MOBILE SDK V2.0 FOR ANDROID

The Ooyala Mobile SDK for Android is a set of Java classes and layout controllers.

This documentation is for version 2.0 of the Mobile SDK for Android.

With the SDK you can rapidly develop rich, custom video application experiences for mobile devices. The SDK includes methods/classes to play videos, display VAST and Ooyala ads, query Ooyala content, and more, including but not limited to APIs for

- Playback
- Playlists/Channels - Thumbnails
- Info
- Related Media
- Live (New)
- Ads
- Analytics
- Fully integrated analytics to understand mobile usage
- Closed Captions
- APIs for loading and displaying closed captions through DFXP files

NOTABLE CHANGES SINCE THE PREVIOUS RELEASE

The following changes to the Mobile SDK have been made since the previous release:

- **Security improvements:** Because it was a security risk, the requirement in the sample applications to specify your API key and secret has been eliminated.
- **Closed captions:** The Mobile SDK now includes a layout controller for working with closed captions.
- **Reference documentation:** The reference documentation has been substantially improved.
- **Omniture:** A sample application integrating with Omniture is now available. Contact your account manager for the necessary software.
- **Mobile Connect (XTV):** A plug-in/SDK is available for displaying videos playing on mobile devices on connected TVs, set-top boxes, and other devices. Contact your account manager for the necessary software.

DOWNLOADING THE MOBILE SDK

You can download the Mobile SDK from *http://support.ooyala.com/developers/resources#mobile*. Click the appropriate version to download a zipfile of the Mobile SDK package.

SUPPORTED CONFIGURATIONS

The Ooyala Mobile SDK for Android supports:

- Android OS version 2.2 or later
- HTTP Live Streaming (HLS) for Android 4.0 or later. If you need to support video on Android 2.2 or later, you need a thrid-party add-on for HLS. Contact Ooyala for details on this add-on.
- Videos in MP4 container with baseline H.264 format

The SDK is known to work with Google TV, but this is not a supported configuration.

REQUIRED SKILLS

Ooyala assumes that you have the necessary programming skill or prior development experience to work productively with and with curiosity about the Mobile SDK. You should be able to investigate, analyze, and understand the SDK's source code.

GENERAL DESIGN OF THE SDK

In general, the SDK has two parts:

1. "Content management" functions, which include loading a video and its associated metadata from embed code (content identifiers), as well as querying for more videos, and "video playback," which covers controlling video playback on a predefined "skin".

   These functions are essential for playback.
2. "Layout controls" of the user interface (UI), the provided skins, and positioning of the "video surface" within the application.

   These controls can be replaced entirely, either from scratch or with the provided controls as a starting point. For a generalized approach, see *Implementing Custom Controls* on page 18.

REFERENCE DOCUMENTATION FOR THE MOBILE SDK

The Mobile SDK comes with complete reference documentation.

To see the programming documentation for the Ooyala Mobile SDK, after uzippping the paclage, with your browser or other tool, open the file `Documentation/com/ooyala/android/package-summary.html`.

ABOUT ACCESS CONTROL

The SDK is designed to work a minimum of security constraints. If you want to restrict access, you have the following options. Any other access control you might require is beyond the scope of the SDK but is available through Ooyala, such as Digital Rights Management (DRM) systems or other configurations.

- You can restrict the playback of videos to an Internet domain you have configured in Ooyala Backlot, either through the Backlot UI's detailed in the *Backlot User Guide* or with the Backlot API's publishing rule routes detailed in the *Backlot API Reference*. This is discussed in the tutorial.
- You can use Ooyala Player Token (OPT) to control access to individual assets (videos). (For documentation on Ooyala Player Token, see the Ooyala documentation site.)

If you want to use OPT, you need to implement the SDK's `EmbedTokenGenerator` interface. For an example, see any of the sample application, described briefly in *Sample Applications for the Mobile SDK* on page 38. For documentation about OPT, see *Ooyala Player Token* in the *Ooyala Content Protection Developer Guide*.

SETUP IN ECLIPSE

A few short steps are all that's needed to setup the SDK in the popular Eclipse development environment.

You need the following to get started with the SDK:

- The Mobile SDK for Android itself
- A development environment for Android applications. This guide refers to the popular Eclipse development environment.

To set up Eclipse for the Ooyala Mobile SDK for Android, follow these steps to create a skeletal project. You can reuse this skeleton again and again for every Ooyala Mobile SDK application you develop.

1. Unzip the downloaded OoyalaSDK.zip file.
2. Copy the `OoyalaSDK/OoyalaSDK.jar` file to a suitable location of your preference.
3. Open Eclipse.
4. Click **File**, select **New**, and click **Android Project**. If **Android Project** does not appear, click **File**, select **New**, and click **Other**. Then, expand **Android** and click **Android Project**.
5. Follow the steps to create a new project.
6. Add the `OoyalaSDK.jar` to your Android application. For example, you might do the following:
   - Create a `lib` folder by selecting **File**, pointing to **New**, and clicking **Folder**.
   - Drag the `OoyalaSDK.jar` to the `lib` folder.
7. Add the `OoyalaSDK.jar` to the `classpath` of your Android application. For example, you might do the following:
   a) Click **Project** and select **Properties**.
   b) Select **Java Build Path** in the left pane.
   c) Click **Add Jars**.
   d) Navigate to and select the `OoyalaSDK.jar` file.
   e) Click **OK**.
8. Select the `AndroidManifest.xml` file and do the following:
   a) Click the **Permissions** tab.
   b) Click **Add**.
   c) Select **Uses Permission** and click **OK**.
   d) From the **Name** list box, select `android.permission.INTERNET`.
   e) Click the `AndroidManifest.xml` tab to verify that the following entry has been added:

   ```
   <uses-permission android:name="android.permission.INTERNET"/>
   ```
9. Ensure the application will work with Android Version 2.2 and later by changing:

   ```
   <uses-sdk android:minSdkVersion="15" />
   ```

   to:

   ```
   <uses-sdk android:minSdkVersion="8" />
   ```
10. This step depends on your creating an actual project, whose name is shown as *project*. Double-click *project*`/res/layout/main`.
11. Select the `main.xml` tab.
12. Add the following within the `LinearLayout` element:

    ```
    <com.ooyala.android.OoyalaPlayerLayout android:id="@+id/ooyalaPlayer"
      android:layout_width="match_parent"
     android:layout_height="match_parent">
    </com.ooyala.android.OoyalaPlayerLayout>
    ```
13. Save the file.

You now have a reusable program skeleton.

TUTORIAL: GETTING STARTED WITH THE MOBILE SDK

Let's take a close look at the "Getting Started" sample application.

For this tutorial, you need:

- Your Ooyala-supplied provider code (pcode). To see your pcode, login to the Backlot UI, and go to the **ACCOUNT** tab, **Developers** subtab. The pcode is in the upper left.
- The embed code (content ID or asset ID) of a video you want to play

The `SampleApps` directory in the SDK distribution contains a `GettingStartedSampleApp` directory. This basic application shows the rudimentary steps for creating a video player.

In Eclipse (or your own development environment), open the source file:

```
OoyalaSDK-Android/SampleApps/GettingStartedSampleApp/com/ooyala/android/
sampleapp/GettingStartedSampleAppActivity.java
```

The application looks like this.

```
package com.ooyala.android.sampleapp;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import com.ooyala.android.OoyalaPlayer;
import com.ooyala.android.OoyalaPlayerLayout;
import com.ooyala.android.OoyalaPlayerLayoutController;

public class GettingStartedSampleAppActivity extends Activity {

  final String EMBED  = "yourEmbedCodeHere";  //Embed Code, or Content ID
  final String PCODE  = "yourPcodeHere";
  final String DOMAIN = "www.ooyala.com";

  /**
   * Called when the activity is first created.
   */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    OoyalaPlayerLayout playerLayout = (OoyalaPlayerLayout)
 findViewById(R.id.ooyalaPlayer);
    OoyalaPlayerLayoutController playerLayoutController = new
 OoyalaPlayerLayoutController(playerLayout, PCODE, DOMAIN);
    OoyalaPlayer player = playerLayoutController.getPlayer();
    if (player.setEmbedCode(EMBED)) {
      player.play();
    } else {
      Log.d(this.getClass().getName(), "Something Went Wrong!");
    }
  }
}
```

1. The three imports are standard for use of the Ooyala SDK:
   - `import com.ooyala.android.OoyalaPlayer;`

- `import com.ooyala.android.OoyalaPlayerLayout`; For more sophisticated uses, this `import` depends on which layout controller you want to use. For more about layouts, see *Layout Controls* on page 38.
- `import com.ooyala.android.OoyalaPlayerLayoutController;`

2. The sample application extends the standard Android `Activity` object.
3. Set the following constants:

```
final String EMBED  = "yourEmbedCodeHere";  //Embed Code, or Content ID
final String PCODE  = "yourPcodeHere";
```

   Leave the `DOMAIN` constant unchanged as `www.ooyala.com`. This works in conjunction with **Syndication Controls** (publishing rules) in Backlot. If you have set Internet domain restrictions on videos in Backlot (see the *Backlot User Guide*), the *domain* variable here can be set to one of those allowed domains. If you have not set these **Syndication Controls**, the `DOMAIN` constant here has no effect.
4. Set the layout to the `OoyalaPlayerLayout`. For more about layouts, see *Layout Controls* on page 38.
5. Create a new `OoyalaPlayerController` object (named `playerLayoutController`).
6. Instantiate a player object with the `playerLayoutController.getPlayer()` method.
7. Start the video in an `if` test with `player.setEmbedCode()` function with the identifier of the video to play (`EMBED`). This function is the primary "work horse" of the SDK.

   For your more sophiscated applications, rather than hardcoding the video identifier, you will want to pass a variable to the `setEmbedCode()` function.

SAMPLE APPLICATIONS FOR THE MOBILE SDK

The Mobile SDK comes with several sample applications, located in `OoyalaSDK-Android/SampleApps`.

Name	Location in SDK	Description
Adobe Pass DemoApp	`SampleApps/AdobePassDemoApp`	Illustrates how the Ooyala SDK can work in conjunction with Adobe Pass.
Channel Browser	`SampleApps/ChannelBrowserSampleApp`	This is a more fleshed-out example of a full-featured, including loading a list of videos to play, displaying a list of videos in the UI, and finally playing the video selected by user.
Getting Started	`SampleApps/GettingStartedSampleApp`	A basic program to illustrate the SDK. See *Tutorial: Getting Started with the Mobile SDK* on page 37.
keys	`SampleApps/keys`	Keystore files for some of the sample applications
OOView	`SampleApps/OOView`	This is the most full-featured and complex sample, including custom navigation, loading of videos based on different parameters, different ways to play video and so on.

LAYOUT CONTROLS

The `DefaultControlsSource` directory contains the default Ooyala player skin.

This directory includes source Java programs that control the layout of the video player on the device, both the default and for inline or fullscreen display, including the following:

- `OoyalaPlayerControls.java` is the program that actually implements the layout whose name you pass it.
- `OoyalaPlayerLayoutController.java` is a generic LayoutController that will work in most cases (regardless of the containing Layout type). It uses basic controls and allows additional overlays to be added. Fullscreening is done by opening a full screen Dialog and filling it with a dynamically created OoyalaPlayerLayout. Because of this, playback will be suspended and subsequently resumed during this process. As a result, fullscreening is slower than if the OoyalaPlayerLayout is embedded directly in the Activity's base layout, that base layout is a FrameLayout, and the LayoutController used is FastOoyalaPlayerLayoutController.
- `DefaultOoyalaPlayerFullscreenControls.java`
- `DefaultOoyalaPlayerInlineControls.java`
- The recommended `OptimizedOoyalaPlayerLayoutController.java` is a faster LayoutController that will work only on one specific case: The OoyalaPlayerLayout it controls is a direct child of the Activity's base layout which is a FrameLayout. This LayoutController uses basic controls and allows additional overlays to be added. Fullscreening is done by simply resizing the OoyalaPlayerLayout to fill the entire screen, which does not trigger a player reload thus causing this to be much faster at Fullscreening than OoyalaPlayerLayoutController.

Note: If you decide to use any layout other than the default, be sure to change your project's `LinearLayout` element to use the name of the layout you desire. See the final step in *Tutorial: Getting Started with the Mobile SDK* on page 37.

Preventing Restart on Tilt

This setting prevents the restarting of a video when the viewer changes the orientation of the device.

To prevent a video from restarting when the device changes orientation (vertical to horizontal or horizontal to vertical), do the following:

1. Open the application's `AndroidManifest.xml` file.
2. Add the following attribute to the `activity` element: `android:configChanges="orientation|keyboardHidden"`. For example:

```
<activity android:name=".TestOoyalaSDKActivity" android:label="@string/
app_name"
android:configChanges="orientation|keyboardHidden" >
```

Preventing Screen Blackout/Flicker

This setting prevents the device's screen from blacking out or flickering when a video starts.

In some cases, the screen of your device might simply black out. This is not due to the SDK itself but rather the underlying Android system and is perhaps due to a change to the `SurfaceView`.

To prevent a blackout or flicker on the device, add the following XML to the first layout activity of your application:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ooyala_container"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <SurfaceView android:layout_width="0dp" android:layout_height="0dp"/>
    .
    . your first layout here
    .
```

Implementing Custom Controls

Here is a general approach to implementing your own customized controls.

The SDK does not have a extension or subclassing mechanism for customizing the provided controls, but the following is a generalized approach to this advanced topic:

1. After examining the provided controllers, decide which best suits your needs.
2. Make a copy of that controller, giving it a unique filename.

   Note: Do not simply make changes to the provided source code; such changes might be lost in upgrade of future versions of the SDK.
3. Change the settings of the desired identified UI elements.

Be careful about how you change settings. A simple-seeming change can have large effects.

WORKING WITH EVENTS

You need to extend the Activity object to include Ooyala's notifications.

1. Your activity also needs to implement the Observer object. For example:

```
public class OoyalaAndroidTestAppActivity extends Activity implements
 OnClickListener, Observer
```

2. Attach your activity to the player, like this:

```
 player.addObserver(this);
```

3. Finally, in the activity, implement the `update` method:

```
@Override
 public void update(Observable arg0, Object arg1) {
   Log.d(TAG, "Notification Recieved: " + arg1 + " - state: " +
 player.getState());
   if (arg1 == OoyalaPlayer.CONTENT_TREE_READY_NOTIFICATION) {
     metadataReady = true;
     Log.d(TAG, "AD - metadata true!");
   } else if (arg1 == OoyalaPlayer.METADATA_READY_NOTIFICATION) {
     Log.d(TAG, "Woot, here is the current metadata: " +
 player.getMetadata());
   }
   // if (((String)arg1).equals(OoyalaPlayer.STATE_CHANGED_NOTIFICATION)
 && ((OoyalaPlayer)arg0).getState()
   // == State.READY) {
   // player.play();
   // }
 }
```

- arg0 is always the player instance
- arg1 is the notification

Events are defined in the header file `OoyalaPlayer.java`:

```
public static final String TIME_CHANGED_NOTIFICATION = "timeChanged";
public static final String STATE_CHANGED_NOTIFICATION = "stateChanged";
public static final String BUFFER_CHANGED_NOTIFICATION = "bufferChanged";
public static final String CONTENT_TREE_READY_NOTIFICATION =
"contentTreeReady";
public static final String AUTHORIZATION_READY_NOTIFICATION =
"authorizationReady";
public static final String ERROR_NOTIFICATION = "error";
public static final String PLAY_STARTED_NOTIFICATION = "playStarted";
public static final String PLAY_COMPLETED_NOTIFICATION = "playCompleted";
public static final String CURRENT_ITEM_CHANGED_NOTIFICATION =
"currentItemChanged";
public static final String AD_STARTED_NOTIFICATION = "adStarted";
public static final String AD_COMPLETED_NOTIFICATION = "adCompleted";
public static final String AD_SKIPPED_NOTIFICATION = "adSkipped";
public static final String AD_ERROR_NOTIFICATION = "adError";
public static final String METADATA_READY_NOTIFICATION = "metadataReady";
```

See also *Dealing with Errors* on page 41.

DEALING WITH ERRORS

You need to listen for error notifications on the player object.

First declare the following:

```
.
.
.
 public static final String ERROR_NOTIFICATION = "error" ;  /** Fires when
 an error occurs */
.
.
.
```

After an error notification is received, to determine the exact error, read the `error` property of the player object:

```
public String getError() {
  return _error;
}
```

The errors are as follows, with explanatory comments:

```
public enum OoyalaErrorCode {
  /** Authorization Response invalid */
  ERROR_AUTHORIZATION_INVALID,
  /** Content Tree Response invalid */
  ERROR_CONTENT_TREE_INVALID,
  /** Authorization failed */
```

```
    ERROR_AUTHORIZATION_FAILED,
    /** The signature of the Authorization Response is invalid */
    ERROR_AUTHORIZATION_SIGNATURE_INVALID,
    /** Content Tree Next failed */
    ERROR_CONTENT_TREE_NEXT_FAILED,
    /** An Internal Android Error. Check the Throwable properties. */
    ERROR_INTERNAL_ANDROID,
    /** Playback failed */
    ERROR_PLAYBACK_FAILED,
    /** Authorization Heartbeat failed.  Check properties. */
    ERROR_AUTHORIZATION_HEARTBEAT_FAILED,
    /** Metadata fetch failed*/
    ERROR_METADATA_FETCH_FAILED,

  };
```

AUTHORIZATION ERROR CODES

You can examine the `authCode` on your player object's current item (`player().currentItem().authCode()`) for possible errors, as shown below.

Error	Code
AUTHORIZED	0
UNAUTHORIZED_PARENT	1
UNAUTHORIZED_DOMAIN	2
UNAUTHORIZED_LOCATION	3
BEFORE_FLIGHT_TIME	4
AFTER_FLIGHT_TIME	5
OUTSIDE_RECURRING_FLIGHT_TIMES	6
BAD_EMBED_CODE	7
INVALID_SIGNATURE	8
MISSING_PARAMS	9
MISSING_RULE_SET	10
UNAUTHORIZED	11
MISSING_PCODE	12
UNAUTHORIZED_DEVICE	13
INVALID_TOKEN	14
TOKEN_EXPIRED	15
UNAUTHORIZED_MULTI_SYND_GROUP	16
PROVIDER_DELETED	17
TOO_MANY_ACTIVE_STREAMS	18
MISSING_ACCOUNT_ID	19
NO_ENTITLEMENTS_FOUND	20
NON_ENTITLED_DEVICE	21
NON_REGISTERED_DEVICE	22

LOCALIZING THE USER INTERFACE

You can set up custom localization strings for the UI text.

To change the language of the text strings in the Mobile SDK's user interface, you can add the following key/value pairs to the resource file `res/values/strings.xml`. You can then customize each value as you like:

```
<string name="oo_live">Live</string>
<string name="oo_subtitles_title">Subtitles</string>
<string name="oo_subtitles_none">None</string>
<string name="oo_subtitles_cc">Closed Captions</string>
<string name="oo_subtitles_en">English</string>
<string name="oo_subtitles_es">Spanish</string>
```

For each desired language, specify the two-character ISO 639-1 language code on the `oo_subtitles_` key. For instance, as shown above for Spanish: `oo_subtitles_es`.

To assist in your debugging, the Mobile SDK looks up values from the resource file by the classes `getContext().getResources().getIdentifier(name, "string", getContext().getPackageName()`.

INTEGRATION WITH OMNITURE ON ANDROID

The Ooyala Omniture Integration App demonstrates how you can integrate Omniture analytics capabilities into your Android SDK-based apps. Omniture analytics, now called The Adobe® Marketing Cloud Mobile libraries after Adobe's acquisition of Omniture, allows you to capture native app activity (user, usage, behavior, gestures, etc.) and send that information to Adobe servers for ingestion and use in SiteCatalyst® reporting. You can integrate Ooyala mobile SDK with Omniture SDKs through a step-by-step integration process using our sample app as a model.

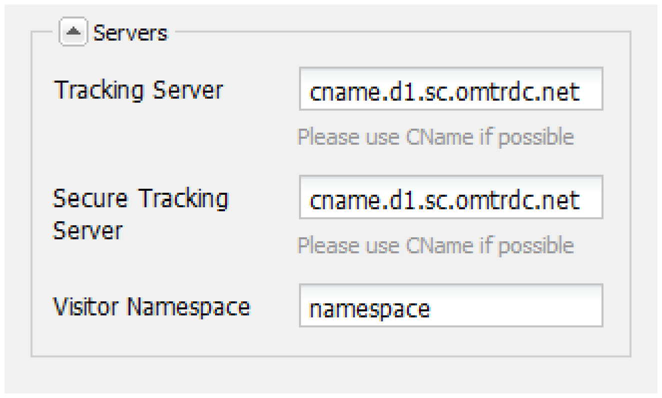
WHAT YOU NEED

To get started with Ooyala's Omniture for Android SDK Integration, you need to download the following items:

1. The *Android SDK*.
2. The *Ooyala Mobile SDK for Android*.
3. The *Ooyala Omniture Integration Sample App*
4. The *Omniture SDK*. Note that Omniture was purchased by Adobe and is now marketed as The Adobe® Marketing Cloud Mobile.
5. *Eclipse IDE*. In this guide, we use the Eclipse IDE to illustrate our integration steps.

As part of its capacity to capture analytics, Omniture draws from and provides information to the Adobe SiteCatalyst website. Before starting, you will also need to do the following:

1. Have or get an account with login credentials for Adobe's SiteCatalyst.
2. Login to *SiteCatalyst*.
3. Get the following information:
   - Report Suite ID
   - Tracking Server. The following image shows sample tracking server information from SiteCatalyst.



Servers

Tracking Server	cname.d1.sc.omtrdc.net
	Please use CName if possible
Secure Tracking Server	cname.d1.sc.omtrdc.net
	Please use CName if possible
Visitor Namespace	namespace

You will use this information in *Edit TrackingHelper.java* on page 45.

OPEN THE ANDROID SAMPLE APP

To get started, all you need to do is open our sample app and integrate a few files into your project. In the following procedure, we are using the Eclipse IDE. The Eclipse tool will help with your Android development effort. To get started with your development project, launch your Eclipse app.

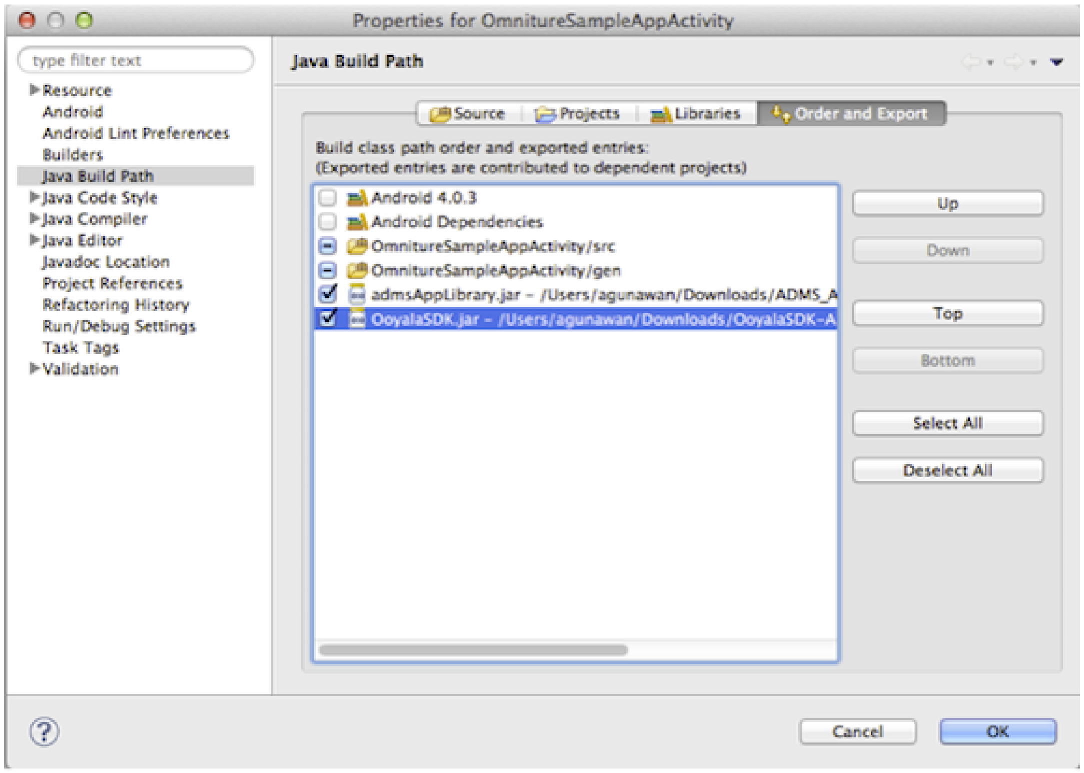
OPEN THE SAMPLE APP PROJECT

1. Click **Open Eclipse** > **File** > **Import**
2. Go to **Android** > **Existing Android Code into Workspace** > **Next**
3. Set the “Root Directory” to the extracted OmintureSampleAppActivity folder.
4. Click **Finish**.

IMPORT REQUIRED LIBRARIES

Your next step is to import some required libraries from the Omniture Android SDK into your Omniture Integration app.

1. Right-click the OmnitureSampleAppActivity project name.
2. Select **Build Path** > **Configure Build Path**...
3. Select **Java Build Path** > **Libraries** tab > **Add External Jars**.
4. Add the `admsAppLibrary.jar` that you downloaded and extracted from the Omniture SDK.jar.
5. Add the `OoyalaSDK.jar` that you downloaded and extracted from the Ooyala Android SDK.
6. Click the **Order and Export** tab.
7. Make sure the admsAppLibrary.jar and OoyalaSDK.jar are selected.



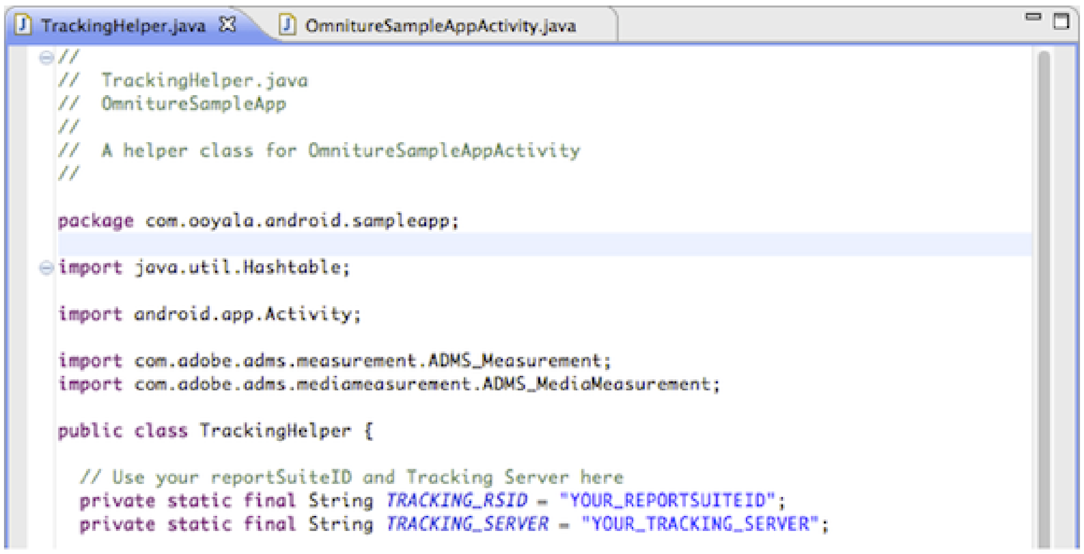
Build class path order and exported entries:
(Exported entries are contributed to dependent projects)

- [ ] Android 4.0.3
- [ ] Android Dependencies
- OmnitureSampleAppActivity/src
- OmnitureSampleAppActivity/gen
- [✓] admsAppLibrary.jar - /Users/agunawan/Downloads/ADMS_A
- [✓] OoyalaSDK.jar - /Users/agunawan/Downloads/OoyalaSDK-A

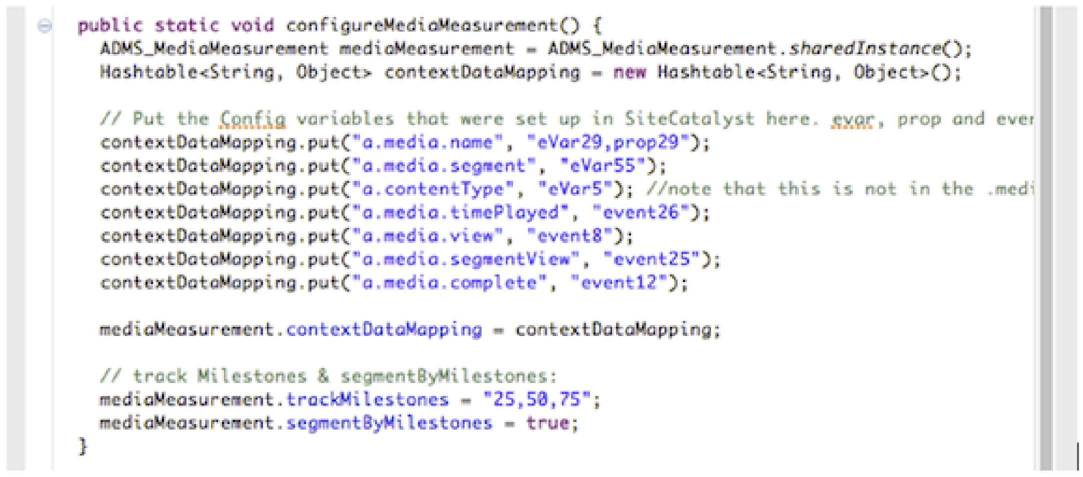
EDIT TRACKINGHELPER.JAVA

The next step in setting up your environment is to add some code to the TrackingHelper. You will need to add the following lines.

1. Open the `TrackingHelper.java` file. You will make some modifications to this file with the information you saved from SiteCatalyst, as described in *What You Need* on page 43.
2. In your `TrackingHelper.java` file, change:
   - `YOUR_REPORTSUITEID` to match the one you got from SiteCatalyst.
   - `YOUR_TRACKING_SERVER` to match the one from SiteCatalyst.
3. In the TrackingHelper.java file, you also need to change the following configuration variables so that they match the equivalent variables in SiteCatalyst:
   - `eVars` - (`s.eVar`*N*)
   - `s.props` – (`s.prop`*N*)
   - `s.events` – (`s.events`)



```
//
//  TrackingHelper.java
//  OmnitureSampleApp
//
//  A helper class for OmnitureSampleAppActivity
//

package com.ooyala.android.sampleapp;

import java.util.Hashtable;

import android.app.Activity;

import com.adobe.adms.measurement.ADMS_Measurement;
import com.adobe.adms.mediameasurement.ADMS_MediaMeasurement;

public class TrackingHelper {

  // Use your reportSuiteID and Tracking Server here
  private static final String TRACKING_RSID = "YOUR_REPORTSUITEID";
  private static final String TRACKING_SERVER = "YOUR_TRACKING_SERVER";
```



```
  public static void configureMediaMeasurement() {
    ADMS_MediaMeasurement mediaMeasurement = ADMS_MediaMeasurement.sharedInstance();
    Hashtable<String, Object> contextDataMapping = new Hashtable<String, Object>();

    // Put the Config variables that were set up in SiteCatalyst here. evar, prop and ever
    contextDataMapping.put("a.media.name", "eVar29,prop29");
    contextDataMapping.put("a.media.segment", "eVar55");
    contextDataMapping.put("a.contentType", "eVar5"); //note that this is not in the .medi
    contextDataMapping.put("a.media.timePlayed", "event26");
    contextDataMapping.put("a.media.view", "event8");
    contextDataMapping.put("a.media.segmentView", "event25");
    contextDataMapping.put("a.media.complete", "event12");

    mediaMeasurement.contextDataMapping = contextDataMapping;

    // track Milestones & segmentByMilestones:
    mediaMeasurement.trackMilestones = "25,50,75";
    mediaMeasurement.segmentByMilestones = true;
  }
```

4. If you followed these steps, your ReportSuiteID, TrackingServer and Configuration variables (eVar, prop and events) should match your SiteCatalyst configurations.
5. Although the sample contains an embed code and pcode, replace these with your own embed code and pcode before running the build.
6. Run the OmnitureSampleAppActivity.java!
7. When you run your build successfully, the Android Simulator is invoked.

You now have everything in place to run your build and test your app.

BUILD YOUR PROJECT

After you have copies all the necessary components into your development environment, select Run to build your project. If successful, you will be able to see Omniture analytics display on the SiteCatalyst web page.

TROUBLESHOOTING

If you have any trouble with your build or build results:

1. Check the SiteCatalyst web page to see if the analytics information is displaying properly.
2. Review your logs to look for any potential issues. Logs are your friend!

3. Remember that the Omniture code is developed by Adobe. If you find an issue with that code, you need to contact your relevant Adobe documentation or representative, if necessary.