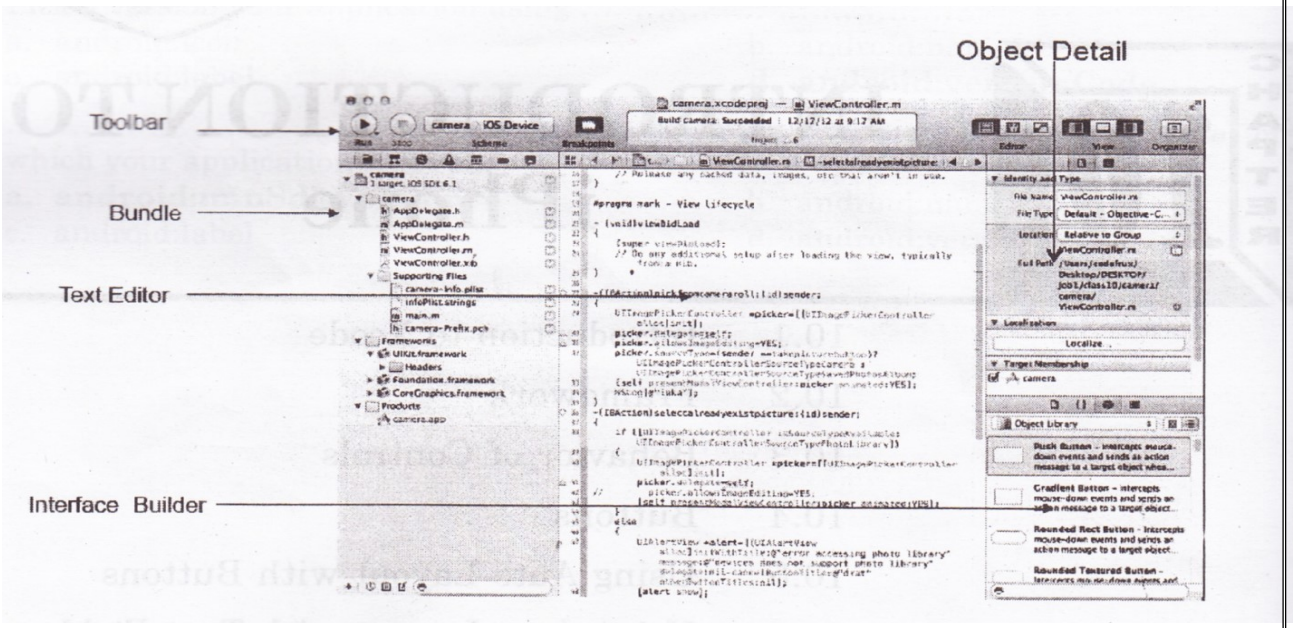
5 Introduction to iPhone

Topics Covered

- Introduction to X-code
- What is Storyboard
- Framework Design User Interface for button text view text field etc.
- Creating And Building Simple Application
- Cocoa Touch And MVC

Introduction to X-code

- X-Code is an integrated development environment (IDE).
- It contains software development tools developed by Apple for developing software for OS X and iOS First released in 2003.
- And it's latest version is available via the Mac App Store free of charge for Mac OS X Lion, OS X Mountain Lion and OS X Mavericks users.
- Registered developers can download preview releases and previous versions of the suite through the Apple Developer website.
- It is also used to develop application of iPhone.
- The X code interface integrates code editing, user interface design, asset management, testing, and debugging within a single workspace window.



- Xcode checks your source code as you type it, and when Xcode notices a mistake, the source code editor highlights the error.
- The source code editor then offers to fix it. Xcode speeds up your typing with intelligent code completion.
- Xcode reduces your typing by offering ready-to-use code snippets and source file templates. You can easily configure the source editor to display multiple views of the same file or to view multiple related files at once.

- Search-and-replace and refectory operations help you make extensive changes to your code quickly and safely.
- With these and other capabilities, Xcode makes it easier for you to write better code more quickly than you thought possible.

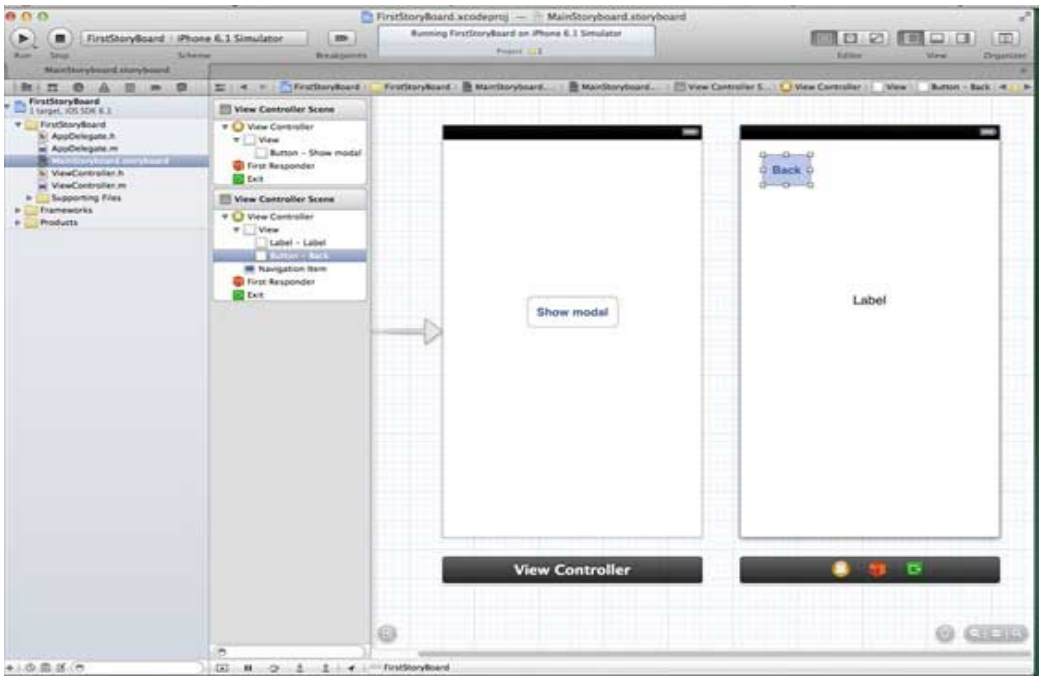
Introduction to Storyboard

- The storyboard is first introduced in iOS 5 to save time building user interfaces for the iOS applications.
- It is a visual representation of the user interface of an iOS app.
- It can be defined as the sequence of screens, each of which represents the ViewController and the Views.
- The transitions between two storyboard screens need a segue object, which represents a transition between two ViewControllers.
- The Storyboard is built using a visual editor provided by XCode, in which we can layout and design the user interfaces of the application by adding the widgets from the media library such as buttons, views, table views, text fields, etc.
- 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.

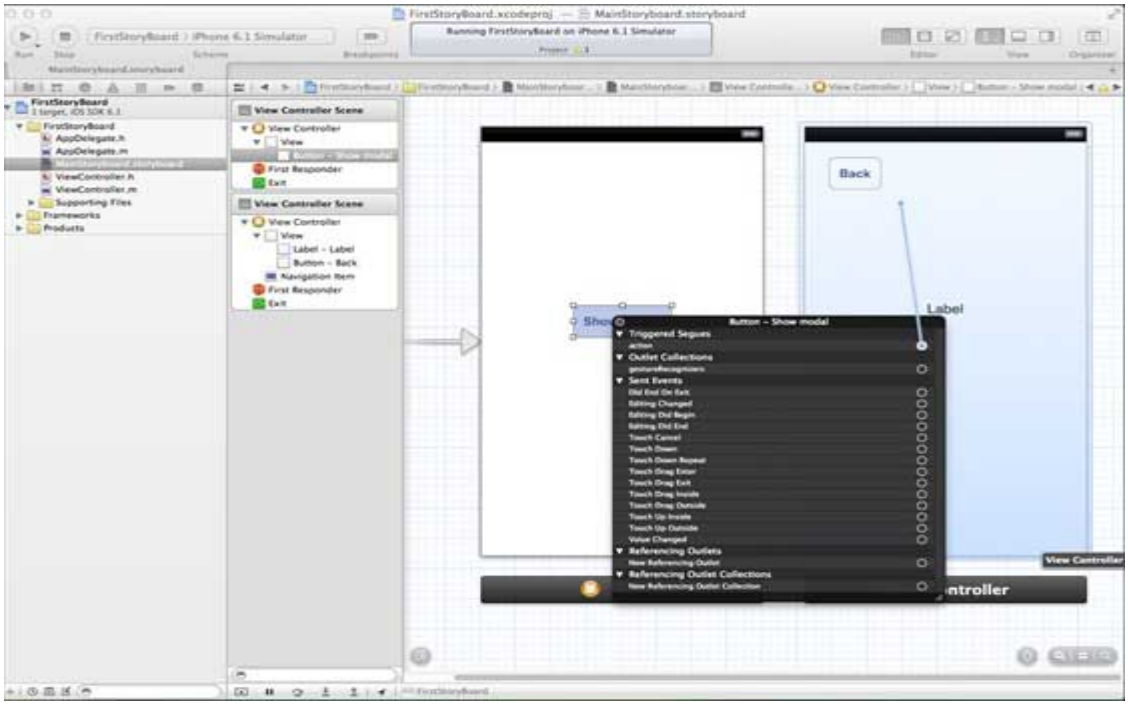
Steps Involved

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

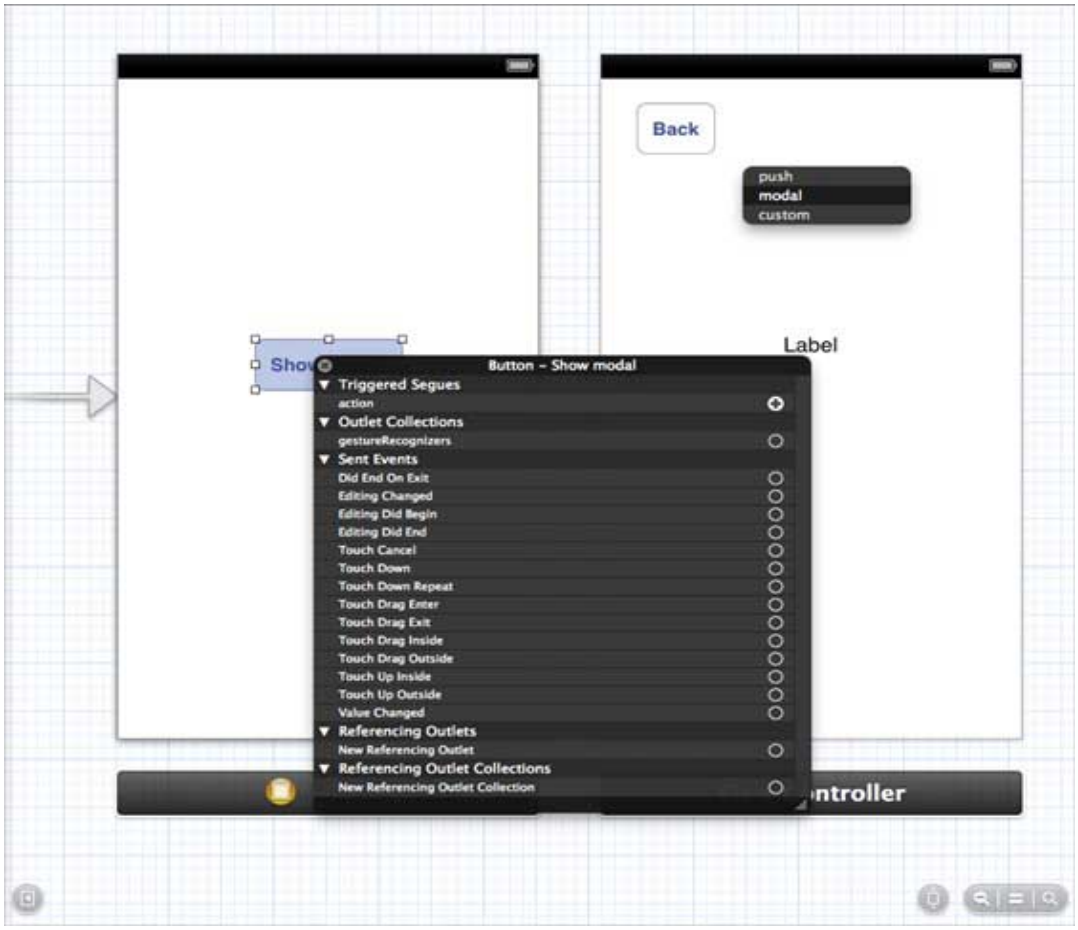
Step 2 − Select **MainStoryboard.storyboard** where you can find single view controller. Add one more view controllers and update the view controllers as shown below.



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.



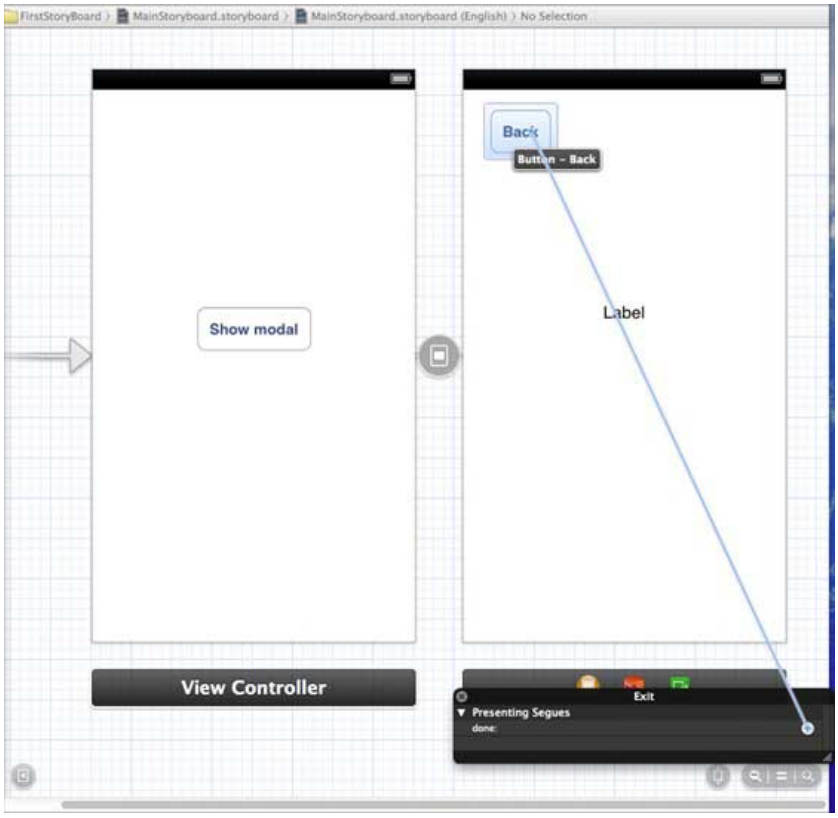
Step 5 − Update **ViewController.h** as follows –

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

@interface ViewController : UIViewController

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

@end
```

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

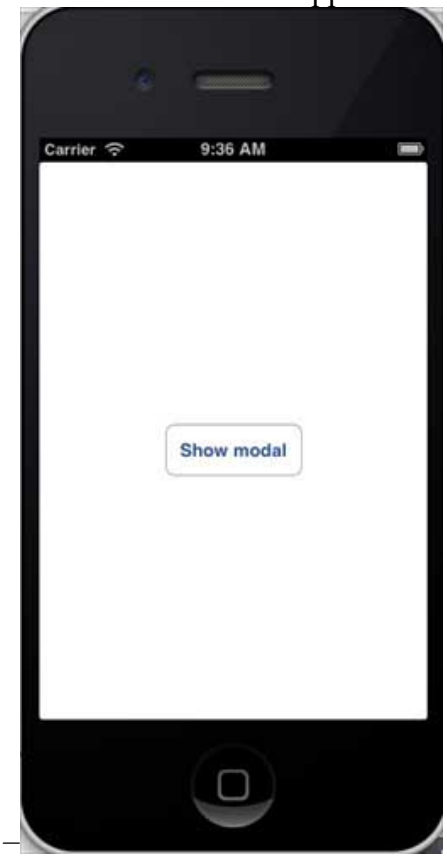
```
#import "ViewController.h"

@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 −

Framework,Design user interface for button,textview, textfield etc.

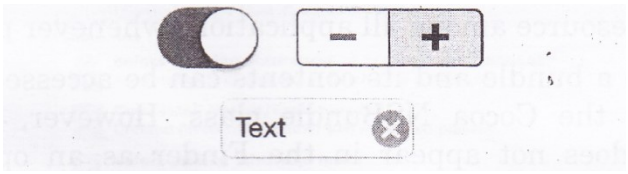
FRAMEWORK

- A framework is a hierarchical directory that encapsulates shared resources, such as dynamic shared library, xib files, image files, localized strings, header files, and reference documentation in a single package.
- Multiple applications can use all of these resources simultaneously.
- The system loads them into memory as needed and shares the one copy of the resource among all applications whenever possible.
- A framework is also a bundle and its contents can be accessed using Core Foundation Bundle Services or the Cocoa NSBundle class.
- A framework bundle is a standard directory that the user can navigate.
- Frameworks serve the same purpose as static and dynamic shared libraries, that is.
- They provide a library of routines that can be called by an application to perform a specific task. For example, the Application Kit and Foundation frameworks provide the programmatic interfaces for the Cocoa classes and methods.
- Frameworks offer the following advantages over static-linked libraries and other types of dynamic shared libraries:
  - → Frameworks group related resources together. This grouping makes it easier to install, uninstall, and locate those resources.
  - → Frameworks can include a wider variety of resource types than libraries. For example, a framework can include any relevant header files and documentation.
  - → Multiple versions of a framework can be included in the same bundle. This makes it possible to be backward compatible with older programs.
  - → Only one copy of a framework's read-only resources reside physically in-memory at any given time, regardless of how many processes are using those resources.

→ This sharing of resources reduces the memory footprint of the system and helps improve performance.

Controls:

- A control is a communication tool between a user and an app.
- Controls convey a particular action or intention to the app through user interaction, and can be used to manipulate content, provide user input, navigate within an app, or execute other pre-defined actions.
- Controls are simple, straightforward, find familiar to users because they appear throughout many iOS apps.
- The Ul Control class is the base class for all controls on iOS, and defines the functionality that is common to all controls.



Purpose: Controls allow users to:

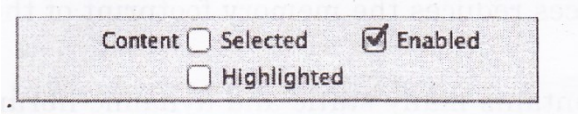
→ Interact with an app
→ Manipulate or edit app content
→ Convey user intent to the app in a straightforward way

BEHAVIOR OF CONTROLS

Control States:

- A control state describes the current interactive state of a control: normal, selected, enabled, or highlighted.
- A control can have more than one state at a time, and you can change a control's state at any point.
- When a user interacts with a control, the control's state changes appropriately.

Inspector:



- When a control is enabled, a user can interact with it. When a control is disabled, it appears grayed out and does not respond to user interaction. Controls are enabled by default;
- A control enters a temporary highlighted state when a touch enters and exits during tracking and when there is a touch up event.
- A highlighted state is temporary.
- You can customize the highlighted appearance of some controls, such as buttons. Controls are not highlighted by default; to set a control's initial state to highlighted, check the "Highlighted" (highlighted) box in the Attributes Inspector.
- When a user taps on a control, the control enters the selected state.

Control Events:

A control event represents various physical gestures that users can make on controls, such as lifting a finger from a control, dragging a finger into a control, and touching down within a text field.

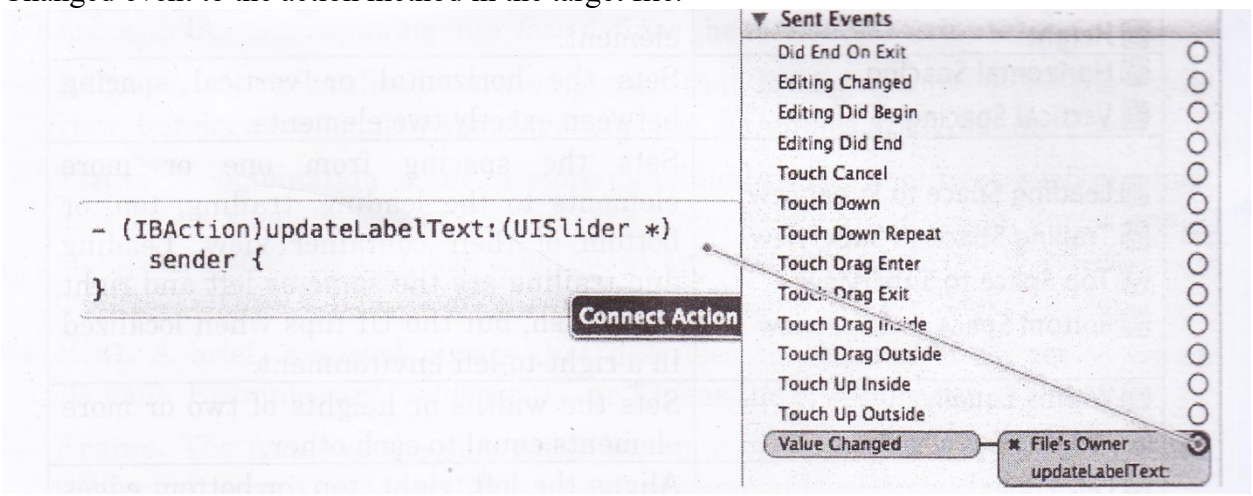
Target-Action Mechanism:

- The target-action mechanism is a model for configuring a control to send an action message to a target object after a specific control event.
- For example, when a user interacts with a slider, it generates a UlControlEvent Value Changed control event.
- You could use this event to update a label's text to the current value of the slider.
- In this case, the sender is the slider, the control event is Value Changed. The action is updating the label's text, and the target is the controller file containing the label as an IBOutlet.
- To create a relationship between the slider, the control event, the target, and the action, you can do one of two things:

[1] Call the addTarget:action:forControlEvents: method within your target file:

→ [self.mySlider addTarget:self
→ action:@selector (myAction:)
→ forControlEvents:UlControlEventValueChanged];

[2] Use the Connections Inspector in Interface Builder to Control-drag the slider's Value Changed event to the action method in the target file.



[3] Control-click the slider in Interface Builder, and drag its Value Changed event to the target object in your Storyboard. Select the appropriate action from the list of actions available for the target.

Using Auto Layout with Controls:

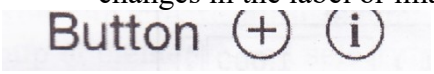
- The auto layout system allows you to define layout constraints for user interface elements, such as views and controls.
- You can create auto layout constraints by selecting the appropriate element or group of elements and selecting an option from the menu in the bottom right corner of Xcode Interface Builder.
- Auto layout contains two menus of constraints: pin and align.

- The Pin menu allows you to specify constraints that define some relationship based on a particular value or range of values. Some apply to the control itself (width) while others define relationships between elements (horizontal spacing). The following tables describe what each group of constraints in the auto layout menu accomplishes:

Constraint Name	Purpose
Width Height	Sets the width or height of a single element.
Horizontal Spacing Vertical Spacing	Sets the horizontal or vertical spacing between exactly two elements.
Leading Space to Superview Trailing Space to Superview Top Space to Superview Bottom Space to Superview	Sets the spacing from one or more elements to the leading, trailing, top, or bottom of their container view. Leading and trailing are the same as left and right in English, but the UI flips when localized in a right-to-left environment.
Widths Equally Heights Equally	Sets the widths or heights of two or more elements equal to each other.
Left Edges Right Edges Top Edges tom Edges	Aligns the left, right, top, or bottom edges of two or more elements.
Horizontal Centers Vertical Centers Baselines	Aligns two or more elements according to their horizontal centers, vertical centers, or bottom baselines. Note that baselines are different from bottom edges. These values may not be defined for certain elements.
Horizontal Center in Container Vertical Center in Container	Aligns the horizontal or vertical centers of one or more elements with the horizontal or vertical center of their container view.

BUTTONS

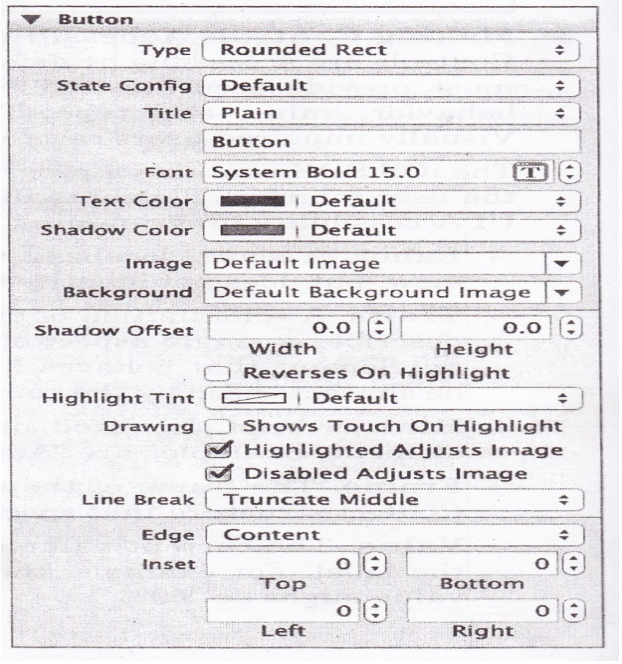
- Buttons let a user initiate behavior with a tap.
- You communicate a button's function through a textual label or with an image. Your app changes button appearance based upon user touch interactions, using highlighting, changes in the label or image, color, and state to indicate the button action dynamically.



→ Purpose : Buttons allow users to
→ Initiate behavior with a tap
→ Initiate an action in the app with a single simple gesture

Content of Buttons:

Set a button's content using the Type (button Type) field in the Attributes Inspector.
In iOS 7, the rounded button type has been deprecated in favor of the system button, There is also a custom type for great versatility in defining a unique interface.

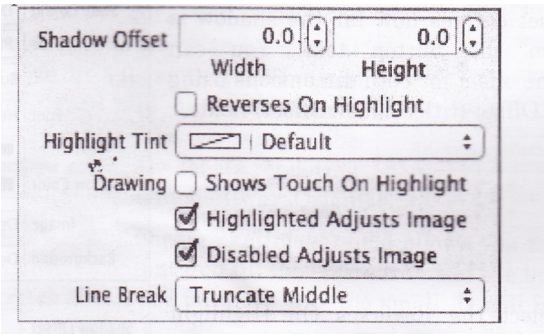


Behavior of Buttons:

- A button sends the UlConrrolEventTouchUplnside event when the user taps it.
- You can respond to this event by performing some corresponding action in your app, such as saving information.
- You register the target-action methods for a button as shown below.

```
[self.myButton addTarget: self action: @selector(myAction:)
                        forControlEvents :UlCdntrolEventValueChanged];
```

Alternatively, you can Control-drag the button's Value Changed event from the Connections Inspector to the action method.

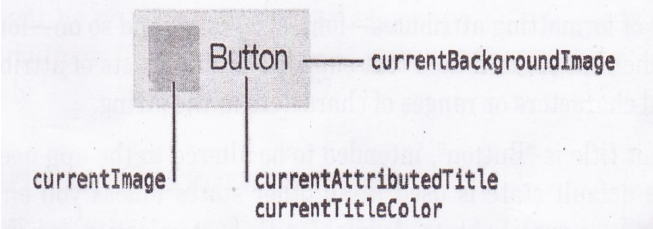


- If the Shows Touch On Highlight (showsTouchWhenHighlighted) box is enabled, when a user presses on the button, there will be a white glow where the touch event occurred on the button.
- If your button has a custom image, the Highlighted Adjusts Image (adjustslmageWhenHighlighted) and Disabled Adjusts Image (adjustslmageWhenDisabled) options allow you to specify whether highlighted or disabled states affect the appearance of the image.

- For example, with those options enabled, the image might get darker when the button is highlighted. and dimmer when the button is disabled.

Appearance of Buttons

You can customize the appearance of a button by setting the properties shown below.



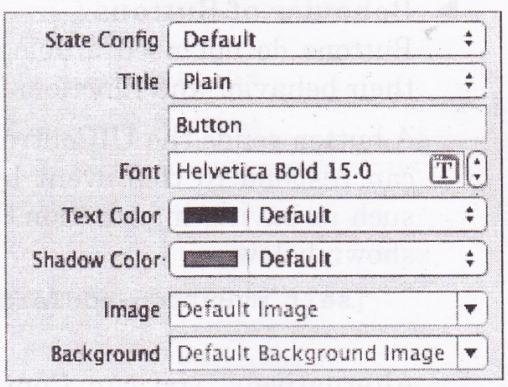
State:

A button has four states to configure for appearance—default. highlighted, elected, and disabled. To configure the button's appearance for each state, first select the state from the State Config menu in the Attributes Inspector and then use the other menus and text boxes in the Attributes Inspector's appearance property group.

Shadow: shadow offset define how Far the the shadow from the btn text you can customize the offset for the both dimensions using the shadow offset [Title Shadow offset] When Highlighted field.

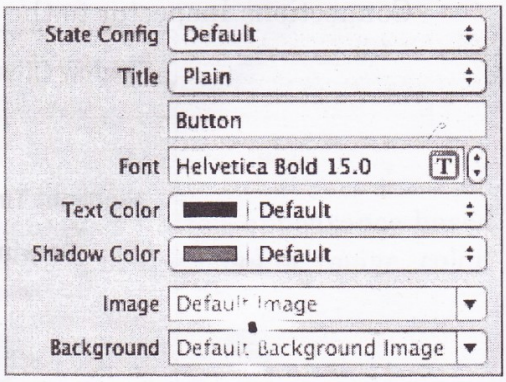
Tint Color:

- You can specify a custom button tint using the tintColor property.
- This property sets the color of the button image and text.
- If you do not explicitly set a tint color, the button will inherit its super view's tint color.



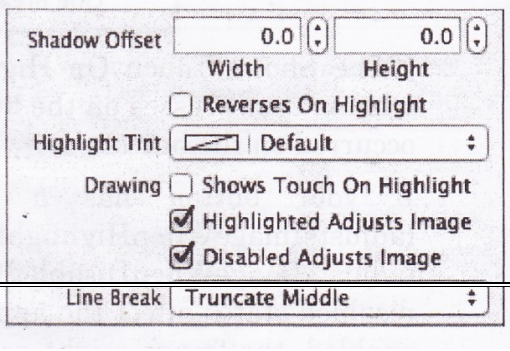
Title Attributes:

- Button can have one of two types of text: plain or attributed.
- Plain text supports a single set of formatting attributes—font, size, color, and so on—for the entire string. On the other hand, attributed text supports multiple sets of attributes that apply to individual characters or ranges of characters in the string.

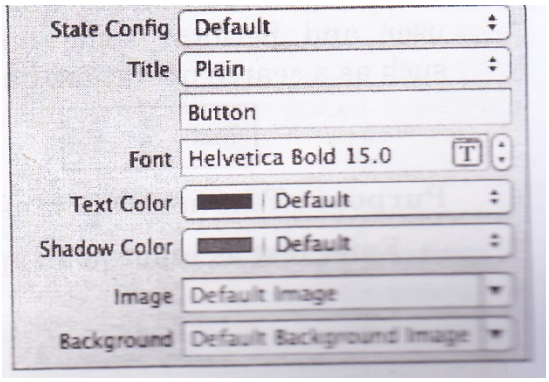


Images:

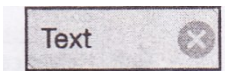
Using the Image (currentImage) field, you can specify an image to appear within the content of your button. If the button has a title, this image appears to the left of it, and centered otherwise. The image does not stretch or



compact, so make sure to select an image that is the proper size to appear in your button. Note that this image will be automatically rendered as a templat image within the button, unless you explicitly set its rendering mode to UIImageRenderingModel always original



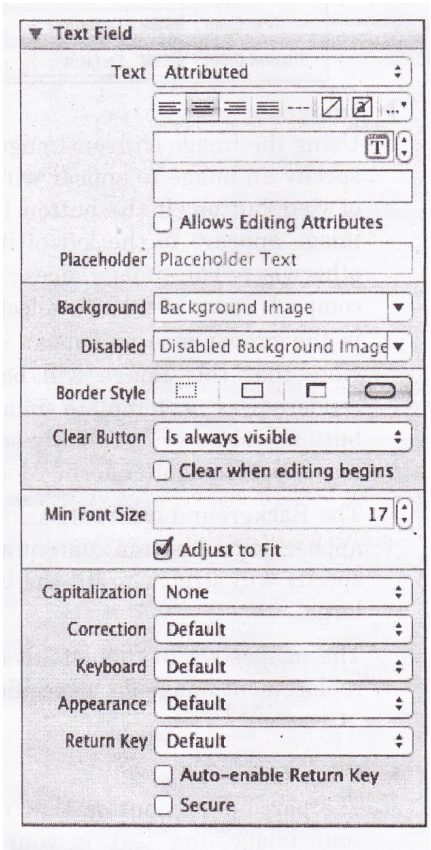
Text Fields:



Text fields allow the user to input a single line of text into an app. You typically use text fields to gather small amounts of text from the user and perform some immediate action, as a search operation, based on that text.

Purpose:

- Text fields allow users to
- Enter text as input to an app



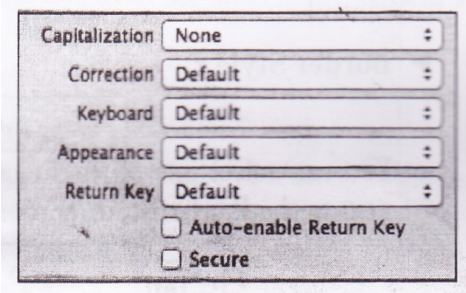
Content of Text Fields:

Set the content of the text field using the Text (text) field. You can select whether you want plain or attributed text. A user can use the Clear button to delete all text in the text field, and display the placeholder string if one is set.

Behavior of Text Fields:

- Text fields need a delegate to handle any custom behaviors, such as displaying additional overlay views when a user begins editing it.
- By assigning the parent view controller as the text fields delegate and implementing any or all of the UlTextFieldDelegate methods. you can implement such custom behaviors.

- Alternatively, you cart Control-drag the text field's Value Changed event from the Connections Inspector to the action method .A user types content into a text field using a keyboard, which has a number of customization options:



→ **Keyboard layout**. The Keyboard field allows you to select from a number of different keyboard layouts. If the user will be entering a web address, select the URL keyboard. The default keyboard layout is an alphanumeric keyboard in the device's default language.

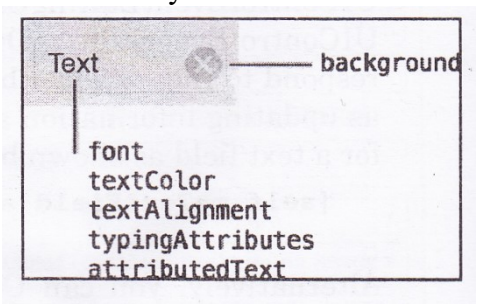
→ **Return key**. The return key, which appears in the bottom right of the keyboard. allows the user to notify the system when they are finished editing the text field.
Capitalization scheme. The Capitalization field specifies how text should be capitalized in the text field: no capitalization, every word, every sentence. or every character.

→ **Auto-correction.** The Correction field simply disables or enables auto-correct in the text field.

→ **Secure content**. The Secure option is off by default. Enabling it causes the text field to obscure text once it is typed, allowing the user to safely enter secure content—such as a password—into the field.

→ You can use the text field delegate methods to handle custom keyboard dismissal.

Appearance of Text Fields:



You can customize the appearance of a text field by setting the properties depicted below.

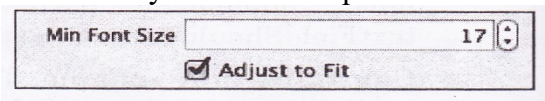
To customize the appearance of all text fields in your app, use the appearance proxy (for example, [UlTextField appearance]), or just of a single control.

Border Style:

You can select one of the fo1lowixg border styles for your text field by selecting it next to the Border Style (borderStyle) field:

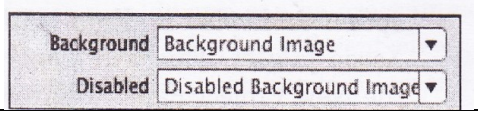
Font Size:

By default, the Adjusts to fit (adjustsFontSizeToFitWidth) box is selected in the Attributes Inspector. When this option is enabled, the font size of your text label will automatically scale to fit inside the text field. If you anticipate your text label to change—such as if the string is localized—you should keep this selected.



Images:

A text field can have background image that sits under the content of text field.



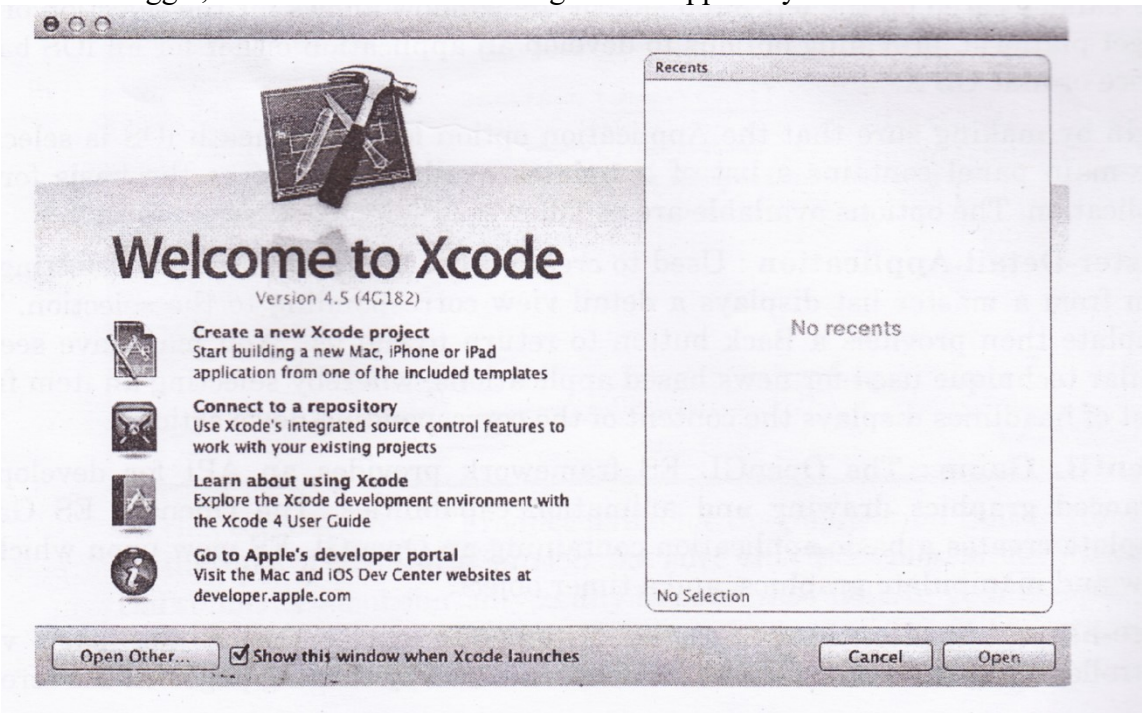
USING AUTO LAYOUT WITH TEXT FIELDS

- You can create Auto Layout constraints between a text field and other user interface elements. You can create any type of constraint for a text field.
- You will generally need to specify what a text field is intended for. You can use a label to do this. Place the label to the left of the text field and give the label and text field a "Horizontal Spacing" constraint.

Application development in X-Code

Starting Xcode 4:

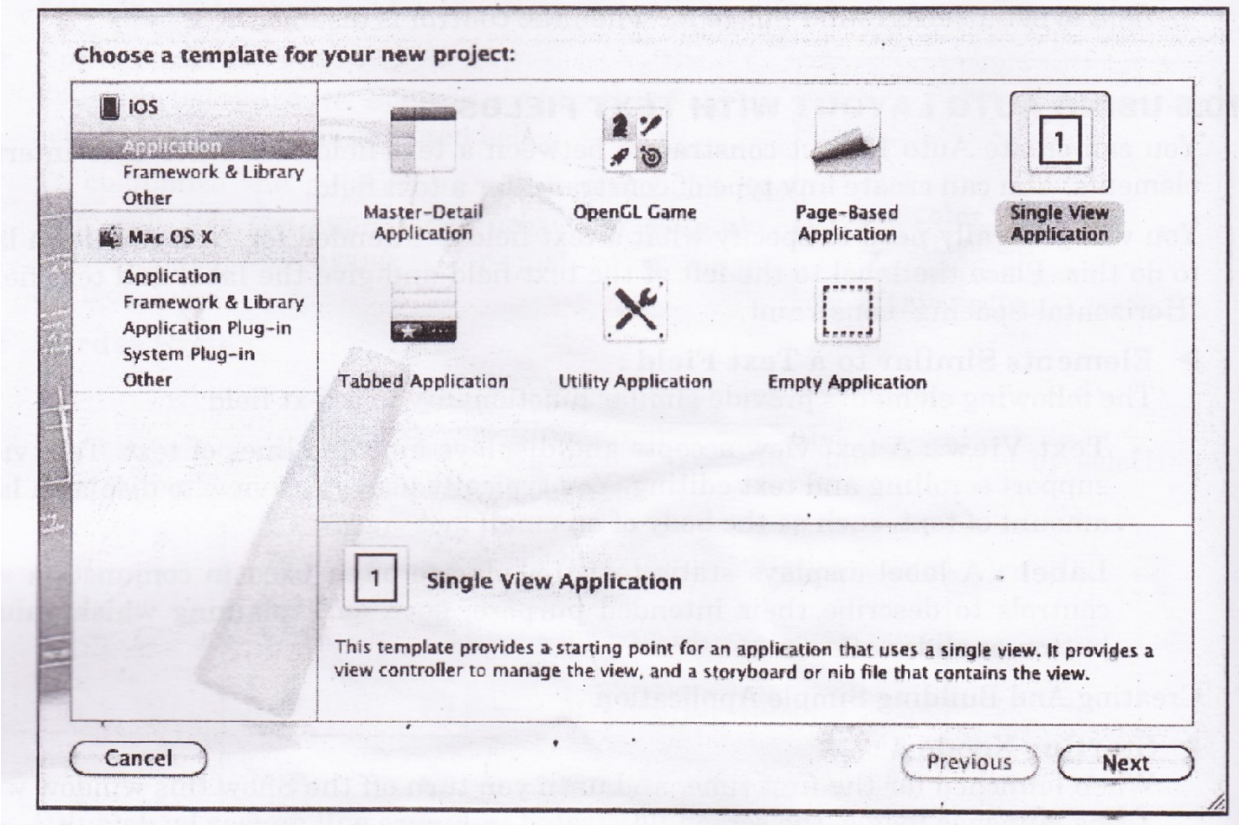
When launched for the first time, and until you turn off the Show this window when Xcode launches toggle, the screen illustrated in Figure will appear by default:



The panel located on the left hand side of the window allows for the selection of the target platform, providing options to develop an application either for an iOS based device or Mac OS X.

Begin by making sure that the Application option located beneath iOS is selected.

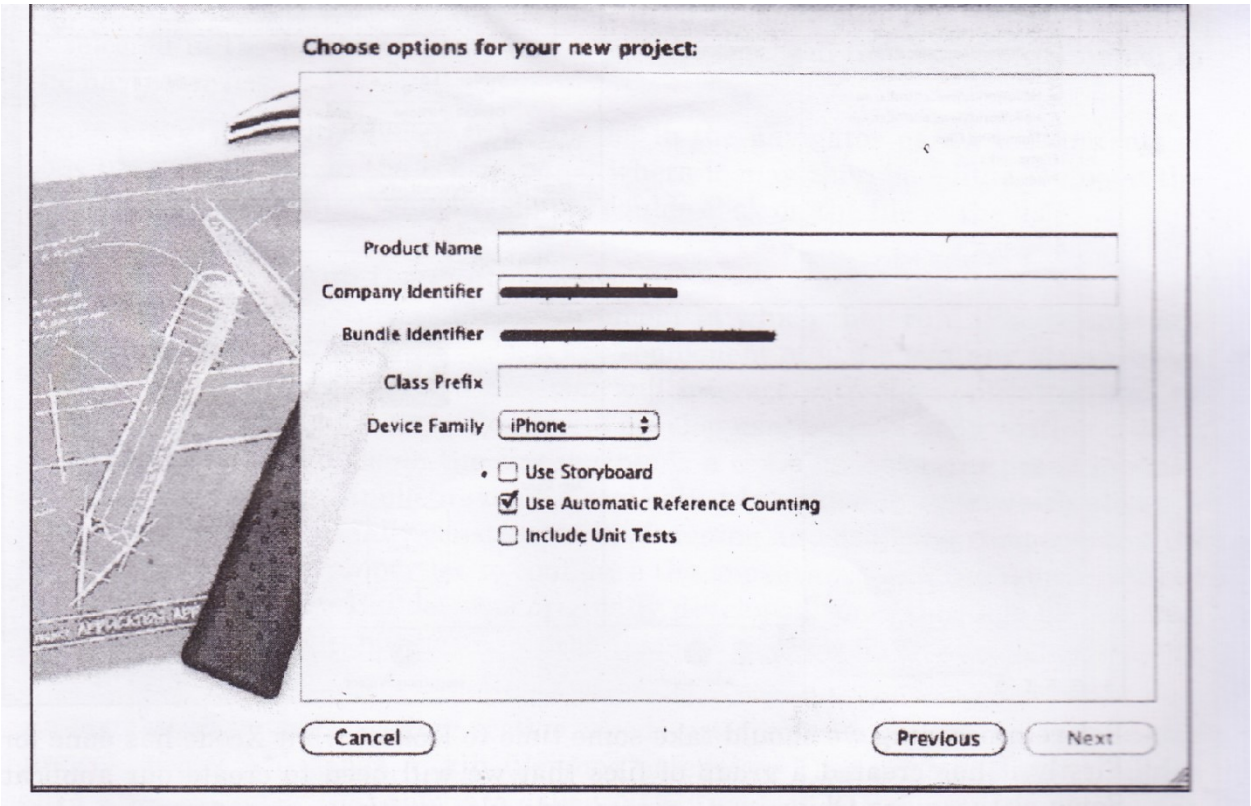
The main panel contains a list of templates available to use as the basis for an application. The options available are as follows:



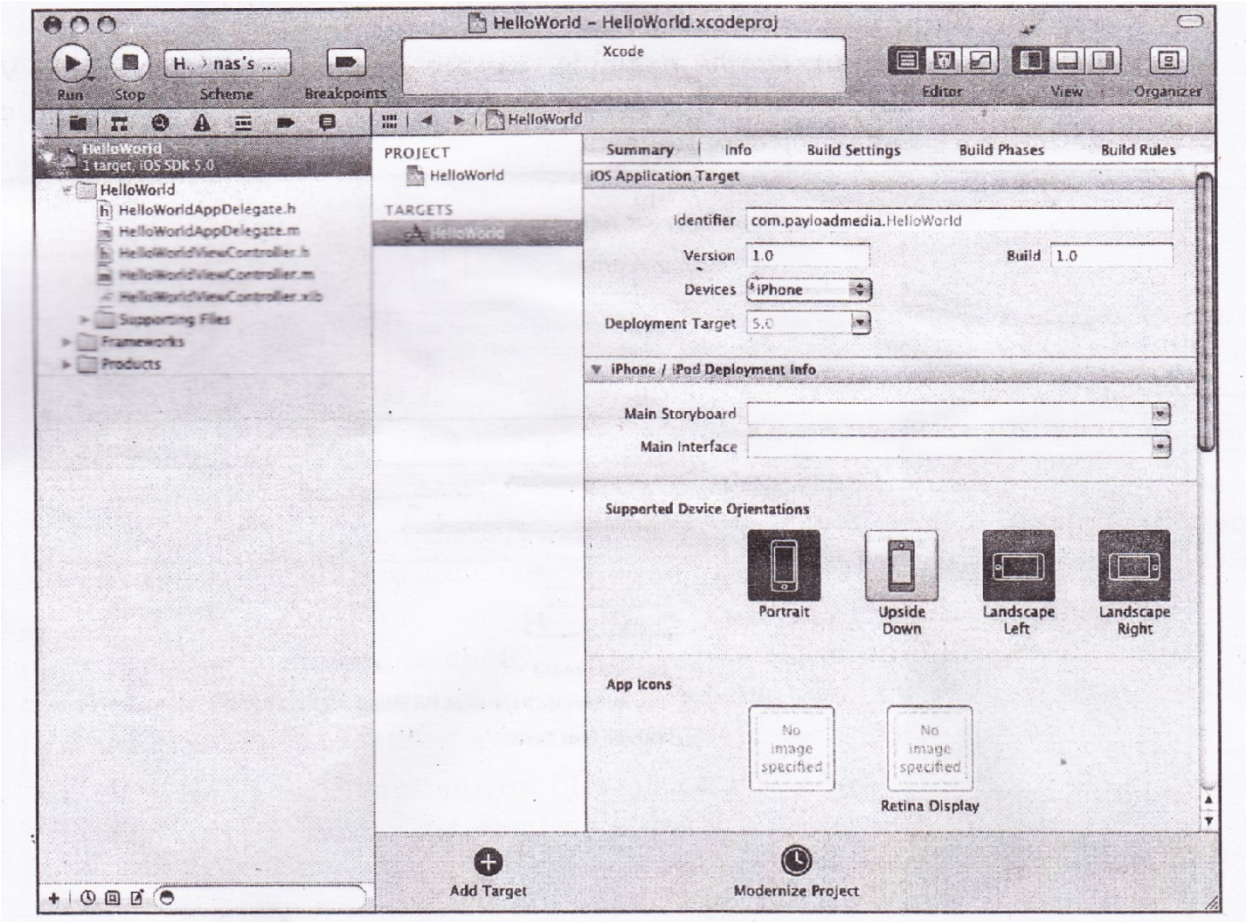
Master-Detail Application: Used to create a list based application. Selecting an item from a master list displays a detail view corresponding to the selection.

OpenGL Game: The openGL ES framework provides an API for developing advanced graphics drawing and animation capabilities. The OpenGL ES Game template creates a basic application containing an OpenGL ES view upon which to draw and manipulate graphics, and a timer object.

Page-based Application: Creates a template project using the page view controller designed to allow views to be transitioned by turning pages on the screen.



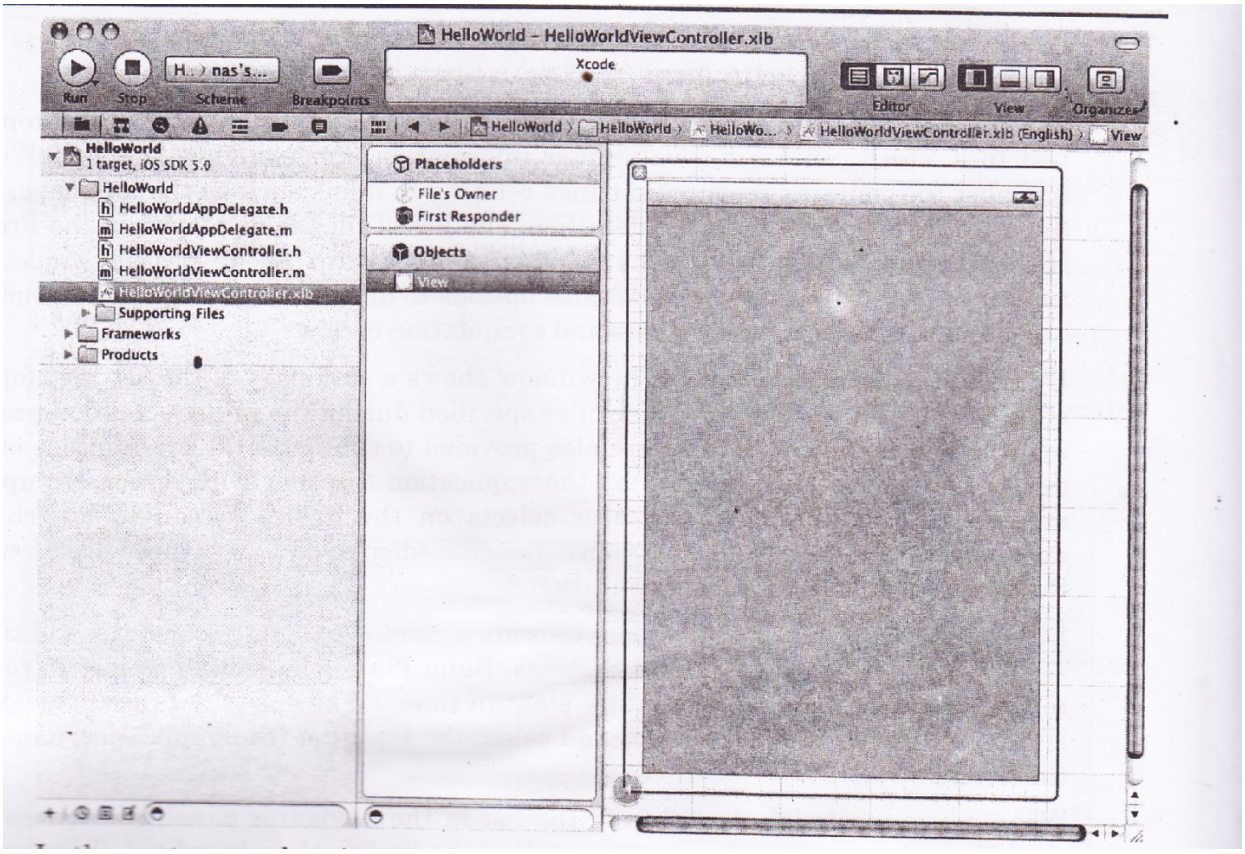
Make sure that iPhone is currently selected from the Devices menu and that neither the Use Storyboard nor the Include Unit Tests options are currently selected.
Automatic Reference Counting is a feature included with the Objective-C compiler which removes much of the responsibility from the developer for releasing objects when they are no longer needed. This is an extremely useful new feature and, as such, the option should be selected before clicking the Next button to proceed. On the final screen, choose a location on the file system for the new project to be created and click on Create.
Once the new project has been created, the main Xcode window will appear as illustrated in Figure:



Before proceeding we should take some time to look at what Xcode has done for us.
Firstly it has created a group of files that we will need to create our application. Some of these are Objective-C source code files (with a .m extension) where we will enter the code to make our application work, others are header or interface files (.h) that are included by the source files and are where we will also need to put our own declarations and definitions. In addition, the .xib file is the save file used by the Interface Builder tool to hold the user interface design we will create. Older versions of Interface builder saved designs in files with a .nib extension so these files, even today, are called as NIB file

Creating the iOS App User interface:

Simply by the very nature of the environment in which they run, iPhone apps are typically visually oriented. As such, a key component of just about any app involves a user interface through which the user will interact with the application and. in turn, receive feedback. As it is possible to develop user interfaces by writing code to create and position items on the screen, this is a complex and error prone process.
As mentioned in the preceding section, Xcode pre-created a number of files for our project, one of which has a .xib filename extension. This is an Interface Builder save file (remember that they are called NIB files, not XIB files). The file we are interested in for our HelloWorid project is called HelloWorldViewController.xib. To load this file into Interface Builder simply select the file name in the list in the left hand panel. Interface Builder will subsequently appear in the-centre panel as shown in Figure:



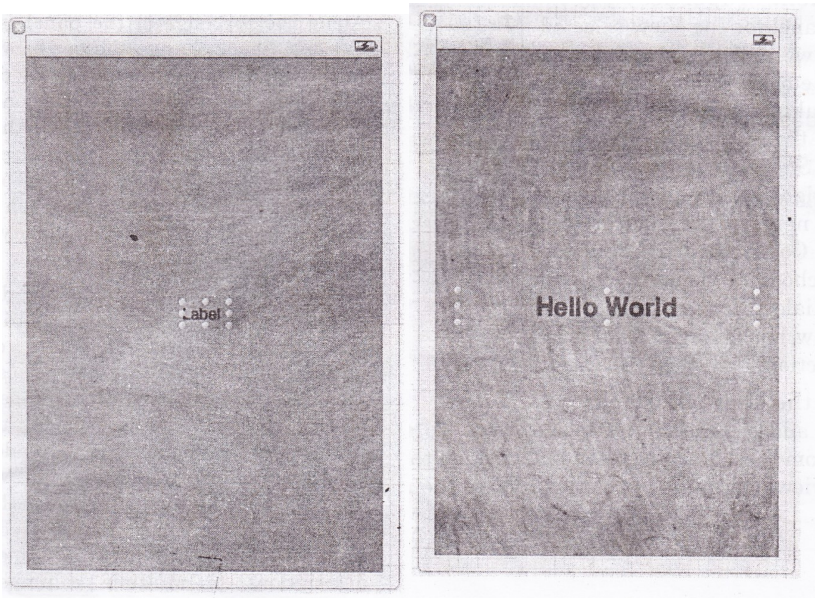
In the center panel a visual representation of the user interface of the application is displayed. Initially this consists only of the UlView object. This UlView object was added to our design by Xcode when we selected the Single View Application option during the project creation phase. We will construct the user interface for our Hello Word app by dragging and dropping user interface objects onto this UlView object. Designing a user interface consist primarily of dragging and dropping visual components onto the canvas and setting a range of properties and settings.

Changing Component Properties:
With the property panel for the View selected in the main panel, we will begin our design work by changing the background color of this view. Begin by making sure the View is selected and that the Attribute Inspector (View -> Utilities.> Show Attribute Inspector) is displayed in the right hand panel. Click on the gray rectangle next to the Background label to invoke the Colors dialog. Using the color selection tool, choose a visually pleasing color and close the dialog. You will now notice that the view window has changed from gray to the new color selection.

Adding Objects to the User Interface:
The next step is to add a Label object to our view. To achieve this, select Cocoa Touch -> Controls from the library panel menu, click on the Label object and drag it to the center of the view. Once it is in position release the mouse button to drop it at that location:

Using the resize marker surrounding the label border ,stretch first the left & the right side of the label out to the edge of the view until the vertical blue dotted line marking the recommend border of the view appear. With the label still selected ,clicked on the center alignment of button on the Attribute Inspector [view>utilities>show attribute Inspector]
To center the next in the middle of the screen.

To achieve this, select File -> Save or use the Comrnancl+S keyboard shortcut.



Building and Running an IOS App

Before an app can first be compiled. Once success compiled it may be run either within simulator or on a physical iPhone. iPad or ipod Touch device The process for testing a physical

Within the main Xcode project window, make sure that the menu located in the top left hand corner of the window to the right of the Stop button) has the iPhone 6.0 Simulator option selected and then click on the Run toolbar button to compile the code and run the app in the simulator. The small iTunes style window in the center of the Xcode toolbar will report the progress of the build process together with any problems or errors that cause the build process to fail. Once the app is built, the simulator will start and the Hello Word app will run:

COCOA TOUCH

Definition - What does Cocoa Touch mean?

- Cocoa Touch is a user interface framework provided by Apple for building of
- Applications for products like iPhone, iPad and iPod Touch.
- It is primarily written in Objective C language and is based on Mac OS X, Cocoa Touch was develop based on model view controller software architecture.
- The high-level application programming interfaces available in Cocoa Touch help to make animation, networking, and adding the appearance and behavior of the native platform.
- the developed applications possible with less code development.

The main features of Cocoa Touch include:

→ Core Animation: Helps to create rich user experiences by allowing for the smooth movement of visual elements. It also fills in the interim frames of animation with automatic timing and adjustment.

→ Core Audio

→ Core Data: Provides an object-oriented data management solution and aids in defining an application's data model in a logical

→ Cocoa Touch is made up of several frameworks, but the key ones are:

→ Audio and Video:.
Core Audio
OpnAL
Media Library
AV Foundation

→ Data Management:
Core Data
SQLite

→ Graphics and Animatidn:
Core Animation
OpenGLES
Quartz 2D

→ Networking and Internet:
Bonjour
BSD Sockets

→ User Applications:
Address Book
Core Location
Map Kit
Store Kit

MVC - INTRODUCTION

As implied with the name, the MVC software design pattern refers to three separate roles: the model, the view and the controller.
The idea is that the objects in your application will take on these roles and work together to create and manage the user interface. Not all objects will assume one of these 3 roles; For example you might have a class that contains a bunch of helper methods however, the model, view and controller roles are the central actors to making your app function.

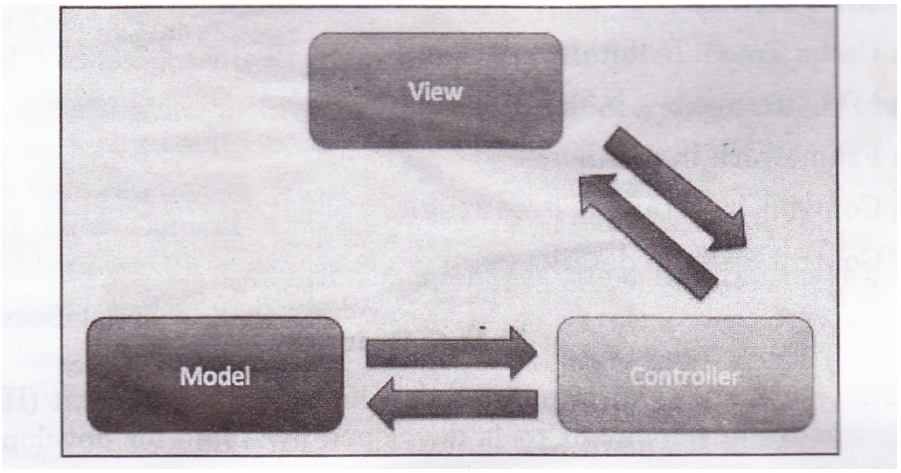
The Model : The model represents the data in your application. It's responsible for sorting, validating and organizing your data.

The View : The view is the user interface or what the user sees and interacts with. You can create views programmatically through code using Apple classes such as
UI View or you can create a XIB file to represent the view and visually layout your elements through Interface Builder.

The controller manages the communication between the view and the model and communicates it to the view for display .In the same layer, the controller also takes the changed data (due to user interaction or something else) and it back to the model.

The Model and View Never Talk To Each Other
They shouldn't even know each other in the ideal case. The benefit of this modular architecture is separation of roles allows us to make modifications easily with less bugs.

For example, if in the future. you need to make a change to the way the data is fetched or organized. All you need to do is switch out the model. As long as you keep the interface of the model the same (the header file), then the views, and controllers will be none the wiser.
Here's a diagram illustrating what we've talked about:



Its commbinition of
.h(Decelaration)
.m(coding)
.Xib(nib)(desiging)