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 Android
May 19, 2015

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

Table of Contents

Getting Started

The AppNexus Open AdStream Mobile SDK for Android allows developers to incorporate mobile-friendly text and image banners as well as rich media, full-screen ads known as interstitials into applications running on an Android device. The SDK also supports pre-roll video ads as well as interstitial video ads.

System Requirements

To implement the application your need following basic requirements,

- Eclipse Juno
- Android SDK
- ADT Plug-in
- JDK 1.6

Intended Audience

Android application developers using the AppNexus Open AdStream Mobile SDK for Android

Prerequisites

To integrate AppNexus Open AdStream Mobile Ad SDK, user must first have the Android development environment added to their Eclipse development environment. The Android development SDK may be downloaded from the URL listed below. This site also provides information on how to add the Android SDK to Eclipse.

http://developer.android.com/sdk/index.html

Integrating AppNexus Open AdStream Mobile SDK

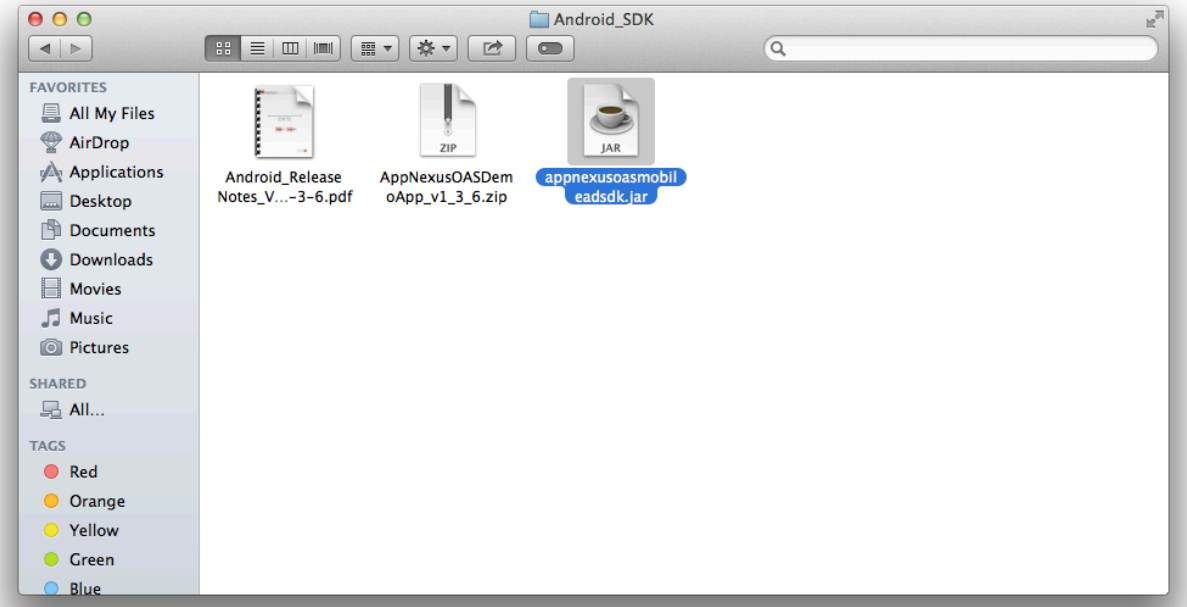
For purposes of demonstration, the rest of these instructions will use a project named **AppNexusOASDemoApp**. If your project has a different name, use this project name whenever the documentation refers to **AppNexusOASDemoApp**.

The AppNexus Open AdStream Mobile SDK for Android zip file includes an SDK library **AppNexusOASMobileAdSDK**, a demo app source code, and release notes.

First, you need to retrieve the **AppNexusOASMobileAdSDK** library (appnexusoasmobileadsdk.jar) from the zip file.

How do I locate the SDK library (appnexusoasmobileadsdk.jar)?

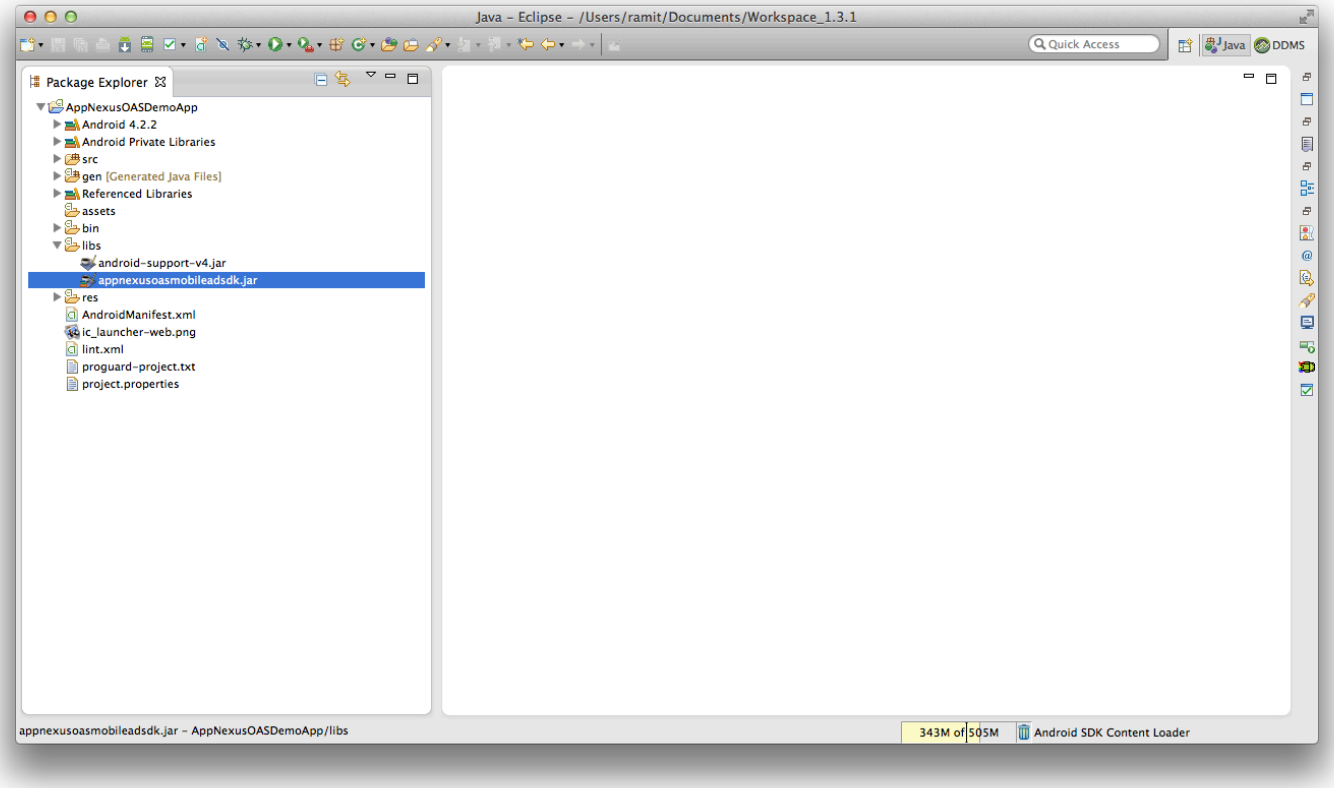
- Extract the Android SDK zip file.
- Copy the appnexusoasmobileadsdk.jar from the extracted folder.



How do I add the SDK library (appnexusoasmobileadsdk.jar) to my application?

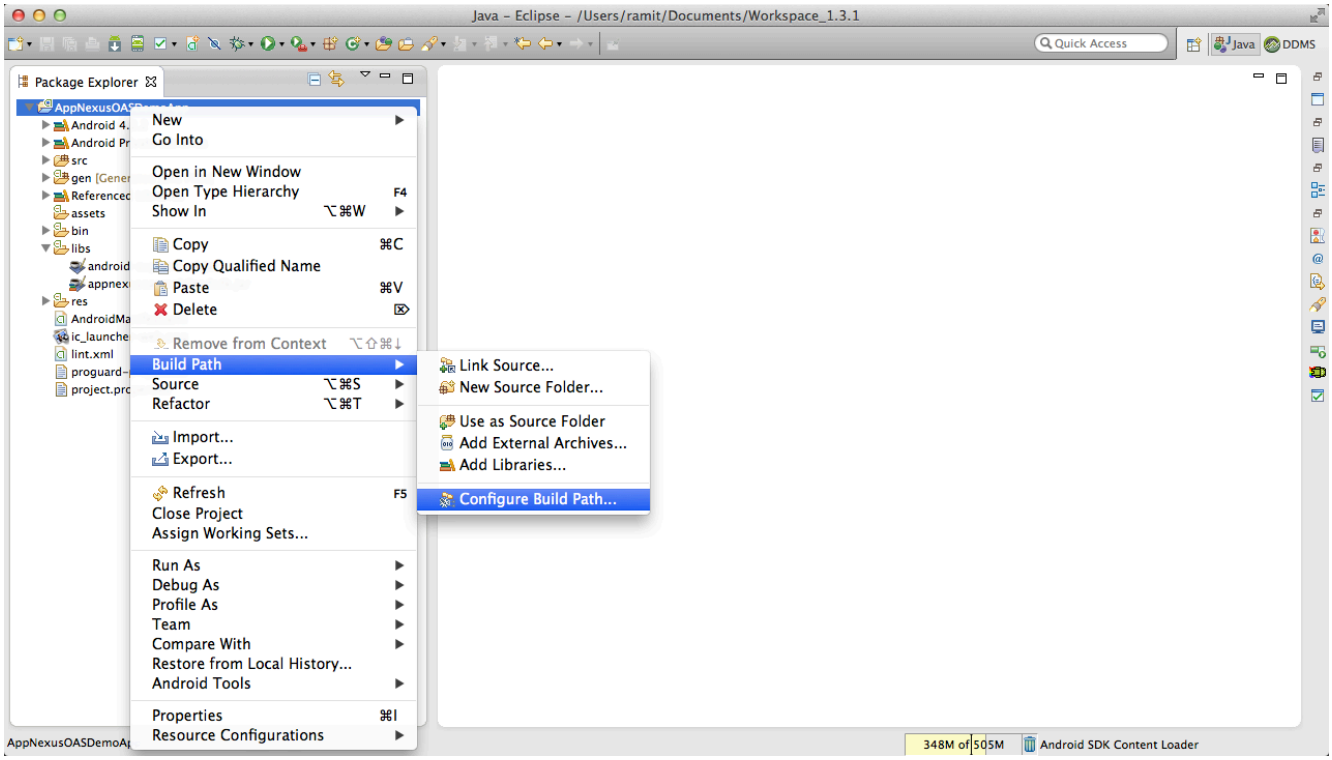
After you have copied the appnexusoasmobileadsdk.jar file, you need to copy into the **libs** folder for your application:

- Select **AppNexusOASDemoApp** Project from the project listing.
- Right-click on **libs** folder
- Select **Paste** from the menu options. This action will add appnexusoasmobileadsdk.jar to your project.

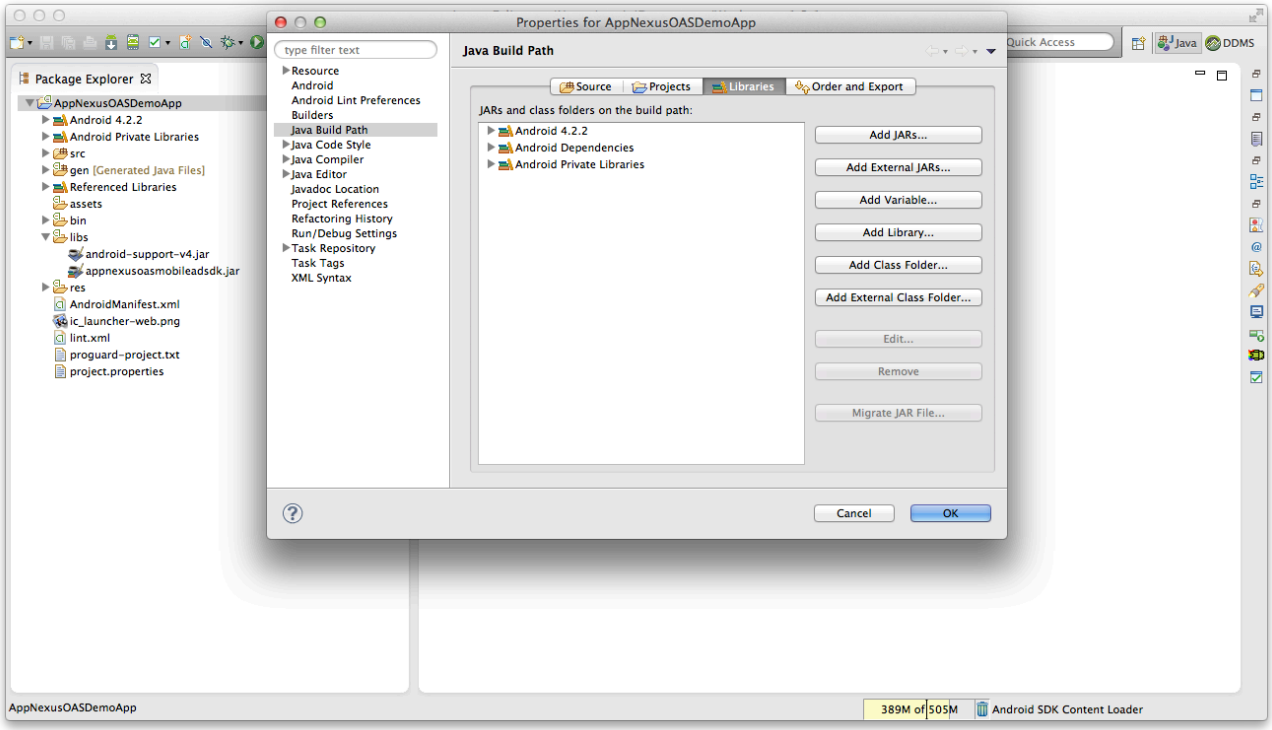


After copying the SDK jar file to the libs folder of your project, you then need to configure the build path for your application to include the appnexusoasmobileadsdk.jar file to your project:

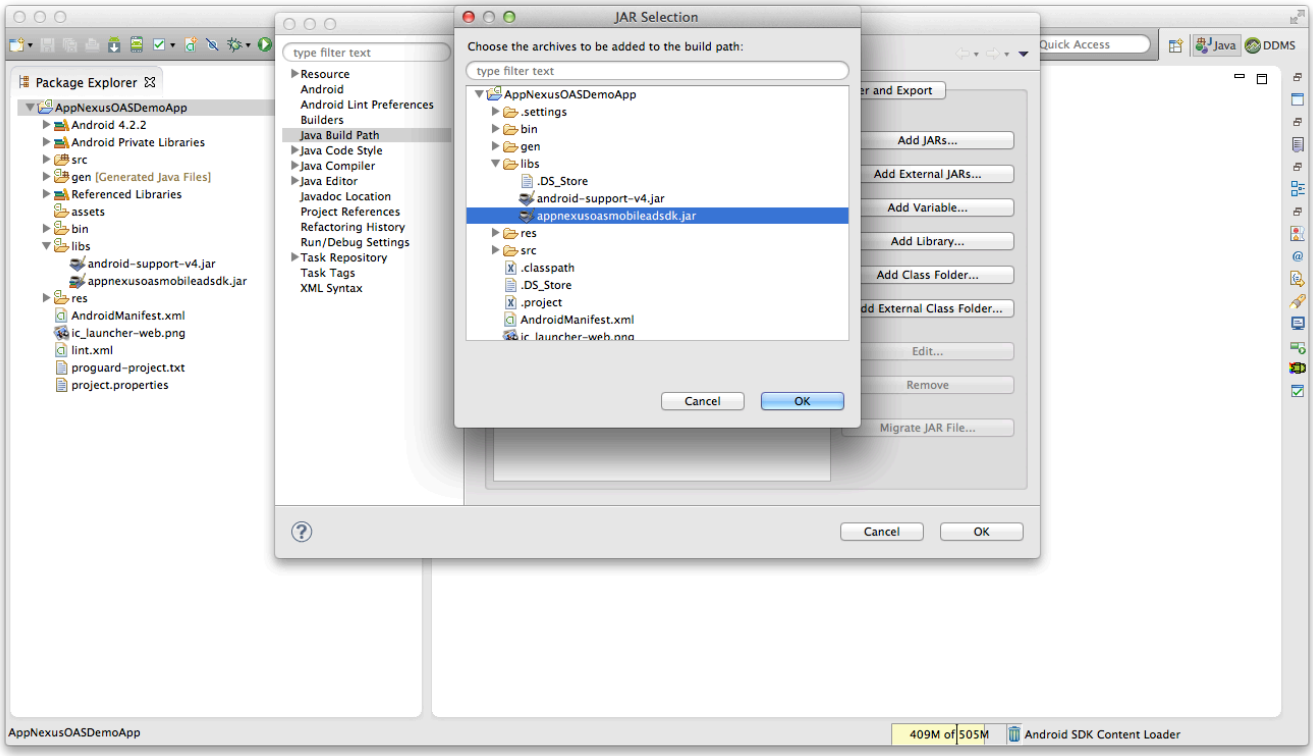
- Select **AppNexusOASDemoApp** Project from the project listing
- Right-click on the project
- Select **Build Path→Configure Build Path** menu option



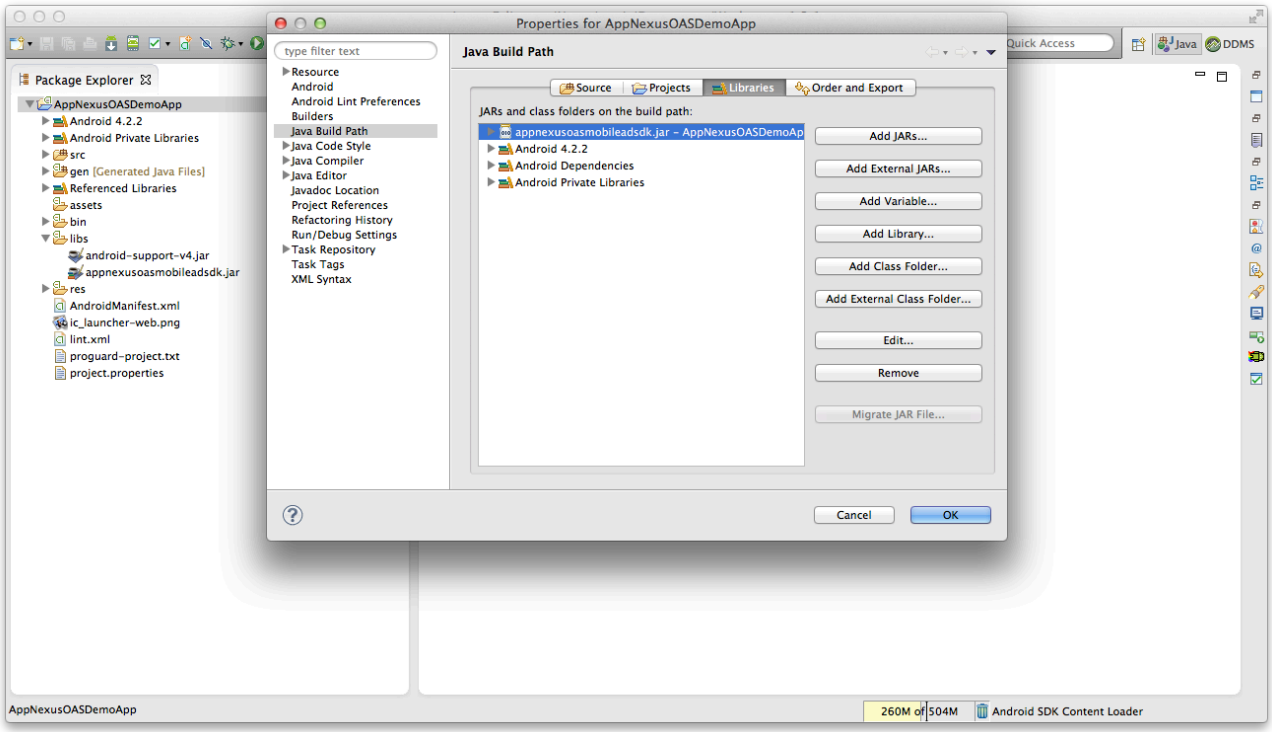
- Click the **Add JARs...** button.



- Select **appnexusoasmobileadsdk.jar** from the **AppNexusOASDemoApp** project.



- After the SDK jar file has been added to the list of Libraries in the Java Build Path, click **OK**.



Getting Started with AppNexus Open AdStream Mobile Ad SDK

The purpose of this section is to provide a quick overview on how to get started using the AppNexus Open AdStream Mobile Ad SDK to include ads in your Android applications. Sample code is provided here to illustrate how to the use SDK. The SDK zip file also contains a sample application, which you may want to import into your development environment as a means for getting more familiar with the features of this SDK.

Steps to Import Sample SDK Application into Eclipse

- Unzip **AppNexusOASDemoApp** directory from **Android_SDK.zip** to the **AppNexusOASDemoApp** directory.
- Open Eclipse and select **File →Import...** menu option.

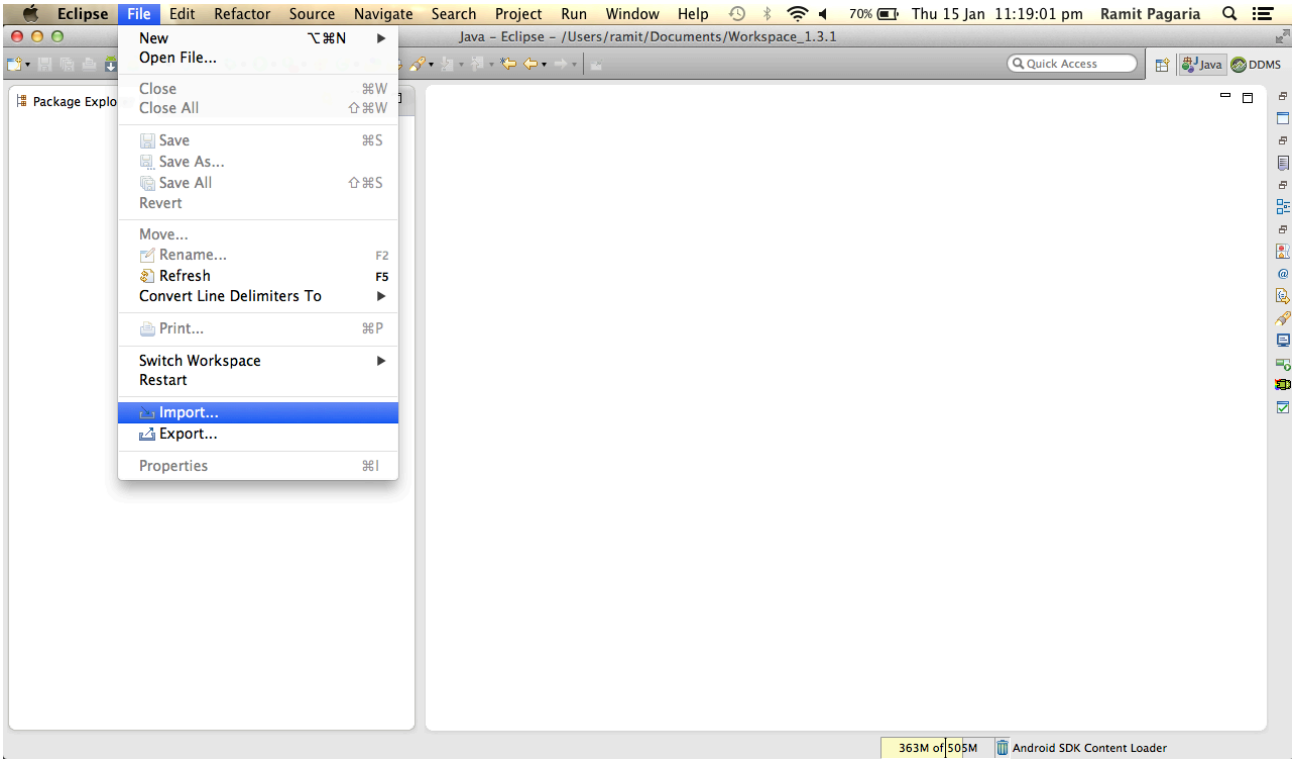


Figure 1 Import

- Select the import source type: **Android>Exisiting Android Code Into Workspace**
  The demo application will be loaded into your Eclipse environment and will be compiled and ready to run.

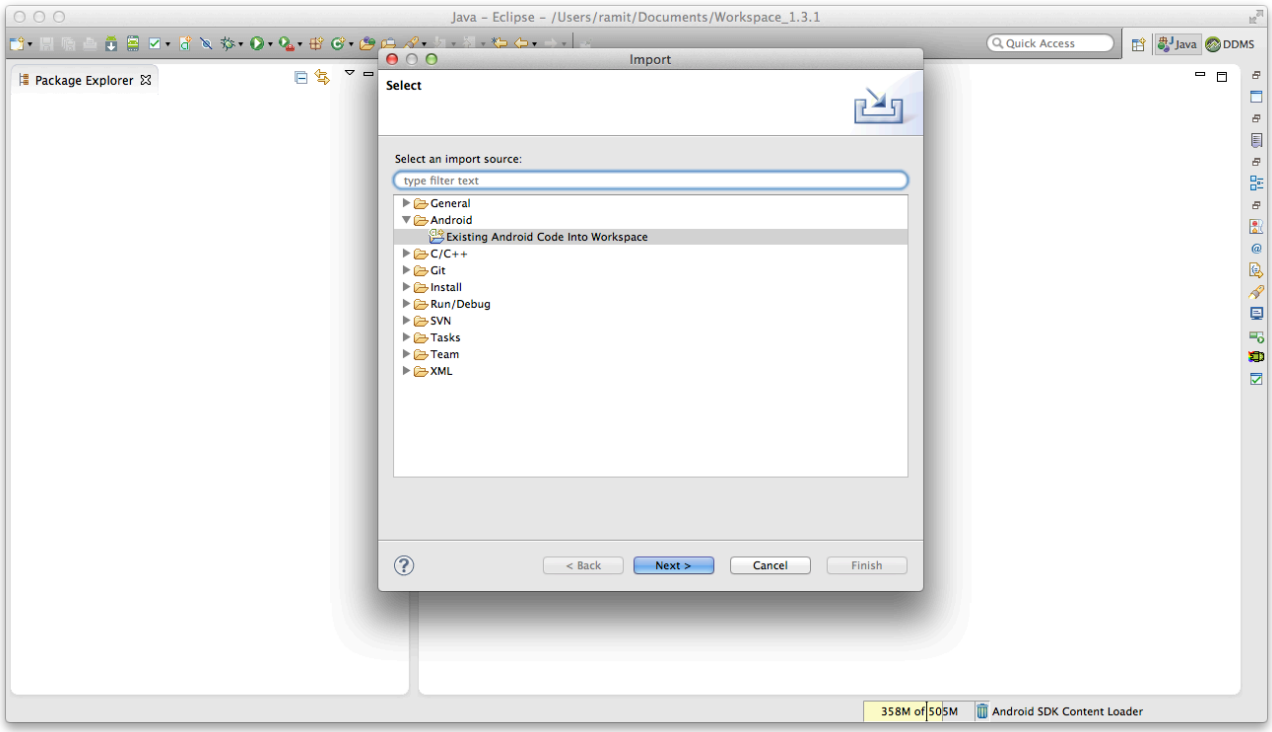


Figure 2 Import existing code

- Attach an Android device to your development PC via the USB connection.
  - Make sure you have the appropriate settings on the Android device enabled to permit launching an application on the device from Eclipse.
  - It may be helpful to have **USB Debugging** enabled. This option is listed under **Developer Options** in the **Settings** application on the device.

- From Eclipse, right click on the demo application and select **Run As→Android Application.** This should load and launch the demo application on your device.

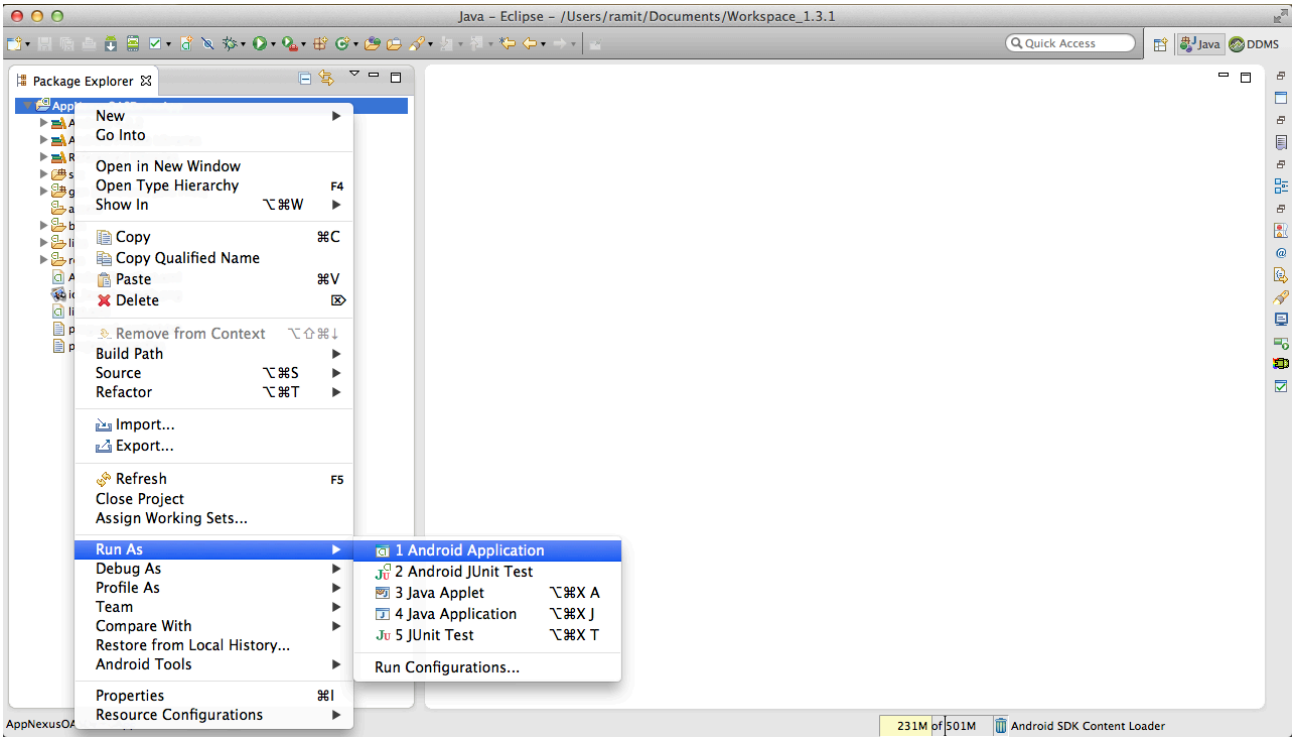


Figure 3 Run as Android Application

Getting Started With Banner Ads

Banner Ads are small display ads that may appear somewhere on the face of your application. Banner ads may simply load in a graphic or it may play a simple animation or a small video.

The AppNexus Open AdStream Mobile Ad SDK provides the **XAdView** class for the purpose of displaying a banner ad.
You may add and initialize the XAdView instance to your application by:

- Instantiating an **XAdView** object in your code, such as your Android Activity class, as shown here:

```
XAdView xAdView = new XAdView(this, adListener);
xAdView. loadAd(domainName, pageName, adPosition, queryparams, keywordParams);
```

- Or by creating the **XAdView** object in Android Activity XML, as shown here:

```
Add the following xml snippet in the layout xml file.

<com.appnexus.oas.mobilesdk.XAdView
   xmlns:oas="http://schemas.android.com/apk/lib/com.appnexus.oas.mobilesdk"
   android:id="@+id/xadview"
   android:layout_width="320dp"
   android:layout_height="50dp"
   android:layout_centerHorizontal="true"
   oas:adPosition="Bottom"
   oas:coppaEnabled="false"
   oas:domainName="delivery.uat.247realmedia.com"
   oas:pageName="demo_standardbanner"
   oas:refreshInterval="0"
   oas:rtb="false"
  oas:scale="true" />
```

- To load in a specific banner ad, you need specify parameters used to retrieve the ad tag that defines the ad. These parameters include **domainName**, **pageName**, **adPosition**, **queryParams** and **keywordParams** name. These parameters are specified in one of two ways:
    - In the call to **loadAd()** in your Activity code, or
    - As **oas:** parameters in your layout xml.
- You can add this view in your container layout (which can be linearLayout, frameLayout, relativeLayout or any other viewGroup).

XAdView communicates with your application via callbacks. **XAdView** expects to send callbacks to a class that implements the **IReceiveAd** interface defined in this SDK. The most common practice is to have your Activity or some other delegate class implement the **IReceiveAd** interface. If other optional events are needed, your Activity class or delegate must also implement these optional

interfaces. (See the Mobile SDK API Reference documentation for additional information about these interfaces.)

To receive callbacks for **IReceiveAd**:

- Save the instance of the class implementing the **IReceiveAd** in a variable.

```
IReceiveAd adListener = new IReceiveAd(){…};
```

  Note: You may use **this** in place of the variable name if your Activity class implements this interface directly.

- If you are using the constructor to create the instance of **XAdView**, pass in the reference to the interface implementor as one of the arguments in the constructor (see the API documentation on the constructor).

```
XAdView xAdView = new XAdView(this, adListener);
```

- If you are using the layout to create the instance of **XAdView**, you need first find the **XAdView** view by its ID and then use the **setAdListener()** method to pass in the reference to the implementor of the **IReceiveAd** interface.

```
findViewById(<xAdView id>).setAdListener(adListener);
```

Setting a custom banner size:

To set a custom banner size, you need to set the width and height (in dp) in the layoutParams. For setting this value in xml layout, please refer code snippet:

```
<com.appnexus.oas.mobilesdk.XAdView
    xmlns:oas="http://schemas.android.com/apk/lib/com.appnexus.oas.mobilesdk"
    android:id="@+id/xadview"
    android:layout_width="320dp"
    android:layout_height="50dp"
    android:layout_centerHorizontal="true"
    oas:adPosition="Bottom"
    oas:coppaEnabled="false"
    oas:domainName="delivery.uat.247realmedia.com"
    oas:pageName="demo_standardbanner"
    oas:refreshInterval="0"
    oas:rtb="false"
    oas:scale="true" />
```

You can also set this value programmatically using the following code snippet:

```
XAdView xAdView = new XAdView(this, adListener);
RelativeLayout.LayoutParams layoutparams = new
RelativeLayout.LayoutParams(getSizeInDP(320),getSizeInDP(50));

layoutparams.addRule(RelativeLayout.ALIGN_PARENT_TOP);
layoutparams.addRule(RelativeLayout.CENTER_HORIZONTAL);
```

```
xAdView.setLayoutParams(layoutparams);
xAdView.loadAd(domainName, pageName, adPosition, queryparams, keywordParams);
```

```
IReceiveAd adListener = new IReceiveAd() {

      @Override
      public boolean xAdShouldDisplay(View XAdView, WebView adWebView, String
htmlContent) {
            return true;
      }

      @Override
      public void xAdLoaded(View arg0) {
      }

      @Override
      public void xAdFailed(View arg0, XAdException arg1) {
      }
};
```

```
private int getSizeInDP(int pixelSize) {
      float scale = getResources().getDisplayMetrics().density;
      int sizeInDP = (int) (pixelSize * scale);
      return sizeInDP;
}
```

According to Android developer guide, defining layout dimensions with pixels is a problem because different screens have different pixel densities, so the same number of pixels may correspond to different physical sizes on different devices. Therefore, when specifying dimensions, always use either dp or sp units. A dp is a density-independent pixel that corresponds to the physical size of a pixel at 160 dpi.

As for programmatic initialization, Android API accepts the value of width and height only in pixels, so you need to convert the pixel value into density independent pixel (dp). You can do this by determining the scale according to the device's density and multiplying it with the pixel value. While using a programmatic approach, you must use the above mentioned getSizeInDP(context, pixelSize) method to achieve expected result.

SDK also accepts match_parent in xAdView's layout parameters, which would dynamically adjust the ad's dimensions according to its parent view's dimension, provided the ad creative is based on responsive html design.

Please Note:

- As MATCH_PARENT property matches the dimensions of parent view, it is necessary that xAdView has access to parent view when MATCH_PARENT is used for defining layout parameters. This can only happen if xAdView is added to parent view before xAdView.loadAd() method is called. If xAdView is added to parents view after

xAdView.loadAd() then xAdView will not have access to parents dimensions and will fall back to 320x50 size.

- If xAdView's layout parameters are specified in wrap_content then SDK would use banner's default dimensions (320x50).

Getting Started With Interstitials

Interstitial Ads are display ads, which take over the entire "page" of your application. Interstitial ads may simply load in a graphic or it may play a simple animation or a video. They may use MRAID to specify the ad or may be simple HTML/Javascript code. They also support VAST to display video.

The AppNexus Open AdStream Mobile Ad SDK provides the **XInterstitialAdDialog** class for the purpose of displaying an interstitial ad.

To create an interstitial ad:

- Create an instance of **XInterstitialAdDialog**, passing in the arguments needed to retrieve the ad tag for the ad. As with the banner ad, these parameters include: domain name, page name, position, and any query and keyword params that may be needed.

- Set the ad listener on the dialog instance. This is the object that will receive callbacks from the **XInterstitialAdDialog** object and should implement the **IReceiveAd** interface supported by this SDK. If your container class, i.e. the one instantiating **XInterstitialAdDialog**, also implements **IReceiveAd**, you may pass in **this** as the listener.

- If your interstitial ad uses VAST to display a video, you need to also set the video ad listener on the dialog instance. The listener is the object, which implements the **IVideoAd** interface.

- Once **XInterstitialAdDialog** is created and initialized, you simply call the **show()** method on the dialog instance.

```
XInterstitialAdDialog dialog = new XInterstitialAdDialog(this, domainName,
pageName, position, queryParameters, keywordParameters);
dialog.getAdSlotConfiguration().setOpenInExternalBrowser(true);

//For callbacks in interstitial ad, use this method.
//For Ad Success failure callbacks –
dialog.setAdListener(adListener);

//For Interstitial events callbacks
dialog.setInterstitialAdListener(interstitialAdListener);

//For Video events callbacks –
dialog.setVideoAdListener(videoAdListener);

dialog.show();
```

Note: The sample code above assumes that the containing class implements **IReceiveAd** interface.

Behavior of Interstitial ads with hibernation

AppNexus OAS Ad SDK has ability to detect when the app goes to background or when device's screen goes off and accordingly remove the interstitial ad from memory after a certain period (20 secs), if user doesn't return back to the app.

With respect to this feature, earlier versions of SDK (below v1.3.0) would expect a declaration of broadcast receiver in app's manifest file. But in the recent releases, SDK takes care of hibernation internally. Therefore, publishers using SDK v1.3.0 and above need not declare it in app's manifest file as it might throw an initialization exception on some devices.

Getting Started With Pre-Roll Video Ads

The AppNexus Open AdStream Mobile Ad SDK supports the ability to display pre-roll video ads. These video ads are delivered via VAST xml and are played in the application's video player, typically preceding the main content video.

The SDK provides the **XVideoAdController** class to display a pre-roll video ad. To create a pre-roll video ad using the SDK:

- The application should have a layout with includes a **RelativeLayout** object. This relative layout should be the parent of a **VideoView** where the main video and the ad video will be played. Currently, the parent of the **VideoView** object must be a **RelativeLayout**. Future versions of this SDK may support other types of parent layouts.

- The Activity class (or some delegate class) should implement three interfaces used for callbacks as needed by the application:
  - **IVideoAd** – to receive callbacks for video events such as those supported in VAST. For example, the **onPrerollAdFinished()** callback is called when the pre-roll video ad has completed.
  - **IReceiveAd** – to receive callbacks for ad loaded and ad failed events.
  - **IAdClickListener** – to receive callbacks for click events.

- Create an instance of the **XVideoAdController** class and pass in references to the callback listener objects as needed. If you do not intend to listen for a particular class of callbacks, you may pass in the value **null** for that argument.

- Once all the views have been created (including the video view), request the preroll ad by calling the **requestPrerollAd()** method on the instance of **XVideoAdController** you created. When you call this method, pass in the following information:
  - context=context of the Activity
  - videoView=the view used to play main video and ad video
  - relativeLayout=the parent layout of the video view.
  - domainName=domain name of an ad URL.
  - pageName=page within the domain which delivers the video ad.
  - position=a string containing the position where the ad needs to be displayed.
  - queryParams=(Optional) extra query parameters, if any
  - keywordParams=(Optional) extra keyword parameters, if any

- You may optionally set the video ad listener by calling **setVideoAdListener()** method on the controller, if needed. You simply pass in a reference to the object which implements the **IVideoAd** interface.

```
// create the video ad controller
XVideoAdController videoAds = new XVideoAdController(videoAdListener,
     receiveAdListener, adClickListener);
// requesting the video ad
videoAds.requestPrerollAd(getActivity(), videoView, relativeLayout,  "your-
domain-name",  "your-page-name", "your-position","", "");
```

The SDK will take the following steps in the video view to play the pre-roll video ad:

1. Set the event listener on the **VideoView** to track events.
2. Load the video ad and track the events.
3. Overlay the **Skip Ad** button on the **RelativeLayout**.
4. Once the video ad has completed or has been skipped, the SDK will remove the listeners.
5. Invoke the **onPrerollAdFinished()** callback method.

Once the **onPrerollAdFinished()** callback has been invoked, the application will have full control of the **VideoView**. You may then do the following (see example code below):

1. Set the main video content URL.
2. Set the default media controller to view
3. Play the main content video.

```
public class XVastVideoActivity extends Activity implements IVideoAd

@Override
public void onPrerollAdFinished() {
     MediaController controller = new MediaController(this);
     controller.setAnchorView(videoView);
     controller.setMediaPlayer(videoView);
     videoView.setMediaController(controller);
     uri = Uri.parse(mediaContentUrl);
     videoView.setVideoURI(uri);
     videoView.requestFocus();
     videoView.start();

}
```

NOTE: In the above example, the *mediaContentUrl* represents the content URL which the application developer wants to play.

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.

```
xVideoAdContoller.getAdSlotConfiguration().setCountdownLabelPosition(LABEL_POSIT
ION.TOP_CENTER);
```

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 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.

```
xVideoAdContoller.getAdSlotConfiguration().setSkipOffset(20,
SKIP_OFFSET_TYPE.RELATIVE);
```

Dismissing VAST video on click through

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

```
xVideoAdContoller.getAdSlotConfiguration().setDismissVideoAdOnClick(false);
```

Behavior of Vast Video Ads with other streaming apps

SDK requests the audio focus whenever any audio from streaming music app is playing in background and pauses that music momentarily until the ad is finished or skipped. This is done in order to avoid audio overlap and give a better experience to user. In case of pre-roll ads, publisher can also request for audio focus for his/her content video to avoid audio overlap.

Modifications to the Android Manifest Xml

It is recommended that you make some modifications to the AndroidManifest.xml associated with your project, as shown below.

- Add the following xml inside the application tag, if the app is using support library v4:

```
<activity
android:name="com.appnexus.oas.mobilesdk.ui.custom.XInAppBrowser"
android:configChanges="orientation|keyboardHidden|screenLayout|uiMode|screenSize
" >
</activity>
```

- In case the app uses appcompat-v7 with Theme.AppCompat as default theme, then SDK would use XInAppBrowserMaterialDesign Activity for in-app browser. Therefore following xml declaration should be used.

```
<activity
android:name="com.appnexus.oas.mobilesdk.ui.custom.XInAppBrowserMaterialDesign"
android:configChanges="orientation|keyboardHidden|screenLayout|uiMode|screenSize
" >
</activity>
```

- Add this permission to fetch the ad from server.

```
<uses-permission android:name="android.permission.INTERNET" />
```

- Add these permissions to support all kind of ad behaviors. These permissions are optional and are only required for click to action events.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.CAMERA"/>
```

NOTE: Make sure your device is connected to the Internet to fetch and display the ads.

Handling Callbacks with Delegate Interfaces

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

There are two delegate interfaces available to all ad types: IAdClickListener and IReceiveAd. Banner ads can also implement IBannerAd. Pre-roll Video and Interstitial ads can also implement IVideoAd. The complete details of callback interfaces are described in the SDK documentation.

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

Pre-roll Ads

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 **onPrerollAdFinished()** method in your implementation of **IVideoAd**. Keep in mind that an ad request may fail. In this case, **onPrerollAdFinished()** is still called, do you not need to start video playback with the xAdFailed() method in your implementation of **IReceiveAd.**

For pre-roll ads, **IReceiveAd**'s **xAdShouldDisplay()** method must be implemented if **IReceiveAd** is implemented in order to satisfy the requirements of the interface, but this method is never called by this version of the SDK. The return value of **xAdShouldDisplay()** can be either true or false.

Sample code:

```
private IVideoAd videoAdEventListener = new IVideoAd() {

	@Override
	public void onVideoSkip(long pos) {
	}

	@Override
	public void onVideoResume(long pos) {
	}

	@Override
	public void onVideoPlayerRewind(long posFrom, long posTo) {
	}

	@Override
	public void onPrerollAdFinished() {
		videoView.setVideoURI(Uri.parse("http://www.exmaple.com/video.mp4"));
		videoView.requestFocus();
		videoView.start();
	}
```

```
@Override
public void onVideoPlayerExitFullScreenMode() {
}

        @Override
        public void onVideoPlayerEnterFullScreenMode() {
        }

        @Override
        public void onVideoPlay() {
               }

        @Override
        public void onVideoPause(long pos) {
        }

        @Override
        public void onUnMuteVideo() {
        }

        @Override
        public void onQuartileFinish(int quartile) {
        }

        @Override
        public void onMuteVideo() {
        }

        @Override
        public void onVideoClick(MotionEvent arg0) {
        }
};

IReceiveAd receiveAdListenner = new IReceiveAd() {

        @Override
        public void xAdLoaded(View adView) {
        }

        @Override
        public void xAdFailed(View adView, XAdException ex) {

        }

        @Override
        //must be implemented, but this is not called for preroll video ads
        public boolean xAdShouldDisplay(View adView, WebView adWebView, String
        adWebViewClient) {
               return true;
        }
};
```

Banner 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 **xAdLoaded()** and **xAdFailed()** functions of the **IReceiveAd** interface.

Sample code:

```
IReceiveAd receiveAdListener = new IReceiveAd() {

        @Override
        public void xAdLoaded(View adView) {
                //code to show ad display area here
        }

        @Override
        public void xAdFailed(View adView, XAdException ex) {
                //code to hide ad display area here
        }

        @Override
        public boolean xAdShouldDisplay(View adView, WebView adWebView, String
        adWebViewClient) {
                //code to inspect the ad content here, always
                //returning true for now
                return true;
        }
};
```

Interstitial Presentation

Interstitials are presented in Android dialogs. The loading of the ad starts when the dialog is requested to be shown on screen, but the dialog does not show until the ad is actually loaded. When we implement an interstitial between pages A and B of an app, the dialog should be triggered when page B of the app loads. This will mean that once the interstitial is dismissed, page B will be shown to the user without any further loading. If this loading style is used, the developer might want to hide the content of page B until the dialog is dismissed. The app should set an InterstitialAd listener on the **XInterstitialAdDialog** object like so:

```
//assume dialog is an XInterstitialAdDialog object

dialog.setInterstitialAdListener(interstitialAdListener);
IInterstitialAd interstitialAdListener = new IInterstitialAd() {

        @Override
        public void onInterstitialAdClose() {
                //handle showing of page B here
        }
};
```

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, use the **xAdShouldDisplay()** function on the **IReceiveAd** callback interface.

If **xAdShouldDisplay()** returns true, then SDK handling continues normally. That is, the application will receive the ad loaded callback, and the ad will display as usual.

However, if **xAdShouldDisplay()** returns false, 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, **IReceiveAd**'s **xAdFailed()** function 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 as shown in the "Ad View Control" section above.

- In the case of an interstitial, the interstitial view will not be displayed, and the **IReceiveAd**'s **xAdFailed()** function 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 interface 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. Note that this function must nonetheless be implemented in an implementation of **IReceiveAd** in order to satisfy the interface.

Sample code:

```
IReceiveAd receiveAdListener = new IReceiveAd() {

	@Override
	public void xAdLoaded(View adView) {
	}

	@Override
	public void xAdFailed(View adView, XAdException ex) {
	}

	@Override
	public boolean xAdShouldDisplay(View adView, WebView adWebView, String
	adWebViewContent){
		boolean valid = true;
		//inspect adWebViewContent here
		if( valid ){
```

```
                return true;
            } else {
                return false;
            }
        }
};
```

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">

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 interface methods available. It may be informative to glance at the IAdClickListener, IReceiveAd, IBannerAd and IVideoAd 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.

OAS Mobile SDK API Reference

SDK Interfaces

IReceiveAd

This is an interface, which needs to be implemented by the app developer. It will provide callbacks related to ad load or ad finish to the implementing class. To do this developer needs to add the xAdView manually in the activity's main layout through the xAdLoaded(XAdView xAdView) callback.

xAdLoaded(View xAdView)

This is an interface method, which is called when the ad gets loaded successfully.

Parameter	Type	Description
xAdView	View	View object which caused this event to be called

Returns: void

xAdFailed(View xAdView, XAdException xAdError)

This method is called when the ad fails regardless of the reason.

Parameter	Type	Description
xAdView	View	View object which caused this event to be called
xAdError	XAdError	Error details

Returns: void

xAdShouldDisplay(View adView, WebView adWebView, String adWebViewContent)

This method is called when the ad fails regardless of the reason.

Parameter	Type	Description
adView	View	View object which caused this event to be called
adWebView	WebView	Web View which has been rendered with ad content
adWebViewContent	String	String representing the HTML content of the rendered Web View

Returns: Boolean- true if the ad should display, false otherwise

IVideoAd

This is an interface needs to be implemented by the app developer. It will provide callbacks related to video events to the implementing class.

onVideoPlay()

This callback is called when video starts playing.
Returns: void

onPrerollAdFinished()

This callback is called when the ad video play out is complete; it gets called regardless of failure or success of the ad play out.
Returns: void

onVideoClick(MotionEvent event)

This callback is called when the user clicks on the video ad.

Parameter	Type	Description
event	MotionEvent	MotionEvent object associated with the click

Returns: void

onVideoPause (long currentPosition)

This callback is called when the ad video is paused.

Parameter	Type	Description
currentPosition	long	Current position of the video ad

Returns: void

onVideoResume (long currentPosition)

This callback is called when the ad video is resumed.

Parameter	Type	Description
currentPosition	long	This callback is called when the ad video is

Returns: void

onVideoSkip (long currentPosition)

This callback is called when the ad video is skipped.

Parameter	Type	Description
currentPosition	long	This callback is called when the ad video is

Returns: void

onMuteVideo()

This callback is called when the ad video is muted.
Returns: void

onUnMuteVideo()

This callback is called when the ad video is unmuted.
Returns: void

onQuartileFinish(int xVideoQuartile)

This callback is called when the ad video play out reaches the appropriate quartile.

Parameter	Type	Description
xVideoQuartile	Int	Enum for the quartile stages: • xVideoQuartileFirst • xVideoQuartileMidPoint • xVideoQuartileThird

Returns: void

onVideoPlayerEnterFullScreenMode()

This callback is called when the ad video enters full screen mode.
Returns: void

onVideoPlayerExitFullScreenMode()

This callback is called when the ad video exits full screen mode.
Returns: void

onVideoPlayerRewind(long fromPosition, long toPosition)

This callback is called when the ad video is rewinded.

Parameter	Type	Description
fromPosition	long	Position is video from where the video was rewinded
toPosition	long	Position in video to where it was rewinded to

Returns: void

IBannerAd

This is an interface that may be implemented by the app developer. It will provide callbacks related to Banner events to the implementing class.

onBannerAdExpand(XAdView xAdView)

This callback is called when banner ad is expanded.

Parameter	Type	Description
xAdView	XAdView	Parameter which the app developer can use to differentiate between the ad views.

Returns: void

onBannerResize(int width, int height)

This callback is called when banner ad is resized.

Parameter	Type	Description
width	int	For banner width.
height	Int	For banner height.

Returns: void

onBannerClosed()

This callback is called when a closable banner is closed.

IInterstitialAd

This is an interface that may be implemented by the app developer. It will provide callbacks related to interstitial events to the implementing class.

onInterstitialAdClose()

This callback is called when an interstitial ad is closed.

IAdClickListener

This is an interface that may be implemented by the app developer. It will provide callbacks related to Click events to the implementing class.

onBrowserOpen(XAdView xAdView);

Gets called when user clicks on any ad and browser is opened

Parameter	Type	Description
xAdView	XAdView	Parameter which the app developer can use to differentiate between the ad views.

Returns: void

onBrowserClose(XAdView xAdView)

Gets called when Browser is closed

Parameter	Type	Description
xAdView	XAdView	Parameter which the app developer can use to differentiate between the ad views.

onLeaveApplication(XAdView xAdView)

Gets called when application goes into background from ad view

Parameter	Type	Description
xAdView	XAdView	Parameter which the app developer can use to differentiate between the ad views.

IHandleClickToAction

After implementing this interface, if the user clicks on any action event present in the ad, then following method would get called:

shouldHandleClickToAction (ACTION_TYPE actionType, Intent actionIntent);

Gets called when the user clicks on any action event, which is present in the ad. It has a return type as boolean. If returned true, then the application has to handle the action by taking appropriate input from user, and if returned false, then SDK will handle that action by itself.

If this interface is not implemented then by default SDK will handle the action.
Following click to action events are covered

1. Call
2. Calendar
3. SMS
4. Play store
5. Email

An example demonstrating the delegate with pop-up, has been implemented the demo app. It is implemented for rich media ads and MRAID interstitial ads with callback.

For store picture, there is a built-in confirmation pop-up as specified in MRAID specification. It has been implemented with 3 languages – English (US), English (UK) and French.

Parameter	Type	Description
actionType	**ACTION_TYPE**	Parameter which the app developer can use to differentiate between the types of action events.
actionIntent	**Intent**	Parameter which the app developer can use to handle the action and launch the appropriate application depending upon its actionType.

Returns: Boolean- true if the application wants to handle the action and false if SDK is handling the action.

SDK Classes

XAdView

XAdView is the ad container view which loads all types of ads depending on the ad response type. It supports standard, rich media and video ads. This view can be initialized directly from activity or through layout.

XAdView(Context context, IReceiveAd adListener)

This parameterized constructor is used to initialize the class used as an entry point to the SDK.

Parameter	Type	Description
context	Context	Application's context
adListener	IReceiveAd	Optional callback interface

Returns: (XAdView). This method returns the instantiated object of the XAdView.

loadAd(String domainName, String pageName, String adPosition, String queryParams, String keywordParams)

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	String	Domain name of the server to request the ad
pageName	String	Name of the page
adPosition	String	Position of the ad where it needs to be displayed
queryParams	String	Optional comma separated values to filter the ads based on the keywords (pass empty sting is not needed)
keywordParams	String	Optional key value pairs in the query string format for additional filtering of ads in the query string format (pass empty sting is not needed)

Returns: void

setAdClickListener(IAdClickListner adClickListener)

This method is used to add a click event listener to the ad view object.

Parameter	Type	Description
adClickListener	IAdClickListner	Attaches the IAdClickListner listener to receive the callbacks of the ad

Returns: void

setAdListener(IReceiveAd adListener)

This method is used to add an IReceiveAd listener to the ad view object.

Parameter	Type	Description
adListener	IReceiveAd	Attaches the IReceiveAd listener to receive the callbacks for the ad loaded/failure events

Returns: void

setBannerAdEventsListener(IBannerAd bannerAdEventsListener)

This method is used to add an IBannerAd listener to the ad view object.

Parameter	Type	Description
bannerAdEventsListener	IBannerAd	Attaches the IReceiveAd listener to receive the callbacks for the ad view events

Returns: void

setVideoAdListener(IVideoAd videoAdListener)

This method is used to add an IVideoAd listener to the ad view object.

Parameter	Type	Description
videoAdListener	IVideoAd	Attaches the IVideoAd listener to receive the callbacks for the video ad events

Returns: void

setAdClickToActionListener(IHandleClickToAction adClickToActionListener)

This method is used to add an IReceiveAd listener to the ad view object.

Parameter	Type	Description
adClickToActionListener	IHandleClickToAction	Attaches the IHandleClickToAction listener to receive the callbacks for the action click events present in ad.

Returns: void

clearAdView()

This method removes all the subviews from this view.
Returns: void

XAdSlotConfiguration getAdSlotConfiguration()

This method gives access to the default configuration, which the app developer can use to modify the attributes according to his/her needs.

getSDKVersion()

This method returns the current version of the SDK. Note that this is a static function that should be called on XAdView, not an instance of a XAdView object.
Returns: String

XVideoAdController

This class is responsible for playing in-stream pre-roll ads in the application's video player.

XVideoAdController(IVideoAd videoAdListener, IReceiveAd adListener, IAdClickListener clickListener)

This parameterized constructor is used to initialize the class used as an entry point to the SDK.

Parameter	Type	Description
videoAdListener	IVideoAd	Optional callback interface for video events
adListener	IReceiveAd	Optional callback interface for ad loading / failure events
clickListener	IAdClickListener	Optional callback interface for click events

Returns: (XVideoAdController). This method returns the instantiated object of the XVideoAdController.

requestPrerollAd(Context context, VideoView videoView, RelativeLayout relativeLayout, String domainName, String pageName, String position, String queryParams, String keywordParams)

This method is used to request an in-stream pre-roll ad from the server based on the ad server domain, page name, container position, keywords, and additional query string values. The video is played in the VideoView object within the specific layout object.

Parameter	Type	Description
context	Context	Application's context
videoView	VideoView	View where the ad video will play
relativeLayout	RelativeLayout	Layout wrapper containing the VideoView object
domainName	String	Domain name of the server to request the ad
pageName	String	Name of the page
adPosition	String	Position of the ad where it needs to be displayed
queryParams	String	Optional comma separated values to filter the ads based on the keywords (pass empty sting is not needed)
keywordParams	String	Optional key value pairs in the query string format for additional filtering of ads in the query string format (pass empty sting is not needed)

Returns: void

NOTE: To request a second ad video, it is best to use a thread-safe Handler.

XAdSlotConfiguration

setBannerRefreshInterval (double bannerRefreshInterval)

This method sets the ad refresh interval in seconds for a banner ad.

Parameter	Type	Description
bannerRefreshInterval	double	Refresh interval in seconds for a banner ad.

Returns: void
Default value if not set: 0

getBannerRefreshInterval()

This method returns the refresh interval in seconds for a banner ad.
Returns: double

setCanShowCompanionAds(Boolean canShowCompanionAds)

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

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

Returns: void
Default value if not specified: false

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

canShowCompanionAds()

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

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

setMaintainAspectRatio(Boolean maintainAspectRatio)

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	Boolean	Maintain aspect ratio of the ad on resize

Returns: void
Default value if not specified: false

isMaintainAspectRatio()

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

getBackgroundImage()

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

setBackgroundImage (Drawable image)

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

Parameter	Type	Description
image	Drawable	Background image for the ad slot container

setScalable(Boolean isScalable)

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
isScalable	Boolean	Scaling permission for this ad slot

Returns: void

isScalable()

This method retrieves the scaling permission flag for this ad slot.
Return: Boolean

setOpenInExternalBrowser(boolean openInBrowser)

This method sets the click-through mode of this ad view. If true, the click-through opens in the device's browser. If false, the click-through is displayed inline in a WebView control.

Parameter	Type	Description
openInBrowser	Boolean	true to open in browser, false to show click-through inline.

Returns: void

isOpenInExternalBrowser()

This method returns the value of the click-through mode.
Return: Boolean

setCOPPA (Boolean isCoppa)

This method sets the COPPA compliance flag. If set to true, then COPPA compliance mode is activated, DAPROPS cookies are sent to the ad server.

Parameter	Type	Description
isCoppa	Boolean	COPPA compliance mode flag

Returns: void

getCOPPA ()

This retrieves the COPPA compliance flag as true or false.
Returns: Boolean

setRTB (Boolean isRtb)

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.

Parameter	Type	Description
isRtb	Boolean	RTB mode

Returns: void
Default value if not specified: false

isRTB ()

This method returns the value for RTB mode flag.
Return: Boolean

setCountdownLabelPosition (LABEL_POSITION countdownLabelPosition)

This method sets label position of countdown timer for vast pre-roll and vast interstitial ads. SDK supports 6 different positions over the video view.

Parameter	Type	Description
countdownLabelPosition	LABEL_POSITION	Parameter to set the position of countdown timer position over video view for pre-roll and vast interstitial ads. Possible values: LABEL_POSITION.TOP_RIGHT LABEL_POSITION.TOP_LEFT LABEL_POSITION.TOP_CENTER LABEL_POSITION.BOTTOM_RIGHT LABEL_POSITION.BOTTOM_LEFT LABEL_POSITION.BOTTOM_CENTER

Returns: void
Default value if not specified: LABEL_POSITION.TOP_RIGHT

Please Note: This configuration would only work for VAST protocol based pre-roll and interstitial video ads. It would not work for standard or rich media ads.

getCountdownLabelPosition ()

This method returns the current position of countdown label in pre-roll ads.
Returns: LABEL_POSITION

setSkipOffset (int skipOffset, SKIP_OFFSET_TYPE skipOffsetType)

This method optionally sets the duration after which the skip button should display on an ad video. The value of skipOffset depends on the skipOffsetType parameter. If skipOffsetType is set as ABSOLUTE, then skipOffset value would be considered in seconds, otherwise it would be considered in percentage of the video ad video's total duration.

Offset value if defined by the creative will take precedence over this property.

Parameter	Type	Description
skipOffset	Int	Value of skip offset in seconds if the skipOffsetType is ABSOLUTE or in percentage if the skipOffsetType is RELATIVE
skipOffsetType	SKIP_OFFSET_TYPE	Parameter to set the type of skipOffset parameter for pre-roll and vast interstitial ads. Possible values: SKIP_OFFSET_TYPE.ABSOLUTE SKIP_OFFSET_TYPE.RELATIVE

Returns: void
Default value if not specified: SKIP_OFFSET_TYPE.ABSOLUTE

Please Note: This configuration would only work for VAST protocol based pre-roll and interstitial video ads. It would not work for standard or rich media ads.

getSkipOffset ()

This method returns the skip offset value being set for VAST pre-roll ads and interstitial ads.
Returns: int

setDismissVideoAdOnClick(boolean dismissVideoAdOnClick)

This method allows the publishers to dismiss the video ad on click. It works with pre-roll as well as VAST interstitial ads. If the value is set as "true", the video ad will be dismissed. If the value is set

as "false", then video ad would be paused on click and resumed when the user returns back. Default value is set as "false"

Parameter	Type	Description
dismissVideoAdOnClick	Boolean	It is a flag, which determines whether to dismiss the video ad on click or pause and maintain its state.

shouldDismissVideoAdOnClick()

This method returns the value being set for dismissVideoAdOnClick.
Return: Boolean
Default value if not specified: false

setMediationEnabled(Boolean isMediationEnabled)

This method sets the flag to enable or disable mediation at slot level. If set to "true", the mediation will be enabled. If set to "false", then mediation will be disabled.

Parameter	Type	Description
isMediationEnabled	boolean	True to enable mediation or false to keep mediation disabled

Returns: void
Default value if not specified: false

isMediationEnabled()

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

setMediationPlacementId(String mediationPlacementId)

This method sets the placementId at the slot level that is required by the mediation networks to serve the ad.

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

Returns: void
Default value if not specified: NIL

getMediationPlacementId()

This method returns the mediationPlacementId set on slot level.
Returns: String

setMediatedBannerSize(int mediatedBannerWidth, int mediatedBannerHeight)

This method sets the width and height for banners required by client-side mediation.

Parameter	Type	Description
mediatedBannerWidth	int	Sets the banner width required by the mediation networks.
mediatedBannerHeight	int	Sets the banner height required by the mediation networks.

Returns: void
Default value if not specified: 0

getMediatedBannerWidth()

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

getMediatedBannerHeight()

This method returns the value of banner height used by the mediation networks.
Returns: int

setAgeForMediation(String mediationAge)

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

Parameter	Type	Description
mediationAge	String	Optional parameter to target ads based on age.

Returns: void

Default value if not specified: -1 (undefined)

getAgeForMediation()

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

setGenderForMediation(MEDIATION_GENDER mediationGender)

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

Parameter	Type	Description
mediationGender	enum	Optional parameter to target ads based on gender. Possible values: MEDIATION_GENDER.UNKNOWN, MEDIATION_GENDER.MALE, MEDIATION_FEMALE

Returns: void
Default value if not specified: MEDIATION_GENDER.UNKNOWN

getGenderForMediation()

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

addCustomKeywordsForMediation(String key, String value)

This method sets optional parameter to target mediated ads based on custom keywords. Calling this method multiple times would accumulate these key value pairs in a list and send it with the mediated ad request.

Parameter	Type	Description
key	String	Key required for the optional keywords
value	String	Value required for the optional keywords

Returns: void
Default value if not specified: Empty Dictionary

XGlobalConfiguration

This is a singleton class, which is responsible to maintaining settings required for all the slots.

getInstance()

This method returns a single static instance of XGlobalConfiguration.

Returns: XGlobalConfiguration

setGlobalMediationEnabled(Boolean isMediationEnabled)

Parameter	Type	Description
isMediationEnabled	boolean	True to enable mediation or false to keep mediation disabled

Returns: void
Default value if not specified: false

isGlobalMediationEnabled()

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

setMediationTargetLocation(Location mediationTargetLocation)

This method sets the optional target location to support location based targeting for mediation.

Parameter	Type	Description
mediationTargetLocation	Location	Target native location object required for mediation

XBrowserConfiguration

This is a singleton class, which is responsible to configure in-app browser settings.

getInstance()

This method returns a single static instance of XBrowserConfiguration.

Returns: XBrowserConfiguration

hideToolbarButton(TOOLBAR_BUTTON toolbarButton, Boolean shouldBeHidden)

This method hides the specified toolbar button from the in-app browser.

Returns: Boolean

Parameter	Type	Description
toolbarButton	TOOLBAR_BUTTON	Values of TOOLBAR_BUTTON are as follows: TOOLBAR_BUTTON.BACK TOOLBAR_BUTTON.FORWARD TOOLBAR_BUTTON.REFRESH TOOLBAR_BUTTON.OPEN_IN_BROWSER
shouldBeHidden	Boolean	True to hide a specific toolbar button or false to show.

Returns: void

isToolbarButtonHidden (TOOLBAR_BUTTON toolbarButton, Boolean shouldBeHiddenByDefault)

This method returns whether the specified toolbar button is visible or not in the in-app browser.
Returns: Boolean

Please note:

- OPEN_IN_BROWSER option in the in-app browser is hidden by default. It can be easily enabled by passing *false* in the *hideToolbarButton* method.
- In-App browser's visual attributes can be easily changed by setting appropriate activity's theme in android manifest.

XInterstitialAdDialog

XInterstitialAdDialog(Context context, String domainName, String pageName, String position, String queryParams, String keywordParams)

This constructor is used to create and request an interstitial ad from the server based on the context, domain name, page, container position, as well as additional optional query string and keyword parameters.

Parameter	Type	Description
context	Context	Application's context
domainName	String	Domain name of the server to request the ad
pageName	String	Name of the page
adPosition	String	Position of the ad where it needs to be displayed
queryParams	String	Optional comma separated values to filter the ads based on the keywords (pass empty sting is not needed)
keywordParams	String	Optional key value pairs in the query string format for additional filtering of ads in the query string format (pass empty sting is not needed)

Returns: (XInterstitialAdDialog). This method returns the instantiated object of the XInterstitialAdDialog.

NOTE: To create a second interstitial dialog, it is best to use a thread-safe Handler.

setAdListener(IReceiveAd adListener)

This method attaches the ad receive listener so that once the ad is fetched from the server, callback methods for success or failure can be handled by the application.

Parameter	Type	Description
adListener	IReceiveAd	Optional callback interface

Returns: void

setVideoAdListener(IVideoAd videoAdListener)

This method attaches the video ad listener so that once the ad is fetched from the server, callback methods for video related events can be handled by the application.

Parameter	Type	Description
videoAdListener	IVideoAd	Attaches the IVideoAd listener to receive the callbacks for the video ad events

Returns: void

setClickToActionListener(IHandleClickToAction adClickToActionListener)

This method is used to add an IReceiveAd listener to the ad view object.

Parameter	Type	Description
adClickToActionListener	IHandleClickToAction	Attaches the IHandleClickToAction listener to receive the callbacks for the action click events present in ad.

Returns: void

Appendix 1: Mobile Ad Trafficking

a) In AppNexus|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 campaign is available.

c) Such house ad campaign and creative need to be set up in a way that prevents AppNexus|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 server 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 server 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 server 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 server

   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 AppNexus|OAS	SDK displays the passback targeted ad from AppNexus|OAS	n/a – this should be handled via VAST Wrappers

6. 3rd party ad server passback to OAS server 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