



# **Sumário**

*Capítulo 1: Multitask - OK*

*Capítulo 2: Core Motion - OK*

*Capítulo 3: Core Telephony - OK*

*Capítulo 4: iAd - OK*

*Capítulo 5: Quick Look - OK*

*Capítulo 6: AVFoundation - OK*

**Capítulo 7: Assets Library**

*Capítulo 8: Image I/O OK*

*Capítulo 9: GCD - Grand Central Dispatch - OK*

*Capítulo 10: Core Location - OK*

*Capítulo 11: Map Overlays - OK*

*Capítulo 12: Envio de SMS - OK*

## ***Capítulo 1 - Multitask***



Desde a versão 4.0 do IOS SDK, a Apple introduziu um novo modelo de programação (Paradigma) chamado Blocks.

Os blocos também são conhecidos como “Closures” ou “Lambdas”, e estão presentes em diversas linguagens de programação tais como Python, Ruby, Javascript, entre outras. Blocos são objetos que executam uma determinada tarefa em trechos separados do seu código. É como se fossem funções que podem ser chamadas a qualquer momento em qualquer parte do código.

**Lembre-se: Blocos são funções, porém eles podem ser escritos *inline* juntamente com o resto do seu código *dentro de funções*.**

O caractere ^ (Circunflexo) é o caractere que define a marcação de um determinado bloco. Este caractere deverá ser utilizado para demarcar e chamar os seus blocos.

Apesar da sua sintaxe parecer enigmática, você verá que não é difícil trabalhar com blocos.

Vejamos um exemplo abaixo que faz a utilização de blocos :

```
// Aqui declaramos um bloco que só executa códigos em
sequencia. Este bloco não retorna nenhum valor (void!).
```

```
void(^contador)(void) = ^{
    int x;
    for (x = 1; x <= 10; x++) {
        printf("%i ",x);
    }
};
```

```
// Este outro bloco recebe 2 parametros e retorna a soma dos
dois valores.
```

```
int (^soma)(int, int) = ^(int num1, int num2) {
    return num1 + num2;
};
```

```
// Repare que os exemplos acima se parecem com funções, e não
só se parecem, eles são exatamente funções!
```

```
- (void)viewDidLoad {
    [super viewDidLoad];
    CGRect cacheFrame = self.imageView.frame;
    [UIView animateWithDuration:1.5
```

```

animations:^(
    CGRect newFrame = self.imageView.frame;
    newFrame.origin.y = newFrame.origin.y + 150.0;
    self.imageView.frame = newFrame;
    self.imageView.alpha = 0.2;
}
completion:^(BOOL finished) {
    if (finished) {
        self.imageView.frame = cacheFrame;
        self.imageView.alpha = 1.0;
    }
}
];

```

```

NSString *area = @"Europe";
NSArray *timeZonesNames = [NSTimeZone
knownTimeZoneNames];
NSIndexSet *areaIndexes = [timeZonesNames
indexesOfObjectsWithOptions:NSEnumerationConcurrent
passingTest:^(id obj, NSUInteger idx, BOOL *stop) {
    NSString *tmpString = (NSString *)obj;
    NSLog(@"%i %@", idx, (NSString *)obj);
    return [tmpString hasPrefix:area];
}];
NSLog(@"Encontrado: %d", [areaIndexes count]);

NSLog(@"-----");

// Aqui chamamos um bloco como se fosse uma função;
contador();

// Chamamos o bloco soma utilizando 2 parâmetros, e
convertemos para um objeto NSNumber.
NSNumber *total = [NSNumber numberWithInt:soma(20,50)];
NSLog(@"Total: %@", total);

id totalBlock = ^{
    int valor1 = 120;
    int valor2 = 300;
    return (valor2 - valor1);
};

```

```

        NSLog(@"Total block = %@",[NSNumber
numberWithInt:totalBlock]);
    }

```

Executando aplicativos em background utilizando blocos:

Verifique se o dispositivo aceita processos em background:

```

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

    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
    /*
    UIBackgroundTaskIdentifier bgTask = [[UIApplication
sharedApplication]beginBackgroundTaskWithExpirationHandler:^
{}]];
    */
    UIDevice* device = [UIDevice currentDevice];
    BOOL backgroundSupported = NO;
    if ([device respondsToSelector:@selector
(isMultitaskingSupported)]) {
        backgroundSupported = device.multitaskingSupported;
        NSLog(@"Background suportado!");
    }

    return YES;
}
- (void)applicationDidEnterBackground:(UIApplication
*)application {

    NSLog(@"Entrou em background!");
    UIApplication *app = [UIApplication
sharedApplication];

    bgTask = [app
beginBackgroundTaskWithExpirationHandler:^{
    [app endBackgroundTask:bgTask];
    bgTask = UIBackgroundTaskInvalid;
}]];

```

```
dispatch_async(dispatch_get_global_queue  
(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
```

// Aqui colocamos todo o código que será executado em background. Exemplo de um loop em background:

```
int x;  
for(x = 1; x <= 10; x++) {  
    NSLog(@"%i\n", x);  
    // Aguarda por 1 segundo até fazer a próxima  
    // iteração.  
    [NSThread sleepForTimeInterval:1];  
}
```

```
[app endBackgroundTask:bgTask];  
bgTask = UIBackgroundTaskInvalid;  
});
```

```
}
```

## ***Capítulo 2 - CoreMotion***





O Framework Core Motion, é o framework responsável por permitir que o desenvolvedor acesse o acelerômetro e em alguns modelos o giroscópio dos dispositivos.

No exemplo abaixo mostramos como acessar o acelerômetro:

O primeiro passo é adicionar o framework CoreMotion.framework, e no arquivo de interface adicionar o header <CoreMotion/CoreMotion.h>.

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

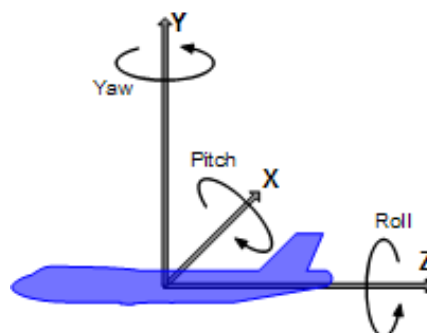
    manager = [[CMMotionManager alloc] init];

    // Verificamos se o dispositivo possui um giroscópio.
    if (manager.gyroAvailable) {
        [manager startDeviceMotionUpdates];
        [NSTimer scheduledTimerWithTimeInterval:0.1
target:self selector:@selector(updateInterface) userInfo:nil
repeats:YES];
    }
}

-(void)updateInterface {

    // Dados combinados do acelerômetro e giroscópio
    CMDeviceMotion* motion = manager.deviceMotion;

    // Attitude representa a orientação de um corpo à um
    frame (X,Y,Z - Sistema de coordenadas). É utilizado como
    base, os ângulos de Euler,
    // que nada mais são as rotações em cada eixo.
```



```

    CMAttitude* attitude = motion.attitude;

    // Pitch é a rotação no eixo lateral que é passado de um
    lado para o outro
    float pitch = attitude.pitch;

    // Roll é a rotação em volta de um eixo longitudinal, que
    é passada de cima para baixo.
    float roll = attitude.roll;

    // Yaw é uma rotação em volta de um eixo vertical do
    dispositivo. Ela é perpendicular ao corpo do dispositivo,
    // com sua origem no centro de gravidade e direcionada
    para a parte inferior do dispositivo.
    float yaw = attitude.yaw;
    NSLog(@"pitch %.2f roll %.2f yaw %.2f ", pitch, roll,
    yaw);
    attitudeLabel.text = [NSString stringWithFormat:@"Pitch:
    %.2f, Roll: %.2f, Yaw: %.2f", pitch, roll, yaw];

    // Quaternion, é utilizado para achar o eixo e o ângulo
    de rotação em uma Matriz. Utiliza as equações de Hamilton.
    CMQuaternion quaternion = attitude.quaternion;
    float quaternionX = quaternion.x;
    float quaternionY = quaternion.y;
    float quaternionZ = quaternion.z;
    float quaternionW = quaternion.w;
    NSLog(@"X: %.2f, Y: %.2f, Z: %.2f, W: %.2f ",
    quaternionX, quaternionY, quaternionZ, quaternionW);
    quaternionLabel.text = [NSString stringWithFormat:@"X: %.
    2f, Y: %.2f, Z: %.2f, W: %.2f ", quaternionX, quaternionY,
    quaternionZ, quaternionW];

    // Rotation Matrix é utilizado para retornar resultados
    da rotação em um espaço Euclidiano
    CMRotationMatrix rotationMatrix =
    attitude.rotationMatrix;
    float m11 = rotationMatrix.m11;
    float m12 = rotationMatrix.m12;
    float m13 = rotationMatrix.m13;
    float m21 = rotationMatrix.m21;

    NSLog(@"m11 %.2f m12 %.2f m13 %.2f m21 %.2f ", m11, m12,
    m13, m21);

```

```

    rotationMatrixLabel.text = [NSString
stringWithFormat:@"m11 %.2f m12 %.2f m13 %.2f m21 %.2f ",
m11, m12, m13, m21];
    // e etc... m11 - m33

    // Taxa de rotação em torno dos 3 eixos
    CMRotationRate rotationRate = motion.rotationRate;
    float rotationRateX = rotationRate.x;
    float rotationRateY = rotationRate.y;
    float rotationRateZ = rotationRate.z;
    NSLog(@"X: %.2f, Y: %.2f, Z: %.2f ", rotationRateX,
rotationRateY, rotationRateZ);
    rotationRateLabel.text = [NSString stringWithFormat:@"X:
%.2f, Y: %.2f, Z: %.2f ", rotationRateX, rotationRateY,
rotationRateZ];

    // Gravidade em torno do 3 eixos.
    CMAcceleration gravity = motion.gravity;
    float gravityX = gravity.x;
    float gravityY = gravity.y;
    float gravityZ = gravity.z;

    NSLog(@"X: %.2f, Y: %.2f, Z: %.2f ", gravityX, gravityY,
gravityZ);
    gravityLabel.text = [NSString stringWithFormat:@"X: %.2f,
Y: %.2f, Z: %.2f", gravityX, gravityY, gravityZ];
}

```

## ***Capítulo 3 - CoreTelephony***



O Core telephony é um framework que nos permite obter informações sobre a rede de telefonia celular.

As operadoras de telefonia pode obter essas informações para fornecer serviços específicos para os seus assinantes, por exemplo serviços de VoIP.

As classes abaixo fazem parte do Framework CoreTelephony:

1. [CTCall](#)
2. [CTCallCenter](#)
3. [CTCarrier](#)
4. [CTTelephonyNetworkInfo](#)

```
#import "TelephonyViewController.h"

@implementation TelephonyViewController

// Bloco da propriedade callEventHandler
void (^meuBlock)(CTCall*) = ^(CTCall* myCall){
    NSLog(@"myCall %@", myCall.callState);
};

- (void)viewDidLoad {
    [super viewDidLoad];

    // Instancia a classe que obtém informações sobre a rede de telefonia
    networkInfo = [[CTTelephonyNetworkInfo alloc] init];

    // Obtém a propriedade que retorna o informações da operadora
    theCarrier = networkInfo.subscriberCellularProvider;

    NSLog(@"The Carrier: %@", theCarrier);
    carrierTextView.text = [NSString stringWithFormat:
        @"Carrier Name: %@\n ISO Country Code: %@\nMobile Country Code: %@\nMobile Network Code: %@\nAllows VOIP %i",
        theCarrier.carrierName,
        theCarrier.isoCountryCode,
        theCarrier.mobileCountryCode,
        theCarrier.mobileNetworkCode,
```

```

[NSNumber
numberWithBool:theCarrier.allowsVOIP]];

// Esta classe permite obter uma lista das ligações que
estão sendo recebidas no momento, inclusive o seu estado.
callCenter = [[CTCallCenter alloc] init];

// Esta propriedade é disparada todas das vezes que um
evento de chamada for alterado.
callCenter.callEventHandler = meuBlock;

// [NSTimer scheduledTimerWithTimeInterval:0.3 target:self
selector:@selector(updateInterface) userInfo:nil
repeats:YES];
statesArray = [[NSMutableArray alloc] initWithCapacity:
4];
}

-(void)updateInterface {
    // Pega qualquer objeto na lista de objetos
    theCall = [callCenter.currentCalls anyObject];

    callTextView.text = [NSString stringWithFormat:@"Caller
ID: %@\n\nCall State: %@\n\n", theCall.callID,
theCall.callState];

    NSLog(@"Caller ID: %@\n\nCall State: %@\n\n",
theCall.callID, theCall.callState);
}

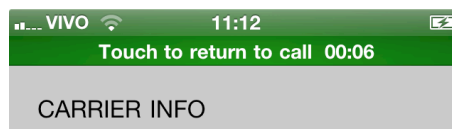
```



Carrier Name: VIVO  
ISO Country Code: br  
Mobile Country Code: 724  
Mobile Network Code: 10  
Allows VOIP 1041942452

#### CALL INFO

Caller ID: (null)  
Call State: (null)



Carrier Name: VIVO  
ISO Country Code: br  
Mobile Country Code: 724  
Mobile Network Code: 10  
Allows VOIP 1041942452

#### CALL INFO

Caller ID: 213DE6C7-E215-4270-9310-3A43E5C486CC  
Call State: CTCallStateConnected

## ***Capítulo 4 - iAd***



O iAd permite que o desenvolvedor adicione banners de publicidade no seu aplicativo.

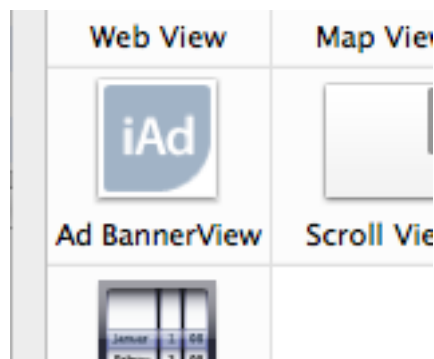
Todas as vezes que o usuário abrir e interagir com o banner o desenvolvedor será remunerado em 60% do valor da publicidade.

Porém há uma desvantagem, você ira perder 50 pontos quando sua aplicação estiver no modo porta-retrato e 32 pontos quando estiver no modo paisagem.

É muito simples implementar o iAd Banner na sua aplicação, basta seguir os procedimentos abaixo:

Adicione o framework iAd.Framework na sua aplicação.

Abra o Interface Builder e adicione um Ad BannerView à sua aplicação:



No seu arquivo de interface importe o header principal do framework. Exemplo:

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

Agora implemente os seguintes métodos:

```
- (void)viewDidLoad {  
    // Define o delegate  
    bannerView.delegate = self;  
    [super viewDidLoad];  
}  
  
// Este método delegado é chamando quando um banner é  
// terminado de carregar.  
-(void) bannerViewDidLoadAd:(ADBannerView *)banner {  
    NSLog(@"Carregou uma publicidade");  
}
```



```

}

// Quando houver falha ao receber um banner este método
delegado será chamado.
-(void) bannerView:(ADBannerView *)banner
didFailToReceiveAdWithError:(NSError *)error {
    NSLog(@"Erro ao receber banner: %@", error);
}

// Este método é chamado logo após o usuário fechar a
propaganda.
-(void) bannerViewActionDidFinish:(ADBannerView *)banner {
    UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Recado" message:@"Obrigado por ler nossos
comerciais" delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil] autorelease];
    [alert show];
    NSLog(@"Fechou!");
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation {
    // Redimensiona o banner quando o dispositivo estiver no
modo paisagem.
    if (UIInterfaceOrientationIsLandscape
(interfaceOrientation)) {

// A propriedade currentContentSizeIdentifier, identifica o
tamanho corrente da propaganda.
        bannerView.currentContentSizeIdentifier =
ADBannerContentSizeIdentifier480x32;
        CGSize bannerViewSize = [ADBannerView
sizeFromBannerContentSizeIdentifier:ADBannerContentSizeIdenti
fier480x32];
        bannerView.center = CGPointMake(bannerViewSize.width/
2, 300 - bannerViewSize.height/2);
    }
    else {
        bannerView.currentContentSizeIdentifier =
ADBannerContentSizeIdentifier320x50;
    }
}

```

```
        CGSize bannerViewSize = [ADBannerView
sizeFromBannerContentSizeIdentifier:ADBannerContentSizeIdentifi
fier320x50];
        bannerView.center = CGPointMake(bannerViewSize.width/
2, 460 - bannerViewSize.height/2);

    }
    return YES;
}
```

## ***Capítulo 5 - Quicklook e Document Interaction***

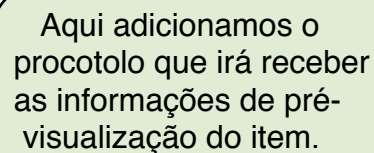
O Quick Look framework permite visualizar arquivos em formatos que o iPhone não pode manipular, tais como iWork e Microsoft Office. Ele oferece maior controle que o framework UIDocumentInteractionController, nos permitindo escolher se o conteúdo será exibido em um navigation controller ou em tela cheia (Fullscreen).

Já a classe UIDocumentInteractionController permite o usuário escolher a ação que deverá ser tomada, por exemplo, exibir, abrir ou até copiar o arquivo.

Iremos trabalhar com os 2 frameworks . Seque baixo o exemplo:

Aqui criamos uma classe que irá ser a responsável pela exibição do item:

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



Aqui adicionamos o protocolo que irá receber as informações de pré-visualização do item.

```
@interface QuickLookItem : NSObject <QLPreviewItem> {
    NSString* previewItemTitle;
    NSURL* previewItemURL;
}

-(id)initWithURL:(NSURL*)url;

@property (readonly) NSString* previewItemTitle;
@property (readonly) NSURL* previewItemURL;

@end
```

```
#import "QuickLookItem.h"

@implementation QuickLookItem

@synthesize previewItemTitle;
@synthesize previewItemURL;

-(id)initWithURL:(NSURL*)url {
    if((self = [super init])){
        previewItemURL = url;
    }
    return self;
}

@end
```

## Arquivo de interface do ViewController

```
#import <UIKit/UIKit.h>
#import <QuickLook/QuickLook.h>
#import "QuickLookItem.h"

@interface DocumentPreviewViewController : UIViewController
<UIDocumentInteractionControllerDelegate,
QLPreviewControllerDelegate, QLPreviewControllerDataSource>{

    UIDocumentInteractionController* preview;
    QuickLookItem* item;
}

-(IBAction)abrirPDF;
-(IBAction)abrirPPT;
-(IBAction)abrirDOC;
-(IBAction)abrirXLS;
-(IBAction)abrirKeyNote;
-(IBAction)abrirNumbers;

@end
```

## Arquivo de implementação do ViewController

```
#import "DocumentPreviewViewController.h"

@implementation DocumentPreviewViewController

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

//
#pragma mark UIDocumentInteractionControllerDelegate
// Este método delegado, é o método responsável por retornar
qual o controller será utilizado.
-(UIViewController *)
documentInteractionControllerViewControllerForPreview:
(UIDocumentInteractionController *)controller {
    return self;
}

#pragma mark QLPreviewControllerDataSource
// Este método delegado, é responsável por retornar quantos
items existem para serem exibidos.
- (NSInteger)numberOfPreviewItemsInPreviewController:
(QLPreviewController *) controller {
    return 1;
}

// Retorna o item a ser exibido.
-(id <QLPreviewItem>)previewController:(QLPreviewController
*)controller previewItemAtIndex:(NSInteger)index {
    return item;
}

#pragma mark -
#pragma mark Application Life

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

-(IBAction)abrirPDF {
    NSString* path = [[NSBundle mainBundle]
pathForResource:@"exemplo" ofType:@"pdf"];
}
```

```

    NSURL* url = [NSURL fileURLWithPath:path];

    //
    preview = [[UIDocumentInteractionController
interactionControllerWithURL:url] retain];
    preview.delegate = self;
    // Aqui será apresentado um menu (Action Sheet)
    permitindo ao usuário escolher
    // qual ação deverá ser tomada.
    [preview presentOptionsMenuFromRect:CGRectZero
inView:self.view animated:YES];
}

-(IBAction)abrirPPT {
    NSString* path = [[NSBundle mainBundle]
pathForResource:@"exemplo" ofType:@"ppt"];
    NSURL* url = [NSURL fileURLWithPath:path];

    preview = [[UIDocumentInteractionController
interactionControllerWithURL:url] retain];
    preview.delegate = self;
    // Neste outro exemplo, o arquivo será aberto
    automaticamente e será exibido no
    // no navigation controller, um botão para o usuário
    escolher a ação a ser tomada.
    [preview presentPreviewAnimated:YES];
}

-(IBAction)abrirDOC {
    NSString* path = [[NSBundle mainBundle]
pathForResource:@"exemplo" ofType:@"docx"];
    NSURL* url = [NSURL fileURLWithPath:path];

    preview = [[UIDocumentInteractionController
interactionControllerWithURL:url] retain];
    preview.delegate = self;
    [preview presentPreviewAnimated:YES];
}

-(IBAction)abrirXLS {
    NSString* path = [[NSBundle mainBundle]
pathForResource:@"exemplo" ofType:@"xlsx"];
    NSURL* url = [NSURL fileURLWithPath:path];

    item = [[QuickLookItem alloc] initWithURL:url];

```

```

        if ([QLPreviewController canPreviewItem:item]){
            QLPreviewController* quickLook =
[[QLPreviewController alloc] init];
            quickLook.dataSource = self;
            quickLook.delegate = self;
            [self presentViewController:quickLook
animated:YES];
            [quickLook release];
        }
    }

-(IBAction)abrirKeyNote {
    NSString* path = [[NSBundle mainBundle]
pathForResource:@"exemplo" ofType:@"key"];
    NSURL* url = [NSURL fileURLWithPath:path];
    // A partir da versão 4.0 do IOS, também foi introduzido
o manipulador de arquivos da
    // família iWork (Pages, Numbers, Keynote), que permite
ao UIDocumentInteractionController
    // abrir arquivos desta família.
    preview = [[UIDocumentInteractionController
interactionControllerWithURL:url] retain];
    preview.delegate = self;
    // Caso você tenha o Keynote instalado no seu iOS Device
(Atualmente iPad), irá aparecer a opção para abrir
    // no Keynote ou somente pré-visualizar o arquivo.
    [preview presentOptionsMenuFromRect:CGRectZero
inView:self.view animated:YES];
}

-(IBAction)abrirNumbers {
    NSString* path = [[NSBundle mainBundle]
pathForResource:@"exemplo" ofType:@"numbers"];
    NSURL* url = [NSURL fileURLWithPath:path];

    item = [[QuickLookItem alloc] initWithURL:url];

    if ([QLPreviewController canPreviewItem:item]){
        QLPreviewController* quickLook =
[[QLPreviewController alloc] init];
        quickLook.dataSource = self;
        quickLook.delegate = self;
    }
}

```



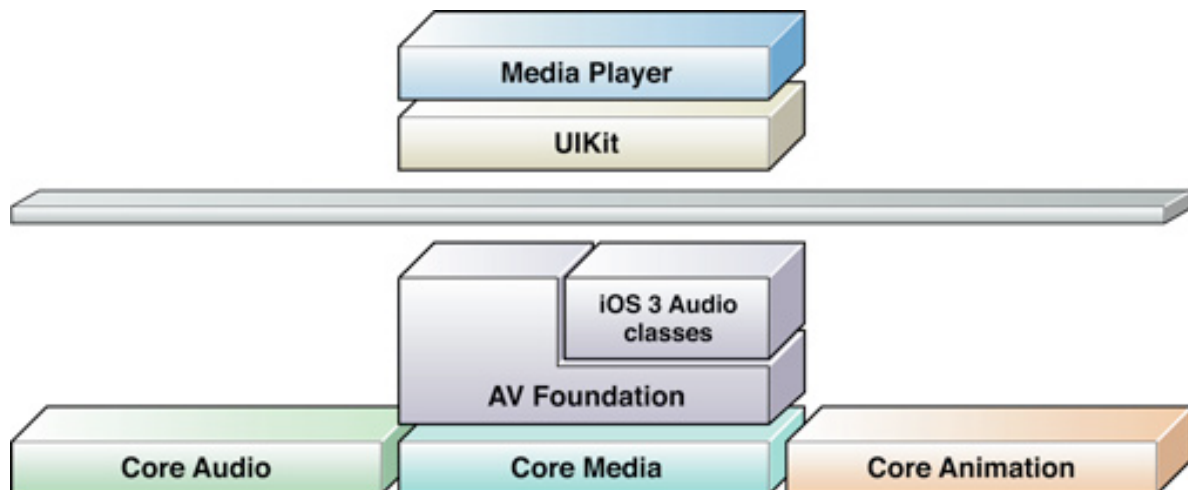
```
        [self presentViewController:quickLook
animated:YES];
        [quickLook release];
    }
}
```

## ***Capítulo 6 - AVFoundation***



O AV Foundation é um framework para manipular Audio e video no IOS. Com ele é possível acessar informações detalhadas de um recurso, como por exemplo acessar tag's id3 de um arquivo MP3, mixar um audio com um video, ou até tocar dois audios simultaneamente.

Na figura abaixo vemos a estrutura multimedia do IOS:



Como podemos observar o AV Foundation está em uma camada intermediária entre o nível mais baixo (Core Audio, Core Media e Core Animation) e o nível mais alto (Media Player).

O AV Foundation deve ser utilizado quando precisamos criar, editar ou recodificar um audio ou video, obter dados de um dispositivo de entrada, como o fone por exemplo, ou editar metadados de um recurso.

O exemplo abaixo demonstra como tocar dois arquivos de audio simultaneamente e como controlar a velocidade de reprodução e a posição dos mesmos:

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
    // Cria a url com a localização do item a ser tocado.  
    NSURL *item1URL = [NSURL fileURLWithPath:[NSBundle  
mainBundle] pathForResource:@"Marrakech" ofType:@"mp3"]];  
    // AVPlayerItem define a media que será utilizada.  
    audio1 = [AVPlayerItem playerItemWithURL:item1URL];  
    // Instanciamos a classe AVPlayer inicializando o item  
que será tocado.
```

```

player1 = [[AVPlayer alloc] initWithPlayerItem:audio1];

NSURL *item2URL = [NSURL fileURLWithPath:[NSBundle
 mainBundle] pathForResource:@"Famicom" ofType:@"mp3"]];
audio2 = [AVPlayerItem playerItemWithURL:item2URL];
player2 = [[AVPlayer alloc] initWithPlayerItem:audio2];

isPlaying1 = NO;
isPlaying2 = NO;

UIImage *greenImage = [[UIImage
 imageNamed:@"green_button.png"]
 stretchableImageWithLeftCapWidth:12.0 topCapHeight:0.0];
[playPause1Button setBackgroundImage:greenImage
 forState:UIControlStateNormal];
[playPause2Button setBackgroundImage:greenImage
 forState:UIControlStateNormal];

[NSTimer scheduledTimerWithTimeInterval:0.01 target:self
 selector:@selector(getCurrentTime1) userInfo:nil
 repeats:YES];
[NSTimer scheduledTimerWithTimeInterval:0.01 target:self
 selector:@selector(getCurrentTime2) userInfo:nil
 repeats:YES];
}

-(void)getCurrentTime1 {
    // CMTimeGetSeconds([player1 currentTime]) converte para
    segundos o tempo decorrente do item que está sendo
    reproduzido.
    timeLabel1.text = [NSString stringWithFormat:@"%f",
    CMTimeGetSeconds([player1 currentTime])];
    positionSlider1.value = CMTimeGetSeconds([player1
    currentTime]);
}

-(IBAction)setCurrentTime1 {
    // A propriedade [audio2 asset].duration recebe o tempo
    de duração de um determinado item
    if (positionSlider1.value < (CMTimeGetSeconds([audio2
    asset].duration))) {
        [player1 seekToTime:CMTimeMake
        (positionSlider1.value, 1)];
        NSLog(@"Current time: %f", positionSlider1.value);
        [self getCurrentTime1];
    }
}

```

```

    }
}

-(IBAction)playPause1 {

    if (isPlaying1) {
        isPlaying1 = NO;
        [player1 pause];
        [playPause1Button setTitle:@"Play"
forState:UIControlStateNormal];
    }
    else {
        isPlaying1 = YES;
        [player1 play];
        NSLog(@"Item 1: %f",CMTimeGetSeconds([audio1
forwardPlaybackEndTime]));
        [playPause1Button setTitle:@"Pause"
forState:UIControlStateNormal];
    }
}

-(IBAction)changeSpeed1 {
    // Define a velocidade de reprodução do player, que vai
de 0.0 até 1.0
    player1.rate = speedSlider1.value;
    speedLabel1.text = [NSString stringWithFormat:@"%f",
speedSlider1.value];
    NSLog(@"Changing speed audio to value:
%f",speedSlider1.value);
}

-(IBAction)AVAudioPlayer1 {
    NSLog(@"Setting speed to normal");
    player1.rate = 1.0;
    speedLabel1.text = @"1.0";
    [speedSlider1 setValue:1];
}

#pragma mark -
#pragma mark Audio 2

-(void)getCurrentTime2 {

```

```

        timeLabel2.text = [NSString stringWithFormat:@"%f",
CMTimeGetSeconds([player2 currentTime])];
    }

    -(IBAction)playPause2 {
        if (isPlaying2) {
            isPlaying2 = NO;
            [player2 pause];
            [playPause2Button setTitle:@"Play"
forState:UIControlStateNormal];
        }
        else {
            isPlaying2 = YES;
            [player2 play];
            [playPause2Button setTitle:@"Pause"
forState:UIControlStateNormal];
        }
    }

    -(IBAction)changeSpeed2 {
        player2.rate = speedSlider2.value;
        speedLabel2.text = [NSString stringWithFormat:@"%f",
speedSlider2.value];
        NSLog(@"Changing speed audio to value:
%f", speedSlider2.value);
    }

    -(IBAction)resetspeed2 {
        NSLog(@"Setting speed to normal");
        player2.rate = 1.0;
        speedLabel2.text = @"1.0";
        [speedSlider2 setValue:1];
    }

    -(IBAction)resetspeed1 {
        NSLog(@"Setting speed to normal");
        player1.rate = 1.0;
        speedLabel1.text = @"1.0";
        [speedSlider1 setValue:1];
    }
}

```



Como podemos observar o exemplo dado, é bastante simples reproduzir audio com o AVFoundation.

Agora iremos fazer um exemplo mais complexo, onde iremos reproduzir um video e um audio simultaneamente.

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

    // SOURCE FILES
    NSString* pathVideo = [[NSBundle mainBundle]
pathForResource:@"sample_iPod" ofType:@"m4v"];
    NSURL* videoUrl = [NSURL fileURLWithPath:pathVideo];

    NSString* pathMusic = [[NSBundle mainBundle]
pathForResource:@"musica" ofType:@"mp3"];
    NSURL* audioUrl = [NSURL fileURLWithPath:pathMusic];

    // Transforma a URL em um Asset (recurso)
    AVURLAsset* audioAsset = [[AVURLAsset alloc]
initWithURL:audioUrl options:nil];
    AVURLAsset* videoAsset = [[AVURLAsset alloc]
initWithURL:videoUrl options:nil];

    // Cria uma nova composição a partir de recursos
    existentes.
    AVMutableComposition* mixComposition =
[AVMutableComposition composition];

    // Permite adicionar um recurso a uma composição,
    mantendo intacto o arquivo original.
    AVMutableCompositionTrack *compositionCommentaryTrack =
[mixComposition addMutableTrackWithMediaType:AVMediaTypeAudio
preferredTrackID:kCMPersistentTrackID_Invalid];

    // Aqui inserimos uma faixa de tempo na composição
    [compositionCommentaryTrack
insertTimeRange:CMTimeRangeMake(kCMTimeZero,
audioAsset.duration) ofTrack:[audioAsset
tracksWithMediaType:AVMediaTypeAudio] objectAtIndex:0]
atTime:kCMTimeZero error:nil];

    // Repetimos os processos acima para o video também.
```



```

    AVMutableCompositionTrack *compositionVideoTrack =
[mixComposition addMutableTrackWithMediaType:AVMediaTypeVideo
preferredTrackID:kCMPersistentTrackID_Invalid];

    [compositionVideoTrack insertTimeRange:CMTimeRangeMake
(kCMTimeZero, videoAsset.duration) ofTrack:[videoAsset
tracksWithMediaType:AVMediaTypeVideo] objectAtIndex:0]
atTime:kCMTimeZero error:nil];

    // Exporta a composição
    AVAssetExportSession* _assetExport =
[[AVAssetExportSession alloc] initWithAsset:mixComposition
presetName:AVAssetExportPresetPassthrough];

    NSString* videoName = @"/Documents/export.mov";
    NSString *exportPath = [NSHomeDirectory()
stringByAppendingString:videoName];
    NSURL *exportUrl = [NSURL fileURLWithPath:exportPath];

    if ([[NSFileManager defaultManager]
fileExistsAtPath:exportPath])
    {
        [[NSFileManager defaultManager]
removeItemAtPath:exportPath error:nil];
    }

    _assetExport.outputFileType = @"com.apple.quicktime-
movie";

    NSLog(@"file type %@",_assetExport.outputFileType);

    _assetExport.outputURL = exportUrl;
    _assetExport.shouldOptimizeForNetworkUse = YES;

    // Este bloco será executado quando a exportação estiver
terminada.
    [_assetExport exportAsynchronouslyWithCompletionHandler:
^(void ) {
        // your completion code here
        NSLog(@"Terminei de exportar!");
        [self performSelectorOnMainThread:@selector
(playMovie:) withObject:exportUrl waitUntilDone:NO];
    }
];

```

```
}

-(void)playMovie:(NSURL*)exportUrl {
    MPMoviePlayerViewController* player =
    [[MPMoviePlayerViewController alloc]
    initWithContentURL:exportUrl];
    [self presentViewController:player animated:YES];
}
```

## ***Capítulo 8 - Image I/O***



Image I/O permite que o programador acesse informações da maioria dos formatos de imagens atualmente utilizados.

Os tipos de informações que podem ser acessados vão desde, altura da imagem, largura da imagem, quantidade de pixels, data da imagem, entre outras. Caso a camera possua GPS, também serão exibidas as informações de posicionamento tais como, latitude, longitude, altura, etc.

Para que possamos trabalhar com o Framework Image I/O primeiramente devemos adicionar o framework ImageIO.framework, e logo em seguida adicionar o header ImageIO/ImageIO.h

Abaixo um exemplo de como obter as informações de uma imagem.

```
NSString* path = [[NSBundle mainBundle]
pathForResource:@"lucas" ofType:@"jpg"];

imageView.image = [UIImage imageWithContentsOfFile:path];

NSURL* url = [NSURL fileURLWithPath:path];
// Lê uma imagem a partir de uma URL
CGImageSourceRef source = CGImageSourceCreateWithURL
((CFURLRef)url, NULL);

// A funcao CGImageSourceCopyPropertiesAtIndex é
responsável por ler as propriedades de uma image.
// Sintaxe: CGImageSourceCopyPropertiesAtIndex(imagem,
index, opcoes)
NSDictionary* propsDict = (NSDictionary*)
CGImageSourceCopyPropertiesAtIndex(source, 0, NULL);

// kCGImagePropertyExifDictionary é a constante que
retorna as informações EXIF (Exchangeable Image File Format)
de uma imagem.
NSDictionary* exifDict = [propsDict objectForKey:
(NSString*)kCGImagePropertyExifDictionary];

// kCGImagePropertyGPSDictionary recebe as informações de
GPS, caso a camera tenha
NSDictionary* gpsDict = [propsDict objectForKey:
(NSString*)kCGImagePropertyGPSDictionary];

infoText.text = [NSString stringWithFormat:
@"Speed: %@\nExposure Time: %@\nFocal
Length: %@\n Latitude %@\n Longitude %@",
```

```

        [exifDict objectForKey:(NSString*)
kCGImagePropertyExifISOSpeedRatings],
        [exifDict objectForKey:(NSString*)
kCGImagePropertyExifExposureTime],
        [exifDict objectForKey:(NSString*)
kCGImagePropertyExifFocalLength],
        [gpsDict objectForKey:@"Latitude"],
        [gpsDict objectForKey:@"Longitude"]
    ];

    // Abra o console (⌘R) e visualize as informações da
    imagem.
    NSLog(@"Informações da imagem:
    \n-----\n%@", propsDict);

```

Também é possível obter informações da camera que a foto foi tirada:  
Cameras Canon:

```

CFStringRef kCGImagePropertyMakerCanonOwnerName;
CFStringRef kCGImagePropertyMakerCanonCameraSerialNumber;
CFStringRef kCGImagePropertyMakerCanonImageSerialNumber;
CFStringRef kCGImagePropertyMakerCanonFlashExposureComp;
CFStringRef kCGImagePropertyMakerCanonContinuousDrive;
CFStringRef kCGImagePropertyMakerCanonLensModel;
CFStringRef kCGImagePropertyMakerCanonFirmware;
CFStringRef kCGImagePropertyMakerCanonAspectRatioInfo;

```

Cameras Nikon:

```

CFStringRef kCGImagePropertyMakerNikonISOSetting;
CFStringRef kCGImagePropertyMakerNikonColorMode;
CFStringRef kCGImagePropertyMakerNikonQuality;
CFStringRef kCGImagePropertyMakerNikonWhiteBalanceMode;
CFStringRef kCGImagePropertyMakerNikonSharpenMode;
CFStringRef kCGImagePropertyMakerNikonFocusMode;
CFStringRef kCGImagePropertyMakerNikonFlashSetting;
CFStringRef kCGImagePropertyMakerNikonISOSelection;
CFStringRef kCGImagePropertyMakerNikonFlashExposureComp;
CFStringRef kCGImagePropertyMakerNikonImageAdjustment;
CFStringRef kCGImagePropertyMakerNikonLensAdapter;
CFStringRef kCGImagePropertyMakerNikonLensType;
CFStringRef kCGImagePropertyMakerNikonLensInfo;
CFStringRef kCGImagePropertyMakerNikonFocusDistance;
CFStringRef kCGImagePropertyMakerNikonDigitalZoom;
CFStringRef kCGImagePropertyMakerNikonShootingMode;

```

```
CFStringRef kCGImagePropertyMakerNikonShutterCount;  
CFStringRef kCGImagePropertyMakerNikonCameraSerialNumber;
```

## ***Capítulo 9 - Grand Central Dispatch***



A Grand Central Dispatch é uma tecnologia que foi desenvolvida pela Apple para que o sistema operacional, tire maior proveito dos múltiplos núcleos dos processadores atualmente utilizados por ela.

Hoje toda linha de Macintosh's e alguns modelos portáteis (iPhone e iPad) possuem processadores com múltiplos núcleos, o que é uma enorme vantagem, pois podemos executar processos em paralelo, aumentando a performance dos nossos programas.

A GCD (Grand Central Dispatch) como é conhecida, é uma biblioteca que permite ao programador, desenvolver aplicativos com suporte a processamento paralelo de forma fácil e dinâmica.

Algumas características da GCD:

- \* Permite ao programador extrair o máximo desempenho de sistemas Multi-núcleos;
- \* A Thread's são gerenciadas pelo sistema operacional, ao invés das aplicações.
- \* Permite que as thread's sejam distribuídas entre os núcleos dos dispositivos.
- \* A biblioteca que implementa a GCD é a libdispatch, e foi disponibilizada pela Apple sob a Licença Apache, o que permite outros sistemas utilizá-la. Esta biblioteca já está sendo utilizada pelas versões mais novas de FreeBSD e algumas implementações de Debian.

Os tipos de FIFO oferecidos pela GCD são 3:

Main: As tarefas são executadas na Main thread de forma serial.

Concurrent (simultâneo): As tarefas são executadas na ordem da fila, porém simultaneamente.

Serial: As tarefas são executadas na ordem da fila (FIFO).



O trecho de código abaixo executa um processo paralelo na fila global:

```
-(void)startParallelProcess {
    dispatch_async(dispatch_get_global_queue
(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        int x;
        for (x = 1; x <= 10000; x++) {
            [NSThread sleepForTimeInterval:0.5];
            NSLog(@"%i\n",x);
        }
    });
}
```

Neste segundo exemplo mostramos como executar um processo quando o dispositivo entrar em background:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];

    // Aqui criamos um processo que irá entrar na fila.
    Sintaxe: dispatch_queue_create("Um identificador
    único",atributo);
    // Atualmente o atributo não é utilizado e deve ser
    passado como NULL.
    queue_task_one = dispatch_queue_create
    ("br.com.quaddro.GCDTesteGrupos.queue_task_one", NULL);

    queue_task_two = dispatch_queue_create
    ("br.com.quaddro.GCDTesteGrupos.queue_task_two", NULL);

    // Retorna a queue principal
    main_queue = dispatch_get_main_queue();

    // Retorna a label de uma queue;
    NSLog(@"Main Queue: %@", [NSString
    stringWithCString:dispatch_queue_get_label(main_queue)
    encoding:NSUTF8StringEncoding]);

    return YES;
}
```

```

- (void)applicationDidEnterBackground:(UIApplication *)
application {
    NSLog(@"Entrou em background!");
    [application beginBackgroundTaskWithExpirationHandler:^
    {}];

    // Verifica se a task está suspensa
    if (isTaskPaused) {
        isTaskPaused = NO;
        // Reativa a execução da segunda task
        dispatch_resume(queue_task_two);
    }

    dispatch_async(main_queue, ^{
        // Aqui iremos executar um looping crescente em
background até o fim sem ser interrompido
        int x;
        for (x = 1; x <= 100; x++) {
            dispatch_async(queue_task_one, ^{
                NSLog(@"Task one: %i\n", x);
//                sleep(1);
                [NSThread sleepForTimeInterval:1];
            });
        }
        // Este looping será suspenso quando a
aplicação sair do background
        int i;
        for (i = 100; i >= 1; i--) {
            dispatch_async(queue_task_two, ^{
                NSLog(@"Task two: %i\n", i);
//                sleep(1);
                [NSThread sleepForTimeInterval:1];
            });
        }

    });
}

- (void)applicationWillEnterForeground:(UIApplication *)
application {
    // Suspende a execução da segunda task
    dispatch_suspend(queue_task_two);
    isTaskPaused = YES;
}

```

## ***Capítulo 10 - CoreLocation***



O Framework CoreLocation é o framework que permite ao desenvolvedor acessar os recursos de localização do iPhone. Com ele, é possível determinar a posição e a direção do usuário.

Ele também permite definir áreas geográficas, e monitorar quando um usuário entra em uma região pré-estabelecida.

A seguir veremos como monitorar quando um usuário entra em uma região:

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Com este método de classe, verificamos se o serviço de
    localização está habilitado.
    if ([CLLocationManager locationServicesEnabled]) {
        // Se estiver habilitado, instanciamos a classe
        CLLocationManager
        CLLocationManager* manager = [[CLLocationManager
        alloc] init];
        manager.delegate = self;
        // Iniciamos o monitoramento
        [manager startUpdatingLocation];

        // Este método de classe, verifica se há sinal
        suficiente para monitorar a mudança de posição.
        if ([CLLocationManager
        significantLocationChangeMonitoringAvailable]){
            // Diz para o CLLocationManager iniciar o
            monitoramento de região.
            [manager
            startMonitoringSignificantLocationChanges];
        }
    }
}

#pragma mark CLLocationManagerDelegate

// Este método delegado é o responsável por atualizar as
informações de mudança de região.
-(void) locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation fromLocation:
(CLLocation *)oldLocation {

    if (!gotFirstLocation){
```

```

        firstLocation = [newLocation retain];
        gotFirstLocation = YES;
    }

    // Exibe as informações de Latitude e Longitude
    locationLabel.text = [NSString stringWithFormat:@"%f %f",
newLocation.coordinate.latitude,
newLocation.coordinate.longitude];

    // Exibe a distancia do ponto inicial para o ponto atual.
    distanceLabel.text = [NSString stringWithFormat:@"%f
metros do ponto inicial", [firstLocation
distanceFromLocation:newLocation]];

    // Verifica se o monitoramento de região está disponível.
    if ([CLLocationManager regionMonitoringAvailable]){
        // Cria um objeto que define a área geográfica que
pode ser monitorada.
        // O primeiro parametro define a área a ser
monitorada a partir de um ponto central. O parametro radius
define o raio em metros a partir do ponto central.
        // Já o parametro identifier, serve para identificar
a area monitorada na sua aplicação.

        CLRegion* region = [[CLRegion alloc]
initWithCircularRegionWithCenter:newLocation.coordinate radius:10
identifier:@"myLocation"];
        // Inicia o monitoramento definindo a exatidão
desejada. Segue abaixo as constantes para o parametro
desiredAccuracy:
        /*
        kCLLocationAccuracyBestForNavigation;
        kCLLocationAccuracyBest;
        kCLLocationAccuracyNearestTenMeters;
        kCLLocationAccuracyHundredMeters;
        kCLLocationAccuracyKilometer;
        kCLLocationAccuracyThreeKilometers;
        */
        [manager startMonitoringForRegion:region
desiredAccuracy:kCLLocationAccuracyBest];
        statusLabel.text = @"Monitoramento Habilitado!";
        [region release];
    }

```

```

    }
}

// Este método delegado é chamado todas as vezes que o
usuário entrar na regioao determinada
-(void) locationManager:(CLLocationManager *)manager
didEnterRegion:(CLRegion *)region {
    UIAlertView* alert = [[UIAlertView alloc]
initWithTitle:@"Alerta!" message:@"Você entrou na região"
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
    [alert release];
    statusLabel.text = @"Entrou na região";
}

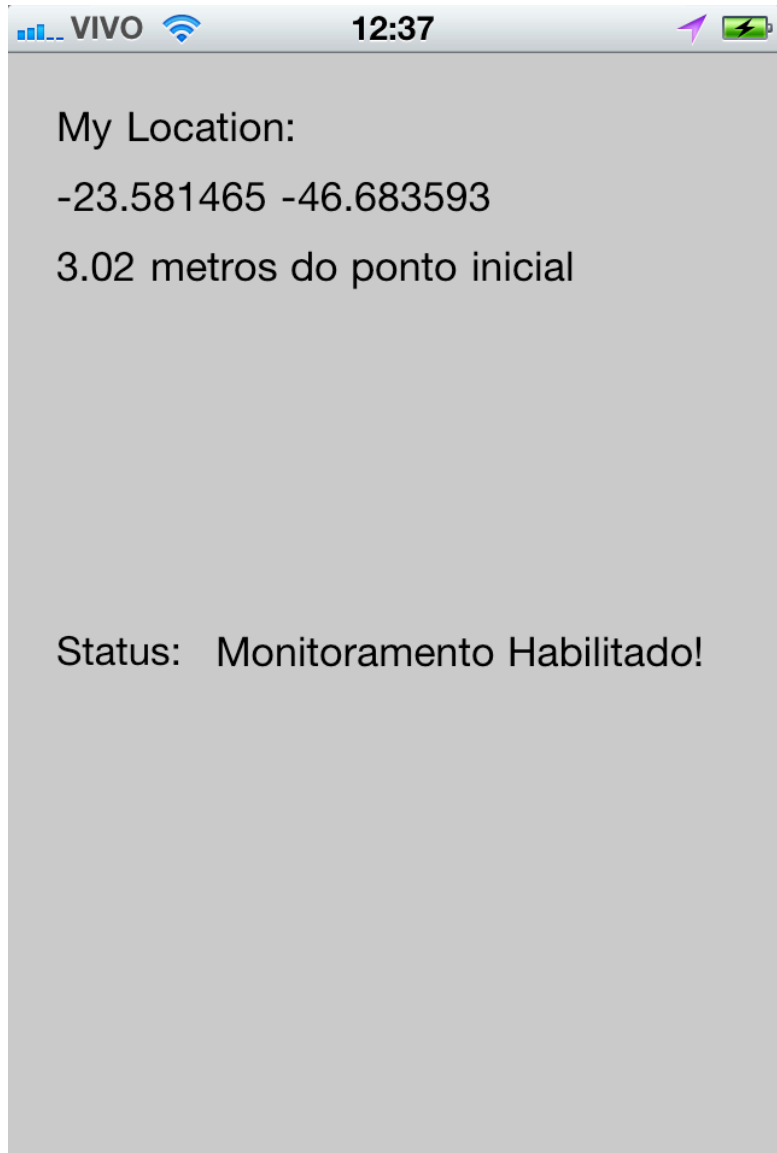
// Este método delegado é chamado todas as vezes que o
usuário sair da regioao determinada
-(void) locationManager:(CLLocationManager *)manager
didExitRegion:(CLRegion *)region {
    UIAlertView* alert = [[UIAlertView alloc]
initWithTitle:@"Alerta!" message:@"Você SAIU da região"
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
    [alert release];
    statusLabel.text = @"Saiu da região";
}

```

```

// Este método é chamado quando houver erros no monitoramento
- (void) locationManager:(CLLocationManager *)manager
monitoringDidFailForRegion:(CLRegion *)region withError:
(NSError *)error {
    UIAlertView* alert = [[UIAlertView alloc]
initWithTitle:@"Alerta!" message:@"Deu erro no
monitoramento." delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil];
    [alert show];
    [alert release];
    statusLabel.text = [NSString stringWithFormat:@"%@",
error];
}

```



## ***Capítulo 11 - Map Overlays***





Os maps overlays são utilizados quando queremos desenhar sobre os mapas do Map Kit.

Em conjunto com o framework CoreGraphics podemos desenhar círculos, linhas, e qualquer outro tipo de forma geométrica que você desejar.

O primeiro passo é adicionar os frameworks necessários, para isso clique com o botão direito em cima do grupo Frameworks -> Add -> Existing Frameworks e adicione os frameworks abaixo:

```
CoreLocation.framework  
CoreGraphics.framework  
MapKit.framework
```

Agora devemos importar os headers referentes aos frameworks. Vá até o seu arquivo de Interface da ViewController e importe os seguintes headers:

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

Também é preciso adicionar o protocolo do Delegate MKMapViewDelegate, Ex:

```
@interface MapViewController : UIViewController  
<MKMapViewDelegate> {  
    // Adicione um MapView  
    IBOutlet MKMapView* mapa;  
}
```

Depois de ter configurado o arquivo de interface, você deverá ir até o arquivo de implementação do seu ViewController. Insira os códigos como o exemplo a seguir:

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
  
    // CIRCULO NO MEIO  
  
    // Cria um círculo a partir de uma coordenada central. O  
    // parametro radius, é medido em  
    // metros a partir do centro da coordenada.
```

```

    MKCircle *circle = [MKCircle
circleWithCenterCoordinate:mapa.centerCoordinate radius:
1000000];
    // Adiciona o overlay no mapa
    [mapa addOverlay:circle];

    // PONTO
    // Cria uma anotação com um determinado titulo
    MKPointAnnotation *point = [[MKPointAnnotation alloc]
init];
    point.coordinate = CLLocationCoordinate2DMake
(mapa.centerCoordinate.latitude + 30,
mapa.centerCoordinate.longitude + 30);
    point.title = @"Meu ponto";
    [mapa addAnnotation:point];

    // TRIANGULO
    // Para fazer um triangulo, nós precisamos de 3 pontos
    int arrayCount = 3;
    CLLocationCoordinate2D coordsTriangulo[arrayCount];
    // coordenadas
    coordsTriangulo[1] = CLLocationCoordinate2DMake
(mapa.centerCoordinate.latitude * (-1.5),
mapa.centerCoordinate.longitude * (-1.5));
    coordsTriangulo[2] = CLLocationCoordinate2DMake
(mapa.centerCoordinate.latitude * (-1.5),
mapa.centerCoordinate.longitude * (1.5));
    coordsTriangulo[3] = CLLocationCoordinate2DMake
(mapa.centerCoordinate.latitude * (1.5),
mapa.centerCoordinate.longitude * (1.5));

    // Cria um poligono com as coordenadas
    MKPolygon* polygon = [MKPolygon
polygonWithCoordinates:coordsTriangulo count:arrayCount];

    // Adiciona o overlay no mapa
    [mapa addOverlay:polygon];

    // LINHA
    arrayCount = 2;
    CLLocationCoordinate2D coordsLinha[arrayCount];
    coordsLinha[1] = CLLocationCoordinate2DMake
(mapa.centerCoordinate.latitude - 80,
mapa.centerCoordinate.longitude + 100);
    coordsLinha[2] = CLLocationCoordinate2DMake
(mapa.centerCoordinate.latitude - 100,
mapa.centerCoordinate.longitude + 100);

```

```

    // Cria uma linha a partir das coordenadas
    MKPolyline* line = [MKPolyline
polylineWithCoordinates:coordsLinha count:arrayCount];

    [mapa addOverlay:line];

    // LINHA A PARTIR DE PONTOS DE LOCALIZAÇÃO
    arrayCount = 3;
    MKMapPoint points[arrayCount];

    // Cria um ponto de localização
    points[0] = MKMapPointMake(0, 0);
    points[1] = MKMapPointMake(104333312, 114333312);
    points[2] = MKMapPointMake(204333312, 304333312);

    // Cria uma linha a partir dos pontos
    MKPolyline* linePontos = [MKPolyline
polylineWithPoints:points count:arrayCount];
    [mapa addOverlay:linePontos];

}

#pragma mark MKMapViewDelegate

// Este método é chamado toda vez que um overlay é adicionado
no mapa
- (MKOverlayView *)mapView:(MKMapView *)mapView
viewForOverlay:(id <MKOverlay>)overlay {
    NSLog(@"Chamou viewForOverlay!");

    if ([overlay isKindOfClass:[MKCircle class]]){
        MKCircleView* circleView = [[MKCircleView alloc]
initWithOverlay:overlay] autorelease];

        circleView.strokeColor = [UIColor greenColor];
        circleView.lineWidth = 3.0;
        circleView.fillColor = [UIColor clearColor];
        return circleView;
    }
    if ([overlay isKindOfClass:[MKPolygon class]]){
        MKPolygonView* polyView = [[MKPolygonView alloc]
initWithOverlay:overlay] autorelease];
        polyView.strokeColor = [UIColor redColor];
        polyView.lineWidth = 3.0;
        return polyView;
    }
}

```

```

        if ([overlay isKindOfClass:[MKPolyline class]]){
            MKPolylineView* polyView = [[[MKPolylineView alloc]
initWithPolyline:overlay] autorelease];
            polyView.strokeColor = [UIColor blueColor];
            polyView.lineWidth = 3.0;
            return polyView;
        }

        return nil;
    }

// Este método é chamado toda vez que o mapa é movido ou
// redimensionado. para qualquer direção
-(void) mapView:(MKMapView *)mapView regionDidChangeAnimated:
(BOOL)animated {
    NSLog(@"Moveu mapa!");
    MKCircle* circle = [MKCircle
circleWithCenterCoordinate:mapa.centerCoordinate
radius:mapView.region.span.latitudeDelta * 10000];
    [mapa addOverlay:circle];
}

```

## ***Capítulo 12 - Envío de SMS***



A partir da versão 4.0 o IOS nos oferece uma interface padrão para redigir mensagens SMS. É muito simples de compor e enviar SMS, basta exibir o MFMessageComposeViewController, e implementar o método delegado

– `messageComposeViewController:didFinishWithResult:`

O Image I/O permite que o programador acesse informações da maioria dos formatos de imagens atualmente utilizados.

O primeiro passo, é imporar o framework MessageUI.framework, e no arquivo de interface importar o header

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

A seguir utilizamos o método de classe “canSendText” para verificar se o aparelho está habilitado a enviar mensagens SMS. Caso ele esteja habilitado podemos exibir o controller padrão que irá nos permitir redigir e enviar as mensagens SMS. Abaixo segue um trecho de código de exemplo:

```
-(IBAction)enviarSMS{
    if ([MFMessageComposeViewController canSendText]) {
        MFMessageComposeViewController* composer =
        [[MFMessageComposeViewController alloc] init];

        composer.messageComposeDelegate = self;
        [self presentViewController:composer
        animated:YES];
        // [composer release];
    }
    else {
        UIAlertView* alert = [[UIAlertView alloc]
        initWithTitle:@"Erro" message:@"Seu device não manda SMS"
        delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
}
```

```

// Este método delegado é chamado todas as vezes que o
controller sair da tela.
- (void)messageComposeViewController:
(MFMessageComposeViewController *)controller
didFinishWithResult:(MessageComposeResult)result {
    [self dismissModalViewControllerAnimated:YES];

    NSString* message;

    if (result == MessageComposeResultFailed)
        message = @"Deu erro";
    else if (result == MessageComposeResultSent)
        message = @"Mensagem enviada";
    else if (result == MessageComposeResultCancelled) {
        message = @"Envio Cancelado";
    }

    UIAlertView* alerta = [[UIAlertView alloc]
initWithTitle:@"Mensagem" message:message delegate:self
cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alerta show];
    [alerta release];
}

```