Emediate iOS SDK
Developer Guide

Table of Contents

Overview

This document contains the following descriptions:

- Introduction
- Specification
- Development Requirement
- SDK Contents
- SDK Installation Instructions
- SDK API References
- Usages of this SDK

Introduction

Emediate iOS SDK is designed for facilitating iOS native mobile apps development with Emediate Mobile Ads, and speeds up both software development time as well as minimizing integration/design issues.

Specification

Emediate iOS SDK is a MRAID v1.0 and ORMMA level 1 compliant SDK. Reference implementations for MRAID and ORMMA can be found as follows:

- http://code.google.com/p/ormma/
- http://www.iab.net/mraid

Development Requirement

- iOS SDK --- iOS 5.0+
- Xcode
- The SDK uses ARC.

SDK Contents

- EmediateSDK --- Emediate SDK Project files
- SampleApp --- Emediate SDK Test Demo Application

The EmediateAds SDK project consists of the following groups:

1. Private
2. Public
3. Third party
4. Resources
5. Products

1. Private: This group consists of 4 sub groups: Data model, Cache implementations, JavaScript bridge, and Utility.

Data model: This sub group provides the data storage capabilities to the SQLite database.

Cache Implementations: This sub group is responsible for making requests to the server and storing them in a local cache.

Java script bridge: This sub group provides the communication between the MRAID/ORMMA script and the iOS webview component.

Utility: Utility classes which prevents the iOS webview component from bouncing and finds device orientation.

2. Public: The Public group consists of the classes that make up the public interface of the EmediateAds SDK. This includes views that hold the iOS webview component that loads the MRAID/ORMMA ads, finds and stores unique identifier of a device (a.k.a UDID) and the browser that loads the URL's in iOS webview with full screen.

3. Third party: This group is responsible for sending HTTP requests, SQLite data base implementations, and a few extensions for existing apple classes (UIColor & UIDevice).

4. Resources: This group holds the SQLite DB files, JavaScript files which implements the MRAID/ORMMA commands, and resources such as images.

5. Products: This group consists of the final library that understands the script with MRAID commands and processes the commands appropriately, and a bundle that holds the images required for the library.

SDK Installation Instructions

Installation of iOS SDK

Official iOS SDK download url: https://developer.apple.com/devcenter/ios/index.action

Download the iOS SDK, and set up your development environment according to the installation instruction supported by the above link.

Installation of Emediate iOS SDK

Instructions for installing the SDK are as follows:

Steps to create a library

1. Open the EmediateSDK project
2. Build the project.
3. Once the build finishes, navigate to the following path to find the library file(s). /Library/Developer/Xcode/DerivedData/EmediateSDK-Unique-identifier/Build/Products
4. In the above-mentioned path there will be 3 folders.
   a. Debug-iphoneos
   b. Debug-iphonesimulator
   c. Debug-universal
5. Add all the files in the Debug-universal folder into the project where EmediateSDK needs to be incorporated. Debug-universal folder contains following files
   a. EmediateAdView.h
   b. libEmediateSDK.a
   c. MRAID.bundle
   d. MRAIDView.h
   e. MRAIDWebBrowserViewController.h
   f. MRAIDAVPlayer.h

Steps to enable location services in the iOS Simulator

1. Enable location services in OSX "System preference -> Security & Privacy -> Privacy tab -> Enable location Services"
2. Enable location services in the iOS Simulator "iOS Simulator -> Debug -> Location -> Custom Location"

Add the following libraries and frameworks into the project:

1. libz.1.1.3.dylib.
   2. libsqlite3.0.dylib
3. Add the following frameworks in addition to the UIKit.framework, FoundationKit.framework:
   a. MediaPlayer.framework
   b. SystemConfiguration.framework
   c. QuartzCore.framework
   d. MobileCoreServices.framework
   e. MessageUI.framework
   f. EventKit.framework
   g. EventKitUI.framework
   h. CoreMotion.framework
   i. CoreLocation.framework
   j. CoreFoundation.framework
   k. CoreData.framework
   l. CFNetwork.framework
   m. CoreGraphics.framework
4. Import EmediateAdView.h into the class where you wish to display the ad.
5. Set the ad refresh rate using fetchCampgainWithRefreshRate: method. Defaults to 60

seconds.

6. Set the preload count for the campaign the adView is about to load using enablePreloadWithCount: method. Defaults to 5.
7. Set the baseURL property.
8. Send the loadCreativeWithParameters: message to the EmediateAdView object, passing the parameters as a dictionary. These are parameters for the baseURL to fetch the ads.
9. Set the mraidDelegate property to receive callbacks from the ads.
10. Add "-all_load" to "Other Linker Flags" in Build Settings

Emediate Mobile Ads URL

URL Example

http://stage.emediate.eu/eas?cu=123&cre=mu&eas_uid=555&EASTSDK=1&EASTorientation=landscape&EASTscreen=bi g&EASTsection=sports

Model

Emediate Mobile Ads URL includes a certain number of URL parameters which declare the specification of the ads.

- **DEVICE_ID**
  - KeyName: eas_uid
  - ValueType: unique 64bit unsigned integer
  - IsMandatory: Yes
  - Description: This will be proceeded by SDK. This should be generated in such a way that the same number is always generated on the same device. The native device ID must not be used. i.e. hash the unique device id to generate the DEVICE _ID.
- **SYSTEM_URL**
  - ValueType: String
  - IsMandatory: Yes
  - DefaultValue: http://stage.emediate.eu/eas
- **CONTENT_UNIT_ID**
  - KeyName: cu
  - ValueType: unique unsigned integer
  - IsMandator: Yes
  - Description: Generate and store the Unique ID for current device, if not stored already.
- **Optional parameters**

- KeyNameType: must be 16 chars or less
- ValueType: must be 32 chars or less
- Format: *EAST{KeyName}={Value}*

SDK API References

Class EmediateAdView

Global functions

- (void)**loadCreativeWithParameters**:(NSDictionary *)params;

Pass the required parameters to get the ads. Parameters dictionary contains key value pairs that are used to prepare a request.

Parametes has the form key=value.

Eg: `cu=512;cre=mu;target=_blank`

For the above parameters , the dictionary should be prepared as follows:

```
NSDictionary *dictionary = [[NSDictionary alloc]
initWithObjectsAndKeys:@"512", @"cu", @"mu", @"cre",
@"_blank", @"target", nil];
```

Device UDID is appended at the end of the parameter list internally.

- (void)**fetchCampaignWithRefreshRate**:(NSInteger)refresh;

The time interval at which the campaign gets refreshed. Can be 0 or negative which means “no refresh”.

- (void)**enablePreloadWithCount**:(NSInteger)count;

The number of ads that will be prealoaded (and cached) for the requested campaign. Can be 0 which means that preload is disabled and the SDK will query the server every time it needs to display an ad.

- (void)**refreshCreative**;

Refreshes the campaign.

- (void)**stop**;

Stops loading the campaign. Note that in this case **refreshCreative** will have no effect. To restart the campaign, **loadCreativeWithParameters:** should be called again.

Global Properties

@property (nonatomic, retain) NSString ***baseURL**;

The base URL for the ads.

@property (nonatomic, retain) NSDictionary ***parameters**;

Example parameters:

```
NSDictionary *dictionary = [[NSDictionary alloc]
initWithObjectsAndKeys:@"512",    @"cu", @"mu", @"cre",
@"_blank", @"target", nil];
```

@property (nonatomic) NSInteger **refreshRate**;

The rate at which an ad will be refreshed.

@property (nonatomic) NSInteger **preloadCount**;

The number of ads to preload.

Protocol MRAIDViewDelegate

Required functions

– (UIViewController *)**mraidViewController**;

Returns the view controller that is the owner

Optional functions

- (NSString *)**javascriptForInjection**;

Called to allow the application to inject javascript into the creative.

- (void)**handleRequest**:(NSURLRequest *)request **forAd**:(MRAIDView *)adView;

Notifies the consumer that it should handle the specified request.

NOTE: REQUIRED IF A PROTOCOL IS REGISTERED

- (NSString *)**onLoadJavaScriptForAd**:(MRAIDView *)adView;

Called to allow the application to execute javascript on the creative at the time the creative is loaded.

- (void)**failureLoadingAd**:(MRAIDView *)adView;

Called when an ad fails to load.

- (void)**willResizeAd**:(MRAIDView *)adView **toSize**:(CGSize)size;

Called before the ad is resized in place to allow the parent application to animate things if desired.

- (void)**didResizeAd**:(MRAIDView *)adView **toSize**:(CGSize)size;

Called after the ad is resized in place to allow the parent application to animate things if desired.

- (void)**adWillShow**:(MRAIDView *)adView;

Called just before to an ad is displayed.

- (void)**adDidShow**:(MRAIDView *)adView;

Called just after to an ad is displayed

- (void)**adWillHide**:(MRAIDView *)adView;

Called just before an ad is hidden.

- (void)**adDidHide**:(MRAIDView *)adView;

Called just after an ad is hidden.

- (void)**willExpandAd**:(MRAIDView *)adView **toFrame**:(CGRect)frame;

Called just before an ad expands.

- (void)**didExpandAd**:(MRAIDView *)adView **toFrame**:(CGRect)frame;

Called just after an ad expands.

- (void)**adWillClose**:(MRAIDView *)adView;

Called just before an ad closes.

- (void)**adDidClose**:(MRAIDView *)adView;

Called just after an ad closes.

- (void)**appShouldSuspendForAd**:(MRAIDView *)adView;

Called when the ad will being loading heavy content (usually when the ad goes into full

screen display)

- (void)**appShouldResumeFromAd**:(MRAIDView *)adView;

Called when the ad is finished with the heavy content (usually when the ad returns from full screen display)

- (void)**placePhoneCall**:(NSString *)number;

Allows the application to override the phone call process to, for example, display an alert to the user before placing a call.

- (void)**placeCallToAppStore**:(NSString *)urlString;

Allows the application to override the action to open the App Store, for example to display an alert to the user before opening the store.

- (void)**createCalendarEntryForDate**:(NSDate *)date **title**:(NSString *)title **body**:(NSString *)body;

Allows the application to override the process for creating calendar entries to, for example, display an alert before the event view is created.

- (void)**showURLFullScreen**:(NSURL *)url **sourceView**:(UIView *)view;

Allows the application to inject itself into the full screen browser menu to handle the "go" method (send to Safari, Facebook etc.)

Usages of SDK

As illustrated, the "EmediateAdView" is the Mobile Ad container, which will be invoked within your project source code. Similarly, "EmediateAdView" is extended as a iOS UIWebView.

Basic Class import

#import "EmediateAdView.h"

Add EmediateAdView

ViewController.h

EmediateAdView* emediateAdView;

ViewController.m

```
emediateAdView = [[EmediateAdView alloc] initWithFrame:(CGRect){0, 0, 320, 50}];
[emediateAdView setMraidDelegate:self]; //To recieve call backs from MRAID script for interactions by user
[emediateAdView fetchCampaignWithRefreshRate:20]; //Rate at which ad changes...
[emediateAdView enablePreloadWithCount:10]; //Number of ads to preload. Can be the same ad if the campaign has only one ad to serve.
[emediateAdView setBaseURL:@"http://stage.emediate.eu/eas"]; //Base URL.
[self.view addSubview:emediateAdView];

   NSDictionary *dictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"512", @"cu", @"mu", @"cre", @"_blank", @"target", nil];
   [emediateAdView loadCreativeWithParameters:dictionary];
```

Remove EmediateAdView

ViewController.m

```
 if (emediateAdView)
 {
   [emediateAdView stop];
   [emediateAdView setMraidDelegate:nil];
   [ emediateAdView removeFromSuperview];
   [emediateAdView release];
   emediateAdView = nil;
 }
```