Partner Documentation

Mobclix SDK Start Guide for iOS 5.0.1

Release Date: 5/3/2012

Table of Contents

Introduction

This document describes how to use the Mobclix SDK for iOS to build and customize applications.

In This Document

This document describes how to:

- add the Mobclix SDK files to your project
- set linker flags
- integrate with or without Interface Builder
- Integrate Your Code and set up App Delegate
- Work with ads

If you want to…	See...
Get started and add Mobclix SDK	*Adding the Mobclix SDK files to your Project*
Set linker flags	*Setting Linker Flags*
Integrate Code and Understand Interface Builder	*Integrating with Interface Builder*
Understand and Work with ads	*Managing and Working With Ads*
Full-Screen Ads	*Integrating Full-Screen Ads*

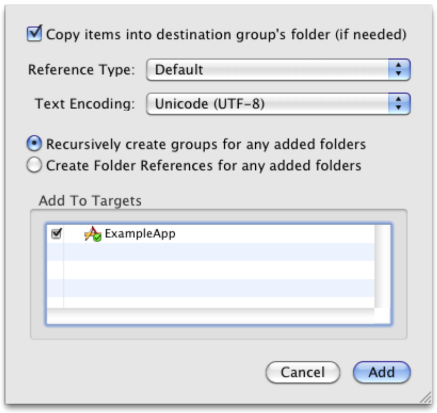
Getting Started

Before you start, you should have:

change.log	A list of changes that have occurred to the SDK in each release
Mobclix SDK	Contains the Mobclix for iOS SDK that most developers will use
Mobclix SDK (Non-Thumb)	Contains the Mobclix for iOS SDK for developers who have disabled “Compiled for Thumb” in their build options. If you don’t know what this means, you likely don’t need this version of the SDK. It’s commonly used in OpenGL based applications.
Demo Application	Fully functional iOS application that integrates all features of the Mobclix SDK.
Integration Guide	Documentation to help new developers integrate the Mobclix SDK for iOS into their applications.

Adding the Mobclix SDK files to your Project

1. Open up your iOS project in Xcode and drag the Mobclix SDK folder into the **Groups & Files** section.
2. When the dialog box appears, check the Copy Items into destination group’s folder checkbox.
3. In **Add To Targets**, make sure the checkbox next to your application is also checked.



☑ Copy items into destination group's folder (if needed)

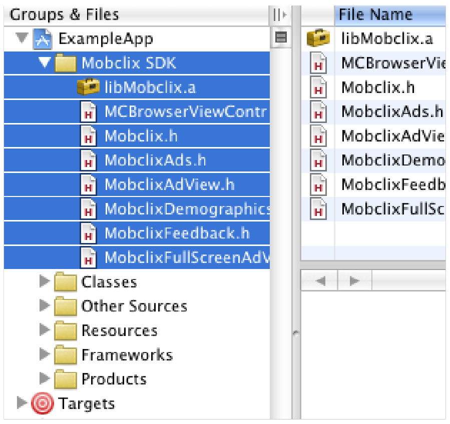
Reference Type: Default

Text Encoding: Unicode (UTF-8)

[✓ Recursively create groups for any added folders] Create Folder References for any added folders

Add To Targets

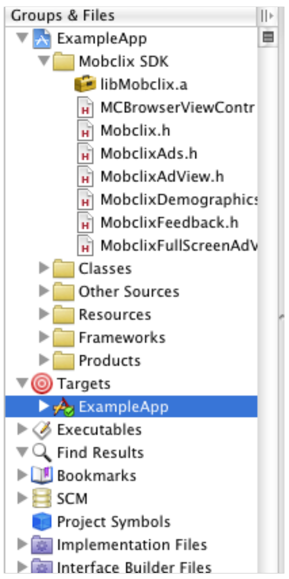
☑ ExampleApp



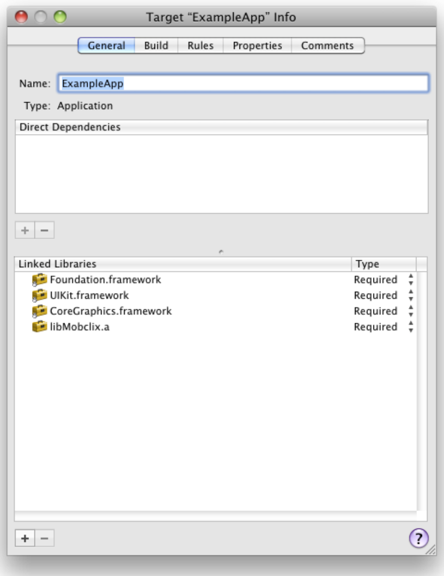
Note: While libMobclix.a is roughly 9MB in size, the actual added size to your application will only be a few hundred KB per architecture.

Linking to the Dependency Frameworks

There are a few required framework dependencies for the Mobclix SDK 4.2 for iOS. These will not add any additional size to your application since they're already on the device, but you'll need to link against them.



1. In Xcode, select your applications target. This can be found in the **Groups & Files** pane**.** Find the group labeled **Targets**, accompanied by a red bullseye image, and then



Name: ExampleApp
Type: Application

Linked Libraries	Type
Foundation.framework	Required
UIKit.framework	Required
CoreGraphics.framework	Required
libMobclix.a	Required

expand it.

2. Select your application and hit **command+i** on your keyboard. This will reveal the "**Get Info**" window.
3. In the dialog box that appears, click the **General** tab, then click the plus icon in the bottom left corner under the Linked Libraries section.
4. Select the following frameworks, then click **Add** at the bottom right. You can select multiple frameworks by holding down the command key.

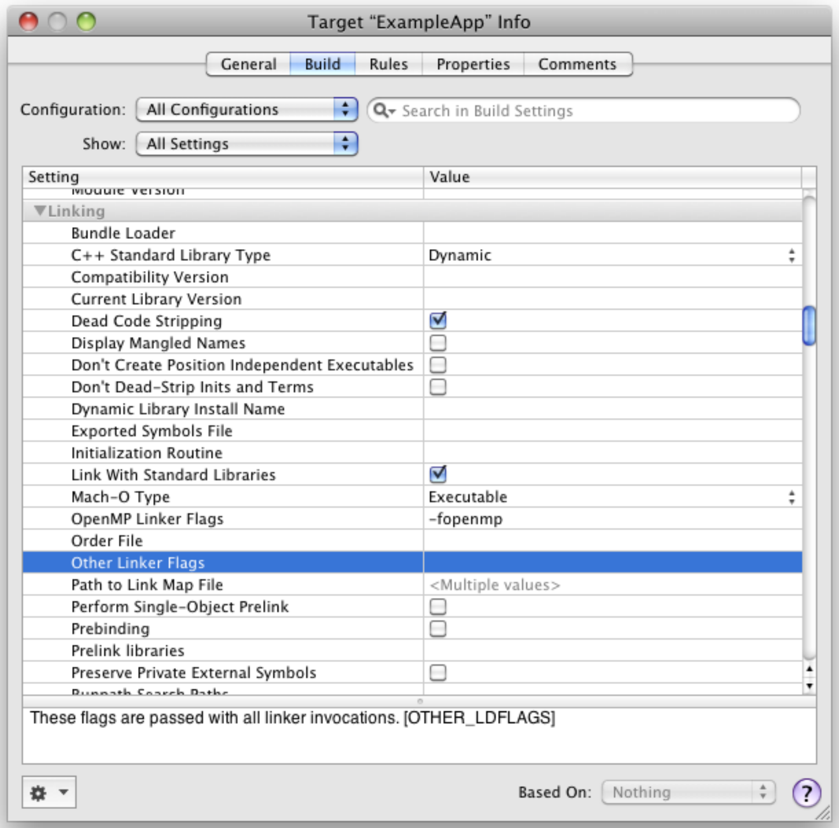
AddressBook	AVFoundation	MediaPlayer	SystemConfiguration
AddressBookUI	CoreMotion	MessageUI	
AudioToolbox	EventKit	QuartzCore	

The standard UIKit, Foundation, and CoreGraphics frameworks are also required. If you've removed any of them from your project, you'll need to re-add them.

When targeting 3.x devices, the CoreMotion and EventKit frameworks should be weak-linked. To do this, switch the "required" option next to each framework to "weak."

Setting Linker Flags

The next thing you'll need to do is add the "-ObjC" linker flag to your target.



Configuration: All Configurations

Show: All Settings

Setting	Value
▼Linking	
Bundle Loader	
C++ Standard Library Type	Dynamic
Compatibility Version	
Current Library Version	
Dead Code Stripping	☑
Display Mangled Names	☐
Don't Create Position Independent Executables	☐
Don't Dead-Strip Inits and Terms	☐
Dynamic Library Install Name	
Exported Symbols File	
Initialization Routine	
Link With Standard Libraries	☑
Mach-O Type	Executable
OpenMP Linker Flags	-fopenmp
Order File	
Other Linker Flags	
Path to Link Map File	<Multiple values>
Perform Single-Object Prelink	☐
Prebinding	☐
Prelink libraries	
Preserve Private External Symbols	☐

These flags are passed with all linker invocations. [OTHER_LDFLAGS]

1. In the **Get Info** dialog box that you used to add the required frameworks, select the **Build** tab and scroll to **Other Linker Flags** in the **Linking** section.

2. Double click **Other Linker Flags** and another dialog box will appear. Click the plus icon (+)and enter **-ObjC**.

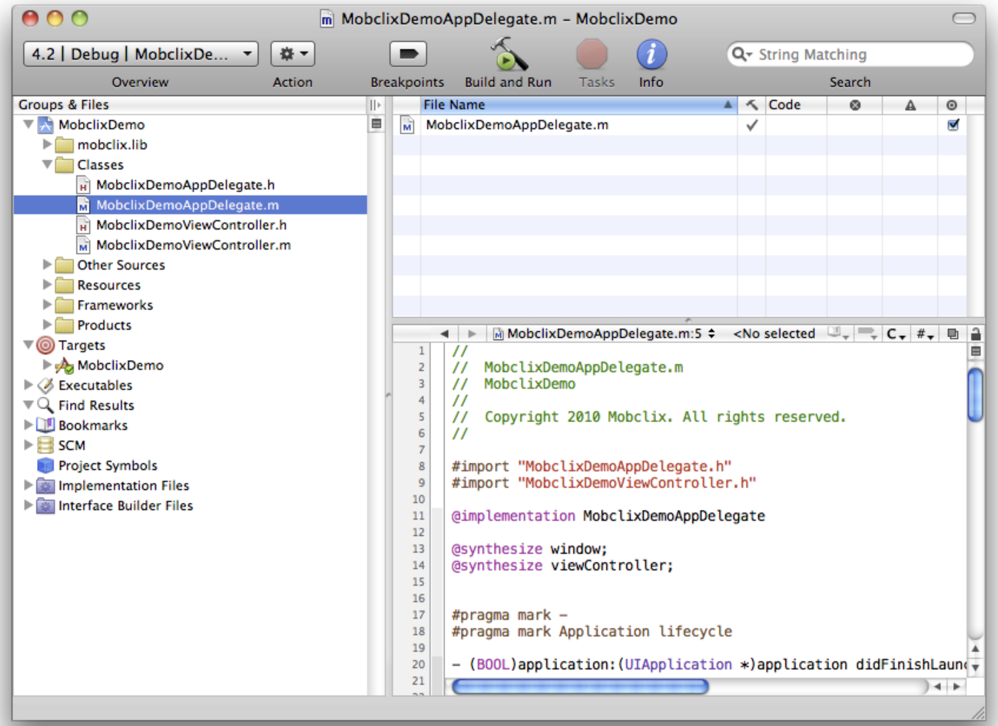
3. Click **OK** to save your changes.

Note: Make sure **All Configurations** is selected in the **Configuration** drop down, otherwise you'll only set it for one configuration.

Integrating with Your Code

Setting up your App Delegate

You will need to “start” Mobclix in your App Delegate. In most cases, this file is called



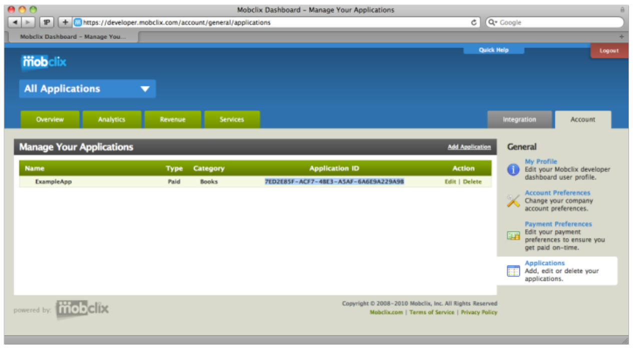
```
//
//  MobclixDemoAppDelegate.m
//  MobclixDemo
//
//  Copyright 2010 Mobclix. All rights reserved.
//

#import "MobclixDemoAppDelegate.h"
#import "MobclixDemoViewController.h"

@implementation MobclixDemoAppDelegate

@synthesize window;
@synthesize viewController;


#pragma mark -
#pragma mark Application lifecycle

- (BOOL)application:(UIApplication *)application didFinishLaun
```

AppNameAppDelegate.m and found in the **Groups & Files** pane under **Classes**.

1. At the top of this file you should add #import "Mobclix.h".
2. The only other line you’ll need to add to this file is [Mobclix startWithApplicationId:]. Depending on your application, this should go at the bottom of the applicationDidFinishLaunching: method, or application:didFinishLaunchingWithOptions: method. Here is an example:

```
#import "ExampleAppAppDelegate.h"
#import "RootViewController.h"

// Make sure to import the Mobclix header file
#import "Mobclix.h"

@implementation ExampleAppAppDelegate

            (void)applicationDidFinishLaunching:(UIApplication*)application {
// Add this line to start up Mobclix
[Mobclix startWithApplicationId:@"insert-your-application-id"];

// Add this line to start up Mobclix
[window addSubview:[navigationController view]];
[window makeKeyAndVisible];
}
```

3. You should set the value of this ID to your Mobclix Application ID, found in the Mobclix Dashboard by selecting the **Account** tab and then clicking into the **Applications** section.



Name	Type	Category	Application ID	Action
ExampleApp	Paid	Books	7ED2E85F-ACF7-48E3-A5AF-6A6E9A229A98	Edit | Delete

My Profile
Edit your Mobclix developer dashboard user profile.

Account Preferences
Change your company account preferences.

Payment Preferences
Edit your payment preferences to ensure you get paid on-time.

Applications
Add, edit or delete your applications.

Copyright © 2008-2010 Mobclix, Inc. All Rights Reserved
Mobclix.com | Terms of Service | Privacy Policy

4. If you haven't added your application to Mobclix, now would be a great time to do so. You can follow the steps found here. You'll receive your Application ID when you're finished adding it.

5. Next, you'll need to integrate the ads into your application. You can do this with either Interface Builder or by initializing the views yourself. We have outlined both methods in this document.

Integrating with Interface Builder

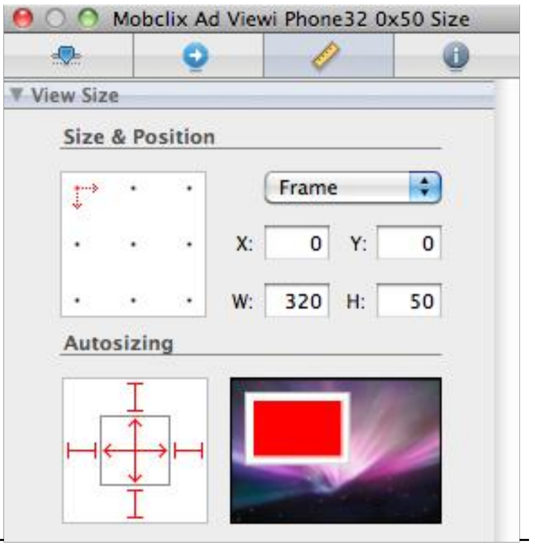
1. Set up your Interface file (YourViewController.h) with “adView” as an outlet, as shown in the following:

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

@interface RootViewController : UIViewController {
@private
        MobclixAdView* adView;
}

@property(nonatomic,retain) IBOutlet MobclixAdView* adView;
@end
```

2. Next, you should synthesize the adView property that was set-up in your Implementation file (YourViewController.m).

```
#import "RootViewController.h"


@implementation RootViewController
@synthesize adView;
```

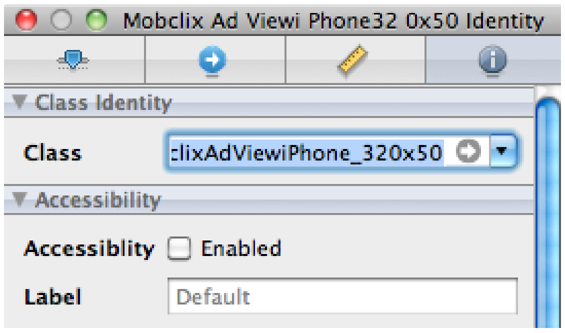
3. Set up your Interface Builder file (usually YourViewController.xib) with an ad view.
4. Double click **View** to bring your main view into focus.
5. Drag a new View from your library window onto the main view.
6. Make sure the newly created view is selected and open the **Inspector** window. Select the **Ruler** tab and change the width to **320** and the height to **50**.
7. Drag the view to whereever you’d like it to appear on the screen. You should also update the autoresizing masks and disable



X: 0 Y: 0
W: 320 H: 50

flexible width and flexible height. You can do this by clicking each solid line, so the only solid line left is the top one. The Inspect window opens.

8. In the Inspect window, select the tab with the “i” icon. Change the class to “MobclixAdViewiPhone_320x50” as shown in the following:


Mobclix Ad Viewi Phone32 0x50 Identity

Class Identity

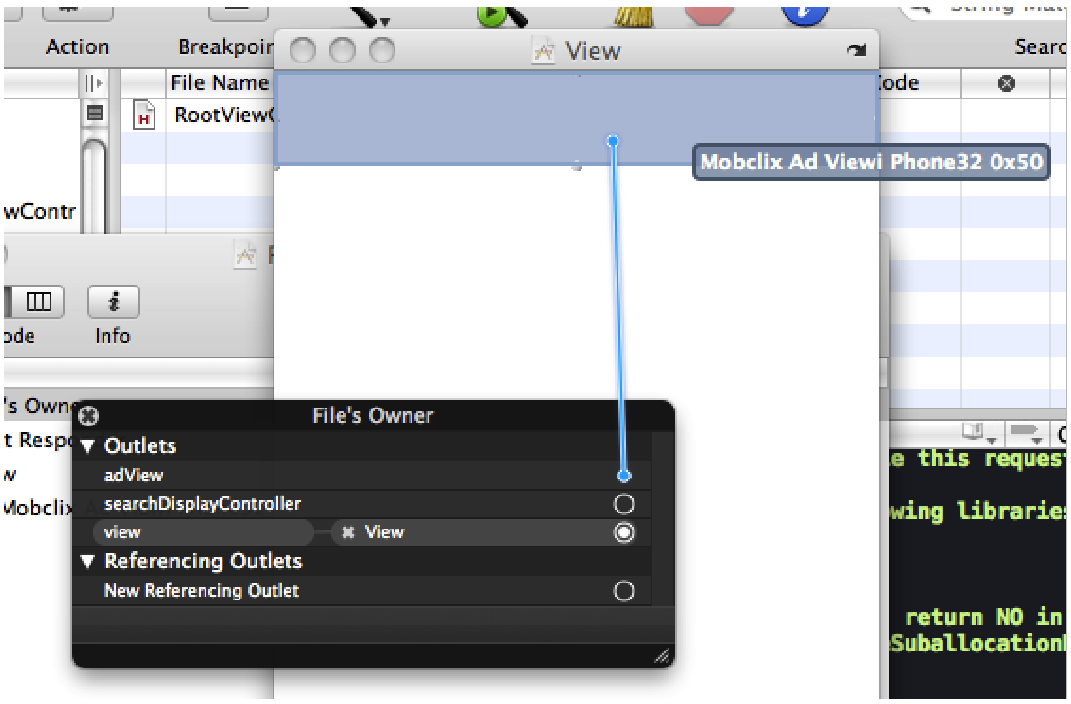
Class: :lixAdViewiPhone_320x50

Accessibility

Accessiblity: Enabled

Label: Default

9. In the Document window, right click your view controller, and drag the circle to the ad view you created in your main view, as shown in the following:



10. You can now save and close your Interface Builder file. You can go to the **Requesting and Managing Ads** section to start working with your ads

Integrating without Interface Builder

For more advanced implementations of ads, you'll want to create and place the views in your code. We're going to show you a basic implementation using viewDidLoad. Integrating this way is straightforward and only requires three steps:

1. Set up your Interface file (YourViewController.h) with "adView", as shown below:

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

@interface RootViewController : UIViewController {
@private
        MobclixAdView* adView;
}

@property(nonatomic,retain) MobclixAdView* adView;
@end
```

2. Next, you should synthesize the adView property that was set up in your Implementation file (YourViewController.m).

```
#import "RootViewController.h"


@implementation RootViewController
@synthesize adView;
```

3. Now all you need to do is create the view and add it to your view hierarchy, as shown:.

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

        self.adView = [[[MobclixAdViewiPhone_320x50 alloc]
initWithFrame:CGRectMake(0.0f, 0.0f, 320.0f, 50.0f)] autorelease];
        [self.view addSubview:self.adView];
}
```

Managing and Working With Ads

It is very important to request ads properly. If you don't follow the instructions properly, you will have lower click through rates, which will lead to lower CPM's.

Automatically request new ads

In your view controller's viewDidAppear: method, you should start the ad refresh timer by calling resumeAdAutoRefresh. If this is the first time you call this method, it will automatically call getAd. Subsequent calls to this method will reschedule the auto refresh timers.

```
- (void)viewDidAppear:(BOOL)animated {
        [super viewDidAppear:animated];
        [self.adView resumeAdAutoRefresh];
}
```

Pause ad requests when your view disappears

When your view goes off screen, you should pause the ad refresh timers. Do this by calling pauseAdAutoRefresh. If you don't pause ads when your view goes off screen, you will not only lower your CPMs and CTRs, you may impair your application's overall performance.

```
- (void)viewWillDisappear:(BOOL)animated {
        [super viewWillDisappear:animated];
        [self.adView pauseAdAutoRefresh];
}
```

Properly canceling and releasing your ad view

The number one cause of crashes when using ads is developers not properly canceling and releasing the ad view when their view controller unloads or terminates. When your view unloads or your view controller deallocates, three calls should be made:

1. Call cancelAd to stop any current ads from loading, and cancel the ad refresh timers.
2. Set your ad view's delegate to nil. This is important, as your ad view might not deallocate immediately and could send a delegate message to a released object if you don't set it to nil.
3. Release your ad view.

A correctly implemented example:

```
@implementation RootViewController

/* ... */

- (void)viewDidUnload {
        [self.adView cancelAd];
        self.adView.delegate = nil;
        self.adView = nil;
}


- (void)dealloc {
        [self.adView cancelAd];
        self.adView.delegate = nil;
        self.adView = nil;

    [super dealloc];
}


@end
```

Integrating Full Screen Ads

You can integrate Full Screen ads two different ways:

1. You can pre-request the ad and then display it at an appropriate point in your application or game
2. You can call requestAndDisplay and have it display as soon as it loads

Pre-Requesting Ads, Then Displaying

Integrating full screen ads by pre-requesting requires a more extensive code and integration process than using a requestAndDisplay call. Pre-requesting does result in a much more favorable user experience because you can ensure the user will see the full screen ad before they've started interacting with the new section of your application or game.

1. Setup your Interface file (YourViewController.h) with “adView” as an outlet, as shown below:

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

@interface RootViewController : UIViewController {
@private
        MobclixFullScreenAdViewController* fullScreenAdViewController;
}

@property(nonatomic,retain) MobclixFullScreenAdViewController*
fullScreenAdViewController;
@end
```

2. Next, you should synthesize the fullScreenAdViewController property that was setup above, in your Implementation file (YourViewController.m).

```
#import "RootViewController.h"


@implementation RootViewController
@synthesize fullScreenAdViewController;
```

Now you can setup the fullScreenAdViewController in viewDidLoad and request an ad

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

         self.fullScreenAdViewController = [[[MobclixFullScreenAdViewController alloc] init]
autorelease];
         self.fullScreenAdViewController.delegate = self;
         [self.fullScreenAdViewController requestAd];
}
```

3. Next, when a user triggers an action that you wish to provide an intestitial for, you can display your already requested ad:

```
- (IBAction)someAction:(UIButton*)button {
         if([self.fullScreenAdViewController hasAd]) {
         [self.fullScreenAdViewController displayRequestedAdFromViewController:self];
         }
}
```

4. You'll need to setup a delegate callback to automatically request new ads once a user presses one:

```
-
(void)fullScreenAdViewControllerDidDismissAd:(MobclixFullScreenAdViewController*)fSA
VC {
         [self.fullScreenAdViewController requestAd];
}
```

Requesting and Automatically Displaying Ads

This method is simpler to implement than pre-requesting. It can, however, result in a poor user experience because when ads are displayed as soon as they're loaded, it might be after the user has already begun interacting with the new section that triggered the full screen ad.

1. Setup your Interface file (YourViewController.h) with "adView" as an outlet, as shown below:

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

@interface RootViewController : UIViewController {
@private
        MobclixFullScreenAdViewController* fullScreenAdViewController;
}

@property(nonatomic,retain) MobclixFullScreenAdViewController*
fullScreenAdViewController;
@end
```

2. Next, you should synthesize the fullScreenAdViewController property that was setup above, in your Implementation file (YourViewController.m).

```
#import "RootViewController.h"


@implementation RootViewController
@synthesize fullScreenAdViewController;
```

3. Now you can setup the fullScreenAdViewController in viewDidLoad and request an ad

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

        self.fullScreenAdViewController = [[[MobclixFullScreenAdViewController alloc] init]
autorelease];
}
```

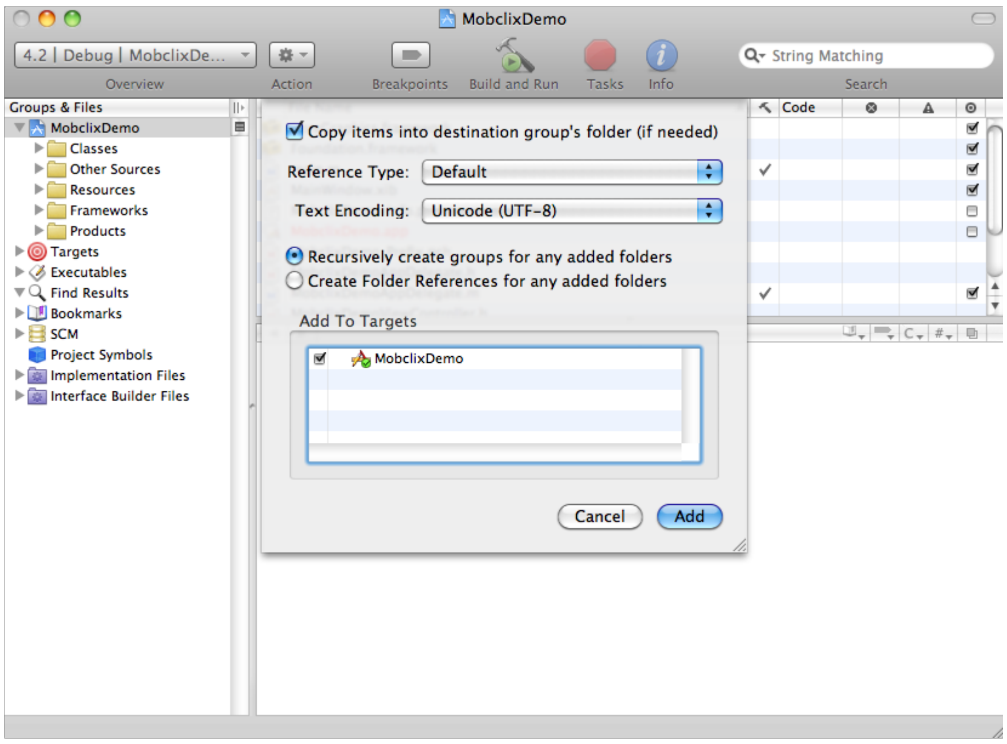
4. Next, when a user triggers an action that you wish to provide an interstitial for, you can automatically request and then display the ad:

```
- (IBAction)someAction:(UIButton*)button {
        [self.fullScreenAdViewController requestAndDisplayAdFromViewController:self];
}
```

Open Allocation -and Adding additional SDK's to your App

The Mobclix iOS SDK enables you to utilize other Ad SDKs within a single management framework. The Mobclix Developer Dashboard enables you to remotely change the allocation behavior of the Mobclix SDK after your app goes live. It is easy to allocate a percentage of the ad requests from the SDK to Mobclix, iAd , Admob, Millennial or your own code. Open allocation involves integrating each of the non-Mobclix ad networks' SDKs into your application. The first step therefore is adding the Ad Network's iOS library or SDK into your application. Since the integration of each ad network SDK is different and constantly changing, you should refer to specific integration guides for how to properly add an SDK to your project. However, in general the way to achieve this is:

1. Open up your iOS project in Xcode and drag the folder containing the library or SDK into the section on the left titled **Groups & Files**. A dialog box will appear. **Copy**



☑ Copy items into destination group's folder (if needed)

Reference Type: Default

Text Encoding: Unicode (UTF-8)

◉ Recursively create groups for any added folders

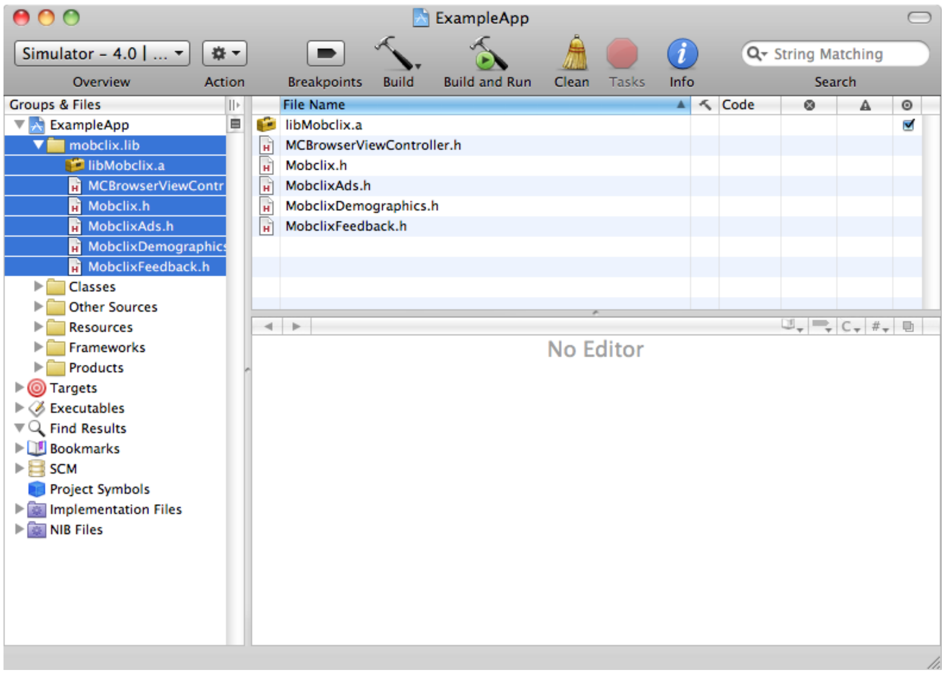
○ Create Folder References for any added folders

Add To Targets

☑ MobclixDemo

Items into destination group's folder

2. Select the checkbox next to your Application in the **Add To Targets** section. The SDK folder in the **Group & Files pane** will open.



Ad Network Adapter - Adding AdMob

AdMob requires that two things be added to your application, their iOS SDK as well as a library named TouchJSON. Both are included when downloading the AdMob SDK from the Google website

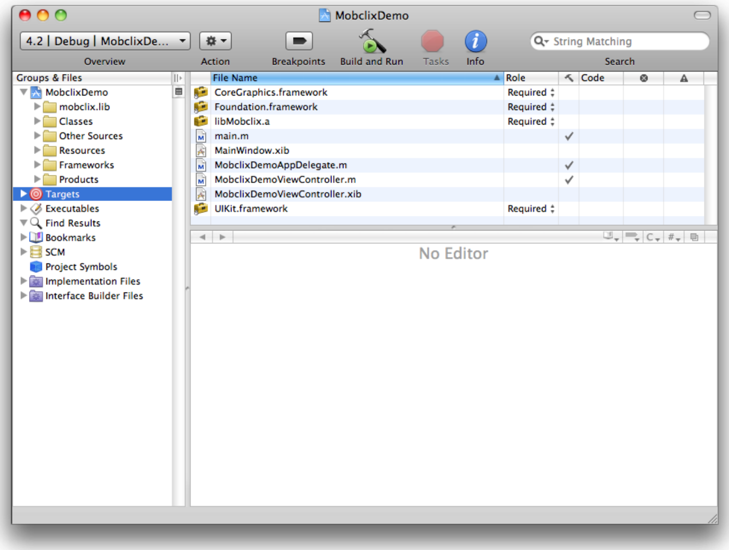
Ad Network Adapter - Adding Millennial Media

To integrate Millennial Media, you need to download the MMSDK from their website and drag it into your Xcode project. Your Millennial Key is set on the developer dashboard, you **do not** need to implement the **publisherKeyForSuballocationRequest**: callback method. When signing up for the Millennial Media network on the Dashboard (by clicking the "Services" tab and then navigating to the "Networks" page), you'll be asked to enter your APID. You can change the APID by going to the **Networks** page later and clicking the **Settings** button under the Open Millennial Media network on the **Networks** page. Be sure to select the proper platform and unit when signing up/editing your Millennial Media network settings.

Ad Network Adapter and Adding iAd

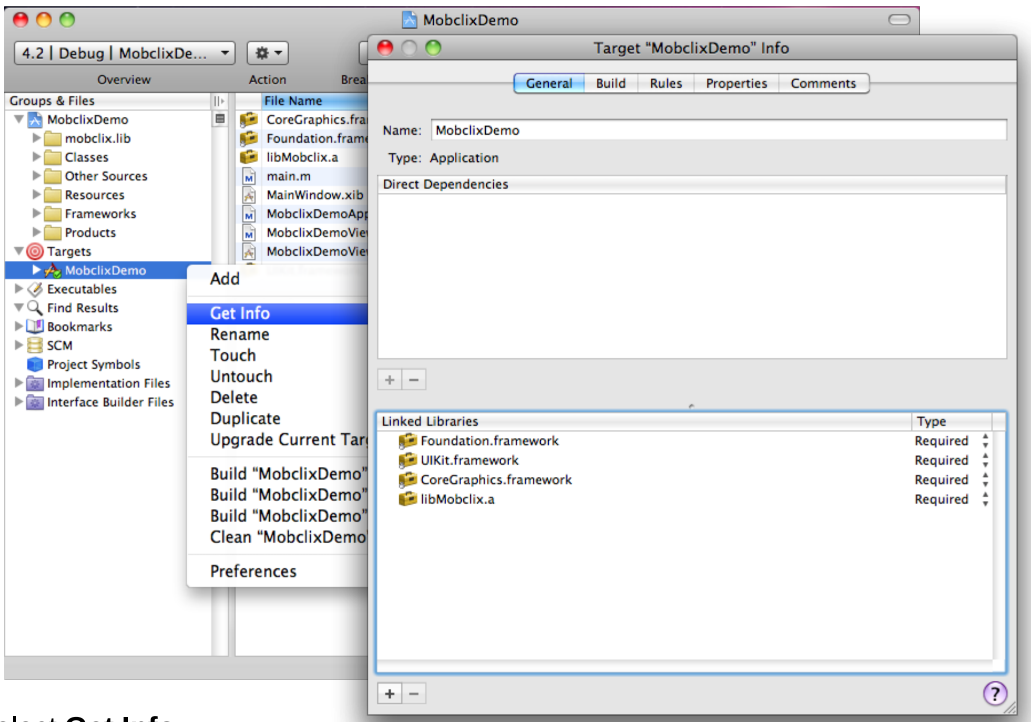
iAd doesn't actually require you to copy a library into your application, instead all you need to do is add the iAd.framework. To do this:

1. In Xcode, in the "Groups & Files" pane on the left side find the group labeled "Targets" accompanied by a bull's-eye image



File Name	Role		Code		
CoreGraphics.framework	Required				
Foundation.framework	Required				
libMobclix.a	Required				
main.m		✓			
MainWindow.xib					
MobclixDemoAppDelegate.m		✓			
MobclixDemoViewController.m		✓			
MobclixDemoViewController.xib					
UIKit.framework	Required				

Expand it by clicking on the arrow next to the bullseye, find your application then right click



on it and select **Get Info**

2. At the top of the dialog box that appears, click the **General** tab, then click the plus icon in the bottom left corner under the section titled **Linked Libraries**.

3. Select iAd.framework, then click **Add**.

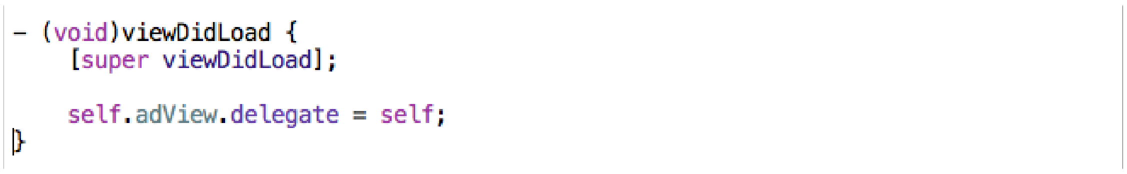
Implementing MobclixAdView Delegate Methods

When the Mobclix SDK receives a message from the Mobclix servers that an open allocation network should be used to show an advertisement it looks for specific delegate methods for instructions on how to proceed. These methods should all be defined within the class that is serving as the MobclixAdViewDelegate (usually the UIViewController containing the ad).

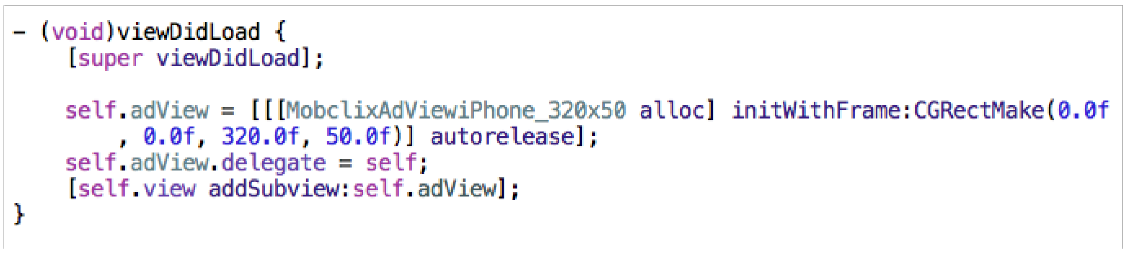
NOTE: If you haven't previously set the delegate for your Mobclix Ad View, you should do that now. If you implemented ads as shown in the Mobclix SDK Quick Start Guide all you'll need to do is add the line

self.adView.delegate = self;

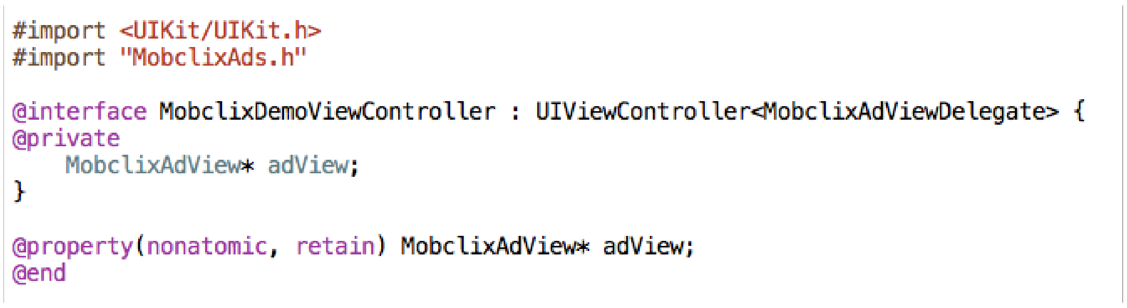
after the adView initialization. Therefore if you integrated using Interface Builder your viewDidLoad function will look like:



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

    self.adView.delegate = self;
}
```



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

    self.adView = [[[MobclixAdViewiPhone_320x50 alloc] initWithFrame:CGRectMake(0.0f
        , 0.0f, 320.0f, 50.0f)] autorelease];
    self.adView.delegate = self;
    [self.view addSubview:self.adView];
}
```



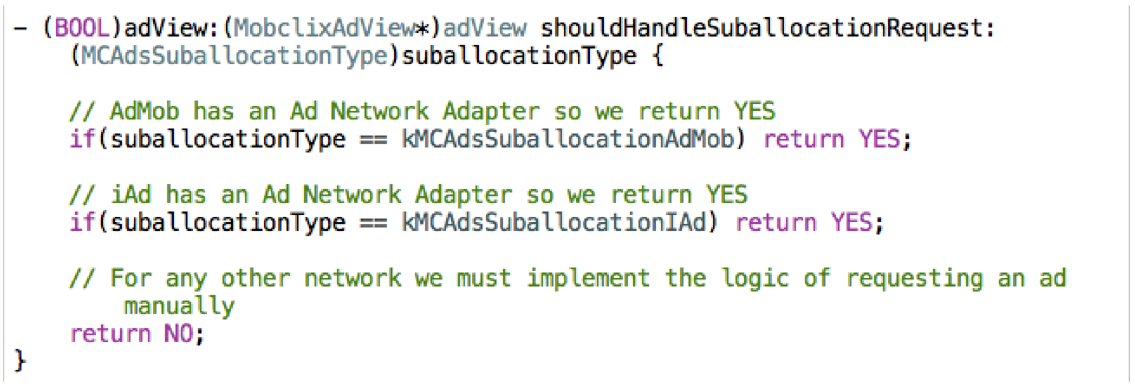
```
#import <UIKit/UIKit.h>
#import "MobclixAds.h"

@interface MobclixDemoViewController : UIViewController<MobclixAdViewDelegate> {
@private
    MobclixAdView* adView;
}

@property(nonatomic, retain) MobclixAdView* adView;
@end
```

Lastly, to prevent compiler warnings you should register the class you're using as the delegate as implementing the MobclixAdViewDelegate protocol. This is done by adding <MobclixAdViewDelegate> in the interface declaration in the header (.h) file as shown below.

The shouldHandleSuballocationRequest Method

This is the first method which is called when the Mobclix SDK receives a message to show an advertisement from an open allocation network. This method determines whether the SDK should try to automatically call the appropriate network (available for networks with an Ad Network Adapter **only**) or if you wish to manually handle the request with your own code.

This method should look something like the code below (with one or more lines removed depending on the open allocation networks you are implementing)



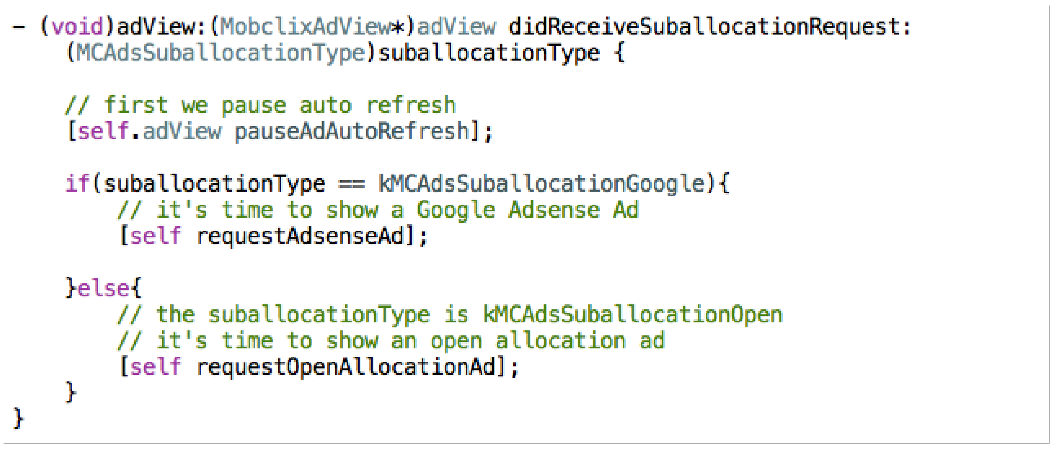
```
- (BOOL)adView:(MobclixAdView*)adView shouldHandleSuballocationRequest:
    (MCAdsSuballocationType)suballocationType {

    // AdMob has an Ad Network Adapter so we return YES
    if(suballocationType == kMCAdsSuballocationAdMob) return YES;

    // iAd has an Ad Network Adapter so we return YES
    if(suballocationType == kMCAdsSuballocationIAd) return YES;

    // For any other network we must implement the logic of requesting an ad
        manually
    return NO;
}
```

The didReceiveSuballocationRequest for Networks Without Ad Network Adapters

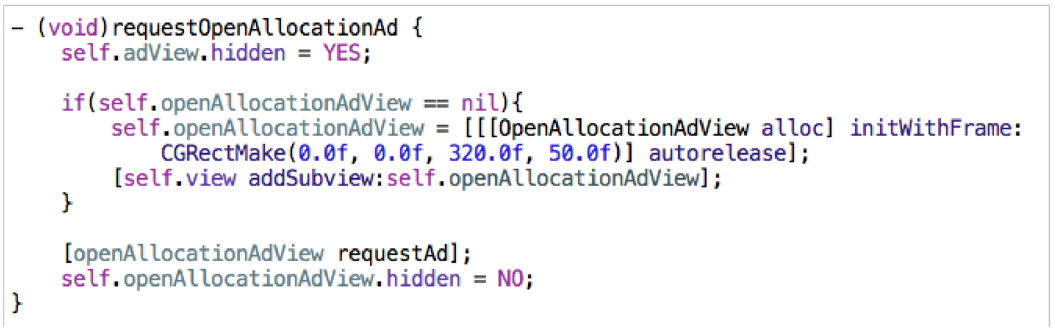
This method is called when NO is returned in shouldHandleSuballocationRequest (i.e. for networks without an Ad Network Adapter). Within this method should be the logic to request and display an ad from the appropriate network as well as pause the advertisement auto refresh until it's time to show a new ad. From a high level the logic should be as follows:

1. Pause autorefresh
2. Request an ad from the appropriate ad network
3. Once an ad is returned, hide the mobclix adView and show the one from the open allocation network
4. Once the ad has been shown for an appropriate amount of time, resume auto-refresh, hide the open allocation adView and show the mobclix one

This method should look something like the code below:



```
- (void)adView:(MobclixAdView*)adView didReceiveSuballocationRequest:
    (MCAdsSuballocationType)suballocationType {

    // first we pause auto refresh
    [self.adView pauseAdAutoRefresh];

    if(suballocationType == kMCAdsSuballocationGoogle){
        // it's time to show a Google Adsense Ad
        [self requestAdsenseAd];

    }else{
        // the suballocationType is kMCAdsSuballocationOpen
        // it's time to show an open allocation ad
        [self requestOpenAllocationAd];
    }
}
```



```
- (void)requestOpenAllocationAd {
    self.adView.hidden = YES;

    if(self.openAllocationAdView == nil){
        self.openAllocationAdView = [[[OpenAllocationAdView alloc] initWithFrame:
            CGRectMake(0.0f, 0.0f, 320.0f, 50.0f)] autorelease];
        [self.view addSubview:self.openAllocationAdView];
    }

    [openAllocationAdView requestAd];
    self.openAllocationAdView.hidden = NO;
}
```

and requestAdsenseAd / requestOpenAllocationAd looking something like:

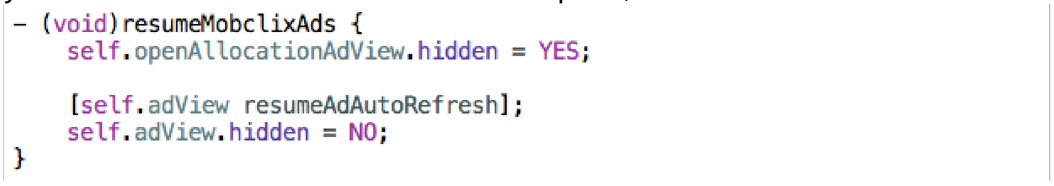
Once the open allocation advertisement has been shown for an appropriate amount of time a method like the following should be called to resume requesting ads.

continueRequest

In the case that you have returned NO to the shouldHandleSuballocationRequest method, attempted to request an Ad from another source, and then failed to receive an Ad, you can tell the Mobclix SDK to continue with the next configured Open Allocation Network (configured by the Mobclix Dashboard) immediately.

Mobclix Developer Dashboard.
For example, given the following priority order:
1. Open Allocation: AdMob
2. Open Allocation: Other
3. Mobclix

If you handle the AdMob request manually and receive no ad then you can call continueRequest on the Adview in order to immediately resume with the next candidate. If you choose to use our default network adapters, this continuation is the default behavior.



```
- (void)resumeMobclixAds {
    self.openAllocationAdView.hidden = YES;

    [self.adView resumeAdAutoRefresh];
    self.adView.hidden = NO;
}
```

The publisherKeyForSuballocationRequest for AdMob

This method is part of the AdMob Ad Network Adapter and is called when the server returns saying that it's time to show an AdMob ad. Within this method, you should return your application's AdMob Publisher ID.

Additional Support

If you have any additional questions, there are a few more resources available to you:

- The Mobclix Example Application, found here, shows the Mobclix SDK fully integrated and running.
- You can view the Mobclix SDK Documentation online for a complete overview of the SDK and all classes and methods available.
- The header files provided with the Mobclix SDK contain documentation above each method.
- The Mobclix iPhone & iPad SDK Google Group is a great way to interact with other developers using the Mobclix SDK as well as asking a question directly to the engineers at Mobclix.

If you have any other questions, send us an email at support@mobclix.com and we'll be happy to help!