Getting Started with the Mocean iOS SDK

Mocean Mobile is unlike any other mobile ad serving platform available. Developed specifically for mobile devices, the Mocean Mobile Ad Serving Technology streamlines the many moving parts in mobile advertising for publishers, app stores, and networks. Mocean Mobile was built by mobile advertising experts so that the real opportunity of this exciting new media could be fully harnessed**.**

The Mocean iOS SDK makes it easy for developers to incorporate mobile ads into iOS applications. This introduction to the SDK is written for iOS developers new to the advertising platform, and gives developers a quick introduction to the first ad integration project.

Contents

[1. Prerequisites 2](#_Toc328658052)

[2. User Interface / Layout (Design) 2](#_Toc328658053)

[3. Creating Ad View 3](#_Toc328658054)

[4. Getting Initial Ad View Content 5](#_Toc328658055)

[5. Creating an Interstitial Ad View 6](#_Toc328658056)

[6. Handling Rotation Changes 7](#_Toc328658057)

[7. Customizing the Viewport 8](#_Toc328658058)

[8. Detecting Ad Load Failures 9](#_Toc328658059)

[8.1 Important note when using a delegate 10](#_Toc328658060)

[9. Where to go next 11](#_Toc328658061)

## Prerequisites

Install the iOS ad SDK available from

<http://code.google.com/p/mocean-sdk-ios/downloads/list>

as described in documentation. Once complete, you should have an open iOS project:

* Referencing the public MASTAdView.h and MASTAdViewDelegate.h headers
* Referencing the AdMobileSDK.xcodeproj project
* The SDK’s libAdMobileSDK.a added to the “Link Binary With Libraries” section in the target’s Build Phases tab (this will also add it to the Target Dependencies section)
* At least one controller and view where the ad view will be placed

If you are not comfortable with iOS development, we suggest you review the online iOS developer documentation available at: <https://developer.apple.com/devcenter/ios/>.

You will find more thorough, complex examples and additional use cases in the sample application distributed with the SDK. Both the sample app and the SDK itself are available in source code form from our Google Code project site at <http://code.google.com/p/mocean-sdk-ios/>.

You can also find additional documentation, information, and other supported platforms on our developer wiki at: <http://developer.moceanmobile.com/Main_Page>.

## User Interface / Layout (Design)

The first step is deciding where you want to incorporate ads in your application.

The simplest approach is to integrate a small horizontal banner ad into the user-interface (UI). A typical form factor is a 50-pixel tall (which would equal 100 pixels of the display for Retina devices), full width rectangle which does not crowd the existing UI elements or break the appearance and flow. As an example, consider the following Flickr image viewer before and after a banner ad has been inserted. We will show the steps to setup and display this ad below.

|  |
| --- |
|  |

Figure 2 - UI with Top Banner Ad

Figure 1 - Original UI

## Creating Ad View

Once you know where you want to put an ad in your UI, the next step is to create the ad view. As is typical with any iOS UIView, the ad view can be added to any other view. Since the ad view may need to be managed beyond just being a simple view in the stack it should be retained (or for ARC, a strong reference) in the controller. This allows updating, delegate callbacks and any other maintenance of the ad. It also keeps the ad from being deallocated when the view is cleaned up either due to low memory conditions or navigation tasks that remove the controller’s view from the main window (or other parent view like the navigation controller or tab bar controller).

Import the headers for the SDK into the project implementation where the ad view will be used:

|  |
| --- |
| #import “MASTAdView.h”  #import “MASTAdViewDelegate.h” |

Figure 3.1 – Header Import

In the class interface declare a new property or variable to hold the ad view object. This can be done in the header or implementation so long as it’s in a @interface block. For ARC, use strong vs. retain.

|  |
| --- |
| // Property to hold the ad view object, added to the controller interface  @property (nonatomic, retain) MASTAdView\* adView; |

Figure 3.2 – Interface Property

In the class implementation, synthesize the new property or create accessor and setter methods:

|  |
| --- |
| // Synthesize the new property  @synthesize adView; |

Figure 3.3 – Implementation Property

The dealloc method should handle setting the delegate to nil and releasing the ad view. If there are no plans to use the delegate it can be left alone, but it’s good practice to nil delegates when releasing their owning object. Also, by using the property set to nil the code is compatible with non-ARC and ARC based projects since release is not directly called.

|  |
| --- |
| // Nil the delegate, using the setter method in case adView is nil.  [self.adview setDelegate:nil];  // Release the adView by invoking it’s setter to do the work  self.adView = nil; |

Figure 3.4 – Implementation for –(void)dealloc

Finally, create and initialize the ad view. Also set some basic properties. Depending on how the controller is created this can be done in init, loadView or viewDidLoad (or anywhere else if need be). Depending on how the controller is created the implementation may need to ensure that the existing ad view is cleaned up (see dealloc code) before creating a new (or replacement) ad view. The code below assumes the adView object is being set for the first and only time in the controller. It also places the ad view at the top of the controller. Other content should be adjusted down based on the size of the new ad view being inserted.

|  |
| --- |
| // Create a 320x50 ad view  adView = [[MASTAdView alloc] initWithFrame:CGRectMake(0, 0, 320, 50)];  // Configure the site and zone for ad content.  adView.site = 12345;  adView.zone = 67890;  // Update every minute vs. the default of two minutes.  adView.updateTimeInterval = 60;  // Add the ad view to the controllers view.  [self.view adSubview:adView];  // Existing code to add a table, picture box or other application UI  // elements.  // ... |

Figure 3.5 – MASTAdView initialization

This example shows a few properties of the ad view being set including:

* **Publisher Site**: this is setup through the Mocean Mobile UI when you setup ad feeds to display content in your application. Typically a “site” will be used to identify one of your applications and distinguish it from another of your applications. The site is required in order to request an ad.
* **Ad zone**: this is used to identify one specific ad placement in your application. In this example we have created one placement so far, the banner ad to be displayed at the top of the screen. If we choose to display ads in another part of this application, a different placement will be used for that location. Zones are created through the Mocean Mobile UI and target content to ad placements in your application. A given zone falls under one site. The zone is required in order to request an ad.
* **Ad update interval**: This configured the time period (in seconds) after which the ad view will retrieve a new ad from the back-end.

## Getting Initial Ad View Content

Once the ad view has been setup, the initial ad needs to be fetched for display to the user. This is usually done after the view has been loaded in the controller. The following implementation would go in viewDidLoad to follow the previous sample code.

|  |
| --- |
| // Start requesting, receiving and displaying ads.  [self.adView update]; |

Figure 4 - Fetch Initial Ad Content

The SDK itself does the rest of the work, including spawning threads to download ad content from the network without slowing down the UI, etc. The adView object can also be used to customize and manage other properties and behaviors as described in the SDK documentation.

After the initial ad content is displayed, you can continue to invoke the update method manually to refresh the ad content when desired; however, if you have defined an update time interval as shown in the samples above, this is not necessary. The SDK sets a timer and will automatically download updated ad content for you based on the timer setting.

## Creating an Interstitial Ad View

Interstitial ads are full screen ads displayed at transition points in the application (for example when the app is launched, or when moving between screens, etc.) Interstitial ads always include a close button, and optionally can be configured to close automatically after some time has elapsed.

Because of how iOS layers views from one parent UIWindow, interstitial ads can be displayed in several ways. The easiest method is to create and manage the ad view from the controller that will be covered by the interstitial ad. Using this method the ad can then be placed in the controllers view layout to keep visibility of a tab or navigation bar or it can be added to the main window to consume the entire screen.

The following code shows how to add an interstitial ad to the current controllers view. It also makes use of a delegate method to remove the interstitial ad view when the ad view is closed.

|  |
| --- |
| - (void)displayInterstitialAd  {  // In the implementation create the interstitial ad view and display it.  self.interstitialAdView = [[MASTAdView alloc] initWithFrame:self.view.bounds];  self.interstitialAdView.site = 11111;  self.interstitialAdView.zone = 33333;  // Optional: Automatically close after 10 seconds  self.interstitialAdView.autocloseInterstitialTime = 10;  // Optional: Pause 2 seconds before showing close button  self.interstitialAdView.showCloseButtonTime = 2;  // Get notified about the ad close request from the user  self.interstitialAdView.delegate = self;  // Display the ad  [self.view addSubview:self.interstitialAdView];  // Start the ad update to receive content  [self.interstitialAdView update];  }  - (void)closeInterstitialAd  {  [self.interstitialAdView removeFromSuperview];  [self.interstitialAdView setDelegate:nil];  self.interstitialAdView = nil;  }  - (void)didCloseAd(id)sender usageTimeInterval:(NSTimeInterval)usageTimeInterval  {  if (sender == self.interstitialAdView)  {  [self performSelectorOnMainThread:@selector(closeInterstitialAd) withObject:nil waitUntilDone:NO];  }  } |

Figure 5 – Interstitial Ad Display

## Handling Rotation Changes

To support rotation changes on iOS the application and controller must have support for rotation. To learn more about iOS rotation visit the following section in the SDK documentation:

[http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/RespondingtoDeviceOrientationChanges/RespondingtoDeviceOrientationChanges.html](http://developer.apple.com/library/ios/%23featuredarticles/ViewControllerPGforiPhoneOS/RespondingtoDeviceOrientationChanges/RespondingtoDeviceOrientationChanges.html)

Since the MASTAdView object derives from UIView it can also be configured to support rotation. For banner ads the normal use case is to have the banner retain its current position relative to the top or bottom of the screen and resizes its width based on the resulting rotation orientation.

In the sample code above, the following lines can be added to allow for rotation:

|  |
| --- |
| // Start requesting, receiving and displaying ads.  adView.autoresizingMask = UIViewAutoresizingFlexibleRightMargin  | UIViewAutoresizingFlexibleWidth; |

Figure 6.1 –Auto-resizing masks

The above mask will allow the right marine of the ad view to expand to maintain its position relative to the parent view and also allow it to expand along its width. This way when orientation changes from portrait to landscape the ad will grow accordingly.

Since the ad will be resized on rotate, update should be called after the rotation completes. This can be done in the controller’s didRotateFromInterfaceOrientation method as follows:

|  |
| --- |
| -(void)didRotateFromInterfaceOrientation:(UIInterterfaceOrientation)fromInterfaceOrientation  {  // Give the application time to finish it’s rotation animations  // then call update on the adView to request new content based  // on the post-rotation size.  [self.adView performSelector:@selector(update) withObject:nil afterDelay:.5];  } |
| Figure 6.2 – Post-rotation ad content update |

## Customizing the Viewport

Internally the ad view makes use of the UIWebView view to render ad content. For the purposes of ad display, and supporting various device screen sizes and resolutions, there are a number of settings it is important to be aware of that affect the appearance and behavior of content shown in this view (such as width, height and scale.) These can be customized with a viewport property.

Default viewports (W is derived from UIDevice’s mainScreen):  
**Non-Retina:**  
<meta name=’viewport’ content=’width=W; initial-scale=1.0; minimum-scale=1.0; user-scalable=0’/>

**Retina:**<meta name=’viewport” content=’width=W; initial-scale=0.5; minimum-scale=0.5; user-scalable=0’/>

The defaults are intended to render ad content in the native physical resolution of the display. In this way an ad that is 320x50 on a non-Retina device with an ad view with a bounds of 320x50 will fill the full ad view. On a Retina the ad content would need to be 640x100 to fill the full rectangle since the ad view of 320x50 will actually render as 640x100 on the display.

This value can be customized using the *injectionHeaderCode* property of the ad view. For example, in a recent DRAFT MRAID 2 specification, the IAB is requiring future compliant SDK’s to use the string:  
<meta name=’viewport’ content=’user-scalable=no initial-scale=1.0’/>

The following example shows how this can be set NOW if you are working with ad creators who are already moving towards the evolving MRAID standard:

|  |
| --- |
| // Customize the viewport injection string for ad views created through SDK.  // This example uses setting required in the latest IAB MRAID 2 draft spec.  // Also included is the default style code to center ad content in the ad  // view.  self.adView.injectionHeaderCode =  “<meta name='viewport' content='user-scalable=no initial-scale=1.0'/><style>body{margin:0;padding:0;display:-webkit-box-orient:horizontal;-webkit-box-pack:center;-webkit-box-align:center;}</style>”; |

Figure 7 – Customizing the Ad ViewPort

For more information on the full capabilities of the viewport property, consult the iOS developer documentation on this topic at:

[http://developer.apple.com/library/ios/#DOCUMENTATION/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html](http://developer.apple.com/library/ios/%23DOCUMENTATION/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html)

## Detecting Ad Load Failures

Sometimes a developer might want to take a special action if no ad is available that satisfies the current constraints sent to the mobile ad server. This might occur if a particular ad type or minimum size was requested, and no matching ad is available. This could also happen if all ads scheduled for the requested zone have reach the maximum daily or monthly cap.

The SDK includes an optional *MASTAdViewDelegate* protocol which applications can implement to receive notifications when download related ad events occur. This protocol includes the following methods that relate to ad download status in addition to others not described in this section:

* *willReceiveAd:* which is invoked when the request is sent to the mobile ad server.
* *didReceiveAd:* which is invoked after the ad content has been downloaded successfully.
* *didFailToReceiveAd:error:*  which is invoked if downloading ad content fails for any reason.

The *didFailToReceiveAd :* method will be invoked if no ad is received from the ad server. An example implementation of this protocol which shows how to detect this condition is as follows:

|  |
| --- |
| - (void)willReceiveAd:(MASTAdView\*)sender  {  // Implement any pre-download ad processing here  }  - (void)didReceiveAd:(MASTAdView\*)sender  {  [self performSelectorOnMainThread:@selector(showAdView) withObject:nil waitUntilDone:NO];  }  - (void)didFailToReceiveAd:(MASTAdView\*)sender withError:(NSError\*)error  {  [self performSelectorOnMainThread:@selector(hideAdView) withObject:nil waitUntilDone:NO];  }  -(void)showAdView  {  if (self.adView.hidden == NO)  return;  // Adjust the application’s content view so there’s room  CGRect frame = self.appContentView.frame;  frame.origin.y += self.adView.bounds.size.height;  frame.size.height -= self.adView.bounds.size.height;  self.appContentView.frame = frame;  self.adView.hidden = NO;  }  -(void)hideAdView  {  if (self.adView.hidden)  return;  // Adjust the application’s content view to consume  // space taken by the ad view. This can be done with  // an animation if desired.  CGRect frame = self.appContentView.frame;  frame.origin.y -= self.adView.bounds.size.height;  frame.size.height ++ self.adView.bounds.size.height;  self.appContentView.frame = frame;  self.adView.hidden = YES;  } |

Figure 8.0 –Detecting ad loading and failure

### 8.1 Important note when using a delegate

The code creation section can be amended to set the ad view’s delegate property to the current controller with the following:

|  |
| --- |
| // Set this controller as the delegate for the ad view.  self.adView.delegate = self; |

Figure 8.1 – Setting ad delegate

See the dealloc sample to properly nil the delegate value before releasing the ad view object. If not done properly the ad view may attempt to call a delegate method on a controller that no longer exists.

The delegate methods all pass a sender which is the ad view responsible for invoking the method. This is useful for cases where one controller may be a delegate for multiple ad views such as a custom UITableViewController that is hosting various ad rows.

The delegate methods may not and are not guaranteed to be invoked on the main thread. In iOS all UI interaction should occur on the main thread. In the code sample above the desired delegate behavior is implemented in other class methods and invoked from the delegate method with performSelectorOnMainThread to ensure the UI work (changing the hidden flag and frames) is done on the main thread.

For more information on protocol and delegate programming refer to the iOS SDK documentation. See the ad SDK sample projects for sample implementations of the ad delegate protocol.

## Where to go next

You will find more thorough, complex examples and additional use cases in the sample application distributed with the SDK. Both the sample app and the SDK itself are available in source code form from: <http://code.google.com/p/mocean-sdk-ios/>.

You can also find additional documentation, information, and other supported platforms on our developer wiki at: <http://developer.moceanmobile.com/Main_Page>.