our leader board and we will get an alert as shown below −

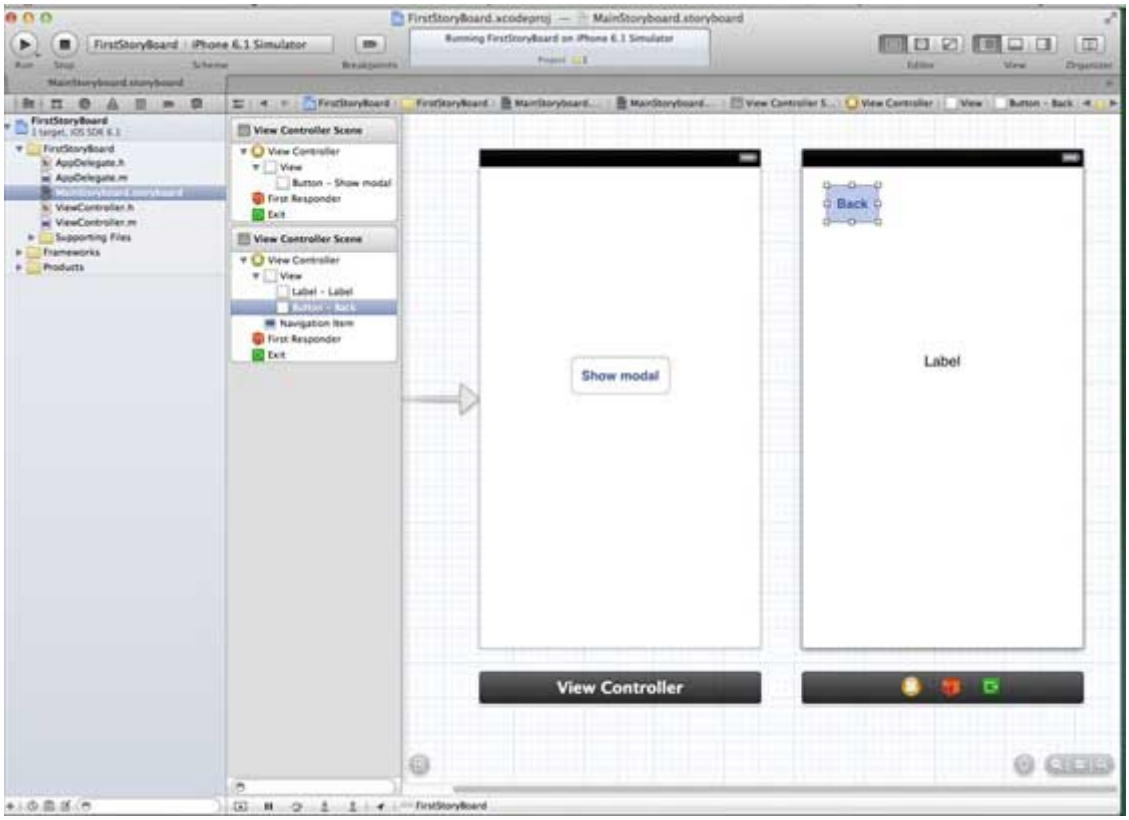
iOS - Storyboards

Storyboards are introduced in iOS 5. When we use storyboards, our deployment target should be 5.0 or higher. Storyboards help us create all the screens of an application and interconnect the screens under one interface MainStoryboard.storyboard. It also helps in reducing the coding of pushing/presenting view controllers.

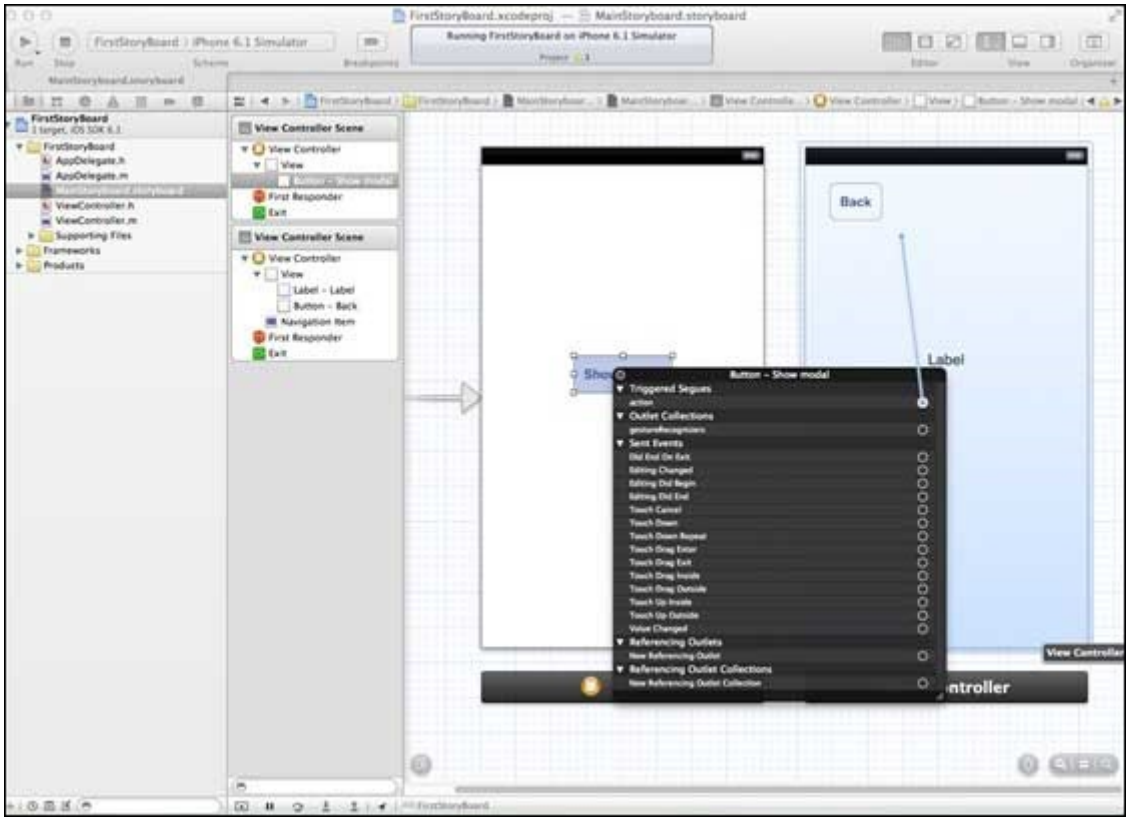
Steps Involved

Step 1 − Create a **single view application** and make sure that you select **storyboard** checkbox while creating the application.

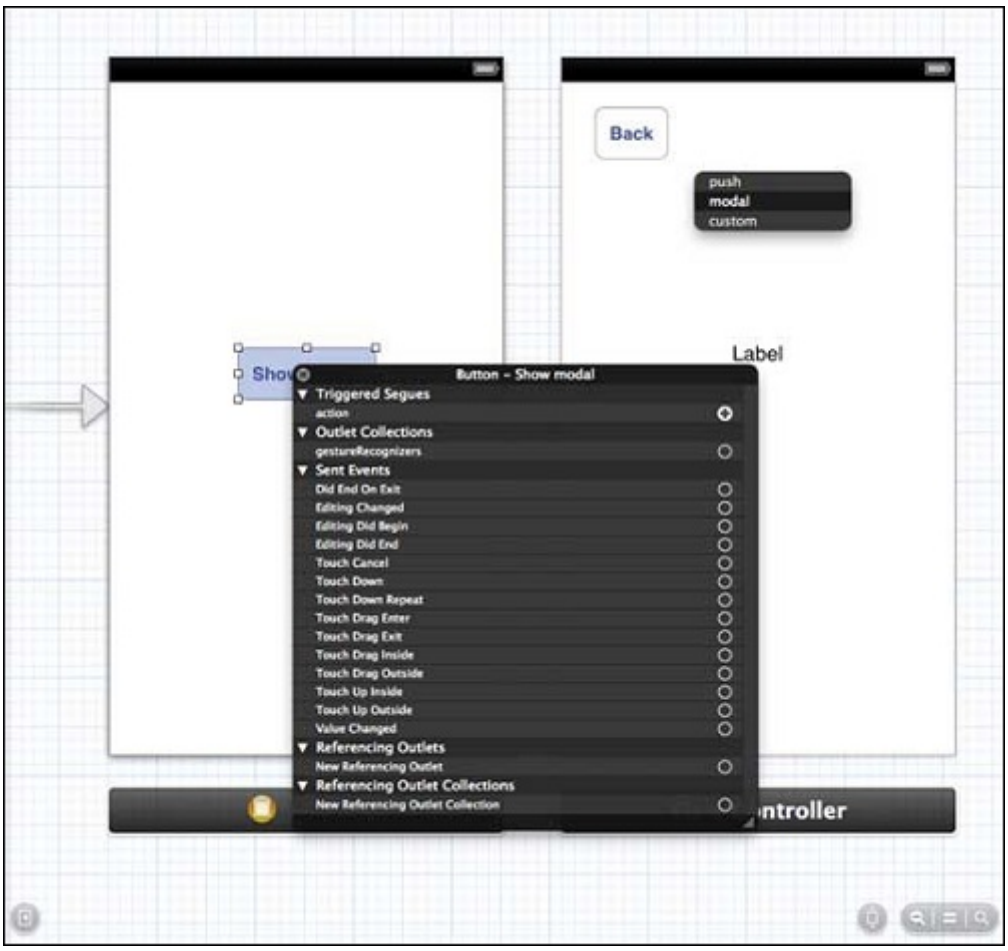
Step 2 − Select **MainStoryboard.storyboard** where you can find single view controller. Add one more view controllers and update the view controllers as shown below.



Step 3 − Let us now connect both the view controllers. Right-click on the "show modal" button and drag it to the right view controller in the left side view controller as shown below.



Step 4 − Select modal from the three options displayed as shown below.



Step 5 − Update **ViewController.h** as follows −

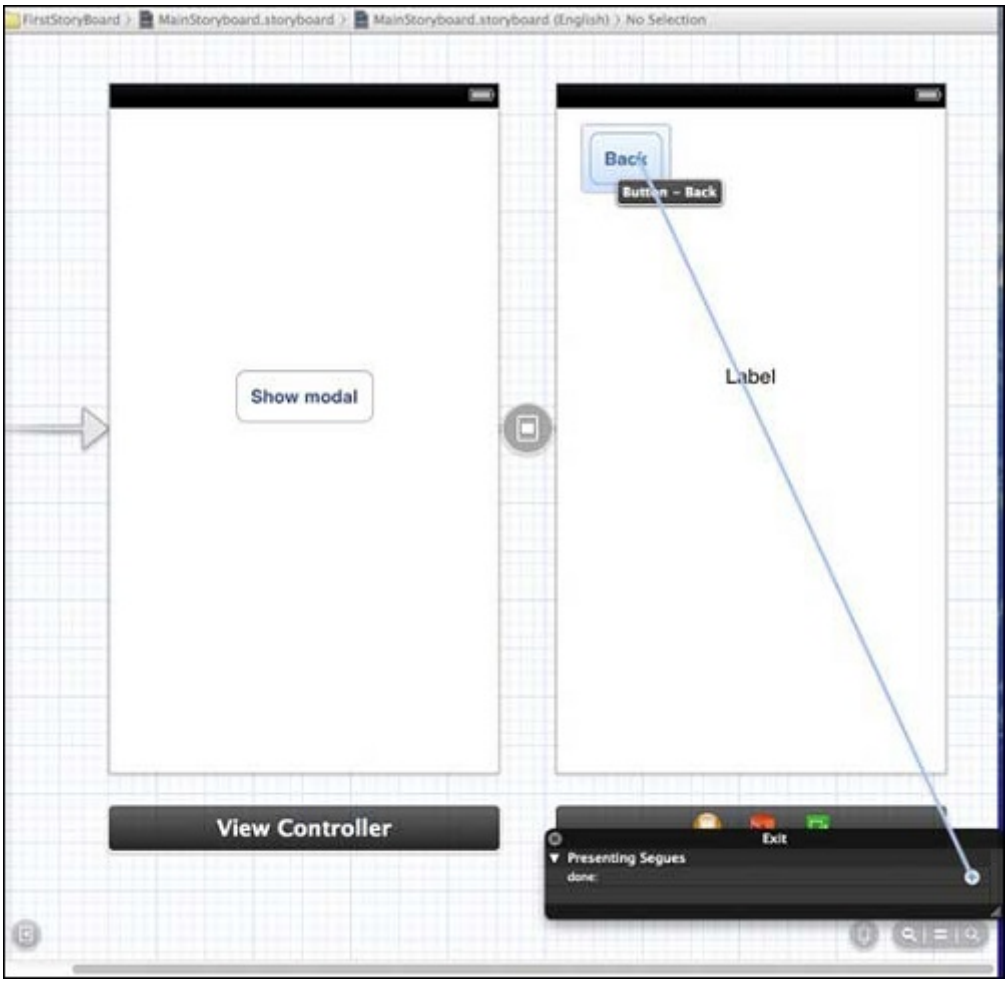
```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

-(IBAction)done:(UIStoryboardSegue *)seque;

@end
```

Step 6 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(IBAction)done:(UIStoryboardSegue *)seque {
   [self.navigationController popViewControllerAnimated:YES];
}
```

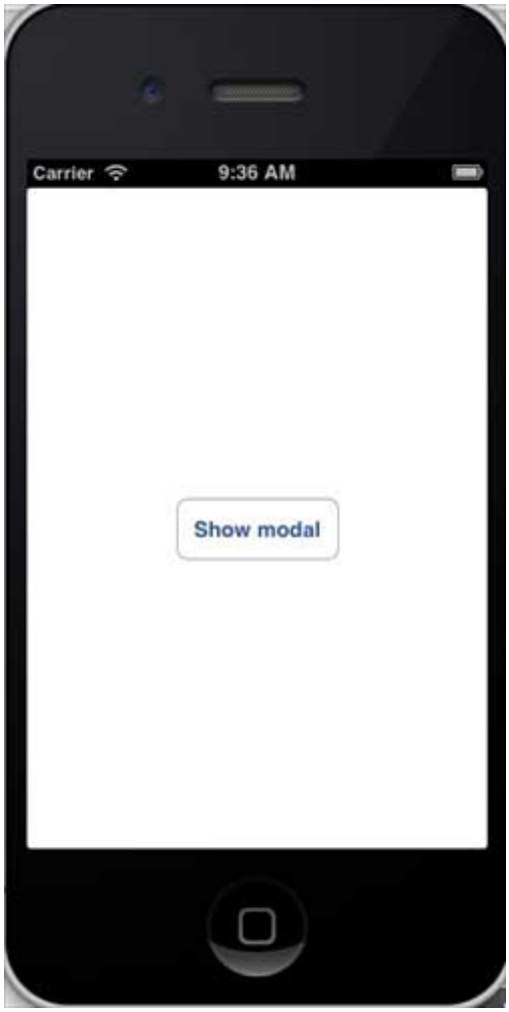
```
@end
```

Step 7 − Select the MainStoryboard.storyboard and right-click on the Exit button in the right side view controller, select done and connect with the back button as shown below.



Output

When we run the application in an **iPhone** device, we'll get the following output −



When we select "show modal", we will get the following output −

iOS - Auto Layouts

Auto-layouts were introduced in **iOS 6.0.** When we use auto-layouts, our deployment target should be 6.0 and higher. Auto-layouts help us create interfaces that can be used for multiple orientations and multiple devices.

Goal of Our Example

We will add two buttons that will be placed in a certain distance from the center of the screen. We will also try to add a resizable text field that will be placed from a certain distance from above the buttons.

Our Approach

We will add a text field and two buttons in the code along with their constraints. The constraints of each UI Elements will be created and added to the super view. We will have to disable auto-resizing for each of the UI elements we add in order to get the desired result.

Steps Involved

Step 1 − Create a simple view-based application.

Step 2 − We will edit only ViewController.m and it is as follows −

```
#import "ViewController.h"
@interface ViewController ()
@property (nonatomic, strong) UIButton *leftButton;
@property (nonatomic, strong) UIButton *rightButton;
@property (nonatomic, strong) UITextField *textfield;

@end
@implementation ViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   UIView *superview = self.view;

   /*1. Create leftButton and add to our view*/
   self.leftButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
   self.leftButton.translatesAutoresizingMaskIntoConstraints = NO;
   [self.leftButton setTitle:@"LeftButton" forState:UIControlStateNormal];
   [self.view addSubview:self.leftButton];

   /* 2. Constraint to position LeftButton's X*/
   NSLayoutConstraint *leftButtonXConstraint = [NSLayoutConstraint
   constraintWithItem:self.leftButton attribute:NSLayoutAttributeCenterX
   relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:superview attribute:
   NSLayoutAttributeCenterX multiplier:1.0 constant:-60.0f];

   /* 3. Constraint to position LeftButton's Y*/
   NSLayoutConstraint *leftButtonYConstraint = [NSLayoutConstraint
   constraintWithItem:self.leftButton attribute:NSLayoutAttributeCenterY
   relatedBy:NSLayoutRelationEqual toItem:superview attribute:
   NSLayoutAttributeCenterY multiplier:1.0f constant:0.0f];
```

```
   /* 4. Add the constraints to button's superview*/
   [superview addConstraints:@[ leftButtonXConstraint,
   leftButtonYConstraint]];

   /*5. Create rightButton and add to our view*/
   self.rightButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
   self.rightButton.translatesAutoresizingMaskIntoConstraints = NO;
   [self.rightButton setTitle:@"RightButton" forState:UIControlStateNormal];
   [self.view addSubview:self.rightButton];

   /*6. Constraint to position RightButton's X*/
   NSLayoutConstraint *rightButtonXConstraint = [NSLayoutConstraint
   constraintWithItem:self.rightButton attribute:NSLayoutAttributeCenterX
   relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:superview attribute:
   NSLayoutAttributeCenterX multiplier:1.0 constant:60.0f];

   /*7. Constraint to position RightButton's Y*/
   rightButtonXConstraint.priority = UILayoutPriorityDefaultHigh;
   NSLayoutConstraint *centerYMyConstraint = [NSLayoutConstraint
   constraintWithItem:self.rightButton attribute:NSLayoutAttributeCenterY
   relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:superview attribute:
   NSLayoutAttributeCenterY multiplier:1.0f constant:0.0f];
   [superview addConstraints:@[centerYMyConstraint,
   rightButtonXConstraint]];

   //8. Add Text field
   self.textfield = [[UITextField alloc]initWithFrame:
   CGRectMake(0, 100, 100, 30)];
   self.textfield.borderStyle = UITextBorderStyleRoundedRect;
   self.textfield.translatesAutoresizingMaskIntoConstraints = NO;
   [self.view addSubview:self.textfield];

   //9. Text field Constraints
   NSLayoutConstraint *textFieldTopConstraint = [NSLayoutConstraint
   constraintWithItem:self.textfield attribute:NSLayoutAttributeTop
   relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:superview
   attribute:NSLayoutAttributeTop multiplier:1.0 constant:60.0f];
   NSLayoutConstraint *textFieldBottomConstraint = [NSLayoutConstraint
   constraintWithItem:self.textfield attribute:NSLayoutAttributeTop
   relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:self.rightButton
   attribute:NSLayoutAttributeTop multiplier:0.8 constant:-60.0f];
   NSLayoutConstraint *textFieldLeftConstraint = [NSLayoutConstraint
   constraintWithItem:self.textfield attribute:NSLayoutAttributeLeft
   relatedBy:NSLayoutRelationEqual toItem:superview attribute:
   NSLayoutAttributeLeft multiplier:1.0 constant:30.0f];
   NSLayoutConstraint *textFieldRightConstraint = [NSLayoutConstraint
   constraintWithItem:self.textfield attribute:NSLayoutAttributeRight
   relatedBy:NSLayoutRelationEqual toItem:superview attribute:
   NSLayoutAttributeRight multiplier:1.0 constant:-30.0f];
   [superview addConstraints:@[textFieldBottomConstraint ,
   textFieldLeftConstraint, textFieldRightConstraint,
   textFieldTopConstraint]];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}
@end
```

Points to Note

In steps marked 1, 5, and 8, we just programmatically added two buttons and a text field respectively.

In the rest of the steps, we created constraints and added those constraints to the respective super views, which are actually self-views. The constraints of one of the left buttons is as shown below −

```
NSLayoutConstraint *leftButtonXConstraint = [NSLayoutConstraint
constraintWithItem:self.leftButton attribute:NSLayoutAttributeCenterX
relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:superview attribute:
NSLayoutAttributeCenterX multiplier:1.0 constant:-60.0f];
```

We have constraintWithItem and toItem which decide between which UI elements we are creating the constraint. The attribute decides on what basis the two elements are linked together. "relatedBy" decides how much effect the attributes have between the elements. Multiplier is the multiplication factor and constant will be added to the multipler.

In the above example, the X of leftButton is always greater than or equal to -60 pixels with respect to the center of the super view. Similarly, other constraints are defined.

Output

When we run the application, we'll get the following output on the iPhone simulator −

When we change the orientation of the simulator to landscape, we will get the following output −

When we run the same application on iPhone 5 simulator, we will get the following output −

When we change the orientation of the simulator to landscape, we will get the following output −

iOS - Twitter and Facebook

Twitter has been integrated in **iOS 5.0** and Facebook has been integrated in **iOS 6.0**. Our tutorial focuses on using the classes provided by Apple and the deployment targets for Twitter and Facebook are iOS 5.0 and iOS 6.0 respectively.

Steps Involved

Step 1 − Create a simple view-based application.

Step 2 − Select your project file, then select **targets** and then add **Social.framework** and **Accounts.framework** in **choose frameworks**.

Step 3 − Add two buttons named facebookPost and twitterPost and create ibActions for them.

Step 4 − Update **ViewController.h** as follows −

```
#import <Social/Social.h>
#import <Accounts/Accounts.h>
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

-(IBAction)twitterPost:(id)sender;
-(IBAction)facebookPost:(id)sender;

@end
```

Step 5 − Update **ViewController.m** as follows −

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController
```

```
- (void)viewDidLoad {
   [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
   [super didReceiveMemoryWarning];
   // Dispose of any resources that can be recreated.
}

-(IBAction)facebookPost:(id)sender {
   SLComposeViewController *controller = [SLComposeViewController
   composeViewControllerForServiceType:SLServiceTypeFacebook];
   SLComposeViewControllerCompletionHandler myBlock =
   ^(SLComposeViewControllerResult result){

      if (result == SLComposeViewControllerResultCancelled) {
         NSLog(@"Cancelled");
      } else {
         NSLog(@"Done");
      }
      [controller dismissViewControllerAnimated:YES completion:nil];
   };
   controller.completionHandler = myBlock;

   //Adding the Text to the facebook post value from iOS
   [controller setInitialText:@"My test post"];

   //Adding the URL to the facebook post value from iOS
   [controller addURL:[NSURL URLWithString:@"http://www.test.com"]];

   //Adding the Text to the facebook post value from iOS
   [self presentViewController:controller animated:YES completion:nil];
}

-(IBAction)twitterPost:(id)sender {
   SLComposeViewController *tweetSheet = [SLComposeViewController
   composeViewControllerForServiceType:SLServiceTypeTwitter];
   [tweetSheet setInitialText:@"My test tweet"];
   [self presentModalViewController:tweetSheet animated:YES];
}
@end
```

Output

When we run the application and click facebookPost, we will get the following output −

When we click twitterPost, we will get the following output −

iOS - Memory Management

Memory management in iOS was initially non-ARC (Automatic Reference Counting), where we have to retain and release the objects. Now, it supports ARC and we don't have to retain and release the objects. Xcode takes care of the job automatically in compile time.

Memory Management Issues

As per Apple documentation, the two major issues in memory management are −

Freeing or overwriting data that is still in use. It causes memory corruption and typically results in your application crashing, or worse, corrupted user data.

Not freeing data that is no longer in use causes memory leaks. When allocated memory is not freed even though it is never going to be used again, it is known as memory leak. Leaks cause your application to use ever-increasing amounts of memory, which in turn may result in poor system performance or (in iOS) your application being terminated.

Memory Management Rules

We own the objects we create, and we have to subsequently release them when they are no longer needed.

Use Retain to gain ownership of an object that you did not create. You have to release these objects too when they are not needed.

Don't release the objects that you don't own.

Handling Memory in ARC

You don't need to use release and retain in ARC. So, all the view controller's objects will be released when the view controller is removed. Similarly, any object’s sub-objects will be released when they are released. Note that if other classes have a strong reference to an object of a class, then the whole class won't be released. So, it is recommended to use weak properties for delegates.

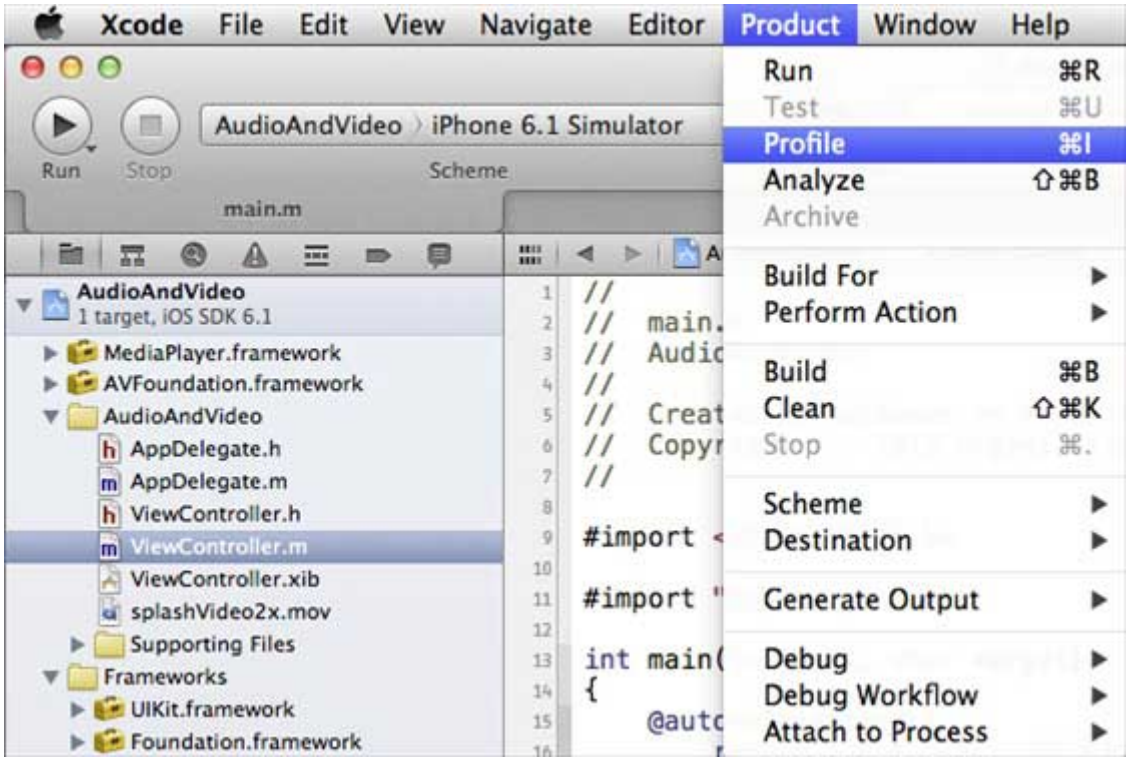
Memory Management Tools

We can analyze the usage of memory with the help of Xcode tool instruments. It includes tools such as Activity Monitor, Allocations, Leaks, Zombies, and so on.

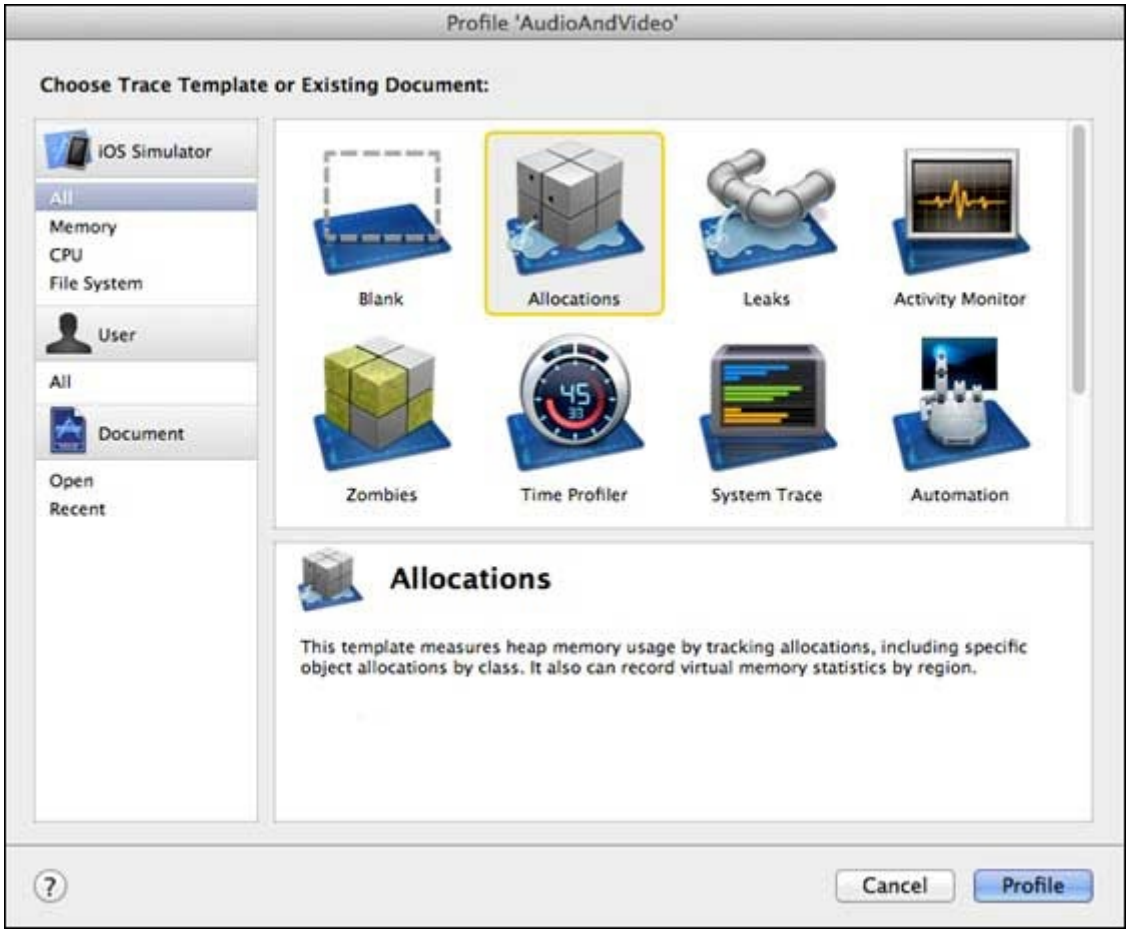
Steps for Analyzing Memory Allocations

Step 1 − Open an existing application.

Step 2 − Select Product and then Profile as shown below.

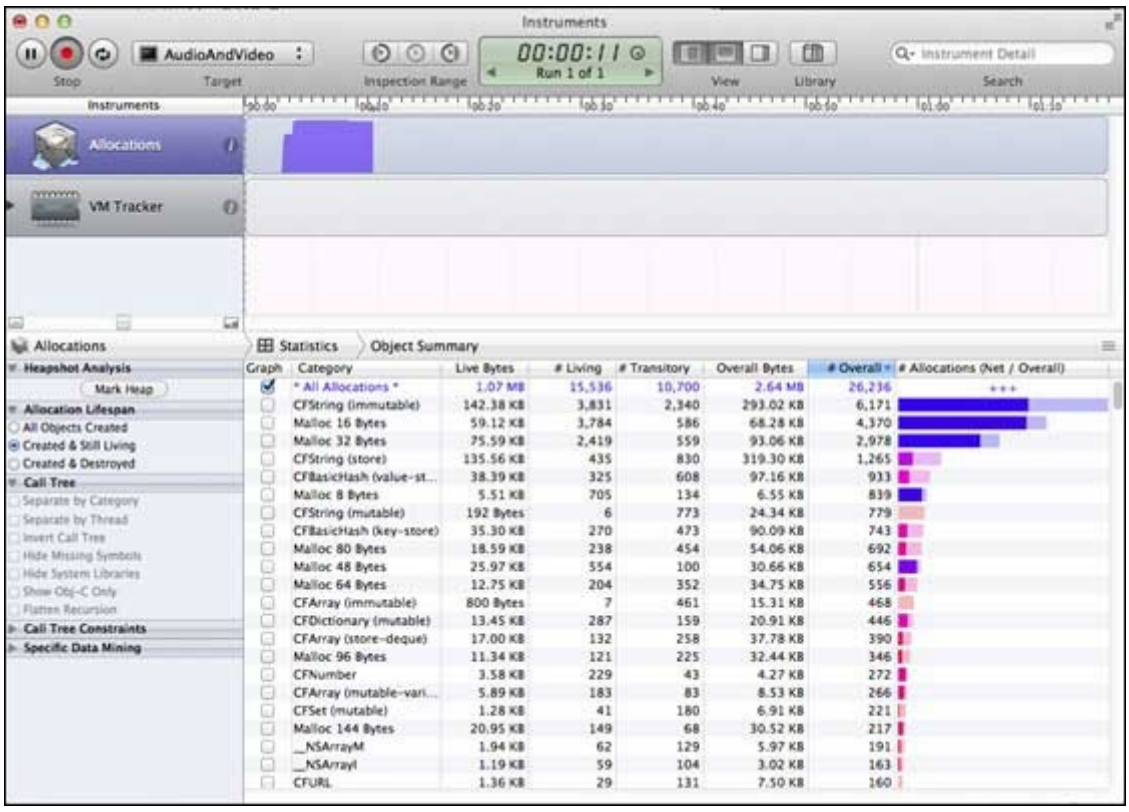


Step 3 − Select Allocations in the next screen shown below and select Profile.

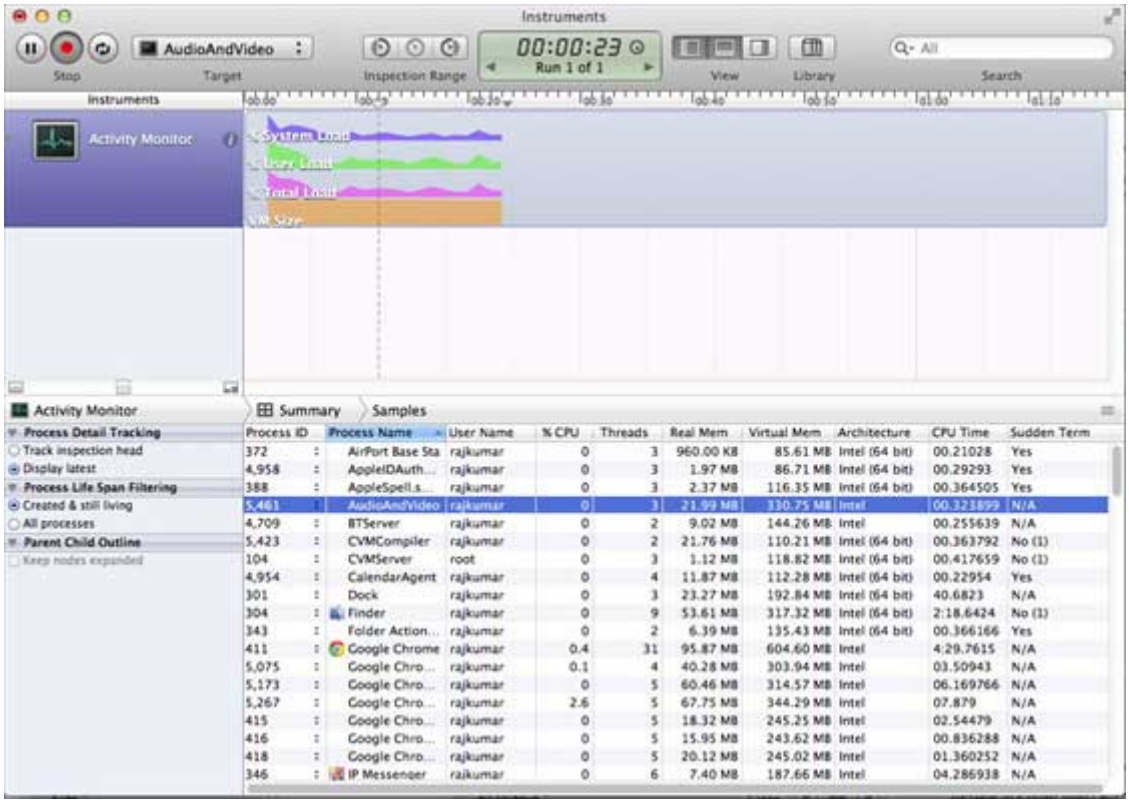


Step 4 − We will see the allocation of memory for different objects as shown below.

Step 5 − You can switch between view controllers and check whether the memory is released properly.



Step 6 − Similarly, instead of Allocations, we can use Activity Monitor to see the overall memory allocated for the application.



Step 7 − These tools help us access our memory consumption and locate the places where possible leaks have occurred.

iOS - Application Debugging

We may commit mistakes while developing an application, which can lead to different kinds of errors. In order to fix these errors or bugs, we need to debug the application.

Selecting a Debugger

Xcode has two debuggers namely, GDB and LLDB debuggers. GDB is selected by default. LLDB is a debugger that is a part of the LLVM open-source compiler project. You can change the debugger by "edit active schemes" option.

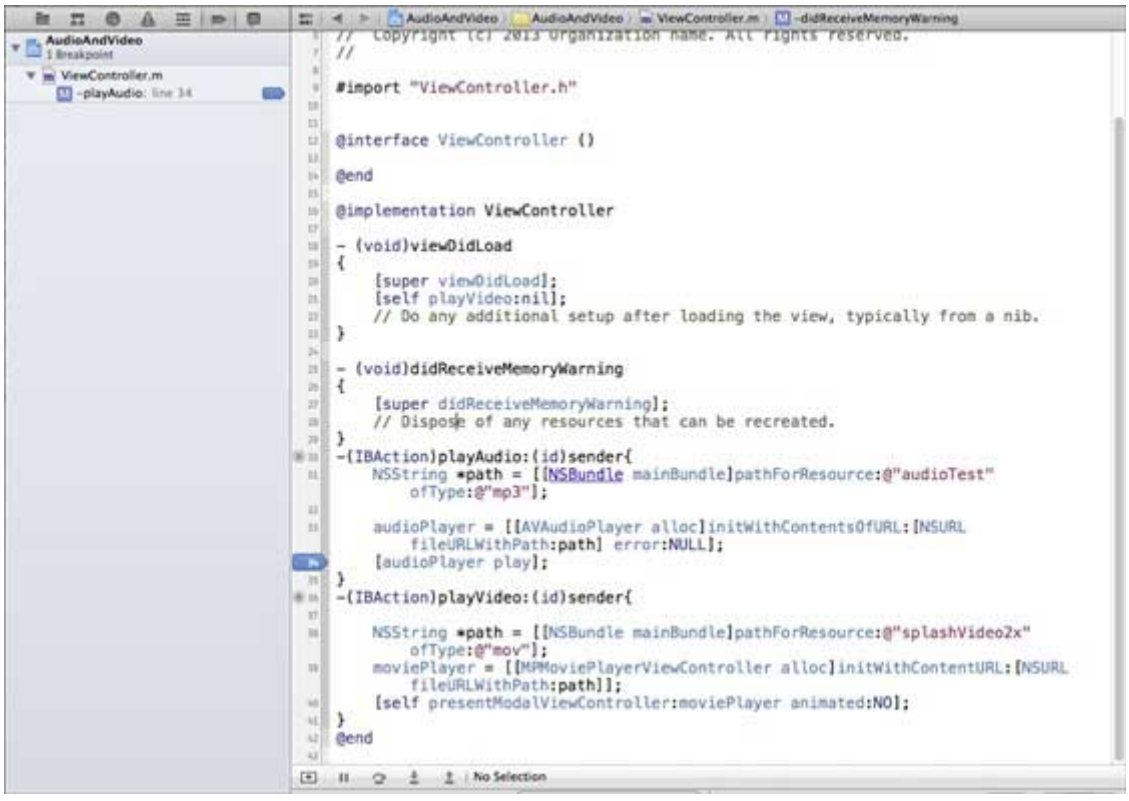
How to Find Coding Errors?

To locate coding-related errors, you need to build your application which will compile the code. In case the code contains errors, the compiler will display all the messages, errors, and warnings with their possible reasons.

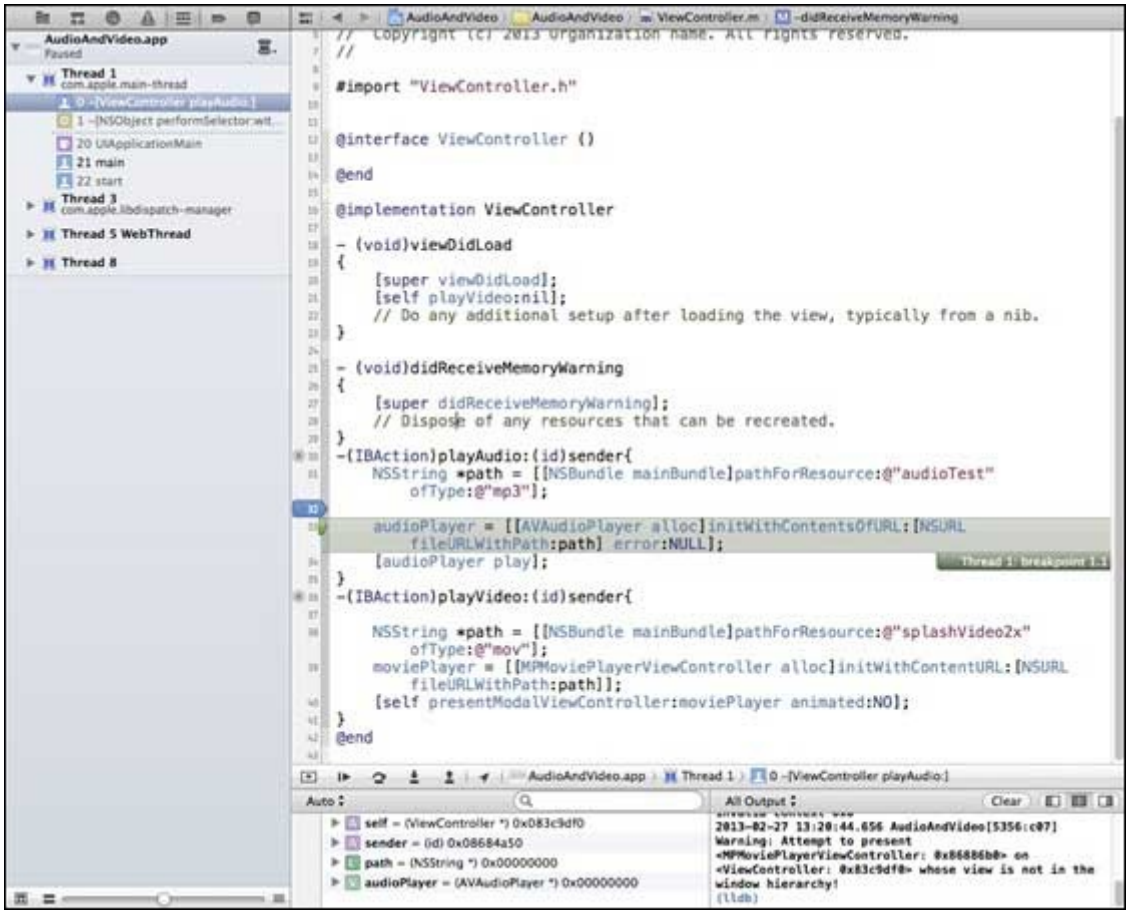
You can click Product and then Analyze to locate possible issues in an application.

Set Breakpoints

Breakpoints help us to know the different states of our application objects, which help us identifying many flaws including logical issues. We just need to click over the line number to create a breakpoint. To remove a breakpoint, simply click and drag it out. The following screenshot shows how to set a breakpoint −



When we run the application and select the playVideo button, the application will pause at the line number where we had set the breakpoint. It allows us the time to analyze the state of the application. When the breakpoint is triggered, we will get an output as shown below.



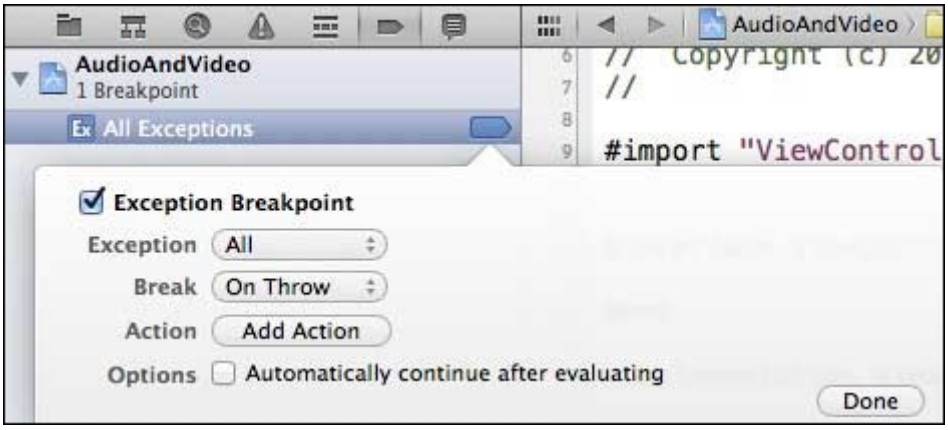
You can easily identify which thread has triggered the breakpoint. In the bottom, you can see objects like self, sender and so on, which hold the values of the corresponding objects and we can expand some of these objects, and see what is the state of each of these objects.

To continue the application we will select the continue button (left most button), in the debug area shown below. The other options include step in, step out and step over.

Exception Breakpoint

We also have exception breakpoints that trigger an application to stop at the location where the exception occurs. We can insert exception breakpoints by selecting the + button after selecting the debug navigator. You will get the following window.

Then we need to select Add Exception Breakpoint, which will display the following window.



You can collect more information on debugging and other Xcode features from Xcode 4 user guide .

Previous Page Next Page