Getting Started with the Mocean Android SDK

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. This introduction to the SDK is written for Android developers new to the advertising platform, and gives developers a quick introduction to the first ad integration project.

Contents

[1. Prerequisites 2](#_Toc328415146)

[2. User Interface / Layout (Design) 2](#_Toc328415147)

[3. Creating a Banner Ad View 3](#_Toc328415148)

[3.1 Layout Based Ad View Creation 3](#_Toc328415149)

[3.2 Code Based Ad View Creation 5](#_Toc328415150)

[3.2.1 Displaying the Ad View 5](#_Toc328415151)

[4. Getting Initial Ad View Content 6](#_Toc328415152)

[5. Creating an Interstitial Ad View 6](#_Toc328415153)

[6. Handling Rotation Changes 8](#_Toc328415154)

[7. Customizing the Viewport 10](#_Toc328415155)

[8. Detecting Ad Load Failures 10](#_Toc328415156)

[9. Where To Go Next 12](#_Toc328415157)

## Prerequisites

Install the Android ad SDK available from

<http://code.google.com/p/mocean-sdk-android/downloads/list>

as described in documentation. Once complete, you should have an open Android project:

* Linked to the AdserverView SDK library project via the build path
* With required project manifest updated for permissions and minimum SDK version attributes
* At least one activity & layout containing the UI elements of your application

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>.

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 our Google Code project site at <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>.

## 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 the standard *WebView* control, the full set of *WebView* properties are also available for use by the developer as needed.

### 3.1 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.MASTAdView.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"*  site=*"11111"*  zone=*"22222"*  updateTime=*"60"*  android:visibility=*"visible"* />  <! -- Any remaining layout -- >  <! -- … -- > |

Figure 3.1 - 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.*
* **Publisher Site**: this is setup through the Mocean Mobile UI when you setup ad feeds to display content in your application. Typically a “site” will be used to identify one of your applications and distinguish it from another of your applications. The site is required in order to request an ad.
* **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 created through the Mocean 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.
* **Ad update interval**. This configured the time period (in seconds) after which the ad view will retrieve a new ad from the back-end.

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.

### 3.2 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 using site and zone registered with mocean mobile ui  int myAdSite = 11111;  int myAdZone = 22222;  MASTAdView adserverView = new MASTAdView(this, myAdSite, myAdZone);  // Set update interval  adserverView.setUpdateTime(60);  // Set layout: full width of screen, 100 pixels tall  adserverView.setLayoutParams(  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(adserverView); |

Figure 3.2 - Code Ad View

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

* The site and zone are configured. Note that these are set when creating the MASTAdView object.
* 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.

### 3.2.1 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, you must first obtain a java reference to the ad view object created from your layout definition. Building upon the XML layout from earlier, the first line of code below does the job. Referring back to Figure 3, you can see that the ID of the ad view component was specified as “mainTopAdView”. A reference to the component created by “inflating” this XML layout can be obtained using the standard Android *findViewById()* method as shown here.

Then, regardless of how you setup the ad view object, you invoke the *update*() method on it, as shown below:

|  |
| --- |
| // Get reference to ad view object from XML layout, if needed.  // NOTE: If you used the code option above, you already have a MASTAdView  // objet and this line is not needed.  MASTAdView adserverView = (MASTAdView)findViewById(R.id.mainTopAdView);  // Update view to fetch first ad  adserverView.update(); |

Figure 4.1 - 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 *show*() 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 myAdSite = 11111;  int myInterstitialZone = 33333;  MASTAdView interstitialView =  new MASTAdView(this, myAdSite, myInterstitialZone);  // OPTIONAL: Automatically close ad after 10 seconds  interstitialView.setAutoCloseInterstitialTime(10);  // OPTIONAL: Pause 2 seconds before showing close button  interstitialView.setShowCloseButtonTime(2);  // Set layout: full size of screen  interstitialView.setLayoutParams(  new ViewGroup.LayoutParams(ViewGroup.LayoutParams.FILL\_PARENT,  ViewGroup.LayoutParams.FILL\_PARENT));  interstitialView.requestLayout();  // Update view to fetch first ad  interstitialView.update();  // Show ad; will pop up in front of app screens  interstitialView.show(); |

Figure 5.1 – Interstitial Ad Display

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



Figure 5.2 - 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 layout parameters are set so this ad view will fill the screen.
* The *show*() 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.

Two optional features are used in this example to customize the interstitial behavior:

* Auto-close is configured so that after the specified number of seconds, the interstitial ad will close without the user needing to take any action.
* A two second delay is introduced before the (mandatory) close button is shown; this allows a brief period of time for the ad to be shown without any possibility of obstruction from that close button.

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.

## 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 a 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.

|  |
| --- |
| Screenshot_2012-06-14-14-34-30.pngScreenshot_2012-06-14-14-21-33.png  Figure 6.1 - Portrait Ad, Full Width, 100 tall  Figure 6.2 - 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. You can augment this method with code such as the following to fetch a new ad. Note that the maximum allowed ad width is reset using the current screen dimension information so that the best ad possible can be served for display in this placement.

|  |
| --- |
| @Override  public void onConfigurationChanged(Configuration newConfig)  {  // Pass through to standard android handler first  super.onConfigurationChanged(newConfig);  // ... perform any other manipulation needed by application ...  // Get screen dimensions  WindowManager windowManager =  (WindowManager)getSystemService(Context.*WINDOW\_SERVICE*);  DisplayMetrics metrics = **new** DisplayMetrics();  windowManager.getDefaultDisplay().getMetrics(metrics);  // Update notion of desired ad width; uses ad reference created above  **int** width = metrics.widthPixels;  adserverView.setMaxSizeX(width);  // Call update() to get a new ad  adserverView.update();  } |

Figure 6.3 – Configuration Change Handler

## Customizing the Viewport

As noted in section 3 above, the ad view is an extension of the standard *WebView* control. For the purposes of ad display, and supporting various device screen sizes and resolutions, there are a number of settings it is important to be aware of that affect the appearance and behavior of content shown in this view (such as width, height, scale factor and density.) These can be customized with a *viewport* property and an HTML stylesheet applied to the ad view.

The default viewport used when the SDK displays ad content is as follows:

<meta name="viewport" content="target-densitydpi=device-dpi"/>

And the style sheet applied to the body of the ad content is:

<style>body{margin: 0px; padding: 0px; display:-webkit-box;-webkit-box-orient:horizontal;-webkit-box-pack:center;-webkit-box-align:center;}</style>

This value can be customized using the *setInjectionHeaderCode*() method of the SDK. For example, in a recent DRAFT MRAID 2 specification, the IAB is requiring future compliant SDK’s to use the viewport string:

<meta name='viewport' content='user-scalable=no initial-scale=1.0'/>

The following example shows how this can be set NOW if you are working with ad creators who are already moving towards the evolving MRAID standard:

|  |
| --- |
| // Customize the viewport injection string for ad views created through SDK.  // This example uses setting required in the latest IAB MRAID 2 draft spec.  String viewport =  “<meta name='viewport' content='user-scalable=no initial-scale=1.0'/>”  adserverView.setInjectionHeaderCode(viewport + MASTAdView.defaultBodyStyle); |

Figure 7.1 – Customizing the Ad Viewport

For more discussion and information about the full capabilities of the viewport property, consult the Android developer documentation on this topic at:

<http://developer.android.com/guide/webapps/targeting.html>.

## 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 *MASTOnAdDownload* interface which applications can implement to receive notifications when download related ad events occur. This interface includes three methods as follows:

* *begin*() which is invoked when the request is sent to the mobile ad server.
* *end()* which is invoked after the ad content has been downloaded successfully.
* *error()* which is invoked if downloading ad content fails for any reason.

The *error*() 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:

|  |
| --- |
| adserverView.setOnAdDownload(**new** MASTOnAdDownload()  {  @Override  **public** **void** error(MASTAdView sender, String error)  {  // **Check for no ad error response**  if (error.compareTo(Constants.STR\_EMPTY\_SERVER\_RESPONS) == 0)  {  // “No ad error”, adjust size, change zones,  // remove ad view here, etc.  sender.setVisibility(View.GONE);  }  }    @Override  **public** **void** end(MASTAdView sender) {  // **Any on-success processing here**  }    @Override  **public** **void** begin(MASTAdView sender) {  // **Any pre-request logic here**  }  }); |

Figure 8.1 –Detecting “no ad available”

As shown here, when no ad was found the message passed to the error() will always be the constant string STR\_EMPTY\_SERVER\_RESPONSE (was STR\_EMPTY\_SERVER\_RESPONS in previous SDK versions) defined in the Constants class. Once this condition is detected, the application can take appropriate action such as using different criteria when requesting an ad (for example, if a minimum height and/or width were used before, remove those), use a different zone on which ads might be found, or simply remove the ad view altogether.

## 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>.