Guía iPhone

maestros del web

Guía de desarrollo de aplicaciones móviles para iPhone / iPad

Javier Cala Uribe

@maestros

facebook.com/maestrosdelweb

Introducción

En la actualidad existe un fuerte interés por parte de los programadores en el desarrollo de aplicaciones para dispositivos móviles como: iPad, iPhone e iPod Touch.

Dispositivos móviles que se hacen cada vez más populares en el mercado de las comunicaciones, porque proporcionan una plataforma con sistema operativo como iPhone OS 4.0. La adquisición de este tipo de aparatos por parte de los usuarios se incrementa en la medida que encuentran en ellos nuevas aplicaciones o servicios que satisfacen sus necesidades básicas, profesionales, educativas y de entretenimiento.

Tienes en tu pantalla la Guía de desarrollo de aplicaciones para iPhone/ iPad de Maestros del Web diseñada con el objetivo de enseñarte las principales características del entorno de programación para dispositivos móviles con una serie de ejemplos y recursos que muestran la facilidad de las herramientas de desarrollo.

Autor

Javier Cala Uribe

Javier Cala Uribe, Ingeniero de Sistemas, actualmente es desarrollador para iOS (Objective-C) con 2 años de experiencia en la plataforma y más de 4 años como programador. Ha publicado 15 aplicaciones en la App Store de Apple y cooperado en el desarrollo de tres proyectos de aplicaciones. Contribuyó al diseño, análisis e implementación de interfaces entre sistemas SAP Enterprise Resource Planning (ERP) y un sistema legado basado en RPG/COBOL sobre i5/OS (iSeries IBM). También diseñó e implementó el ciclo de pruebas para un sistema de administración de recursos basado en ASP.NET

Versión 1 / noviembre 2010

Autor: Javier Cala Uribe
Guías de desarrollo para aplicaciones móviles

Nivel: Intermedio y Avanzado

Indice

Guía de desarrollo de aplicaciones móviles para iPhone / iPad

La primera sección de la guía estará dedicada al desarrollo de aplicaciones para los dispositivos móviles iPhone, iPad, iPod Touch.

- Uso del navegador en una app.....6
- Uso del correo.....16
- Capturar imágenes desde la cámara del iPhone.....24
- Uso de varias vistas.....31
- Trabajando con bases de datos SQL Lite.....36
- Trabajando con el API de Facebook Connect.....41
- Trabajando con el API de Twitter.....47
- Trabajando con el API de OpenFeint.....53

Desarrollo de vídeo juegos
La segunda sección de la guía estará dedicada a la creación de vídeo juegos para dispositivos móviles a través del uso de Chipmunk y Cocos2D.

Chipmunk:

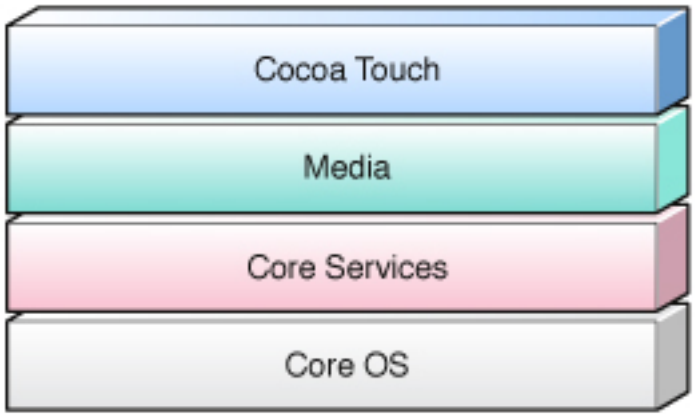
- Motor de física 2D – Parte 1.....62
- Motor de física 2D – Parte 2.....66
- Motor de física 2D – Parte 3.....72

Cocos 2D:

- Framework para desarrollar vídeo juegos en 2D – Parte 1.....76
- Framework para desarrollar vídeo juegos en 2D – Parte 2.....80
- Framework para desarrollar vídeo juegos en 2D – Parte 3.....86

CAPITULO 1:
Uso de navegador en una app

iPhone OS es el sistema implementado por Apple que permite ejecutar aplicaciones nativas en los dispositivos móviles: iPhone, iPod Touch e iPad. La arquitectura de esta plataforma toma como base el núcleo del Sistema Mac OS X e incorpora una nueva capa que da soporte a la interfaz multi-touch y al acelerómetro.



Los requisítos mínimos para desarrollar en esta plataforma son:

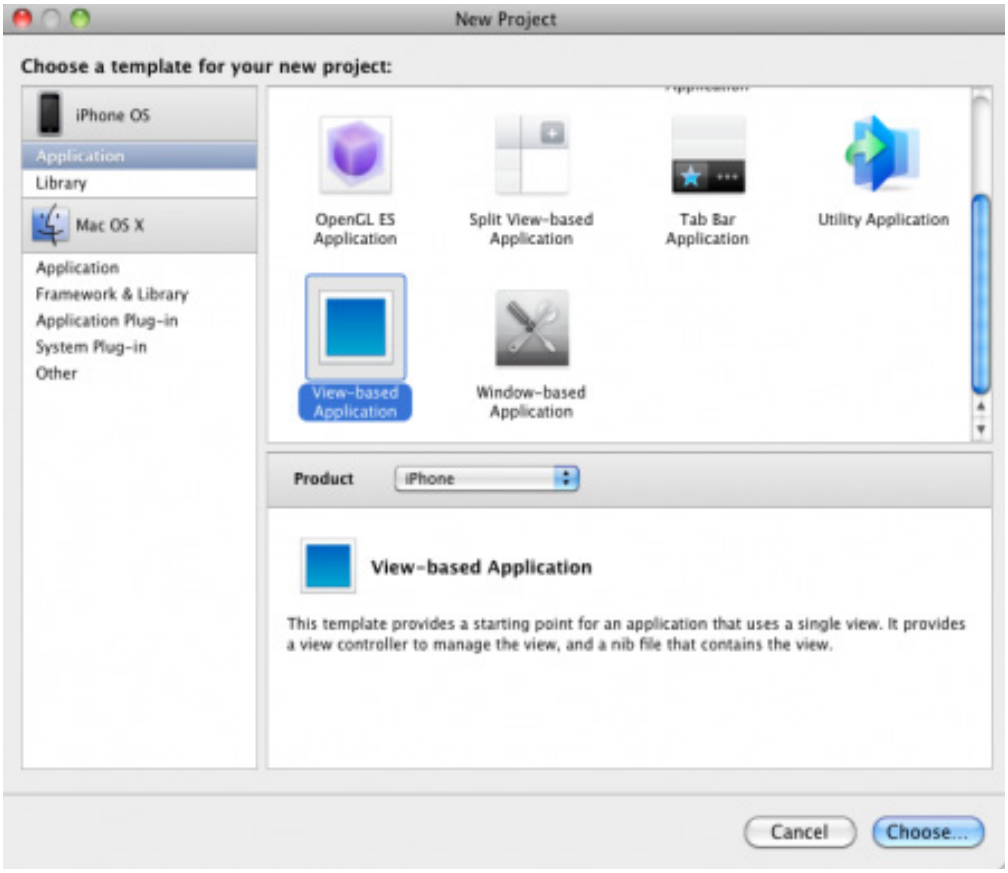
- Mac OS X 10.5 (Leopard) o posterior
- iPhone SDK 2.0 o posterior
- Dispositivo móvil para pruebas (opcional)

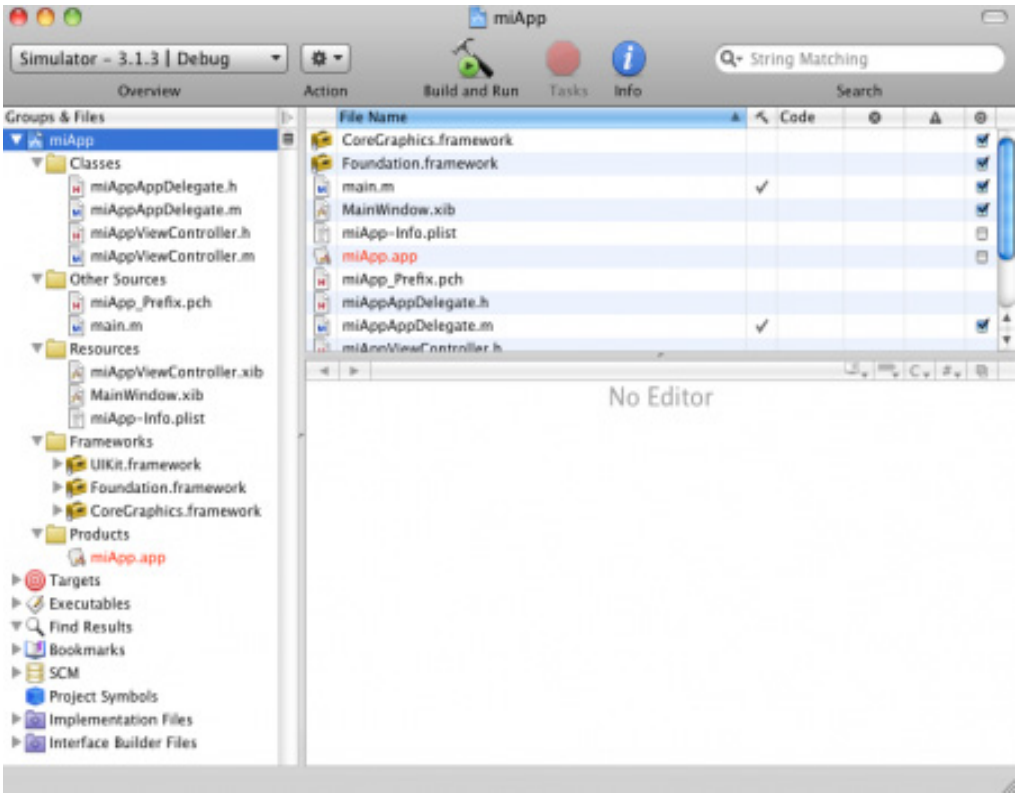
El iPhone SDK contiene el código, la información y las herramientas necesarias para desarrollar, probar, ejecutar, depurar y ajustar las apps para el iPhone OS. Dentro de este kit encontramos tres aplicaciones fundamentales:

- **Xcode:** contiene un conjunto de herramientas para el desarrollo de las aplicaciones, permite editar, depurar y compilar el código fuente.
- **Interface Builder:** permite la creación de interfases gráficas y vinculación con Xcode.
- **iPhone Simulator:** ejecuta las aplicaciones desarrolladas en un emulador del dispositivo.

Creando un proyecto en Xcode con Interface Builder y el iPhone Simulator

1. Abrir Xcode e ir a **“File->New Project”** y seleccionar **“View-based Application”**





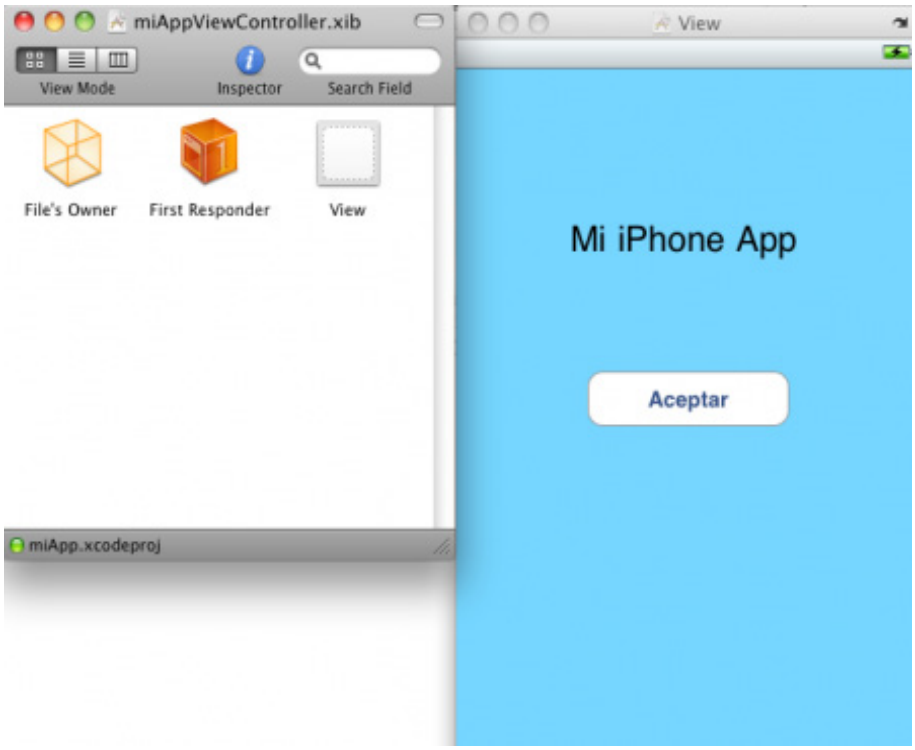
2. Declarar una etiqueta (**UILabel)** y una función (**IBAction**) en la clase **"miAppViewController.h"**

```
#import <UIKit/UIKit.h>
@interface miAppViewController : UIViewController {
UILabel *miEtiqueta;
}
@property (nonatomic,retain) IBOutlet UILabel *miEtiqueta;
-(IBAction)cambiarEtiqueta;
@end
```

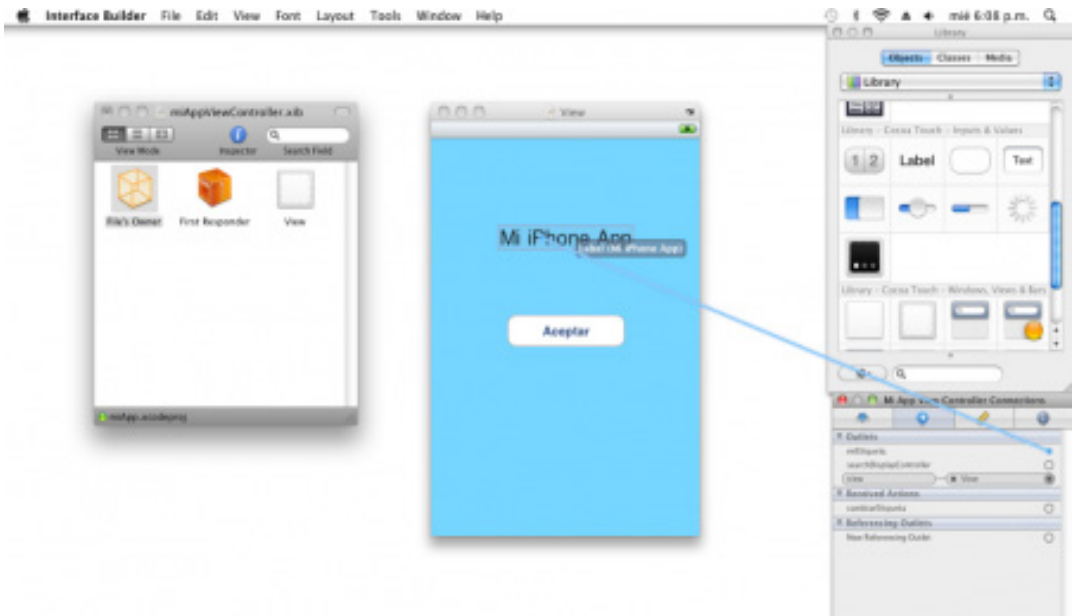
3. Definir la función (IBAction) en "miAppViewController.m"

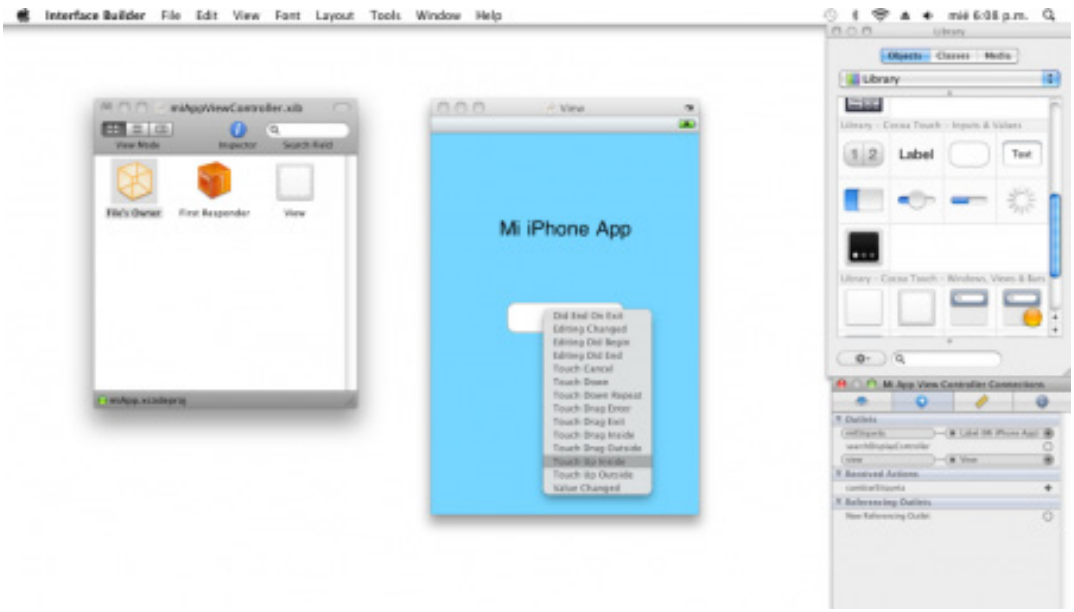
```
@synthesize miEtiqueta;
-(IBAction)cambiarEtiqueta{
miEtiqueta.text = @"Bazzinga!";
}
- (void)dealloc {
[miEtiqueta release];
[super dealloc];
}
```

4. Abrir con doble clic el archivo **"miAppViewController.xib"**, agregar una etiqueta (**UILabel**) y un botón (**UIButton**) en la vista del controlador.



5. En la ventana *"Connections Inspector"* arrastrar **miEtiqueta** hasta la etiqueta creada y la función **"cambiarEtiqueta"** hasta el botón creado.





6. Presionar en Xcode el botón **“Build and Run”**

La aplicación se ejecutará en el **iPhone Simulator** como lo muestra la imagen anterior. Para encontrar más información sobre el entorno de programación del sistema iPhone OS pueden revisar:
http://developer.apple.com/technologies/tools/xcode.html

Uso del navegador en una app

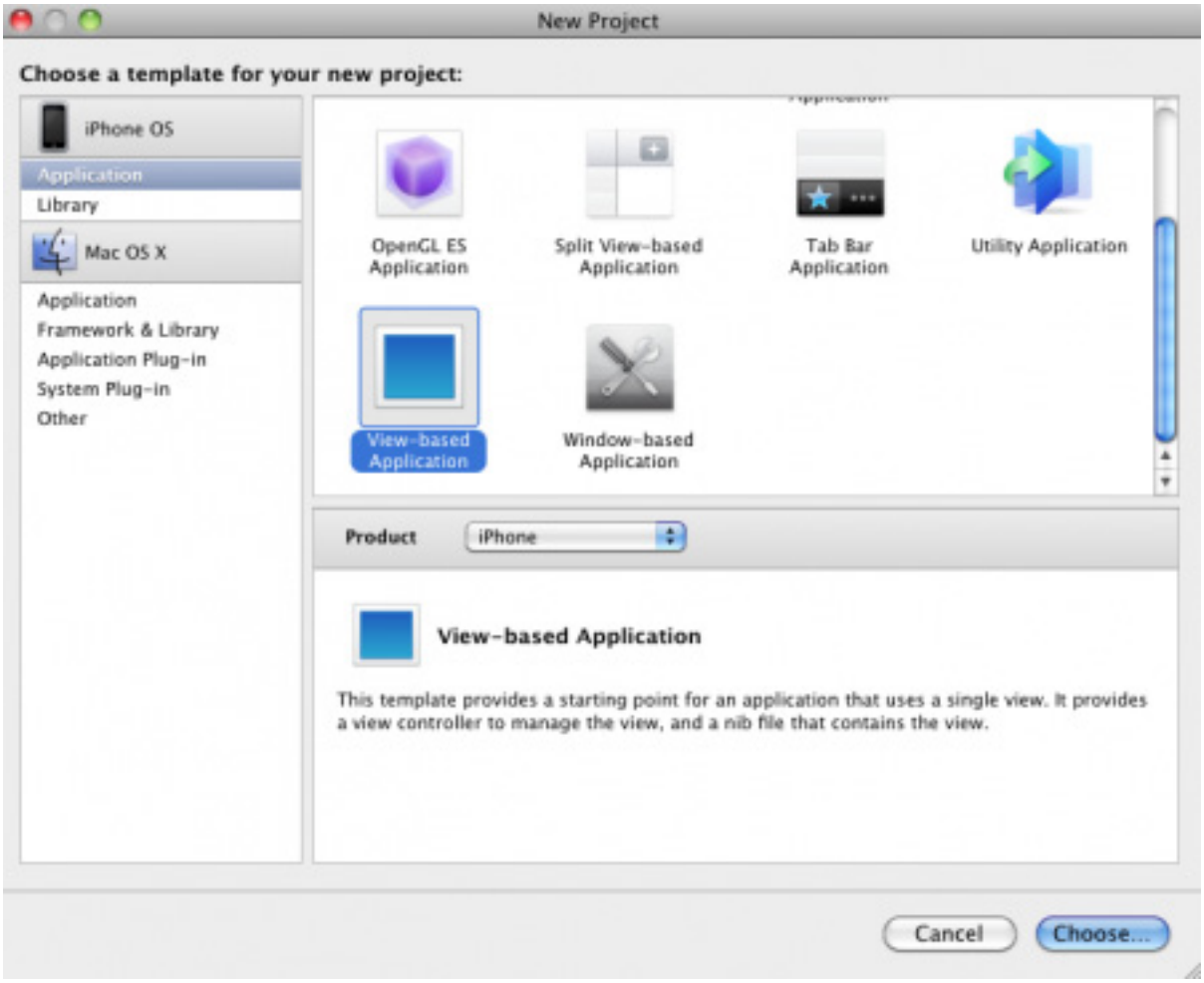
Ahora, aprenderemos a utilizar la clase UIWebView que nos permite mostrar contenido web embebido en el dispositivo móvil.

Conociendo UIWebView:

UIWebView despliega información Web embebida en nuestra aplicación sin necesidad de salir de la misma, es decir, el usuario puede ver contenidos Web en la aplicación sin abrir Safari en el dispositivo. Aunque claramente la clase UIWebView esta basada en Safari, no requiere cerrar la aplicación para mostrar los contenidos. La implementación es realmente sencilla, solo se debe crear un objeto UIWebView y cargar el contenido web. También se puede agregar la opción de avanzar o retroceder en el historial de navegación.

UIWebView en acción

1. Crear un nuevo proyecto en Xcode de tipo **"View-Based Application"**

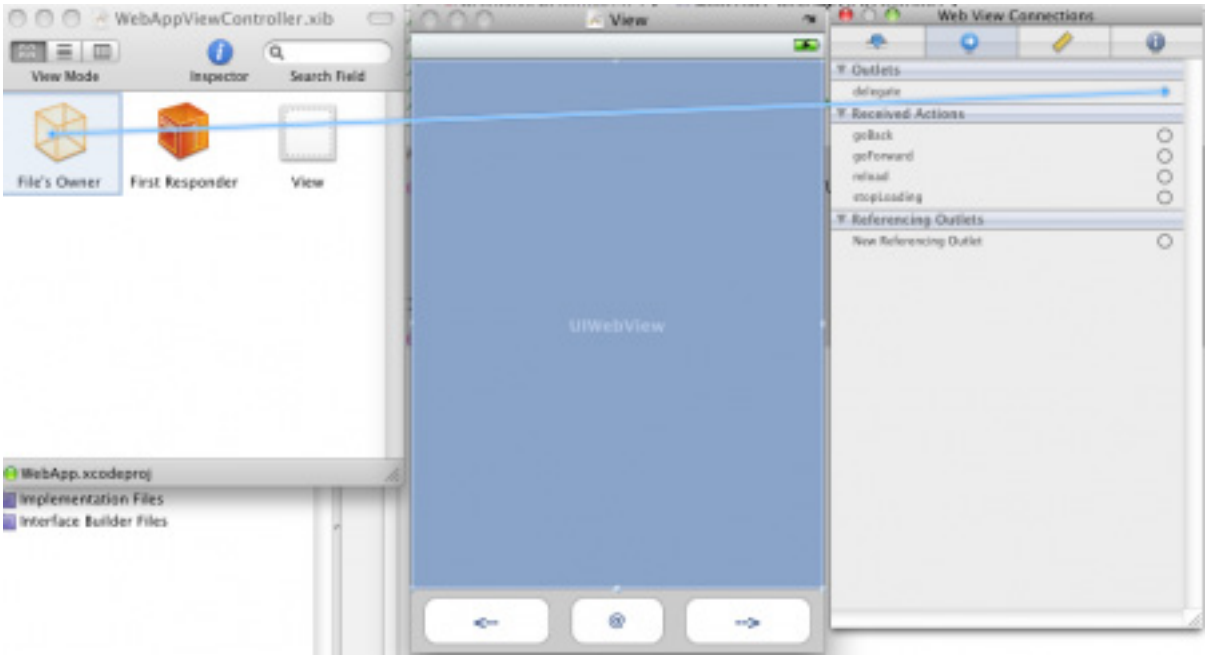


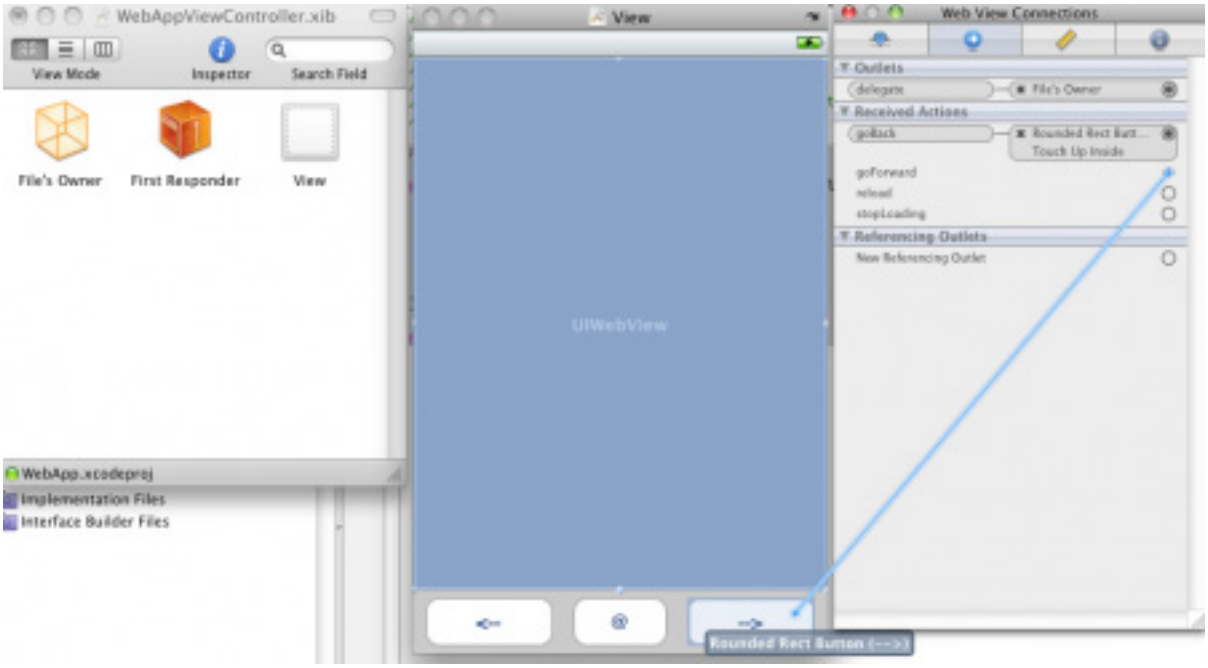
2. En **"NombreAppViewController.h"** declarar los elementos

```
#import <UIKit/UIKit.h>
@interface WebAppViewController : UIViewController &lt;UIWebViewDelegate&gt; {
IBOutlet UIWebView *webView;
IBOutlet UIButton *goBackBtn;
IBOutlet UIButton *goForwardBtn;
IBOutlet UIButton *reloadBtn;
}
@end
```

3. Abrir el archivo **“NombreAppViewController.xib”** y agregar los siguientes elementos declarados previamente: **1 UIWebView y 3 botones**

4. Seleccionar el objeto **UIWebView** y desde el **Inspector Connections** arrastrar: delegate, goBack, goForward, reload.





5. En Xcode se debe definir la función en **"NombreAppViewController.m"**

```
 - (void)viewDidLoad {
[super viewDidLoad];
NSURL *url = [NSURL URLWithString:@"http://www.maestrosdelweb.com"];
NSURLRequest *loadURL = [[NSURLRequest alloc] initWithURL:url];
[webView loadRequest:loadURL];
[loadURL release];
}
```

CAPITULO 2:
Uso del Correo

6. Compilar y ejecutar:

Si ha salido todo bien debe cargar la página previamente definida en la función **(void) viewDidLoad.** La clase UIWebView es de gran utilidad para mantener la información actualizada de una aplicación y también permite mayor libertad en el diseño para la presentación de los contenidos. Igualmente se pueden cargar archivos HTML localmente, previamente definidos en la aplicación.

En este capítulo revisaremos un tema de gran utilidad para compartir información: el uso del correo a partir de la versión 3.0 del iPhone OS, esta disponible la clase MFMailComposeViewController para envíar correos desde el iPhone. Las versiones anteriores utilizan la aplicación Mail que trae por defecto el dispositivo. En este capítulo veremos como implementar la clase MFMailComposeViewController y hacer el llamado a la aplicación Mail para las versiones anteriores al iPhone OS 3.0.

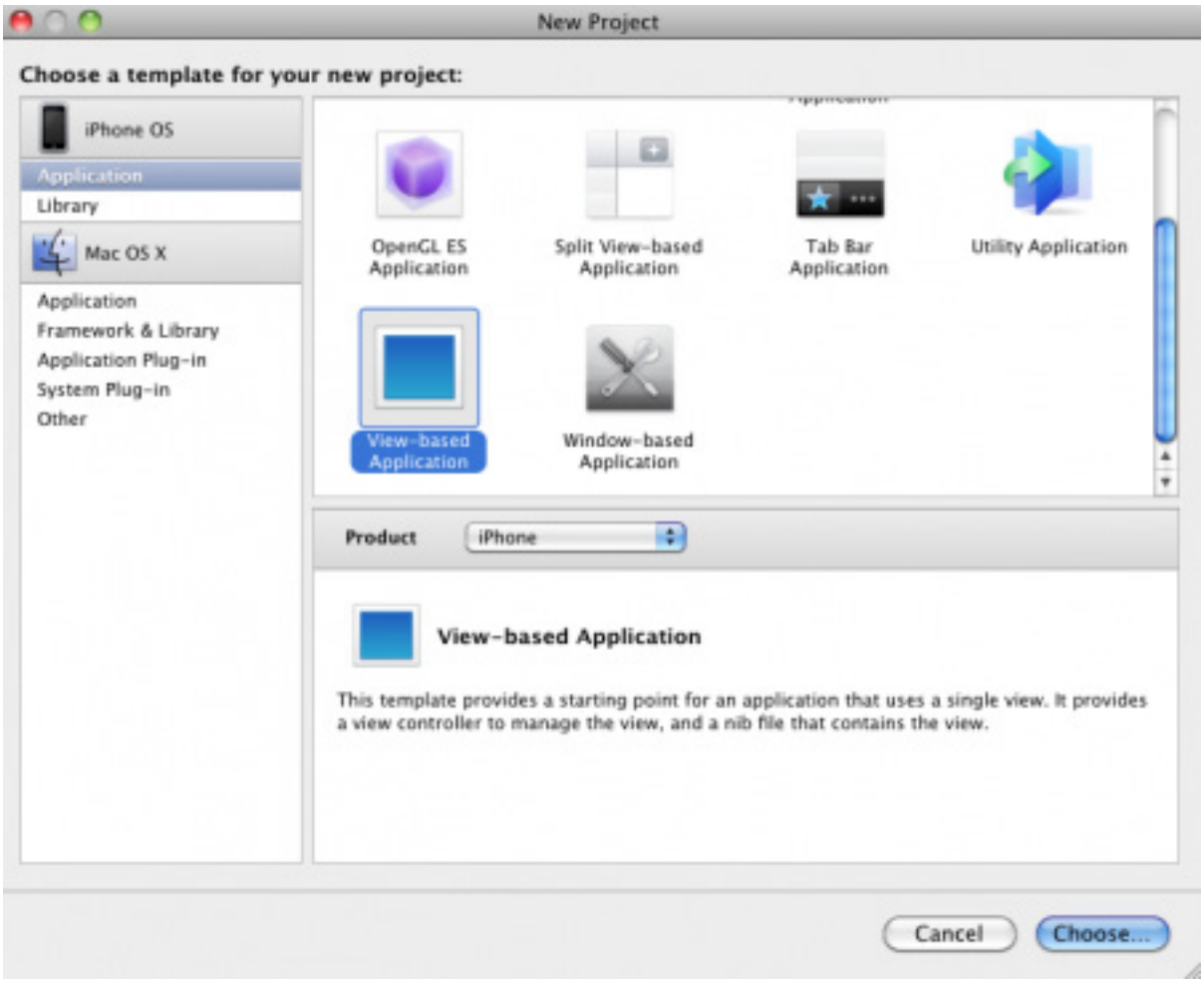
Explorando MFMailComposeViewController:

Esta clase brinda una interfase que permite administrar, editar y enviar correos electrónicos. Cuenta con un formulario predefinido con los campos: subject, email recipients, body text y attachments, es decir, un formulario normal de correo. Uno de los inconvenientes con esta clase es que no permite verificar si efectivamente el correo enviado llegó a su destino.

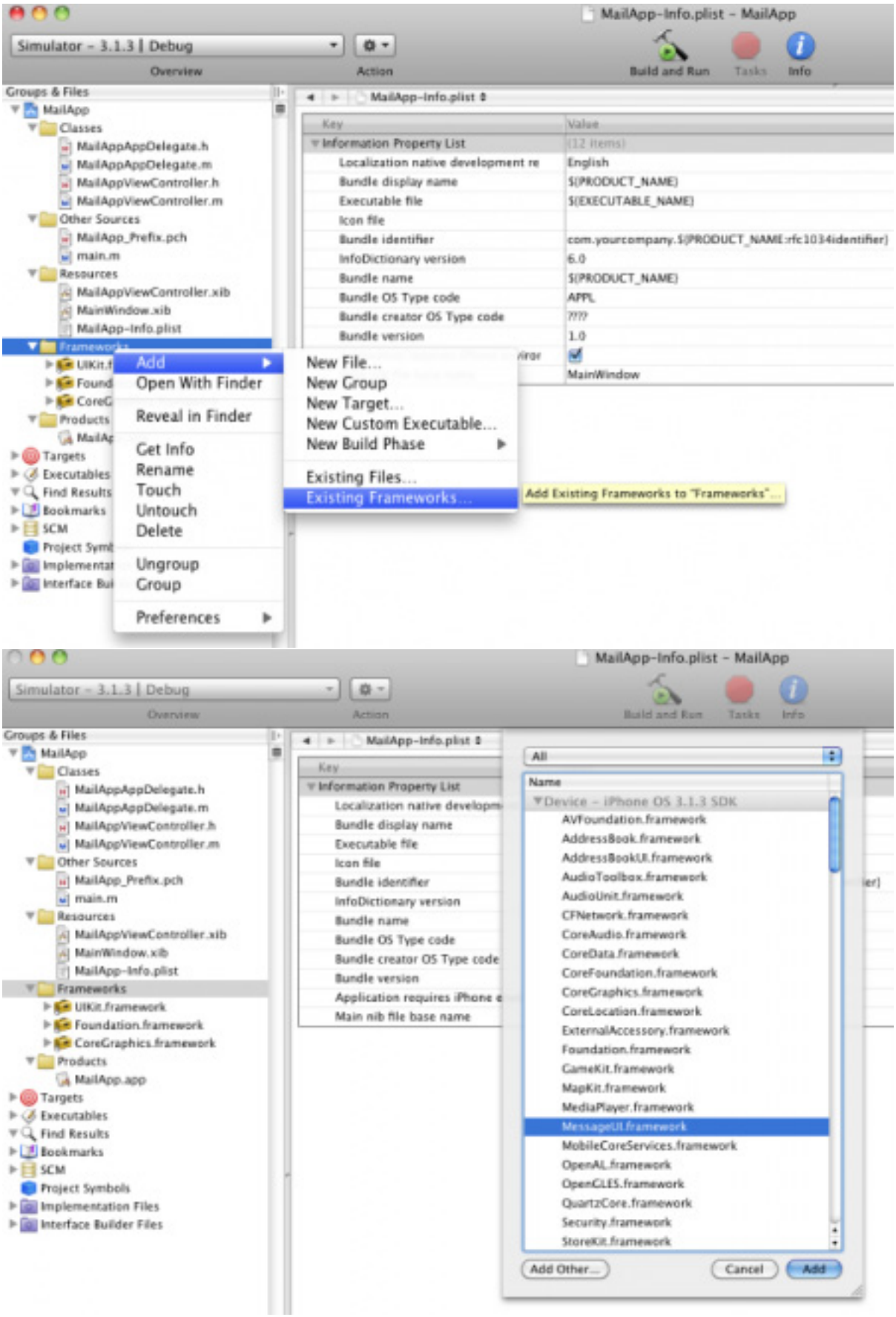
La clase se encarga de colocar los mensajes enviados en el buzón de salida de la aplicación Mail, resultando útil para el envió de correos electrónicos cuando no se cuenta con conexión a la red, pero inapropiado para confirmar el envío.

Sent from my iPhone Simulator

1. Crear un nuevo proyecto en Xcode de tipo **"View-Based Application"**



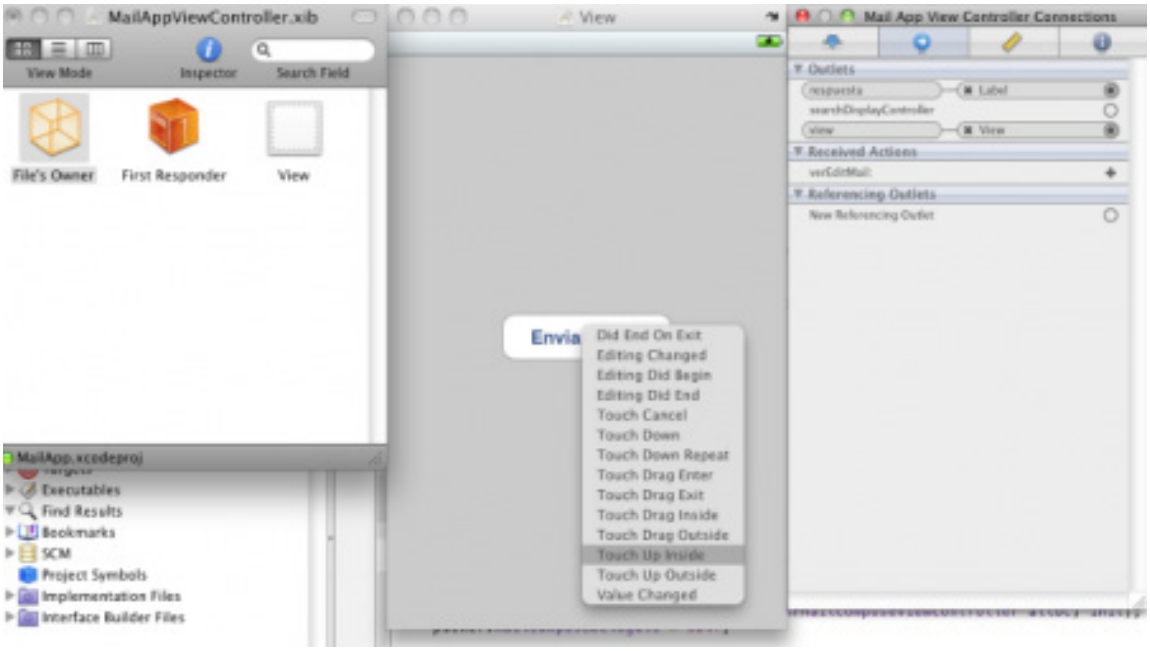
2. Agregar el framework **MessageUI:**



3. En **“NombreAppViewController.h”** agregar los elementos:

```
#import <UIKit/UIKit.h>
#import <MessageUI/MessageUI.h>
#import <MessageUI/MFMailComposeViewController.h>
@interface MailAppViewController : UIViewController
// Delegate de la clase MFMailComposeViewController
&lt;MFMailComposeViewControllerDelegate&gt;
{
IBOutlet UILabel *respuesta;
}
@property (nonatomic, retain) IBOutlet UILabel *respuesta;
// Verifica si esta disponible la clase MFMailComposeViewController
-(IBAction)verEditMail:(id)sender;
// Configura vista para editar y enviar un email
-(void)configurarMail;
// Ejecuta la App Mail del dispositivo
-(void)ejecutarMailApp;
@end
```

4. Abrir el archivo **“NombreAppViewController.xib”** en *Interface Builder*, agregar los elementos **UIButton, UILabel** y relacionarlos con la App:

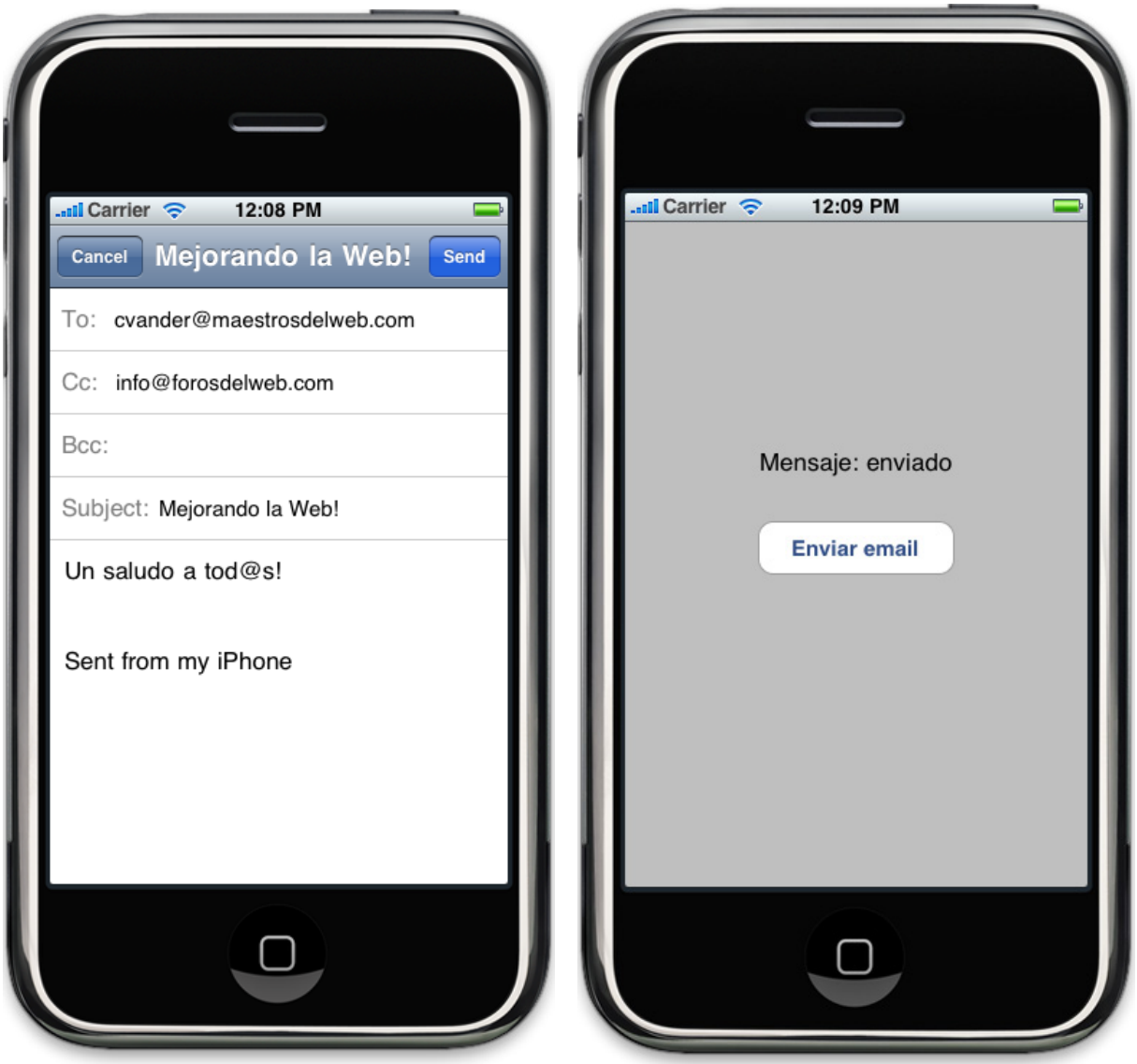


5. Definir las siguientes funciones de **"NombreAppViewController.m"** en *Xcode:*

```
(IBAction)verEditMail:(id)sender {
Class mailClass = (NSClassFromString(@"MFMailComposeViewController"));
if (mailClass != nil) {
// Verifica que este habilitada la opcion para enviar correos en el dispositivo
if ([mailClass canSendMail])
[self configurarMail];
else
[self ejecutarMailApp];
}
else {
[self ejecutarMailApp];
}
}
// Configura vista para editar y enviar un email
-(void)configurarMail {
MFMailComposeViewController *mailView = [[MFMailComposeViewController alloc] init];
mailView.mailComposeDelegate = self;
[mailView setSubject:@"Mejorando la Web!"];
// Destinatarios
NSArray *toRecipients = [NSArray arrayWithObject:@"cvander@maestrosdelweb.com"];
NSArray *ccRecipients = [NSArray arrayWithObjects:@"info@forosdelweb.com",  nil];
[mailView setToRecipients:toRecipients];
[mailView setCcRecipients:ccRecipients];
// Mensaje
NSString *emailBody = @"Un saludo a tod@s!";
[mailView setMessageBody:emailBody isHTML:NO];
[self presentModalViewController:mailView animated:YES];
[mailView release];
}
```

```
- (void)mailComposeController:(MFMailComposeViewController*)controller didFinishWithResult:(M
FMailComposeResult)result error:(NSError*)error{
respuesta.hidden = NO;
// Notifica al usuario los resultados del envio
switch (result)
{
case MFMailComposeResultCancelled:
respuesta.text = @"Mensaje: cancelado";
break;
case MFMailComposeResultSaved:
respuesta.text = @"Mensaje: guardado";
break;
case MFMailComposeResultSent:
respuesta.text = @"Mensaje: enviado";
break;
case MFMailComposeResultFailed:
respuesta.text = @"Mensaje: falló";
break;
default:
respuesta.text = @"Mensaje: no enviado";
break;
}
[self dismissModalViewControllerAnimated:YES];
}
// Ejecuta la App Mail del dispositivo
-(void)ejecutarMailApp
{
NSString*recipients=@"mailto:cvander@maestrosdelweb.com?cc=info@forosdelweb.
com&amp;amp;subject=Mejorando la Web!";
NSString *body = @"&amp;amp;body=Un saludo a tod@s!";
NSString *email = [NSString stringWithFormat:@"%@%@", recipients, body];
email = [email stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
[[UIApplication sharedApplication] openURL:[NSURL URLWithString:email]];
}
```

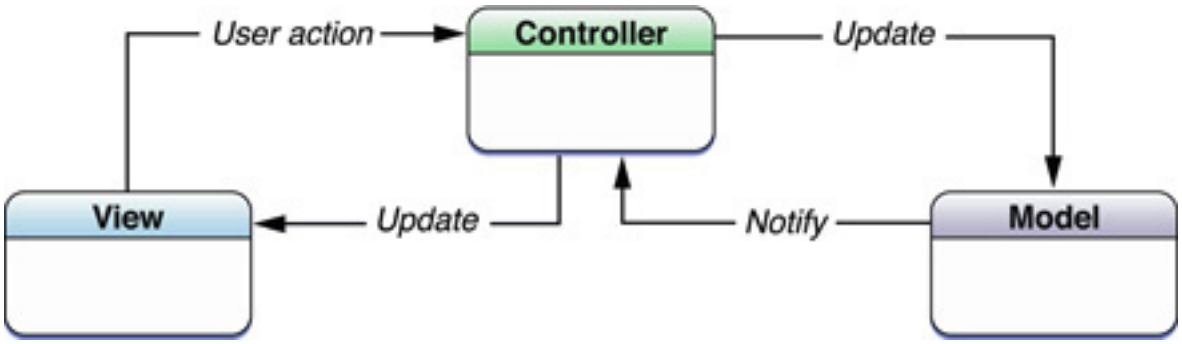
6. Compilar y ejecutar:



CAPITULO 3:
Capturar imágenes desde la cámara del iPhone

Utilizaremos el controlador **UIImagePickerController** para acceder a las imágenes guardadas en el dispositivo o capturar imágenes desde un iPhone.

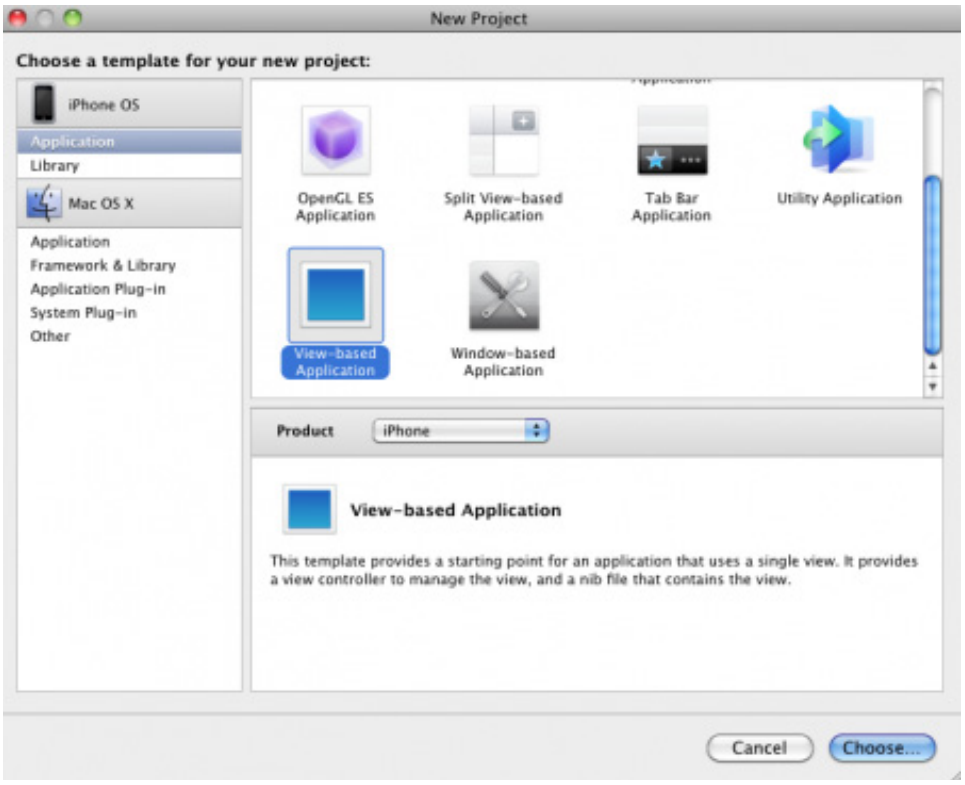
Arquitectura de Cocoa-Touch
Antes de entrar en detalle repasemos un poco la arquitectura utilizada por Cocoa-Touch (iPhone SDK) para reconocer que elementos del framework implementaremos según nuestras necesidades:



Esta arquitectura (Modelo-Vista-Controlador) separa los datos, interfaz de usuario y lógica de control en tres componentes como se aprecia en el gráfico. Para el caso del iPhone SDK podemos ejemplificar la arquitectura de la siguiente manera : Core Data (Modelo), UIView (Vista), UIViewController (Controlador).

En este capítulo utilizaremos el controlador UIImagePickerController, el cual se encarga de gestionar la implementación de imágenes o vídeo en una aplicación. Esta clase controla la interfase de usuario y retorna el mensaje una vez terminada su utilización.

UIImagePickerController en acción:
1. Crear un nuevo proyecto en Xcode de tipo **"View-Based Application":**



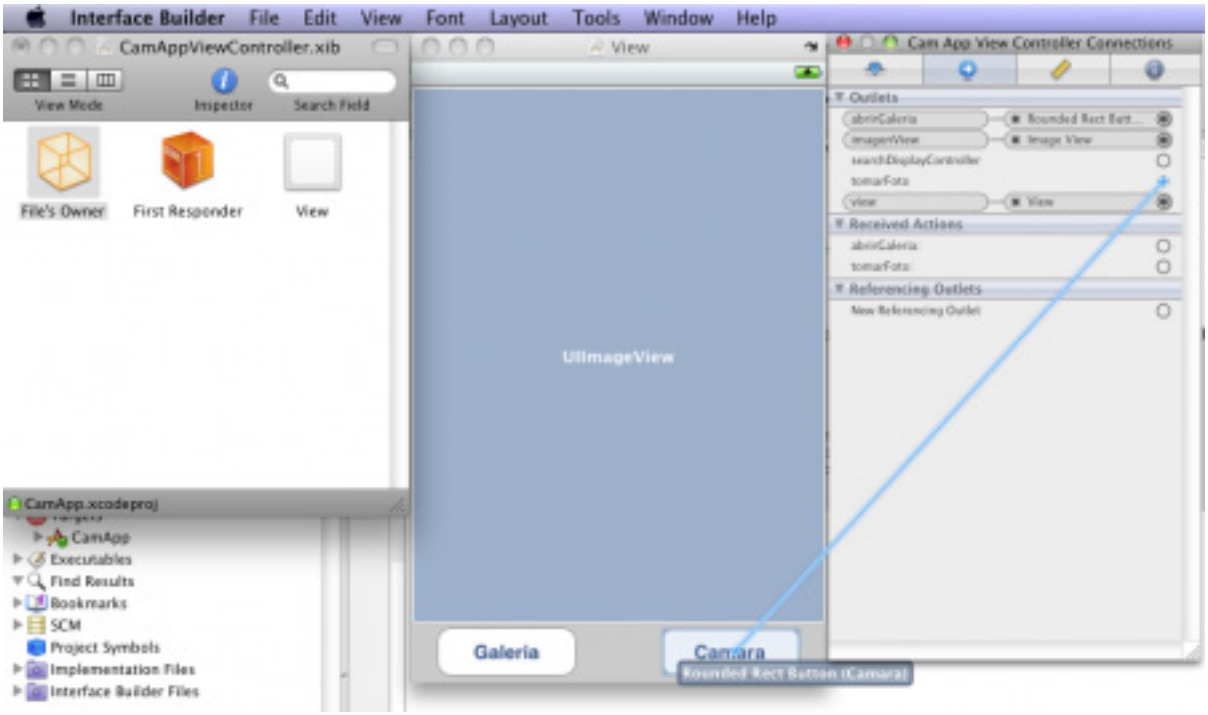
2. Agregar los siguientes elementos en **"NombreAppViewController.h":**

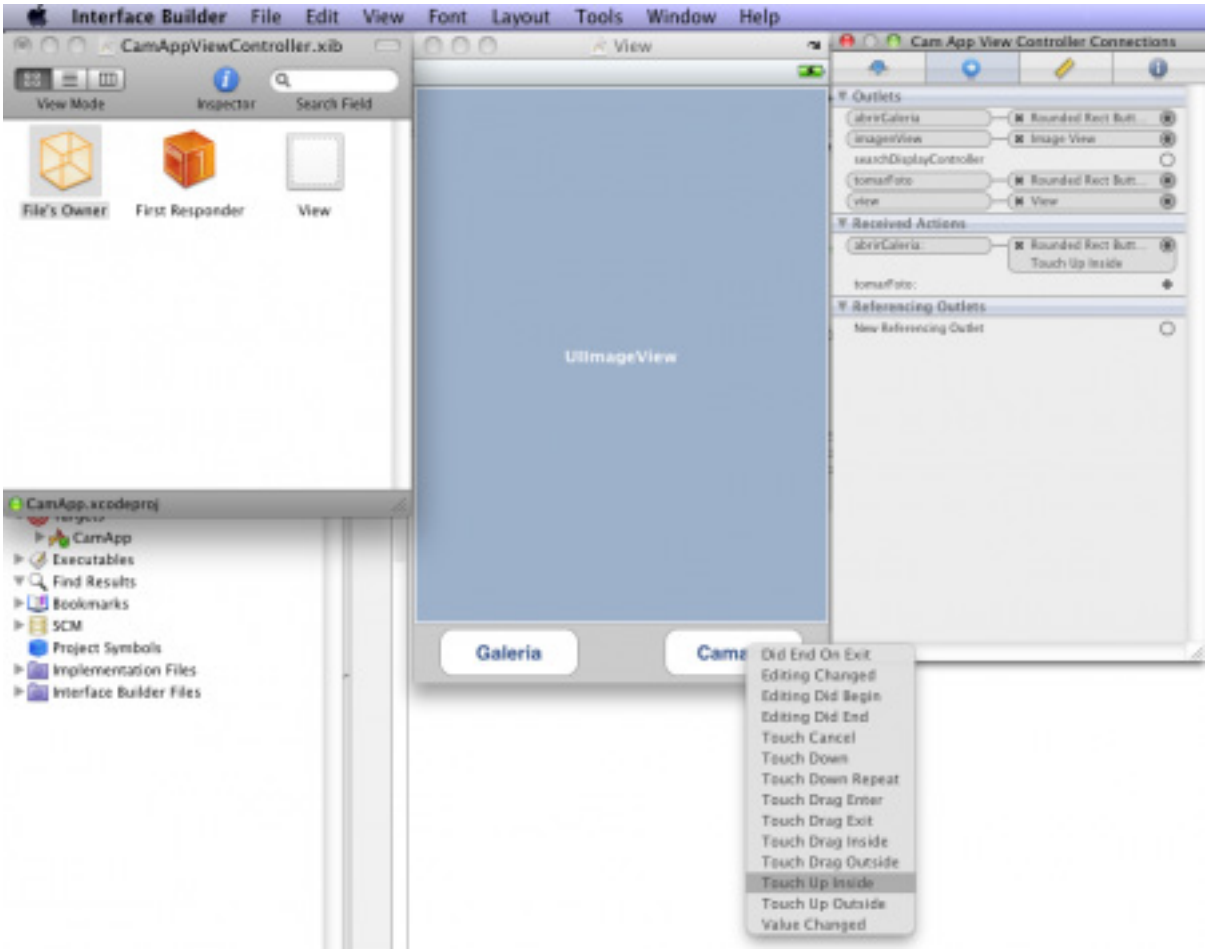
```
@interface CamAppViewController : UIViewController <UIImagePickerControllerDelegate, UINavi-
gationControllerDelegate> {
UIImageView *imagenView;
UIButton    *abrirGaleria;
UIButton    *tomarFoto;
}
@property (nonatomic, retain) IBOutlet UIImageView *imagenView;
@property (nonatomic, retain) IBOutlet UIButton *abrirGaleria;
@property (nonatomic, retain) IBOutlet UIButton *tomarFoto;
- (IBAction)abrirGaleria:(id)sender;
- (IBAction)tomarFoto:(id)sender;
```

3. Abrir el archivo **"NombreAppViewController.xib"** (Interface Builder) y agregar **1 UIImageView** y **2 UIButton**


View
UIImageView
Galeria
Camara

4. Desde el *Connections Inspector* enlazar los elementos creados en Interface Builder con Xcode:





5. Definir en Xcode las funciones declaradas previamente:

```
#import “CamAppViewController.h”
@implementation CamAppViewController
@synthesize imagenView, abrirGaleria, tomarFoto;
- (IBAction)abrirGaleria:(id)sender{
// Inicia el Controlador
UIImagePickerController * picker = [[UIImagePickerController alloc] init];
// Define el Delegate
picker.delegate = self;
// Establece el origen de la imagen
picker.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;
// Agrega la vista del controlador a la pantalla
[self presentModalViewController:picker animated:YES];
}
- (IBAction)tomarFoto:(id)sender{
UIImagePickerController * picker = [[UIImagePickerController alloc] init];
picker.delegate = self;
picker.sourceType = UIImagePickerControllerSourceTypeCamera;
[self presentModalViewController:picker animated:YES];
}
// Recibe el mensaje cuando el controlador a finalizado
- (void)imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(
NSDictionary *)info {
// Quita la vista del controlador
[picker dismissModalViewControllerAnimated:YES];
// Establece la imagen tomada en el objeto UIImageView
imagenView.image = [info objectForKey:@”UIImagePickerControllerOriginalImage”];
}
- (void)dealloc {
[imagenView release];
[abrirGaleria release];
[tomarFoto release];
[super dealloc];
}
@end
```

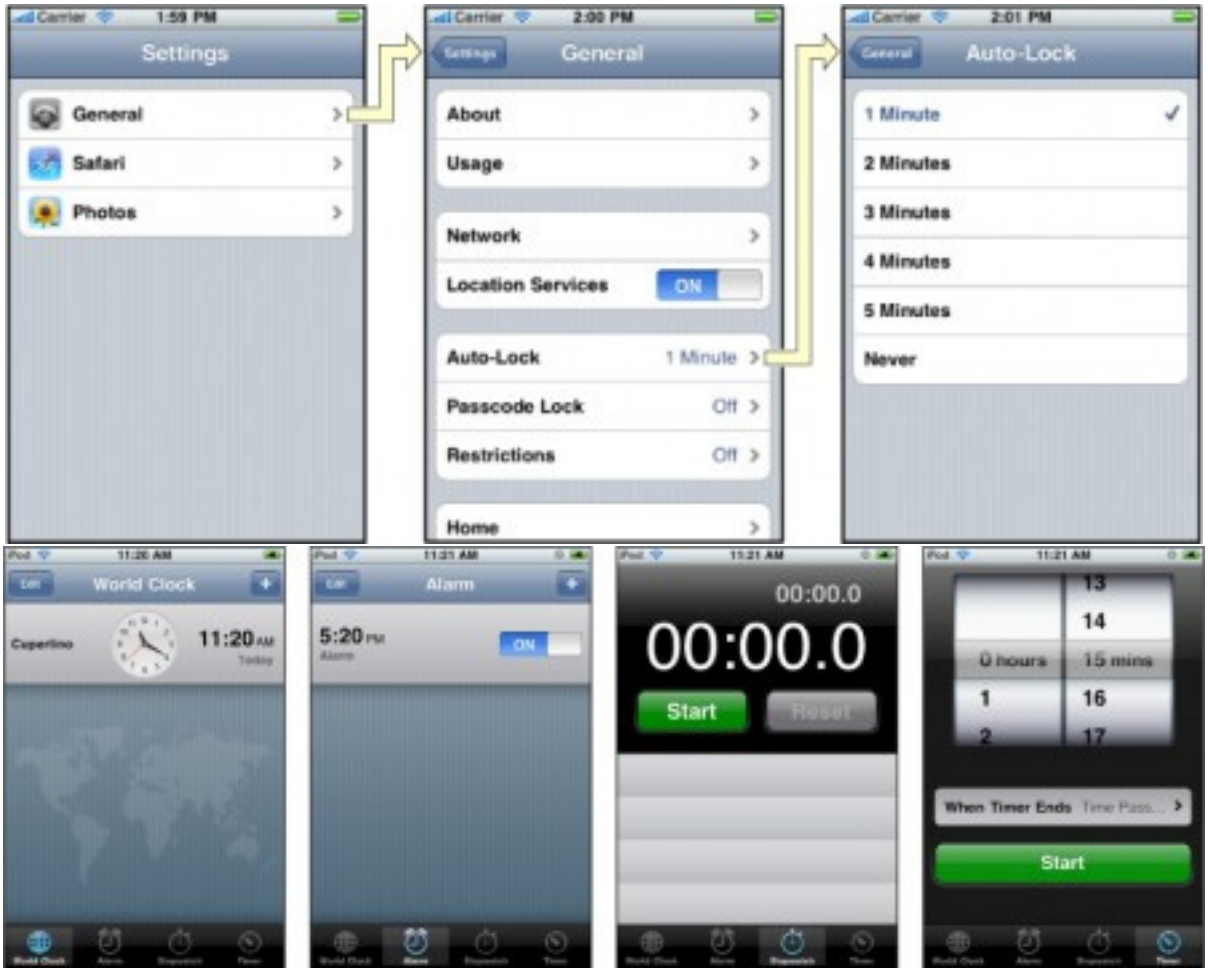
6. Compilar y ejecutar:

Si esperas tomar fotos desde el iPhone Simulator recibirás un sorprendente mensaje en la consola.

CAPITULO 4:
Uso de varias vistas

Revisaremos los principales métodos utilizados para navegar dentro de una aplicación: UINavigationController y UITabBarController. Dependiendo del diseño que definamos, se puede utilizar cualquiera de los dos, incluso los ambos en la misma aplicación.

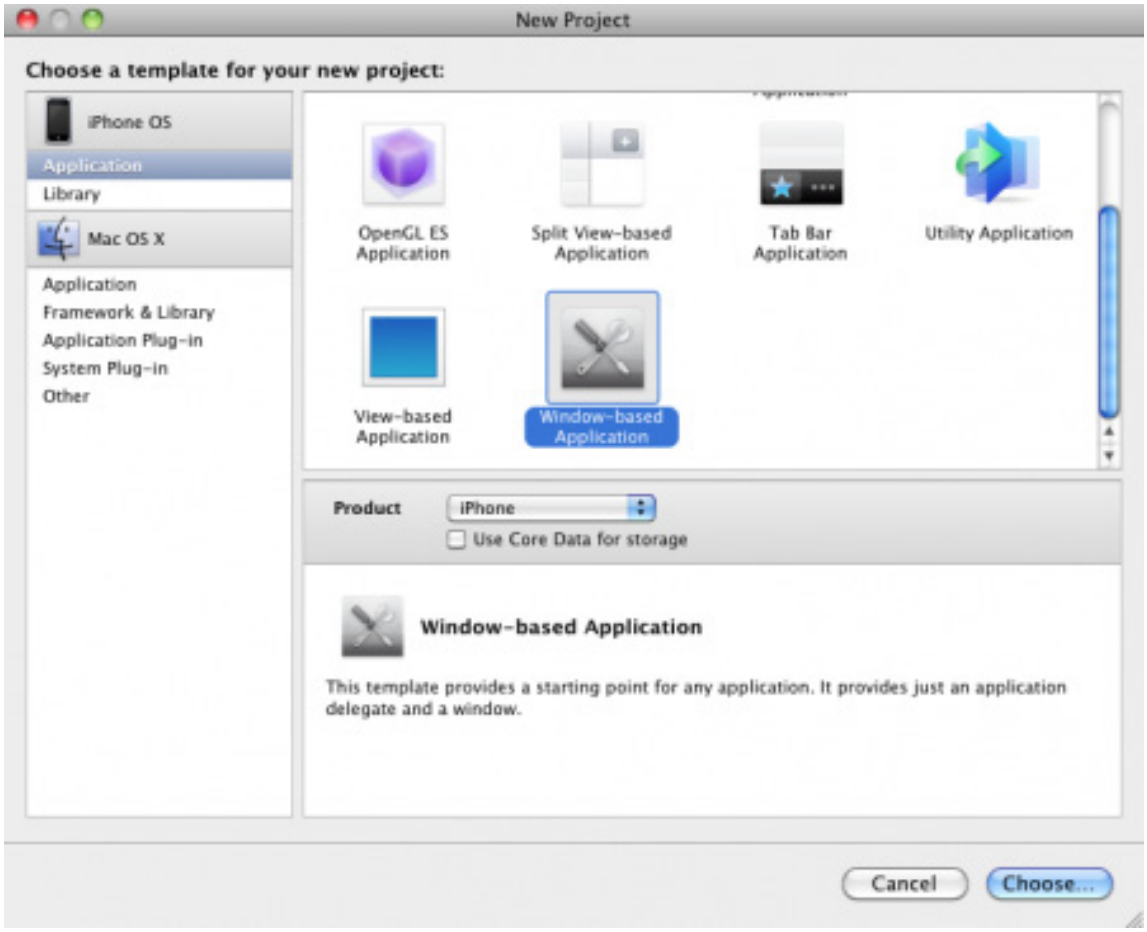
Diferencia entre UINavigationController y UITabBarController
Una imagen dice mas que mil palabras, asi que dos dicen muchísimo más:



UINavigationController crea una jerarquía de vistas, a diferencia de UITabBarController que genera instancias independientes de vistas. También es posible implementar el controlador UINavigationController dentro de los elementos de un UITabBarController.

Implementando UINavigationController y UITabBarController:

1. Crear un nuevo proyecto en Xcode de tipo **Windows-Based Aplication**



2. Agregar los siguientes elementos en NombreAppAppDelegate.h:

```
@interface UINavTabAppAppDelegate : NSObject <UIApplicationDelegate>
{
    UIWindow *window;
    UITabBarController *tabBarController;
    UINavigationController *navigationController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@end
```

3. Inicializa los controladores con solo código sin usar Interface Builder desde NombreAppAppDelegate.h

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
launchOptions
{
    // Inicializa los controladores
    navigationController = [[UINavigationController alloc] init];
    tabBarController       = [[UITabBarController alloc] init];

    // Inicializa la primer vista del TabBarController
    UIViewController *primerViewController = [[UIViewController alloc] init];
    primerViewController.title = @"Primero";

    // Inicializa la segunda vista del TabBarController
    UIViewController *segundoViewController = [[UIViewController alloc] init];
    segundoViewController.title = @"Segundo";

    // Agrega la segunda vista al controlador UINavigationController
    [navigationController pushViewController:segundoViewController animated:NO];

    // Agrega las vistas creadas al controlador UITabBarController
    // 1. Primer Vista = UIViewController
    // 2. Segunda Vista = UINavigationController
     [tabBarController setViewControllers:[NSArray arrayWithObjects:primerViewController, naviga-
tionController, nil] animated:YES];

    // Libera memoria con los controladores ya utilizados
    [primerViewController release];
    [segundoViewController release];

    // Agrega el controlador UITabBarController a la ventana principal
    [window addSubview:[tabBarController view]];

    [window makeKeyAndVisible];

    // Retorna YES XD
    return YES;
}
```

4. Compilar y Ejecutar:

Al final obtendrás dos vistas desde el UITabBarController, una de ellas contiene un UINavigationController. También se pueden implementar estos controladores utilizando Interface Builder que de manera muy intuitiva permite arrastrar y establecer los controladores respectivos. En caso de presentar algún inconveniente en los pasos puedes descargar y realizar el proceso de "Build and Go".

Descargar:
http://www.maestrosdelweb.com/images/2010/06/UINavTabApp.zip

CAPITULO 5:
Trabajando con bases de datos SQL Lite

En el desarrollo de aplicaciones para el iPhone constantemente nos encontramos con la necesidad de guardar información y disponer de ella más adelante. Para dar solución a esto podemos implementar SQLite un pequeño, rápido y confiable sistema de gestión de bases de datos que está disponible para el iPhone.

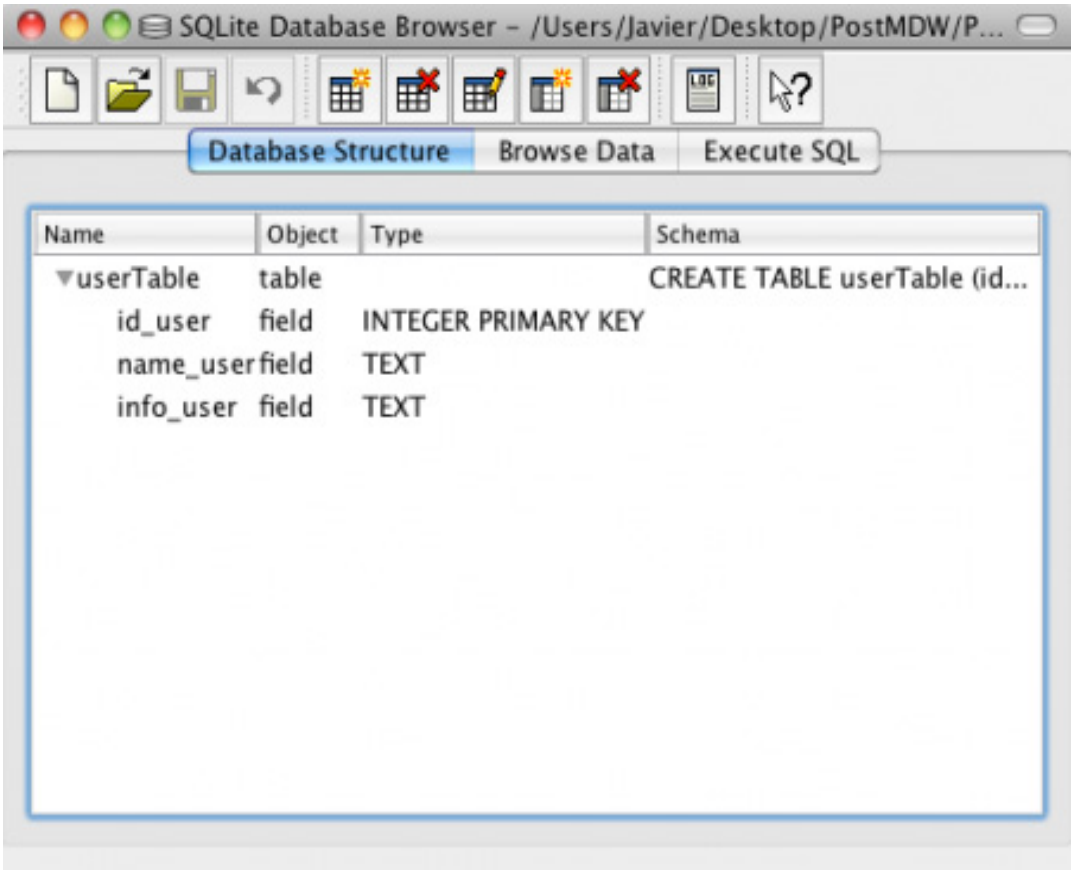
Características de SQLite:

- Es un motor de base de datos SQL embebido y no tiene un proceso de servidor independiente.
- Lee y escribe directamente en archivos de disco normal. Está contenida en un archivo de disco único y una completa base de datos con tablas, índices y vistas.
- Formato de archivo de base de datos multi-plataforma (32-bits y 64-bits).
- No considerar SQLite como un reemplazo para Oracle, sino como un sustituto de fopen ()

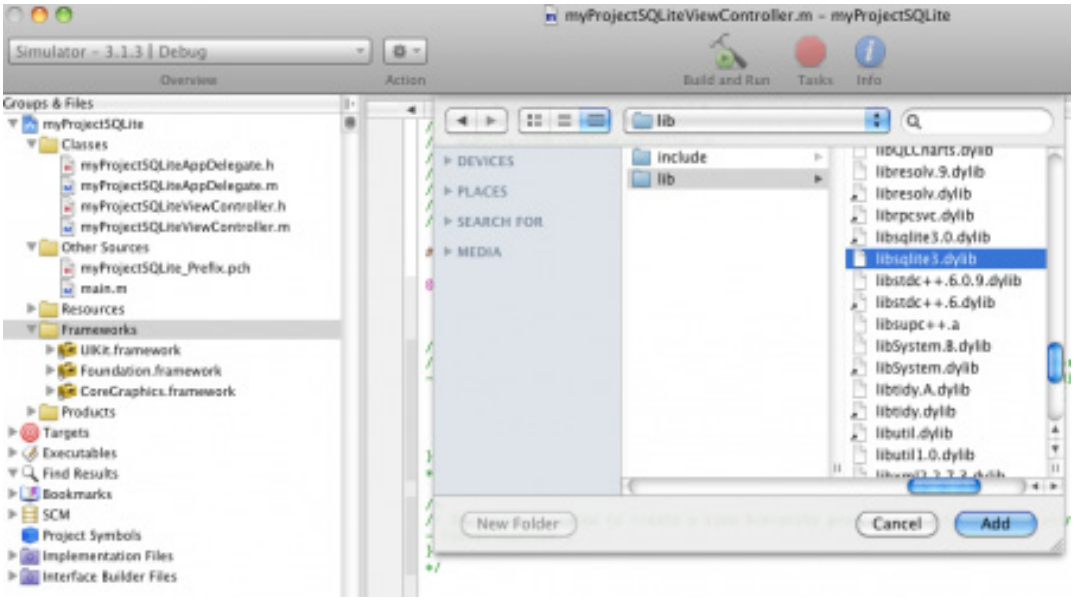
Implementando SQLite en una aplicación:

Para administrar los archivos creados que se utilizarán en nuestra aplicación podemos descargar SQLite Database Browser, una aplicación gratuita que nos brinda una interfaz gráfica para crear, diseñar y editar archivos de base de datos compatibles con SQLite. Empezaremos creando un proyecto en Xcode:

1. Crear un archivo de base de datos con las siguiente estructura:



2. Agregar el archivo de base de datos y la librería **“libsqlite3.dylib”** al proyecto:



3. Definir un método privado para la creación del archivo de base de datos en la aplicación:

```
@interface myProjectSQLiteAppDelegate (Private)
- (void)createEditableCopyOfDatabaseIfNeeded;
@end

- (void)createEditableCopyOfDatabaseIfNeeded {
BOOL success;
NSFileManager *fileManager = [NSFileManager defaultManager];
NSError *error;
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomain-
Mask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *writableDBPath = [documentsDirectory stringByAppendingPathComponent:@”myDB”];
success = [fileManager fileExistsAtPath:writableDBPath];

// Si ya existe el archivo, no lo crea -_-
if (success) return;

// Crea una copia del archivo en el dispositivo móvil

NSString *defaultDBPath = [[[NSBundle mainBundle] resourcePath] stringByAppendingPathComp
onent:@”myDB”];
success = [fileManager copyItemAtPath:defaultDBPath toPath:writableDBPath error:&error];
if (!success) {
NSAssert1(0, @”Failed to create writable database file with message ‘%@’.”, [error localized-
Description]);
}
}
```

4. Definir el método para realizar las consultas en la base de datos:

```
-(void)executeSentence:(NSString *)sentence sentenceIsSelect:(BOOL )isSelect{

// Variables para realizar la consulta
static sqlite3 *db;
sqlite3_stmt *resultado;
const char* siguiente;

// Buscar el archivo de base de datos
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomain-
Mask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *path = [documentsDirectory stringByAppendingPathComponent:@"myDB"];

// Abre el archivo de base de datos
if (sqlite3_open([path UTF8String], &db) == SQLITE_OK) {

// Verifica el tipo de consulta
if (isSelect){

// Ejecuta la consulta

if ( sqlite3_prepare(db,[sentence UTF8String],[sentence length],&resultado,&amp;siguiente) ==
SQLITE_OK ){

// Recorre el resultado
while (sqlite3_step(resultado)==SQLITE_ROW){
NSLog([NSString stringWithFormat:@"ID:%@ NAME:%@ INFO:%@",
[NSString stringWithUTF8String: (char *)sqlite3_column_text(resultado, 0)],
[NSString stringWithUTF8String: (char *)sqlite3_column_text(resultado, 1)],
[NSString stringWithUTF8String: (char *)sqlite3_column_text(resultado, 2)] ]
);
}
}
}
else {

// Ejecuta la consulta
if ( sqlite3_prepare_v2(db,[sentence UTF8String],[sentence length],&resultado,&siguiente) ==
SQLITE_OK ){
sqlite3_step(resultado);
sqlite3_finalize(resultado);
}
}
}
// Cierra el archivo de base de datos
sqlite3_close(db);
}
```

5. Definir las consultas a realizar

```
- (void)viewDidLoad {
[super viewDidLoad];
NSString *sentencetDB = @"insert into userTable values ( NULL, 'Javier', 'Programador' )";
[self executeSentence:sentencetDB sentenceIsSelect:NO];
sentencetDB = @"select * from userTable";
[self executeSentence:sentencetDB sentenceIsSelect:YES];
}
```

6. Compilar y ejecutar:

E la consola se podrán observar los resultados de la consulta realizada. Para asegurar que todo esta bien, pudes descargar el proyecto para realizar el genial procedimiento de **"Build and Go".**

Descargar: http://www.maestrosdelweb.com/images/2010/04/myProjectSQLite.zip

CAPITULO 6:
Trabajando con el API de Facebook Connect

Para avanzar con el tema de vinculación de redes sociales en este capítulo veremos como implementar Facebook Connect en nuestro proyecto. Si aún no tienes claro porqué implementar Facebook Connect quizá estas razones te ayuden.

Razones:
http://www.maestrosdelweb.com/editorial/aprovechando-las-ventajas-de-facebook-connect-sobre-vbulletin/

Características de Facebook Connect:

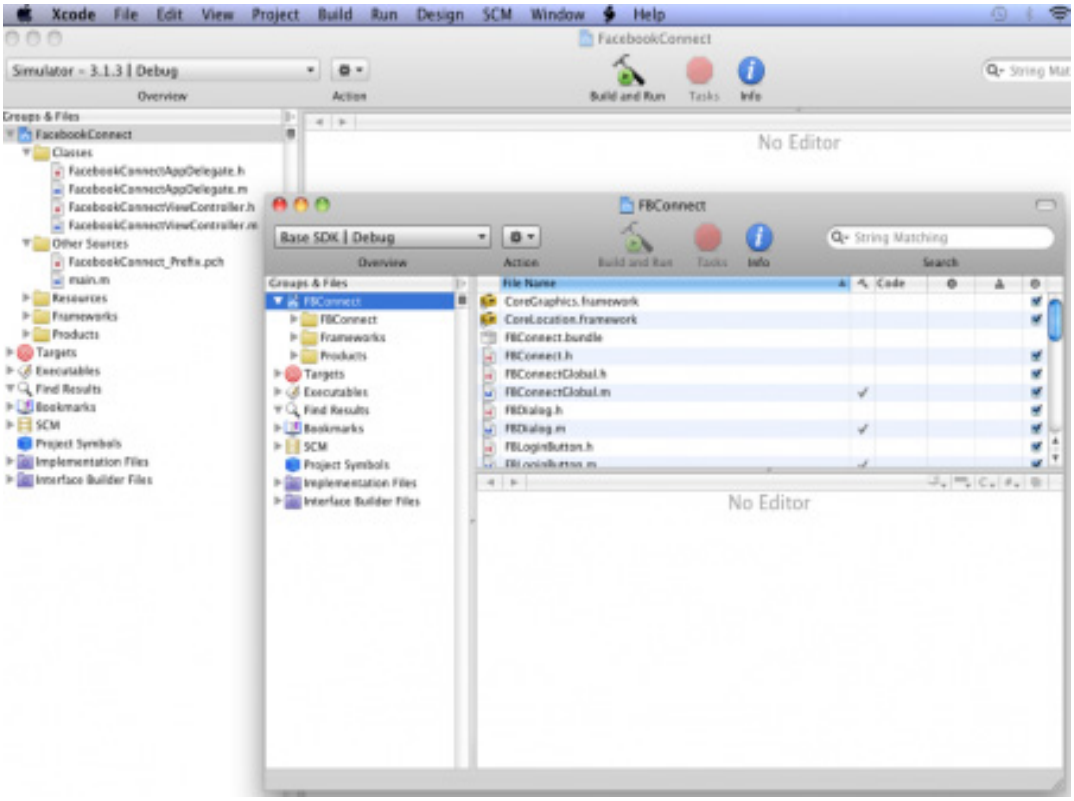
- **Conectar:** la cuenta e información de Facebook con nuestra aplicación.
- **Compartir:** información con los contactos que también utilicen la aplicación.
- **Comprobar:** la identidad real de los usuarios.
- **Actualización:** de la información de los usuarios y las directivas de privacidad constantemente.
- **Facebook Feed:** permite compartir fácilmente información con los contactos.

Implementando Facebook Connect en una aplicación:

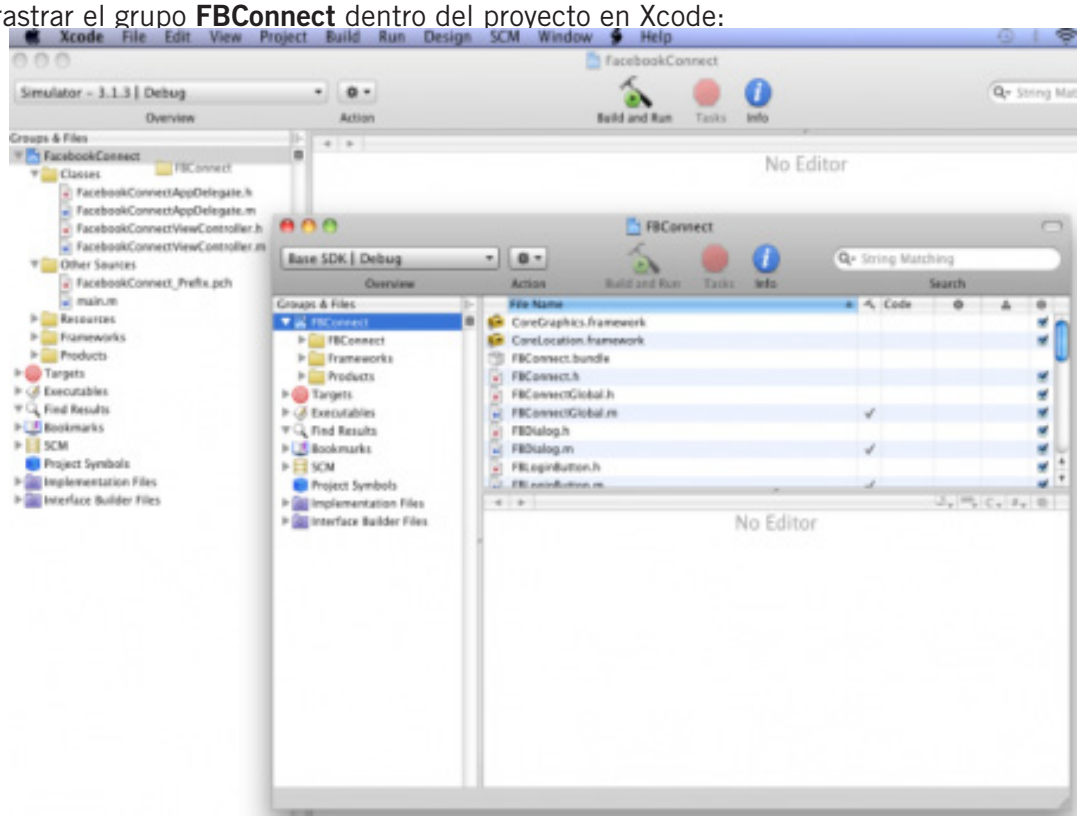
Damos inicio creando un nuevo proyecto en Xcode de tipo **"View-based Application"**

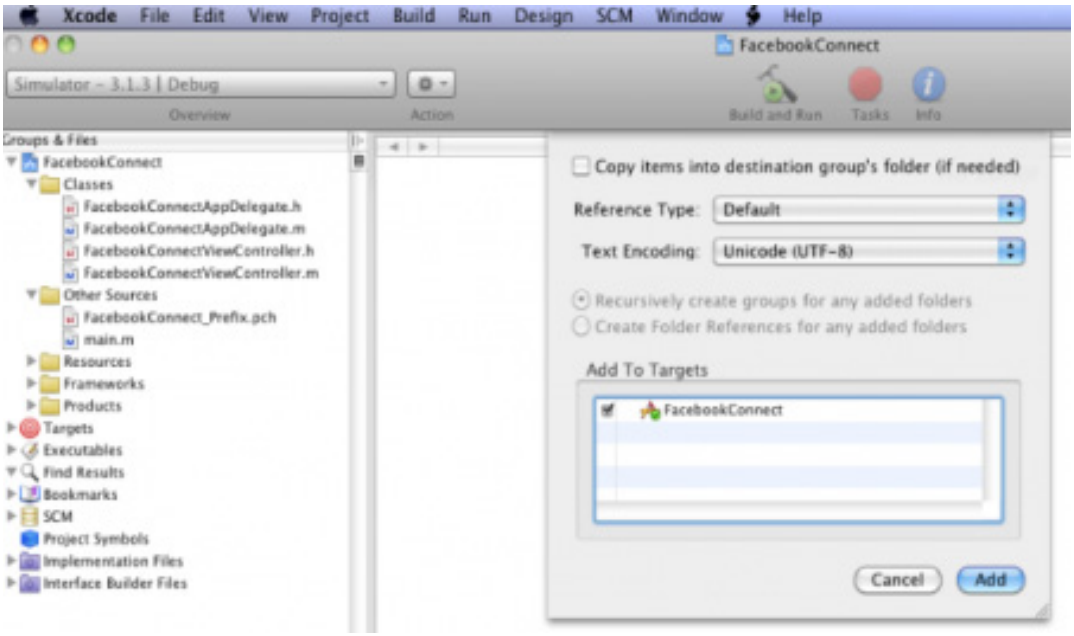
1. Descargar el **SDK de Facebook Connect** desde:
http://www.maestrosdelweb.com/editorial/aprovechando-las-ventajas-de-facebook-connect-sobre-vbulletin/

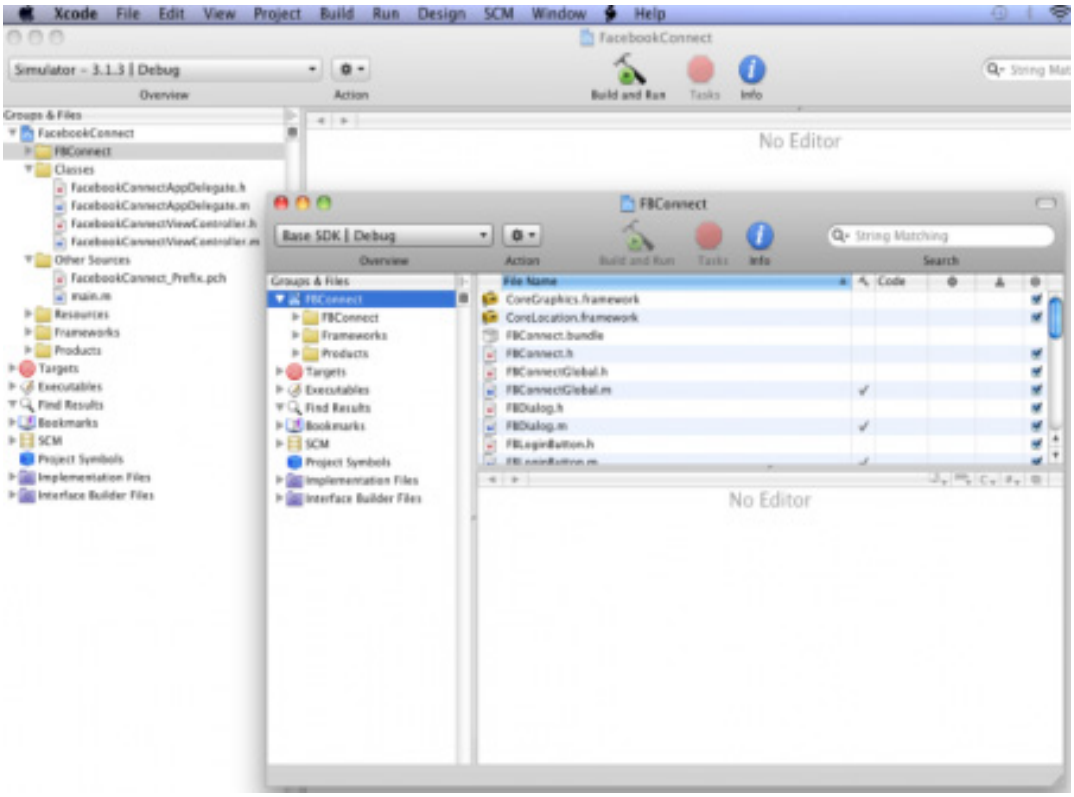
2. Descomprimir y abrir el proyecto **"scr/FBConnect.xcodeproj"**



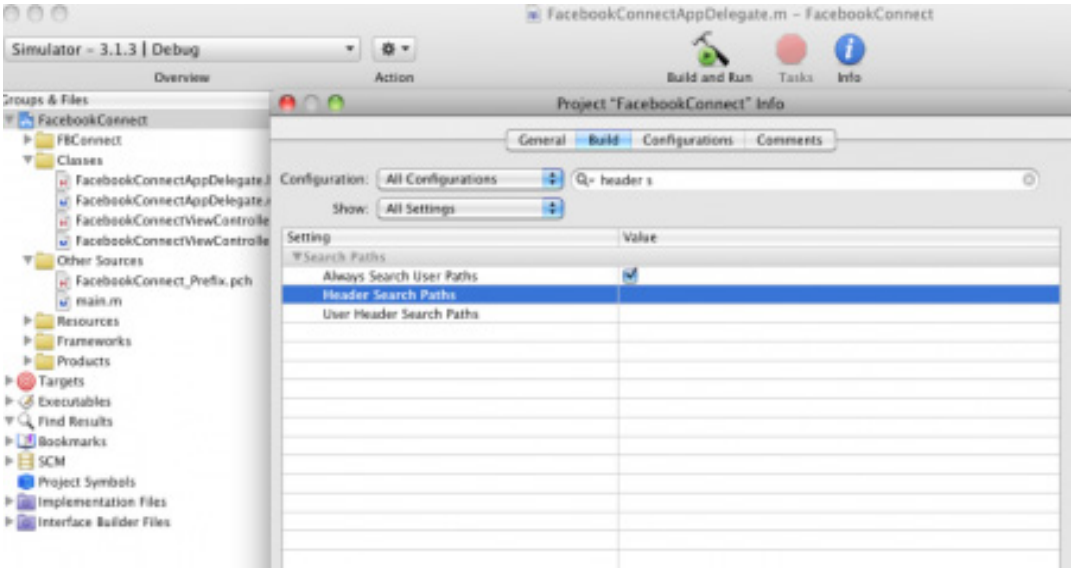
3.Arrastrar el grupo **FBConnect** dentro del proyecto en Xcode:

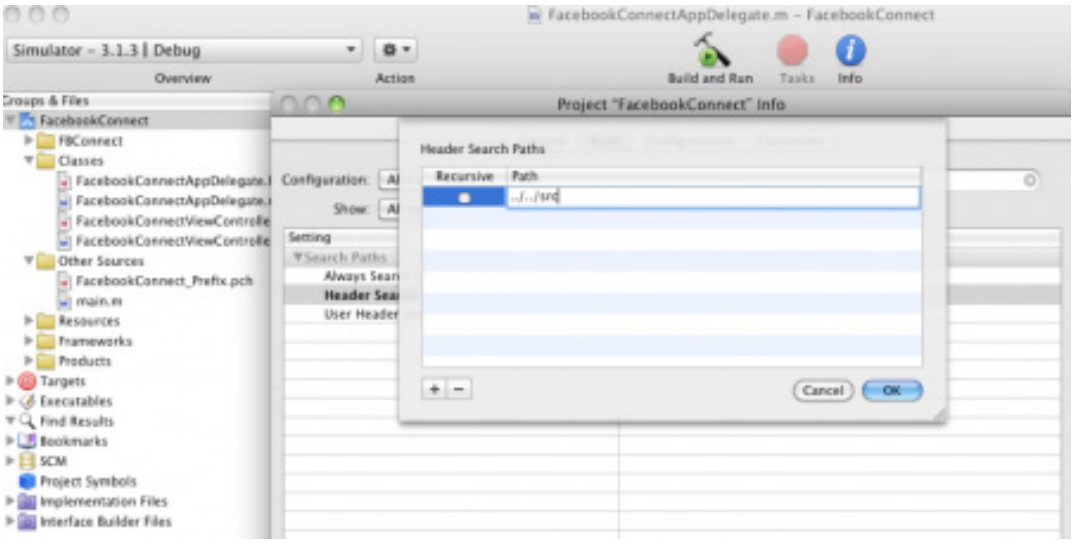






4. Agregar la ruta de **FBConnect** en **"Project->Edit Project Settings":**





5. Importar e implementar las librerías **"FBConnect/FBConnect.h":**

```
#import "FBConnect/FBConnect.h"
- (void)viewDidLoad {
FBSession *session;
session=[FBSession sessionForApplication:@"yourApiKey" secret:@"yourAppSecret" delegate:self];
FBLoginButton* button = [[[FBLoginButton alloc] init] autorelease];
[self.view addSubview:button];
}<
```

6. Agregar el delegate **<FBSessionDelegate>** y la función:

```
"(void)session:(FBSession*)session didLogin:(FBUID)uid"
@interface FacebookConnectViewController : UIViewController  &lt;FBSessionDelegate&gt; {
}
- (void)session:(FBSession*)session didLogin:(FBUID)uid {
NSLog(@"Usuario : %lld conectado.", uid);

}
```

7. Compilar y ejecutar:

Obtendremos un botón que nos permitirá iniciar sesión en Facebook y acceder a la información que necesitemos. La documentación oficial nos permite conocer los principales métodos que podemos utilizar en la implementación. También ofrecen un vídeo tutorial del procedimiento.

Documentación official:
http://wiki.developers.facebook.com/index.php/Facebook_Connect_for_iPhone

Vídeo tutorial: http://vimeo.com/3616452

CAPITULO 7:
Trabajando con el API de twitter

En el capítulo 7 aprenderás a vincular en el proyecto la red social Twitter conocida como uno de los medios de comunicación más populares, con alto alcance y muy utilizado en aplicaciones para dispositivos móviles.

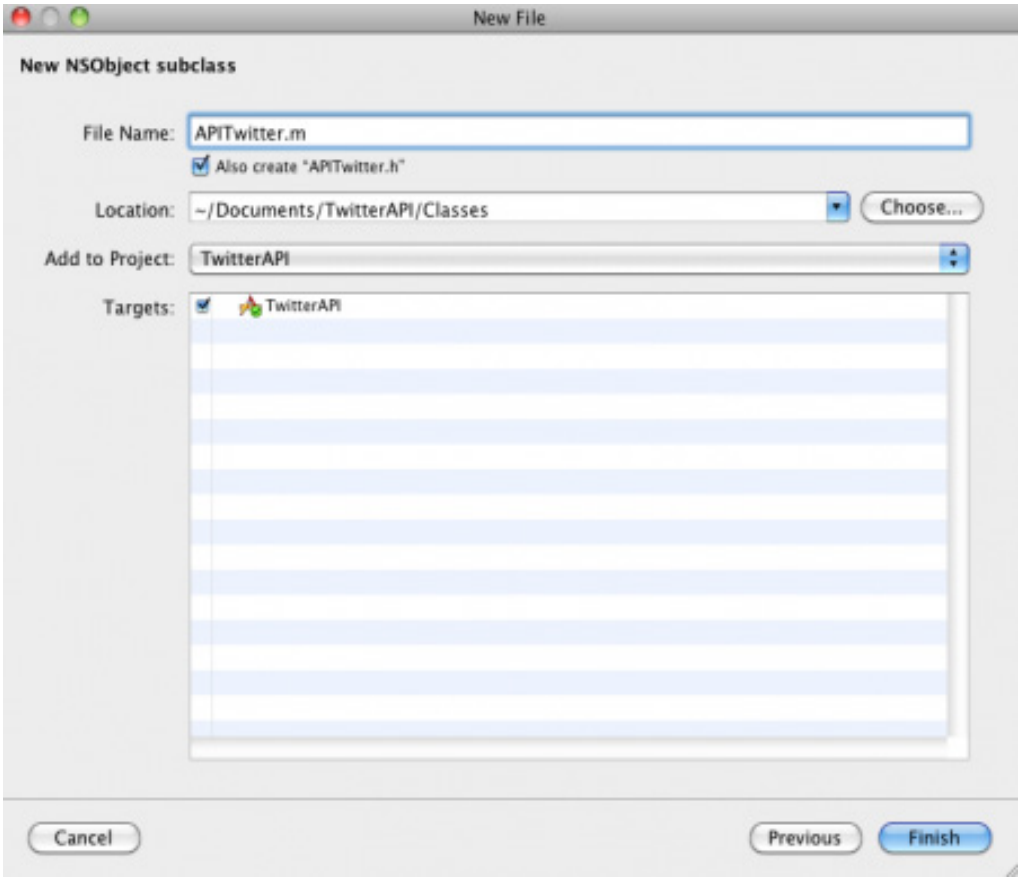
Características de la API de Twitter:
- Limita por día el número de actualizaciones, mensajes directos y solicitudes de "Follow".
- Se basa completamente en HTTP, por lo tanto implementa los métodos GET y POST.
- Intenta conservar los principios de diseño de la Transferencia de Estado Representacional (REST). En la documentación se establecen los formatos disponibles para cada uno de los métodos. Dentro de estos formatos de datos se encuentran: XML, JSON, RSS, ATOM.
- Una línea de comando es todo lo que se requiere para empezar a usar la API. Ofrece un gran número de librerías disponibles para el lenguaje de programación.

Revisar:
http://www.maestrosdelweb.com/editorial/trabajando-con-la-api-de-twitter-desde-php/

Iniciando Twitter en nuestra aplicación:

A continuación implementaremos una clase que nos permita enviar y recibir información a Twitter desde nuestras aplicaciones :

1. Agregamos un nuevo archivo al proyecto (File->New File->NSObject) "APITwitter.h" y definimos las siguientes variables y métodos



```
#import <Foundation/Foundation.h>
@interface APITwitter : NSObject {
NSString    *user;
NSString    *pass;
NSMutableData    *inputInfo;
NSMutableURLRequest    *request;
NSURLConnection    *theConnection;
id    delegate;
SEL    callback;
SEL    errorCallback;
BOOL    isPost;
NSString    *requestBody;
}
@property(nonatomic, retain) NSString    *user;
@property(nonatomic, retain) NSString    *pass;
@property(nonatomic, retain) NSMutableData *inputInfo;

@property(nonatomic, retain) id    delegate;
@property(nonatomic) SEL    callback;
@property(nonatomic) SEL    errorCallback;
-(void)request:(NSURL *)url;
-(void)statuses_update:(NSString *)status delegate:(id)requestDelegate requestSelector:(SEL)re-
questSelector;
-(void)getSearch:(NSString *)status delegate:(id)requestDelegate requestSelector:(SEL)requestSe-
lector;
@end
```

2. Definimos la función para realizar consultas en Twitter:

```
-(void)getSearch:(NSString *)status delegate:(id)requestDelegate requestSelector:(SEL)requestSe-
lector; {

// Define el tipo de método a utilizar GET o POST
isPost = NO;
self.delegate = requestDelegate;
self.callback = requestSelector;

// Cambia los espacios en blanco " " por "+" para la búsqueda
NSString *newStr = [status stringByReplacingOccurrencesOfString:@" " withString:@"+"];

// Define el formato de envio a Twitter de la búsqueda
NSString *searchUrl = [NSString stringWithFormat:@"http://search.twitter.com/search.
atom?q=%@",newStr];
NSURL *url = [NSURL URLWithString:searchUrl];

// Procesa la solicitud
[self request:url];
}
```

3. Definimos la función para enviar actualizaciones a Twitter:

```
-(void)statuses_update:(NSString *)status delegate:(id)requestDelegate requestSelector:(SEL)re-
questSelector; {

// Define el tipo de método a utilizar GET o POST
isPost = YES;
self.delegate = requestDelegate;
self.callback = requestSelector;

// Define el formato de envio a Twitter de la actualización
NSURL *url = [NSURL URLWithString:@"http://twitter.com/statuses/update.xml"];
requestBody = [NSString stringWithFormat:@"status=%@",status];

// Procesa la solicitud
[self request:url];
}
```

4. Definimos la función para conectar con el servidor y procesar las solicitudes:

```
-(void)request:(NSURL *) url {
request = [[NSMutableURLRequest alloc] initWithURL:url];
if(isPost) {
NSLog(@"es Post");
[request setHTTPMethod:@"POST"];
[request setValue:@"application/x-www-form-urlencoded" forHTTPHeaderField:@"Content-
Type"];
[request setHTTPBody:[requestBody dataUsingEncoding:NSASCIIStringEncoding
allowLossyConversion:YES]];
[request setValue:[NSString stringWithFormat:@"%d",[requestBody length] ]
forHTTPHeaderField:@"Content-Length"];
}
theConnection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
if (theConnection) {
inputInfo = [[NSMutableData data] retain];
} else {
NSLog(@"Error en la Conexion");
}
}
```

5. Definimos las funciones restantes para control de la solicitud con el servidor:

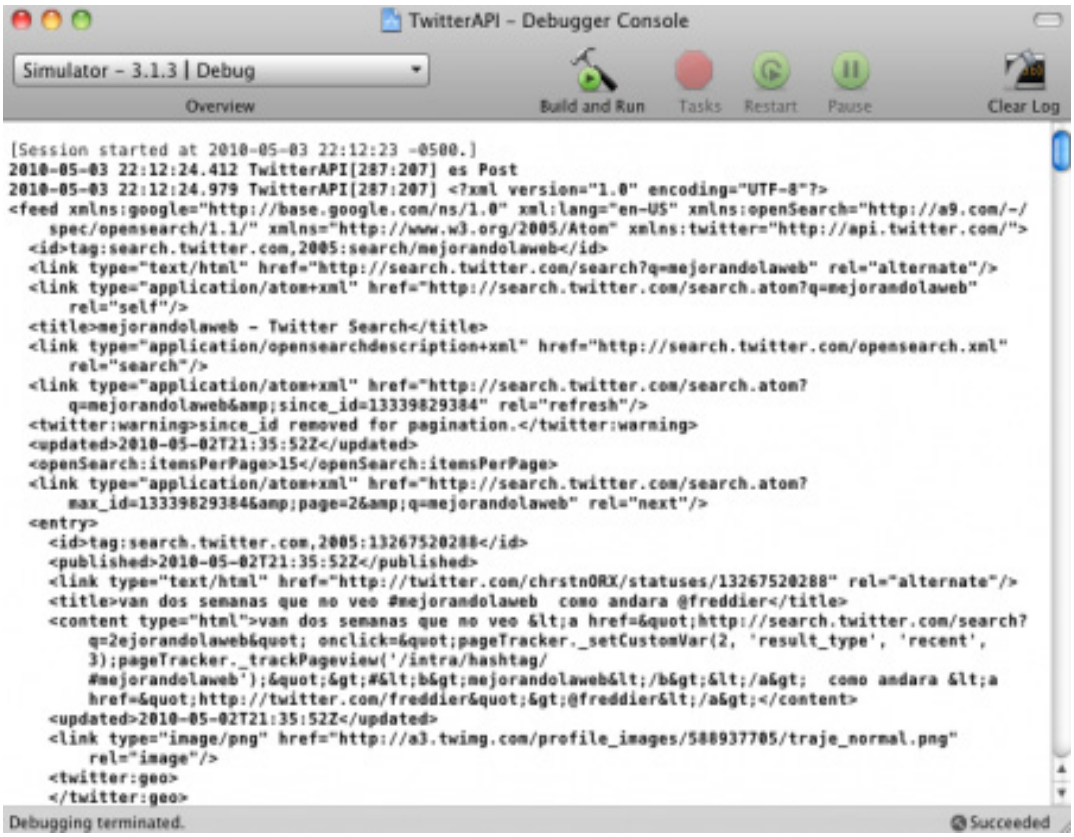
```
- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:(NSURLA
uthenticationChallenge *)challenge {
if ([challenge previousFailureCount] == 0) {
NSURLCredential *newCredential;
newCredential=[NSURLCredential credentialWithUser:[self user]
password:[self pass]
persistence:NSURLCredentialPersistenceNone];
[[challenge sender] useCredential:newCredential
forAuthenticationChallenge:challenge];
} else {
```

```
[[challenge sender] cancelAuthenticationChallenge:challenge];
UIAlertView* alertView = [[UIAlertView alloc] initWithTitle:@"Error API Twitter"
message:@"Usuario y/o Contraseña Incorrecto"
delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil];
[alertView show];
[alertView release];
}
}
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)
response {
[inputInfo setLength:0];
}
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {


NSLog([[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding]);
[inputInfo appendData:data];
}
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error{
[connection release];
[inputInfo release];
[request release];
NSLog(@"Conexion Fallida! Error - %@ %@",
[error localizedDescription],
[[error userInfo] objectForKey:NSErrorFailingURLStringKey]);
if(errorCallback) {
[delegate performSelector:errorCallback withObject:error];
}
}
- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
if(delegate && callback) {
if([delegate respondsToSelector:self.callback]) {
[delegate performSelector:self.callback withObject:inputInfo];
} else {
NSLog(@"No hay respuesta desde delegate");
}
}
[connection release];
[inputInfo release];
[request release];
}
```

6. Implementamos la clase creada “APITwitter.h” en nuestra aplicación:

```
#import “APITwitter.h”
// Método para Actualizar el estado
- (void)updateTwitterAPI{
APITwitter *twitter = [[APITwitter alloc] init];
twitter.user = @”userTwitter”;
twitter.pass = @”passTwitter”;
NSString *textTwitter = @”Prueba API de Twitter en Objective-C”;
[twitter statuses_update:textTwitter delegate:self requestSelector:nil];
[twitter release];
}
// Método para realizar consultas
- (void)searchTwitterAPI{
APITwitter *twitter = [[APITwitter alloc] init];
NSString *textTwitter = @”mejorandolaweb”;
[twitter getSearch:textTwitter delegate:self requestSelector:nil];
[twitter release];
// Los resultados de la consulta se muestran en la consola
}
```

7. Compilamos y ejecutamos:



Si ha salido todo bien, probablemente tengas experiencia con **Objective-C**, porque he saltado algunas cosas para no hacerlo tan extenso. Puedes descargar el proyecto con la clase implementada para hacer **“Build and Go”**

Descargar proyecto:
http://www.maestrosdelweb.com/images/2010/04/TwitterAPI.zip

CAPITULO 8:
Trabajando con el API de OpenFeint

Quienes desarrollan aplicaciones para el iPhone deben conocer la importancia de vincularlas con otros servicios como Facebook y Twitter comunidades virtuales a las que muchos pertenecen. Actualmente, existen para el caso específico de los vídeo juegos en el iPhone tres comunidades que han tomado fuerza:

1. Plus: http://plusplus.com/
2. ScoreLoop: http://www.scoreloop.com/
3. OpenFeint: http://www.openfeint.com/

En este capítulo, veremos como implementar OpenFeint por ser una de las más populares entre los usuarios y por permitir a los desarrolladores independientes acceder fácilmente a su API.

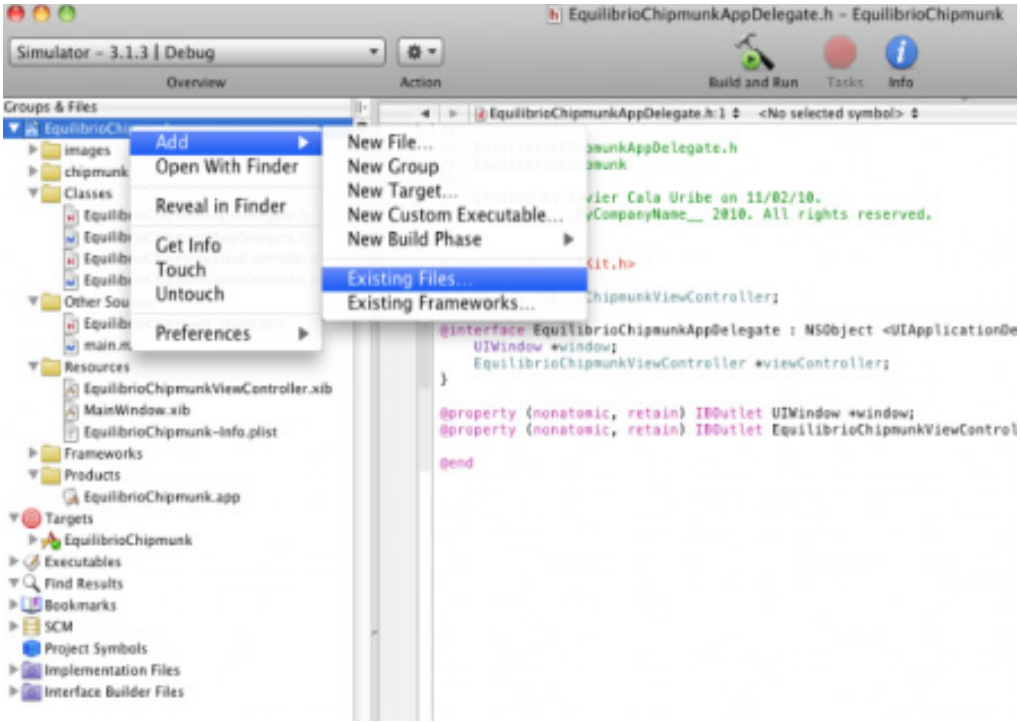
Características de OpenFeint:

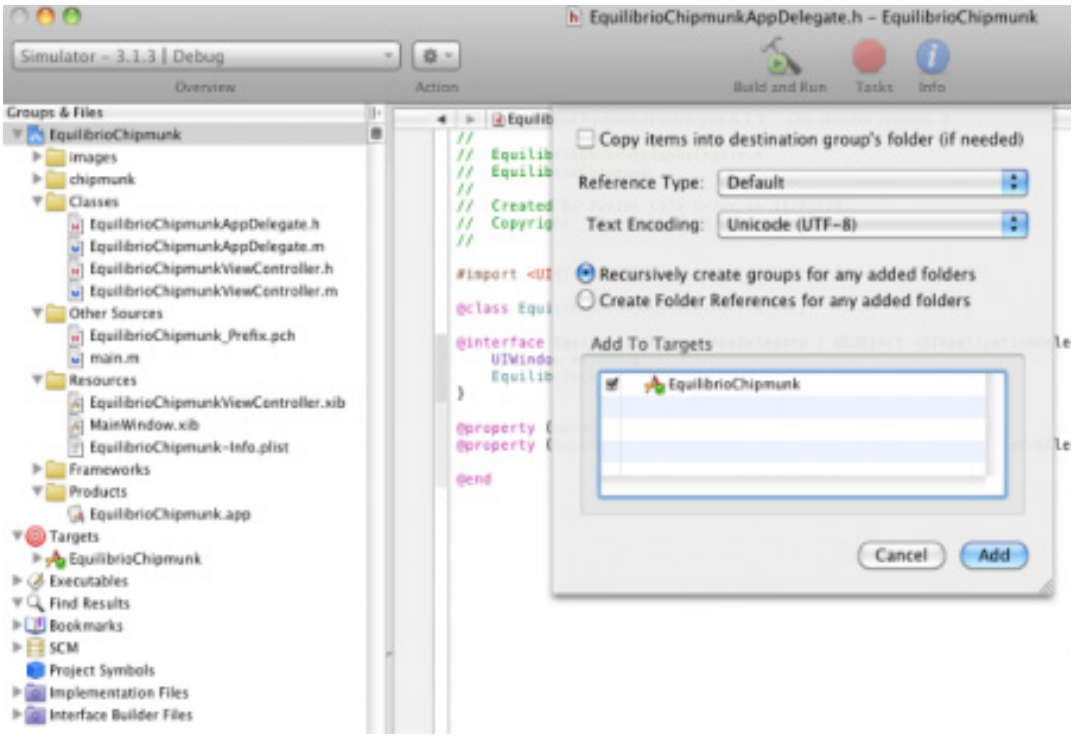
- Contactar con otros jugadores por medio de anuncios, boletines y foros.
- Ver que jugadores están conectados en ese momento.
- Tabla de clasificación con los mejores puntajes y logros alcanzados.
- Geolocalización de la tabla de clasificación integrada con GoogleMaps.
- Ofrece diferentes formas de promocionar nuestras aplicaciones dentro de la comunidad.

Implementando OpenFeint en una aplicación:

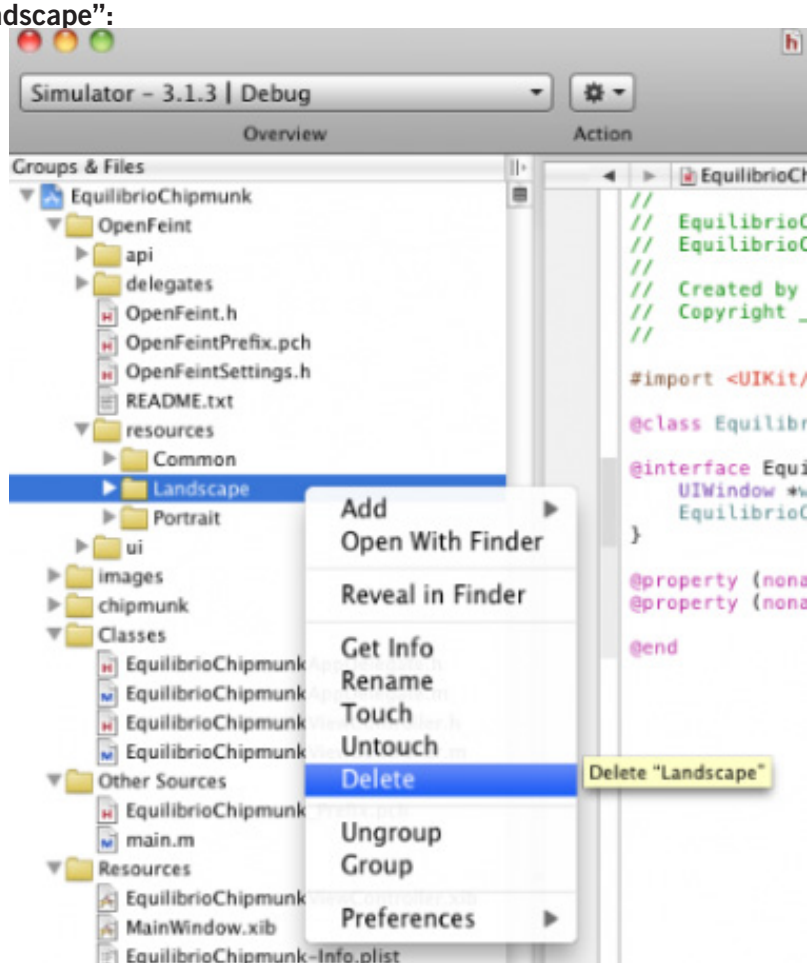
Ahora veremos cómo implementar esta herramienta en nuestra aplicación:

1. Lo primero que debemos hacer es registrarnos en OpenFeint.
2. Una vez registrados, podemos descargar el OpenFeint SDK en la parte superior derecha.
3. Descomprimimos y agregamos la carpeta “OpenFeint” a nuestro proyecto.

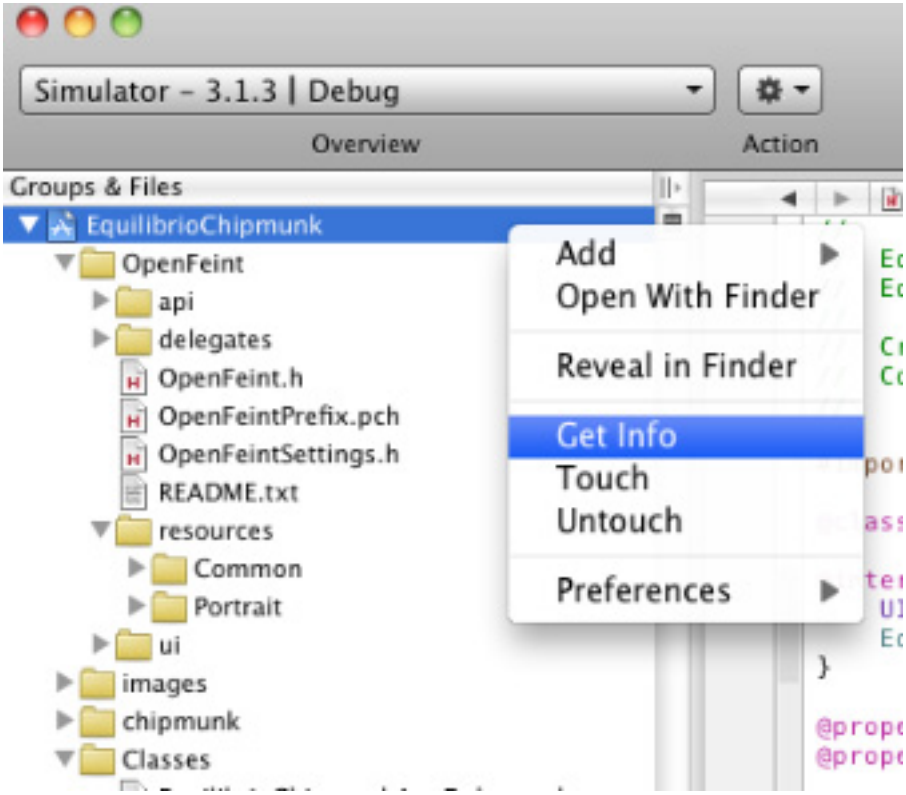




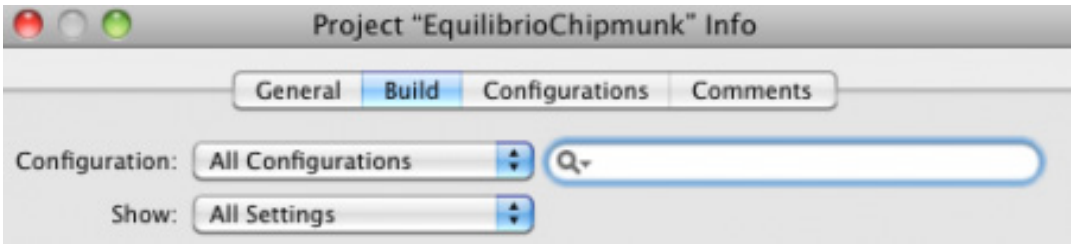
4. Según la vista que utilicemos es recomendable borrar una de las siguientes carpetas de nuestro proyecto: *"Resources/Landscape"* o *"Resources/Portrait"*, para este ejemplo borraremos **"Resources/Landscape":**



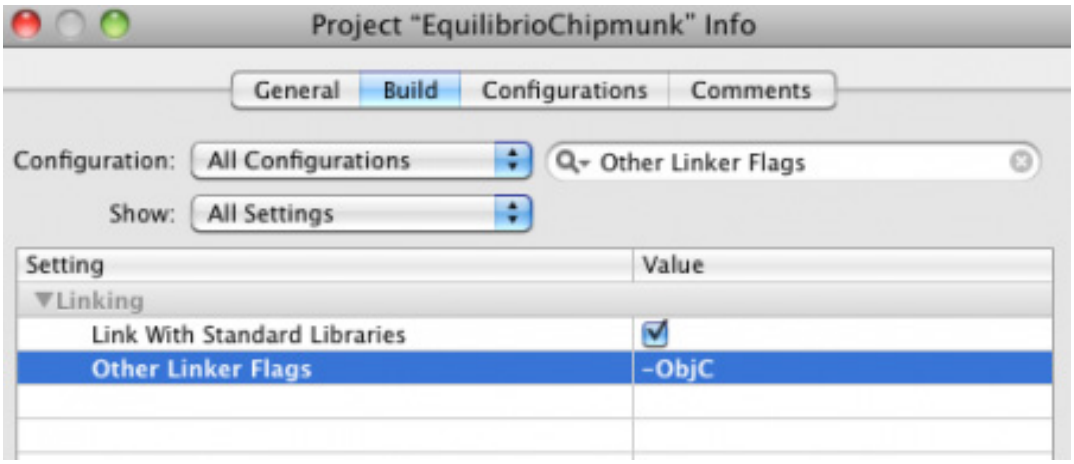
5. Hacemos clic derecho sobre nuestro proyecto en el panel **"Groups & Files"** y seleccionamos **"Get Info":**



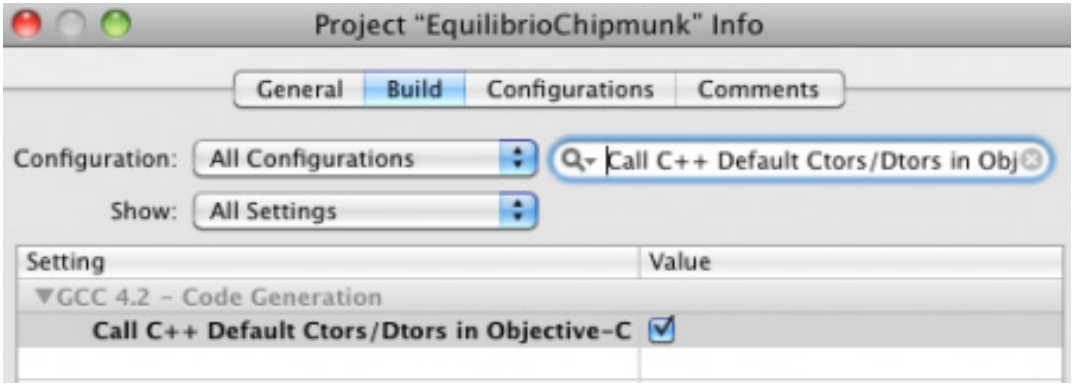
6. Seleccionamos la pestaña **"Build"** y nos aseguramos de tener en **"Configuration" : "All Configurations":**



7. Agregamos en **"Other Linker Flags"** el valor **"-ObjC":**

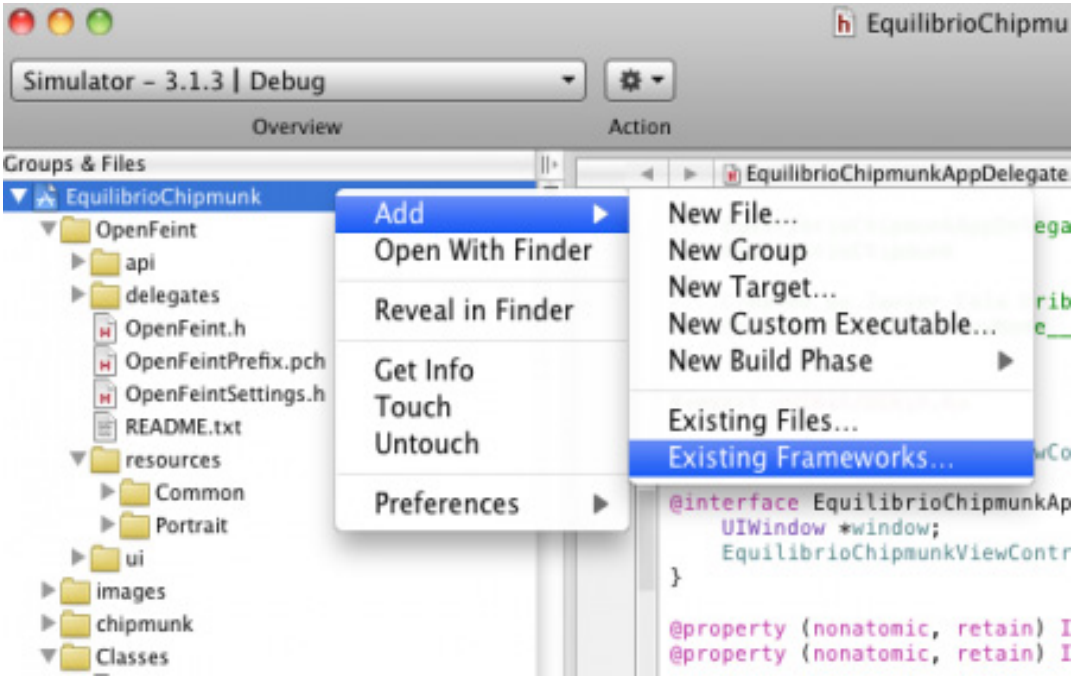


8. Nos aseguramos que esté seleccionado **"Call C++ Default Ctors/Dtors in Objective-C"** en la sección **"GCC 4.2 – Code Generation":**



9. Incluir los siguientes **Frameworks** al proyecto:

- Foundation
- UIKit
- CoreGraphics
- QuartzCore
- Security
- SystemConfiguration
- CFNetwork
- CoreLocation
- MapKit (solo para SDK 3.0 o posterior)
- libsql3.0.dylib (Ubicado en "Carpeta iPhoneSDK"/usr/lib/)



10. Incluir **“OpenFeintPrefix.pch”** en **“NombreProyecto_Prefix.pch”:**

```
#ifdef __OBJC__
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#endif
#import “OpenFeintPrefix.pch”
```

11. Ahora podemos ingresar los datos de nuestra aplicación en la web de OpenFeint para obtener el **“ProductKey”** y el **“ProductSecret”** que nos da acceso a la API.

12. Renombramos el archivo **“NombreProyectoAppDelegate.m”** agregando una **“m”** adicional al final **“NombreProyectoAppDelegate.mm”**

13. Agregamos el archivo de cabecera **“OpenFeint.h”** e iniciamos OpenFeint en nuestra aplicación.

```
>#import “OpenFeint.h”
- (void)applicationDidFinishLaunching:(UIApplication *)application {
[window addSubview:viewController.view];
[window makeKeyAndVisible];
// Inicia OpenFeint
NSDictionary* settings = [NSDictionary dictionaryWithObjectsAndKeys:
[NSNumber numberWithInt:UIInterfaceOrientationPortrait], OpenFeintSettingDashboardOrientation,
@”Nombre App”, OpenFeintSettingShortDisplayName,
[NSNumber numberWithBool:YES], OpenFeintSettingEnablePushNotifications,
[NSNumber numberWithBool:NO], OpenFeintSettingDisableUserGeneratedContent,
[NSNumber numberWithBool:NO], OpenFeintSettingAlwaysAskForApprovalInDebug,
window, OpenFeintSettingPresentationWindow,
nil
];
[OpenFeint initializeWithProductKey:@”yourProductKey”
andSecret:@”yourProductSecret”
andDisplayName:@”Nombre App”
andSettings:settings
andDelegates:nil];
// Muestra Pantalla Principal de OpenFeint
[OpenFeint launchDashboard];
}
```

14. Incluimos estas tres **funciones de control** para OpenFeint:

```
>(void)applicationWillTerminate:(UIApplication *)application {
// Finaliza OpenFeint
[OpenFeint shutdown];
}
(void)applicationDidBecomeActive:(UIApplication *)application{
[OpenFeint applicationDidBecomeActive];
}
(void)applicationWillResignActive:(UIApplication *)application{
[OpenFeint applicationWillResignActive];
}
```

15. Compilar y ejecutar

En estos momentos tenemos implementada la **API de OpenFeint** en nuestro proyecto. Ahora debemos agregar dentro del código fuente las funciones necesarias según los servicios que incorporemos: **Achievements, Leaderboards,** etc.

Documentación oficial: http://www.openfeint.com/developers/support/

Desarrollo de vídeo juegos para dispositivos móviles

La segunda sección de la guía estará dedicada a la creación de vídeo juegos para dispositivos móviles a través del uso de Chipmunk y Cocos2D.

Chipmunk:

- Motor de física 2D – Parte 1
- Motor de física 2D – Parte 2
- Motor de física 2D – Parte 3

Cocos 2D:

- Framework para desarrollar vídeo juegos en 2D – Parte 1
- Framework para desarrollar vídeo juegos en 2D – Parte 2
- Framework para desarrollar vídeo juegos en 2D – Parte 3

CAPITULO 9:
Chipmunk, motor de física 2D

Parte 1

Para los interesados en el desarrollo de vídeojuegos para el iPhone y con poca experiencia en esta área les presento un motor de física 2D sencillo de implementar y de gran ayuda al momento de realizar simulaciones dentro de la aplicaciónChipmunk.

Introducción a Chipmunk:

Un motor de física es un componente software desarrollado principalmente para simular la mecánica newtoniana de objetos modelados dentro de un entorno determinado. El motor considera las variables gravedad, fricción, masa, velocidad, entre otras para simular la física de los objetos de manera aproximada a la física que presentaría ese objeto en el mundo real. También permiten implementar los sistemas de partículas y detectar colisiones, que son de gran ayuda al momento de desarrollar vídeojuegos.

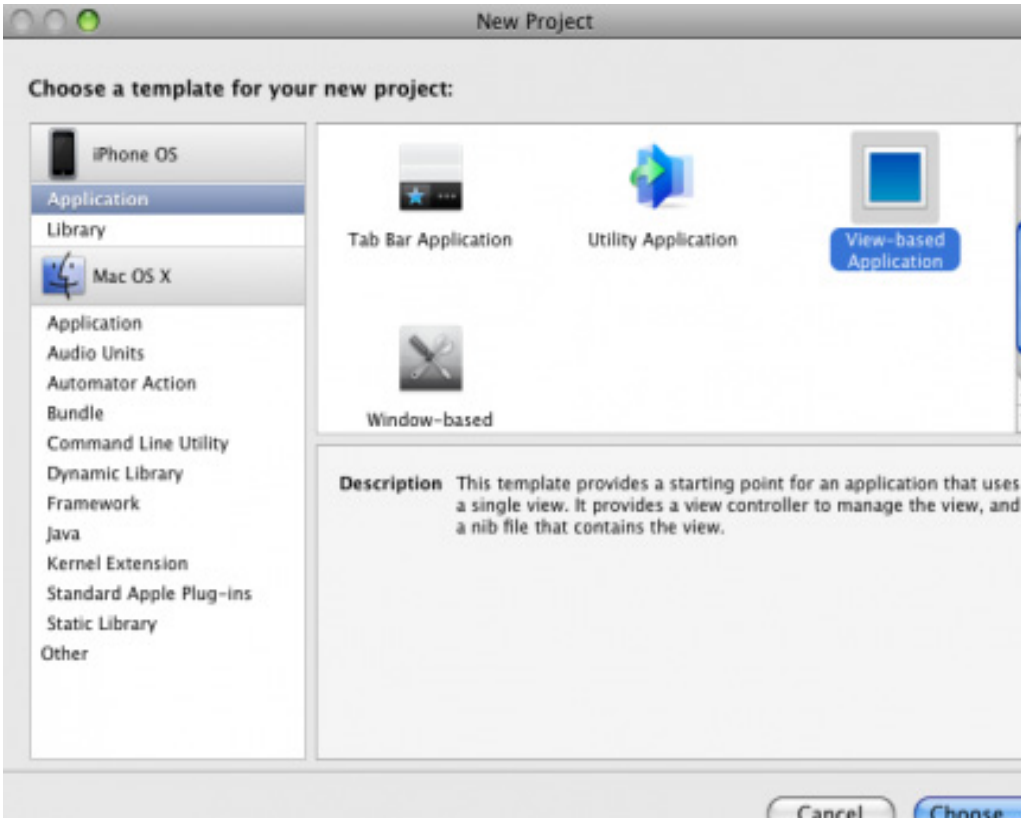
Chipmunk es un motor de física 2D de código abierto desarrollado por Scott Lembcke en "C" bajo licencia MIT. Dentro de sus características están:

- Ideal para el desarrollo de videojuegos en 2D
- Rápido y ligero para modelar cuerpos rígidos
- Flexible sistema de detección de colisiones

Iniciando la implementación con Xcode:

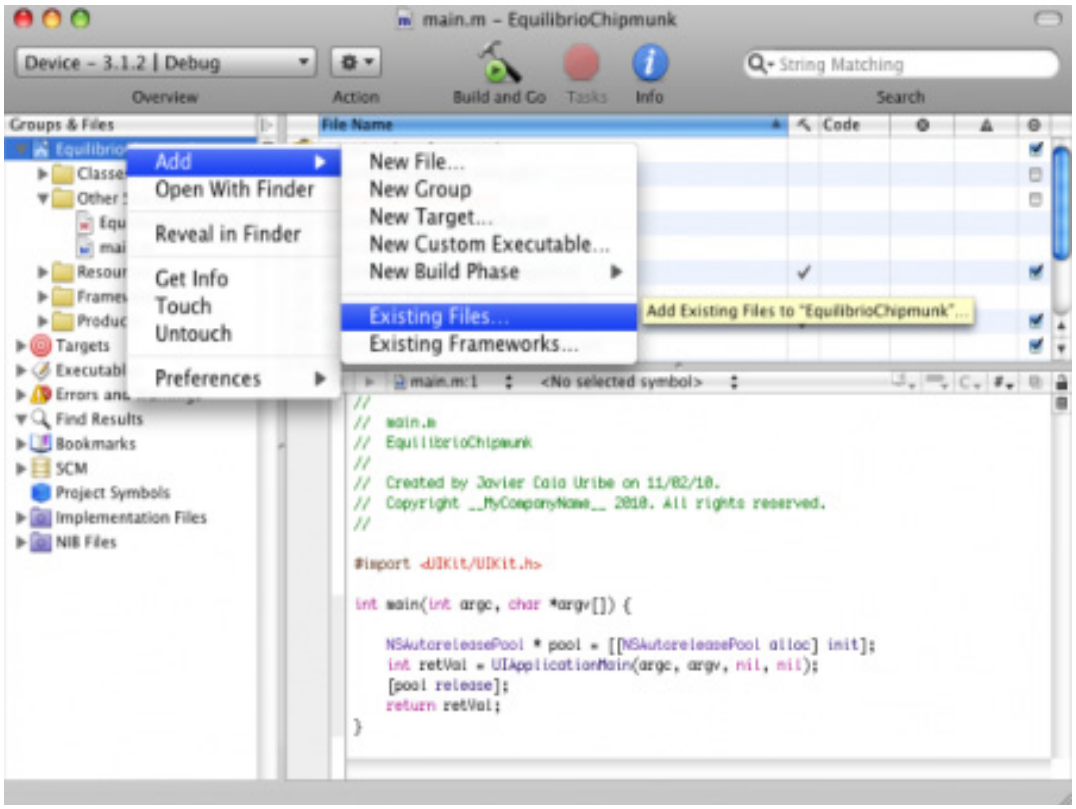
A continuación veremos cómo crear un proyecto en Xcode que incorpore las librerías de Chipmunk.

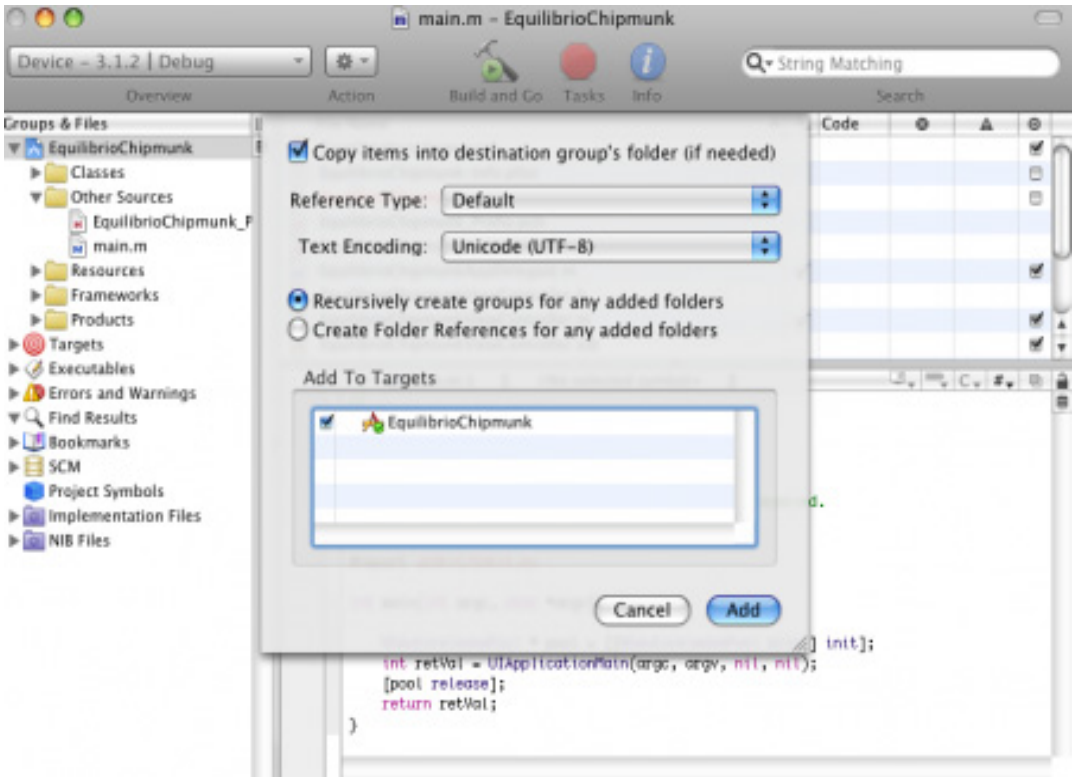
1. Crear un proyecto nuevo en Xcode para iPhone de tipo **"View-Based Application"**



2. Descargar las librerías de Chipmunk disponibles para descargar:
http://www.maestrosdelweb.com/images/2010/04/chipmunk.zip

3. Extraer los archivos y agregar el contenido de la carpeta CHIPMUNK al proyecto





4. Agregar estas dos imágenes al proyecto.

Imagen 1: 

Imagen 2: 

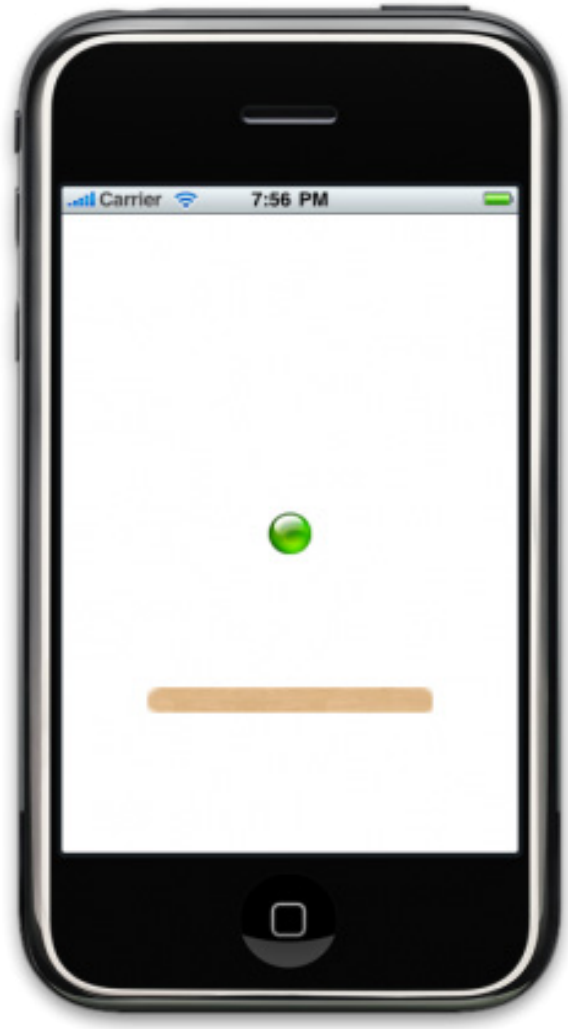
5. Declarar dos variables **UIImageView** en el **"NombreProyectoViewController.h"**

```
#import <UIKit/UIKit.h>
@interface EquilibrioChipmunkViewController : UIViewController {
UIImageView *barra;
UIImageView *esfera;
}
@end
```

6. Definir en la función **(void)viewDidLoad** de **"NombreProyectoViewController.m":**

```
- (void)viewDidLoad {
[super viewDidLoad];
barra = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@&amp;quot;barra.png&amp;quot;]];
barra.center = CGPointMake(160, 350);
esfera = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@&amp;quot;esfera.png&amp;quot;]];
esfera.center = CGPointMake(160, 230);
[self.view addSubview:barra];
[self.view addSubview:esfera];
[self.view setBackgroundColor:[UIColor whiteColor]];
}
```

7. Compilar y ejecutar



Al finalizar los siete pasos descritos, obtendremos una imagen como la anterior.

CAPITULO 10:
Chipmunk, motor de física 2D

Parte 2

Es momento de ver los conceptos básicos del motor de física 2D y su implementación en una aplicación para el iPhone. En el capítulo anterior solo creamos un proyecto en Xcode, agregado dos imágenes y vinculado las librerías de Chipmunk al proyecto, pero no hemos utilizado aún el motor de física.

Conceptos básicos:

Los principales conceptos que debemos tener en cuenta para la implementación del motor de física son:

- **Space:** es la unidad de simulación básica que contiene todos los objetos creados, es el entorno donde interactúan los objetos. En él se establecen las reglas generales que afectan a todos los objetos de la simulación como por ejemplo: la gravedad.
- **Body:** son cuerpos rígidos que contienen las propiedades físicas de un objeto como: masa, posición, rotación, velocidad, etc. No poseen forma (shape) por si mismos y por lo tanto no colisionan con otros cuerpos.
- **Shape:** son las diferentes partes de un cuerpo (body), con los shape le damos forma a los cuerpos permitiendo la colisión entre ellos. Existen tres tipos de shapes: circular, segmentado y poligonal.
- **Constraints:** permiten conectar dos cuerpos de diferentes formas.
- **Forces:** son vectores (x,y) utilizados para interactuar con los objetos creados en el entorno de simulación. No es recomendable modificar directamente las propiedades de los objetos (posición, velocidad, etc), para realizar esto se deben utilizar las funciones que asignan estos vectores a los objetos.

La implementación en Xcode:

Tomando como base el proyecto creado en el capítulo anterior, prosigamos con la implementación de las librerías de Chipmunk:

1. Incluir en **NombreProyetoViewController.h** el archivo de cabecera **"chipmunk.h"**, un objeto de la clase **cpSpace** y las tres funciones que utilizaremos para iniciar e implementar Chipmunk:

```
#import <UIKit/UIKit.h>
#import "chipmunk.h"
@interface EquilibrioChipmunkViewController : UIViewController {
UIImageView *barra;
UIImageView *esfera;
cpSpace *space;
}
- (void)configurarChipmunk;
- (void)delta:(NSTimer *)timer;
void updateShape(void *ptr, void* unused);
@end
```

2. Realizar el llamado de **“(void)configurarChipmunk”** en la función **“(void)viewDidLoad”:**

```
- (void)viewDidLoad {
[super viewDidLoad];
barra = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@”barra.png”]];
barra.center = CGPointMake(160, 350);
esfera = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@”esfera.png”]];
esfera.center = CGPointMake(160, 230);
[self.view addSubview:barra];
[self.view addSubview:esfera];
[self.view setBackgroundColor:[UIColor whiteColor]];
[self configurarChipmunk];
}
```

3. Implementar las tres funciones declaradas anteriormente:

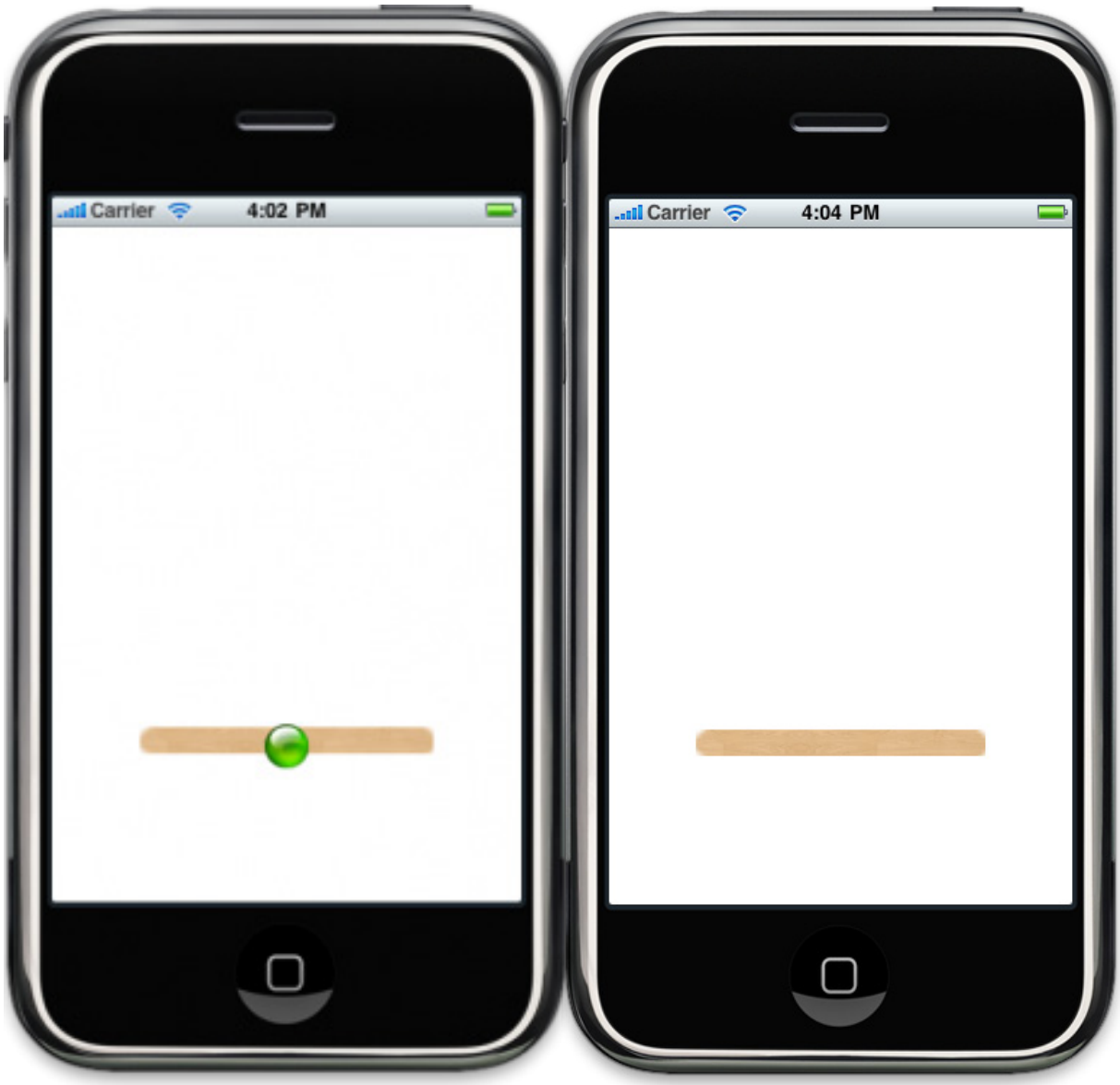
```
- (void)configurarChipmunk {

// Inicia el motor de fisica 2D Chipmunk
cpInitChipmunk();
// Crea un nuevo Space
space = cpSpaceNew();
// Define la dirección y magnitud de la gravedad en el Space
space->gravity = cpv(0, -100);
// Implementa el NSTimer encargado de realizar las animaciones
[NSTimer scheduledTimerWithTimeInterval:1.0f/60.0f target:self selector:@selector(delta:)
userInfo:nil repeats:YES];
}
- (void)delta:(NSTimer *)timer {
// Actualiza información del Space
cpSpaceStep(space, 1.0f/60.0f);
// Actualiza información de los Shapes definidos
cpSpaceHashEach(space->activeShapes, &updateShape, nil);
}
void updateShape(void *ptr, void* unused) {
cpShape *shape = (cpShape*)ptr;
// Validación del Shape recibido
if(shape == nil || shape->body == nil || shape->data == nil) {
NSLog(@”Invalido shape revisar ...”);
return;
}
// Actualiza la posición del Shape
if([(UIView *)shape->data isKindOfClass:[UIView class]]) {
[(UIView *)shape->data setCenter:CGPointMake(shape->body->p.x, 480 - shape->body->p.y)];
}
else
NSLog(@”Shape actualizado fuera de la función : updateShape “);
}
```

4. Definir *Body* y *Shape* de un objeto dinámico circular en la función:

"(void)configurarChipmunk"

```
- (void)configurarChipmunk {

// Inicia el motor de fisica 2D Chipmunk
cpInitChipmunk();

// Crea un nuevo Space
space = cpSpaceNew();

// Define la dirección y magnitud de la gravedad en el Space
space->gravity = cpv(0, -100);

// Implementa el NSTimer encargado de realizar las animaciones
[NSTimer scheduledTimerWithTimeInterval:1.0f/60.0f target:self selector:@selector(delta:)
userInfo:nil repeats:YES];

// Crea un Body con masa 50 y momento INFINITY
cpBody *esferaBody = cpBodyNew(50.0f, INFINITY);

// Establece posición inicial
esferaBody->p = cpv(160, 250);

// Agrega el Body al Space
cpSpaceAddBody(space, esferaBody);

// Crea un Shape tipo Circle con radio 15 asociado al Body "esferaBody"
cpShape *esferaShape = cpCircleShapeNew(esferaBody, 15.0f, cpvzero);
esferaShape->e = 0.5f; // Elasticidad
esferaShape->u = 0.8f; // Fricción
esferaShape->data = esfera; // Asocia Shape con UIImageView
esferaShape->collision_type = 1; // Las colisiones son agrupadas por tipo

// Agrega el Shape al Space
cpSpaceAddShape(space, esferaShape);
}
```

5. Compilar y ejecutar



Si hemos hecho todo bien al final se verá mal, la esfera caerá en dirección de la gravedad definida y no encontrará obstáculos en el espacio.

CAPITULO 11:
Chipmunk, motor de física 2D y el acelerómetro

Parte 3

En este capítulo finalizaremos el tema sobre el motor de física 2D, Chipmunk. Hasta el momento hemos visto los conceptor básicos (Parte 1) y los elementos dinámicos presentes en las simulaciones (parte 2)_. En este ultimo capítulo revisaremos los elementos estáticos y la implementación del acelerómetro con nuestra aplicación.

Para continuar explorando sobre Chipmunk recomiendo revisar la documentación disponible y seguir probando el desempeño de las simulaciones en los dispositivos móviles (iPhone / iPod Touch/ iPad) para lograr una óptima implementación del motor de física 2D.

Documentación Chipmunk:
http://code.google.com/p/chipmunk-physics/

Conceptos:

Antes de implementar los elementos estáticos en nuestra aplicación es importante recordar los tipos de *Shapes* que existen en Chipmunk y aclarar uno de ellos:

- **Circular:** Genera una circunferencia para el cuerpo rígido referenciado cpCircleShapeNew(cpBody * body, cpFloat radius, cpVect offset).
- **Segmentado:** Genera una forma lineal entre los puntos a y b. cpSegmentShapeNew(cpBody * body, cpVect a, cpVect b, cpFloat radius).
- **Poligonal:** Genera un polígono Convexo definido previamente por medio de sus vértices. En Chipmunk no se pueden modelar polígonos Concavos, para esos casos se debe utilizar Segment. cpPolyShapeNew(cpBody * body, int numVerts, cpVect * verts, cpVect offset).

Finalizando la implementación en Xcode

Prosigamos con el proyecto creado para estos menesteres en el capítulo anterior:

1. Definir *Body* y *Shape* de un objeto estático lineal en la función:

```
 "(void)configurarChipmunk"
- (void)configurarChipmunk {
// Inicia el motor de fisica 2D Chipmunk
cpInitChipmunk();
// Crea un nuevo Space
space = cpSpaceNew();
// Define la dirección y magnitud de la gravedad en el Space
space->gravity = cpv(0, -100);
// Implementa el NSTimer encargado de realizar las animaciones
[NSTimer scheduledTimerWithTimeInterval:1.0f/60.0f target:self selector:@selector(delta:)
userInfo:nil repeats:YES];
// Crea un Body con masa 50 y momento INFINITY
cpBody *esferaBody = cpBodyNew(50.0f, INFINITY);
// Establece posición inicial
esferaBody->p = cpv(160, 250);
// Agrega el Body al Space
cpSpaceAddBody(space, esferaBody);
// Crea un Shape tipo Circle con radio 15 asociado al Body "esferaBody"
cpShape *esferaShape = cpCircleShapeNew(esferaBody, 15.0f, cpvzero);
esferaShape->e = 0.5f; // Elasticidad
esferaShape->u = 0.8f; // Fricción
esferaShape->data = esfera; // Asocia Shape con UIImageView
esferaShape->collision_type = 1; // Las colisiones son agrupadas por tipo
// Agrega el Shape al Space
cpSpaceAddShape(space, esferaShape);
// ==============================
// Implementación Objeto Estático
// ==============================
// Crea un Body con masa y momento INFINITY
barraBody = cpBodyNew(INFINITY, INFINITY);
// Establece la posición inicial
barraBody->p = cpv(160,120);
// El Body de los objetos estáticos no debe agregarse al Space
// Si quieres saber qué pasa, agregalo <img src="http://www.maestrosdelweb.com/wp-includes/
images/smilies/icon_wink.gif" alt=";)" class="wp-smiley">
// Crea un Shape tipo Segment asociado al Body barraBody
barraShape = cpSegmentShapeNew(barraBody, cpv(-105, -3), cpv(105, -3), 10.0);
barraShape->e = 0.7f; // Elasticidad
barraShape->u = 0.4f; // Fricción
barraShape->data = barra; // Asocia Shape con UIImageView
barraShape->collision_type = 0; // Se agrupa en un tipo de colisión diferente a "esferaShape"
// Agrega el Shape al Space
```

```
cpSpaceAddShape(space, barraShape);
}
```

2. Agregar **“<UIAccelerometerDelegate>”** en **“NombreProyectoViewController.h”**

```
#import <UIKit/UIKit.h>
#import “chipmunk.h”
@interface EquilibrioChipmunkViewController : UIViewController &lt;UIAccelerometerDelegate&
gt;{
UIImageView *barra;
UIImageView *esfera;
cpSpace *space;
cpBody *barraBody;
cpShape *barraShape;
}
- (void)configurarChipmunk;
- (void)delta:(NSTimer *)timer;
void updateShape(void *ptr, void* unused);
@end
```

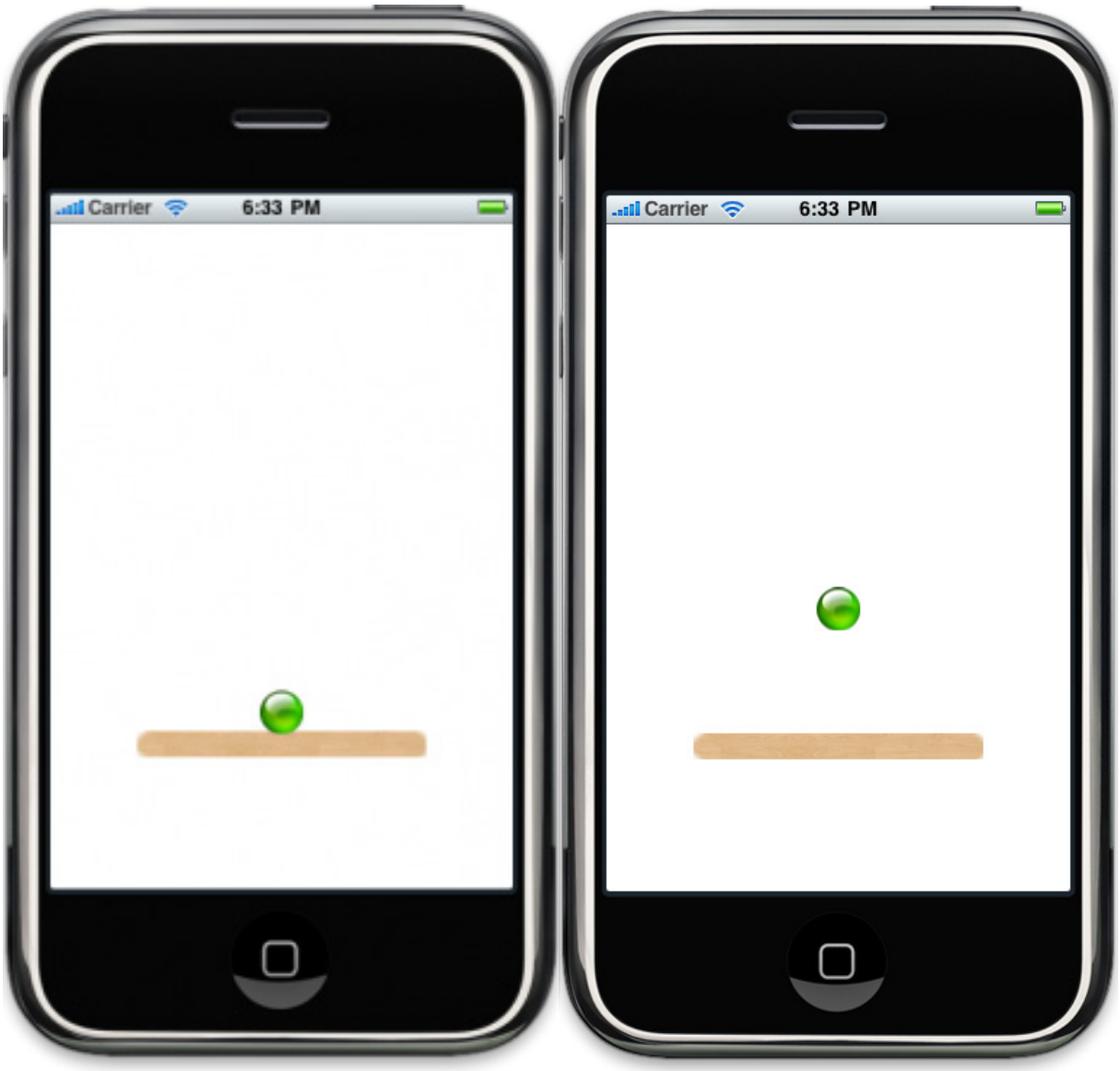
3. Definir la función que implementa el acelerómetro:

```
- (void)accelerometer:(UIAccelerometer*)accelerometer didAccelerate:(UIAcceleration*)accelera-
tion{
// Modifica el ángulo de la Barra con relación a la inclinación del dispositivo
cpBodySetAngle(barraBody, M_PI * (acceleration.x * 0.6f) );
// Notifica a Chipmunk el movimiento de un Objeto Estático
cpSpaceRehashStatic(space);
// Actualiza el Shape de la Barra (Objeto Estático)
[(UIView *)barraShape-&gt;data setTransform:
CGAffineTransformMakeRotation(-barraShape-&gt;body-&gt;a)];
}
```

4. Inicializamos el acelerómetro en la función:

```
“(void)viewDidLoad”
- (void)viewDidLoad {
[super viewDidLoad];
barra = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@”barra.png”]];
barra.center = CGPointMake(160, 350);
esfera = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@”esfera.png”]];
esfera.center = CGPointMake(160, 230);
[self.view addSubview:barra];
[self.view addSubview:esfera];
[self.view setBackgroundColor:[UIColor whiteColor]];
[self configurarChipmunk];
// Inicia el acelerómetro
[[UIAccelerometer sharedAccelerometer] setUpdateInterval:(1.0 / 60)];
[[UIAccelerometer sharedAccelerometer] setDelegate:self];
}
```

5. Compilar y ejecutar



Si hemos hecho todo bien al final obtendremos un objeto circular que se sostiene con un madero el cual gira con el movimiento del dispositivo. Se puede seguir probando con distintos valores de masa, fricción, elasticidad, gravedad, entro otros para conocer un poco más el comportamiento de los objetos dentro del espacio creado.

CAPITULO 12:
Cocos2D, Framework para desarrollar vídeojuegos en 2D (Parte 1)

Para el desarrollo de vídeojuegos en la plataforma iPhone OS (iPhone/iPod Touch/iPad) es
importante conocer las distintas herramientas disponibles y evaluar cuál de ellas se ajusta más a nuestras necesidades. Varias aplicaciones actualmente son desarrolladas utilizando uno de estos componentes:

- **OpenGL:**
  http://www.khronos.org/opengles/
- **UIKit:**
  http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIKit_Framework/_index.html
- **Core Animation + Quartz 2D:**
  http://developer.apple.com/technologies/ios/graphics-and-animation.html

UIKit es el más sencillo de utilizar en comparación con OpenGL ES que requiere de un mayor tiempo de implementación pero ofrece el mejor rendimiento. La buena noticia es que existe un framework que simplifica esta implementación: Cocos2D.

Conociendo Cocos2D:

Cocos2D es un framework para el desarrollo de juegos en 2D y aplicaciones con alto contenido interactivo. Cocos2D para iPhone es basado en Cocos2D pero implementaObjetive-C como lenguaje de programación en lugar de Python. Dentro de las características principales de este framework encontramos:

- Integración con motores de física: Box2D, Chipmunk.
- Manejo de Escenas y efectos de transiciones.
- Compatibilidad con eventos Touch y el acelerómetro.
- Texturas PVRTC de 2-bit y 4-bit, texturas RGBA de 16 bits y 32-bit.
- Basado en OpenGL ES 1.1

Dentro de Cocos2D encontramos tres conceptos principales a considerar: escenas, capas y objetos. Las *escenas* equivalen a los niveles o vistas implementadas en un juego. Las *capas* se incorporan a las escenas y pueden contener uno o más *objetos*: menús, botones, etiquetas, cuerpos, etc. A su vez una escena puede contener una o más capas.

Implementación de Cocos2D en una aplicación:

1. Descarga las liberías de Cocos 2D para iPhone: http://code.google.com/p/cocos2d-iphone/
2. Ejecutar el siguiente comando en la consola (dentro de la carpeta descargada): ./install_template.sh
3. Crear un nuevo proyecto en Xcode de tipo cocos2d-0.99.1 Application

4. Compilar y ejecutar:

La plantilla creada nos muestra los principales elementos de una aplicación:

- **CCDirector:** Es el controlador principal de nuestra aplicación.
  [CCDirector setDirectorType:CCDirectorTypeDefault];

- **CCScene:** Implementa las escenas o vistas
  CCScene *scene = [CCScene node];

- **CCLayer:** Permite la creación de capas (HelloWorld es un objeto tipo CCLayer)
  HelloWorld *layer = [HelloWorld node];

- **CCLabel:** Uno de los tipos de objetos que se pueden implementar dentro de una capa
  CCLabel* label = [CCLabel labelWithString:@"Hola Mundo"
  fontName:@"Marker Felt" fontSize:64];

Tenemos implementadas las librerías de Cocos2D en nuestro proyecto de Xcode. Ahora podemos utilizar los diferentes componentes que nos brindan estas librerías y lograr una mejora en el desempeño de nuestras aplicaciones.

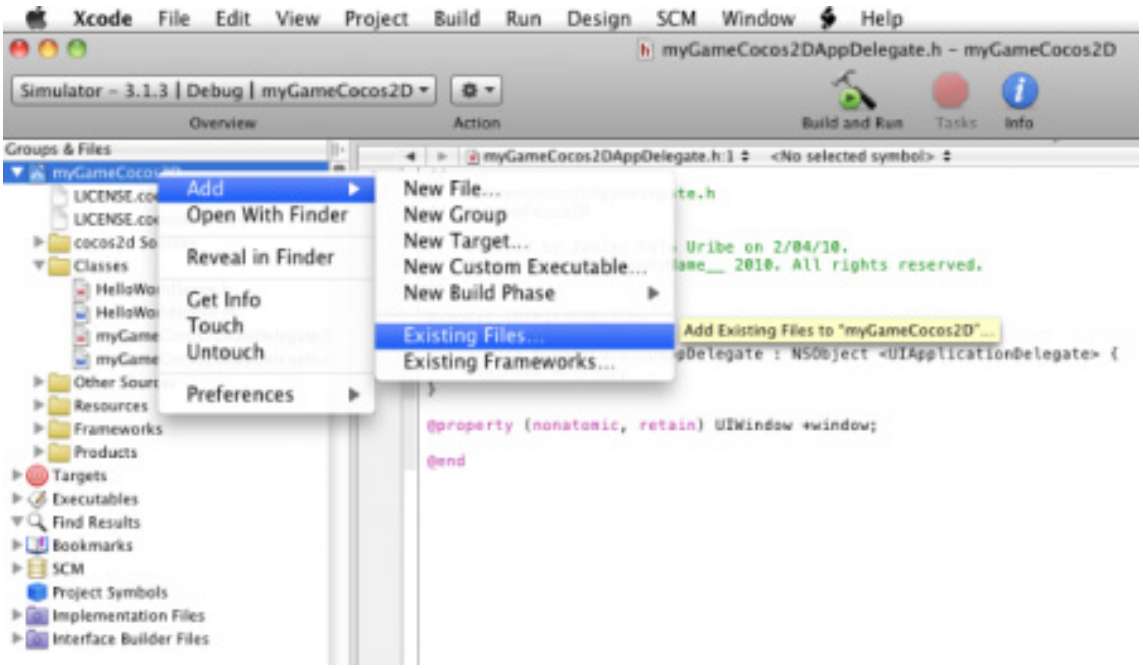
CAPITULO 13:
Cocos2D, Framework para desarrollar vídeojuegos en 2D (Parte 2)

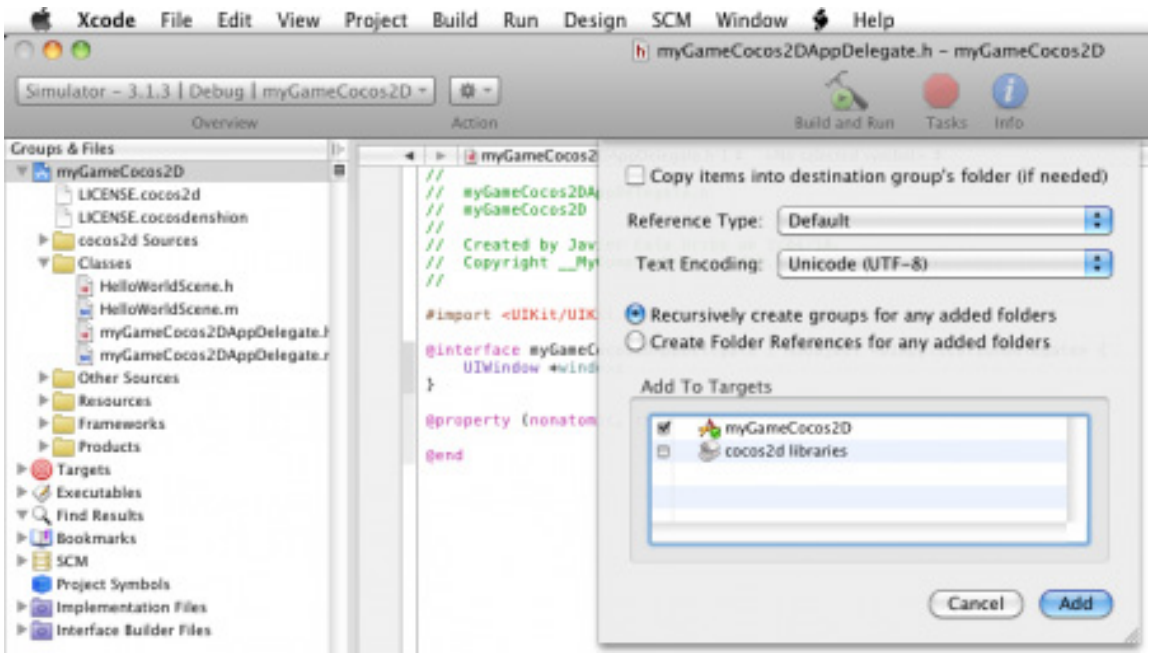
Prosiguiendo con Cocos2D el framework basado en OpenGL ES para el desarrollo de juegos en 2D en la plataforma de iPhone OS, continuaremos con la implementación de objetos y animaciones en el proyecto creado anteriormente.

Antes recordemos que la animación es un proceso para generar la sensación de movimiento de un objeto por medio de la superposición de varias imágenes. Cada una de las imágenes que conforman una animación se denomina fotograma o *frame*, para las animaciones en el iPhone con Cocos2D, dependiendo de los objetos creados, se establece un número predeterminado de 60 fotogramas por segundo.

Implementación de Cocos2D:

Ahora vamos a proseguir con el proyecto creado en el capítulo anterior. Pero antes debemos agregar esta imagen en nuestro proyecto:





1. Redefinimos la clase HelloWorld de CCLayer a CCColorLayer, para modificar de manera fácil el color de fondo

```
#import “cocos2d.h”
@interface HelloWorld : CCColorLayer // Antes CCLayer
{
}
+(id) scene;
@end
```

2. Agregamos el objeto **CCSprite** a la capa **HelloWorld**

```
-(id) init{
   // ccc4 Estable el color de Fondo
   if( (self=[super initWithColor:ccc4(255,255,255,255)] )) {

        CGSize winSize = [[CCDirector sharedDirector] winSize];
        // Crea un objeto tipo CCSprite que contiene la imagen agregada
        CCSprite *player = [CCSprite spriteWithFile:@"giftOne.png"
        rect:CGRectMake(0, 0, 85, 121)];
        // Establece la posición del objeto
        player.position = ccp(player.contentSize.width/2, winSize.height/2);
        // Agrega el objeto a la Capa
        [self addChild:player];
   }
   return self;
}
```

3. Compilar y ejecutar:

4. Ahora agregaremos movimiento a nuestro objeto con el siguiente método:

```
-(void)moveObject {
   // Acción girar
   id rotateAction = [CCRotateBy actionWithDuration:2 angle:180*2];
   // Acción saltar
   id jumpAction = [CCJumpBy actionWithDuration:2 position:ccp(260,0) height:50 jumps:2];
   // Agrupa 2 o más acciones
   id fordward = [CCSpawn actions:rotateAction, jumpAction, nil];
   // Todas las acciones permiten regresar la acción con el método "reverse"
```

```
    id backwards = [fordward reverse];
    // Permite ejecutar una acción despúes de otra
    id sequence = [CCSequence actions: fordward, backwards, nil];
    // Repite una acción el número de veces que se requiera o
    // hasta el infinito y más allá con "RepearForEver"
    id repeat = [CCRepeat actionWithAction:sequence times:2];
    // Ejecuta la acción sobre el objeto
    [player runAction:repeat];
}
```

5. Realizamos el llamado a la función definida anteriormente:

```
// Método que implementa el evento Touch en Cocos2D
- (void)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    [self moveObject];
}
```

6. Habilitamos el evento Touch en (id)init

```
self.isTouchEnabled = YES;
```

7. Compilar y ejecutar:

Si todo ha salido bien obtendremos un objeto que gira y salta al momento de tocar la pantalla. Puedes descargar el proyecto para realizar el procedimiento **"Build and Go".**

Descargar el proyecto:
http://www.maestrosdelweb.com/images/2010/04/myGameCocos2D.zip

CAPITULO 14:
Cocos2D, Framework para desarrollar vídeojuegos en 2D (Parte 3)

Para finalizar con los capítulos sobre Cocos2D revisaremos la librería que implementa los sonidos en una aplicación para el iPhone. Si bien podemos utilizar directamente el framework disponible para la plataforma: Core Audio y OpenAL cocos2D nos facilita la implementación de varios archivos de audio en nuestro proyecto.

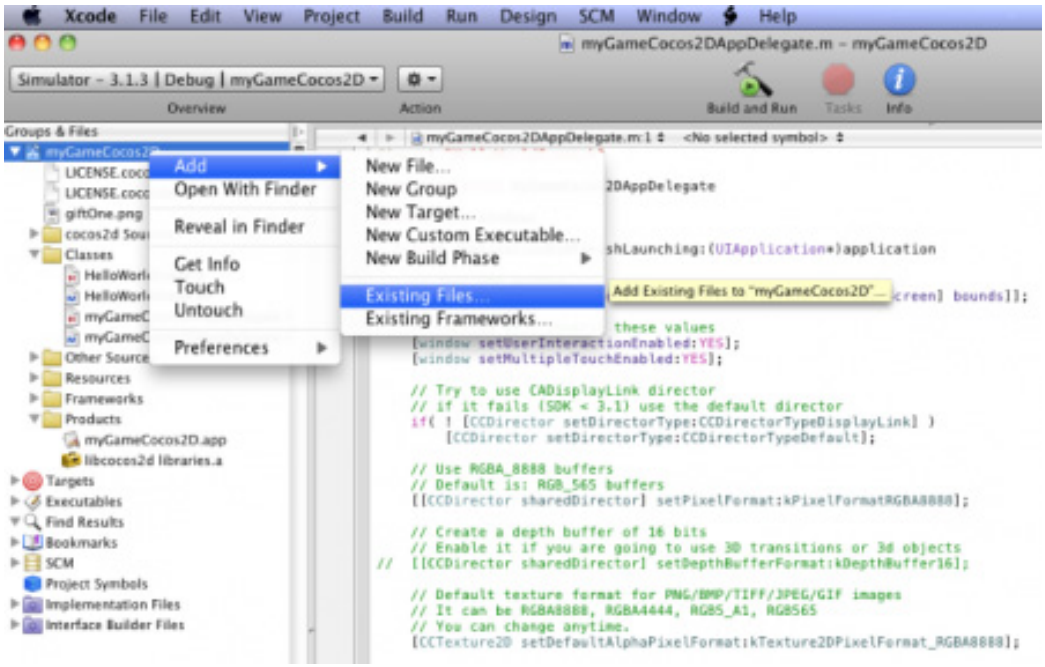
Librerías de audio disponibles:

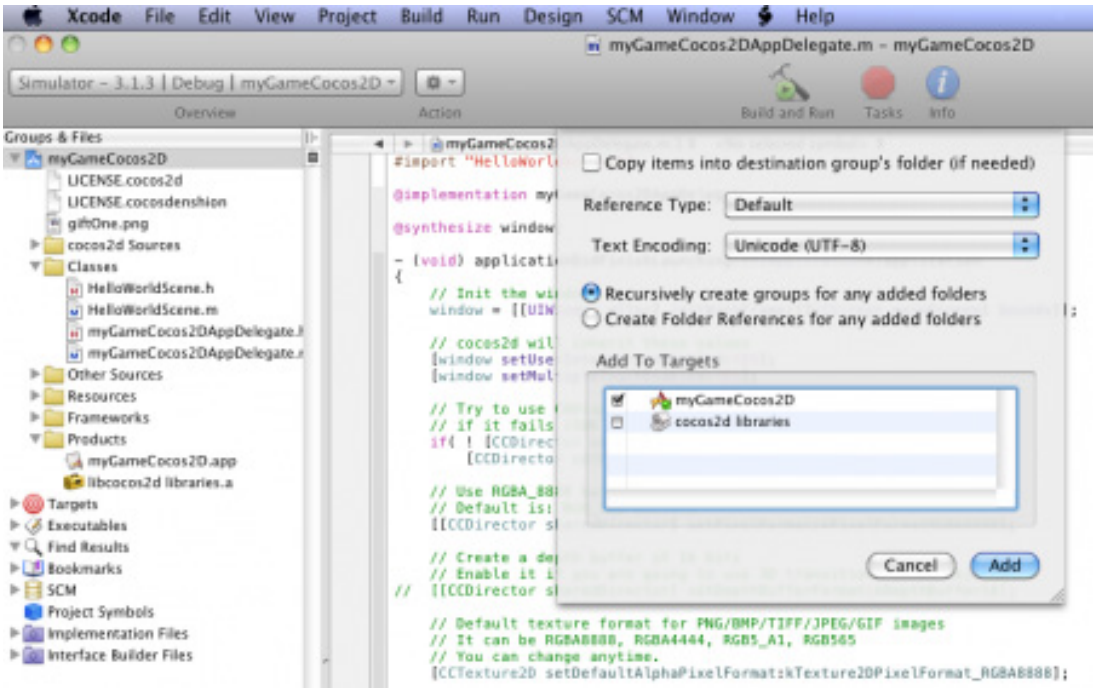
Core Audio es la librería que permite reproducir, procesar y grabar audio en nuestras aplicaciones. Además permite reproducir simultáneamente uno o más sonidos y el direccionamiento de audio automático cuando se utilizan auriculares, parlantes y auriculares Bluetooth. Por su parte OpenAL es ideal para reproducir sonidos en un espacio 3D.

Implementación de archivos de audio con Cocos2D:

1. Agregar estos dos archivos de audio a nuestro proyecto:
   SoundBackground:
   http://www.maestrosdelweb.com/images/2010/04/soundBackground.m4a

   Sound Effect:
   http://www.maestrosdelweb.com/images/2010/04/soundEffect.m4a





2. Iniciar el archivo de audio agregado en el método (id)init

```
-(id) init{
    // ccc4 Estable el color de Fondo
    if( (self=[super initWithColor:ccc4(255,255,255,255)] )) {
        CGSize winSize = [[CCDirector sharedDirector] winSize];
        // Crea un objeto tipo CCSprite que contiene la imagen agregada
        player = [CCSprite spriteWithFile:@"giftOne.png"
        rect:CGRectMake(0, 0, 186, 186)];
        // Establece la posición del objeto
        player.position = ccp(player.contentSize.width/2, winSize.height/2);
        // Agrega el objeto a la Capa
        [self addChild:player];
        // Habilita el evento Touch
        self.isTouchEnabled = YES;
        // Agrega el archivo de audio "soundBackground.m4a"
        [[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"soundBackground.m4a"];
    }
    return self;
}
```

3. Implementar el segundo archivo de audio, en el evento TouchBegan

```
- (void)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    [self moveObject];
    [[SimpleAudioEngine sharedEngine] playEffect:@"soundEffect.m4a"];
}
```

4. Compilar y ejecutar:

En estos momentos hemos implementado los dos archivos de audio en nuestra aplicación: el primero como música de fondo y el segundo como efecto de sonido al momento de tocar la pantalla. Como se puede observar es un proceso realmente sencillo pero de gran utilidad para lograr un mejor desempeño en nuestras aplicaciones.

Puedes descargar el proyecto para realizar el liberador procedimiento de “Build and Go”.

Descargar el proyecto:
http://www.maestrosdelweb.com/images/2010/04/myGameCocos2D1.zip