appnexus 28 west 23rd street, floor 4, new york, ny 10010
www.appnexus.com

AppNexus Open AdStream Mobile SDK Integration Guide and API Reference for iOS
December 3, 2015

P (646) 825-6460
F (646) 825-6465
info@appnexus.com
appnexus

Table of Contents

Getting Started

AppNexus Open AdStream Mobile SDK allows app developers to incorporate ads into their native iOS applications.
AppNexus Open AdStream Mobile SDK supports the following ad formats:

- Simple banner ads
- HTML/JavaScript based rich media banner ads
- MRAID 1.0 and 2.0 rich media banner ads
- Simple interstitial ads
- HTML/JavaScript based rich media interstitial ads
- MRAID 1.0 and 2.0 rich media interstitial ads
- VAST 2.0 and 3.0 video interstitial ads
- VAST 2.0 and 3.0 in-stream pre-roll video ads

System Requirements

The following are the basic requirements to build and run the demo application:

- iOS version 6.0 or later
- Xcode 7.0 or later

Intended Audience

This document is for iOS native application developers who want to incorporate ads into their applications.

Integrating AppNexusOASSDK Static Library with Swift

AppNexusOASSDK being developed in Native Objective-C language, there is a compatibility issue with Swift environment. To overcome these shortcomings, publisher will have to create an Objective-C bridging header to enable compatibility between AppNexusOASSDK static library and Publisher's application.

Note:

- AppNexusOASSDK static library is now available in two variants –
  - AppNexusOASSDK with BitCode
  - AppNexusOASSDK without BitCode
- The static libraries now require atleast iOS 6.0 or above.

Steps:

1. To integrate the AppNexusOASSDK Static Library, Please follow the instructions at page 14.

2. Create the Bridging Header file as instructed at page 16. This step is very important when working with swift and AppNexusOASSDK, without which the AppNexusOASSDK will never be found in the Swift application.
3. Add the following frameworks to the application –
   MediaPlayer.framework
   AVFoundation.framework
   EventKit.framework
   CoreTelephony.framework
   CoreData.framework
   SystemConfiguration.framework
   libz.dylib /libz.tbd
   CoreGraphics.framework
   UIKit.framework
   Foundation.framework
   MessageUI.framework
   StoreKit.framework
4. Done. Build and the app should build without any errors.

Integrating AppNexusOASSDK Framework with Swift

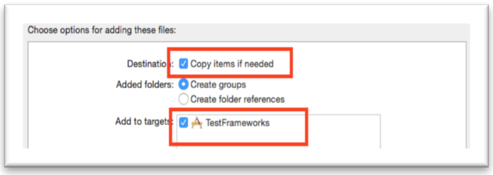
Swift is by nature incompatible with frameworks developed in Objective-C. Hence, we would have to create a bridge to enable talks between AppNexusOASSDK framework and Swift application.

Note:

- AppNexusOASSDK framework is available in two variants –
  - AppNexusOASSDK with BitCode
  - AppNexusOASSDK without BitCode
- The frameworks require atleast iOS 8.0 or above.

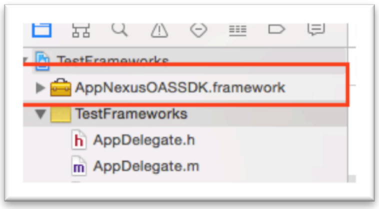
Steps:

1. Unzip the AppNexusOASSDK folder iOS_SDK.zip
2. Locate the framework to be used – with BitCode / without BitCode
3. Drag and drop the framework to the Swift Application under project navigator
   Select "Copy files If Needed" and "Target" when the popup appears.

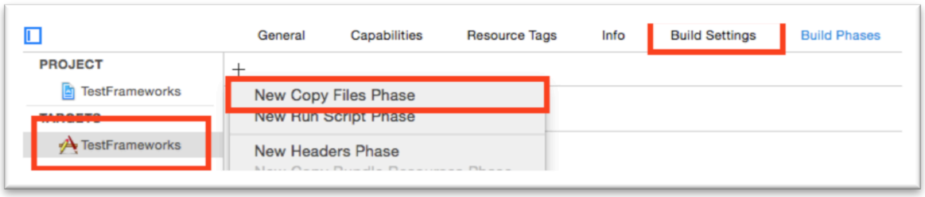


Choose options for adding these files:

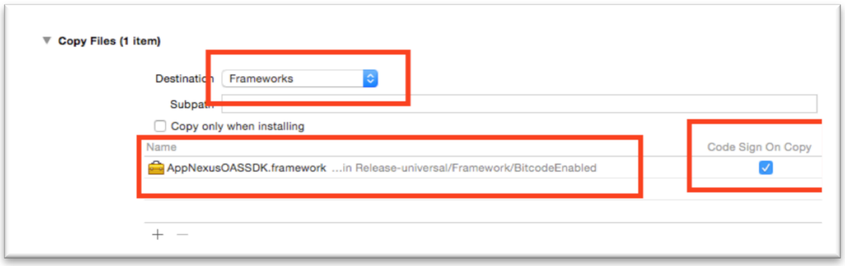
Destination: ✓ Copy items if needed
Added folders: [✓ Create groups] Create folder references
Add to targets: ✓ TestFrameworks



4. Go to Application target build phases and add a new copy files phase.



5. Drag and drop the framework from project navigator into copy files phase. Ensure you select Destination as Frameworks and tick "Code Sign On Copy"


Copy Files (1 item)

Destination: Frameworks

Subpath:

Copy only when installing

Name	Code Sign On Copy
AppNexusOASSDK.framework ...in Release-universal/Framework/BitcodeEnabled	✓

6. Now we need to Create the Bridging Header file as instructed in Create Bridging section at page 16.
7. Add the following frameworks to the application –
   MediaPlayer.framework
   AVFoundation.framework
   EventKit.framework
   CoreTelephony.framework
   CoreData.framework
   SystemConfiguration.framework
   libz.dylib /libz.tbd
   CoreGraphics.framework
   UIKit.framework
   Foundation.framework
   MessageUI.framework
   StoreKit.framework
8. Done. Build and the app should build without any errors.

Importing AppNexusOASSDK Headers into application classes

As you add the AppNexusOASSDK headers to the Objective-C Bridge, it then becomes available to the Swift application and there is no need to import any headers individually. The class names and class methods can directly be used within the methods and events.

Integrating AppNexus Open AdStream Mobile SDK (using COCOAPODS)

To demonstrate the integration of the AppNexusOASSDK, we will assume that the target iOS application into which AppNexusOASSDK needs to be integrated is named AppNexusOASMobileSDKSampleApp.

Pod Description and Requirements

Pods is now available in four variants –

1. AppNexusOASSDK
   Contains Static Library with **BitCode Option Disabled**, Headers and Resources
   Requires iOS 6.0 and above.

2. AppNexusOASSDKBitCode
   Contains Static Library with **BitCode Option Enabled**, Headers and Resources
   Requires iOS 6.0 and above.

3. AppNexusOASSDKFramework
   Contains Dynamic Framework with **BitCode Option Disabled**, Headers and Resources
   Requires iOS 8.0 and above.

4. AppNexusOASSDKFrameworkBitCode
   Contains Dynamic Framework with **BitCode Option Enabled**, Headers and Resources
   Requires iOS 8.0 and above

Following are the steps to integrate AppNexusOASSDK into user's application.

1. Navigate to Application Root Folder (folder where the xcodeproj file resides for the application)
2. Create a PodFile using the following command in Terminal –
   a. `pod init` (This will create the podfile to be used)
   b. `open -a xcode podfile` (Opens the podfile in xcode for editing)
3. Add the following lines to the pod file according to the type of library you want to install

```
# use the below syntax to install AppNexusOASSDK static library without bitcode

platform :'ios', '6.0'

target 'AppNexusOAS' do

pod 'AppNexusOASSDK'

end
```

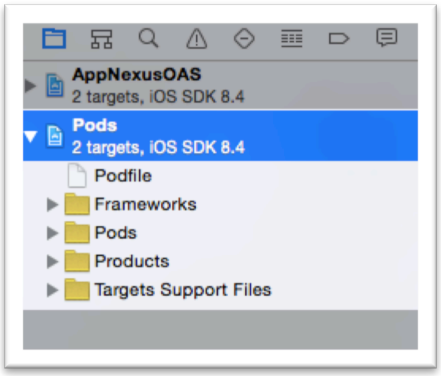
```
# use the below syntax to install AppNexusOASSDK static library with bitcode

platform :'ios', '6.0'

target 'AppNexusOAS' do

pod 'AppNexusOASSDKBitCode'

end
```

```
#use the below syntax to install AppNexusOASSDK framework without bitcode

platform :'ios', '8.0'

use_frameworks!

target 'AppNexusOAS' do

pod 'AppNexusOASSDKFramework'

end
```

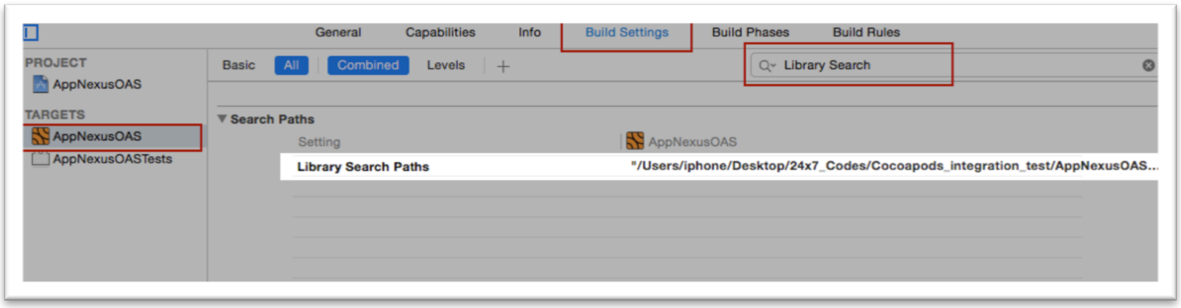
```
#use the below syntax to install AppNexusOASSDK framework with bitcode

platform :'ios', '8.0'

use_frameworks!

target 'AppNexusOAS' do

pod 'AppNexusOASSDKFrameworkBitCode'

end
```

4. Replace "AppNexusOAS" with "your custom name" – Pod will be integrated with this identity.
5. Save and close the podfile

6. Open Terminal and navigate to the folder containing the recently created podfile
   - Type the following command – "`pod install`". To update the existing pod, type in "`pod update`"
7. Close the application if already open in XCode
8. Open the application using **xcworkspace** instead of **xcodeproj**
9. An additional project is added to the workspace other than the application project.

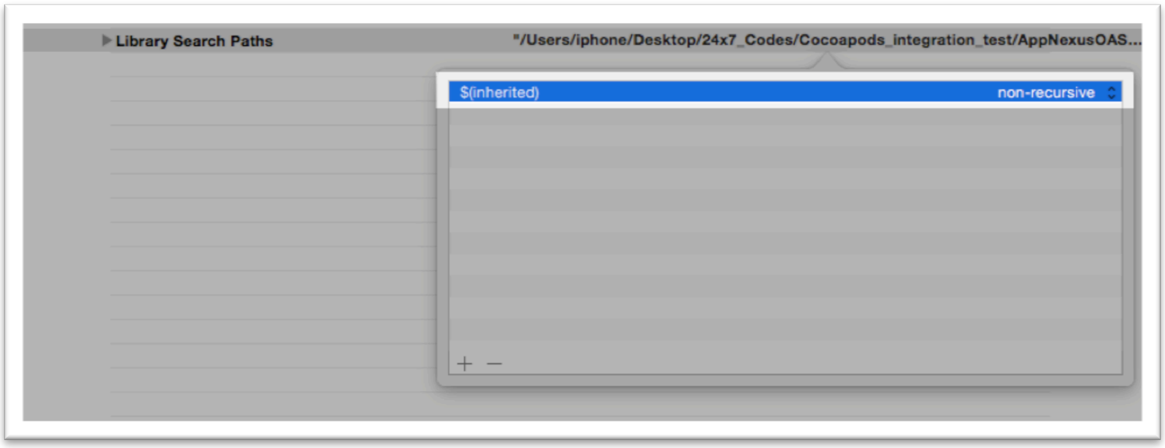


10. Select the desired target for user application and look for "Library Search Paths" under "Build Settings"



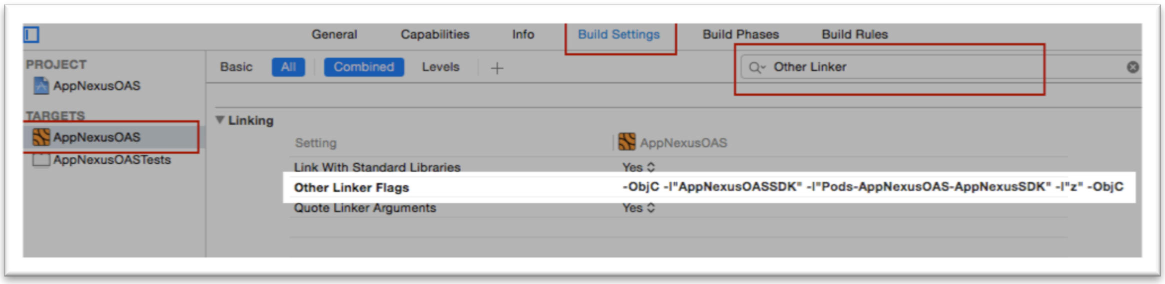
Library Search Paths "/Users/iphone/Desktop/24x7_Codes/Cocoapods_integration_test/AppNexusOAS...

11. Add "$(inherited)" as the first entry to "Library Search Paths"



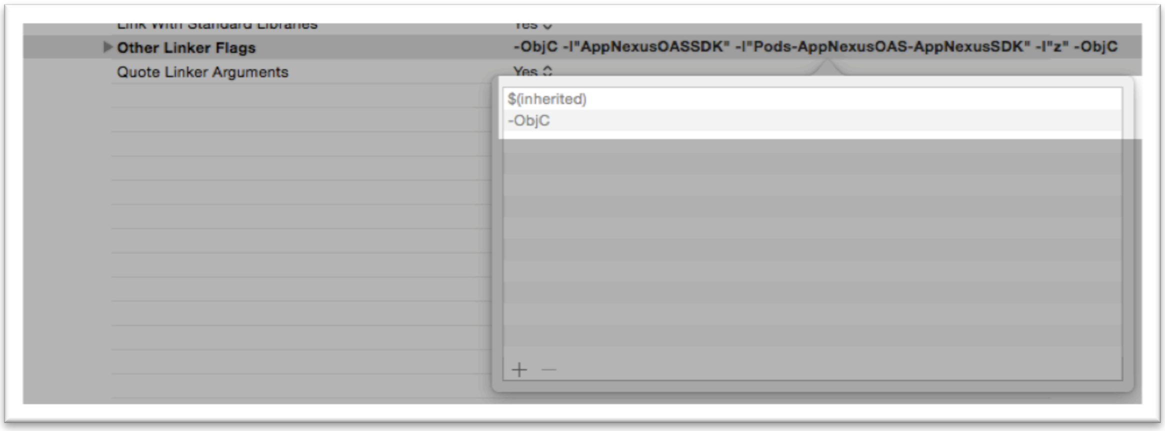
Library Search Paths "/Users/iphone/Desktop/24x7_Codes/Cocoapods_integration_test/AppNexusOAS...
$(inherited) non-recursive

12. Now, look for "Other Linker Flags"



Setting	AppNexusOAS
Link With Standard Libraries	Yes
Other Linker Flags	-ObjC -l"AppNexusOASSDK" -l"Pods-AppNexusOAS-AppNexusSDK" -l"z" -ObjC
Quote Linker Arguments	Yes

13. Add "$(inherited)" as the first entry to the values for "Other Linker Flags"



Other Linker Flags -ObjC -l"AppNexusOASSDK" -l"Pods-AppNexusOAS-AppNexusSDK" -l"z" -ObjC
Quote Linker Arguments Yes

```
$(inherited)
-ObjC
```

14. Make sure the following frameworks and library files are added:

MediaPlayer.framework
AVFoundation.framework

EventKit.framework
CoreTelephony.framework
CoreData.framework
SystemConfiguration.framework
libz.dylib
CoreGraphics.framework
UIKit.framework
Foundation.framework
MessageUI.framework
StoreKit.framework

15. Done! The project should build just fine with these settings.

If installing AppNexusOASSDK using cocoapods in a Swift application then a bridging header will be required in addition to the above mentioned integration steps. This is to ensure Swift talks properly with AppNexusOASSDK and serves the ad upon request. To create the bridge file, kindly follow the steps mentioned at page 16.

Once the bridging header is established then you can call loadAd method on AppNexusOASSDK views without any need to import the headers.

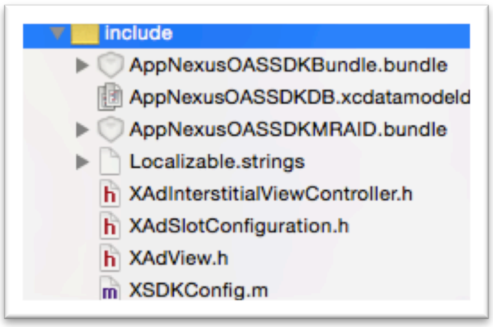
Integrating AppNexus Open AdStream Mobile SDK (Standard)

To demonstrate the integration of the AppNexus Open AdStream Mobile SDK we will assume that the target iOS application into which AppNexus Open AdStream Mobile SDK needs to be integrated is named AppNexusOASMobileSDKSampleApp.

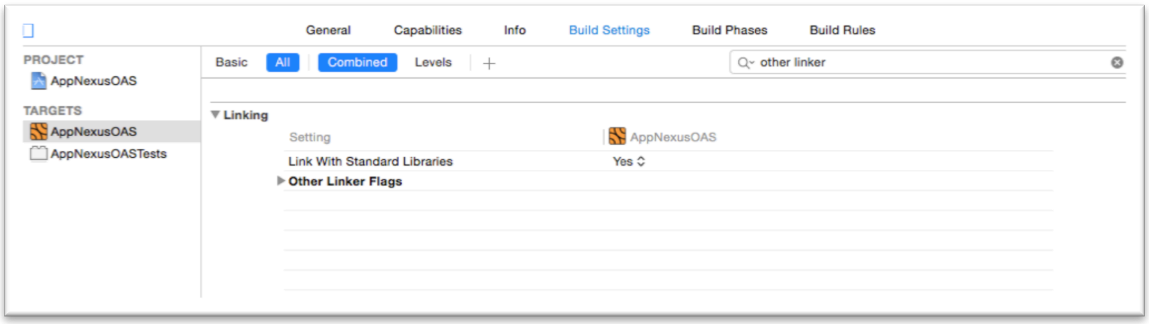
The decompressed SDK consists of Objective-C headers, a runtime library, additional supported libraries for mediation, as well as the release notes.

The following are the steps needed to integrate AppNexus Open AdStream Mobile SDK into AppNexusOASMobileSDKSampleApp application:

1. Right-click on your project in Xcode, choose **Add Files** to " AppNexusOASMobileSDKSampleApp"
2. Add "include folder" which comes in the package.

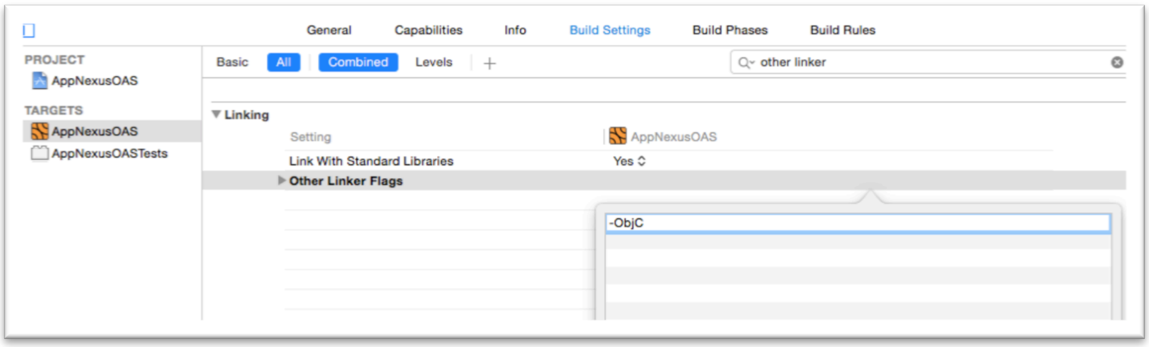

include
AppNexusOASSDKBundle.bundle
AppNexusOASSDKDB.xcdatamodeld
AppNexusOASSDKMRAID.bundle
Localizable.strings
XAdInterstitialViewController.h
XAdSlotConfiguration.h
XAdView.h
XSDKConfig.m

3. Go to Build settings and search for other linker flags.



Setting	AppNexusOAS
Link With Standard Libraries	Yes
Other Linker Flags	

4. Set other linker flags to "-ObjC" (without double quotes)



Setting	AppNexusOAS
Link With Standard Libraries	Yes
Other Linker Flags	-ObjC

5. Make sure the following frameworks and library files are added:

MediaPlayer.framework
libAppNexusOASSDK.a (Provided as part of this SDK package)
AVFoundation.framework
EventKit.framework

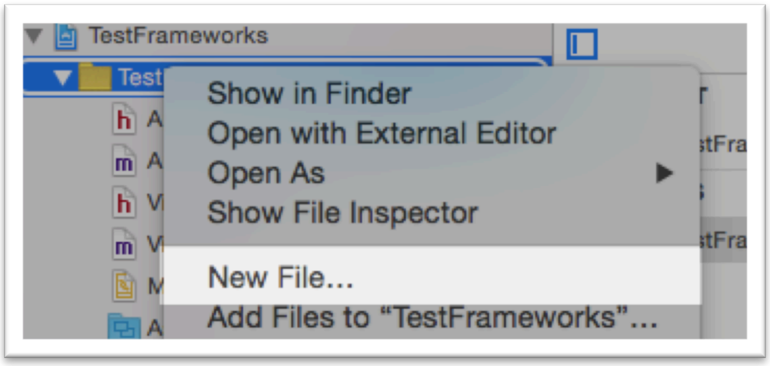
CoreTelephony.framework
CoreData.framework
SystemConfiguration.framework
libz.dylib
CoreGraphics.framework
UIKit.framework
Foundation.framework
MessageUI. Framework
StoreKit.framework

6. Clean and build the project

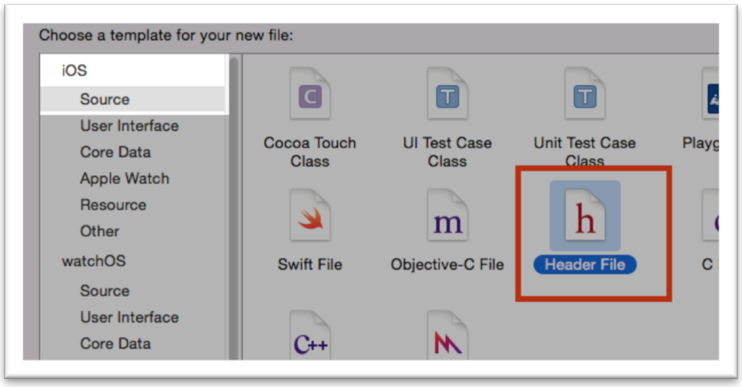
Creating a Bridging Header file for use with Swift Environment

Steps:

1. Open the client application
2. Select the desired target folder on the project Navigator window and add new file by right clicking on the folder and selecting "New File…" from the contextual menu.



3. Under iOS – Source, select header file template

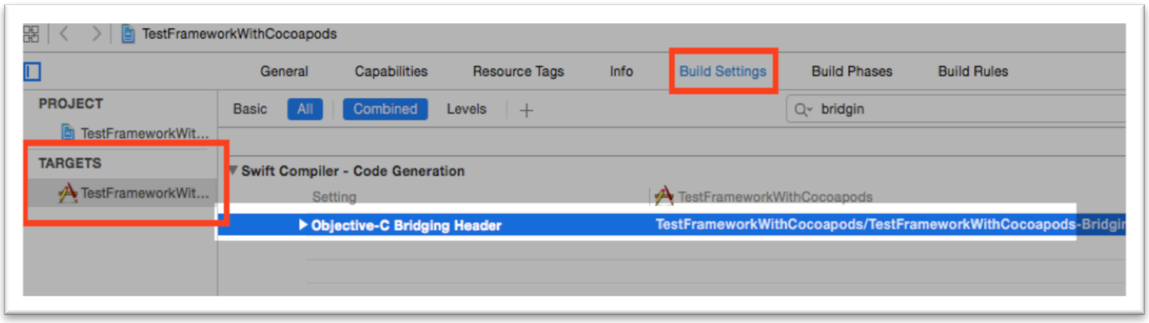


4. While Naming take care to name it as "Your_Project_Name-Bridging-Header.h"
5. Select the folder to save the header file and click on "Create"
6. Now, open the bridging header file that we just created in XCode and add the following lines to it.

   Add the below lines if you are using AppNexusOASSDK Framework Library
   #import <AppNexusOASSDK/XAdView.h>
   #import <AppNexusOASSDK/XAdInterstitialViewController.h>
   #import <AppNexusOASSDK/XAdSlotConfiguration.h>
   #import <AppNexusOASSDK/XBrowserConfiguration.h>
   #import <AppNexusOASSDK/XGlobalConfiguration.h

   Add the below lines if you are using AppNexusOASSDK Static Library
   #import "XAdView.h"
   #import "XAdInterstitialViewController.h"
   #import "XAdSlotConfiguration.h"
   #import "XBrowserConfiguration.h"
   #import "XGlobalConfiguration.h"

7. Now open the target build settings and look for "Objective-C Bridging Header"



Objective-C Bridging Header   TestFrameworkWithCocoapods/TestFrameworkWithCocoapods-Bridgin

8. Enter the name of the just created Bridging header file name for e.g., "Contained_Folder_name/Your_Project_Name-Bridging-Header.h"

Optional Settings

Problem Case:

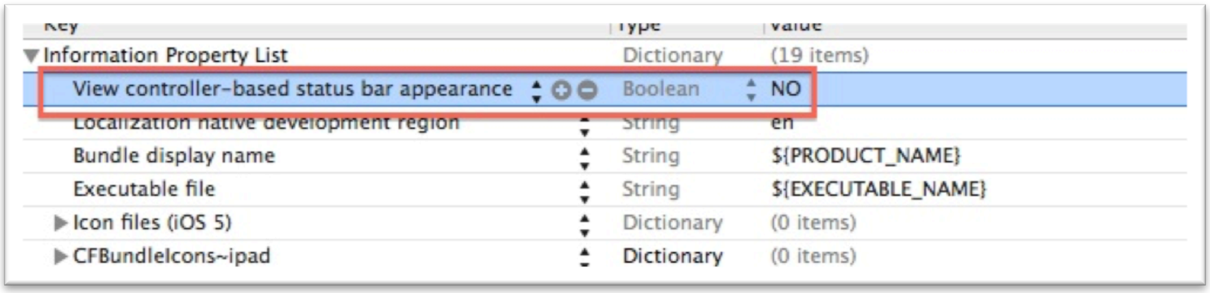
While displaying any ads modally, SDK programmatically hides the status bar. If the status bar comes up due to any application request or due to any phone calls or notifications, the modally displayed ad shifts little down, however the close button on the ad is partially hidden.

Solution:

To handle this problem case, the publisher has to set a flag in the application plist file. The flag is called "**View controller-based status bar appearance**". This flag takes **Boolean** values. For an effective use of this flag, the publisher must set it to "NO" for the OS to respond to the

setStatusBarHidden method of **UIApplication** for iOS 7 and above. This flag can be set as detailed below -

1) Go to application plist file
2) Add "View controller-based status bar appearance" item in the plist
3) Set the value to "NO"



Key	Type	Value
Information Property List	Dictionary	(19 items)
View controller-based status bar appearance	Boolean	NO
Localization native development region	String	en
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
Icon files (iOS 5)	Dictionary	(0 items)
CFBundleIcons~ipad	Dictionary	(0 items)

Application Transport Security (ATS)

Problem Case:

SDK fails to show ads OR SDK fails to open Browser on Clicks OR SDK fails to open apps upon Click-to-action events. In all above cases, if the error console shows any error related to "ATS" i.e., Application Transfer Protocol, then please follow the solution provided.

Solution:

It can be fixed with a configuration change in the application's plist file to handle urls via HTTPS protocol. Apple has temporarily made a provision to disable ATS validation via the plist file. Meaning, if publishers know all the domains that they use, they can exclude those domains from the ATS validation temporarily until Apple discontinues this provision completely. Below is the sample on how this can be achieved in the application plist.

Step 1:
Open application plist file

Step 2:
Add a new row with key "NSAppTransportSecurity" and Type "Dictionary"

Key	Type	Value
NSAppTransportSecurity	Dictionary	

Step 3:
Click on the "+" sign next to the newly added key "NSAppTransportSecurity"

Step 4:
Add a new item to the key as given below –

Key	Type	Value
NSAllowsArbitraryLoads	Boolean	YES

Opening custom URL schemes

Problem Case:

While using AppNexusOASSDK for devices with iOS 9 and over, publishers may experience an error and may see following error message in the console -

```
-canOpenURL: failed for URL: "<scheme>://" - error: "This app is not allowed to query for scheme
<scheme>"
```

Solution:

This error appears because apple has added a new security feature for iOS 9 and above. To support the schemes required by AppNexusOASSDK so that the SDK can function seamlessly, the following entries are required to be added to the publisher’s application plist file.

Step 1:
Open application plist file

Step 2:
Add a new row to the plist with Key “LSApplicationQueriesSchemes” and Type “Dictionary”

Key	Type	Value
LSApplicationQueriesSchemes	Dictionary	

Step 3:
Click on “+” next to the newly added key to add a new row under the key

Step 4:
Add the following items to the key “LSApplicationQueriesSchemes” –

Key	Type	Value
item 0	String	app
item 1	String	mailto
item 2	String	mraid
item 3	String	tel
item 4	String	sms

item 5	String	itunes
item 6	String	facetime

Building a demo app

To build the demo app, you need to delete the references to the 'include' folder and the libAppNexusOASSDK.a, and replace them by following steps 1 and 2 above. This ensures that the paths to these library files are set correctly. Please ensure that the library path is specified correctly in the "Library Search Paths" section of "Build Settings".

Integration Overview

Showing Banner Ads

Initialize XAdView with your project bannerAdView
Refer to the following code for more details.

Objective-C

```
@property(nonatomic,strong)XAdView *bannerAdView;
```

Swift

```
var bannerAdView:XAdView?
```

1) In viewWillAppear initialize banner view with your frame
2) Add your bannerView as SubView
3) The following steps are optional:
    a. Assign the XAdView delegate
    b. Initialize slot configuration
    c. Set bannerRefreshInterval to the desired value
    d. Set scallingAllowedProperty to the desired mode

4) For fetching and displaying ads from server, call loadWithDomainName. Set the DomainName, PageName attribute and adPosition attributes, keywords:attribute(s), queryString:attribute(s).

Example:

Objective-C

```
-(void)viewWillAppear:(BOOL)animated

{

[super viewWillAppear:animated];

/* Initialising the XAdView and fetching the ad */

self.bannerAdView = [[XAdView alloc]initWithFrame:CGRectMake(x_position,

y_position,
xadView_width,

xadView_height)];

self.bannerAdView.delegate = self;

XAdSlotConfiguration *configuration = [[XAdSlotConfiguration alloc] init];
configuration.bannerRefreshInterval = 120.0f;

configuration.scalingAllowed = NO; configuration.openClickThroughURLInDeviceBrowser = NO;

configuration.RTBRequired = NO;

configuration.COPPAPermissions =YES;

self.bannerAdView.slotConfiguration = configuration;

[self.view addSubview:self.bannerAdView];


[self.bannerAdView loadWithDomainName:@"delivery.uat.247realmedia.com"
pageName:@"www.mobilesdkdemo.com/page_320x50" adPosition:@"@x23" keywords:nil
queryString:nil];}
```

Swift

```
func viewWillAppear(animated: Bool) {
    super.viewWillAppear(animated)
    self.bannerAdView = XAdView(frame: CGRectMake(x_position, y_position, xadView_width,
xadView_height))
    self.bannerAdView.delegate = self
    var configuration: XAdSlotConfiguration = XAdSlotConfiguration()
    configuration.bannerRefreshInterval = 120.0
    configuration.scalingAllowed = false
    configuration.openClickThroughURLInDeviceBrowser = false
    configuration.RTBRequired = false
    configuration.COPPAPermissions = true
    self.bannerAdView.slotConfiguration = configuration
    view.addSubview(bannerAdView)
    bannerAdView(domainName: "delivery.uat.247realmedia.com", pageName:
"www.mobilesdkdemo.com/page_320x50", adPosition: "@x23", keywords: nil, queryString: nil)
}
```

Note	Keywords and queryString can be passed as NIL or actual value

Showing Interstitial Ads

Initialize XAdInterstitialViewController in your project.
Refer to the following code for more details:

Objective-C

```
@property (nonatomic, strong) XAdInterstitialViewController *interstitial;
```

Swift

```
var interstitial:XAdInterstitialViewController?
```

1) Initialize Interstitial
2) Present Interstitial view
3) The following steps are optional:
    a. Set the XAdInterstitialViewController delegate
    b. Initialize slot configuration
4) For fetching and displaying ads from server, call:

Objective-C

```
loadWithDomainName:domainName:pageName:adPosition:keyword:queryString
```

Swift

```
loadWithDomainName(domainName,pageName,adPosition,keyword,queryString)
```

5) Set the PageName, adPosition, keyword, QueryString and DomainName attributes.

Objective-C

```
interstitial = [[XAdInterstitialViewController alloc] init];

interstitial.delegate = self;

[self presentViewController:interstitial animated:YES completion:nil];
[interestitial loadWithDomainName:@"delivery.uat.247realmedia.com" pageName:@"MSDK-Joule-
banner-TF1_Eurosport_iPad_RM_ban-249063"  adPosition:@"Left" keywords:nil
queryString:nil];
```

Swift

```
interstitial = XAdInterstitialViewController()
interstitial.delegate = self
presentViewController(interstitial, animated: true, completion: nil)
interstitial(domainName: "delivery.uat.247realmedia.com", pageName: "MSDK-Joule-banner-
TF1_Eurosport_iPad_RM_ban-249063", adPosition: "Left", keywords: nil, queryString: nil)
```

Notes: It is important not to call presentViewController from within the calling view controller's viewWillAppear. When the interstitial dismisses, viewWillAppear to be called again, leading to a situation where iOS throws an exception when trying to present a controller while dismissing it at the same time.

Additionally, if presenting the interstitial on viewDidLoad, keep in mind that viewDidLoad will be called again when the interstitial is dismissed for any reason. It is good practice to maintain a flag that indicates whether the interstitial was displayed to avoid an infinite loop.

You may choose to present the interstitial view controller on the success callback xAdInterstitialDidLoad. This is especially useful to prevent the interstitial from displaying at all when the server does not return an ad.

Showing Pre-roll Video Ads

1) Initialize MPMoviePlayerController instance
2) Set the frame of the movie player
3) Add the view of the movie player instance as subview
4) Initialize XAdView object
5) Assign movie player to the moviePlayerInstance property of the XAdView object
6) For fetching and displaying ads from server, call loadWithDomainName. Set the pageName, adPosition, dataFormat, queryString and DomainName attribute values.

Example:

Objective-C

```
NSURL *url = [NSURL fileURLWithPath:@"http://yourserver.com/moviename.mp4"];

moviePlayerControllerInstance = [[MPMoviePlayerController alloc]

initWithContentURL:url];

CGFloat height = [UIScreen mainScreen].bounds.size.height;

[moviePlayerControllerInstance.view setFrame:CGRectMake(x_position, y_position,

view_width, view_height)];

[self.view addSubview:moviePlayerControllerInstance.view];

adview = [[XAdView alloc] init];

adview.moviePlayerInstance = moviePlayerControllerInstance;

adview.delegate = self;

[adview loadWithDomainName:@"network.realmedia.com" pageName:@"BZ71581"
adPosition:@"@Frame2" keywords:nil queryString:nil];
```

Swift

```
var url: NSURL = NSURL(string: "http://yourserver.com/moviename.mp4")

moviePlayerControllerInstance = MPMoviePlayerController(contentURL: url)

var height: CGFloat = UIScreen.mainScreen().bounds.size.height

moviePlayerControllerInstance.view.frame = CGRectMake(x_position, y_position,
view_width, view_height)
```

```
view.addSubview(moviePlayerControllerInstance.view)

adview = XAdView()

adview.moviePlayerInstance = moviePlayerControllerInstance

adview.delegate = self

adview(domainName: "network.realmedia.com", pageName: "BZ71581", adPosition: "@Frame2",
keywords: nil, queryString: nil)
```

Implementing XAdViewDelegate for Pre-roll Video Ads

The application will need to know when the pre-roll play out has finished. When this delegate method is called, the application resumes responsibility for the player. The movie player controller must not be playing or configured to autoplay when this method is called. Alternatively, the movie player controller can be used just to display an ad, and the delegate can dismiss the controller's view to again show the app's main content to the user.

Objective-C

```
-(void)xadView:(XAdView *)xadView
prerollDidFinishWithPlayer:(MPMoviePlayerController*)player {


//Hook up notifications now that the preroll has finished.


//Play the main video
}


-(void)xAdView:(XAdView *)xAdView didFailWithError:(NSError *)error

{

}
```

Swift

```
func xadView(xadView: XAdView!, prerollDidFinishWithPlayer player:
MPMoviePlayerController!) {
}

func xAdView(xAdView: XAdView!, didFailWithError error: NSError!) {
}
```

Handling Callbacks with Delegates

The application may choose to handle callbacks from the Mobile SDK. These callbacks are implemented with Objective-C delegates, and allow the application to respond to particular events that occur during the lifecycle of an ad request and display. Although all of the delegate methods are optional, an application will typically want to handle at least a few of the more common delegate methods.

There are two delegates available, one for XAdView, and another for XAdInterstitialViewController. They are called *XAdViewDelegate* and *XAdInterstitialViewDelegate*, respectively. The complete list of callbacks is described in the SDK documentation.

There are several very common instances where these delegates are useful. These use-cases are described below.

Pre-roll Completion

In a video pre-roll scenario, it is important to know when the pre-roll has completed. When the pre-roll has finished, the SDK gives up control of the video area back to the application. Often the application will want to start playing the video right away. Do this with the **xAdView:prerollDidFinishWithPlayer:** message. Keep in mind that an ad request may fail. In this case, you will also want to start video playback when the ad fails. Do this with the **xAdView:didFailWithError:** message.

Sample code:

Objective-C

```
-(void)xAdView:(XAdView *)xAdView
prerollDidFinishWithPlayer:(MPMoviePlayerController*)

                                  moviePlayerController

{

    NSLog(@"Preroll finished, let's continue playing our video");

    [self.videoPlayer play];

}


-(void)xAdView:(XAdView *)xAdView didFailWithError:(NSError *)error

{

    if (xAdView == self.preroll)
    {

        NSLog(@"Ad failed to load, start our video");

        [self.videoPlayer play];

    }

}
```

Swift

```
func xAdView(xAdView: XAdView!, prerollDidFinishWithPlayer moviePlayerController:
MPMoviePlayerController!) {
    NSLog("Preroll finished, let's continue playing our video");
    videoPlayer.play()
}

func xAdView(xAdView: XAdView!, didFailWithError error: NSError!) {
    if (xAdView == preroll) {
        NSLog("Ad failed to load, start our video");
        videoPlayer.play()
    }
}
```

Ad View Control

Consider the case where you want to display an ad in a banner, and you only want to add the ad view into the layout when the ad was successfully loaded. Or alternatively, you want to remove the ad banner area from the layout if the ad failed rather than display the default background. In these cases you should handle the **xAdViewDidLoad:** and **xAdView:didFailWithError:** messages.

Sample code:

Objective-C

```
-(void)xAdViewDidLoad:(XAdView *)adView
{
      //Ad was successfully loaded. Add it to the layout.
     [self.view addSubView:adView];
}

-(void)xAdView:(XAdView *)xAdView didFailWithError:(NSError *)error
{
    //Ad server did not return a valid ad. There is nothing to show.
    [xAdView setHidden:YES];
}
```

Swift

```
func xAdViewDidLoad(adView: XAdView) {
    view.addSubView(adView)
}

func xAdView(xAdView: XAdView, didFailWithError error: NSErrorPointer) {
    xAdView.hidden = true
}
```

Interstitial Presentation

There are two general ways to present an interstitial. One is to call presentViewController:animated:completion immediately after the call to loadWithDomainName:page:position. The other is to defer the call to presentViewController:animated:completion until the interstitial was successfully loaded. While the former is simpler code, if the ad load fails, the user will see a blank interstitial for a short time. This is because the view controller will be displayed immediately, and then dismissed automatically after the SDK determines that a failure occurred. The latter creates a bit of a better user experience if the ad fails. In this case, the application will show the interstitial view controller only when it knows the ad load was successful. To do this, handle the **xAdInterstitialDidLoad:** and **xAdInterstitial:didFailWithError** messages.

Important:
In case when the mediated interstitial ad is served, SDK will handle the presentation of the interstitial ad by itself and would pass the xAdInterstitialViewController param as null. Publishers are requested to perform a null check to handle the mediated ads and must not present the controller.

Objective-C

```
-(void)xAdInterstitialDidLoad:(XAdInterstitialViewController*)
                                  interstitialAdViewController
{
      if (interstitialAdViewController) { //Interstitial successfully loaded
             [self.navigationController presentViewController:
             interstitialAdViewController animated:YES completion:nil];
      }
}

-(void)xAdInterstitial:(XAdInterstitialViewController*) interstitialAdViewController
                                  didFailWithError:(NSError *)error
{
    //Interstitial failed. Continue with what we were doing.
    [self showPostInterstitial];
}
```

Swift

```
func xAdInterstitialDidLoad(interstitialAdViewController: XAdInterstitialViewController!)
{
  navigationController.presentViewController(interstitialAdViewController, animated:
true) { () -> Void in
            NSLog("%@", "Ad Loaded Successfully.");
        }
}

func xAdInterstitial(interstitialAdViewController: XAdInterstitialViewController!,
didFailWithError error: NSError!){
        showPostInterstitial()
}
```

Interstitial Completion

Often an interstitial is used between two application states, such as between game levels, or when navigating to a new section in the application. In these cases, it is important to know when the interstitial has completed so that additional setup work and/or navigation can continue. You will need to handle the **xAdInterstitialDismissed:** message.

Sample code:

Objective-C

```
-(void)xAdInterstitialDismissed:(XAdInterstitialViewController*)
interstitialAdViewController
{
    NSLog(@”Interstitial finished.”);
    [self performSegueWithIdentifier:@”NextLevelSegue” sender:self];
}
```

Swift

```
func xAdInterstitialDismissed(interstitialAdViewController:
XAdInterstitialViewController!) {

        NSLog("Interstitial finished.")
        self.performSegueWithIdentifier("NextLevelSegue", sender: self)
}
```

Third Party "No Ad" responses

While the SDK is capable of detecting 'no-ad' responses from Open AdStream, it is often trickier to detect the case where a no-ad response was served by a third party ad-server as a result of a redirect (both explicit and implicit). This is exacerbated by the fact that different publishers use different third party ad servers, and the no-ad responses are very ad server specific.

To aid in this case, the SDK provides a callback to the application so that the developer can inspect the contents of the webview and determine based on its own rules whether or not the response was a valid ad. To use this feature, handle the **xAdView:shouldDisplayAdOnWebViewFinishRender:** message (or the interstitial equivalent **xAdInterstitialViewController:shouldDisplayAdOnWebViewFinishRender:)**.

If this delegate returns YES, then SDK handling continues normally. That is, the application will receive the ad loaded callback, and the ad will display as usual.

However, if the delegate returns NO, the SDK will treat this as an error condition, and the standard error handling logic will be executed as follows:

- In the case of a banner, the xAdView:didFailWithError: callback will be called, and the SDK will show the default image if one is provided by the application. If the app developer chooses the hide the ad area, they can do so in response to xAdView:didFailWithError: as shown in the "Ad View Control" section above.

- In the case of an interstitial, the interstitial view will not be displayed, and the xAdInterstitial:didFailWithError: callback will be called. Typically, this is where the developer will handle the case of a failed ad for an interstitial as shown in the "Interstitial Presentation" section above.

- This delegate method is never called for the case of a pre-roll. A pre-roll is always VAST, which is a standard, and has a specific no-ad response format which doesn't vary between ad servers. Any ad that is not VAST which is served for a pre-roll is considered an error by the SDK, so there is no need for this callback.

Sample code:

Objective-C

```
-(BOOL)xAdView:(XAdView *)xAdView shouldDisplayAdOnWebViewFinishRender:(UIWebView*)
                                                                        webView
{
    NSString* html = [webView stringByEvaluatingJavaScriptFromString:
        @"document.body.innerHTML"];

    //This calls your app-supplied method to check if the response is valid, or contains
    //an indication from the third party server that no ad was served.
    BOOL isValid = [self checkIsValidAdForHTML:html];
    return isValid;
}

-(void)xAdView:(XAdView *)xAdView didFailWithError:(NSError *)error
{
    //Ad server did not return a valid ad. There is nothing to show.
    //This is also called if xAdView:shouldDisplayAdOnWebViewFinishRender: returns NO
    [xAdView setHidden:YES];
}
```

Swift

```
func xAdView(xAdView: XAdView!, shouldDisplayAdOnWebViewFinishRender webView:
UIWebView!) -> Bool {

     var htmlString:NSString?



     htmlString =
webView.stringByEvaluatingJavaScriptFromString("document.body.innerHTML");



        return checkForHTMLCorrectness(htmlString);

    }


func xAdView(xAdView: XAdView!, didFailWithError error: NSError!) {

     xAdView.hidden = true;
}
```

Please Note:

Using third party script redirects containing javascript’s window.location cannot be easily detected and SDK would render the content as it is. This is because there can be numerous conditional ways window.location can be programmed, therefore it becomes very difficult to detect.

Recommended approach to support such kind of redirects is to use <meta http-equiv=”refresh”> tag. SDK detects meta tag using regex and therefore it is necessary that creative code uses correct syntax of <meta> tag. In case of complex ad scripts, if SDK fails to detect **<meta http-equiv="refresh"** using regex**,** then SDK would pass on available ad response in xAdShouldDisplay callback.

It is recommended to use a simple and correct syntax to initialize meta-refresh tag. Following is an e.g.

<meta http-equiv="refresh" content="0;http://www.exampleurl.com">

Low Memory Warning

When the SDK detects an OS-sent low memory warning, it will tear down any current ads in an attempt to let the application reclaim as much memory as possible. Although the application will get its own such notification from the operating system, the SDK also lets the application know when this happens, using the **xAdViewDidDismissOnMemoryWarning:** message (and the interstitial equivalent **xAdInterstitialDidDismissOnMemoryWarning:**).

Sample code:

Objective-C

```
-(void)xAdViewDidDismissOnMemoryWarning:(XAdView *)adView
{
    //Ad view was cleared because of low memory conditions.
    [adView removeFromSuperview];
}
```

Swift

```
func xAdViewDidDismissOnMemoryWarning(adView: XAdView!) {

        adView.removeFromSuperview();
}
```

Click to Actions

In order to provide flexibility to application developers to display alerts as per the context or theme of application, a delegate method is necessary. Also, this delegate helps supporting stricter policies on alerts in certain countries. For example, in France, it is mandatory to display a user confirmation pop-up for click to call action.

To achieve this, SDK provides an optional delegate method, which can be implemented by application developer. If display of custom pop-up is required, this delegate should return NO. It means that the MSDK stops the flow executing Click to Action. And application developer needs to add an AlertView into this delegate in order to show a custom pop-up. This is required because of the asynchronous nature of the AlertView. As a result this delegate is needed to stop the flow of the SDK to wait for user's reaction.

Starting iOS 9.0 and over, to make click to actions work seamlessly with the application, a few special entries are required to be made in the application plist file. To know more, please visit section "Opening custom URL schemes".

Important:
The mediation networks may not support click to Actions, when the publisher enables mediation.

Exemption: There is an exemption to this implementation for click to store picture action. According to the IAB standards, click to store picture already requires showing a confirmation dialog box before accessing the phone gallery. As a result this delegate will not be fired for the click to store picture use case. Instead it is handled by the SDK. To make it multi-language complaint, we have entered the following keys:

- Message text
- "Yes" button
- "No" button

into the resource files, presently for France, English-US and English-UK. These language files are exposed to the client in the include folder. If there is a need to extend the multi- language support for another language, then app developer will have to simply add the new language file to the include folder with the pre-defined keys and their values in the native language. This way the implementation is flexible for any language supported by the iOS devices.

Sample code:

Objective-C

```
- (BOOL)xAdInterstitialViewController:(XAdInterstitialViewController
*)xAdInterstitialViewController shouldHandleClickToAction:(XClickToAction)actionType
parameters:(NSDictionary *) parameters{

    switch (actionType) {
        case XClickToActionOpenBrowser:
        case XClickToActionCall:
        case XClickToActionSMS:
        case XClickToActionAppstoreItunes:
        case XClickToActionCalendar:
        case XClickToActionEmail:
        {
            myActionType = actionType;
            myParameters = parameters;
             UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Alert"
             message:@"<Alert message goes here?>" delegate:self cancelButtonTitle:@"No"
             otherButtonTitles:@"Yes", nil];
            [alertView show];

            return NO;
        }

        default:
            break;
    }
    return YES;

}
```

Swift

```
func xAdView(xAdView: XAdView!, shouldHandleClickToAction actionType: XClickToAction,
parameters: [NSObject : AnyObject]!) -> Bool {


        switch actionType.rawValue{

        case XClickToActionOpenBrowser.rawValue, XClickToActionCall.rawValue,
XClickToActionSMS.rawValue, XClickToActionAppstoreItunes.rawValue,
XClickToActionCalendar.rawValue, XClickToActionEmail.rawValue:

            let alertView = UIAlertView()

            alertView.title = "Alert"

            alertView.message = "Alert Message"

            alertView.addButtonWithTitle("NO")

            alertView.addButtonWithTitle("YES")

            alertView.delegate = self

            alertView.show()

            return false

        default:

            break

        }

        return true
    }
```

Above sample code is required for Interstitial Banner. If you want to use the same approach on the GeneralBanner, you need to use xAdView:shouldHandleClickToAction:parameters.
App Developer also needs to implement the delegate for UIAlertView. App developer has to make an explicit call to an SDK method performClickToAction:parameters which is required by the SDK to execute with showing the dialogs for the specific actions as per the SDK requirements. Once this delegate is implemented, failing to call SDK method performClickToAction will terminate the flow.

Sample code:

Objective-C

```
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if (buttonIndex == alertView.cancelButtonIndex)
    {
        //Do nothing
    }
    else if (buttonIndex == alertView.firstOtherButtonIndex)
    {
        [self.interestitial performClickToAction:myActionType parameters:myParameters];
    }
}
```

Swift

```
func alertView(alertView: UIAlertView, clickedButtonAtIndex buttonIndex: Int) {

        if (alertView.buttonTitleAtIndex(buttonIndex) == "NO"){

            //Do Nothing

        }else if(alertView.buttonTitleAtIndex(buttonIndex) == "YES"){

            self.performClickToAction(myActionType:XClickToAction! parameters
myParameters:[NSObject : AnyObject]!)

        }
 }
```

Click to Actions: Handling SMS body and recipients

Currently only iOS 8 supports pre-populating SMS body from URL in the SMS app. To handle this on all versions of iOS, current version of SDK parses the SMS url, extracts the body tag and recipients, and opens the in-app SMS composer with pre-populated body and recipients. SDK supports many different types of SMS URL formats. Following are the examples:

1. sms://987654321,123123323,488888555&body=hello
2. sms:123123121&body=hello
3. sms://1233423423&body=hello
4. sms:12312312123?body=hello
5. sms://12312312123?body=hello
6. sms:123324232;body=hello
7. sms://123324232;body=hello
8. sms:123123123,345345345,983459834
9. sms://123123123,345345345,983459834

However, to support this feature with some other formats of URLs, SDK provides the SMS URL to publisher via parameters dictionary object available in the shouldHandleClickToAction delegate method.

Publishers can perform following steps to open the pre-populated SMS composer using that URL.

1. Create and initialize an object of NSMutableDictionary.

Objective-C

```
myParameters = [[NSMutableDictionary alloc] init];
```

Swift

```
var myParameters:NSMutableDictionary! = NSMutableDictionary()
```

2. Copy the parameters dictionary into this new object using following code:

Objective-C

```
[myParameters setValuesForKeysWithDictionary:parameters];
```

Swift

```
myParameters.setValuesForKeysWithDictionary(parameters)
```

3. Extract the URL from dictionary in the clickToAction callback.
   It can be extracted using following code:

Objective-C

```
NSURL *url = [parameters objectForKey:XParameterCommandURL];
```

Swift

```
var url:NSURL! = myParameters.objectForKey(XParameterCommandURL)
```

4. Parse it and extract body and recipients.
5. Form the new URL in one of the formats that SDK supports.
6. Set it back again in the dictionary object.

Objective-C

```
[myParameters setValue:[NSURL URLWithString:newUrl forKey:XParameterCommandURL];
```

Swift

```
myParameters.setValue(NSURL(string:newURL), forKey: XParameterCommandURL)
```

7. Call the existing SDK method - performClickToAction with the updated dictionary object.

Objective-C

```
[self.bannerAdView performClickToAction:myActionType parameters:myParameters];
```

Swift

```
self.bannerAdView.performClickToAction(myActiontype:XClickToAction!, parameters
myParameters:[NSObject:AnyObject!])
```

Additionally, Starting iOS 9.0 and over, to make click to actions work seamlessly with the application, a few special entries are required to be made in the application plist file. To know more, please visit section “Opening custom URL schemes”.

Custom Click Action

With the latest version of SDK (2.2.0), application can launch its own screen or perform any other custom action on ad click. To achieve that, SDK introduced a new optional delegate:

Objective-C

```
-(void)xAdView:(XAdView *)xAdView shouldHandleCustomURL:(NSURL *)url{
    /**
      SDK flow is terminated.
      Publishers can take action based on the contents of click URL.
     **/
}
```

Swift

```
func xAdView(xAdview:XAdView!, shouldHandleCustomURL url:NSURL!){

    /**
      SDK flow is terminated.

      Publishers can take action based on the contents of click URL.
     **/
}
```

Whenever a user clicks the ad, SDK performs a check on click-through URL. If it detects a custom URL scheme (app://), it triggers an action, and sends a callback to application. Publishers can utilize this callback and implement their own logic to complete the click to action event inside the aforesaid optional delegate. SDK will terminate the flow once the custom URL scheme is identified.

Starting iOS 9.0 and over, to make click to actions work seamlessly with the application, a few special entries are required to be made in the application plist file. To know more, please visit section “Opening custom URL schemes”.

Please Note: This feature works only with MRAID based ads which uses mraid.open(“app://…”) method.

Other Callbacks

The SDK attempts to be as flexible as necessary to make fully robust applications using advertising possible. Although the most common use-cases were described, there are many other delegate methods available. It may be informative to glance at the XAdViewDelegate and XAdInterstitialViewDelegate API sections to familiarize yourself with what additional information it provides. Because they are all optional, feel free to use them or ignore them as needed.

Enabling SDK debug logs

To enable and see the detailed logs the publisher can set the BOOL value on enableDebugLogs property available with XGlobalConfiguration class. To set the flag, please refer to the following code snippet:

Objective-C

```
XGlobalConfiguration sharedInstance].enableDebugLogs = NO;
```

Swift

```
XGlobalConfiguration.sharedInstance.enableLogs = false
```

Customizing In-App Browser appearance

In-App Browser can be customized to better suit the publisher needs by setting different attributes on toolbar and toolbar buttons. This implementation is provided inside of the XBrowserConfiguration class that is contained by the XGlobalConfiguration.

Set Toolbar Position

Toolbar can be placed either at the top or at the bottom. The XToolbarPosition enum has the supported positions.

Objective-C

```
 XBrowserConfiguration *browserConf = [[XBrowserConfiguration alloc] init];
 browserConf.toolbarPosition = XToolbarPositionBottom;
 [[XGlobalConfiguration sharedInstance] setBrowserConfiguration:browserConf];
```

Swift

```
var browserConfiguration:XBrowserConfiguration! = XBrowserConfiguration()

 browserConfiguration.toolbarPosition = XToolbarPositionBottom
XBrowserConfiguration.sharedInstance.browserConfiguration = browserConfiguration
```

Set Toolbar Background Color

Background color of a toolbar can be set to match the publisher application UI.

Objective-C

```
 XBrowserConfiguration *browserConf = [[XBrowserConfiguration alloc] init];
 browserConf.toolbarBGColor = [UIColor whiteColor];
 [[XGlobalConfiguration sharedInstance] setBrowserConfiguration:browserConf];
```

Swift

```
 var browserConfiguration:XBrowserConfiguration! = XBrowserConfiguration()
 browserConfiguration.toolbarBGColor = UIColor.whiteColor()

XBrowserConfiguration.sharedInstance.browserConfiguration = browserConfiguration
```

Set BarStyle of the toolbar

The bar style of the toolbar can be changed to improvise the overall look and feel of the toolbar.

Objective-C

```
XBrowserConfiguration *browserConf = [[XBrowserConfiguration alloc] init];
browserConf.barStyle = UIBarStyleBlackTranslucent;
[[XGlobalConfiguration sharedInstance] setBrowserConfiguration:browserConf];
```

Swift

```
var browserConfiguration:XBrowserConfiguration! = XBrowserConfiguration()
browserConfiguration.barStyle = UIBarStyle.BlackTranslucent
XBrowserConfiguration.sharedInstance.browserConfiguration = browserConfiguration
```

Set Background Image on toolbar

Background image of the toolbar can be changed to match the theme of publisher application.

Objective-C

```
XBrowserConfiguration *browserConf = [[XBrowserConfiguration alloc] init];
browserConf.toolbarBGImageName = @"toolbarBGImage.png";
[[XGlobalConfiguration sharedInstance] setBrowserConfiguration:browserConf];
```

Swift

```
var browserConfiguration:XBrowserConfiguration! = XBrowserConfiguration()
browserConfiguration.toolbarBGImageName = @"toolbarBGImage.png"

XBrowserConfiguration.sharedInstance.browserConfiguration = browserConfiguration
```

Hiding the toolbar buttons

Specific toolbar buttons can be hiden or shown based on the publisher requirements.

Objective-C

```
XBrowserConfiguration *browserConf = [[XBrowserConfiguration alloc] init];
[browserConf hideToolbarButton:XToolbarButtonBack withValue:YES];
[[XGlobalConfiguration sharedInstance] setBrowserConfiguration:browserConf];
```

Swift

```
var browserConfiguration:XBrowserConfiguration! = XBrowserConfiguration()
browserConfiguration.hideToolbarButton(XtoolbarButtonBack:XtoolbarButtons!, withValues
true:BOOL!)
XBrowserConfiguration.sharedInstance.browserConfiguration = browserConfiguration
```

Note: Setting YES will hide the buttons, and NO, will show the buttons.

Setting Toolbar Button Images

Every toolbar button on the toolbar can have a customized image.

Objective-C

```
XBrowserConfiguration *browserConf = [[XBrowserConfiguration alloc] init];
[browserConf setToolbarButton:XToolbarButtonBack withImageName:@"back.png"];
[[XGlobalConfiguration sharedInstance] setBrowserConfiguration:browserConf];
```

Swift

```
var browserConfiguration:XBrowserConfiguration! = XBrowserConfiguration()
browserConfiguration.hideToolbarButton(XtoolbarButtonBack:XtoolbarButtons!, withImageName
"back.png":NSString!)
XBrowserConfiguration.sharedInstance.browserConfiguration = browserConfiguration
```

Setting the Countdown timer position for VAST videos

Countdown timers can now be placed at 6 different locations on the screen: Top-Left, Top-Center, Top-Right, Bottom-Left, Bottom-Center, and Bottom-Right. Below is the code snippet to demonstrate one of the positioning. Others follow the same code pattern.

Objective-C

```
XAdSlotConfiguration *configuration = [[XAdSlotConfiguration alloc] init];
configuration.countdownTimerPosition = XCountdownTimerPositionTopLeft;
```

Swift

```
var configuration:XAdSlotConfiguration! = XAdSlotConfiguration ()
configuration.countdownTimerPosition = XCountdownTimerPositionTopLeft
```

Setting the skip-offset value for VAST videos

To support configurable skip offset feature of VAST 3.0 in VAST 2.0, OAS Mobile SDK includes a new feature, which allows the publishers to set the relative or absolute value of skip offset via the ad slot configuration.
If the skip-offset type is set to relative, it would accept the skip offset time in percentage of the total ad video duration. If skip offset type is set to absolute, it would accept the skip-offset time in seconds.

Objective-C

```
XAdSlotConfiguration *configuration = [[XAdSlotConfiguration alloc] init];
[configuration setSkipOffsetTime:10];
[configuration setSkipOffsetType:XSkipOffsetRelative];
```

Swift

```
var configuration:XAdSlotConfiguration! = XAdSlotConfiguration ()
configuration.skipOffsetTime = 10

configuration.setSkipOffsetType = XSkipOffsetRelative
```

Dismissing VAST video on click through

As a default behavior, AppNexus-OAS SDK (v2.1.0 and above) pauses the video when user clicks and opens the browser. To dismiss the video ad on click, SDK provides following configuration:

Objective-C

```
XAdSlotConfiguration *configuration = [[XAdSlotConfiguration alloc] init];
[configuration setDismissVideoOnClickThrough:YES]
```

Swift

```
var configuration:XAdSlotConfiguration! = XAdSlotConfiguration ()
configuration.setDismissVideoOnClickThrough = true
```

iOS Device Based Targeting

OAS Mobile SDK v2.1.0 and above supports iOS device based targeting in the OAS server. The following table lists the devices with the device mapping in OAS for device level targeting.

Device Model	OAS Device Mapping
iPhone 4 / 4s	Apple-iPhone 4-4105196
iPhone 5 / 5s / 5c	Apple-iPhone 5-4105198
iPhone 6	Apple-iPhone 6-7180628
iPhone 6 plus	Apple-iPhone 6 Plus-7180687
iPad	Apple-iPad-1826129
iPad 2	Apple-iPad 2-4105199
iPad Retina	Apple-iPad /retina display-4107112
iPad Air	Apple-iPad /retina display-4107112
iPad Mini	Apple-iPad 2-4105199
iPad Mini 2	Apple-iPad /retina display-4107112
iPod Touch	Apple-iPod Touch-312415

OAS Mobile SDK API Reference

SDK Classes and Methods

XAdView

XVideoQuartile

This is an enum used for tracking video quartiles.

XClickToAction

This is an enum used for handling popups for click to actions. The add developer will be able to differentiate the calls with the help of these enum items.

XMediationTargetedGender

This is an enum used for assigning gender to the slot configuration mediationTargetedGender property while requesting for a mediated ad.

init

This is the constructor used to initialize the class which is the entry point to the SDK.
Returns: (id) this method returns the instantiated XAdView object

loadWithDomainName:pageName:adPosition:keywords:

This method is used to request an ad from the server based on the ad server domain, page name, container position, and keywords.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be displayed
keywords	NSString	Comma separated values to filter the ads based on the keywords

Returns: void

loadWithDomainName:pageName:adPosition:keywords:queryString:

This method is used to request an ad from the server based on the ad server domain, page name, container position, keywords, and additional query string values.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be displayed
keywords	NSString	Comma separated values to filter the ads based on the keywords
queryString	NSString	Key value pairs in the query string format for additional filtering of ads in the query string format

Returns: void

loadWithDomainName:pageName:adPosition:queryString:

This method is used to request ad from the server based on the ad server domain name, page name, container position, and query sting values.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be displayed
queryString	NSString	Key value pairs in the query string format for additional filtering of ads in the query string format

Returns: void

loadWithDomainName:pageName:adPosition:

This method is used to request ad from the server based on the ad server domain, page name, and the container position.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be

Parameter	Type	Description
		displayed

Returns: void

performClickToAction:parameters

This method is used get the control back from the app developer into the SDK after displaying the conformation dialog box to the user and accepting YES/NO from the user, after which the SDK will take control of opening the respective click to action controllers.

Parameter	Type	Description
actionType	XClickToAction	Enum for different Click to Action Events
Parameters	NSDictionary	Key/Value Pair of values required to perform the click to action event

Returns: void

appNexusOASSDKVersion

This is a static method that is used to get current SDK version
Returns: NSString

setMoviePlayerInstance

This method sets the movie player instance. This is used to provide a player to the SDK to allow a pre-roll video ad to be played in the same player as the main content video.

Parameter	Type	Description
moviePlayerInstance	MPMovieplayerController	Initializes the instance of the video player

Returns: void

Note: The moviePlayerInstance must not be playing or configured to shouldAutoplay when the instance is passed to SDK. If the moviePlayerInstance is already playing a playback, then SDK will not stop it to play the pre-roll ad. If the moviePlayerInstance controller starts regular playback while a pre-roll ad is playing, the ad stops playing immediately and the main content of the movie player controller is played.

moviePlayerInstance
This method returns an instance of MPMovieplayerController if it was set by the call to setMoviePlayerInstance earlier.
Returns: MPMovieplayerController

setDelegate

This method sets the XAdViewDelegate for the given ad.

Parameter	Type	Description
delegate	XAdViewDelegate	Delegate

Returns: void

delegate
This method returns the XAdViewDelegate for this ad.
Returns: XAdViewDelegate

setSlotConfiguration

This method sets the ad slot configuration.

Parameter	Type	Description
slotConfiguration	XAdSlotConfigurationin	Slot configuration required at ad slot level

Returns: void

slotConfiguration
This method returns the slot configuration related to this ad.
Returns: XAdSlotConfiguration

XAdSlotConfiguration

XCountdownTimerPosition

This is an Enum, which is used to set the position of the countdown timer on the video player for VAST ad types.

XSkipOffsetType

This is an Enum. This enum is used to set the type of offset for displaying delay close button. It provides two values, Absolute and Relative.

setBannerRefreshInterval

This method sets the banner refresh interval for the ads displayed.

Parameter	Type	Description
bannerRefreshInterval	float	Refresh Interval for ad in seconds.

Returns: void

Default value if not specified: 120 seconds

bannerRefreshInterval

This method returns the value of the refresh interval for the slot in seconds.
Returns: float

setCanShowCompanionAd

This method is used to indicate if this banner ad slot can also be used for video companion ad.

Parameter	Type	Description
canShowCompanionAd	BOOL	A flag indicating if this banner ad slot can also be used for video companion ads

Returns: void
Default value if not specified: NO

Note	Current version of the Mobile SDK doesn't yet support video companions – this feature will be added in the next version.

canShowCompanionAd

This method returns the flag indicating if this banner slot can be used for video companion ads.
Returns: BOOL

Note	Current version of the Mobile SDK doesn't yet support video companions – this feature will be added in the next version.

setMaintainAspectRatio

This method is used to set the flag indicating if the aspect ratio of an ad needs to be maintained when needs to be resized.

Parameter	Type	Description
maintainAspectRatio	BOOL	Maintain aspect ratio of the ad on resize

Returns: void
Default value if not specified: NO

maintainAspectRatio

This method returns the value of the maintain aspect ratio on resize flag. If the value is true it suggests that the aspect ratio for the ad is to be maintained in case the ad being resized. If the value is false, then it suggests that the aspect ratio will not be considered while expanding the ad and the ad will be expanded.
Returns: BOOL

setBackGroundImage:UIImage

This method sets the placeholder background image for the ad slot container. This image will be displayed if the ad server fails to deliver an ad.

Parameter	Type	Description
backgroundImage	UIImage	Background image for the ad slot container

Returns: void
Default value if not specified: nil

backGroundImage

This method returns the placeholder background image for the ad slot container.

Returns: UIImage

setScalingAllowed

This method will set the scaling permission for an ad slot. If the value of this flag is true then the ad is scaled; otherwise it will not be scaled.

Parameter	Type	Description
scalingAllowed	BOOL	Scaling permission for this ad slot

Returns: void
Default value if not specified: NO

scalingAllowed

This method retrieves the scaling permission flag for this ad slot.
Returns: BOOL

setAccessToGeoLocation

This method will allow the app developer to give the SDK permission for accessing geo based location service to extend the ad server capabilities. If the value is true then SDK will access the geo location to get the lat/lon and send the same to the ad server. However, this further requires permission from the device end user to access user's current location.

Parameter	Type	Description
accessToGeoLocation	BOOL	Permission for accessing geo based location.

Returns: voide
Default value if not specified: NO

Note	Current version of the Mobile SDK doesn't yet support Ad GeoTargeting – this feature will be added in the future versions.

accessToGeoLocation

This method retrieves the permission flag for geo-location service.
Returns: BOOL

Note	Current version of the Mobile SDK doesn't yet support Ad GeoTargeting – this feature will be added in the future version.

setCOPPAPermissions

This method sets the COPPA compliance flag. If set to true, then COPPA compliance mode is activated in which case only frequency capping and DAPROPS cookies are sent to the ad server.

Parameter	Type	Description
COPPAPermission	BOOL	COPPA compliance mode flag

Returns: void
Default value if not specified: NO

COPPAPermissions

This will retrieve the COPPA compliance flag as true or false.
Returns: BOOL

setRTBRequired

This method turns the Real Time Bidding (RTB) mode on/off. If RTB mode is on, then the SDKL version of the DX tag is used, otherwise SDK version is used. Also, if RTB mode is on, then SDKL version of DX structure is returned; otherwise it's JSON version of the DX structure.

Parameter	Type	Description
rtbRequired	BOOL	RTB mode

Returns: void
Default value if not specified: NO

RTBRequired

This method returns the value for RTB mode flag.
Returns: BOOL

setShouldOpenClickThroughURLInAppBrowser

This method sets the click-through mode of this ad view. If YES, the click-through opens in the SDK's inline app browser. If NO, the click-through is displayed in the device's native browser.

Parameter	Type	Description
openInBrowser	BOOL	NO to open in native browser. YES to show click-through inline.

Returns: void
Default value if not specified: NO (open in device browser)

shouldOpenClickThroughURLInAppBrowser

This method returns the value of the click-through mode.
Returns: BOOL

setCanMediate

This method sets the flag to enable or disable mediation at slot level. If set to 1, the client-side mediation will be enabled. If set to 0, then mediation will be disabled. If -1, then mediation is undefined.

Parameter	Type	Description
canMediate	int	1 or YES to enable mediation, 0 or NO to disable mediation

Returns: void
Default value if not specified: NO

canMediate

This method returns whether mediation is enabled or disabled.
Returns: int

setMediationPlacementId

This method sets the placementId at the slot level that is required for client-side mediation to serve an ad.

Parameter	Type	Description
placementId	NSString	String value to be passed as placementId to get ads from mediation network.

Returns: void
Default value if not specified: NIL

mediationPlacementId

This method returns the placementId set on slot level.
Returns: NSString

setMediationBannerWidth

This method sets the width for banners required for mediation.

Parameter	Type	Description
mediationBannerWidth	float	Sets the banner width required for mediation.

Returns: void
Default value if not specified: 0

mediationBannerWidth

This method returns the value of banner width used for mediation.
Returns: float

setMediationBannerHeight

This method sets the height for banners required by the mediation networks.

Parameter	Type	Description
mediationBannerHeight	float	Sets the banner height required by the mediation networks.

Returns: void
Default value if not specified: 0

mediationBannerHeight

This method returns the value of banner height used for mediation.
Returns: float

setMediationTargetedAge

This method sets the targeted age while requesting ads via mediation. This is an optional parameter.

Parameter	Type	Description
mediationTargetedAge	Int	Optional parameter to target ads based on age.

Returns: void
Default value if not specified: -1 (undefined)

mediationTargetedAge

This method returns the targeted age used by the mediation network.
Returns: int

setMediationTargetedGender

This method sets optional parameter to target mediated ads based on gender.

Parameter	Type	Description
mediationTargetedGender	XMediationTargetedGender	Optional parameter to target ads based on gender. -1 is undefined, XMediationTargetedGenderFemale for a female and XMediationTargetedGenderMale for a male.

Returns: void
Default value if not specified: -1 (Undefined)

mediationTargetedGender

This method returns targeted gender set for mediation.
Returns: int

setMediationTargetedKeywords

This method sets optional parameter to target mediated ads based on custom keywords.

Parameter	Type	Description
mediationTargetedKeywords	NSDictionary	Key-Value pair to set the optional keywords

Returns: void
Default value if not specified: Empty Dictionary

mediationTargetedKeywords

This method returns the custom keyword set on mediation
Returns: NSDictionary

setCountdownTimerPosition

This method sets optional position for displaying the countdown timer on vast video and pre-roll video ads.

Parameter	Type	Description
countdownTimerPosition	NSUInteger	Integer to store XCountdownTimerPosition Enum values

Returns: void
Default value if not specified: XCountdownTimerPositionTopRight

countdownTimerPosition

This method returns the position for displaying countdown timer on vast video and pre-roll video ads.
Returns: NSUInteger

setDismissVideoOnClickThrough

This method optionally sets whether to dismiss or not to dismiss the ad video on click through event.
As a default behavior, AppNexus-OAS SDK (v2.1.0 and above) pauses the ad video when user clicks and opens the browser.

Parameter	Type	Description
dismissVideoOnClickThrough	BOOL	Boolean value. **YES** will dismiss the ad video on click-through. **NO** will retain the state of the ad video on click-through

Returns: void
Default value if not specified: NO

DismissVideoOnClickThrough

This method returns the Boolean value for dismissing the ad video on click-through.
Returns: BOOL

setSkipOffsetTime

This method optionally sets the duration after which the skip button should display on a video. Offset value if defined by the creative will take precedence over this property.

Parameter	Type	Description
skipOffsetTime	NSInteger	Integer value in seconds.

Returns: void
Default value if not specified: -1

skipOffsetTime

This method returns the skip duration in integer value.
Returns: NSInteger

setSkipOffsetType

This method optionally sets the type of offset to consider for displaying skip button on video ads in VAST and pre-roll.

Parameter	Type	Description
skipOffsetType	XSkipOffsetType	Enum. **Relative:** will be in percentage of the total video time. **Absolute:** will be less than or equal to the total video time in actual.

Returns: void
Default value if not specified: XSkipOffsetAbsolute

skipOffsetType

This method returns the skipoffset type to display the skip button after specified duration.
Returns: XSkipOffsetType

XGlobalConfiguration

sharedInstance

This method is used to get the shared instance of XGlobalConfiguration.
Returns: XGlobalConfiguration

setCanMediate

This method sets the int value for mediation network.

Parameter	Type	Description
canMediate	int	-1: undefined, 1: mediation is enabled 0: mediation is disabled

Returns: void
Default value if not specified: -1 (Undefined)

canMediate

This method returns the value for mediation enabled or disabled.
Returns: int

setEnableDebugLogs

This method sets the bool value to enable or disable the SDK logs.

Parameter	Type	Description
enableDebugLogs	BOOL	YES: logs is enabled, NO: logs is disabled

Returns: void
Default value if not specified: NO

setMediationTargetedLocation

This method sets the value for user location. The mediation network to target the ads based on location will further use this.

Parameter	Type	Description
mediationTargetedLocation	CLLocation	User location

Returns: void
Default value if not specified: NIL

mediationTargetedLocation

This method returns the user location set for mediation.
Returns: CLLocation

browserConfiguration:

Returns the Browser Configuration object used to customize the In-App Browser toolbar and buttons.
Returns: XBrowserConfiguration

setBrowserConfiguration:

Sets the Browser Configuration object that has the customizable attributes for the In-App Browser

Parameter	Type	Description
browserConfiguration	XBrowserConfiguration	Browser Configuration used for customizing the toolbar and toolbar buttons used in the In-App Browser

Returns: void

XBrowserConfiguration

XToolbarButtons:

This is an enum. This enum is used to inform SDK of the affected tool bar button.

XToolbarPosition:

This is an enum. This enum tells SDK where the tool bar must be positioned on the In-App Browser

toolbarPosition:

This returns toolbarPosition to be used for In-App Browser
Returns: XToolbarPosition

setToolbarPosition:

Sets the toolbar position as required for the In-App Browser

Parameter	Type	Description
toolbarPosition	XToolbarPosition	Tool bar Position to be used

Returns: void

toolbarBGColor:

This method returns the background color used on the toolbar for In-App Browser
Returns: UIColor

setToolbarColor:

Sets the background color on the toolbar used for In-App Browser

Parameter	Type	Description
toolbarColor	UIColor	Background color used on the toolbar

Returns: void

toolbarBGImageName:

This method returns the background image used on the toolbar for In-App Browser
Returns: NSString

setToolbarBGImageName:

Sets the background image on the toolbar for In-App Browser

Parameter	Type	Description
toolbarBGImageName	NSString	Image name used by the background

Returns: void

barStyle:

This returns the bar Style used by the toolbar for In-App Browser
Returns: UIBarStyle

setBarStyle:

Sets the bar style on toolbar for In-App Browser

Parameter	Type	Description
barStyle	UIBarStyle	Bar Style used by the toolbar

Returns: void

setToolbarButton:withImageName:

Sets the image on toolbar button. Call this method multiple times to set image on multiple buttons.

Parameter	Type	Description
toolbarButton	XToolbarButton	Button on which the image is to be set
buttonImage	NSString	Name of the image to be used

Returns: void

hideToolbarButton:withValue:

Sets the visibility on the toolbar button. Call this method multiple times to show/hide multiple toolbar buttons.

Parameter	Type	Description
toolbarButton	XToolbarButton	Button on which the image is to be set
Visibility	BOOL	YES/NO. YES to hide the button and NO to show the button.

Returns: void

XAdViewDelegate

xAdViewDidLoad:

This call back is called when an ad is successfully loaded.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView that was successfully loaded

Returns: void

xAdView: didFailWithError

This call back is called when SDK encounters an error while retrieving an ad. This method is also called when the ad server successfully returns, but with no ad available.

Parameter	Type	Description
error	NSError	NSError that will contain the error description.
xAdView	XAdView	Instance of XAdView class

Returns: void

xAdViewDidClickOnAd:

This call back is called when the user clicks on the ad.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdDidExpand:

This call back is called when the ad is expanded.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdDidCollapse:

This call back is called when the ad is collapsed.

Parameter	Type	Description
xAdView	XAdView	instance of XAdView.

Returns: void

xadView:prerollDidFinishWithPlayer:

This call back is called after XAdView finishes playing or fails to play an in-stream pre-roll ad.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView
moviePlayerController	MPMoviePlayerController	MPMoviePlayerController which has finished playing the pre-roll

Returns: void

xAdViewWllLeaveApplication

This call back is called when a click-through event causes SDK to open the click-through URL in external browser.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdViewWillOpenInInAppBrowser:

This call back is called when the in-app browser is launched in response to a user click-through event.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdViewWillCloseInAppBrowser:

This call back is called when the in-app browser is closed.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdViewDidDismissOnMemoryWarning:

This call back is called when XAdView is dismissed because of an OS memory warning. Note that in the case of a preroll, the xadView:prerollDidFinishWithPlayer: is also called.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdView:didPauseVideo:

This call back is called when a video within an ad has paused.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView
currentTime	NSTimeInterval	Time at which video was paused

Returns: void

xAdView:didResume:

This call back is called when a video within an ad has resumed.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView
currentTime	NSTimeInterval	Time at which video was resumed

Returns: void

xAdView:didSkipVideo:

This call back is called when a video within an ad was skipped.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView
currentTime	NSTimeInterval	Time at which video was skipped

Returns: void

xAdView:didFinishQuartile:

This call back is called when a video within an ad has hit a quartile point.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView
Quartile	XVideoQuartile	The quartile that was hit

Returns: void

xAdViewDidEnterFullScreen:

This call back is called when a video within an ad went into fullscreen mode.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdViewDidExitFullScreen:

This call back is called when a video within an ad exited fullscreen mode.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdViewDidRewind:

This call back is called when a video within an ad is rewound.

Parameter	Type	Description
xAdView	XAdView	Instance of XAdView

Returns: void

xAdView:shouldDisplayAdOnWebViewFinishRender:

Asks the delegate if webview should display ad after webview finish rendering. If the application implements this, it should inspect the contents of the webView to interpret the contents of the HTML to detect if it is a 3rd party no-ad response. If that is the case, is should return NO. Otherwise, it should return YES. If the application doesn't implement this, the default ad processing continues normally as if a YES were returned by this method.

Parameter	Type	Description
xAdView	XAdView	The instance of XAdView
webView	UIWebView	The instance of UIWebView

Returns: BOOL

xAdView:shouldHandleClickToAction:parameters

This delegate is used to handle the popups for click to action events. App developer will use this delegate to show customized popup message with changeable title, message, and button texts on the popup. This delegate will ask if the popup is handled or not handled by the app developer. If display of custom pop-up is required, this delegate should return NO. It means that the SDK stops the flow of executing Click to Action. Moreover In order to show a popup, app developer needs to add AlertView into this delegate.

Parameter	Type	Description
xAdView	XAdView	The instance of XAdView
actionType	XClickToAction	Enum for click to action events
Parameters	NSDictionary	Key/value pairs with values required for handling the actions

Returns: BOOL

interstitialAdDismissed:xadView

This delegate notifies that the interstitial ad is dismissed and app developer can take any action on the controller.

Parameter	Type	Description
xAdView	XAdView	The instance of XAdView

Returns: void

interstitialAdDismissedOnMemoryWarning:xadView

This delegate notifies that the interstitial ad is dismissed due to memory warning and app developer can take any action on the controller.

Parameter	Type	Description
xAdView	XAdView	The instance of XAdView

Returns: void

xAdView:shouldHandleCustomURL

This delegate notifies publisher of the click-through event. Provides the click-through URL for publisher's convenience. This delegate is fired only in case of click to actions. The flow will be terminated by SDK when "app://" is encountered in the URL scheme and further handle will be provided to the publisher.

If publisher implements this delegate, then the publisher would see a console log – customURLScheme "app://" found. Publisher will handle customURL. Terminating SDK Flow.

If publisher does not implement this delegate, then a console log would be seen as follows – Publisher did not handle the customURLScheme "app://". Ignoring the request.

Parameter	Type	Description
url	NSURL	click-through URL

Returns: void

XAdInterstitialViewControllerDelegate

xAdInterstitialDidLoad:

This call back is called when an interstitial view controller successfully loads an ad.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitial:didFailWithError:

This call back is called when an ad view fails to load an ad. This method is also called when the ad server returns successfully, but with no ad available.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message
error	NSError	An NSError object describing the error that occurred

Returns: void

xAdInterstitialDidClick:

This call back is called when an interstitial ad is clicked.

Parameter	Type	Description
interstitial	interstitialAdViewController	The ad view controller sending the message

Returns: void

xAdInterstitialDidDismissOnMemoryWarning:

This call back is called when XAdInterstitialViewController is dismissed due to an OS memory warning. Note that the xAdInterstitialDismissed callback is also called.

Parameter	Type	Description
interstitial	interstitialAdViewController	The ad view controller sending the message

Returns: void

xAdInterstitialDismissed:

This call back is called when the interstitial is dismissed.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitialWllLeaveApplication

This call back is called when a click-through event causes SDK to open the click-through URL in external browser.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitialWillOpenInInAppBrowser:

This call back is called when the in-app browser is launched in response to a user click-through event.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitialWillCloseInAppBrowser:

This call back is called when the in-app browser is closed.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitial:didPauseVideo:

This call back is called when a video within an ad has paused.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message
currentTime	NSTimeInterval	Time at which video was paused

Returns: void

xAdInterstitial:didResume:

This call back is called when a video within an ad has resumed.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message
currentTime	NSTimeInterval	Time at which video was resumed

Returns: void

xAdInterstitial:didSkipVideo:

This call back is called when a video within an ad was skipped.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message
currentTime	NSTimeInterval	Time at which video was skipped

Returns: void

xAdInterstitial:didFinishQuartile:

This call back is called when a video within an ad has hit a quartile point.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message
Quartile	XVideoQuartile	The quartile that was hit

Returns: void

xAdInterstitialDidEnterFullScreen:

This call back is called when a video within an ad went into fullscreen mode.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitialDidExitFullScreen:

This call back is called when a video within an ad exited fullscreen mode.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitialDidRewind:

This call back is called when a video within an ad is rewound.

Parameter	Type	Description
interstitial	XAdInterstitialViewController	The ad view controller sending the message

Returns: void

xAdInterstitialViewController:shouldDisplayAdOnWebViewFinishRender:

Asks the delegate if webview should display ad after webview finish rendering. If the application implements this, it should inspect the contents of the webView to interpret the contents of the HTML to detect if it is a 3rd party no-ad response. If that is the case, is should return NO. Otherwise, it should return YES. If the application doesn't implement this, the default ad processing continues normally as if a YES were returned by this method.

Parameter	Type	Description
interstitial	XAdInterstitialViewcontroller	The instance of XAdView
webView	UIWebView	The instance of UIWebView

Returns: BOOL

xAdInterstitialViewController:shouldHandleClickToAction:parameters

This delegate is used to handle the popups for click to action events. App developer will use this delegate to show customized pop-up message with changeable title, message, and button texts on the popup. This delegate will ask if the popup is handled or not handled by the app developer. If display of custom pop-up is required, this delegate should return NO. It means that the SDK stops the flow of Click to Action execution. Moreover, in order to show a pop-up app developer needs to add AlertView into this delegate.

Parameter	Type	Description
xAdInterstitialViewController	XAdInterstitialViewController	Instance of XAdInterstitialviewController
actionType	XClickToAction	Enum for click to action events
Parameters	NSDictionary	Key/value pairs with values required for handling the actions

Returns: BOOL

xAdInterstitialViewController:shouldHandleCustomURL

This delegate notifies publisher of the Clickthrough event. Provides the ClickThrough URL for publisher's convenience. This delegate is fired only in case of click to actions. This delegate method expects a Boolean value to be returned. The flow will be terminated by SDK when "app" is encountered in the URL scheme and further handle will be provided to the publisher.

If publisher implements this delegate, then the publisher would see a console log – customURLScheme "app://" found. Publisher will handle customURL. Terminating SDK Flow.

If publisher does not implement this delegate, then a console log would be seen as follows – Publisher did not handle the customURLScheme "app://". Ignoring the request.

Parameter	Type	Description
xAdInterstitialViewController	XAdInterstitialViewController	Instance of XAdInterstitialviewController
url	NSURL	ClickThrough URL

Returns: void

XAdInterstitialViewController

loadWithDomainName:pageName:adPosition:keywords:

This method is used to request an ad from the server based on the ad server domain, page name, container position, and keywords.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be displayed
keywords	NSString	Comma separated values to filter the ads based on the keywords

Returns: void

loadWithDomainName:pageName:adPosition:keywords:queryString:

This method is used to request an ad from the server based on the ad server domain, page name, container position, keywords, and additional query string values.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be displayed
keywords	NSString	Comma separated values to filter the ads based on the keywords
queryString	NSString	Key value pairs in the query string format for additional filtering of ads in the query string format

Returns: void

loadWithDomainName:pageName:adPosition:queryString:

This method is used to request ad from the server based on the ad server domain name, page name, container position, and query sting values.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page

Parameter	Type	Description
adPosition	NSString	Position of the ad where it needs to be displayed
queryString	NSString	Key value pairs in the query string format for additional filtering of ads in the query string format

Returns: void

loadWithDomainName:pageName:adPosition:

This method is used to request ad from the server based on the ad server domain, page name, and the container position.

Parameter	Type	Description
domainName	NSString	Domain name of the server to request the ad
pageName	NSString	Name of the page
adPosition	NSString	Position of the ad where it needs to be displayed

Returns: void

setDelegate

This method sets the XAdInterstitialViewControllerDelegate for the given ad.

Parameter	Type	Description
delegate	XAdInterstitialViewControllerDelegate	Delegate

Returns: void

delegate

This method returns the XAdInterstitialViewControllerDelegate for this ad.
Returns: XAdInterstitialViewControllerDelegate

setSlotConfiguration

This method sets the ad slot configuration.

Parameter	Type	Description
slotConfiguration	XAdSlotConfiguratioin	Slot configuration required at ad slot level

Returns: void

slotConfiguration

This method returns the slot configuration related to this ad.
Returns: XAdSlotConfiguration

setIsVastInterstitial

This method sets the flag for vast interstitial ads

Parameter	Type	Description
isVastInterstitial	BOOL	Bool Value for vast interstitial

Returns: void

isVastInterstitial

This method returns the vast interstitial flag
Returns: BOOL

appNexusOASSDKVersion

This is a static method that is used to get current SDK version
Returns: NSString

Appendix 1: Mobile Ad Trafficking

a) In OAS, setting up house ad campaign and creative is recommended for utilizing ad slot space when no paid campaign is available.

b) When 3rd party ad campaigns are involved, setting up house ad campaign and creative is recommended for utilizing ad slot space when no paid campaigns are available.

c) Such house ad campaign and creative need to be set up in a way that prevents OAS from returning an empty ad response in the case of a passback.

Appendix 2: 3rd Party Redirect and Passback Use Cases

The following defines the use cases and expected behavior:

1. OAS returns the "no ad" DX response

   This is a common OAS use case.

	Banner	**Interstitial (both video and non-video)**	**In-stream Video**
Behavior	SDK displays the default image provided by app developer	Interstitial ad window is not displayed	No ad is played and control of the video player is returned back to the app

2. 3rd party ads trafficked in OAS as script blocks

   This is a common 3rd party ad use case.

	Banner	**Interstitial (non-video)**	**Video (both interstitial and in-stream)**
Behavior	SDK displays 3rd party ads	SDK displays 3rd party ads	n/a – this should be handled via VAST Wrappers

3. 3rd party ads trafficked in OAS as redirect (HTTP 302) creative

   This is a less common use case.

	Banner	**Interstitial (non-video)**	**Video (both interstitial and in-stream)**
Behavior in non-RTB Mode	SDK displays 3rd party ads	SDK displays 3rd party ads	n/a – this should be handled via VAST Wrappers
Behavior in RTB Mode	SDK displays the default image provided by app developer. A callback is issued that allows the app to hide the banner ad area.	Interstitial ad window is not displayed	n/a – this should be handled via VAST Wrappers

4. 3rd party ad server redirect (HTTP 302) to another 3rd party ad server

   This is a less common use case.

	Banner	Interstitial (non-video)	Video (both interstitial and in-stream)
Behavior	SDK displays 3rd party ads	SDK displays 3rd party ads	n/a – this should be handled via VAST Wrappers

5. 3rd party ad server passback to OAS

   This is a common passback use case.

	Banner	Interstitial (non-video)	Video (both interstitial and in-stream)
Behavior	SDK displays the passback targeted ad from OAS	SDK displays the passback targeted ad from OAS	n/a – this should be handled via VAST Wrappers

6. 3rd party ad server passback to OAS resulting in an empty OAS ad response

   This is a possible passback use case.

	Banner	Interstitial (non-video)	Video (both interstitial and in-stream)
Behavior	SDK displays the default image provided by app developer. A callback is issued that allows the app to hide the banner ad area.	Interstitial ad window is not displayed.	n/a – this should be handled via VAST Wrappers

7. 3rd party ad server returns empty response (equivalent to empty.gif in OAS)

   This is not a common use case.

	Banner	**Interstitial (non-video)**	**Video (both interstitial and in-stream)**
Behavior	A callback is issued that allows the app which detect a no-ad use case and returns "No", in which case SDK displays the default image provided by app developer. The app can hide the banner ad area.	A callback is issued that allows the app which detect a no-ad use case and returns "No", in which case the interstitial is not displayed.	n/a – this should be handled via VAST Wrappers