Mocean Android SDK

Developer Guide

For Android SDK Version 3.3

Table of Contents

Table of Contents 2

Overview 3

Section 1 - Setup 3

What changed in 3.3: 3

Upgrading from 3.2: 4

System requirements: 4

SDK contents: 4

Installation instructions: 4

Section 2 - Getting Started with Development 10

1. User Interface / Layout (Design) 10

2. Creating a Banner Ad View 10

2.1 Layout Based Ad View Creation 11

2.2 Code Based Ad View Creation 13

2.2.1 Displaying the Ad View 13

3. Getting Initial Ad View Content 13

4. Creating an Interstitial Ad View 14

5. Native Ad Integration 17

6. Handling Rotation Changes 19

7. Detecting Ad Load Failures 20

8. Where To Go Next 22

## Overview

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

## Section 1 - Setup

### What changed in 3.3:

* The SDK has been reworked to closely match the interfaces for the iOS and Windows Phone releases. The code base has also been refactored to provide simpler application integration and less resource usage.
* Image ads now render in native image views. Animated GIFs are decoded and rendered as native image frames.
* All add configuration is set with methods directly on MASTAdView (ex: setZone and getAdRequestParameters).
* Callback interfaces have been updated and renamed. Interfaces are now set directly on MASTAdView instances.
* All public methods intended for SDK users now have javadoc documentation. HTML versions of this are included in the Documentation\javadoc folder of the SDK release, replacing the version formerly included in this document.
* Native Ads Support - Native ads allow a publisher to display ads using the same structure and formatting of their site/app without hindering the user's experience. Such ads grasp the attention of the visitors more than the traditional ad formats. Mocean and PubMatic now allow publishers to serve native ads on their inventory.

Note: See the source README.txt document for latest build release notes.

### Upgrading from 3.2:

SDK 3.3 has a new namespace to avoid collisions. Upgrading is a matter of removing the older 3.2 SDK library project/JAR and importing new 3.3 SDK library project/JAR

### System requirements:

* Android SDK (API level 8, platform version 2.2 or later)
* Eclipse 4.2 (Juno) or later
* 10 Mb free disk space

### SDK contents:

* Lib - SDK library files
* Sample - sample usage/test app

### Installation instructions:

*Installing Android SDK & Eclipse IDE with ADT Plugin*

Download and install Android SDK and the Eclipse Integrated Develop Environment (IDE) with the ADT Plug-in for Android development following the instructions at

<http://developer.android.com/sdk/installing/installing-adt.html>

If you are not comfortable with Android development, we suggest you review the online Android developer documentation available at: <http://developer.android.com/guide/index.html>.

Once SDK has been installed follow to the next step to install Android mOcean SDK.

*Installing Android mOcean SDK*

The SDK is distributed as a library source code project. To add the SDK to a project, the developer must configure the project properties to indicate the location of SDK files, as well as the names of library dependencies.

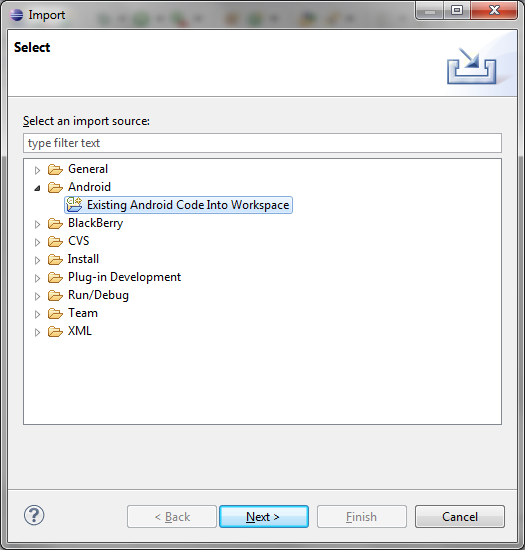
1. Unpack the SDK zip file into a convenient location in your source code working area.
2. Open or create a new Android project in the Eclipse development environment.
3. Import the SDK library project into your workspace as an existing Android project.
   1. Choose *Import* from the *File* menu, then *Existing Android Code Into Workspace* item under the Android heading, as show in Figure 1 below.  
        
      

Figure 1- Import Existing Project

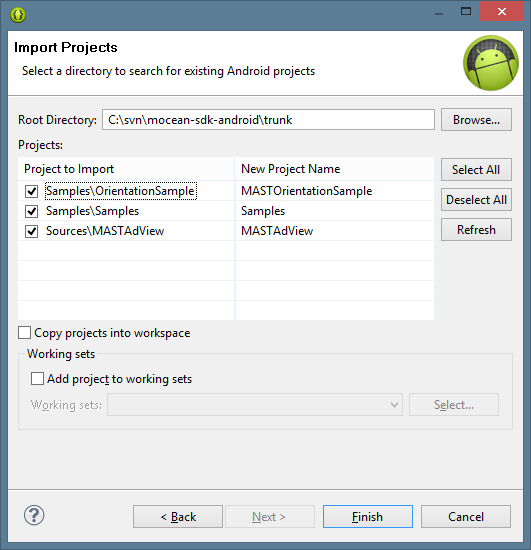
* 1. Browse to the location where you unpacked the SDK file and import the *com.moceanmobile.mast.MASTAdView* project; you can also optionally import the Samples project if you want to work with the SDK sample application. See Figure 2 below for an example.  
       
     

Figure 2 - Import SDK Project

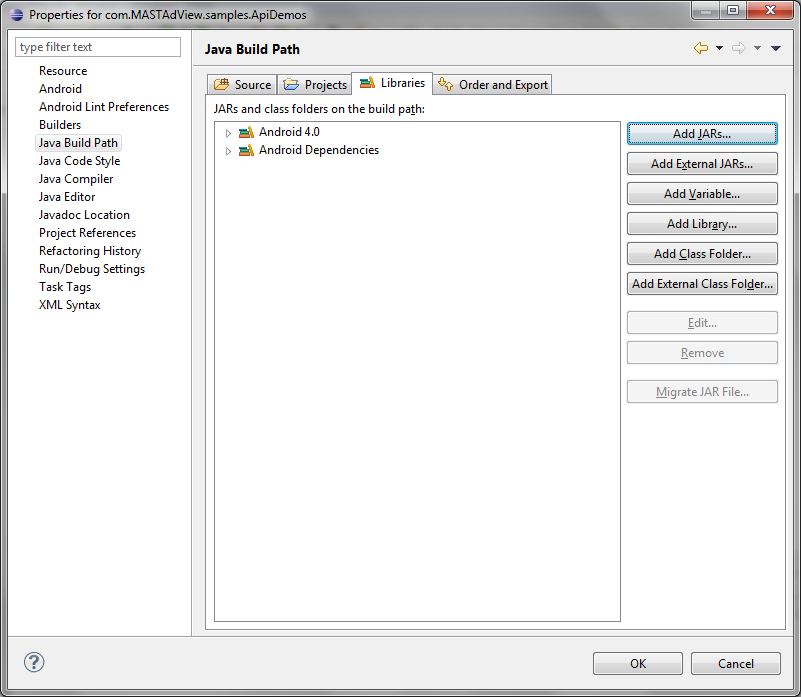
* 1. To add the SDK as library project, link the imported SDK project in your application project as android library project. Or to add the SDK as jar file dependency, use the following steps.
  2. Add the SDK project to your application project. Choose *Properties* from the *Project* menu, and then select the *Java Build Path* category followed by the *Libraries* tab, as shown in Figure 3 below.  
       
     

Figure 3 - Add Library Jar

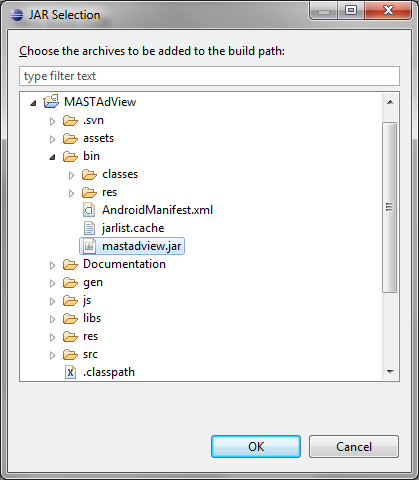
* 1. Choose the *Add Jar* button, and then navigate into the bin folder of the MASTAdView project and choose the mastadview.jar Jar file as shown in Figure 4 below.  
       
     

Figure 4 - Ad SDK Jar File

* 1. **IMPORTANT**: If using release 18 or later of the Android SDK tools, choose the *Order* *and* *Export* tab, and check the box to export the SDK Jar file as shown in Figure 5 below.   
       
     ***Without this, applications will compile but the resulting apk file will not include the required SDK code and the app will crash at runtime due to missing symbols.***

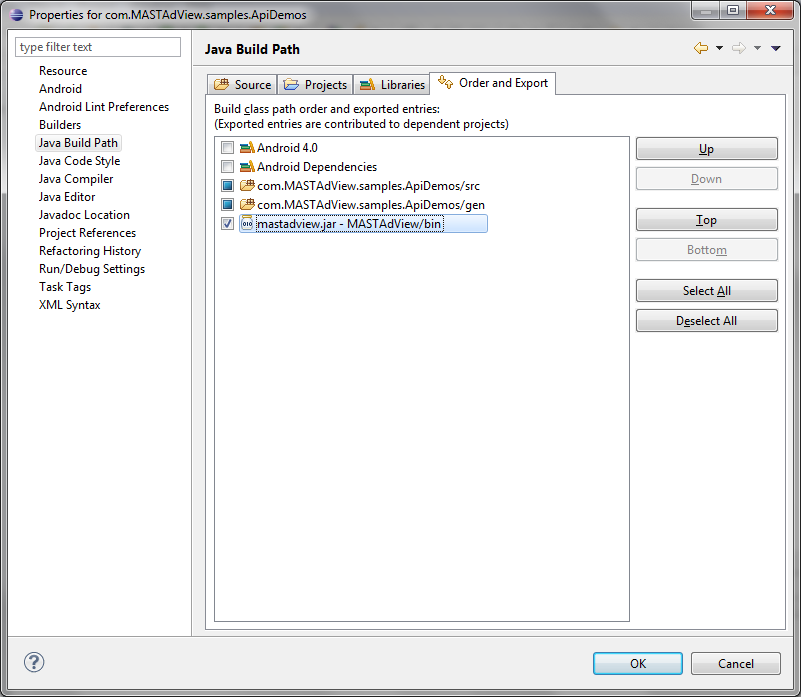


Figure 5 - Export SDK Jar File

*Updating the manifest file (AndroidManifest.xml)*

Add “minSdkVersion” parameter in project manifest file (AndroidManifest.xml)

Example: <uses-sdk android:minSdkVersion="8" />

Set the security permissions in your manifest file (AndroidManifest.xml). At a minimum you ***must*** add these permissions for the ad view to work:

|  |  |
| --- | --- |
| **Permission** | **Description & Manifest XML fragment** |
| *INTERNET* | Access the Internet. Required for ad content download.  <uses-permission android:name=*"android.permission.INTERNET"*></uses-permission> |
| *Network State* | Access the network state. Required for ad request parameter setting, and MRAID support.  <uses-permission android:name=*"android.permission.ACCESS\_NETWORK\_STATE"*></uses-permission> |

Depending on the ad content you display in your app, the following ***may*** also be needed:

|  |  |
| --- | --- |
| **Permission** | **Description & Manifest XML fragment** |
| *Fine Location* | Use GPS to obtain location information. Needed if SDK enables location detection; off by default.  <uses-permission android:name=*"android.permission.ACCESS\_FINE\_LOCATION"*></uses-permission> |
| *Phone State* | Read state of phone data connection. Required for ad request parameter setting.  <uses-permission android:name=*"android.permission.READ\_PHONE\_STATE"*></uses-permission> |
| *Read Calendar* | Read calendar events. Needed if MRAID ad makes use of calendar features.  <uses-permission android:name=*"android.permission.READ\_CALENDAR"*></uses-permission> |
| *Write Calendar* | Write calendar events. Needed if MRAID ad makes use of calendar features.  <uses-permission android:name=*"android.permission.WRITE\_CALENDAR"*></uses-permission> |
| *Call Phone* | Initiate a phone call. Needed if an ad makes use of the MRAID feature to place a phone call.  <uses-permission android:name=*"android.permission.CALL\_PHONE"*></uses-permission> |
| *Send SMS* | Send an SMS (text) message. Needed if an ad makes use of the MRAID feature to send a text message.  <uses-permission android:name=*"android.permission.SEND\_SMS"*></uses-permission> |
| *External Storage* | Access the SD card storage area. Required for debug logs, photo, and file access to support SDK logging and MRAID features.  <uses-permission android:name=*"android.permission.WRITE\_EXTERNAL\_STORAGE"*></uses-permission> |

## Section 2 - Getting Started with Development

## User Interface / Layout (Design)

The first step is deciding where you want to incorporate ads in your application. There are two basic ad types to consider:

* Banner ads, which are typically intermingled with ad content; banner ads usually span the screen width but occupy only a small part of the horizontal space.
* Interstitial ads, which are full screen ads frequently displayed when the app is launched, or when transitioning between screens or functions in the application.

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

|  |
| --- |
| FV-Top-Banner.png  FV-No-Ad.png |

Figure 2.2 - UI with Top Banner Ad

Figure 2.1 - Original UI

## Creating a Banner Ad View

Once you know where you want to put an ad in your UI, the next step is to create the ad view. As is typical with any Android UI element, the ad view component can be added to your activity in one of two ways: dynamically by creating the view in code and adding it to a layout, or in an XML layout definition.

An example of creating a banner ad with each approach follows. Note that these examples show a small set of the ad view properties that developers can use to customize the appearance and behavior of the ad view. The full set is described in the SDK documentation, and since the ad view itself is an extended version of standard Android views, the full set of view properties are also available for use by the developer as needed.

### 2.1 Layout Based Ad View Creation

Open the layout XML file for your activity and insert the com.MASTAdView.MASTAdView component into the XML view. An example of how this might look is as follows:

|  |
| --- |
| <! -- Main layout manager for this activity -- >  <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:id=*"@+id/mainManager"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  android:orientation=*"vertical"*  android:background=*"#000000"*>  <! -- Ad view component -- >  <com.moceanmobile.mast.MASTAdView  xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:id=*"@+id/mainTopAdView"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"100px"*  android:layout\_gravity=*"center\_horizontal"*  zone=*"22222" (sample, see description below)*  android:visibility=*"visible"* />  <! -- Any remaining layout -- >  <! -- … -- > |

Figure 2.1 - XML Ad Layout

Here the MASTAdView component has been added as the first visible element inside a standard (vertical) LinearLayout manager (because we chose to display this as a banner ad at the top of the application UI area.)

In addition to the standard view properties (such as the id) there are a variety of ad view properties (such as the zone) that can be configured in the XML layout, as described in the SDK documentation. This example shows a few of each, chosen for this sample application, including:

* **View ID**: this is important if you will be using code to manipulate this ad view later; this is a common practice and will be illustrated below.
* **Layout Width**: we have chosen to make this a full width banner ad, so the standard “*fill\_parent*” attribute is used. Alternatively, a fixed pixel size could be specified.  
  *We do NOT recommend using the “wrap\_content” attribute for your ad view. It is best to choose a size that will fit your UI needs to specify it here.*
* **Layout Height**: we have chosen to use a fixed 100 pixel banner ad, so this specific size is used. The standard Android variations such as density independent pixels (dip) are generally advised when configuring a pixel size to aid with supporting multiple devices.  
  *We do NOT recommend using the “wrap\_content” attribute for your ad view. It is best to choose a size that will fit your UI needs to specify it here.*
* **Ad** **zone**: this is used to identify one specific ad placement in your application. In this example we have created one placement so far, the banner ad to be displayed at the top of the screen. If we choose to display ads in another part of this application, a different placement will be used for that location. Zones are provided by your Mocean account representative, or created through the Motion Mobile UI, and target content to ad placements in your application. A given zone falls under one site. The zone is required in order to request an ad.

These same parameters can be set (or updated) in code. Consult the documentation distributed with the SDK for more information about the full range of configurable parameters and options.

### 2.2 Code Based Ad View Creation

In the java code for your activity, first be sure to import our object definitions into your java class with the statement:

|  |
| --- |
| import com.MASTAdView.MASTAdView; |

Then use code such as the following to create and setup a MASTAdView component:

|  |
| --- |
| // Construct view and set the zone registered with mocean mobile ui  int myAdZone = 22222; // *sample, see description above*  MASTAdView adView = new MASTAdView(this);  // Set the zone  adView.setZone(myAdZone);  // Set update interval  adView. setUpdateInterval (60);  // Set layout: full width of screen, 100 pixels tall  ViewGroup.LayoutParams layoutParams =  new ViewGroup.LayoutParams(ViewGroup.LayoutParams.FILL\_PARENT, 100));  // Add this view to the application UI activity  LinearLayout linearLayout = (LinearLayout)findViewById(R.id.frameAdContent);  linearLaout.addView(adView, layoutParams); |

Figure 2.2 - Code Ad View

The code above sets similar properties to those previously shown in the XML layout, notably:

* The zone is configured.
* The update time is set to the same 60 second interval.
* The view layout parameters are set to the same values: the full width of the screen, and 100 pixels tall.

### 2.2.1 Displaying the Ad View

Once the ad view has been create and configured, it has to be add to the activity layout in the appropriate spot. In this example, a LinearLayout manager named “frameAdContent” exists in the overall layout definition for this activity, placed where we want the ad to appear. This makes placing the ad a simple process of finding the named layout manager and adding our newly created ad view object to it.

## Getting Initial Ad View Content

Once the ad view has been setup, the initial ad needs to be fetched for display to the user.

If you used the XML layout option, the initial fetch is automatic. If you created the view with code, invoke the *update*() method on it, as shown below:

|  |
| --- |
| // Update view to fetch first ad  adView.update(); |

Figure 3.1 - Fetch Initial Ad Content

From this point on, operation is identical whether the XML or code approach was used to create the ad view component. In particular, the *update*() method can be invoked to download the initial ad content. The SDK itself does the rest of the work, including spawning threads to download ad content from the network without slowing down the UI, etc. The adserverView object can also be used to customize and manage other properties and behaviors as described in the SDK documentation.

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

## Creating an Interstitial Ad View

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

Unlike banner ads, you do not need to add an interstitial view to a layout. Instead, you use the custom ad view *showInterstitial*() method, and the ad will pop up in front of your activity screens until dismissed. Because they don’t appear in layouts or need to be added to managers, you create interstitial ads in code exclusively, not with an XML definition as shown for banner ads above. An example of creating and displaying an interstitial ad in Java is shown below:

|  |
| --- |
| // Construct view using site and zone registered with mocean mobile ui  int myInterstitialZone = 33333; // *sample, see description above*  boolean isInterstitial = true;  MASTAdView interstitialView =  new MASTAdView(this, isInterstitial);  // Set the zone  interstitialView.setZone(myInterstitialZone);  // Update view to fetch first ad  interstitialView.update();  // Show ad; will pop up in front of app screens  interstitialView.showInterstitial(); |

Figure 4.1 – Interstitial Ad Display

And visually, here is what the user might actually see when the ad is displayed:



Figure 4.2 - Interstitial Ad

Note that the basics of creating an interstitial ad view are the same as for a banner ad. There are two significant differences:

* The interstitial is presented as a full screen dialog.
* The *showInterstitial*() method is used to display the ad instead of adding the view to a manager along with other content on screen as seen in the banner example above.

Note that this examples show a small set of the properties that developers can use to customize the appearance and behavior of the ad view. The full set is described in the SDK documentation.

## Native Ads Integration

You can use the same SDK/framework to integrate native ads in your app.

You need to initialize **MASTNativeAd** to use Native ads. A sample code is given below.

Add the following in your activity file:

|  |
| --- |
| private MASTNativeAd ad = null;  ad = new MASTNativeAd(this);  // Register your activity to receive notification events  ad.setRequestListener(this);  // ZONE\_ID received from Mocean Portal  ad.setZone(ZONE\_ID);  // AD\_URL received from Mocean Portal  ad.setAdNetworkURL(AD\_URL);  // Request all the assets.  ad.setNativeContent("1,2,3,4,5,6");  ad.setDescriptionLength(150);  ad.setTitleLength(30);  ad.setIconImageSize(NativeAdSize.ICON\_IMAGE\_300X300);  ad.setMainImageSize(NativeAdSize.MAIN\_IMAGE\_1200X627);  ad.setLocationDetectionEnabled(true);  // If requesting ads in test mode, use this.  ad.setTest(true);  /\* Set to false if you want Mocean SDK to handle instantiation of \* third-party SDK and set to true if you want to have control of instantiating third-party SDK.  If false then ***NativeRequestListener.onNativeAdReceived(MASTNativeAd ad)***is called on successfully receiving ad contents.  If set to true, then ***NativeRequestListener.onReceivedThirdPartyRequest(MASTNativeAd ad,***  ***Map<String, String> properties, Map<String, String> parameters)*** is called.  The default value is ***false.***  \*/  ad.overrideAdapterLoading(false);  // Request native ad  ad.update(); |

**Deallocating MASTNativeAd**

|  |
| --- |
| @Override  protected void onDestroy() {  super.onDestroy();  ad.destroy();  } |

**Receiving Notification from MASTNativeAd**

|  |
| --- |
| @Override  public void onNativeAdReceived(final MASTNativeAd ad) {  if (ad != null) {  // Use this method to tell Mocean SDK to handle clicks and send the impression trackers as well as click trackers  ad.trackViewForInteractions(mLayout);  runOnUiThread(new Runnable() {  @Override  public void run() {  Image iconImage = ad.getIconImage();  Image mainImage = ad.getMainImage();  if (iconImage != null) {  ad.loadImage(imgLogo, iconImage.getUrl());  }  if (mainImage != null) {  ad.loadImage(imgMain, mainImage.getUrl());  }  txtTitle.setText(ad.getTitle());  txtDescription.setText(ad.getText());  txtCtaText.setText(ad.getCallToAction());  }  });  }  }  @Override  public void onNativeAdFailed(MASTNativeAd ad, Exception ex) {  ex.printStackTrace();  }  @Override  public void onReceivedThirdPartyRequest(MASTNativeAd ad,  Map<String, String> properties, Map<String, String> parameters) {  }  @Override  public void onNativeAdClicked(MASTNativeAd ad) {  } |

**Using Mediation**

If you want to use the mediation framework, just follow the integration documents of **Facebook Audience Network** and **MoPub**, and set ***NativeAd.overrideAdapterLoading(false).*** Using this, the Mocean SDK will automatically handle instantiation of third-party SDKs and hence fetching the required details from third-party SDKs.

Facebook Audience Network (see the native section for more details): <https://developers.facebook.com/docs/audience-network/getting-started>

MoPub (see the native section for more details): <https://dev.twitter.com/mopub/android/getting-started>

**Downloading Adapters**

The adapters for Facebook Audience Network and MoPub will be downloaded along with the Mocean SDK.

Please add the above adapters only if you are using Facebook Audience Network and MoPub as mediation SDKs.

**Facebook Audience Network-specific Change**

|  |
| --- |
| /\*Add your device id for Facebook Audience network to get test ads.  \* This will be printed in the logs once you launch the application for  \* the first time.\*/  ad.addTestDeviceIdForNetwork(MediationNetwork.FACEBOOK\_AUDIENCE\_NETWORK, DEVICE\_ID); //Please change DEVICE\_ID with actual ID |

## Handling Rotation Changes

By default, when certain configuration changes (such as screen orientation and/or physical keyboard availability) occur, Android restarts the current activity (by invoking the *onDestroy*() and then *onCreate*() methods.) This will typically cause a full reload of all resources, and a refresh of screen content, including the ad view.

Sometimes this complete restart is not desired, and application developers override the default behavior so that only those resources which actually need to be reloaded do so. Consult the Android developer documentation for more information about the general approach to handling configuration changes; this specific topic is addressed here:

<http://developer.android.com/guide/topics/resources/runtime-changes.html>.

Depending on your UI layout, it is common for the ad view to be one of the elements which SHOULD be reloaded after a screen orientation change (and/or the related physical keyboard change as well.) For example, using our sample layout from section 3 above, the banner ad is mean to use the full width of the screen. After the screen rotates, it is desirable to request a new ad that will better fit into the available space can be displayed (for example, if the ad started off in 480x800 vertical orientation, and then rotates, the width is now 800 and the server might have an ad available that is better suited for this display size.) An example of an app showing different ads in portrait and landscape view is shown in the figures below.

The following value is recommended for the activity tag configChanges attribute in the manifest for activities that will contain MASTAdView instances:  
keyboardHidden|orientation|screenSize

|  |
| --- |
| Screenshot_2012-06-14-14-21-33.pngScreenshot_2012-06-14-14-34-30.png  Figure 5.1 - Portrait Ad, Full Width, 100 tall  Figure 5.2 - Landscape Ad, Full Width, 100 tall |

If you have followed the Android developer documentation referenced above, you will have added a *configChanges* property to the activity definition in your project manifest, and implemented the *onConfigurationChanged*() method in your java activity source code.

See the Samples/OrientationSamples project for the means to detect the orientation ad and automatically update the MASTAdView instance after it’s layout is complete so the ad request size matches the new orientation layout size.

## Detecting Ad Load Failures

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

The SDK includes an optional *MASTAdViewDelegate.RequestListener* interface which applications can implement to receive notifications when download related ad events occur. This interface includes four methods as follows:

* *onReceivedAd ()* which is invoked when the ad content is inserted into a view for display.
* *onFailedToReceiveAd ()* which is invoked if downloading ad content fails for any reason.

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

|  |
| --- |
| adView.setRequestListener(**new** MASTAdViewDelegate.RequestListener()  {  @Override  **public** **void** onFailedToReceiveAd(**final** MASTAdView adView, Exception ex)  {  runOnUiThread(**new** Runnable()  {  @Override  **public** **void** run()  {  adView.setVisibility(View.*GONE*);  }  });  }  // … other listener methods here …  }); |

Figure 6.1 –Detecting “no ad available”

## Where To Go Next

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

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