4564646

iPhone Application Development

A Beginner's Guide

Type to enter text

Radha Krishna Nalla

Rapid Technologies

www.rapidtechnologies.org

Skype : rnalla346

Email : krishna706@gmail.com

C Language

```
#import <UIKit/UIKit.h>
    #import "cwork.h"
    int main(int argc, char *argv[]) {
     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
     sayHello();
     int retVal = UIApplicationMain(argc, argv, nil, nil);
     [pool release];
     return retVal;
    }
```

Data Types and Operators

Data Types : char short int long float double

Operators != > < >= <= ! && ||

Control, Functions, and Conditional Statements

```
if(myInt < 2) {
     printf("the value is not equal");
     myOtherInt = 3;
   }
   else if (myInt == 5) {
     myOtherInt = 7;
   }
   else {
     myOtherInt = 3;
}
```

Switch

```
switch (myInt) {
     case 1:
       myOtherInt = 3;
       printf("case one");
       break;
     case 2:
       myOtherInt = 5;
       break;
     default:
       myOtherInt = 4;
}
```

C Comments

C's comment syntax is the same as Java's. A double forward slash indicates everything following on the same line is a comment. A slash followed by an asterisk indicates everything following until the next asterisk and slash is a comment. For instance, the following are both comments. The first is a single-line comment, while the second is a multiline comment.

```
    // This is a comment in C code.
    /* This is a multiline comment in C Code.
    This comment can be multiple lines. */
```

Comments are, of course, ignored by the compiler, and are ways for you to provide code explanation for future programmers who might debug or modify your code.

Arrays and Structures

C arrays are similar to Java arrays. You declare arrays the same, but C has no new keyword; you simply start using the array

```
int myArray[100];
  myArray[0] = 1;
  myArray[1] = 2;
```

C has structs; Java doesn't have a struct data type. In C, a struct is similar to a class, but has no methods or inheritance

```
struct myBox {
    int length;
    int width;
}
```

Arrays can hold structures; for instance, you might declare an array to hold 100 myBox instances

```
struct myBox myBoxes[100];
  myBoxes[0].length = 10;
  myBoxes[0].width = 2;
```

Functions

You declare functions the same way using C as you do using Java, only C does not restrict a function's visibility. C has no public or private functions. Like Java, you declare a C function with a return type, a name, and argument list. You write function declarations in header files. You write function definitions in C source files. Functions that don't return anything use void as their return type. If a function takes no arguments, you can optionally list the arguments as void.

```
  void sayHello(void);
```

```
static void sayHello(void){ printf("hello\n");}
```

Note that you don't declare the static method's prototype in a header file. You simply write the method in the source file using the method.

The Printf Method

C uses the printf function for outputting to the standard output stream. Its declaration is as

follows:

```
  int printf( const char *format, arg1, arg2, ..., argn);
```

The function takes a pointer to the characters you wish sending to the standard output stream and zero or more items for formatting. For instance consider the following printf statement.

```
  printf("Hello world...%d times", 22);
```

This statement results in the following output.

```
  Hello world...22 times
```

Another common argument is %s for character strings. For instance, the following code illustrates a character array and then prints it.

```
  char * hello = "hello turkey";
  printf("%s\n", hello);
```

Pointers

Java does away with pointers; however, Objective-C relies extensively upon pointers. A pointer is a reference to another variable, or more technically, a pointer is a variable that references another variable's memory space.

You indicate pointers using the asterisk (*). Pointers point to a location in memory of another variable. The ampersand (&) indicates a variable's address in memory.

```
#include <stdio.h>
    int main (int argc, const char * argv[]) {
      int avalue = 10;
      int *pavalue = &avalue;
      printf("address:%p value:%d", pavalue, *pavalue);
      return 0;
}
```

iPhone App Development

List Of Smart Phones:

1.Android	-- Android OS
2.Black Berry	-- Black Berry
3.Palm Pre	-- Palm
4.Simbian	-- Simbian OS
5.Windows	-- Windows 8
6.Apple iPhone	-- iOS 7.0

iPhone:

iPhone versions : 2G,3G,3GS,4G,4S,5,5S,5C

The Latest iPhone 5S having A7 Processor,8Mega Pixel CAM,
64GB hard Disk,Siri Features.

And iPhone Operating System Supports Multi tasking,Multi mail System,iCloud,Directories creation for the Apps,printing features.

320*480 pixels 3.5 inches —> iphone 4,4S
320*568 pixels 4 inches —> iPhone 5
768*1024 pixels 9.5 inches —> iPad

Requirements for Doing an iPhone App:

Operating System	: MAC OS 10.7 and above(Present Version -10.9)
IDE	: XCODE (Latest version - 5)
Programming language	: Objective C 2.0
Designing Tool	: Interface Builder(IB)
Testing Tool	: iOS Simulator/Device
Data Base	: SQLite
Peformence Tool	: Instruments
Frame Works	: Cocoa Touch Frame Work

MAC OS (Macintosh Operating System)

Company	Apple
Programed in	C,C++,Objective C
Latest Release	10.9(Maverics)
OS Family	UNIX

MAC versions:

10.0	Cheetah
10.1	Puma
10.2	Jaguar
10.3	Panther
10.4	Tiger
10.5	Leopard
10.6	Snow Leopard
10.7	Lion
10.8	Mountain Lion

10.9 — Maverics

MAC 10.6 Features:

1.32 bit to 64 bit Transition:

MAC OS Shifts single version compatibility with 32 bit and 64 bit.

2.GCD(Grand Central Dispatch)

It is a revolutionary new system wide technology.In MAC OS dynamically scales Application to take full advantage of Multi Core processor.MAC is using single programming model for these.It improves performance efficiency and responsiveness of the software.

3.Open CL

This Language is written in C.Apple uses vast computing power of modern Graphical Processor Unit.By using GPU system operations become very fast.

4.Quick Time X

Snow Leopard introduces Quick time player X.It is a major forward that advantages of modern media.

MAC OS 10.7 Features:

1.LaunchPad
2.machine Control
3.Finder Search
4.Multi gestures
5.Resume Windows
6.Account Switching
7.Safari Reader Button
8.Photo Booth
9.Face Time
10.iCloud 5GB
11.air drop

MAC OS 10.8 Features:

1.Notifications
2.iMessages

3.Notes
4.Reminders
5.Dictation
6.Safari Changes

MAC OS 10.9 Features:

1.iBooks
2.Maps
3.Calendar
4.Safari
5.iCloud Keychain
6.Multiple Displays
7.Notifications
8.Finder tabs
9.Tags

——

iOS :

iOS is nothing but iPhone Operating system.Works for only Apple mobile Devices like iPhone,iPad,iPod touch.And Latest version is iOS 7.

iOS 5 Features:

1.Status Bar Notifications
2.Twitter Support
3.News Stand
4.Safari reader button for iPad
5.iCloud
6.iMessages
7.Reminder App
8.Camera Button
9.Photo Editing
10.Siri
And 200 more new features.

iOS 6 Features:

1.Maps
2.Call
3.Pass Book
4.Siri Features
5.Mails-VIP,new drag refresh button

6.FB Share

iOS 7 Features:

1.Controller Center
2.New UI
3.Air Drop
4.New Search

iPhone Application Development Introduction:

Development of iOS Apps is pleasant and good.To convert your ideas into products we should use XCODE IDE(Integrated Development Environment) and Compiler is gcc 4.x

With XCODE we can organize, Edit your Source Files, View Documentation,Build your Applications and optimize your Application performance.

Note:

1.To Develop iOS Applications we must be registered as Apple Developer

2.To Run the App on Device or To send the App to App Store we must be a member of iPhone Developer Program.

Standard Edition:

1year ---99$ ---100 Devices --Download Videos,Execute on Device,Send Apps to App Store

Enterprise Edition

1 year --299$ --->500 Devices --Download Videos,Execute on Device,Send Apps to App Store

Apple Site:

www.developer.apple.com

Apple Site:

www.apple.com

iTunes Store:

www.itunesconnect.apple.com

Note:

We can give our own rate to the App.And 70% comes to developer and 30% for Apple on each Download.We have to give the bank details to Apple on www.itunesconnect.apple.com.

Every months 1st week we got Amount for downloads.

iOS Application Development Process:

1.Creating Your XCODE Project
2.Design UI
3.Writing Code
4.Run and Debug Your Application
5.Measuring Your Application performence

1.Creating Your XCODE Project:

iPhone SDK(Software Development Kit) -- 4.8GB

1.Navigation Based Application

An Application that presents data hierarchy using multiple screens.
eg: Settings Application.

2.Utility Based

Application that accepts main view and let the users accept flip side view to perform simple customzations.
e.g.:Stock Application.

3.Split View Based App

This Template is for iPad.If the App having more than one view on screen we use this Template.

4.Tab bar Application

Application that presents radio interface that let the users choose from multiple screens.
Eg: App Store

5.open GL

An Application that uses open GL Based view to present images or animations.

6.View based App

An Application that uses single view to implement user interface.

7.Window Based App

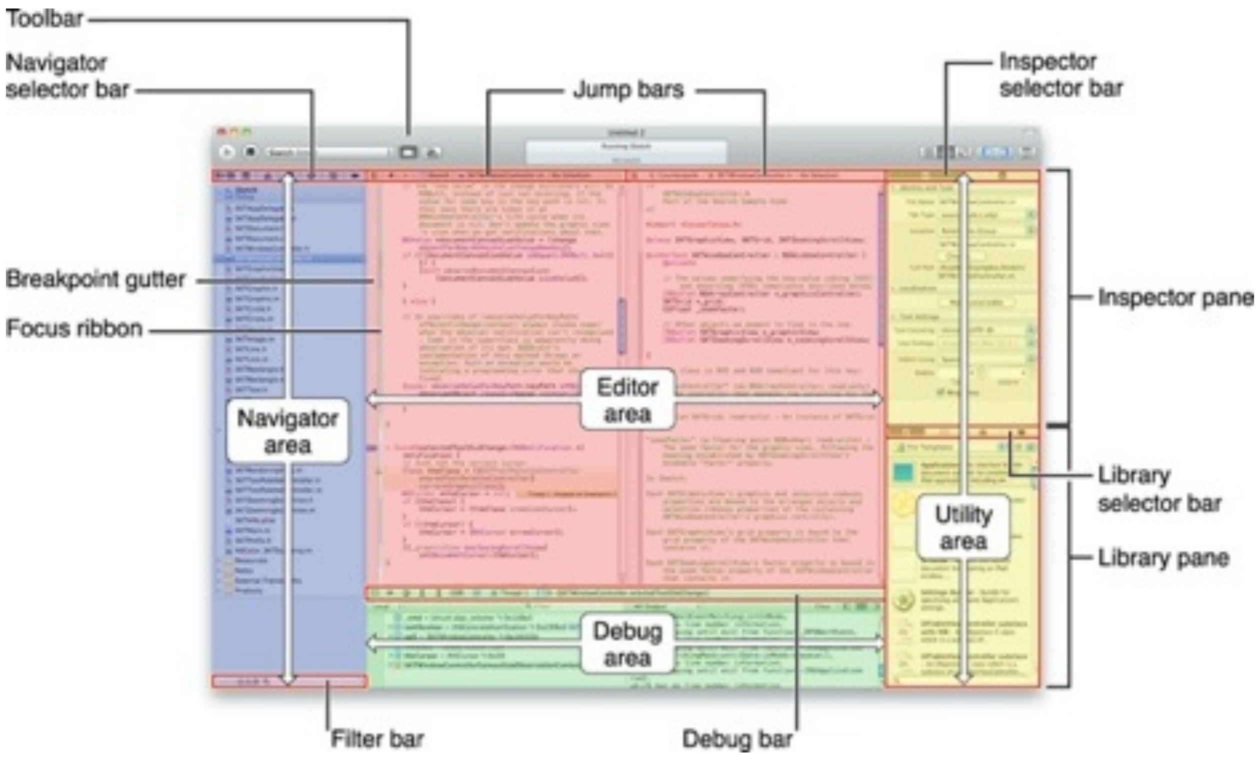
This Template serves as starting point for any Application.When you want to implement your own view hierarchy we use this template.

Q.How to open XCODE Project?

Ans.Goto spot light -->Type XCODE -->Enter --->File --->New -->Project --->Select iOS -->Select Template -->Give Location and name -->Enter.

XCODE:

Xcode is the integrated development environment (IDE) designed for developing iOS and Mac apps. The Xcode IDE includes editors used to design and implement your app, such as a source code editor and a user interface editor. Xcode also supports multiperson development using source control management (SCM) systems. As you write source code, Xcode can show you mistakes in both syntax and logic, and even suggests fixes.



Xcode has many features to make your job easier.

- **Single-window interface.** You perform most of your development workflows in one window (you can have multiple workspace windows and multiple tabs per window).
- **Graphical user interface design.** With the Xcode user interface editor, called Interface Builder, you specify most of the details of your app's user interface, such as the layout of the user interface controls, and their connection to your app's business logic and the data it manages, using a powerful and intuitive graphical user interface. Interface Builder works closely with other editors, such as the source code editor, to take you from design to implementation as fast as possible.
- **Assisted editing.** When you need to work on different aspects of the same component, such as user interface layout and the implementation of the user interface functionality, you can use multiple editors that open the content you need when you need it. For example, when you work on an implementation file in the primary editor, Xcode can open the corresponding header file in a secondary editor pane.
- **Automatic error identification and correction.** Xcode checks the source code you type as you type

it. When Xcode notices a mistake, the source code editor highlights it. You can then find out details about the error and ask Xcode to fix it for you.

- **Source control.** You can safeguard all your project files in Git and Subversion source code repositories.

Note that every element in the path in the jump bar is a pop-up menu that you can use to navigate through your project. Hold down the Option key when selecting a file in the jump bar to open Assistant and display the file in the assistant editor pane (you can change this behavior in the General pane of Xcode preferences).

Groups And File Section:

In XCODE 4.x overall 3 groups are present.

1.Group1(Project Name)

a.

AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m

ViewController.xib

b.Supporting Files

info.plist

main.m

.pch

2.FrameWorks

Collection of classes is nothing but Frame Work.in XCODE by default 3 frameworks are already added to the iPhone Project.

1.Foundation Frame Work(All classes starts with NS)
2.UIKit Framework(All classes starts with UI)
3.Core Graphics Frame work(Starts with CG)

3.Products

projectName.app

The Final out put is in .app format.We convert into .ipa and sync with provisioned iOS Devices.And we can also send .app file to App Store with iTunes Account.

info.plist:

plist is nothing but property list.it contains entire project information.Its like key value pair.

e.g: key	value
bundle display name	abc
icon file	1.png
version	1.0

main.m:

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

int main(int argc, char *argv[])
{
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil,
NSStringFromClass([AppDelegate class]));
    }
}
```

.pch File(Pre Compiled header)

```
#ifdef __OBJC__
    #import <UIKit/UIKit.h>
    #import <Foundation/Foundation.h>
#endif
```

Organizer:

1.Devices

Shows the List of connected Devices and console.All the Provisioning Profiles information available here.

2.Repositories

We can save our code safely here.Multiple embers can work on same project using Repositories.

3.Projects

It will show the List of projects we opened.

4.Archives

Archives having the list of .ipa Files.From here we can distribute our App to App store or else we can do AdHoc Distribution.

5.Documentation

Here You can find Documentation for All the API and classes and method. XCODe Automatically install the Documentation library after installation of XCODE.

Interface Builder (IB)

Interface Builder was developed in 1988,It is used for designing the Applications for NEXT STEP,OPEN STEP,MAC OS,iOS.IB Supports 2 formats

.nib
.xib

IB Has 4 Windows:
1.File's Owner Window
2.Inspector
3.Library
4.View

1.File's Owner Window:
File owner having the copy of entire nib file.And it is always be the first icon in any xib file.It has 3 icons

1.File's Owner

2.First Responder
3.View

2.Inspector

Inspector is used for setting the properties of View Elements.Inspector having 6 icons.

1.File Identity
2.Quick help
3.CLass Identity
4.Attribute Inspector
5.Size Inspector
6.Connections

3.View

View is presentation layer to there user.And It has 2 modes portrait and Land scape.
for iPhone -320*480 pixels View.
for iPad - 768*1024 pixels View

4.Library

UIApplication

Every iPhone application has one UIApplication. UIApplication is an iPhone application's starting point and is responsible for initializing and displaying your application's UIWindow. It is also responsible for loading your application's first UIView into the UIWindow. Another

responsibility UIApplication has is managing your application's life cycle. UIApplication fulfils this management responsibility using a delegate called UIApplicationDelegate. Although UIApplication receives events, it's the UIApplicationDelegate that handles how an application responds to those events. Events the UIApplicationDelegate might handle include application life cycle events, such as startup and shutdown, and system events like incoming phone calls and calendar alerts

----

MVC (Model View Controller)

In 1970's small talk define an architecture called MVC.Cocoa Framework used MVC.The Main use of this

Architecture is to separate Model and View.Objective C follows MVC.

M -- Model ---- Writing Code .h and .m
V -- View ---- Design .xib
C -- Controller -- It connects Model and View

Reasons for using MVC:
They are reusable,And when the problem occurs just follow the pattern and adopt it necessary.They Are Expressive,By using MVC applications become more expressive.

Model:
It consists of set of classes present in the XCODE and does not know about out side the world.

View:
It's like presentation layer to the user.

Controller:
By using Controller we can connect the Model and View.

Delegation

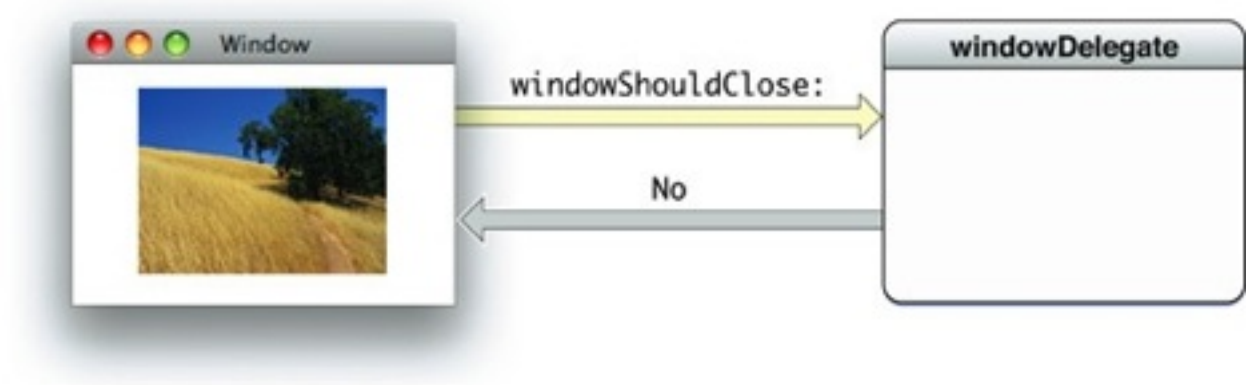
Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object.

Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object. The delegating object keeps a reference to the other object—the delegate—and at the appropriate time sends a message to it. The message informs the delegate of an event that the delegating object is about to handle or has just handled. The delegate may respond to the message by updating the appearance or state of itself or other objects in the application, and in some cases it can return a value that affects how an impending event is handled. The main value of delegation is that it allows you to easily customize the behavior of several objects in one central object.

Delegation and the Cocoa Frameworks

The delegating object is typically a framework object, and the delegate is typically a custom controller object. In a managed memory environment, the delegating object maintains a weak reference to its delegate; in a garbage-collected environment, the receiver maintains a strong reference to its delegate. Examples of delegation abound in the Foundation, UIKit, AppKit, and other Cocoa and Cocoa Touch frameworks.

An example of a delegating object is an instance of the NSWindow class of the AppKit framework. NSWindow declares a protocol, among whose methods is windowShouldClose:. When a user clicks the close box in a window, the window object sends windowShouldClose: to its delegate to ask it to confirm the closure of the window. The delegate returns a Boolean value, thereby controlling the behavior of the window object.

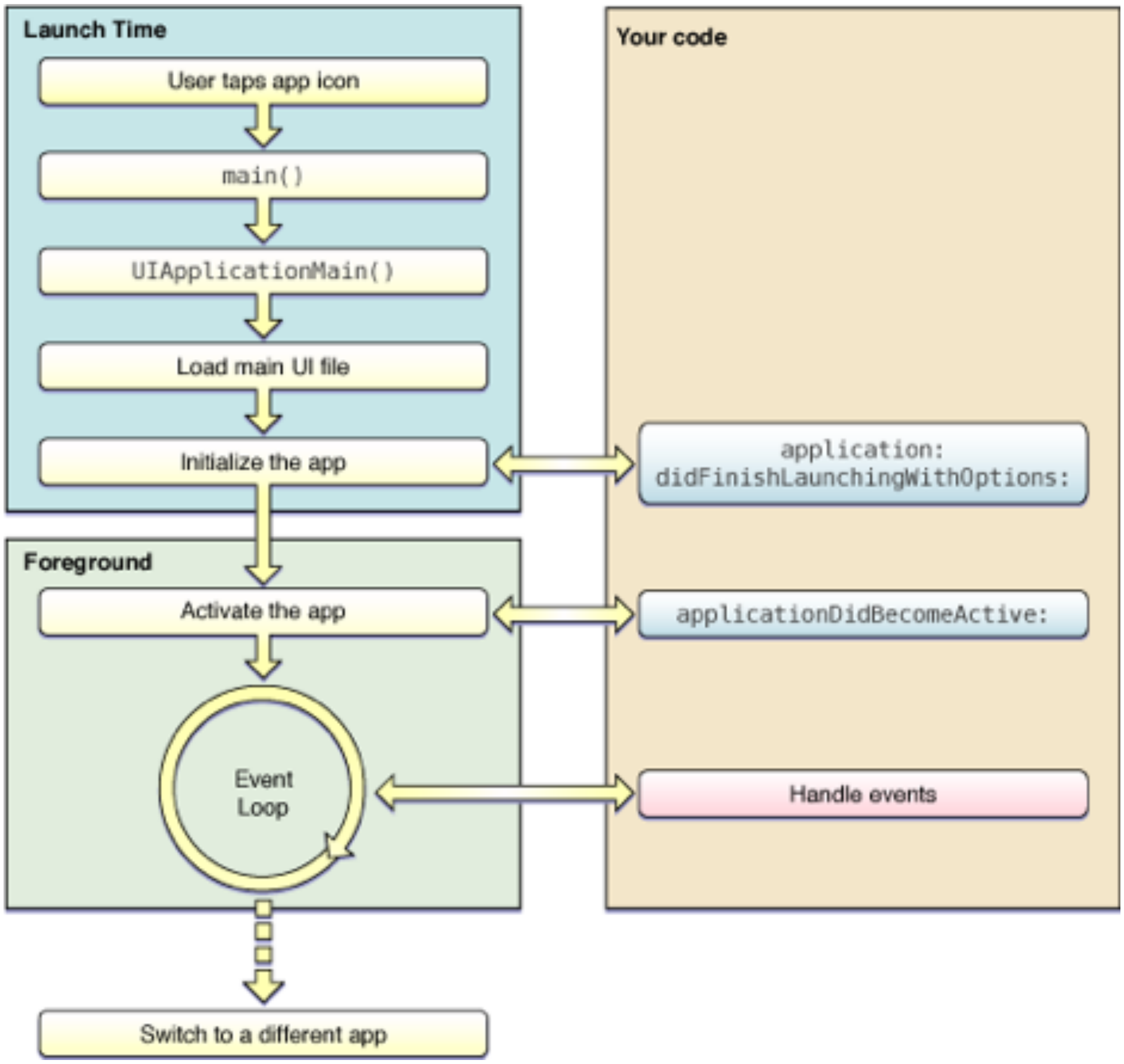


The App Life Cycle

When your app is launched, it moves from the not running state to the active or background state, transitioning briefly through the inactive state. As part of the launch cycle, the system creates a process and main thread for your app and calls your app's main function on that main thread. The default main function that comes with your Xcode project promptly hands control over to the UIKit framework, which does most of the work in initializing your app and preparing it to run.

the sequence of events that occurs when an app is launched into the foreground, including the app delegate methods that are called.

Launching an app into the foreground



If your app is launched into the background instead—usually to handle some type of background event—the launch cycle changes slightly to the one shown in Figure 3-3. The main difference is that instead of your app being made active, it enters the background state to handle the event and then is suspended shortly afterward. When launching into the background, the system still loads your app's user interface files but it does not display the app's window.

background, the property contains the value UIApplicationStateBackground instead. You can use this difference to adjust the launch-time behavior of your application:didFinishLaunchingWithOptions: method accordingly.

Objective C

Objective C is a Very old Programming language.It was designed and developed in 1980's.Now objective C become popular once again as it is being selected by Apple to develop Applications for MAC OS X and iOS.

Objective C is designed by **Brad Cox** in his company STEP STONE corporation.It was designed to enable powerful feature of Object Oriented Programming language.It works as powerful set of extinction to the C language.Objective C takes best features from C and Small talk.it is easy to learn and has fully objective Oriented capabilities.Apple selected Objective C as primary programming language for MAC and iPhone.

Like an Objective Oriented language Objective C revolves around objects.And it has 3 parts

1.Interface

Interface of a class is generally defined in header file suffixed .h is the declaration of a class.

2.Implementation

Actual code is written in implementation of a class generally defined in suffixed .m

3.Instantiation

After declaring and defining of a class It can be instantiated by allocating memory to the new object of the class

Q.Why we use Objective C?

It is a powerful language and easy to learn and Objective oriented version of C.It provides Dynamic Binding,easy to understand the code and well organised language.That's why objective C is selected for Cocoa Frame work.

Features of Objective C:

1.Class/Object
2.Properties
3.Protocol
4.Categories
5.Selectors
6.Exception handling
7.Threading
8.Messaging
9.Memory Management
10.id/Dynamic Binding/over riding
11.inheritance/enum

Objective C Keywords:

Objective C has some additional keywords, to avoid conflict with other languages it uses @ at the beginning of the each keyword.

List of Keywords

1.@interface
2.@implementation
3.@end

4.@synthesize
5.@protocol
6.@property

Access Specifiers
@private
@protected
@public

Exception Handling Directives
@try
@throws
@catch
@finally

Directives used for particular Purpose
@class
@selector
@"string"
@syncronized

Keywords for Memory Management
alloc init retain release autorelease copy drain new strong weak

Property Keywords
atomic nonatomic assign readonly

Some Other Keywords
BOOL;
self
super
SEL
id

Preprocessor Directives
#import #define

eg:
```
BOOL y = YES;  //1
BOOL z = NO;  //0
```

Class

class is nothing but collection of variables and Methods.It is blue print for creating the objects.Class consists of state and behaviour ,state is like variables and Behaviour is like methods.

In Objective C Class is declared using @interface directive.Class is written in .h file.
And the implementation of a class is in .m file.The Variables declared in a class can be accessed through out the class,And these variables are instance variables.By default all instance variables are protected,they can not access to out side classes.

Eg:

C++ Class:

```
Class A:B
{
   int x;
   float y;

   addition(int x,float y)
   {
      this.x =x;   //this is like current Object
   }
}
```

Objective C:

A.h File

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

@interface A :B  <Protocol 1, Protocol 2, Protocol 3>
{
    //Variables Declaration
   int x;
   float y;
}
//Properties Declaration

//Methods Declaration
void addition();
void subtraction();
@end
```

.m File

```
#import A.h
@implementation A

void addition()
{
   x=2;
```

```
  int y;
}
void subtraction()
{

}
@end
```

Understanding Inheritance

You have already seen how Objective-C classes inherit from parent classes. In the interface, you specify that a class inherits from another class by placing the parent's name after the class's name and a colon.

```
   @interface SimpleChild : Simple
```

Like any object-oriented language, Objective-C classes extend ancestors further up
its hierarchy, with new methods and instance variables. Objective-C child classes can also redefine an ancestor's method. But, like Java (and unlike C++), an Objective-C class can only inherit from one parent; Objective-C doesn't support multiple inheritance.

Overriding Methods

Objective-C inheritance allows overriding methods, but not instance variables. You already saw an example of overriding methods when you overrode NSObject's dealloc, retain, and release methods in the class Foo.

```
#import "Foo.h"
@implementation Foo
- (void) dealloc {
 NSLog(@"deallocating Foo....");
 [super dealloc];
}
---snip---
@end
```

Instead of calling NSObject's methods, the runtime first calls Foo's, but since the methods in Foo all call its parent's version of each method, the runtime looks in Foo's inheritance hierarchy for a version of the method until it finds NSObject's version.

NOTE

The term "super" refers to the class's parent. For instance, you might have a method called doIt that matches the parent's doIt method. But doIt might add functionality and not replace functionality, so you should call the parent doIt method as well.

```
- (void) doIt {
    [self doMyStuff];
    [super doIt];
}
```

Overloading Methods

Unlike Java, you cannot overload methods when using Objective-C. In Java you overload a method when you provide a method with the same name but a different signature in the same class. For instance, you could define two methods like the following. The two methods are treated as distinct methods by the Java runtime.

```
   public void myMethod(String name);
   public void myMethod(int age);
```

Not so when using Objective-C, when faced with two methods like below, the compiler issues an error and will not compile your class.

```
   - (void) myMethod: (NSString *) name;
   - (void) myMethod: (int) age;
```

Because Objective-C does not support overloading, in the Objective-C programming community, it is common practice to add the argument's name to the method name.

```
   - (void) myMethodString: (NSString *) name;
   - (void) myMethodInt: (int) age;
```

Finally, you should note that for multiple argument methods, Objective-C's lack of method overloading is not problematic if any argument other than the first is different. Remember, a method's name includes its argument names. The following two method names are not the same.

```
   - (void) myMethod: (int) age name: (NSString *)
theName;
   - (void) myMethod: (NSString *) name age: (int)
theAge;
```

The compiler treats these methods as distinct because their names are actually myMethod: name: and myMethod:age: and not simply myMethod. Get used to this naming convention, it might seem strange at first, but it honestly makes Apple's documentation much easier to use.

The Runtime System

The Objective-C language defers as many decisions as it can from **compile time** and **link time** to **runtime**. Whenever possible, it dynamically performs operations such as creating objects and determining what method to invoke. Therefore, the language requires not just a compiler, but also a runtime system to execute the compiled code. The runtime system acts as a kind of operating system for the Objective-C language; it's what makes the language work. Typically, however, you don't need to interact with the runtime directly. To understand more about the functionality it offers

Messaging

The difference between calling a method and sending a message isn't Objective-C's only difference from Java, C++, and other dot-notation languages. Objective-C uses what's called infix notation. Infix notation mixes operands and operators. You don't really need to fully understand infix notation, other than it means Objective-C looks substantially different from Java and C++. An Objective-C message begins with an opening square brace and ends with a closing square brace followed by a semicolon. The object's name follows the opening brace, followed by a space, followed by the message. Arguments passed to the message follow a colon. You can, of course, have multiple-argument methods, as you will see in the next chapter. For now, though, just consider single-argument methods. Figure 3-4 summarizes an Objective-C message with a single argument.

Objective C Methods:

1.instance Method
2.Class Method

1.instance Method
Instence methods are called using object/ instance of the class.These are starts with '-'.

Eg:

```
    -(void)addition;
    -(void)subtraction:(int)x  Y:(int)y;
```

2.Class Method

Class Methods are called using class name.And the memory location of these methods is separate from the instance methods.And these methods are starts with '+'.

```
Eg:  +(void)addition;
     +(void)subtraction:(int)x  Y:(int)y;
```

Method Calling:

We use messaging syntax to call a method in Objective C.Messaging syntax is introduced from the small talk.All methods are allocated in memory at run time. [] -->is the syntax of messaging.

syntax: [sender receiver];

```
sender    ----> Object or class Name  //Caller
Receiver  ----> Method name   //callee
```

Eg:

```
    //instance method Calling
    [self addition];
    [self subtraction:20 Amount:30  Age:40];
    //Class Method Calling
   [ClassName addition];
```

Nested Arguments

Like Java, you can nest Objective-C messages. For instance, using Java, you might write the following:

```
objMyObject.getFoo(objMyFooIdentifier.getID());
```

In Objective-C:

```
[objMyObject getFoo: [objMyFooIdentifier getID]];
```

Using Java, you might nest an object's constructor in another method.

```
objTester.testFubar(new Fubar(33));
```

In Objective-C:

```
[objTester testFubar[[Fubar alloc] initWithInteger : 33]]];
```

Program:

.h

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

```
@interface ViewController : UIViewController
{
    int p;
}
+(void)multiplication;
-(void)subtraction;
-(void)addition:(int)x  Amount:(int)y   Age:(int)z;
@end
```

.m

```
#import "ViewController.h"
@implementation ViewController
-(void)viewDidLoad
{
    [ViewController multiplication];
    [self subtraction];
    [self addition:20 Amount:30 Age:40];
}
+(void)multiplication
{
    NSLog(@"multiplication called");
}
-(void)subtraction
{
    int x = p;
```

```
   NSLog(@"Subtraction called");
}
-(void)addition:(int)x  Amount:(int)y   Age:(int)z
{
   NSLog(@"addition called and amount = %d",y);
}
@end
```

Objects

As the name implies, object-oriented programs are built around **objects**. An object associates data with the particular operations that can use or affect that data. Objective-C provides a data type to identify an object variable without specifying a particular class of the object.

In Objective C we use alloc and init methods for object creation.alloc is for allocating the object.And init is for initialising the object.alloc is a class method and init is a instance method.These methods are present in the Object class.

Eg:

Java : A a = new A();

Objective C : A *a = [[A alloc] init];

//3 ways to create an object in Objective C

1. A *a = [[A alloc] initWithX:20 Y:30]; //own object

2. NSString *str = [NSString stringWithString:@"hi"]; //autorelease Object Class Methods or convenience Methods

3.A *a = [A new]; //own Object

id

id x

The id variable is a data type that represents an object's address. Because it's just an address, id can be any object, and because its type is a pointer, you don't need to include the * symbol, as the * symbol signifies a pointer to a specific type. For instance,

```
Foo * myFoo;
```

is a pointer to a Foo object. The compiler knows the pointer points to an address that contains a Foo type. However, the following,

```
id myFoo;
```

provides no such information to the compiler. The compiler only knows that myFoo is a pointer—the compiler knows where the pointer is pointing, but doesn't know the data type of what myFoo points to. Only at runtime can it be determined what myFoo actually points to

eg:

```
id x = object;
```

```
NSString *str = [[NSString alloc] init];
id x = str;
```

eg 2:

```
-(id)addition:(id)x  Y:(int)y
{


}
```

enum

enum is a user defined data type.And we create our own data types using enum.enum values are by default starts with 0.And It improves the speed of Execution.

eg:

.h

```
enum
{
   RED=100,
    YELLOW,
    WHITE
};
@interface A

@end
```

View Life Cycle

```
-(id)initWithNibName:(NSString *)nibNameOrNil
bundle:(NSBundle *)nibBundleOrNil
{


}
-(id)init
{

}

-(void)loadView
{
}
-(void)viewDidLoad
{


}
-(void)viewWillAppear:(BOOL)animated
{


}
-(void)viewDidUnLoad
{


}
-(void)didReceiveMemoryWarning
{

}

-(BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)toInterfaceOrientation
{
```

}

Properties

Properties are public and instance variables.Properties have some extra features compared to the normal variables.Properties starts with the keyword @property.

1.Properties makes Instance variables as public
2.Properties having special Attributes
3.Properties can be called using self
4.We can call properties using Messaging

1.atomic

2.nonatomic
3.retain
4.copy
5.readonly
6.asign
7.strong
8.weak

Syntax:

```
.h
@interface Abc
{
   int x;
   float y;
   NSString *str;
   NSArray *aArray;
   BOOL status;
   id z;
```

```
}
@property(nonatomic , copy) NSString
*str;
@property int x;

@end




.m

@implementation Abc
@synthesize x;
@dynamic str

//Setter method
-(void)setStr:(NSString *)str1
{
   self.str = str1;
```

```
}

//Getter Method
-(NSString *)str
{
   return self.str;
}

-(void)viewDidLoad
{
    [self setStr:@"hello"];  //Calling
setter method
    [self str];  //calling getter method
}

@end
```

---

setter
getter

```
int x =2;
[self setX:2];

int y = x;
int y = [self x];
```

Accessory Methods -->Setter,getter

Categories

Categories are one of the most useful feature in objective c.Categories allows you to add methods to an existing class with out subclassing.This is useful because we can add methods to predefined classes.If you want to add methods to all instances of

NSString class in your Application just add category.Here we can add only methods.

Eg:

```
.h
@interface NSObject (Algebra)
-(void)addition;
-(void)subtraction;
-(void)multiplication;
@end

.m
@implementation NSObject (Algebra)
-(void)addition
{

}
-(void)subtraction
{

}
-(void)multiplication
{
}
@end
```

Memory management

Modern computer languages using garbage collection.A garbage collection is a run time algorithm that scans the allocated objects in our program and deallocates the objects that we have lost contact.

Cocoa under the iPhone OS does not use garbage collection.So cocoa apps must use managed memory.Applications running on this platform should clean after the use.since iPhone Apps run in a memory constraint environment.

Objective-C uses reference counts to determine if memory should be released or retained. When you create a class instance, the runtime allocates memory for the object and assigns that object a reference count of one. For instance, suppose you had a class named Simple. You first allocate space for it using NSObject's alloc method.

```
Simple *objSimple = [[Simple alloc] init];
```
You then use the object.

```
[objSimple sayHello:@"James"];
```
When finished, you call its release method. If no release method is found, the runtime moves up the classes' inheritance hierarchy until it finds a release implementation. As all classes extend NSObject, if no release instance is found, the runtime calls NSObject's release method.

```
[objSimple release];
```
When an object's reference count reaches zero, the runtime calls the object's dealloc method to deallocate the object. As with retain, if the runtime doesn't find a dealloc method, it moves up the inheritance hierarchy until it finds one. If no dealloc method is found, the runtime calls NSObject's dealloc method and the object is deallocated so the memory can be reclaimed.

You've already seen how alloc, release, and dealloc work; you allocate memory for an object and assign it a reference count of one using the alloc method, and you decrement the reference count by one when calling release. When an object's reference count reaches zero, the program calls NSObject's dealloc method.

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 objects sub objects will be released when they are released. Remember if other classes have strong reference to an object of the class then the whole class won't be released. So it is recommended to use weak properties for delegates.

Keywords for Memory Management:

1.**alloc**

Alloc is a class Method.And which is present in NSObject Class.And It allocates the memory for All instance variables and methods of the class.

Eg: [A alloc];

2.init

init is a instance method,It present in the NSobject Class.And it initialises the all instance variables of the class.
Eg:

```
    A *a  = [[A alloc] init];

-(id)init
{
    x = 0;
   y = 0.0;
  return self;
}
```

3.retain

Retain is used to increment the reference count/Owner ship count of an object.
If we try to release the object it is not reallocated because we retained it.

```
Eg:   A *a = [[A alloc] init];   //retain count 1
       [a  retain];   //retain count 2
       [a release];  //retain count 1   //still stays in
memory
```

4.release

Release is used to decrement the reference count by 1.if the reference count is 0 then object is deallocated by calling dealloc method.

e.g.: [a release]; //retain count 0 then object is no more.

5.autorelease

Managing reference counts manually is tiresome and error-prone. NSObject’s autorelease method manages an object’s reference count for you. The autorelease method uses what’s called a release pool to manage an object’s references. Refer to Listing 3-17 and note that this method’s first step is allocating an NSAutoreleasePool. Its second-to-last step is releasing that pool. Calling autorelease adds the object to the pool, and the pool retains the object for you.

autorelease objects are released automatically.And all autorelease objects are added to Autorelease Pools.And all pools are stored in a stack.When pool is released it pass release message to all pool objects(autorelease objects).

Eg: Creating Auto Release Objects

```
      NSString *str6 = [NSString
stringWithString:@"Hello"];  //retain count 1
      NSString *str7 = [NSString
stringWithString:@"Hello"];  //retain count 1
```

6.**NSAutoReleasePool**/@autoreleasepool

We crate the pools using NSAutoReleasePool Class.All autorelease objects will store in these pools.

Eg:

```
   @autoreleasepool
   {
        NSString *str6 = [NSString
stringWithString:@"Hello"];  //retain count 1
      NSString *str7 = [NSString
stringWithString:@"Hello"];  //retain count 1
   } //str6 and str7 are added to pool and deallocated
after end of the pools

   //old xcode Pool Creation
   NSAutoreleasePool *pool = [[NSAutoreleasePool
alloc] init];
```

```
    NSString *str8 = [NSString
stringWithString:@"Hello"];
   [pool release];
```

7.drain

drain is used to release the pool objects irrespective of retain count of autorelease Objects.

Eg: `[pool drain];`

8.copy

If we use copy It created new object of the class and the new object is the owner of created object

```
      b = a; //shallow copy it only copies the reference
```

Eg: `b = [a copy];   //Deep copy it create the new Object`

9.dealloc

dealloc method is called when the object retain count 0.

```
   Eg: -(void)dealloc
         {
```

}

10.new

new is used to crete the object.It is same as alloc and init way.

Eg: A *a = [A new];

11.retaincount

We use this keyword to know the reference count of an object.

Eg: int count = [a retaincount];

12.Assign

You can also specify a property use assignment by using the assign attribute.

@property(assign) Simple objSimple;
Specifying assign is equivalent to simply assigning a pointer to an object without increasing its retain count. You must take care to not call autorelease or release a temporary object, as the assigned property simple

points to the temporary object. I generally avoid using assign for objects.

Where assign is appropriate is for creating primitive properties. For instance, you might make an integer a property of a class. An integer is a primitive, and so you assign values to the property; you do not assign a pointer to the integer.

@property (assign) int myInteger;

Protocols

A protocol is simply a list of method declarations,unattached to a class definition.For example these methods that report user actions,could be gathered into a protocol.

Any class that wanted to respond to user events could adopt the protocol and implement its methods.Protocols are two types.

1.formal Protocols: Predefined protocols
2.informal protocols:user defined protocols

Protocols having two different methods like optional and required methods.

Eg:
```
#import <Foundation/Foundation.h>

@protocol Algebra <NSObject>

@optional
-(void)addition;
-(void)subtraction;

@required
-(void)division;

@end
```

```
//Implementation in a Class'
#import <UIKit/UIKit.h>
#import "Algebra.h"

@interface ViewController : UIViewController
<Algebra>
{
    NSString *str;
}
@property(nonatomic,strong) NSString *str;
@end
```

Selectors

In objective C selectors are used to refer the name of a method.The keyword we use here is @selector.And SEL is a method type SEL variables can store any methods.

Eg:

```
   SEL s = @selector(addition);   //With out Arguments

  SEL b = @selector(subtraction:)  //With Arguments
```

Eg 1://Passing Method as argument

```
  [btn addTarget:self Action:@selector(addition)
Events:UIControlEventTouchUpInside];
```

Eg 2: //checking method is there in a class

```
   if([self respondstoSelector:@selector(addition)])
  {
    //method is there
  }
  else
  {
   //Method is not there in a class
  }
```

Dynamic Binding and Dynamic Typing

Objective-C accomplishes dynamic behavior using what's called dynamic typing and dynamic binding. Dynamic typing means that an object's type is not determined until runtime. For instance,

a method that takes an id or an instance variable of type id has no way of knowing the object's type until the object is actually sent to the method or assigned to the instance variable.

Dynamic binding means that a method to invoke is not determined until runtime. And, unlike Java, Objective-C often doesn't require casting an object to its specific data type before being used.

Threading

In most of the modern operating systems it is possible for an application to split into many threads that all execute concurently.When a program is split into
many threads each thread acts like its own individual program except that all the threads will work in same memory space.

Multi Threads can run on multiple CPU's providing performance and improvement.A multi threaded application work as well as single CPu system but with added speed.In objective C there is a special class for Threading is NSThread.

NSThread: If you needs to run relative task on the back ground of the application after some task we use

NSThread.NSThread class having functions to create a thread,Sleep a thread etc.

```
-(void)threadMethod
{
    sleep(1);

    [NSThread sleepForTimeInterval:
(NSTimeInterval)];
    [NSThread sleepUntilDate:<#(NSDate *)#>]

    for (int i = 0; i<100; i++) {
        NSLog(@"i = %d Child Thread",i);
    }
}

- (void)viewDidLoad
{
    [NSThread
detachNewThreadSelector:@selector(addition)
toTarget:self withObject:nil];
    for(int i=0;i<100;i++)
        NSLog(@"Main Thread is %d",i);
}
```

Exception handling

Exceptions are generated at run time.the Objective C language having exception handling syntax similar to C++ and java.here we use NSException Class.An Exception is a special condition that interrupts normal flow of execution.

Objective-C's exception handling is similar to Java's and C++'s. Like Java and C++, you use a try-catch block. Code that might raise an exception is wrapped in a try block, followed immediately by a catch block. The @try compiler directive defines a code block that might throw an exception. The @catch directive defines a code block that handles the immediately preceding try block. The @finally directive defines a code block that is always executed,

Keywords:

1.try Block: Try block holds the code that can potentialy throws an exception is enclosed in an try block.

synatx:

```
@try
{
  //Exception related Code

}
```

2.Catch Block: The catch block contains exception handling logic for exceptions throws in a try block.You can have multiple catch blocks.

syntax:

```
 @catch(NSException *ex)
{
  NSLog(@"Exception info is %@",[ex userInfo]);
}
```

3.Finally block: Finally block is a optional block.It contains code for deallocation of resources.

```
@finally
{
  [db release];
}
```

Note:NSException class is a base class for all the exception related classes.

Using the Debugger

Xcode's visual debugger overlays the command-line GNU debugger. The open-source GNU debugger has a long pedigree and is the industry standard when it comes to C and C++ debuggers on UNIX and Linux.We can see the number of threads running and check the values in run time and see the logs.Using debugging issue finding is easy.

Pause/Continue : Pauses the application running in the debugger. Note, when the application is paused, this button says Continue. Continue "un-pauses" the application and resumes processing.

Step Over : Processes the next line of code. If the next line is a function call, it skips over the function, proceeding to the next line.

Step Into : Processes the next line of code. If the line is a function call, it jumps to the code in the function.

Step Out : Processes until the function exits.

Deactivate : Disables the current breakpoint.

Breakpoints : Opens the Breakpoints window.

Console : Opens the Debugger Console window.

Breakpoints

Breakpoints tell the debugger where to pause. If you set no breakpoints and then run the application in the debugger, nothing unusual happens, as you didn't tell the application to pause. There are several ways to set a breakpoint, but the easiest is to click in the Editor window's gutter next to the line of code you wish the debugger to stop at (Figure 5-11). If you wish to disable the breakpoint, click it again and it turns light blue, indicating it is disabled. If you wish to remove the breakpoint, ctrl-click and select Remove Breakpoint from the pop-up menu. Alternatively, you can drag the breakpoint off the gutter to remove it. When you run the application in the debugger, it will pause processing at the first encountered breakpoint.

NSZombieEnabled

When an object is deallocated, if there are any objects with a reference to the deallocated object, they are no longer referencing a valid object. Any messages sent to the deallocated object result in errors. Often, the error is rather cryptic. For instance, the following code fragment is obviously an error.

```
  FooBar * myFooBar = [[FooBar alloc] init];
  NSMutableArray * myArray = [[NSMutableArray
alloc] initWithObjects:
  myFooBar,nil];
  [myFooBar dealloc];
  [[myArray objectAtIndex:0] sayHello];
```

FooBar is allocated, initialized, and added to myArray. There are two references to myFooBar, so its retainCount is two. However, deallocating myFooBar makes both references invalid, pointing to deallocated memory space. The sayHello message is sent to the first object in myArray—the problem is that the object no longer exists

Frame Works:

List Of Frame Works:

1.Foundation Frame work
2.UIKit Frame Work
3.Core Graphics
4.Quartz Core Frame work
5.Message UI Frame Work
6.Media Player Frame work
7.AVFoundation Frame work
8.Address Book UI Frame work
9.Address Book Frame work
10.External Accessiories Frame work
11.Sqlite 3.0 dylib frame work
12.System Configuration Frame work
13.open GL Frame work
14.Network Frame work
15.game Kit
16.Store Kit
17.inApp
18.Map Kit Frame work
19.Core Location

20.Twitter Frame work
21.iAD

1.Foundation Frame work

Foundation Frame work is the Base work which is used for iOS and MAC OS X Application Developers.And all classes are starts with NS(NEXT STEP)

As you begin writing code for your app, you'll find that there are many Objective-C frameworks that you can take advantage of. Of particular importance is the Foundation framework, which provides basic services for all apps. The Foundation framework includes value classes representing basic data types such as strings and numbers, as well as collection classes for storing other objects. You'll be relying on value and collection classes to write much of the code

Value Objects
The Foundation framework provides you with classes that generate value objects for strings, binary data, dates and times, numbers, and other values.

A value object is an object that encapsulates a primitive value (of a C data type) and provides services related to that value. You frequently encounter value objects as the parameters and return values of methods and functions that your app calls. Different parts of a framework—or even different frameworks—can exchange data by passing value objects.
Some examples of value objects in the Foundation framework are: NSString and NSMutableString
NSData and NSMutableData
NSDate NSNumber NSValue

List of Classes:

1.NSObject 2.NSString
3.NSMutableString
4.NSArray 5.NSMutableArray
6.NSSet
7.NSMutableSet 8.NSDictionary
9.NSMutableDictionary
10.NSData 11.NSMutableData
12.NSTimer
13.NSThread 14.NSOperation
15.NSOperationQueue

16.NSUserDefaults 17.NSNotification
18.NSNotificationCenter
19.NSURL 20.NSURLRequest
22.NSXMLParser
23.NSMutableURLRequest 24.NSURLConnection
25.NSMutableURLConnection
26.NSRange 27.NSNotFound
28.NSResponder
29.NSCoder 30.NSFileManager
31.NSSortDescriptor
32.NSCharecterSet 33.NSCalender
34.NSDate
35.NSDateFormatter 36.NSValue
37.NSNumber
38.NSDecimalNumber 39.NSInteger
40.NSIndexPath
41.NSAutoReleasePool 42.NSException
43.NSError
44.NSPredicate 45.NSInvocvation

NSString

Objective-C supports the same conventions for specifying strings as does C: Single characters are enclosed by single quotes, and strings of characters are surrounded by double quotes. But Objective-C frameworks typically don't use C strings. Instead, they use NSString objects.

The NSString class provides an object wrapper for strings, offering advantages such as built-in memory management for storing arbitrary-length strings, support for different character encodings (particularly Unicode), and utilities for string formatting. Because you commonly use such strings, Objective-C provides a shorthand notation for creating NSString objects from constant values. To use this NSString literal, just precede a double-quoted string with the at sign (@), as shown in the following examples:

```
- (void)viewDidLoad
{
    //String Initialization
    NSString *str = [[NSString alloc] init];

    str = @"Hello";


    //using Class method
    NSString *str1 =[NSString
stringWithString:@"Hi"];

    //Printing a string
    NSLog(@"The String is %@",str);

    int length = [str length];
    NSLog(@"The length is %d",length);

    //String Formatting

    int x = 2;
    float y=4.4;

    //      2/4.4
    NSString *formatetdStr=[NSString
stringWithFormat:@"%d/%0.1f",x,y];
    NSLog(@"formatted str is %@",formatetdStr);


    //upper case
    NSString *upperCase= [str uppercaseString];

    //Lower case
    NSString *lowerCase = [str lowercaseString];
    NSLog(@"uppercase is %@ \n lower case is 
%@",upperCase,lowerCase);

    //To Split An String
    NSString *str3 =  @"Ho How are You";
     NSArray *aArray = [str3 
componentsSeparatedByString:@" "];
    NSLog(@"The Split array is %@",aArray);


    //Sub String
    NSString *subStringTo=[str3 substringToIndex:2];
    NSLog(@"SubString to is %@",subStringTo);

    NSString *subStringFrom = [str3 
substringFromIndex:2];
    NSLog(@"SubString From is %@",subStringFrom);


    //String Appending
    NSMutableString *aMutStr = [[NSMutableString 
alloc] init];
    [aMutStr setString:@"Hi"];
    [aMutStr appendString:@" Ha Ha Ha"];
    NSLog(@"Amut Str is %@",aMutStr);
    //String Replacing
    int count = [aMutStr
replaceOccurrencesOfString:@"Ha" withString:@"Hi"
options:NSLiteralSearch range:NSMakeRange(0,
[aMutStr length])];
    NSLog(@"After Replacing the Amut Str is
%@",aMutStr);
    NSLog(@"replace count is %d",count);

    //String Joining
    NSString *joinedStr = [aArray
componentsJoinedByString:@"/"];
    NSLog(@"joined str is %@",joinedStr);



    //insert String
    NSMutableString *amutStr2 = [NSMutableString
stringWithString:@"Ho"];
    [amutStr2 insertString:@"ell" atIndex:1];
    NSLog(@"After inserting the string is
%@",amutStr2);

    //String Comparision
    NSString *enteredUserName = @"rapid";
    NSString *actualUsername = @"rapid";
    if([enteredUserName
isEqualToString:actualUsername]  )
    {
        NSLog(@"user name is correct");
    }
    else
    {
        NSLog(@"username is wrong");
    }

    //Finding Charecter in an String
    int index = [amutStr2 characterAtIndex:1];

    //Convertioning string to Primitive Types
    NSString *age = @"32";
    int age1 = [age intValue];
    float age2 = [age floatValue];
    age1+=4;
    NSLog(@"age in integer is %d",age1);

     NSString *finalAge= [NSString
stringWithFormat:@"%d",age1];
    NSLog(@"Final Age is %@",finalAge);
    //String Wrting
    [str writeToFile:@"File path" atomically:YES
encoding:NSUTF8StringEncoding error:nil];

}
```

String Out Put:

```
2012-09-11 10:26:59.786 b16Sample3[18454:c07] The String is Hello
2012-09-11 10:26:59.788 b16Sample3[18454:c07] The length is 5
2012-09-11 10:26:59.788 b16Sample3[18454:c07] formatted str is 2/4.4
2012-09-11 10:26:59.789 b16Sample3[18454:c07] uppercase is HELLO
 lower case is hello
2012-09-11 10:26:59.789 b16Sample3[18454:c07] The Split array is (
    Ho,
    How,
    are,
    You
)
2012-09-11 10:26:59.789 b16Sample3[18454:c07] SubString to is Ho
2012-09-11 10:26:59.790 b16Sample3[18454:c07] SubString From is  How are You
2012-09-11 10:26:59.790 b16Sample3[18454:c07] Amut Str is Hi Ha Ha Ha
2012-09-11 10:26:59.791 b16Sample3[18454:c07] After Replacing the Amut Str is
Hi Hi Hi Hi
2012-09-11 10:26:59.791 b16Sample3[18454:c07] replace count is 3
2012-09-11 10:26:59.791 b16Sample3[18454:c07] joined str is Ho/How/are/You
2012-09-11 10:26:59.792 b16Sample3[18454:c07] After inserting the string is
Hello
2012-09-11 10:26:59.792 b16Sample3[18454:c07] user name is correct
2012-09-11 10:26:59.817 b16Sample3[18454:c07] age in integer is 36
2012-09-11 10:26:59.818 b16Sample3[18454:c07] Final Age is 36
```

NSArray

```
- (void)viewDidLoad
{
    //Initialization
    NSArray *aArray = [[NSArray alloc ]
initWithObjects:@"Hello", nil];  //retain count 1

    //using Class Method
    NSArray *bArray = [NSArray
arrayWithObjects:@"Hi",@"Hello", nil];

    //get Count
    int count = [bArray count];
    NSLog(@"aArray is %@",bArray);
    NSLog(@"count of an array is %d",count);

    //get object from array
    NSString *firstObj = [bArray objectAtIndex:0];
    NSLog(@"first Obj is %@",firstObj);


    //Adding Object
    NSMutableArray *aMutArray = [[NSMutableArray
alloc] initWithArray:bArray];
    [aMutArray addObject:@"World"];


    //inserting object at index
    [aMutArray insertObject:@"good" atIndex:1];

    //replace object
    [aMutArray replaceObjectAtIndex:1
withObject:@"Abc"];

    //remove Object
    [aMutArray removeObjectAtIndex:1];


    //remove All objects
    [aMutArray removeAllObjects];

    //remove last object
    [aMutArray removeLastObject];


    //Checking Object is there in an array
    if([aMutArray containsObject:@"Hi"])
    {
        NSLog(@"Object is there");
    }
   else
   {
       NSLog(@"Object is not there");
   }

    //getting index of an object
     int index = [aMutArray indexOfObject:@"Hello"];

    //Joining array
     NSString *joinedStr = [aMutArray
componentsJoinedByString:@","];
    NSLog(@"Joined str is %@",joinedStr);

    //Appending other array to an existing array
    [aMutArray
arrayByAddingObjectsFromArray:bArray];

    //write to file
     BOOL status= [aMutArray writeToFile:@"give the
path" atomically:YES];
    NSLog(@"status is %d",status);
    //getting the array from File
    NSArray *fileArray= [NSArray
arrayWithContentsOfFile:@"Path"];
}
```

//out put

```
(
    Hi,
    Hello
)
```

NSDate & NSDateFormatter

```
- (void)viewDidLoad
{
    NSDate *aDate = [NSDate date];
    NSLog(@"Today date is %@",aDate);


    //converting Date to String
    NSDateFormatter *formatter = [[NSDateFormatter
alloc] init];
```

```
    [formatter
setDateStyle:NSDateFormatterShortStyle];
    [formatter
setTimeStyle:NSDateFormatterNoStyle];
    [formatter setDateFormat:@"MM-dd-YYYY eeee
a"];

    NSString *dateStr  = [formatter
stringFromDate:aDate];
    NSLog(@"date str is %@",dateStr);

    //Adding date
    aDate = [aDate dateByAddingTimeInterval:
2*24*60*60];
    NSLog(@"final date after adding 2 days is
%@",aDate);
}
```

Note:

```
[formatter setDateFormat:@"MM-dd-YYYY"];
out put: 09-12-2012

[formatter setDateFormat:@"MM-dd-YYYY eee"];
out put: 09-12-2012 wed
```

```
[formatter setDateFormat:@"MM-dd-YYYY eeee"];
out put: 09-12-2012 wednesday

[formatter setDateFormat:@"MM-dd-YYYY a"];
out put: 09-12-2012 AM
```

NSNumber

```
- (void)viewDidLoad
{
    int x1 =2;
    float y1 = 4.4;

    NSNumber *aNum = [NSNumber
numberWithInt:x1];
    NSNumber *bNum = [NSNumber
numberWithFloat:y1];

    NSArray *numbersArray = [NSArray
arrayWithObjects:aNum,bNum, nil];
    NSLog(@"numbers array is %@",numbersArray);

    //converting number object to prinitive types
     int xVal = [aNum intValue];
    NSLog(@"xvalue is %d",xVal);
}
```

NSDictionary & NSMutableDictionary

```
- (void)viewDidLoad
{
    //initialization
    NSDictionary *aDict = [[NSDictionary alloc]
initWithObjectsAndKeys:@"abc",@"firstName",@"kum
ar",@"lastName",@"abc@gmail.com",@"email",@"12253
45232",@"phnNo", nil];
    NSLog(@"Dict is %@",aDict);

    //using class Method
    NSDictionary *bDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"23",@"age", nil];

    //count
    int count = [aDict count];
    NSLog(@"count is %d",count);

    //getting Object from Dictionary
     NSString *firstName = [aDict
objectForKey:@"firstName"];
```

```
    NSLog(@"first Name is %@",firstName);

    //Getting All keys
    NSArray *allKeys = [aDict allKeys];
    NSLog(@"all keys are %@",allKeys);

    //Getting All Objects
    NSArray *allObjects = [aDict allValues];
    NSLog(@"all objects are %@",allObjects);

    //Adding object
    NSMutableDictionary *aMutDict =
[NSMutableDictionary
dictionaryWithDictionary:aDict];
    [aMutDict setObject:@"500018" forKey:@"zip"];
    [aMutDict setObject:@"Ap" forKey:@"state"];
    NSLog(@"amut Dict is %@",aMutDict);

    //removing Object
    [aMutDict removeObjectForKey:@"zip"];

    //remove All objects
   // [aMutDict removeAllObjects];

    //Adding dicts to an Array
    NSArray *dictsArray = [NSArray
arrayWithObjects:aDict,bDict,aMutDict, nil];
    NSLog(@"Dicts array is %@",dictsArray);


    //getting first Name
    if([dictsArray count]>=1)
    {
        NSDictionary *firstDict = [dictsArray
objectAtIndex:0];
        if(firstDict)
            firstName= [firstDict
objectForKey:@"firstName"];
    }

    //getting particular object from array of
Dictionaries
     NSArray *firstNameArray= [[dictsArray
valueForKey:@"firstName"] allObjects];
    NSLog(@"firstNameArray is %@",firstNameArray);

    //Predicate Begins With
     NSArray *predicateArray= [dictsArray
filteredArrayUsingPredicate:[NSPredicate
predicateWithFormat:@"age beginswith[cd]
%@",@"23"]];
    NSLog(@"predicate Array is %@",predicateArray);

    //Predicate Contains
    NSArray *predicateArray1= [dictsArray
filteredArrayUsingPredicate:[NSPredicate
predicateWithFormat:@"age contains[c] %@",@"3"]];
    NSLog(@"predicate Array is %@",predicateArray1);

    //wrtiting to File
    [aDict writeToFile:@"path" atomically:YES];
}
```

output:

```
2012-09-13 10:34:41.012 b16Sample3[29465:c07] Dict is {
    email = "abc@gmail.com";
    firstName = abc;
    lastName = kumar;
    phnNo = 1225345232;
}
2012-09-13 10:34:41.014 b16Sample3[29465:c07] count is 4
2012-09-13 10:34:41.015 b16Sample3[29465:c07] first Name is abc
2012-09-13 10:34:41.017 b16Sample3[29465:c07] all keys are (
    lastName,
    firstName,
    email,
    phnNo
)
2012-09-13 10:34:41.018 b16Sample3[29465:c07] all objects are (
    kumar,
    abc,
    "abc@gmail.com",
    1225345232
)
2012-09-13 10:34:41.019 b16Sample3[29465:c07] amut Dict is {
    email = "abc@gmail.com";
    firstName = abc;
    lastName = kumar;
    phnNo = 1225345232;
    state = Ap;
    zip = 500018;
}
2012-09-13 10:34:41.020 b16Sample3[29465:c07] Dicts array is (
        {
        email = "abc@gmail.com";
        firstName = abc;
        lastName = kumar;
        phnNo = 1225345232;
    },
        {
        age = 23;
    },
        {
        email = "abc@gmail.com";
        firstName = abc;
        lastName = kumar;
        phnNo = 1225345232;
        state = Ap;
    }
)
2012-09-13 10:34:41.021 b16Sample3[29465:c07] firstNameArray is (
    abc,
    "<null>",
    abc
)
2012-09-13 10:34:41.036 b16Sample3[29465:c07] predicate Array is (
        {
        age = 23;
    }
)
```

NSSet and NSMutableSet

```
- (void)viewDidLoad
{
    NSSet *ASet = [[NSSet alloc]
initWithObjects:@"hi",@"hello", nil];
    NSLog(@"aset is %@",ASet);

    //using class method
    NSSet *bSet = [NSSet setWithObjects:@"world",
nil];



    //count
    int count = [ASet count];
    NSLog(@"count is %d",count);

    //gettin g object from a set
     NSString *firstObj= [ASet anyObject];
    NSLog(@"object is %@",firstObj);

    //getting particular object
      NSString *secondObj=  [[ASet allObjects]
objectAtIndex:1];
    NSLog(@"second Object is %@",secondObj);
    //Adding object
    NSMutableSet *aMutSet = [[NSMutableSet
alloc]initWithSet:ASet];
    [aMutSet addObject:@"Good"];

    //remove object
    [aMutSet removeObject:@"Good"];

    //union
    [aMutSet unionSet:bSet];
    NSLog(@"after union is %@",aMutSet);

    //intersection set
    [aMutSet intersectSet:ASet];
    NSLog(@"after intersection is %@",aMutSet);

    //minus
    [aMutSet minusSet:ASet];
    NSLog(@"minus is %@",aMutSet);
}
```

out put:

```
2012-09-13 10:43:13.726 b16Sample3[29557:c07] aset is {(
    hi,
    hello
)}
2012-09-13 10:43:13.728 b16Sample3[29557:c07] count is 2
2012-09-13 10:43:13.729 b16Sample3[29557:c07] object is hi
```

```
2012-09-13 10:43:13.730 b16Sample3[29557:c07] second Object is hello
2012-09-13 10:43:13.731 b16Sample3[29557:c07] after union is {(
    hi,
    world,
    hello
)}
```

NSRange & NSNotFound

```
NSString *str = @"Hello world";
NSRange range= [str rangeOfAnString:@"world"];
if(range.location != NSNotFound)
{
    NSLog(@"world is there in a String");
}
else
{
    NSLog(@"world is not there in a String");
}

case 2:
NSMakeRange(0,320);
```

NSUserDefaults

The NSUserDefaults class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences. For example, you can allow users to determine what units of measurement your application displays or how often documents are automatically saved. Applications record such preferences by assigning values to a set of parameters in a user's defaults database. The parameters are referred to as defaults since they're commonly used to determine an application's default state at startup or the way it acts by default.

eg:

```
    //save the data to defaults
    NSString *username = @"kumar";
    NSString *pwd = @"123454";
    NSUserDefaults *defaults = [NSUserDefaults 
standardUserDefaults];
    [defaults setObject:username forKey:@"username"];
    [defaults setObject:pwd forKey:@"passWord"];
    [defaults synchronize];
    //Getting data
    NSUserDefaults *defaults2 = [NSUserDefaults
standardUserDefaults];
     NSString *uName= [defaults2
objectForKey:@"username"];
    NSString *passWord = [defaults2
objectForKey:@"passWord"];
    NSLog(@"uname = %@ and pwd =
%@",uName,passWord);



//    [defaults2 setInteger:<#(NSInteger)#>
forKey:<#(NSString *)#>]
//    [defaults2 setFloat:<#(float)#>
forKey:<#(NSString *)#>]
```

NSTimer

```
-(void)addition
{
    count++;
    if(count ==5)
```

```
        [timer invalidate];
    NSLog(@"timer called");
 }
- (void)viewDidLoad
{
     timer = [NSTimer
scheduledTimerWithTimeInterval:1 target:self
selector:@selector(addition) userInfo:nil
repeats:YES];
}
```

NSData

```
- (void)viewDidLoad
{
    //getting data from a File
    NSData *data = [NSData
dataWithContentsOfFile:@"file path"];

    //converting image to data
    NSURL *url = [NSURL URLWithString:@"http://
www.technobuffalo.com/wp-content/uploads/2012/09/
iphone-5-black-apple-press-01-360x270.jpg"];
    NSData *imgdata= [NSData
dataWithContentsOfURL:url];
    UIImage *img = [UIImage imageWithData:imgdata];
    self.view.backgroundColor =[UIColor
colorWithPatternImage:img];


    //getting size of an image
    int size = [imgdata length];
    NSLog(@"the size of an image is %0.2f KB",
(float)size/1024);


    //Image Dimensions
    CGSize sizeImg = img.size;



    //NSMutableData
    NSMutableData *aMutData = [[NSMutableData
alloc] init];
    [aMutData appendData:imgdata];
 NSData *aData = [[NSData alloc] init];

    //converting data to Image
    UIImage *img = [UIImage
imageWithData:aData];
```

```
    self.view.backgroundColor = [UIColor
colorWithPatternImage:img];


    //converting Image to data
        //jpeg
    NSData *imgData =
UIImageJPEGRepresentation(img, 1);
        //png
    NSData  *pngImgdata =
UIImagePNGRepresentation(img);


    //Data to String
    NSString *str = [[NSString alloc]
initWithData:aData
encoding:NSUTF8StringEncoding];

    //String to Data
    NSData *strData = [str
dataUsingEncoding:NSUTF8StringEncoding];
```

```
    //How to get resource file path
     NSString *path = [[NSBundle mainBundle]
pathForResource:@"a" ofType:@"png"];
    NSLog(@"path is %@",path);

    //Getting data from file path
    NSData *fileData = [NSData
dataWithContentsOfFile:path];

    //how to convert url data to NSData
    NSURL *urlString = [NSURL
URLWithString:@"http://www.google.com/
abc.png"];
    NSData *urlData = [NSData
dataWithContentsOfURL:urlString];

    //how to convert NSUrl to FileURl
    NSURL *fileUrl = [NSURL
fileURLWithPath:path];
```

NSNotification & NSNotificationCenter

If you have ever used Java listeners, you already know a notification's basic workings. Cocoa's notifications are similar to Java's listeners, only easier and more flexible. Every Cocoa application has a notification center, NSNotificationCenter. Classes in your application can post notifications, NSNotification, to the notification center. Other classes can register with the notification center to listen for notifications. Through notifications, classes can communicate with each other without even knowing about one another's existence.

```
-(void)addition
{
    NSLog(@"addition called");
}
-(void)subtraction:(NSNotification *)aNotification
```

```
{
    NSDictionary *aDict = [aNotification userInfo];
    NSString *name = [aDict objectForKey:@"name"];
    NSLog(@"subtraction called %@",name);

    //Removing selected Notification
    [[NSNotificationCenter defaultCenter]
removeObserver:self
name:@"ALGEBRA_NOTIFICATION" object:nil];

    //Removing All notifications
    [[NSNotificationCenter defaultCenter]
removeObserver:self];
}
- (void)viewDidLoad
{
    //add Methods to Notification Center
    [[NSNotificationCenter defaultCenter]
addObserver:self selector:@selector(addition)
name:@"ALGEBRA_NOTIFICATION" object:nil];
    [[NSNotificationCenter defaultCenter]
addObserver:self selector:@selector(subtraction:)
name:@"ALGEBRA_NOTIFICATION" object:nil];
```

```
    SecondClass *second = [[SecondClass alloc] init];
    [[NSNotificationCenter defaultCenter]
addObserver:second selector:@selector(division)
name:@"ALGEBRA_NOTIFICATION" object:nil];


    //post notification
    NSDictionary *eDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"kumar",@"name",
nil];
    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ALGEBRA_NOTIFICATION"
object:nil userInfo:eDict];

}
```

UIKit Framework

1.UIView	2.UIButton
3.UIControl	4.UILabel
5.UITextField	6.UISwitch
7.UISegmentedControl	8.UIProgressView
9.UINavigationBar	10.UIBarButtonItem
11.UIToolBar	12.UITabBar

13.UIPageControl 14.UISlider
15.UIImageView 16.UITextView
17.UIWindow 18.UIPickerView
19.UIDatePicker 20.UITableView
21.UITableViewCell 22.UIScrollView
23.UIAlertView 24.UIActionSheet
25.UIGestureRecogniger
26.UIPinchGestureRecogniger
27.UILongPressGestureRecogniger
28.UIPanGestureRecogniger
29. UISwipeGestureRecogniger
30.UIActivityIndicatorView
31.UIAcceleration 32.UIAcceleroMeter
33.UIFont 34.UIColor
35.UIScreen 36.UIDevice
37.UIApplication
38.UILocalNotification 39.UITouch
40.UIWebView
41.UISearchBar 42.UIViewController
43.UINavigationController
44.UITabBarController 45.UISplitViewController
46.UIFlipSideViewController
47.UIImage 48.UIEvent
49.UINib 50.UIPasteBoard

Sample Program in iPhone SDK using UIKit

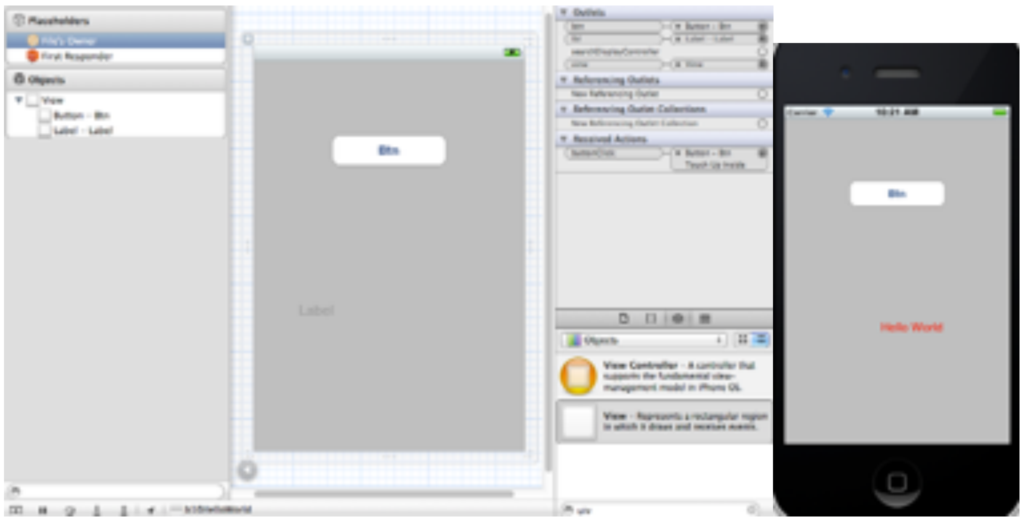
.h file

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

@interface ViewController : UIViewController
{
    IBOutlet UIButton *btn;
    IBOutlet UILabel *lbl;
}
-(IBAction)buttonClick:(id)sender;

@end
```

.m File

```
#import "ViewController.h"

@interface ViewController ()

@end
```

```
@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view,
typically from a nib.
}
-(IBAction)buttonClick:(id)sender
{
    lbl.text = @"Hello World";
    [lbl setTextAlignment:UITextAlignmentCenter];
    [lbl setTextColor:[UIColor redColor]];
}
@end
```



IBOutlet

IBOutlet is a preprocessor directive, evaluates to void, and is ignored by the compiler,
but all you really need to know is that IBOutlet is how Interface Builder knows the variable was created for its use. When adding a class, Interface Builder scans the class for IBOutlets and knows those variables are intended as outlets. You can then easily connect your graphical component to the variable, as Interface Builder adds the outlets to the inspector automatically for you to select. Note, though, Interface Builder doesn't connect the outlets to anything; it just adds them. You are responsible for adding any connections.

IBAction

Actions are messages sent from objects in the nib to methods outside the nib. You define an action in your code using the IBAction keyword. Like IBOutlet, it's a preprocessor directive and evaluates to void. You define an action in your code using the IBAction keyword. Also, like IBOutlet, when you add a class to Interface Builder, it scans the class for IBActions and adds them to the inspector for you to select. You can

then connect user interface events to a specific method in your code. Note that a method designated as an action must not return a value, as the preprocessor replaces IBAction with void, and so all the compiler sees is void.
An action also takes a single parameter, sender. The sender is the id of the control calling the action. So if a UIButton instance called changeLabelValue, the sender would be the pointer to the UIButton.

```
- (IBAction) changeLabelValue: (id) sender;
```

@class precompiler directive

The @class is a compiler directive that informs the compiler that a class of that type will exist. It is what's called a forward declaration, so named because it is informing the compiler before the class is actually declared.

Q:What is that initWithNibName line?

A: This method allows initializing a UIViewController's view from a nib file rather than from code. You use this method when you want to write code that initializes a view controller whose view resides in a nib. You should note that this is just one of several different ways you might load a view from a nib into a class. By book's end, you will be familiar with most of them.

UIView

UIViews are how the iPhone displays information on the screen. The UIView class is responsible for displaying a rectangular area on the screen. The UIViewController manages the UIView and is responsible for handling the view's logic. The UIViewController is the glue
between the view—the UIView—and your data classes—the model.
There are many UIView subclasses you might work with while developing an iPhone
application. In fact, every graphical component you use on an iPhone graphical user interface (GUI) is a UIView subclass. Technically speaking, everything is a view. But typically, when documentation or some other material refers to a "view," it is referring to a content view.

A content view is a view that has a view controller and is responsible for presenting a screen's worth of user interface.

-> UIView's responsibilities are drawing to the screen and handling events associated with a user's interaction with an application.

->A UIView has an associated UIViewController. The UIViewController manages the view.

->The view controller loads and unloads its associated views, manages views, and handles application life cycle events for its views.

-> Every graphical iPhone control you use in Interface Builder is a UIView subclass. Notice that UIScrollView and UIControl both have further subclasses listed in the table's second column. UIViews function as containers, controls, displays, alerts, action sheets, navigation controls, and as the application's window.

Below are the list of Sub Classes of UIView

UIWindow,UILabel,UIPickerView, UIProgressView, UIActivityIndicatorView, UIImageView,UITabBar, UIToolBar, UINavigationBar, UITableViewCell, UIActionSheet,UIWebView,UIControl

UIScrollView —> UITableView,UITextView

UIControl —> UIButton , UIDatePicker ,UIPageControl UISegmentedControl ,UITextField,UISlider, UISwitch

Core Graphics

CGPoint

```
struct CGPoint
{
   CGFloat x;
   CGFloat y;
};
```

```
CGPoint pnt;
pnt.x  =20;
pnt.y = 30;

 CGPoint pnt2= CGPointMake(20,30);
```

2.CGSize

```
struct CGSize
{
   CGFloat width;
   CGFloat height;
};

CGSize size;
size.width = 100;
size.height = 200;

CGSize size1 = CGSizeMake(100,200);
```

3.CGRect

```
struct CGRect
```

```
{
   CGPoint origin;
                    CGSize size;
                  };
```

```
CGRect rc;
rc.origin.x =2;
rc.origin.y = 3;
rc.size.width = 100;
rc.size.height = 200;

CGRect rc1= CGRectMake(2,3,100,200);
```

Frame

Eg 1:

```
CGRect btnFrame = btn.frame;
btn.frame = rc;
```

Eg 2: `btn.center = CGPointMake(20,30);`

Eg 3: `CGSize size = img.size;`

UIButton

The most rudimentary control is arguably the button. What can you say about buttons? You click them, or on an iPhone, you tap them, and something happens. The iPhone has several different button styles

.h

```
#import <UIKit/UIKit.h>
@interface ButtonsBackgroundViewController :
UIViewController {
  IBOutlet UIButton * clearButton;
  IBOutlet UIButton * smallButton;
}
@property (nonatomic, retain) IBOutlet UIButton *
clearButton;
@property (nonatomic, retain) IBOutlet UIButton *
smallButton;
- (IBAction) disableBut: (id) sender;
@end
```

.m

```
#import "ButtonsBackgroundViewController.h"
@implementation ButtonsBackgroundViewController
@synthesize clearButton;
@synthesize smallButton;
```

```
- (IBAction) disableBut: (id) sender {
  if(clearButton.enabled == YES) {
    clearButton.enabled = NO;
    smallButton.enabled = NO;
    [((UIButton *) sender) setTitle:@"Enable" forState:
UIControlStateNormal];
  }
  else {
   clearButton.enabled = YES;
   smallButton.enabled = YES;
   [((UIButton *) sender) setTitle:@"Disable" forState:
UIControlStateNormal];
  }
}
@end
```

UIImageView

Image Display

```
-(void)viewDidLoad
{
    UIImageView *imgView = [[UIImageView alloc]
initWithFrame:CGRectMake(100, 200, 100, 100)];

    //Set image
    UIImage *img = [UIImage
imageNamed:@"intro_wait@2X.png"];
    [imgView setImage:img];

    //set back ground Color
    [imgView setBackgroundColor:[UIColor blackColor]];

   //set Hidden
    [imgView setHidden:NO];


    //set usre interaction
    [imgView setUserInteractionEnabled:YES];

    //set Tag
    [imgView setTag:1];

    //set Alpha
    [imgView setAlpha:1];

    //add imageView to main View
    [self.view addSubview:imgView];
    [imgView release];
}
```

Animation

```
    UIImageView *imgView = [[UIImageView alloc]
initWithFrame:CGRectMake(100, 200, 100, 100)];

    //Adding Animation images
    NSArray *imgsArray =[[NSArray alloc]
initWithObjects:[UIImage
imageNamed:@"intro_wait@2X.png"],[UIImage
imageNamed:@"intro_yellowwait@2X.png"], nil];
    [imgView setAnimationImages:imgsArray];

    //Duration
    [imgView setAnimationDuration:3];


    //Repeat Count
    [imgView setAnimationRepeatCount:-1];


    [imgView startAnimating];
    [self.view addSubview:imgView];
    [imgView release];
}
```

UIToolBar

Toolbars are for adding buttons of type UIBarButtonItem in a bar, usually along a view's bottom. With a little ingenuity, you can place just about anything on a toolbar, although some items you are not really adding to a toolbar, but rather you are placing over the toolbar. You can add the following controls to a toolbar: Bar Button Item, Fixed Space Bar Button Item, Flexible Space, Bar Button Item, Text Field, Switch, Slider, and Button.

UIActivityIndicatorview

A UIActivityIndicatorView class creates an animated indeterminate progress indicator. This control tells the user to "please wait, I'm processing." The control does not tell the user how long he or she must wait

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

@interface b15SliderSwitchViewController :
UIViewController
{


}

.m
- (void)viewDidLoad
{
    //for start Loading using MBProgressHUD
    [MBProgressHUD showHUDAddedTo:self.view
animated:YES];
     //for stop loading
     [MBProgressHUD hideHUDForView:self.view
animated:YES];
}

//Get the MBProgressHUD files from Google
```

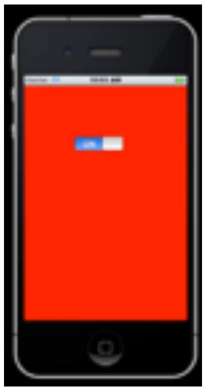
UISwitch

A UISwitch, similar to a toggle button, is on or off. UISwitch's appearance. A UISwitch has a property and method for changing its state. The Boolean property is on, when YES. The switch is off when NO.

.h

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

@interface b15SliderSwitchViewController :
UIViewController
{
    UISwitch *aSwitch;
}
@end
```

```
.m
-(void)switchChanged
{
    if(aSwitch.on)
    {
        self.view.backgroundColor = [UIColor redColor];
    }
    else
    {
        self.view.backgroundColor = [UIColor whiteColor];
    }
}
- (void)viewDidLoad
{
    aSwitch =[[UISwitch alloc]
initWithFrame:CGRectMake(100, 100, 100, 100)];

    //set On
    [aSwitch setOn:YES];

    //add target
    [aSwitch addTarget:self
action:@selector(switchChanged)
forControlEvents:UIControlEventValueChanged];

    //add to main view
    [self.view addSubview:aSwitch];
}
```



UISlider

Sliders are a horizontal bar with a small round indicator that a user can move to the right or left to change the slider's value

.h

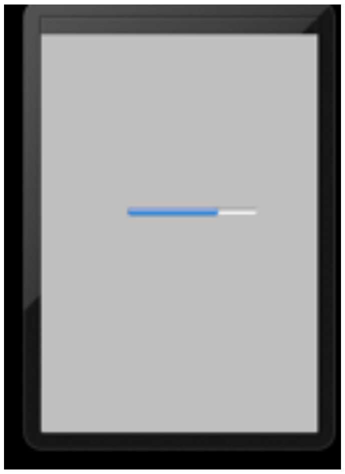
```
#import <UIKit/UIKit.h>
```

```
@interface b15SliderSwitchViewController :
UIViewController
{
    UILabel *lbl;
    UISlider *aSlider;
}
@end


.m

- (void)viewDidLoad
{
    self.view.backgroundColor = [UIColor
viewFlipsideBackgroundColor];

    //slider creation
    aSlider =[[UISlider alloc]
initWithFrame:CGRectMake(100, 100, 150, 40)];


    //minimum value
    [aSlider setMinimumValue:1];
    //maximum value
    [aSlider setMaximumValue:100];


    //set value
    [aSlider setValue:60];


    //Adding Target
    [aSlider addTarget:self
action:@selector(sliderChanged)
forControlEvents:UIControlEventValueChanged];


    //add to main view
    [self.view  addSubview:aSlider];

     //Label creation
    lbl = [[UILabel alloc]
initWithFrame:CGRectMake(260, 100, 50, 40)];
    [lbl setText:@"60"];
    [lbl setBackgroundColor:[UIColor clearColor]];
    [lbl setTextColor:[UIColor whiteColor]];
    [self.view addSubview:lbl];
    [super viewDidLoad];
}

-(void)sliderChanged
{
    NSString *valueStr= [NSString
stringWithFormat:@"%0.0f",aSlider.value];
    lbl.text = valueStr;
}
```

UIProgressView

```
-(void)timechanged
{
    progress.progress+=0.1;
    if(progress.progress==1)
```

```
        [aTimer invalidate];
}
- (void)viewDidLoad
{
     progress  = [[UIProgressView alloc]
initWithFrame:CGRectMake(100, 200, 150, 40)];

    [progress setProgress:0];


     aTimer = [NSTimer scheduledTimerWithTimeInterval:1
target:self selector:@selector(timechanged) userInfo:nil
repeats:YES];


    [self.view addSubview:progress];
}
```



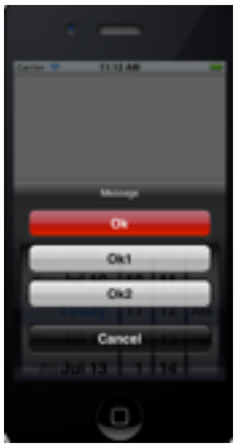
UIAlertView

```
//Delegate Method
- (void)alertView:(UIAlertView *)alertView
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    NSLog(@"clickedButtonAtIndex and button index is
%d",buttonIndex);
    if(buttonIndex == 0)
    {
        self.view.backgroundColor = [UIColor redColor];
    }
    else {
        self.view.backgroundColor = [UIColor
scrollViewTexturedBackgroundColor];
    }
}

- (void)viewDidLoad
{
    //Alert View
    UIAlertView *alertView = [[UIAlertView alloc]
initWithTitle:@"Message" message:@"Good morning"
delegate:nil cancelButtonTitle:nil
otherButtonTitles:@"Ok",@"cancel",nil];
```

```
    [alertView show];
}
```

UIActionSheet

```
- (void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    NSLog(@"clickedButtonAtIndex and button index is
%d",buttonIndex);


}
- (void)viewDidLoad
{
    UIActionSheet *sheet = [[UIActionSheet alloc]
initWithTitle:@"Message" delegate:self
cancelButtonTitle:@"Cancel"
destructiveButtonTitle:@"Ok"
otherButtonTitles:@"Ok1",@"Ok2", nil];

    //set style
    [sheet
setActionSheetStyle:UIActionSheetStyleBlackTranslu
cent];
    [sheet showInView:self.view];
}
```



Application Badges

The iPhone's Mail application illustrates using a badge. For instance, e.g. like e-mails in my inbox. Using this functionality is easy. Simply access your application's applicationBadgeNumber property and set it.

```
[UIApplication
sharedApplication].applicationIconBadgeNumber = 2;
```

UIScreen

```
  CGRect rc = [UIScreen mainScreen].applicationFrame;
    NSLog(@"Screen frame is (%0.0f,%0.0f,%0.0f,
%0.0f)",rc.origin.x,rc.origin.y,rc.size.width,rc.size.heigh
t);

    //Converting Rect to String
    NSLog(@"the frame is
%@",NSStringFromCGRect(rc));
```

UIColor

1.RGB

```
UIColor *color1 = [UIColor colorWithRed:0 green:0
blue:0 alpha:1];
```

2.Image

```
 UIImage *img2 = [UIImage
imageNamed:@"intro_wait@2X.png"];
```

```
    UIColor *color2 = [UIColor
colorWithPatternImage:img2];
```

3.class methods

```
    UIColor *color3= [UIColor redColor];
    UIColor *color4 = [UIColor blackColor];
```

UIFont

1.Bold and Font Size

```
      UIFont *font = [UIFont boldSystemFontOfSize:
18];
```

2.Font Size

```
    UIFont *font2 = [UIFont systemFontOfSize:18];
```

3.Font Name and Size

```
      UIFont *font3 = [UIFont fontWithName:@"" size:
18];
```

UIDevice

```
  //UDID -- 40 digit  simulator - 32
```

```
    UIDevice *device = [UIDevice currentDevice];

    //UDID
    NSString *udid = [device uniqueIdentifier];

    //name
    NSString *name = [device name];

    //version
    NSString *version = [device systemVersion];

    //Model
    NSString *model = [device model];

    //battery level
    float batteryLevel = [device batteryLevel]; // 0 to 1

    NSLog(@"udid = %@ \n  name = %@ \n version = %@
\n Model = %@ \n Battery Level =
%0.1f",udid,name,version,model,batteryLevel);
```

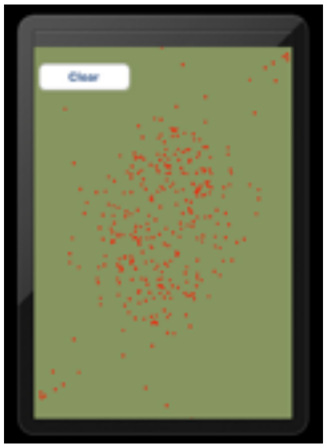
UIWindow

```
  //Getting root Window
 UIWindow *window = [UIApplication
sharedApplication].keyWindow;
```

Touches

```
//Touches
-(void)touchesBegan:(NSSet *)touches withEvent:
(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint touchPoint = [touch
locationInView:self.view];
    NSLog(@"point is (%0.0f,
%0.0f)",touchPoint.x,touchPoint.y);

    //getting Tap Count
    int tapCount = touch.tapCount;
    NSLog(@"tap count is %d",tapCount);

    //getting touch View
      if(touch.view == self.view)
      {
     }

    UILabel *lbl = [[UILabel alloc]
initWithFrame:CGRectMake(touchPoint.x, touchPoint.y,
3, 3)];
    [lbl setBackgroundColor:[UIColor redColor]];
    [self.view addSubview:lbl];
    [lbl release];
}

-(void)touchesMoved:(NSSet *)touches withEvent:
(UIEvent *)event
{
    for (int i=0; i<[touches count]; i++)
    {
        UITouch *touch = [[touches allObjects]
objectAtIndex:i];
        CGPoint touchPoint = [touch
locationInView:self.view];
        NSLog(@"point is (%0.0f,
%0.0f)",touchPoint.x,touchPoint.y);
```

```
        UILabel *lbl = [[UILabel alloc]
initWithFrame:CGRectMake(touchPoint.x, touchPoint.y,
3, 3)];
        [lbl setBackgroundColor:[UIColor redColor]];
        [self.view addSubview:lbl];
        [lbl release];
    }
}

-(IBAction)clearClick:(id)sender;
{
    NSArray *subViews = self.view.subviews;
    for(int i=0;i<[subViews count];i++)
    {
        UIView  *vi = [subViews objectAtIndex:i];
        if(![vi isKindOfClass:[UIButton class]])
           [vi removeFromSuperview];
    }
}
```



WebView

The UIWebView is the class you use to add a web browser to your application. It's based upon the same code foundation as Safari, and so you can use CSS and JavaScript. Using the web view can be easy, or more difficult, depending upon how much you wish your application to interact with the browser.

The UIWebView is responsible for the web view's display. It is an easy means of embedding web content in your application. The loadRequest: method is how you load web content. You can check on the control's progress loading a resource using the loading property. You might also wish moving forward or backward through the user's browsing history; you do this using the goBack and goForward methods.

The stringByEvaluatingJavaScriptFromString: method allows evaluating any JavaScript string. You can access a page's Document Object Model (DOM) through JavaScript, and so you can manipulate an HTML page's content.

The HTML DOM is a W3C standard for manipulating HTML documents.

case 1:
Loading url in a Web View

```
  //Web View
    UIWebView *webView = [[UIWebView alloc]
initWithFrame:CGRectMake(0, 0, 320, 460)];

    NSString *urlStr = @"http://www.google.com";
    NSURL *url = [NSURL URLWithString:urlStr];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
    [webView loadRequest:request];

    [webView setScalesPageToFit:YES];



    [self.view addSubview:webView];
    [webView release];
```

Case 2:Loading HTMl String

```
    UIWebView *webView = [[UIWebView alloc]
initWithFrame:CGRectMake(0, 0, 320, 460)];
    [webView loadHTMLString:@"<html><body>Hiiiii</
body></html>" baseURL:nil];
    [self.view addSubview:webView];
    [webView release];

case 3:
   //Displaying Video
    UIWebView *webView = [[UIWebView alloc]
initWithFrame:CGRectMake(0, 0, 320, 460)];

    NSString *urlStr = @"http://www.youtube.com/
video1";
    NSURL *url = [NSURL URLWithString:urlStr];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
    [webView loadRequest:request];

    [webView setScalesPageToFit:YES];


    [self.view addSubview:webView];
    [webView release];
```

```
case 4:
    //Loading Files from BUndle
    UIWebView *webView = [[UIWebView alloc]
initWithFrame:CGRectMake(0, 0, 320, 460)];
    NSString *filePath =[[NSBundle mainBundle]
pathForResource:@"sample" ofType:@"rtf"];
    NSURL *fileURl = [NSURL
fileURLWithPath:filePath];
    NSURLRequest *request = [NSURLRequest
requestWithURL:fileURl];
    [webView loadRequest:request];
    [self.view addSubview:webView];
    [webView release];

case 5:Displaying Files from Data
  UIWebView *webView = [[UIWebView alloc]
initWithFrame:CGRectMake(0, 0, 320, 460)];
    NSData *fileData = [[NSData alloc] init];
    [webView loadData:fileData MIMEType:@""
textEncodingName:@"NSUTF8StringEncoding"
baseURL:nil];
    [self.view addSubview:webView];
    [webView release];
```

```
-(void)back
{
    [webView goBack];
}
-(void)forward
{
    [webView goForward];
}
-(void)reload
{
    [webView reload];
}
-(void)stop
{
    [webView stopLoading];
}
```

WebView Delegate Methods

```
- (BOOL)webView:(UIWebView *)webView
shouldStartLoadWithRequest:(NSURLRequest
*)request navigationType:
(UIWebViewNavigationType)navigationType
```

```
{
   NSURL *loadUrl = [request URL];
    NSLog(@"load url is %@",loadUrl);
    return YES;
}
- (void)webViewDidStartLoad:(UIWebView *)webView
{
    NSLog(@"webViewDidStartLoad called");

}
- (void)webViewDidFinishLoad:(UIWebView *)webView
{
    NSLog(@"webViewDidFinishLoad called");
}
- (void)webView:(UIWebView *)webView
didFailLoadWithError:(NSError *)error
{
     NSLog(@"didFailLoadWithError called and error
info is %@",[error userInfo]);
}
```

Audio Player

```
//Delegate Methods for Audio Player
- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer
*)player successfully:(BOOL)flag
{
    if(flag)
        NSLog(@"Successful Play");
    else
        NSLog(@"unsuccessful play");
}
- (void)audioPlayerDecodeErrorDidOccur:
(AVAudioPlayer *)player error:(NSError *)error
{
    NSLog(@"audioPlayerDecodeErrorDidOccur called
and error is %@",[error userInfo]);
}
-(void)viewDidLoad
{
    //Audio Player
    NSString *path = [[NSBundle mainBundle]
pathForResource:@"audiofile" ofType:@"mp3"];
    NSURL *url = [NSURL fileURLWithPath:path];
    AVAudioPlayer *player = [[AVAudioPlayer alloc]
initWithContentsOfURL:url error:nil];
```

```
[player setDelegate:self];

//number of loops
[player setNumberOfLoops:100];

//volume
[player setVolume:1];

//Song Duration
float duration = [player duration];
NSLog(@"Total Duration = %f",duration);

//current Duration
float currentDuration = [player currentTime];

//play
[player play];


/*
//pause
[player pause];
```

```
    //stop
    [player stop];
    */
 }
```

Media Playing

```
#import <UIKit/UIKit.h>
#import <MediaPlayer/MediaPlayer.h>
@interface ViewController : UIViewController {
MPMoviePlayerViewController *moviePlayer;
}
-(IBAction)playVideo:(id)sender;
@end
```

.m

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

UITextField

The iPhone uses the UITextField class to render text fields. A text field is associated with a keyboard that appears when a user taps in the text field. Keyboard styles include Number Pad, Phone Pad, URL, and several others. You can set the keyboard style programmatically or using Interface Builder

```
-(void)viewDidLoad
{
    //Text Field
    UITextField *txtField = [[UITextField alloc]
initWithFrame:CGRectMake(100, 100, 100, 30)];

    //set tag
    [txtField setTag:1];

    //set Delegate
    [txtField setDelegate:self];

    //set Text
   // [txtField setText:@"Hello"];

    //set Place holder string
    [txtField setPlaceholder:@"Enter"];

    //set Border Style
    [txtField setBorderStyle:UITextBorderStyleNone];

    //set Clear Button
    [txtField 
setClearButtonMode:UITextFieldViewModeWhileEditin
g];

    //set Key Board Type
    [txtField 
setKeyboardType:UIKeyboardTypeEmailAddress];

    //set keyboard Appearence
    [txtField 
setKeyboardAppearance:UIKeyboardAppearanceDefaul
t];
```

```
    //set Return key Type
    [txtField setReturnKeyType:UIReturnKeySearch];

    //set back ground Image
    [txtField setBackground:[UIImage
imageNamed:@"1.png"]];

    [self.view addSubview:txtField];
    [txtField release];
}
//Delegate Methods for text Field
- (void)textFieldDidBeginEditing:(UITextField
*)textField
{
    NSLog(@"textFieldDidBeginEditing called");
    self.view.backgroundColor = [UIColor
groupTableViewBackgroundColor];
}
- (void)textFieldDidEndEditing:(UITextField
*)textField
{
    NSLog(@"textFieldDidEndEditing called");
    self.view.backgroundColor = [UIColor
scrollViewTexturedBackgroundColor];
}
```

```
- (BOOL)textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
    NSLog(@"Enetered Charecter is %@",string);
    return YES;
}
-(BOOL)textFieldShouldClear:(UITextField
*)textField
{
    NSLog(@"textFieldShouldClear called");
    return YES;
}
- (BOOL)textFieldShouldReturn:(UITextField
*)textField
{
    [textField resignFirstResponder];
    return YES;
}
```

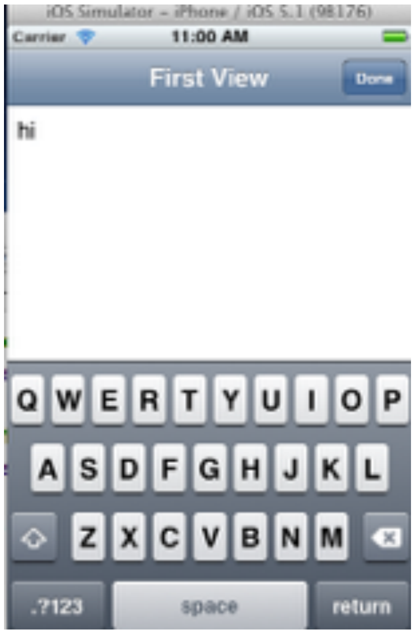
UITextView

UITextView to capture multiple lines of text. Apple's reference documentation describes this control best: "The UITextView class implements the behavior for a

scrollable, multiline text region." There is not really much more you can say about a text view, other than it's a text area,
it is generally used for entering paragraphs of text rather than a single line. There are several properties you can set to customize the control's appearance; including, the font, textColor, editable, and textAlignment properties. You can also check if it has text using the hasText method.

```
-(IBAction)nextClick:(id)sender
{
    [selectedTxtView resignFirstResponder];
}
- (void)textViewDidBeginEditing:(UITextView
*)textView
{
    selectedTxtView = textView;

}
-(void)viewDidLoad
{
     txtView = [[UITextView alloc]
initWithFrame:CGRectMake(0, 44, 320, 460)];
```

```
    //set Text
    [txtView setText:@"hi"];

    //set Font
    [txtView setFont:[UIFont systemFontOfSize:18]];

    //set Delegate
    [txtView setDelegate:self];

    //set Editable
    [txtView setEditable:YES];

    [self.view addSubview:txtView];
}
```



UISegmentedControl

A segmented control groups together two or more segments, where each segment acts as an independent button. The next task illustrates a segmented control.

.h

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

@interface b15SliderSwitchViewController :
UIViewController
{
    UISegmentedControl *segment;
}
@end
```

.m

```
-(void)segmentchanged
{
    if(segment.selectedSegmentIndex == 0)
    {
        self.view.backgroundColor = [UIColor redColor];
    }
    else if(segment.selectedSegmentIndex ==1)
    {
        self.view.backgroundColor = [UIColor
viewFlipsideBackgroundColor];
    }
    else if(segment.selectedSegmentIndex ==2)
    {
         self.view.backgroundColor = [UIColor
scrollViewTexturedBackgroundColor];
    }

}
- (void)viewDidLoad
{
    NSArray *segmenstArray = [NSArray
arrayWithObjects:@"first",@"second",@"third", nil];
     segment = [[UISegmentedControl alloc]
initWithItems:segmenstArray];

    //set style
    [segment
setSegmentedControlStyle:UISegmentedControlStyleB
ar];
```

```
    //set frame
    [segment setFrame:CGRectMake(80, 100, 200, 30)];

    //add target
    [segment addTarget:self
action:@selector(segmentchanged)
forControlEvents:UIControlEventValueChanged];


    //set image for segments
    [segment setImage:[UIImage imageNamed:@"" ]
forSegmentAtIndex:0];


    //set Title
    [segment setTitle:@"one" forSegmentAtIndex:0];


    //add to main view
    [self.view addSubview:segment];

}
```

UIView Animations

View Transitions are effective way of adding one view on another view with a proper transition animation effect.

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

-(void)setUpView{
    view1 = [[UIView
alloc]initWithFrame:self.view.frame];
    view1.backgroundColor = [UIColor lightTextColor];
    view2 = [[UIView
alloc]initWithFrame:self.view.frame];
    view2.backgroundColor = [UIColor orangeColor];
    [self.view addSubview:view1];
    [self.view sendSubviewToBack:view1];

}

-(void)doTransitionWithType:
(UIViewAnimationTransition)animationTransitionType{
    if ([[self.view subviews] containsObject:view2 ]) {
        [UIView transitionFromView:view2
                            toView:view1
                          duration:2
                           options:animationTransitionType
                        completion:^(BOOL finished){
                            [view2 removeFromSuperview];
                        }];
        [self.view addSubview:view1];
        [self.view sendSubviewToBack:view1];
    }
    else{

        [UIView transitionFromView:view1
                            toView:view2
                        duration:2
                         options:animationTransitionType
                      completion:^(BOOL finished){
                          [view1 removeFromSuperview];
                      }];
        [self.view addSubview:view2];
        [self.view sendSubviewToBack:view2];

    }
}


-(IBAction)flipFromLeft:(id)sender
{
    [self
doTransitionWithType:UIViewAnimationOptionTransiti
onFlipFromLeft];

}
-(IBAction)flipFromRight:(id)sender{
    [self
doTransitionWithType:UIViewAnimationOptionTransiti
onFlipFromRight];

}
```

```
-(IBAction)flipFromTop:(id)sender{
    [self
doTransitionWithType:UIViewAnimationOptionTransiti
onFlipFromTop];

}
-(IBAction)flipFromBottom:(id)sender{
    [self
doTransitionWithType:UIViewAnimationOptionTransiti
onFlipFromBottom];

}
-(IBAction)curlUp:(id)sender{
    [self
doTransitionWithType:UIViewAnimationOptionTransiti
onCurlUp];
```

```
}
-(IBAction)curlDown:(id)sender{
    [self
doTransitionWithType:UIViewAnimationOptionTransiti
onCurlDown];
```

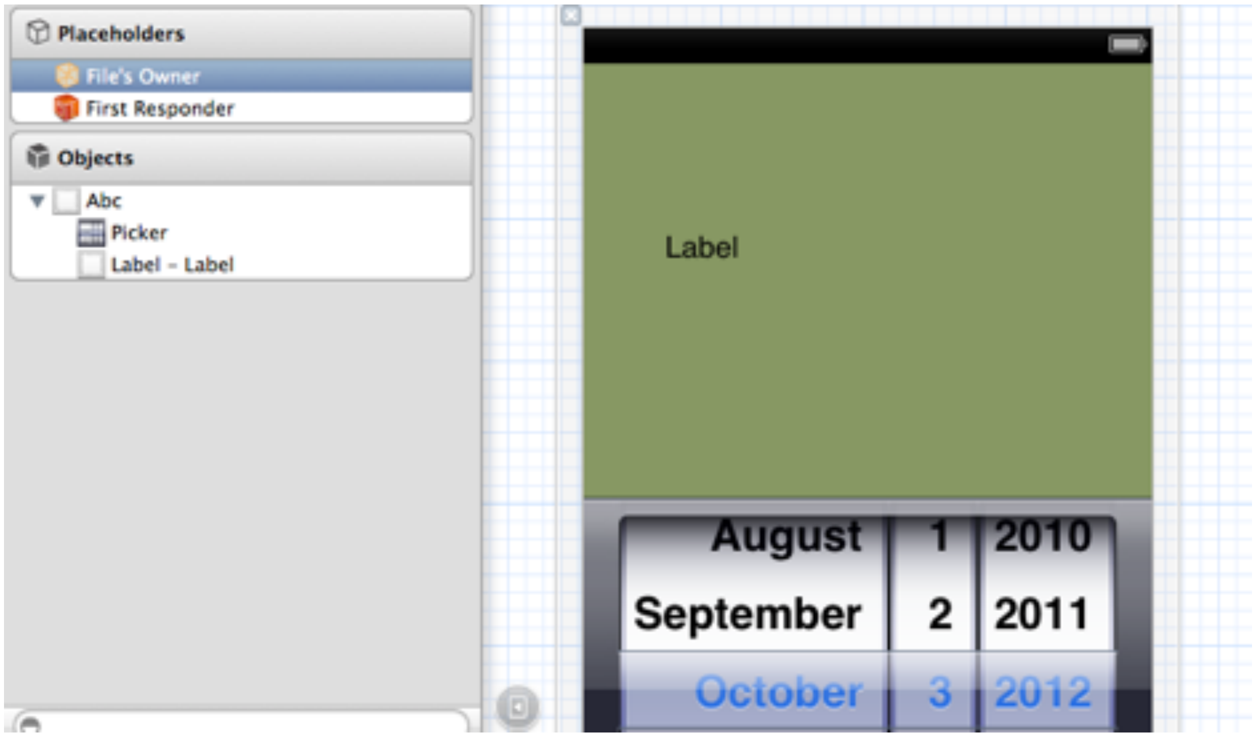
```
}
-(IBAction)dissolve:(id)sender{
    [self 
doTransitionWithType:UIViewAnimationOptionTransiti
onCrossDissolve];

}
-(IBAction)noTransition:(id)sender{
    [self 
doTransitionWithType:UIViewAnimationOptionTransiti
onNone];

}
```

UIDatePicker

Date pickers pick dates and times, placing the selected values in an NSDate class, and are implemented using the UIDatePicker class. If using Interface Builder to develop an iPhone application, you drag a date picker onto your view's canvas from the library and modify the picker’s properties in the Inspector. You then create an IBOutlet for the picker in your view’s associated view controller and connect it to the picker on your view’s canvas. The view controller uses this outlet to obtain the UIDatePicker object’s selected date. The example later in this section illustrates this process.

```
.h
@interface ViewController : UIViewController
<UITextFieldDelegate>
{
    IBOutlet UILabel *lblDate;
    IBOutlet UIDatePicker *datePicker1;
}
-(IBAction)dateChanged:(id)sender;

@end



.m
-(void)viewDidLoad
{
    NSDateFormatter *formatter =
[[ NSDateFormatter alloc] init];
```

```
    [formatter
setDateStyle:NSDateFormatterShortStyle];
    [formatter
setTimeStyle:NSDateFormatterNoStyle];
    [formatter setDateFormat:@"YYYY-MM-dd"];
    lblDate.text = [formatter stringFromDate:[NSDate
date]];

//"EEEE MMMM d',' yyyy" format outputs "Tuesday
August 26, 2008,"
//"MM/dd/yyyy" format outputs August 26, 2008, as
"08/26/2008."



}
-(IBAction)dateChanged:(id)sender
{
    NSDateFormatter *formatter =
[[ NSDateFormatter alloc] init];
    [formatter
setDateStyle:NSDateFormatterShortStyle];
    [formatter
setTimeStyle:NSDateFormatterNoStyle];
    [formatter setDateFormat:@"YYYY-MM-dd"];
    lblDate.text = [formatter
stringFromDate:datePicker1.date];
}
```



iAD Integeration

IAD is used for display ads served by the apple server and helps us in earning revenue from the application. 3. Update ViewController.h as follows.
4. Update ViewController.m as follows.
Steps Involved
1. Create a simple View based application.

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

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

#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];
}
```

```
#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
```

UIPickerView

A UIPickerView allows the selection of one or more value sets. A UIPickerView consists of rows and components. Think of the component as the column and the row as the row. So if you had a three-wheel UIPickerView, the third wheel is the third component. A UIPickerView must have an associated class that adopts the UIPickerViewDelegate and a class that adopts the UIPickerViewDataSource. The same class can adopt both protocols.

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController
{
    IBOutlet UILabel *lbl;
    NSMutableArray *colorsArray,*studentsArray;
    IBOutlet UITextField *txtField;
    IBOutlet UIPickerView *picker;



}
-(IBAction)addClick:(id)sender;
@end

.m
-(void)viewDidLoad
{
    colorsArray = [[NSMutableArray alloc]
initWithObjects:@"red",@"white",@"black",@"orange",
@"violet", nil];
    studentsArray= [[NSMutableArray alloc]
initWithObjects:@"ramu",@"raju",@"kumar",@"prasad
",nil];
}
-(IBAction)addClick:(id)sender
{
    if([txtField.text length]>0 && ![colorsArray
containsObject:txtField.text])
        [colorsArray addObject:txtField.text];
    [picker reloadAllComponents];
}
```

```
- (BOOL)textFieldShouldReturn:(UITextField
*)textField
{
    [textField resignFirstResponder];
}
//data Source Methods For Picker View
- (NSInteger)numberOfComponentsInPickerView:
(UIPickerView *)pickerView
{
    return 2;
}

// returns the # of rows in each component..
- (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component
{
    if(component == 0)
       return [colorsArray count];
    else if(component ==1)
        return [studentsArray count];
}
- (NSString *)pickerView:(UIPickerView *)pickerView
titleForRow:(NSInteger)row forComponent:
(NSInteger)component
{
    if(component ==0)
        return [colorsArray objectAtIndex:row];
    else if(component ==1)
        return [studentsArray objectAtIndex:row];
}

//Delegate Method
- (void)pickerView:(UIPickerView *)pickerView
didSelectRow:(NSInteger)row inComponent:
(NSInteger)component
{
    if(component ==0)
        lbl.text = [colorsArray objectAtIndex:row];
    else if(component ==1)
        lbl.text = [studentsArray objectAtIndex:row];
}
```

.xib

UITableView

Table views display content in a list. Tables have a single column and multiple rows. They can scroll vertically and display large data sets. For example, the Notes application is a good example of an application containing a table.

You can index tables and create tables with zero or more sections. When you create a table, you have a choice of two styles: UITableViewStylePlain or UITableViewStyleGrouped

1.Plain Table View
2.grouped table View

UITableView classes have an associated UITableViewController, a UITableViewDelegate, and a UITableViewDataSource. The UITableViewController is the controller class for a table view. You create an instance of this class to manage the UITableView. The UITableViewDelegate is a protocol you adopt in a custom class you write. This protocol allows you to manage selections, configure headers and footers, and manage cells. The UITableViewDataSource is also a protocol you adopt in a custom class. This protocol allows you to manage a table's data source

UITableViewDelegate and UITableViewDataSource

The UITableViewDelegate and UITableViewDataSource are protocols at least one class
in your application must adopt if your application contains a UITableView. You can create your own custom classes to adopt these protocols, or create a UITableViewController that automatically adopts these protocols. If you choose to use a UITableViewController rather than custom classes, you simply connect the table view's dataSource and delegate outlets to the UITableViewController. You can

then implement both protocols' methods in the UITableViewController.

Delegate Methods for UITableViewDelegate:

tableView:heightForRowAtIndexPath:

tableView:accessoryButtonTappedForRowWithIndexPath:

tableView:willSelectRowAtIndexPath:

tableView:didSelectRowAtIndexPath:

tableView:editingStyleForRowAtIndexPath:

Delegate Methods for UITableViewDataSource

numberOfSectionsInTableView:

tableView:numberOfRowsInSection:

tableView:cellForRowAtIndexPath:

sectionIndexTitlesForTableView:

tableView:commitEditingStyle: forRowAtIndexPath:

tableView:canEditRowAtIndexPath:

tableView:moveRowAtIndexPath:toIndexPath:

UITableViewController

The UITableViewController manages a table view. The UITableView can use objects defined in a table's nib to define a table's delegate and data source, or it can use itself as the delegate and data source. For instance, in the previous example, you set the table's delegate and data source properties to the TableHandler class. You could have added a UITableViewController, set it as the table's File's Owner, and then set its outlets to TableHandler.

If you do not provide a delegate and data source in a table's nib, a UITableViewController sets its data source and delegate to itself. By doing this, the UITableViewController saves you the work of having to create your own classes so they adopt the delegate and data source. You still must implement any data source and delegate methods desired. However, rather than implementing these methods in separate custom classes, you implement them in a

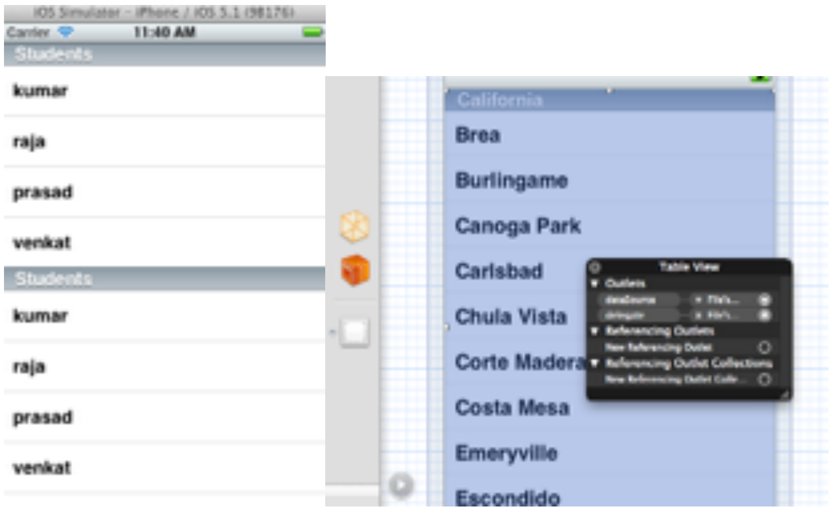
UITableViewController subclass. The UITableViewController then functions as the table's controller, delegate, and data source.

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController
{
   NSMutableArray *studentsArray;
}
@end
```

```
.m
@implementation ViewController

-(void)viewDidLoad
{
    studentsArray =[[NSMutableArray alloc]
initWithObjects:@"kumar",@"raja",@"prasad",@"venk
at", nil];

}

#pragma UITableView Data Source Methods
- (NSInteger)numberOfSectionsInTableView:
(UITableView *)tableView
{
    return 2;
}
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{

    return [studentsArray count];
}
- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 50;
}
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
```

```
{
    return @"Students";
}
- (UITableViewCell *)tableView:(UITableView
*)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if(!cell)
    {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:CellIdentifier];
    }

    //Configure cell
    cell.textLabel.text = [studentsArray
objectAtIndex:indexPath.row];

    return cell;
}
```

.xib



Displaying Array of Dicts in a Table View

.h

```
#import <UIKit/UIKit.h>
#import "Second.h"
@interface ViewController : UIViewController
{
    NSMutableArray *studentsArray;
}
@end
```

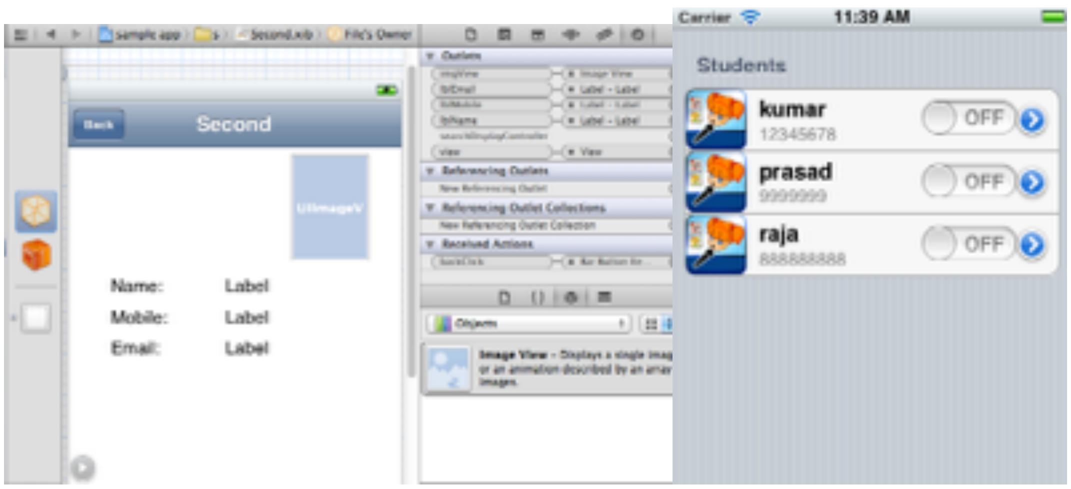
```
.m

-(void)viewDidLoad
{
    [self prepareMyArray];



}
-(void)prepareMyArray
{
    //Preparation of an Array
    studentsArray =[[NSMutableArray alloc] init];
    NSDictionary *aDict = [[NSDictionary alloc]
initWithObjectsAndKeys:@"kumar",@"name",@"12345
678",@"number",@"abc@gmail.com",@"email",
[UIImage imageNamed:@"1.png"],@"image", nil];
    NSDictionary *bDict = [[NSDictionary alloc]
initWithObjectsAndKeys:@"prasad",@"name",@"9999
999",@"number",@"prasad@gmail.com",@"email",
[UIImage imageNamed:@"1.png"],@"image", nil];
    NSDictionary *cDict = [[NSDictionary alloc]
initWithObjectsAndKeys:@"raja",@"name",@"888888
888",@"number",@"raja@gmail.com",@"email",
[UIImage imageNamed:@"1.png"],@"image", nil];
    [studentsArray addObject:aDict];
    [studentsArray addObject:bDict];
    [studentsArray addObject:cDict];
}

#pragma UITableView Data Source Methods
- (NSInteger)numberOfSectionsInTableView:
(UITableView *)tableView
{
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return [studentsArray count];
}
- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 50;
}
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
```

```
{

    return @"Students";
}
- (UITableViewCell *)tableView:(UITableView
*)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if(!cell)
    {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:CellIdentifier];
    }

    //Configure cell
    NSDictionary *aDict = [studentsArray
objectAtIndex:indexPath.row];
    cell.imageView.image =  [aDict
objectForKey:@"image"];
    cell.textLabel.text = [aDict objectForKey:@"name"];
    cell.detailTextLabel.text = [aDict
objectForKey:@"number"];
    [cell
setAccessoryType:UITableViewCellAccessoryDetailDis
closureButton];

    //Adding Switch
    UISwitch *aSwitch = [[UISwitch alloc]
initWithFrame:CGRectMake(200, 10, 30, 30)];
    [cell addSubview:aSwitch];

    return cell;
}

//Delegate Method
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSLog(@"didSelectRowAtIndexPath called");
    Second *secc = [[Second alloc]
initWithNibName:@"Second" bundle:nil];
    secc.selectedDict = [studentsArray
objectAtIndex:indexPath.row];
    [self presentModalViewController:secc
animated:YES];
```

```
}

@end

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

@interface Second : UIViewController
{
   NSDictionary *selectedDict;
   IBOutlet UILabel *lblName,*lblEmail,*lblMobile;
   IBOutlet UIImageView *imgView;
}
@property(nonatomic,retain) NSDictionary
*selectedDict;

-(IBAction)backClick:(id)sender;
@end


second.m
- (void)viewDidLoad
{
   lblName.text = [selectedDict
objectForKey:@"name"];
```

```
    lblMobile.text = [selectedDict 
objectForKey:@"number"];
    lblEmail.text = [selectedDict 
objectForKey:@"email"];
    imgView.image = [selectedDict 
objectForKey:@"image"];
}
```

second.xib



Navigation based Application

//Taking View base as Navigation based Application

AppDelegate.m

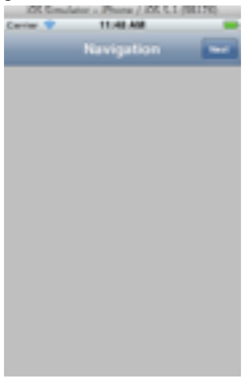
```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:
[[UIScreen mainScreen] bounds]];
    // Override point for customization after application
launch.
    self.viewController = [[ViewController alloc]
initWithNibName:@"ViewController" bundle:nil];
    UINavigationController *navController =
[[UINavigationController alloc]
initWithRootViewController:self.viewController];


    self.window.rootViewController =  navController; //
self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

viewController.h

```
#import <UIKit/UIKit.h>
#import "Second.h"
@interface ViewController : UIViewController

@end
```

viewController.m

```
-(void)nextClick
{
    Second *sec = [[Second alloc]
initWithNibName:@"Second" bundle:nil];
    [self.navigationController pushViewController:sec
animated:YES ];
}
- (void)viewDidLoad
{
    self.title = @"Navigation";
    self.navigationItem.rightBarButtonItem =
[[UIBarButtonItem alloc] initWithTitle:@"Next"
style:UIBarButtonItemStylePlain target:self
action:@selector(nextClick)];

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



Second.h

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

@interface Second : UITableViewController
{
    NSMutableArray *studentsArray;
}
@end
```

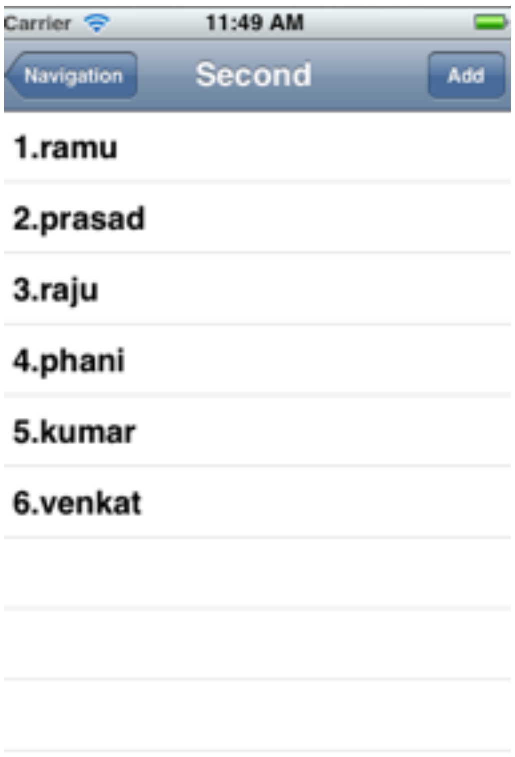
Second.m

```
-(void)viewWillAppear:(BOOL)animated
{
    [self.tableView reloadData];
}
```

```
-(void)addClick
{
    [studentsArray addObject:@"new Student"];
    [self.tableView reloadData];
}
- (void)viewDidLoad
{
    self.title = @"Second";
    studentsArray = [[NSMutableArray alloc]
initWithObjects:@"ramu",@"prasad",@"raju",@"phani"
,@"kumar",@"venkat", nil];

   //Edit Functionality
    //self.navigationItem.rightBarButtonItem =
self.editButtonItem;

    //Add
    self.navigationItem.rightBarButtonItem =
[[UIBarButtonItem alloc] initWithTitle:@"Add"
style:UIBarButtonItemStylePlain target:self
action:@selector(addClick)];
}

- (NSInteger)numberOfSectionsInTableView:
(UITableView *)tableView
```

```
{
#warning Potentially incomplete method
implementation.
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
#warning Incomplete method implementation.
    // Return the number of rows in the section.
    return [studentsArray count];
}

- (UITableViewCell *)tableView:(UITableView
*)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if(!cell)
    {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier];
    }
    // Configure the cell...
    cell.textLabel.text = [NSString
stringWithFormat:@"%d.%@",indexPath.row+1,
[studentsArray objectAtIndex:indexPath.row]];

    return cell;
}

// Override to support editing the table view.
- (void)tableView:(UITableView *)tableView
commitEditingStyle:
(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle ==
UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source

        [studentsArray
removeObjectAtIndex:indexPath.row];
```

```
        [tableView
deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
        [tableView reloadData];
    }
    else if (editingStyle ==
UITableViewCellEditingStyleInsert) {
        // Create a new instance of the appropriate class,
insert it into the array, and add a new row to the table
view
    }
}
```



1.ramu
2.prasad
3.raju
4.phani
5.kumar
6.venkat

UIImagePickerController

UIImagePickerController to manipulate an iPhone's camera and photo library. Using the UIImagePickerController, you take, or select, a photo, optionally edit the photo, and then dismiss the UIImagePickerController, returning control back to your application.

The UIImagePickerController is different from other view controllers. Rather than developers creating the controller's view and adding components to the view's canvas, the UIImage PickerController's views are already created and are part of the UIKit library.Developers simply determine the controller's source type and implement a delegate for the controller. The controller creates and manages the views while the delegate responds to the view being dismissed by a user.

```
-(void)viewDidLoad
{
    //to save an image into Photo Library
       UIImageWriteToSavedPhotosAlbum([UIImage
imageNamed:@"1.png"], nil, nil, nil);
}
-(IBAction)takePhoto:(id)sender
{

    UIImagePickerController *imagePicker =
[[ UIImagePickerController alloc] init];

    //set Source Type
    [imagePicker
setSourceType:UIImagePickerControllerSourceTypePh
otoLibrary];

    //set Editing
    [imagePicker setEditing:YES];

    //set Delegate
    [imagePicker setDelegate:self];
    [self presentModalViewController:imagePicker
animated:YES];
}
//Delegate Methods For Image Picker
- (void)imagePickerController:
(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    NSLog(@"info is %@",info);
   //  UIImage *img = [info objectForKey:@""];
    [self dismissModalViewControllerAnimated:YES];
}
- (void)imagePickerControllerDidCancel:
(UIImagePickerController *)picker
{
      [self dismissModalViewControllerAnimated:YES];
}
```

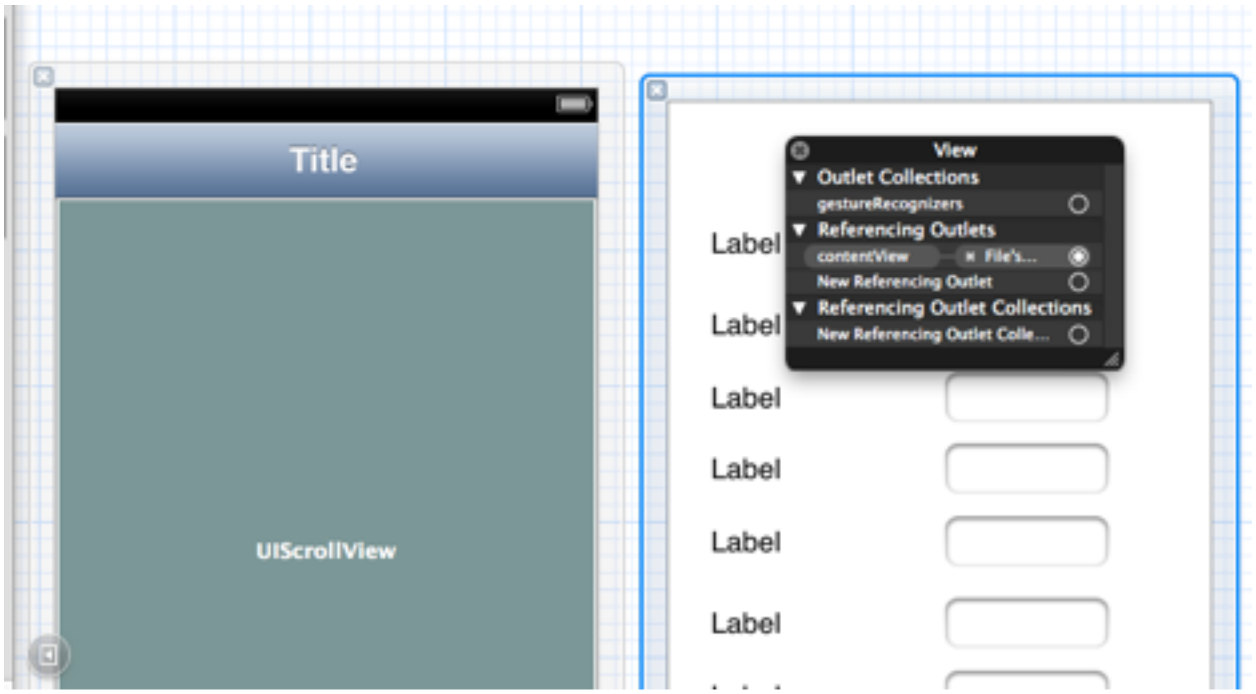
UIScrollView

1.Image Scrolling

```
  UIScrollView *scroll = [[UIScrollView alloc]
initWithFrame:CGRectMake(50, 50, 250, 100)];

    UIImage *img = [UIImage imageNamed:@"1.png"];
    CGSize imgSize = img.size;

    UIImageView *imgView = [[UIImageView alloc]
initWithFrame:CGRectMake(0, 0, imgSize.width,
imgSize.height)];
    [imgView setImage:img];
    [scroll addSubview:imgView];

    [self.view addSubview:scroll];
    [scroll setContentSize:imgView.frame.size];
```

2.View Scrolling

```
  [scroll addSubview:contentView];
    [scroll setContentSize:contentView.frame.size];
```



3.paging

```
-(void)viewDidLoad
{
    int width = 320;
    UIView *vi = [[UIView alloc]
initWithFrame:CGRectMake(0, 0, width*10, 460)];
    int x = 0;
    int y = 0;
    for(int i= 0 ;i<10;i++)
    {
        UIImageView *imgView = [[UIImageView alloc]
initWithFrame:CGRectMake(x, y, width, 460)];
```

```
        [imgView setBackgroundColor:[ViewController
randomColor]];
        [vi addSubview:imgView];
        [imgView release];

        x= x+width;
    }
    UIScrollView *scroll = [[UIScrollView alloc]
initWithFrame:CGRectMake(0, 0, 320, 460)];
    [scroll addSubview:vi];
    scroll.pagingEnabled = YES;
scroll.delegate = self;
    [self.view addSubview:scroll];
    [scroll setContentSize:vi.frame.size];
    [scroll release];

}


//Scroll Delegate Methods
- (void)scrollViewWillBeginDecelerating:(UIScrollView
*)scrollView
{

}
```

```
- (void)scrollViewDidEndDecelerating:(UIScrollView
*)scrollView
{
    int pageNo = scrollView.contentOffset.x/320;

    NSLog(@"page no is %d",pageNo);
}
```

The Settings Bundle

An application's preferences are stored in an Extended Markup Language (XML) file called Root.plist. Root.plist is stored in a bundle called Settings.bundle. A settings bundle is not automatically added to your project, and so you must add a settings bundle to your application if you wish to use the Settings application (Figure 14-2). Besides a Root. plist, a setting bundle contains any additional .plist files, any images used for sliders, and one or more .lproj files. Additional .plist files are for any child preference panes your application might require.

Map VIew

.h

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

@interface ViewController : UIViewController
{
    MKMapView *mapView;
}
@end
```

.m

```
- (void)mapView:(MKMapView *)mapView
didFailToLocateUserWithError:(NSError *)error
{


}
```

```
- (void)mapView:(MKMapView *)mapView
didUpdateUserLocation:(MKUserLocation
*)userLocation
{
    MKReverseGeocoder *geoCoder =
[[MKReverseGeocoder alloc]
initWithCoordinate:userLocation.coordinate];
    [geoCoder setDelegate:self];
    [geoCoder start];
}

- (void)reverseGeocoder:(MKReverseGeocoder
*)geocoder didFindPlacemark:(MKPlacemark
*)placemark
{
     //placemark.country

    //placemark.postalCode

   //placemark.subLocality
 }
- (void)reverseGeocoder:(MKReverseGeocoder
*)geocoder didFailWithError:(NSError *)error
{
}

-(void)viewDidLoad
{
    //showing map view user current Location
    [mapView showsUserLocation];

    //showing Map type
    [mapView setMapType:MKMapTypeStandard];

    //set Delegate
    [mapView setDelegate:self];

}
```

Call Sms and Email

```
-(IBAction)callClicked
{
[[UIApplication sharedApplication] openURL:[NSURL
URLWithString:[NSString stringWithFormat:@"tel:
```

```
%@",phoneNo.text]]];
}
#pragma mark SMSApplication
-(IBAction)smsClicked
{
[[UIApplication sharedApplication] openURL:[NSURL
URLWithString:[NSString stringWithFormat:@"sms:
%@?msg:%@",@"8894879234723",@""Hello]]];
}
#pragma mark EmailApplication
-(IBAction)emailClicked
{
[[UIApplication sharedApplication] openURL:[NSURL
URLWithString:[NSString
stringWithFormat:@"mailto:%@?
subject=testMail&body=its test mail.",email.text]]];
}
```

Mail Composer

```
    MFMailComposeViewController *mailComposer =
[[MFMailComposeViewController alloc] init];
    [mailComposer setSubject:@""];
    [mailComposer setToRecipients:<#(NSArray *)#>];
    [mailComposer setCcRecipients:<#(NSArray *)#>];
    [mailComposer setBccRecipients:];
    [mailComposer setMessageBody:<#(NSString *)#>
isHTML:NO];

    [self presentViewController:mailComposer
animated:YES completion:nil];
```

Message Composer

```
    MFMessageComposeViewController
*messageComposer =
[[MFMessageComposeViewController alloc] init];
    [messageComposer
setMessageComposeDelegate:self];
    [messageComposer setBody:@""];
    [messageComposer setRecipients:<#(NSArray *)#>];

    [self presentViewController:messageComposer
animated:YES completion:nil];
```

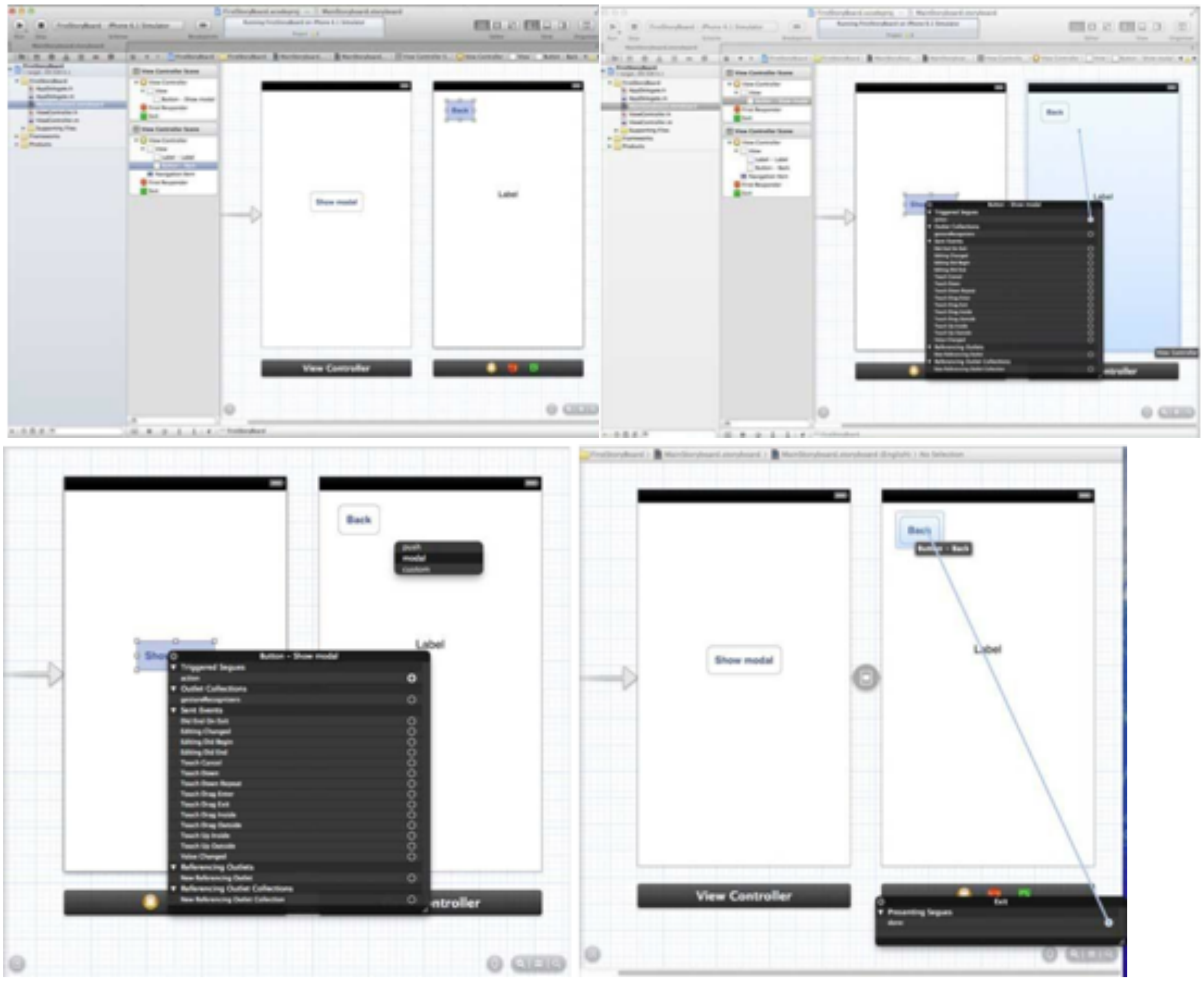
Navigation bar

Navigation bar is displayed along a window's top, just below the status bar. The navigation bar optionally contains a title for each view displayed and one or more buttons. Typically, the button on the bar's right, when clicked, takes the user to the next step in another view,
although sometimes applications use the right button as a "done" button. The button on the bar's left is a Back button, returning to the previous view. Navigation bars are often used with tables, where a table is displayed with a navigation bar above it. When a user clicks a table's rows, the navigation bar's associated navigation controller takes the user to more detailed information about the selected item

More on the UINavigationController

You are not limited to pushing items onto a stack. You can also pop items off the stack. You can also modify the navigation bar, hiding elements, adding new ones, and making other modifications.

You pop view controllers from a stack using the method



popViewControllerAnimated:. This method pops the top view controller off the navigation controller's stack and updates the displayed view to the stack's next view controller's view.

(UIViewController *) popViewControllerAnimated:(BOOL) animated
The method takes one parameter, which indicates if the transition should be animated. The method also returns the popped view controller, should you wish to retain it for future use.

Other methods you might use include popToRootViewControllerAnimated: and popToViewController:animated:. Refer to Apple's online reference, "UINavigationController Class Reference," for more information.

We can hide the navigation bar by calling the navigation controller's setNavigationBarHidden: method.

- (void)setNavigationBarHidden:(BOOL)hidden animated:(BOOL)animated

Storyboards

Storyboards are introduced in iOS 5 and when we use storyboards our deployment target should be 5.0 or higher. Storyboards helps us visual create all the screens of the application along with their flow of screens under one interface MainStoryboard.storyboard. It also helps in reducing coding of pushing/presenting view controllers.

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

2. Select MainStoryboard.storyboard where you can find single view controller. Now add one more view controller and update the view controllers as shown below.

3. Now let connect both the view controllers. For this right click on the button name "show modal" and drag it to the right view controller in the left side view controller as shown below.

4. Now select modal from the three options displayed as shown below.

5. Update ViewController.h as follows.

6. Update ViewController.m as follows.

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

```
@interface ViewController : UIViewController -
     (IBAction)done:(UIStoryboardSegue *)seque;
     @end

#import "ViewController.h"

@interface ViewController () @end

@implementation ViewController

- (void)viewDidLoad

{

[super viewDidLoad];

}

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

[self.navigationController
     popViewControllerAnimated:YES]; }

@end
```

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

Twitter and Facebook

Twitter has been integrated to 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.

1. Create a simple View based application.

2. Select your project file, then select targets and then add Social.framework andAccounts.framework in choose frameworks

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

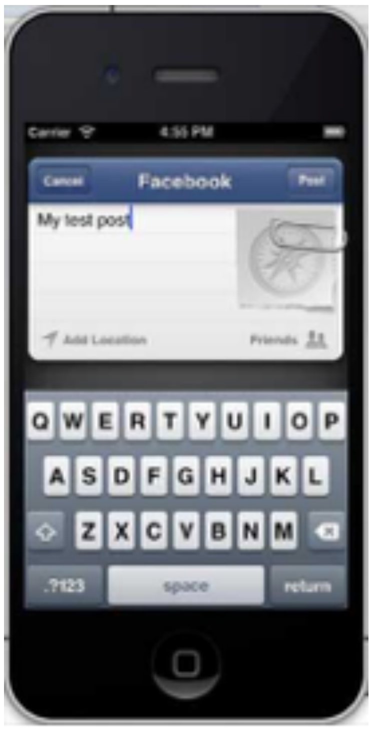
```
#import <Social/Social.h>

#import <Accounts/Accounts.h>

#import <UIKit/UIKit.h>


@interface ViewController : UIViewController
```

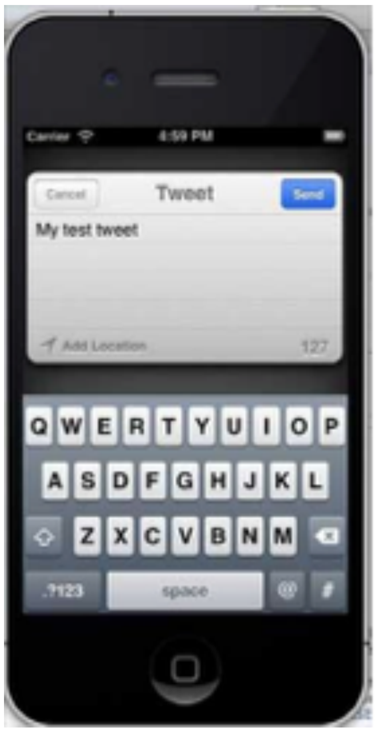
```
-(IBAction)twitterPost:(id)sender; -
    (IBAction)facebookPost:(id)sender;

@end
```

4. 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:SLServiceT
    ypeFacebook];
    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:SLServiceT
      ypeTwitter]; [tweetSheet setInitialText:@"My
      test tweet"];

[self presentModalViewController:tweetSheet
      animated:YES];

}

@end
```

Gamekit

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

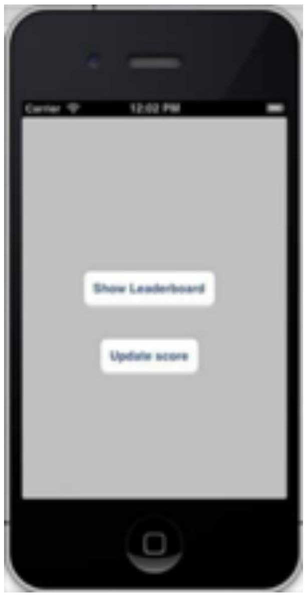
Steps Involved

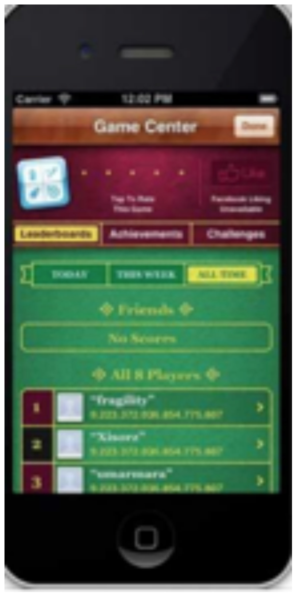
1. In iTunes connect ensure that you have a unique App ID and when we create the application update with

the bundle ID and code signing in Xcode with corresponding provisioning profile.

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

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.

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

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

6. Update the ViewController.xib as shown below.

7. Select your project file, then select targets and then add GameKit.framework. 8. Create IBActions for the buttons we have added.
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
```

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
```

Web Services

"Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network

Web services are two Types

1.Soap service

2.REST service

Soap Service:

SOAP originally defined as **Simple Object Access Protocol**, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format

REST Service:

REST defined as Representational State Transfer.REST attempts to describe architectures that use HTTP or similar protocols by constraining the interface to a set of well-known, standard operations (like GET, POST, PUT, DELETE for HTTP).

Web Service response:

Web Service response are two Types

1.XML Response

2.JSON response

XML Response:

XML:

XML was designed to transport and store data.

HTML was designed to display data.

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive

```
 <?xml version="1.0"?>
<parts-store>
  <part category="Auto">
    <PartID>314</PartID>
    <Name>Flux Capacitor</Name>
    <Description>Requires 1.21 Giga–Watts</
Description>
    <PhotoLink>"http://www.parts.com/parts/
314.jpg"<PhotoLink/>
    <Cost currency="USD">200.10</Cost>
  </part>
  <part category="Dirigible">
.... </part>

</parts-store>
```

JSON Response: JSON defined as **java Script Object Notation**.And It is More light weight than XML.And its looks like a Property list(plist)

JSON Response:

```
{
  "auto-part" : {
       "part-id" : 314,
       "name" : "Flux Capacitor",
       "description" : "Requires 1.21 Giga-Watts",
       "cost-usd" : 200.10,
       "manufacturers" : [ "DrBrown", "AlienInc" ];
   },
   "dirigible-part" : {
... }

}
```

Parsing Data:

1.XML Parsing

2.JSON Parsing

1.XML Parsing

Both SAX and DOM are used to parse the XML document. Both has advantages and disadvantages and can be used in our programming depending on the situation.

SAX (Event Based)(SIMPLE API FOR XML)

- Parses node by node
- Doesn't store the XML in memory
- We cant insert or delete a node
- SAX is an event based parser
- SAX is a Simple API for XML
- doesn't preserve comments
- SAX generally runs a little faster than DOM

eg:NSXMLParser

DOM (Tree Based)

- Stores the entire XML document into memory before processing
- Occupies more memory

- We can insert or delete nodes
- Traverse in any direction.
- DOM is a tree model parser
- Document Object Model (DOM) API
- Preserves comments
- SAX generally runs a little faster than DOM

NSXMLParser events sent to delegate

```
- (void)parser:(NSXMLParser *)parser didStartElement:
(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:
(NSString *)qName
        attributes:(NSDictionary *)attributeDict;
- (void)parser:(NSXMLParser *)parser didEndElement:
(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:
(NSString *)qName;
- (void)parser:(NSXMLParser *)parser foundCharacters:
(NSString *)string;
```

Real Time Examples for Web Services:

//url

http://www.dinetonite.com/api/dinetonite

Webservice Calling 1:

```
-(void)viewDidLoad
{
   //Rest
    //calling synchronous services

    //  http://www.dinetonite.com/api/dinetonite

    NSURL *url = [NSURL URLWithString:@"http://
www.dinetonite.com/api/dinetonite"];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
```

```
NSData *responseData = [NSURLConnection
sendSynchronousRequest:request returningResponse:nil
error:nil];

    //parsing
    NSXMLParser *parser = [[NSXMLParser alloc]
initWithData:responseData];
    [parser setDelegate:self];
    [parser parse];
 }



- (void)parser:(NSXMLParser *)parser
didStartElement:(NSString *)elementName
  namespaceURI:(NSString *)URI qualifiedName:
(NSString *)qName
    attributes:(NSDictionary *)attributeDict
{
   if([elementName isEqualToString:@"dinetonite"])
   {
       restArray = [[NSMutableArray alloc] init];
   }
```

```
    else if([elementName isEqualToString:@"Table"])
    {
        restDict = [[NSMutableDictionary alloc] init];
    }
    else
    {
        soapResults = [[NSMutableString alloc] init];
    }

}
- (void)parser:(NSXMLParser *)parser didEndElement:
(NSString *)elementName
  namespaceURI:(NSString *)URI qualifiedName:
(NSString *)qName
{
    if([elementName isEqualToString:@"dinetonite"])
    {
        //parsing ended
        NSLog(@"rest Array is %@",restArray);



    }
    else if([elementName isEqualToString:@"Table"])
    {
        [restArray addObject:restDict];
    }
    else
    {
        if(soapResults)
        [restDict setObject:soapResults
forKey:elementName];

        soapResults = nil;
    }

}
- (void)parser:(NSXMLParser *)parser
foundCharacters:(NSString *)string
{
    [soapResults appendString:string];

}

@end
```

WebService Calling 2:

```
//Calling REST Service with Arguments   and giving
XMl response
 -(void)viewDidLoad
{
   NSURL *url = [NSURL URLWithString:@"http://
www.dinetonite.com/api/dinetonite?
username=abc&pwd=123"];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
    [NSURLConnection connectionWithRequest:request
delegate:self];
 }
///Delegate Methods For NSURLConnection
- (void)connection:(NSURLConnection *)connection
didReceiveResponse:(NSURLResponse *)response
{
    NSLog(@"didReceiveResponse called");
     int length = [response expectedContentLength];

    responseData = [[NSMutableData alloc] init];
}
- (void)connection:(NSURLConnection *)connection
didReceiveData:(NSData *)data
{
    NSLog(@"didReceiveData called");
    [responseData appendData:data];
}
- (void)connectionDidFinishLoading:(NSURLConnection
*)connection
{
    NSLog(@"connectionDidFinishLoading called");
    //XML Parsing
    NSXMLParser *parser1 = [[NSXMLParser alloc]
initWithData:responseData];
    [parser1 setDelegate:self];
    [parser1 parse];
}
- (void)connection:(NSURLConnection *)connection
didFailWithError:(NSError *)error
{
    NSLog(@"didFailWithError called");
}
```

WebService Calling 3:

```
//Calling Rest Service POST Method and getting XML
response

-(void)viewDidLoad
{
  NSString *uname  = @"naveen";
  NSString *pwd = @"lmf";
  NSString *bodyString = [NSString
stringWithFormat:@"username=%@&password=%@",
[uName
stringByAddingPercentEscapesUsingEncoding:NSUTF8
StringEncoding], [pwd
stringByAddingPercentEscapesUsingEncoding:NSUTF8
StringEncoding]];

      NSMutableURLRequest* request =
[NSMutableURLRequest requestWithURL:[NSURL
URLWithString:"http://testweb.mywolla.com/
mobwebservices/smails_rest.php?"]];

      [request setHTTPMethod:@"POST"];
      [request setTimeoutInterval:30.0];


      [request setHTTPBody:[bodyString
dataUsingEncoding:NSUTF8StringEncoding]];
      NSLog(@"the request is %@",request);


      [NSURLConnection
connectionWithRequest:request delegate:self];
```

}

WebService Calling 4:

```
//Calling SOAP Service POST Method and getting XML
response


#define LOCAL_URL @"asdsdasd"

-(void)GetGroupsList:(NSString *)user_id
{
    NSString *soapFormat = [NSString
stringWithFormat:@"<?xml version=\"1.0\" encoding=
\"utf-8\"?>\n"
                              "<soap:Envelope xmlns:xsi=\"http://
www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=
\"http://www.w3.org/2001/XMLSchema\" xmlns:soap=
\"http://schemas.xmlsoap.org/soap/envelope/\">\n"
                              "<soap:Body>\n"
                              "<getGroupsList  xmlns=\"%@/\">"
                                 "<user_id>%@</user_id>"
                                 "<userName>%@</userName>"
                                               "<userNum>%@</
userNum>"
```

```
                        "</getGroupsList>"
                        "</soap:Body>\n"
                        "</soap:Envelope>
\n",LOCAL_URL,user_id];


   NSLog(@"The request format is %@",soapFormat);

   NSURL *locationOfWebService = [NSURL
URLWithString:[NSString
stringWithFormat:@"http://testweb.mywolla.com/
mobwebservices/sgroups.php"]];

   NSLog(@"web url = %@",locationOfWebService);

   NSMutableURLRequest *theRequest =
[[NSMutableURLRequest
alloc]initWithURL:locationOfWebService];

   NSString *msgLength = [NSString
stringWithFormat:@"%d",[soapFormat length]];


   [theRequest addValue:@"text/xml"
forHTTPHeaderField:@"Content-Type"];
```

```
    [theRequest addValue:[NSString
stringWithFormat:@"%@/sgroups.php",LOCAL_URL]
forHTTPHeaderField:@"SOAPAction"];

   [theRequest addValue:msgLength
forHTTPHeaderField:@"Content-Length"];

    [theRequest setHTTPMethod:@"POST"];
    //the below encoding is used to send data over the
net
    [theRequest setHTTPBody:[soapFormat
dataUsingEncoding:NSUTF8StringEncoding]];

    NSURLConnection *connect = [[NSURLConnection
alloc]initWithRequest:theRequest delegate:self];

}
```

WebService Calling 4:

```
//Calling REST service Service GET Method and
getting JSON response
-(void)viewDidLoad
{
    responseData = [[NSMutableData alloc] init];
    NSURLRequest *request = [NSURLRequest
requestWithURL:
                                        [NSURL
URLWithString:@"http://search.twitter.com/
search.json?q=iPhone&rpp=100"]];
    [[NSURLConnection alloc] initWithRequest:request
delegate:self];
}
///Delegate Methods For NSURLConnection
- (void)connection:(NSURLConnection *)connection
didReceiveResponse:(NSURLResponse *)response
{
    NSLog(@"didReceiveResponse called");
     int length = [response expectedContentLength];

    responseData = [[NSMutableData alloc] init];
}
- (void)connection:(NSURLConnection *)connection
didReceiveData:(NSData *)data
{
    NSLog(@"didReceiveData called");
    [responseData appendData:data];
}
- (void)connectionDidFinishLoading:(NSURLConnection
*)connection
```

```
{
 NSLog(@"connectionDidFinishLoading called");
    NSString *responseStr = [[NSString alloc] 
initWithBytes:[responseData mutableBytes] length:
[responseData length] 
encoding:NSUTF8StringEncoding];
    NSLog(@"The XML is %@",responseStr);

   //Getting JSON Response
    //Parsing the JSON Data
    NSError *error1;
     id parsedObj  = [NSJSONSerialization 
JSONObjectWithData:responseData 
options:NSJSONReadingAllowFragments 
error:&error1];
    if([parsedObj isKindOfClass:[NSArray class]])
    {
        //Array
    }
    else if([parsedObj isKindOfClass:[NSDictionary 
class]])
    {
        //Dict
    }
    else if([parsedObj isKindOfClass:[NSString class]])
    {
        //String
    }

}

}
```

UIAccelerometer

```
- (void)accelerometer:(UIAccelerometer
*)accelerometer didAccelerate:(UIAcceleration
*)acceleration
{
    NSLog(@"x = %0.0f y=%0.0f z = %0.0f",
   acceleration.x,
    acceleration.y,
          acceleration.z);
}

- (void)viewDidLoad
{
    UIAccelerometer *acceleroMeter =
[UIAccelerometer sharedAccelerometer];
    [acceleroMeter setDelegate:self];

}
```

Location Manager

```
//Import Core Location Frame work
#import <CoreLocation/CoreLocation.h>

- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
          fromLocation:(CLLocation *)oldLocation
{
    //newLocation.coordinate.latitude
    //newLocation.coordinate.longitude
    [manager stopUpdatingLocation];
}
- (void)locationManager:(CLLocationManager *)manager
     didUpdateLocations:(NSArray *)locations
{
}
- (void)viewDidLoad
{
    CLLocationManager *manager = [[CLLocationManager
alloc] init];
    [manager setDelegate:self];
    [manager startUpdatingLocation];
}
```

File Manager

Persisting and archiving requires writing data to a file, but an iPhone application can only
read and write to files in the application's sandbox. When installed, an application is placed
in its own home directory. This directory is the application's root directory and should be left untouched, lest you risk corrupting your application. Under the application's home directory are the directories you may write to. These directories are the Documents, Preferences, Caches, and tmp directories.

<application home directory>/Documents

. <application home directory>/Library/Preferences

. <application home directory>/Library/Caches

. <application home directory>/tmp

. The Documents directory is where you should write your application's data files. The Preferences directory is where your application's preferences are stored. These are the preferences set through the iPhone's AppSettings, using the NSUserDefaults class, not preferences you might create programmatically. The Caches directory, like the Documents directory, is another location you can persist files to, although, as the name implies, this directory should be reserved for caching data rather than storing an application's files. The tmp directory is a temporary directory for writing files that do not need persisting between application launches. Your application should remove files from this directory when not needed, but the iPhone operating system also removes files from this folder when an application is not running.

Files placed in the tmp folder are temporary and should be deleted when an application terminates

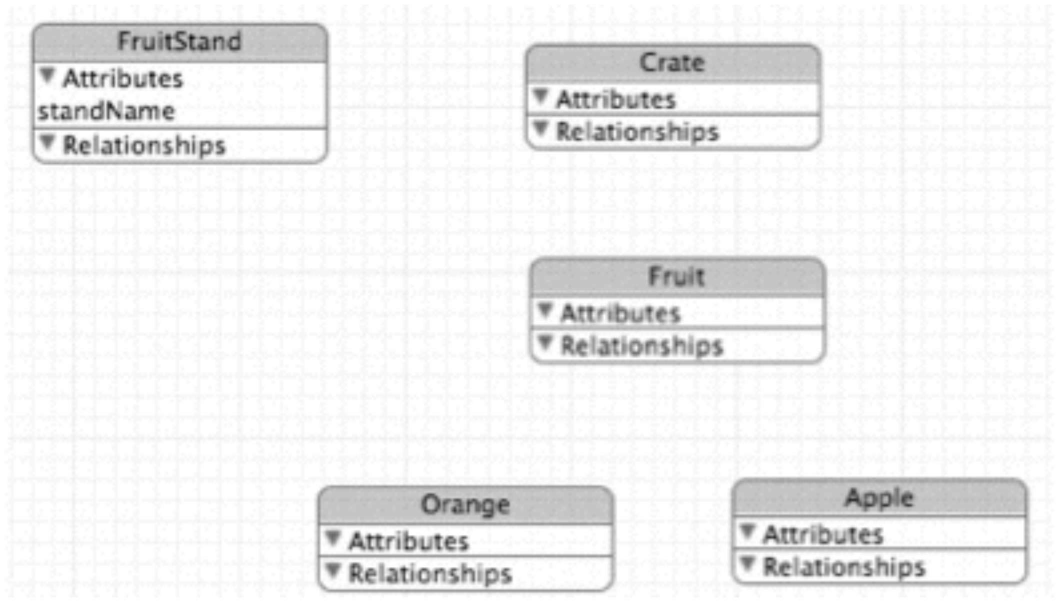
.

Data Storing in iPhone:

1.NSUserDefaults
2.using plist
3.Data base
4.Web services

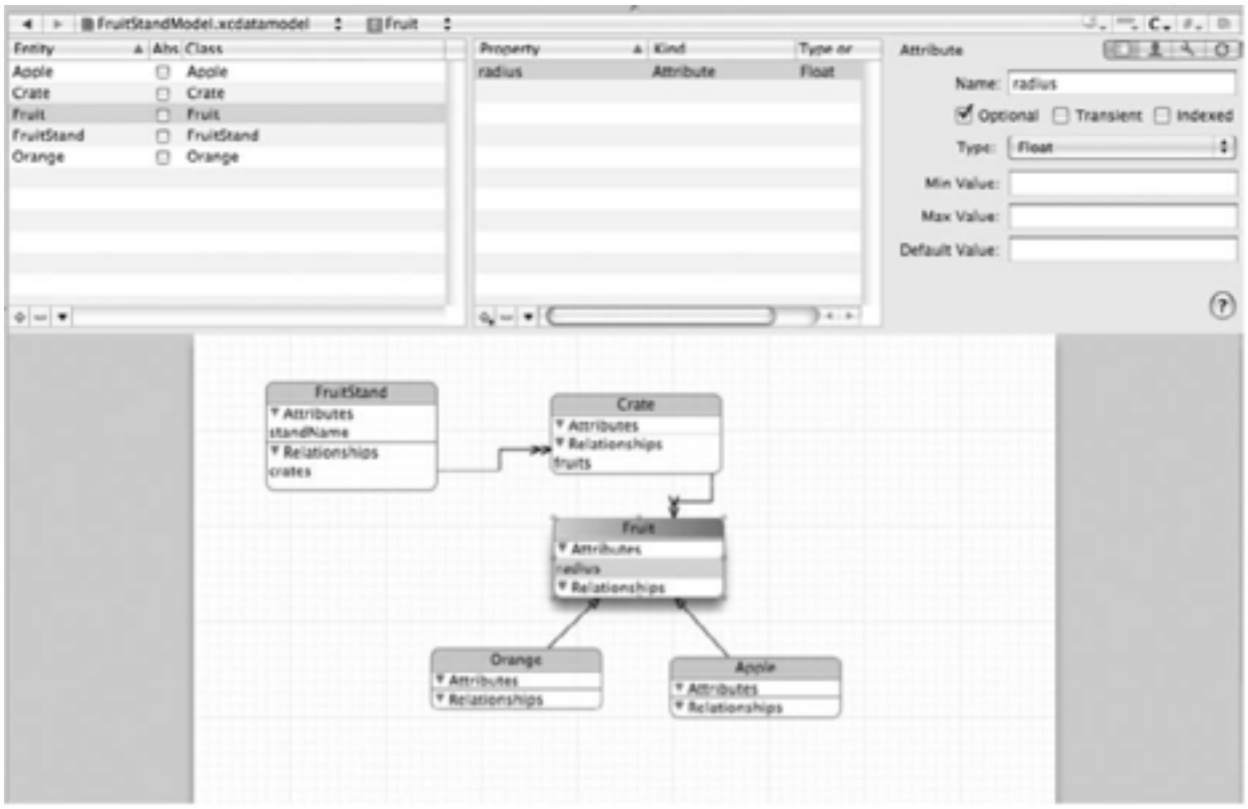
.m

```
- (void)viewDidLoad
{

    //Saving an Array into Document Directory path
    NSArray *aArray = [[NSArray alloc]
initWithObjects:@"Hi",@"Hello", nil ] ;
    [aArray writeToFile:[NSString
stringWithFormat:@"%@/aArray.plist",[self
getDocumentDirectoryPath]] atomically:YES];


    //getting the Array from Documents
    NSArray *bArray= [NSArray
arrayWithContentsOfFile:[NSString
stringWithFormat:@"%@/aArray.plist",[self
getDocumentDirectoryPath]]];
    NSLog(@"bArray is %@",bArray);
    //Saving an Image
    UIImage *img = [UIImage imageNamed:@"1.png"];
    NSData *imgData=UIImagePNGRepresentation(img);
    [imgData writeToFile:[NSString
stringWithFormat:@"%@/1.png",[self
getDocumentDirectoryPath]] atomically:YES];

    //getting image from File
    UIImage *finalImg=[UIImage
```



```
imageWithContentsOfFile:[NSString
stringWithFormat:@"%@/1.png",[self
getDocumentDirectoryPath]]];

   //File Manager
```

```
    NSFileManager *manager = [NSFileManager
defaultManager];

    //creating an Dirctory
    [manager createDirectoryAtPath:[NSString
stringWithFormat:@"%@/abc",[self
getDocumentDirectoryPath]]
withIntermediateDirectories:NO attributes:nil
error:nil];

    //Check File Existing or not
```



```
    NSString *path1 = [NSString 
stringWithFormat:@"%@/aArray.plist",[self 
getDocumentDirectoryPath]];
    if([manager fileExistsAtPath:path1])
    {
        NSLog(@"File Existing");
    }
    else
    {
        NSLog(@"aArray.plist is not found");
    }

    //Copy File
    [manager copyItemAtPath:[NSString 
stringWithFormat:@"%@/1.png",[self 
getDocumentDirectoryPath]] toPath:[NSString 
stringWithFormat:@"%@/abc/1.png",[self 
getDocumentDirectoryPath]] error:nil];

   //Move File
    [manager moveItemAtPath:[NSString 
stringWithFormat:@"%@/1.png",[self 
getDocumentDirectoryPath]] toPath:[NSString 
stringWithFormat:@"%@/abc/1.png",[self 
getDocumentDirectoryPath]] error:nil];
```

```
    //Delete File
    [manager removeItemAtPath:[NSString 
stringWithFormat:@"%@/abc",[self 
getDocumentDirectoryPath]] error:nil];
}

-(NSString *)getDocumentDirectoryPath
{
    NSString *documentsDirectory1  = 
[NSSearchPathForDirectoriesInDomains(NSDocument
Directory, NSUserDomainMask, YES) objectAtIndex:
0];

    NSLog(@"document directory path is 
%@",documentsDirectory1);
    return documentsDirectory1;
}
```

Serialization

To serialize a property list object, use the dataFromPropertyList:format:errorDescription: method.

```
        + (NSData *)dataFromPropertyList:(id)plist
format:
        (NSPropertyListFormat *)format
        errorDescription:(NSString **) errorString


NSMutableDictionary * dict2Serialize =
[[[NSMutableDictionary
     alloc] init] autorelease];
        NSString * name = @"James";
        NSArray * kids = [NSArray
arrayWithObjects:@"Nicolas",
@"Juliana", nil];
    NSNumber * age = [NSNumber numberWithInt:40];
    [dict2Serialize setObject:name forKey:@"name"];
  [dict2Serialize setObject:age forKey:@"age"];

    myData = [NSPropertyListSerialization
dataFromPropertyList:(id)
dict2Serialize
format:NSPropertyListXMLFormat_v1_0
errorDescription:&errorDescription];
    if(myData)
      [myData writeToFile:pathToFile atomically:YES];
    else {
      NSLog(@"Error writing to myData, error: %@",
errorDescription);
      [errorDescription release];
} }
```

Deserializing

To deserialize a property list, use the propertyListFromData:mutabilityOption:format:errorDescription: method.

```
         + (id)propertyListFromData:(NSData *)data
mutabilityOption:(
         NSPropertyListMutabilityOptions) opt format:
         (NSPropertyListFormat *)format
         errorDescription:(NSString **) errorString
```

Eg:

```
   NSPropertyListFormat format;
    NSData * plistData = [NSData
dataWithContentsOfFile:pathToFile];
```

```
    NSDictionary * props = (NSDictionary *)
[NSPropertyListSerialization
propertyListFromData:plistData
mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:&errorDescription];
    NSLog(@"name: %@", [props
objectForKey:@"name"]);
```

Archiving

You create an archive using an NSKeyedArchiver. This class persists any object that adopts the NSCoding protocol. You reconstitute an object by using NSKeyedArchiver's complement, the NSKeyedUnarchiver class. In this section, you learn how to create a class that adopts the NSCoding protocol. You then learn how to archive and unarchive this class.

Archiving a class requires that a class adopt the NSCoding protocol. The class should also adopt the NSCopying protocol when creating a class that adopts the NSCoding protocol.

NSKeyedArchiver

NSKeyedArchiver stores one or more objects to an archive using the initForWritingWith MutableData method. To be archived, an object must implement the NSCoding protocol.

```
NSKeyedArchiver * archiver = [[NSKeyedArchiver
alloc]
initForWritingWithMutableData:theData];
[archiver encodeObject:objectA forKey:@"a"];
[archiver encodeObject:objectB forKey:@"b"];
[archiver encodeObject:objectC forKey:@"c"];
[archiver finishEncoding];
```

After archiving, write the data object, which now contains the archived objects, to a file.

```
[theData writeToFile:"myfile.archive" atomically:YES]
```

NSKeyedUnarchiver

You use NSKeyedUnarchiver to unarchive an archive. NSKeyedUnarchiver reconstitutes one or more objects from a data object that was initialized with an archive. To be unarchived, an object must implement the NSCoding protocol. When programming for the iPhone, you use the initForReadingWithData: method.

```
    NSData * theData =[NSData
dataWithContentsOfFile:"myfile.archive"];

NSKeyedUnarchiver * uarchiver =
[[NSKeyedUnarchiver alloc]
    initForReadingWithData:theData];
```

After initializing the NSKeyedUnarchiver, unarchive the objects previously archived.

```
    A * objA = [[unarchiver decodeObjectForKey:@"a"]
retain];
    B * objB = [[unarchiver decodeObjectForKey:@"b"]
retain];
    C * objC = [[unarchiver decodeObjectForKey:@"c"]
retain];
    [unarchiver finishDecoding];
```

NSCoding

Classes that adopt this protocol must implement the encodeWithCoder: and initWithCoder: methods. The encodeWithCoder: method encodes the object and the object's instance variables so they can be archived.

```
- (void)encodeWithCoder:(NSCoder *)encoder
```

The initWithCoder: method decodes the object and the object's instance variables.

```
- (id)initWithCoder:(NSCoder *)decoder
```

NSCopying

When implementing the NSCoding protocol, best practices dictate that you also implement the NSCopying protocol. Classes that implement the NSCopying protocol must implement the copyWithZone method. Remember, when you set one object to another, you are merely creating another reference to the same underlying physical object. For instance, in the following code, both A and B are pointing to the same Foo that was originally allocated and initialized by A.

```
Foo * A = [[Foo alloc] init];
Foo * B = A;
```

When you copy an object, you obtain a distinct physical object, as if the object obtaining the copy actually allocated and initialized the object.

```
Foo * A = [[Foo alloc] init];
Foo * B = [A copy];
```

The method that allows copying is the copyWithZone: method.

- (id)copyWithZone:(NSZone *)zone
You can use either this method or NSObject's copy method to obtain what is called a "deep copy" of an object.

SQLite

SQLite database is a popular open-source database written in C. The database is small and designed for embedding in an application. Unlike a database such as Oracle, SQLite is a C library included when compiling a program. SQLite is part of the standard open-source Linux/BSD server stack, and as OS X is essentially FreeBSD, it was only natural Apple chose SQLite as the iPhone's embedded database.

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional

SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

Complete SQL database in an ordinary file

·No separate server process

·A complete database contained in a single file

• Simple, Fast, Compact code size ·Ideal for embedded devices

• Reliable –!"ACID" Transactions

·Atomicity –!transactions are all or none

·Consistency –!only valid data committed, invalid data is rolled back

·Isolation –!concurrent transactions never see data in an inconsistent state

·Durability –!transactions survive system failure (e.g. transaction log)

Appropriate Uses For SQLite

SQLite is different from most other SQL database engines in that its primary design goal is to be simple:

- Simple to administer
- Simple to operate
- Simple to embed in a larger program
- Simple to maintain and customize

Data base – Basic Structure

• **Table** –!The main element in a database – TABLE
• **Table Rows** –!represent data entities (or records) – ROW
• **Table Columns** –!represent attributes of an entity – COLUMN

SQLite C API Basics

Core Functions:

1.Open

```
        sqlite *sqlite_open(const char *dbname, int mode,
char **err);
```

2.Close

```
                          void sqlite_close(sqlite *db);
```

3 Execute Statement Or Query

```
    int sqlite_exec( ...params... );
```

in t sqlite3_prepare_v2(... params ...);

int sqlite3_step(...params...);

• Opaque data structure representing database connection

```
typedef struct sqlite3 sqlite3;
```

sqlite3 *db; // your database connection

Small list of constants used as return values:

```
#define SQLITE_OK          0
#define SQLITE_ERROR       1
#define SQLITE_ROW       100
#define SQLITE_DONE      101
```

Eg:

ViewController.m

```
#import "ViewController.h"
@implementation ViewController
```

```
// Implement viewDidLoad to do additional setup after
loading the view, typically from a nib.
- (void)viewDidLoad {
    sqlViewController = [[sqlClass alloc]init];
    [sqlViewController copyDatabaseIfNeeded];
    [sqlViewController gettingData:[sqlViewController
getDBPath]];
    NSLog(@"Member Data is
%@",sqlViewController.membersInfoArray);
}
-(IBAction)addRecord
{
    //prepare dictionary
    NSMutableDictionary *aDict =
[[NSMutableDictionary
alloc]initWithObjectsAndKeys:@"2",@"id",@"phani",@"
name",@"MCA",@"degree",@"9985047204",@"phone",
@"hyd",@"location",@"madhu7204@gmail.com",@"mail
Id",@"9:00-10:00",@"timings",@"2010",@"passedOut",
@"22-11-2010",@"date",nil];

    [sqlViewController addRecord:aDict];
    [sqlClass finalizeStatements];
}
-(IBAction)deleteRecord
```

```
{
	[sqlViewController deleteRecord:3];
}
```

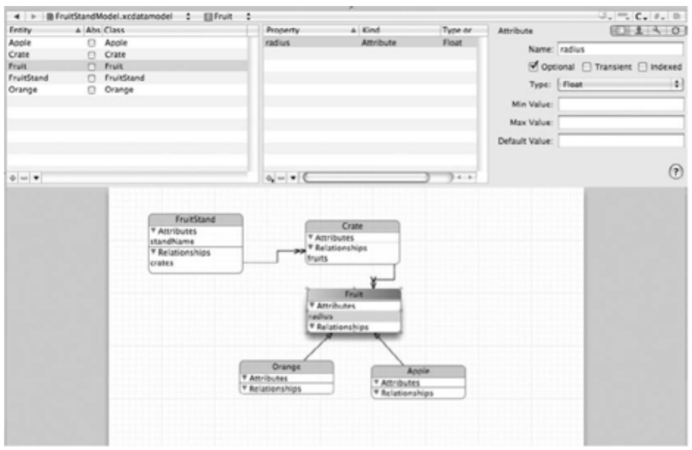
sqlClass.h

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

@interface sqlClass : NSObject {
    NSMutableArray *membersInfoArray;
    NSMutableArray *membersInfoDict;
    NSInteger rowID;
}
@property(nonatomic,retain) NSMutableArray
*membersInfoArray;
@property(nonatomic,retain) NSMutableArray
*membersInfoDict;
@property (nonatomic, readonly) NSInteger rowID;

- (NSString *) getDBPath;
- (void) gettingData:(NSString *)dbPath;
- (void) copyDatabaseIfNeeded ;
```

```
@end


sqlClass.m

#import "sqlClass.h"
static sqlite3 *database = nil;
static sqlite3_stmt *deleteStmt = nil;
static sqlite3_stmt *addStmt = nil;
static sqlite3_stmt *detailStmt = nil;
static sqlite3_stmt *updateStmt = nil;


@implementation sqlClass
@synthesize
membersInfoArray,membersInfoDict,rowID;


- (void) copyDatabaseIfNeeded
{
    membersInfoArray = [[NSMutableArray alloc]init];
    membersInfoDict = [[NSMutableDictionary
alloc]init];
```

```
    //Using NSFileManager we can perform many file
system operations.
    NSFileManager *fileManager = [NSFileManager
defaultManager];
    NSError *error;
    NSString *dbPath = [self getDBPath];
    BOOL success = [fileManager
fileExistsAtPath:dbPath];

    if(!success) {

        NSString *defaultDBPath = [[[NSBundle
mainBundle] resourcePath]
stringByAppendingPathComponent:@"SQL.sqlite"];
        success = [fileManager
copyItemAtPath:defaultDBPath toPath:dbPath
error:&error];

        if (!success)
            NSAssert1(0, @"Failed to create writable
database file with message '%@'.", [error
localizedDescription]);
    }
}
```

```
- (NSString *) getDBPath {

    NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentD
irectory , NSUserDomainMask, YES);
    NSString *documentsDir = [paths objectAtIndex:0];
    return [documentsDir
stringByAppendingPathComponent:@"SQL.sqlite"];
}
+ (void) finalizeStatements {

    if (database) sqlite3_close(database);
    if (deleteStmt) sqlite3_finalize(deleteStmt);
    if (addStmt) sqlite3_finalize(addStmt);
    if (detailStmt) sqlite3_finalize(detailStmt);
    if (updateStmt) sqlite3_finalize(updateStmt);
}

- (void) gettingData:(NSString *)dbPath {
```



Entity	Abs	Class
Apple	☐	Apple
Crate	☐	Crate
Fruit	☐	Fruit
FruitStand	☐	FruitStand
Orange	☐	Orange

Property	Kind	Type or
radius	Attribute	Float

Attribute
Name: radius
☑ Optional ☐ Transient ☐ Indexed
Type: Float
Min Value:
Max Value:
Default Value:

```
 NSLog(@"Data base path
is %@",dbPath);
 if (sqlite3_open([dbPath
UTF8String], &database)
== SQLITE_OK)
```

```
    {
      const char *sql = "select * from rapid";
      sqlite3_stmt *selectstmt;
      if(sqlite3_prepare_v2(database, sql, -1,
&selectstmt, NULL) == SQLITE_OK)
      {
            while(sqlite3_step(selectstmt) ==
SQLITE_ROW)
            {
                  [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 0)] forKey:@"id"];
                  [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 1)]
forKey:@"name"];
                  [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 2)]
forKey:@"degree"];
                  [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 3)]
forKey:@"phone"];
```

```
            [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 4)]
forKey:@"location"];
            [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 5)]
forKey:@"mailId"];
            [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 6)]
forKey:@"timings"];
            [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 7)]
forKey:@"passedOut"];
            [membersInfoDict setValue:[NSString
stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 8)]
forKey:@"date"];
            if(membersInfoDict)
            {
              [membersInfoArray
addObject:membersInfoDict];
              membersInfoDict = nil;
```

```
                    NSLog(@"Entered and return");
                    return;
                }
            }
        }
    }
    else
        sqlite3_close(database); //Even though the open
call failed, close the database connection to release all
the memory.
}


- (void) deleteRecord:(NSInteger )recordId
{

    if(deleteStmt == nil) {
        const char *sql = "delete from Coffee where
coffeeID = ?";
        if(sqlite3_prepare_v2(database, sql, -1,
&deleteStmt, NULL) != SQLITE_OK)
            NSAssert1(0, @"Error while creating delete
statement. '%s'", sqlite3_errmsg(database));
    }
    //When binding parameters, index starts from 1 and
not zero.
    sqlite3_bind_int(deleteStmt, 1, recordId);

    if (SQLITE_DONE != sqlite3_step(deleteStmt))
      NSAssert1(0, @"Error while deleting. '%s'",
sqlite3_errmsg(database));

    sqlite3_reset(deleteStmt);
}

- (void) addRecord:(NSMutableDictionary *)recordDict
{

    if(addStmt == nil) {
      const char *sql = "insert into rapid(id  , name ,
degree , phone ,
Location ,mailId ,timings ,passedOut ,date)
Values(?, ?,?, ?,?, ?,?, ?,?)";
      if(sqlite3_prepare_v2(database, sql, -1, &addStmt,
NULL) != SQLITE_OK)
            NSAssert1(0, @"Error while creating add
statement. '%s'", sqlite3_errmsg(database));
    }
    NSInteger recordId =  [[recordDict
objectForKey:@"id"] intValue];
    NSLog(@"%d",recordId);
    sqlite3_bind_int(addStmt, 1,recordId);
    sqlite3_bind_text(addStmt, 2, [[recordDict
objectForKey:@"name"] UTF8String], -1,
SQLITE_TRANSIENT);
    sqlite3_bind_text(addStmt, 3, [[recordDict
objectForKey:@"degree"] UTF8String], -1,
SQLITE_TRANSIENT);
    sqlite3_bind_text(addStmt, 4, [[recordDict
objectForKey:@"phone"] UTF8String], -1,
SQLITE_TRANSIENT);
    sqlite3_bind_text(addStmt, 5, [[recordDict
objectForKey:@"Location"] UTF8String], -1,
SQLITE_TRANSIENT);
    sqlite3_bind_text(addStmt, 6, [[recordDict
objectForKey:@"mailId"] UTF8String], -1,
SQLITE_TRANSIENT);
    sqlite3_bind_text(addStmt, 7, [[recordDict
objectForKey:@"timings"] UTF8String], -1,
SQLITE_TRANSIENT);
    sqlite3_bind_text(addStmt, 8, [[recordDict
objectForKey:@"passedOut"] UTF8String], -1,
SQLITE_TRANSIENT);
```

```
    sqlite3_bind_text(addStmt, 9, [[recordDict
objectForKey:@"date"] UTF8String], -1,
SQLITE_TRANSIENT);


    if(SQLITE_DONE != sqlite3_step(addStmt))
      NSAssert1(0, @"Error while inserting data. '%s'",
sqlite3_errmsg(database));
    else
      //SQLite provides a method to get the last
primary key inserted by using
sqlite3_last_insert_rowid
      rowID = sqlite3_last_insert_rowid(database);
    NSLog(@"last inserted rowId = %d",rowID);

    //Reset the add statement.
    sqlite3_reset(addStmt);
}

- (void) getRecord:(NSInteger )recordId
{

    if(detailStmt == nil) {
      const char *sql = "select *from rapid where id = ?";
      if(sqlite3_prepare_v2(database, sql, -1,
&detailStmt, NULL) != SQLITE_OK)
            NSAssert1(0, @"Error while creating detail
view statement. '%s'", sqlite3_errmsg(database));
    }

    sqlite3_bind_int(detailStmt, 1, recordId);

    if(SQLITE_DONE != sqlite3_step(detailStmt))
    {



    }
    else
      NSAssert1(0, @"Error while getting the price of
coffee. '%s'", sqlite3_errmsg(database));

    //Reset the detail statement.
    sqlite3_reset(detailStmt);

    //Set isDetailViewHydrated as YES, so we do not
get it again from the database.

}
```

```
@end
```

Core Data

In iPhone OS X 3.0 SDK, Core Data is arguably the best choice for persisting an application's data. It is more robust than using properties and is easier than using SQLite. You can visually lay out your application's data model, much as you would
when using a database-modeling tool. Moreover, it provides the infrastructure for managing the objects you create, freeing you from writing the typical object management code. As this chapter will demonstrate, using Core Data is a natural choice for persisting an

application's data. Core Data allows you to focus on the application rather than on the code persisting the application's data.

Core Data is a framework used to easily manage an application's data objects. Core Data consists of managed object models, managed object contexts, and persistent data stores.
A managed object model contains an object graph. The object graph is a collection of objects and their relationships with one another. You create a graph using Xcode's data modeler. The data modeler is where you visually add entities and create relationships between them.
A managed object context contains the objects created from the entities in the managed object model. A managed object context has a persistent store coordinator that manages one
or more persistent stores. A persistent data store persists objects created by the managed object context. Although an application can have multiple persistent stores,

Creating a Model

A model contains entities. Entities contain attributes. Relationships model how one or more entities relate to one another. You model these concepts using Xcode's data modeler. You add a model to your application by creating a file with an .xcdatamodel extension

- After creating the model, when you select the file, Xcode should automatically display the data modeler in the Editor window

Entities

Entities, represented by the NSEntityDescription class, are patterns describing NSManagedObject instances. NSManagedObjects are what you persist. For Core Data to know how to instantiate new object instances and persist the instances, it needs a pattern. Entities in the model provide those patterns.Entities can have attributes and Consider an attribute as an object's property

Relationships

Entities can have relationships with other entities. For instance, an apple and an orange are both types of fruit, a crate might hold one or more apples and oranges, and a fruit stand might contain one or more crates.

The Apple and Orange entities are subentities of Fruit, or Fruit is the parent of Apple and Orange. This relationship between Fruit and its two subentities is inheritance. Inheritance between entities is exactly like inheritance between classes. The child entities, Apple and

Orange, inherit their parent's attributes and relationships. You can also make a parent abstract by selecting the Abstract check box next to the parent's name in the data modeler

A Crate aggregates zero or more Fruit. A crate does not care if the fruit is an apple or an orange, and so it merely aggregates fruit. A FruitStand aggregates zero or more Crate entities. The following task illustrates adding these relationships to the data model.

NSManagedObjectModel

As discussed earlier, an application's managed object model contains entities and their relationships. It serves as an application's schema by describing the entities used in an application's managed object context. The easiest way to obtain the model is through the mergedModelFromBundles: class method.

```
managedObjectModel = [[NSManagedObjectModel
mergedModelFromBundles:
nil] retain];
```

This method creates a data model by merging all the models it finds into a bundle. Because the previous code specifies nil, the method simply finds all the models in the application's Resources folder and merges them into one NSManagedObjectModel instance.

NSPersistentStoreCoordinator

A persistent store coordinator coordinates one or more persistent stores and associates them with a managed

object model. Although advanced applications might have more than one persistent store, this chapter limits consideration to single store applications. Core Data
can persist data in several different ways. Store types you might use to persist data include NSSQLiteStoreType, NSBinaryStoreType, or NSInMemoryStoreType. Consider the following code snippet. It defines a store by loading a SQLite database from an application's Documents directory.

```
NSURL *storeUrl = [NSURL fileURLWithPath: [[self
applicationDocuments
Directory]
stringByAppendingPathComponent:
@"CoreTemplateApp.sqlite"]];
```

After obtaining the URL to the store, you create an NSPersistentStoreCoordinator instance. The coordinator initializes the store coordinator using a managed object model instance. The persistent store contains the data, while the model defines how to interpret that data.

```
persistentStoreCoordinator =
[[NSPersistentStoreCoordinator alloc]
initWithManagedObjectModel: [self
managedObjectModel]];
```

NSManagedObjectContext

The NSManagedObjectContext represents an application's managed object context. This is where you add entity instances (NSManagedObject classes) to your application. It is where you fetch managed objects to and from the persistent store, and it is where you modify the objects. =Apple describes the context as a big "scratch-pad" because no manipulations to a context are persisted until code explicitly tells the context to persist the changes.

You obtain an application's managed context by allocating and initializing a new NSManagedObjectContext instance. You then set its persistent store coordinator.

```
managedObjectContext = [[NSManagedObjectContext
alloc] init];
[managedObjectContext
setPersistentStoreCoordinator: coordinator];
```

NSManagedObject

The NSManagedObjectContext manages NSManagedObject instances. NSManagedObjects are not entities, but rather, created from entities. An application obtains data from the persistent store and uses the entities in the model to create the NSManagedObjects placed in the context. Consider NSEntityDescriptions as the classes and NSManagedObjects as the objects.

The previous Try This tasks created entities in the xcdatamodel file. Although the NSManagedObjectModel uses these entities, the NSManagedObjectContext does not; it manages NSManagedObjects. Before you can use the entities in an application, you must generate classes that extend the NSManagedObject class. After creating the NSManagedObjects, you can then work with them programmatically. The following task illustrates creating NSManagedObjects from entities.

Adding Objects

The easiest way to create a new managed object is through the NSEntityDescription's class method, insertNewObjectForEntityForName:inManagedObject Context.

```
   + (id)insertNewObjectForEntityForName:(NSString
*) entityName
   inManagedObjectContext:
(NSManagedObjectContext *) context

Apple * apple = (Apple *) [NSEntityDescription
insertNewObjectForEntity
   ForName: @"Apple"
inManagedObjectContext:self.managedObjectContext]
;
```

The context saves changes using its save method. This method persists the context's changes to its associated persistent data store. The method takes an error as a parameter and returns a Boolean indicating success or failure.

```
   - (BOOL)save:(NSError **) error
```

Note :

An NSManagedObjectContext can have an NSUndoManager instance assigned to its undoManager property. An NSUndoManager manages undoing actions. When using Core Data, you can use this class to undo changes made to an application's NSManagedModelContext. For more information, refer to the NSUndoManager Class Reference.

Fetching Data

Managed objects remain in the persistent data store until needed. You request objects from the persistent data store using NSFetchRequests. The NSFetchRequest wraps search criteria for retrieving data from the store. A fetch request consists of an NSEntityDescription, an optional NSPredicate, and an optional NSSortDescriptor.

```
NSFetchRequest * myRequest = [[NSFetchRequest
alloc] init];
  NSEntityDescription * entDesc =
[NSEntityDescription
  entityForName:@"Orange"
inManagedObjectContext:myContext];
  [myRequest setEntity:entDesc];
  NSError * error;
```

```
   NSArray * fetchResults =
[self.managedObjectContext
   executeFetchRequest:myRequest
   error:&error];
   if(fetchResults == nil) {
     NSLog(@"an error occurred");
     [error release];
   }
```

Sample Predicates:

```
NSPredicate * predicate = [NSPredicate
predicateWithFormat: @"radius > %@",
[NSNumber numberWithFloat:(float)2.1]];

NSPredicate* objectivesPredicate = [NSPredicate
predicateWithFormat:@"start_date == %@ &&
objective_type != %@ && objective_id",
self.objStartDate,@"Account
Objective",self.objectiveId];


//single = for int and == for String
```

```
[NSPredicate predicateWithFormat:@"objective_id=
%@",objective.objective_id]


NSPredicate *predicate=[NSPredicate
predicateWithFormat:@"start_date_time >= %@ AND
start_date_time <= %@ AND (visit == nil OR
visit.time_out == nil)", startDate, endDate];


 [NSPredicate predicateWithFormat:@"account_id ==
%@ && order_type == 'Wet
Stock'",self.activeAccount.account_id];


  NSPredicate *inPredicate = [NSPredicate
predicateWithFormat: @"trasaction_id IN %@ AND
status = %@", arrAllDoodlesList,UNREAD];
```

Deleting Entities

You use the NSManagedObjectContext's deleteObject method to delete objects from an application's managed object context. This method takes an NSManagedObject instance of the object to delete. You might pass an Orange instance to the method.

    [myContext deleteObject: (NSManagedObject *) myOrange];
The managed object context is smart enough to mark the particular orange for deletion; remember, the context does not actually delete the object until code calls the context's save method.

Eg Project on Core Data:

Employee.h

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

#import <CoreData/CoreData.h>



@interface Employee : NSManagedObject


@property (nonatomic, retain) NSString * empID;

@property (nonatomic, retain) NSString *name;

@property (nonatomic, retain) NSString * desc;
```

```
@end
```

Employee.m

```
#import "Employee1.h"

@implementation Employee1

@dynamic empID;

@dynamic name;

@dynamic desc;

@end
```

ViewController.m

```
- (void)insertToDB:(NSDictionary*)resultDict

{


    NSManagedObjectContext *context =
appDelegate.managedObjectContext;

    NSError *error;
    Employee *empInfo = [NSEntityDescription

insertNewObjectForEntityForName:@"Employee"

inManagedObjectContext:context];
    empInfo.empID = [NSNumber numberWithInt:
[[resultDict objectForKey:@"empID"] intValue]] ;
    empInfo.name = [resultDict
objectForKey:@"name"];;
    empInfo.designation = [resultDict
objectForKey:@"designation"];



    if (![context save:&error]) {
        NSLog(@"Couldn't save: %@", [error
localizedDescription]);
    } else {
        NSLog(@"content saved successfully");
        emp = empInfo;
    }
}


- (void) queryFromDB
{
      NSError *error;
      NSManagedObjectContext *context = 
appDelegate.managedObjectContext;
      // Test listing all FailedBankInfos from the store
    NSFetchRequest *fetchRequest = 
[[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription 
entityForName:@"Employee"

inManagedObjectContext:context];
    [fetchRequest setEntity:entity];
    resultsArray = [context 
executeFetchRequest:fetchRequest error:&error];
```

```
    NSLog(@"fetchedObjects %@",resultsArray);


    for (int i = 0; i<[resultsArray count]; i++) {

        Employee *empInfo = [resultsArray
objectAtIndex:i];


        NSLog(@"Emp ID  :%@  empName: %@,
empDesignation %@",empInfo.empID, empInfo.name,
empInfo.designation);

    }

}
```

AppDelegate.m

```
#pragma mark - Core Data stack

// Returns the managed object context for the
application.

// If the context doesn't already exist, it is created
and bound to the persistent store coordinator for the
application.
```

```
- (NSManagedObjectContext *)managedObjectContext
{
    if (__managedObjectContext != nil) {
        return __managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self
persistentStoreCoordinator];
    if (coordinator != nil) {
        __managedObjectContext =
[[NSManagedObjectContext alloc] init];
        [__managedObjectContext
setPersistentStoreCoordinator:coordinator];
    }
    return __managedObjectContext;
}

// Returns the managed object model for the
application.
```

```
// If the model doesn't already exist, it is created
from the application's model.

- (NSManagedObjectModel *)managedObjectModel

{

    if (__managedObjectModel != nil) {

        return __managedObjectModel;

    }

    NSURL *modelURL = [[NSBundle mainBundle]
URLForResource:@"Employee"
withExtension:@"momd"];

    __managedObjectModel =
[[NSManagedObjectModel alloc]
initWithContentsOfURL:modelURL];

    return __managedObjectModel;

}


// Returns the persistent store coordinator for the
application.

// If the coordinator doesn't already exist, it is
created and the application's store added to it.
```

```
- (NSPersistentStoreCoordinator
*)persistentStoreCoordinator

{

    if (__persistentStoreCoordinator != nil) {

        return __persistentStoreCoordinator;

    }



    NSURL *storeURL = [[self
applicationDocumentsDirectory]
URLByAppendingPathComponent:@"Student.sqlite"];



    NSError *error = nil;

    __persistentStoreCoordinator =
[[NSPersistentStoreCoordinator alloc]
initWithManagedObjectModel:[self
managedObjectModel]];

    if (![__persistentStoreCoordinator
addPersistentStoreWithType:NSSQLiteStoreType
configuration:nil URL:storeURL options:nil
error:&error]) {
        NSLog(@"Unresolved error %@, %@", error,
[error userInfo]);
        abort();
    }
    return __persistentStoreCoordinator;
}
```

App Sore Submission and Device Execution and .ipa File Creation

1.Setting up for the device Execution
2.ipa file building
3.App Store submission

1.Setting up for the device execution:

a.CSR generation
b.uploading CSR to developers portal and download the Developers profile
c.Adding Devices
d.Creating App Id
e.Generate Provisioning Profile for Debugging the App
f.Generate Distribution for the .ipa File Generation
g.Generate App Store Profile for the App Store Submission

Reference: http://www.youtube.com/watch?v=HlRI3OF6-Ek

a.CSR generation:

key chain Access App --> Certificate Assistent--> Request a Certificate from certificate Authority --> save to disk option and give the Email Address --> Continue --> Done

b.uploading CSR to developers portal and download the Developers profile

goto www.developer.apple.com--->iOS dev center -->login -->iOS Provisioning portal--> Certificates --> Request Certificate --> choose file--> Upload ur CSR File which is create from key chain in ur desktop.

And Click on the refresh button and Download the Developer identity.

And also download the WWDR Certificate

Click on the Distribution Tab -->Request Certificate --> choose file--> Upload ur CSR File which is create from key chain in ur desktop.

And Click on the refresh button and Download the Distribution identity.

And also download the WWDR Certificate

c.Adding Devices

//Getting UDID

1.installing udid app in Device
2.Connect to iTunes and go to summary tab
3.goto organizer in XCODE and select Device tabs

goto www.developer.apple.com--->iOS dev center -->login -->iOS Provisioning portal-->Devices--> Add Device--> give Device name name and UDID
Note : You can get the udid from the UDID App in the App Store or else connect your device and open itunes--> Summary --> copy udid

d.Creating App Id

goto www.developer.apple.com--->iOS dev center -->login -->iOS Provisioning portal-->App Id--> Create new App Id --> give app name and identifier like "com.rapid.abc" (abc is the application name and rapid is the company name)

Note : give (com.*)

Ref: http://www.youtube.com/watch?v=ni-9jZPcZ1E&list=PL277253281B92215C

e.Generate Provisioning Profile for Debugging the App

1.Provisioning Profile	—> Used for Debugging
2.Distribution Profile	—> To Distribute the App
3.App Store Profile	—> To Send to App Store

goto www.developer.apple.com--->iOS dev center -->login -->iOS Provisioning portal-->Provisioning Profile--> Development --> Add new Profile --> Give profile name and Selct the App id and select the Devices and Submit--> and then click on download button.

After download the profile double click on the profile and then it will goto XCODe and it should give green colour tick mark.

Ref:http://www.youtube.com/watch?v=7Rypv7RcMIY&list=PL277253281B92215C

g.Generate App Store Profile for the App Store Submission

goto www.developer.apple.com--->iOS dev center -->login -->iOS Provisioning portal-->Provisioning Profile-->Distibution-->Add new Profile -->Select App Store or ADhoc Distribution -->Give profile name and Select the App id and select the Devices and Submit--> and then click on download button.

After download the profile double click on the profile and then it will goto XCODe and it should give green colour tick mark.

2.ipa file building

Open XCODe Project and the goto Build Settings --> Code Signing --> Select the Latest Enabled profiles of hrs in all 4 sections --> and then Click on Product button in the Status Bar opn MAC--> Clean

And again Product--> Archieve --> Select the Distribute button--> Distribution ---> save on ur MAC.

3.App Store submission

1. Login to https://**itunesconnect**.apple.com/
2. Enter the login credentials.
3. Click on Manage your applications

4. Add New App --> Give the App Name --> Select the build identifier --> and click on next.
5.Give the Description,keywords,Select the Category and subcategory,Screen shots(640*960 and 640*1136),icon files(1024*1024),Give test id and pwd if ur app having the login screen.--> submit
6.And Select your App--> view details --> click on upload binary button (Your app colour changed to yellow)

7.Open XCODE Project select the App Store profile and Deployment target
8.Click On Product button on the top --> Clean and Again Product button click and Archive
9.Select the Distribute button--> App Store Distribution --> give id and pwd --> done.

Ref: http://www.youtube.com/watch?v=dbabzWVvcvE

Status Bar

```
//Hiding Status Bar
[[UIApplication sharedApplication]
setStatusBarHidden:YES];
```

Add a custom method hideStatusbar to our class

```
-(void)hideStatusbar{
[[UIApplication sharedApplication]
setStatusBarHidden:YES
withAnimation:UIStatusBarAnimationFade];
[UIView beginAnimations:@"Statusbar hide"
context:nil]; [UIView setAnimationDuration:0.5];
[self.view setFrame:CGRectMake(0, 0, 320, 480)];
[UIView commitAnimations]; }



- (void)viewDidLoad {
[super viewDidLoad];
// The method hideStatusbar called after 2 seconds
[self performSelector:@selector(hideStatusbar)
withObject:nil afterDelay:2.0];
```

// Do any additional setup after loading the view, typically from a nib.
}

UISplitViewController

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
// Override point for customization after application launch.
MasterViewController *masterViewController = [[MasterViewController alloc] initWithNibName:@"MasterViewController" bundle:nil];
UINavigationController *masterNavigationController = [[UINavigationController alloc] initWithRootViewController: masterViewController];
DetailViewController *detailViewController = [[DetailViewController alloc] initWithNibName:@"DetailViewController" bundle:nil];
UINavigationController *detailNavigationController = [[UINavigationController alloc] initWithRootViewController: detailViewController];

```
masterViewController.detailViewController =
detailViewController;
self.splitViewController = [[UISplitViewController
alloc] init]; self.splitViewController.delegate =
detailViewController;
self.splitViewController.viewControllers =
@[masterNavigationController,
detailNavigationController];
self.window.rootViewController =
self.splitViewController;
```

Search Bar

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

@interface b13SearchBarViewController :
UIViewController
{
    NSMutableArray
*itemsArray,*searchArray,*tableArray;
    BOOL isSearching;
    IBOutlet UITableView *table1;
}
```

```
@end



.m
#import "b13SearchBarViewController.h"

@implementation b13SearchBarViewController


- (void)viewDidLoad
{
   NSDictionary *aDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"iPhone",@"name",nil
];
     NSDictionary *bDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"iPad",@"name",nil];
     NSDictionary *cDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"iPod
Touch",@"name",nil];
     NSDictionary *dDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"iMAC",@"name",nil];
     NSDictionary *eDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"MAC
Book",@"name",nil];
```

```
       NSDictionary *iDict = [NSDictionary 
dictionaryWithObjectsAndKeys:@"MAC Book 
Pro",@"name",nil];
      NSDictionary *fDict = [NSDictionary 
dictionaryWithObjectsAndKeys:@"MAC Book 
Air",@"name",nil];
      NSDictionary *gDict = [NSDictionary 
dictionaryWithObjectsAndKeys:@"MAC 
Mini",@"name",nil];
      NSDictionary *hDict = [NSDictionary 
dictionaryWithObjectsAndKeys:@"ipod 
nano",@"name",nil];

    itemsArray = [[NSMutableArray 
arrayWithObjects:aDict,bDict,cDict,eDict,fDict,gDict,
hDict,iDict,nil] retain];

    tableArray = itemsArray;
    [table1 reloadData];

    [super viewDidLoad];
}
```

```
#pragma Search Bar
- (void)searchBarTextDidBeginEditing:(UISearchBar
*)searchBar
{
    searchBar.showsCancelButton = YES;
}
- (void)searchBarSearchButtonClicked:(UISearchBar
*)searchBar
{
    searchBar.showsCancelButton = NO;
    [searchBar resignFirstResponder];
}
- (void)searchBarTextDidEndEditing:(UISearchBar
*)searchBar
{

}
- (void)searchBar:(UISearchBar *)searchBar
textDidChange:(NSString *)searchText
{
    NSLog(@"textDidChange called serch text is
%@",searchText);
    if([searchText isEqualToString:@""]||
searchText==nil)
```

```
    {
        tableArray = itemsArray;
        isSearching = NO;
        [table1 reloadData];
        [searchBar resignFirstResponder];
        return;
    }
   // NSString *firstCapChar = [[searchText 
substringToIndex:1] capitalizedString];
   // searchText= [searchText 
stringByReplacingCharactersInRange:NSMakeRange(0,1
) withString:firstCapChar];

    isSearching = YES;
    NSInteger counter = 0;
    searchArray = [itemsArray 
filteredArrayUsingPredicate:
                      [NSPredicate 
predicateWithFormat:@"(name beginswith[cd] %@)", 
searchText]];



    tableArray = [[NSMutableArray 
arrayWithArray:searchArray] retain];
    [table1 reloadData];
```

```
}

- (void)searchBarCancelButtonClicked:(UISearchBar *)
searchBar
{
    searchBar.showsCancelButton = NO;
    isSearching = NO;
    tableArray = itemsArray;
    [table1 reloadData];
    [searchBar resignFirstResponder];
}



#pragma UITableViewDataSource

- (NSInteger)numberOfSectionsInTableView:
(UITableView *)tableView
{
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return [tableArray count];
```

```
}

- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 70;
}
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
{
    return @"Items";
}
- (UITableViewCell *)tableView:(UITableView
*)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:nil] autorelease];
    }
    //Customise cell
    cell.textLabel.text = [[tableArray
objectAtIndex:indexPath.row]
objectForKey:@"name"];

    return cell;
}
```

Search bar with Array of Dicts

.h

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

@interface
B11SearchBarForArrayOfDictsViewController :
UIViewController
{
    IBOutlet UITableView *table1;
    IBOutlet UISearchBar *search1;
    BOOL isSearching;
    NSMutableArray
*tableArray,*searchArray,*mainArray;
}

@end
```

.m

```
#import
"B11SearchBarForArrayOfDictsViewController.h"

@implementation
B11SearchBarForArrayOfDictsViewController
-(void)viewDidLoad
{
   NSDictionary *aDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"Radha",@"name",@"
9989633243",@"number",@"abc@gmail.com",@"email",
nil];
      NSDictionary *bDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"Kumar",@"name",@"
223232",@"number",@"abc@gmail.com",@"email", nil];
      NSDictionary *cDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"raju",@"name",@"9
9876437532",@"number",@"abc@gmail.com",@"email",
nil];
      NSDictionary *dDict = [NSDictionary
dictionaryWithObjectsAndKeys:@"prateek",@"name",
@"9987383483",@"number",@"abc@gmail.com",@"ema
il", nil];

    mainArray = [[NSMutableArray alloc]
initWithObjects:aDict,bDict,cDict,dDict,nil];
    searchArray =[[NSMutableArray alloc] init];
    [mainArray retain];
    [searchArray retain];
    isSearching = NO;
     search1.autocorrectionType =
UITextAutocorrectionTypeYes;
    tableArray = [mainArray copy];
}


#pragma Search Bar Delegate Methods
- (void)searchBarTextDidBeginEditing:(UISearchBar
*)searchBar
{
    searchBar.showsCancelButton = YES;
    searchArray = nil;
}
- (void)searchBarSearchButtonClicked:(UISearchBar
*)searchBar
{
    searchBar.showsCancelButton = NO;
    [searchBar resignFirstResponder];
}
- (void)searchBarTextDidEndEditing:(UISearchBar
*)searchBar
{

}
- (void)searchBar:(UISearchBar *)searchBar
textDidChange:(NSString *)searchText
{
    NSLog(@"textDidChange called serch text is
%@",searchText);
    if([searchText isEqualToString:@""]||
searchText==nil)
    {
        isSearching = NO;
        tableArray = mainArray;
        return;
    }
    //Getting first char capital
    //  NSString *firstCapChar = [[searchText
substringToIndex:1] capitalizedString];
```

```
    //searchText= [searchText
stringByReplacingCharactersInRange:NSMakeRange(0,1
) withString:firstCapChar];

    isSearching = YES;
    searchArray = [tableArray
filteredArrayUsingPredicate:
                [NSPredicate
predicateWithFormat:@"(name beginswith[cd] %@)",
searchText]];

    [searchArray retain];
    //[fetchedResultsController.fetchRequest
setPredicate:predicate];




    [table1 reloadData];
}

- (void)searchBarCancelButtonClicked:(UISearchBar *)
searchBar
{
    searchBar.showsCancelButton = NO;
    isSearching = NO;
    tableArray = mainArray;
    [table1 reloadData];
    [searchBar resignFirstResponder];
}

#pragma table view data source
#pragma mark TableView Data source Methods

- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 60;
}

- (NSInteger)numberOfSectionsInTableView:
(UITableView *)tableView
{
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    if(isSearching)
        return [searchArray count];
    else
        return [tableArray count];
}

- (UITableViewCell *)tableView:(UITableView
*)tableView cellForRowAtIndexPath: (NSIndexPath
*)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil)
    {
        cell = [[[UITableViewCell alloc]
initWithFrame:CGRectZero
reuseIdentifier:CellIdentifier] autorelease];
        cell.selectionStyle =
UITableViewCellAccessoryNone;
    }
     if(isSearching)
    {
    //    if( searchArray && [searchArray
count]>indexPath.row)
```

```
        cell.textLabel.text = [ [searchArray
objectAtIndex:indexPath.row]
objectForKey:@"name"];
    }
    else
        cell.textLabel.text =[ [tableArray
objectAtIndex:indexPath.row]objectForKey:@"name"];
    return cell;
}

@end
```

UILocalNotification

```
  UILocalNotification *localNotif =
[[UILocalNotification alloc] init];
    [localNotif setAlertBody:@"fdsfdsfdsf"];
    [localNotif setFireDate:[NSDate date]];
    //[localNotif setSoundName:<#(NSString *)#>];


    [[UIApplication sharedApplication]
scheduleLocalNotification:localNotif];
```

```
    //cancel all notifications
    [[UIApplication sharedApplication]
cancelAllLocalNotifications];

    //i need all notifications
    NSArray *localNotifications  = [[UIApplication
sharedApplication] scheduledLocalNotifications];

    //stop particular notification
    [[UIApplication sharedApplication]
cancelLocalNotification:localNotif];
```

Notes:

1.Project folders creation and adding images and all

2.Naming Conventions and comments and Pragma

-> variable names should be small

-> Class Name starts with Capital

-> variable/method words should be combined with capital letter

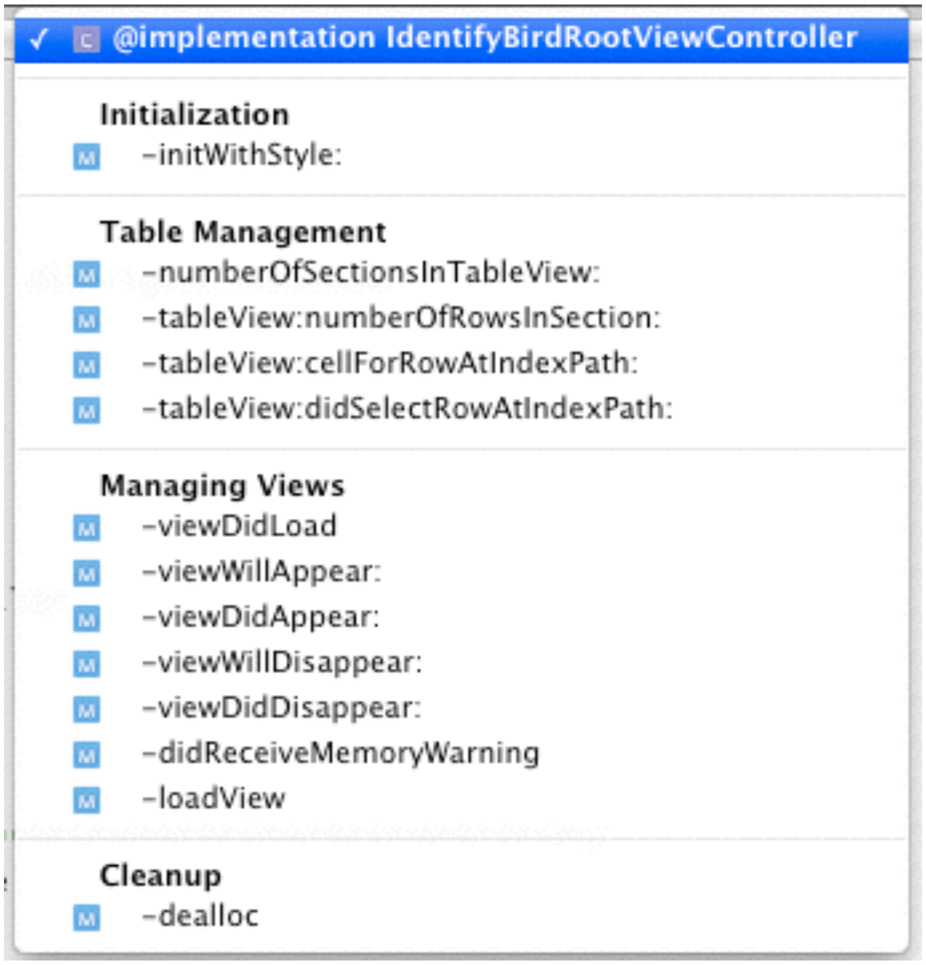
-> method name starts with small letter

-> enum/MACRO/constant names all should be capital

-> image names small

Pragma Mark:

I've started using #pragma mark directives in my code to help with organization as my implementation files grow. #pragma mark is simple to use, for example, insert the following to call out initialization code:



```
✓ @implementation IdentifyBirdRootViewController

Initialization
    -initWithStyle:

Table Management
    -numberOfSectionsInTableView:
    -tableView:numberOfRowsInSection:
    -tableView:cellForRowAtIndexPath:
    -tableView:didSelectRowAtIndexPath:

Managing Views
    -viewDidLoad
    -viewWillAppear:
    -viewDidAppear:
    -viewWillDisappear:
    -viewDidDisappear:
    -didReceiveMemoryWarning
    -loadView

Cleanup
    -dealloc
```

3.Single ton class and AppDelegate Object

A singleton-pattern restricts the instantation of a class to one object. This pattern can be used in many patterns, like a factory, scenegraph or manager if you want to ensure that only one instance of this class exists. A good example is the interface of a subsystem, which is used to access all the functionality in the subsystem. This interface is usually a singleton, since you want to ensure the subsystem has only ONE point of entry.

The singleton is also considered an anti-pattern, since most programmers tend to misuse it to access "global variables" stored in this pattern across the application. Handles are not needed, since the management of the, single instance, is done within the singleton itself. This construction can seem useful, but having references to one single instance all over your system usually leads to code which is unreadable for your fellow programmers.

.h

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

@interface BaseClass : NSObject

+ (BaseClass *) sharedAppController;
@end
```

.m

```
#import "BaseClass.h"
static BaseClass *_sharedAppController = nil;
@implementation BaseClass

+ (BaseClass *) sharedAppController
{
    if (!_sharedAppController)
    {
        _sharedAppController = [[BaseClass alloc] init];
    }
    return _sharedAppController;
}

@end

//Cretting Object for the above Single ton class
[[BaseClass sharedAppController] addition];
————
//Creating an Object for App Delegate class
    AppDelegate *appDelegate  = [UIApplication
sharedApplication].delegate;
    [appDelegate getValuesFromDataBase];
```

4.MACROS real time

```
    #define MAX1 100
//#define  BASE_URL @"http://www.sadad.com/
services"
#define  BASE_URL @"http://www.sadad.com/
servicesasdasd"
#define BASE_COLOR [UIColor redColor]
```

5.enum real time

```
     enum  {
    ORANGE = 0,
    YELLOW = 1
    };
```

6.when u create only .h and .m file

If you want to create model like only coding part like methods and variables in onc enlace then we create .h and .m files

7.icon sizes and info.plist file

Devices	iOS 6 size (in pixel)	iOS 7 size (in pixel)
iPhone Non Retina	57 x 57	**not available**
iPhone Retina	114 x 114	**120 x 120**
iPad Non Retina (mini and 2nd gen)	72 x 72	**76 x 76**
iPad Retina	144 x 144	**152 x 152**

8.How to rename the iPhone Project

Goto 3rd option in View options on XCODE Tool Bar —> Rename it and done

9.How to change the Display name of App

Goto info.plist and then Display name there you can change it

10.How to execute iPhone App in iPad/ iPad Simulator

11.XCODE 5

KVC

Key-Value Coding and Key-Value Observing are really powerful concepts found throughout Apple's frameworks. They allow you to write code more dynamically and even observe and act accordingly when properties in your classes change. This can be really useful, because you can do anything when a property changes, including but not limited to check if its value

is valid, send a message to someone when something changes to a certain value and so on. The options are unlimited.
NSKeyValueCoding informal protocol and implemented by NSObject.NSDictionary has custom implementation of KVC that attempts to match keys against keys in the dictionary

KVO

Move Work off the Main Thread

Be sure to limit the type of work you do on the main thread of your app. The main thread is where your app handles touch events and other user input. To ensure that your app is always responsive to the user, you should never use the main thread to perform long-running or potentially unbounded tasks, such as tasks that access the network. Instead, you should always move those tasks onto background threads. The

preferred way to do so is to use Grand Central Dispatch (GCD) or operation objects to perform tasks asynchronously.
Moving tasks into the background leaves your main thread free to continue processing user input, which is especially important when your app is starting up or quitting. During these times, your app is expected to respond to events in a timely manner. If your app’s main thread is blocked at launch time, the system could kill the app before it even finishes launching. If the main thread is blocked at quitting time, the system could similarly kill the app before it has a chance to write out crucial user data.

Blocks

Blocks are a new feature that was introduced in iOS 4.0 and Mac OSX 10.6. Blocks can greatly simplify code. They can help you reduce code, reduce dependency on delegates, and write cleaner, more readable code.

At its core, a Block is a chunk of code that can be executed at some future time.Blocks are first-class functions, which is a fancy way of saying that Blocks are regular Objective-C objects. Since they're objects, they can be passed as parameters, returned from methods and functions, and assigned to variables.

Blocks are called closures in other languages such as Python, Ruby and Lisp, because they encapsulate state when they are declared. A block creates a const copy of any local variable that is referenced inside of its scope.

Before blocks, whenever you wanted to call some code and have it call you back later, you would typically use delegates or NSNotificationCenter. That worked fine,

except it spreads your code all over – you start a task in one spot, and handle the result in another.
Blocks are nice because they keep your code related to handling a task all in one place

The declaration format for Blocks is as follows:

```
return_type (^block_name)(param_type,
param_type, ...)
```

If you've programmed in any other C-type language, this should look pretty familiar to you, except for that ^ symbol. The ^ symbol is what denotes "this thing we are declaring is a block."

```
int (^add)(int,int)
```

here's a block definition:

```
// Block Definition
^return_type(param_type param_name, param_type
param_name, ...) { ... return return_type; }
```

It begins with the ^ symbol and is followed by the parameters, which must be named at this point, and must match the type and order of the parameter list of the Block declaration to which it's assigned. This is followed by the actual code.

Eg:

```
^(int number1, int number2){ return
number1+number2 }
```

Block declaration and definition example together

```
int (^add)(int,int) = ^(int number1, int number2){
                                  return number1+number2;
}
```

Now let's look at the same code as above using fast-enumeration:

```
BOOL stop;
int idx = 0;
for (id obj in theArray) {
    NSLog(@"The object at index %d is %@",idx,obj);
    if (stop)
        break;
    idx++;
}
```

And now with Blocks:

```
 NSArray *aArray = [[NSArray alloc]
initWithObjects:@"one",@"two",@"three",@"four",
nil];
    [aArray enumerateObjectsUsingBlock:^ (id
obj,NSUInteger idx, BOOL *status){
        NSLog(@"the object is %@",obj);
    }];
```

1. **Simplicity**. Using Blocks, we have access to the object, the object index in the array, and a stop variable, all without having to write any code. This means less code, which means less chance of a coding error (not that we make any coding errors).

Eg 1:

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    [UIView animateWithDuration:5.0
                     animations:^{
                        [animatingView setAlpha:0];
                        [animatingView
setCenter:CGPointMake(animatingView.center.x+50.0,

animatingView.center.y+50.0)];
                     }
                     completion:^(BOOL finished) {
                         [animatingView removeFromSuperview];
                     }];
}
```

Eg 2:

```
+ (NSArray*)retrieveInventoryItems {
    // 1 - Create variables
    NSMutableArray* inventory = [NSMutableArray
new];
    NSError* err = nil;
    // 2 - Get inventory data
    NSArray* jsonInventory = [NSJSONSerialization
JSONObjectWithData:[NSData
dataWithContentsOfURL:[NSURL
URLWithString:kInventoryAddress]]

options:kNilOptions
                                                  error:&err];
    // 3 - Enumerate inventory objects
    [jsonInventory enumerateObjectsUsingBlock:^(id
obj, NSUInteger idx, BOOL *stop) {
        NSDictionary* item = obj;
        [inventory addObject:[[IODItem alloc]
initWithName:[item objectForKey:@"Name"]
                                          andPrice:[[item
objectForKey:@"Price"] floatValue]
                                     andPictureFile:[item
objectForKey:@"Image"]]];
    }];
    // 4 - Return a copy of the inventory data
    return [inventory copy];
}
```

Multithreading and Grand Central Dispatch (GCD)

4564646

4564646

4564646

4564646

4564646

4564646