PubMatic

PubMatic Android SDK

Developer Guide Document

SDK Version: 5.1.1

January 20, 2017

© 2017 PubMatic Inc. All rights reserved. Copyright herein is expressly protected at common law, statute, and under various International and Multi-National Treatises (including, but by no means limited to, the Berne Convention for the Protection of Literary and Artistic Works).

The following documentation, the content therein and/or the presentation of its information is proprietary to and embodies the confidential processes, designs, and technologies of PubMatic Inc. All copyrights, trademarks, trade names, patents, industrial designs, and other intellectual property rights contained herein are, unless otherwise specified, the exclusive property of PubMatic Inc. The ideas, concepts, and/or their application, embodied within this documentation remain and constitute items of intellectual property which nevertheless belong to PubMatic Inc.

The information (including, but by no means limited to, data, drawings, specification, documentation, software listings, source and/or object code) shall not be disclosed, manipulated, and/or disseminated in any manner inconsistent with the nature and/or conditions under which this documentation has been issued.

The information contained herein is believed to be accurate and reliable. PubMatic Inc. accepts no responsibility for its use in any way whatsoever. PubMatic Inc. shall not be liable for any expenses, damages, and/or related costs, which may result from the use of any information, contained hereafter.

PubMatic Inc. reserves the right to make any modification to this manual or the information contained herein at any time without notice.

CORPORATE HEADQUARTERS

PubMatic, Inc.

305 Main Street, Suite 100

Redwood City, CA 94063

USA

www.pubmatic.com

Contents

Introduction

The PubMatic Android SDK makes it easy to incorporate ads into their Android applications.

The PubMatic Android SDK is completely modular, which provides the following benefits:

- You can integrate with either single or multiple ad formats at same time.
- Considerably reduces SDK size
  Complete flexibility to choose between PubMatic and Mocean Ad Server

Features

- PubMatic Android SDK supports following ad formats
  - Banner
  - Interstitial
  - Native
- PubMatic Android SDK supports latest Android version 6.0 Marshmallow
- Rich Media MRAID 2.0 compliant Banner ads

Getting Started

Prerequisites

1. Minimum Android SDK version 2.3, API level 9 or later
2. Compile with Android SDK version 4.3, API level 18 or later
3. Android Studio 1.5.0 or later
4. Android SDK tools 24.1.1 or later
5. Android Support Library 23.1.1 or later
6. Gradle 2.8 or later

Integrate PubMatic SDK

Option 1: Integrate via central repository (Only for gradle based project)

PubMatic SDK is published in JitPack central repository.

Users of this SDK will need to add the jitpack.io repository in the build.gradle file of the root project.

```
allprojects {
   repositories {
      jcenter()
      maven {
         url "https://jitpack.io"
      }
   }
}
```

And, Need to add the following code in the build.gradle of the application module:

```
compile 'com.github.PubMatic.pubmatic-sdk-android:common-sdk:5.1.1'
compile 'com.github.PubMatic.pubmatic-sdk-android:banner-sdk:5.1.1'
compile 'com.github.PubMatic.pubmatic-sdk-android:native-sdk:5.1.1'
```

Above snippet is in form of 'com.github.User.Repo:library:releasetag'

Where PubMatic has separate SDK library for banner & native ad. User has an option to include the SDK based on the requirement.

It is mandatory to include **common-sdk** along with banner and/or native sdk.

5.1.1 is the latest released version of PubMatic SDK. User can also choose an option to always get the "tip" of the master branch by using below template:

compile 'com.github.user.Repo:library:-SNAPSHOT'

For example:

Replace 5.1.1 with -SNAPSHOT

```
compile 'com.github.PubMatic.pubmatic-sdk-android:common-sdk:-SNAPSHOT'
compile 'com.github.PubMatic.pubmatic-sdk-android:banner-sdk:-SNAPSHOT'
compile 'com.github.PubMatic.pubmatic-sdk-android:native-sdk:-SNAPSHOT'
```

Option 2: Download the source from github

PubMatic SDK is available as Open Source project for download from GitHub.

You can download or clone PubMatic SDK source from following path:

https://github.com/PubMatic/pubmatic-sdk-android

Integration with Gradle-based projects

To integrate the PubMatic SDK as source code into your Android Studio project:

1. Copy the required modules from PubMatic-Android-SDK project as modules into your application project. Note that **common-sdk** module is mandatory for adding any ad format SDK.
   For example: To integrate Banner ads into your application project, copy the **common-sdk** and the **banner-sdk** modules from PubMatic-Android-SDK to your project folder.
2. Modify your project's **`settings.gradle`** file.
3. Add PubMatic SDK's `banner-sdk`, `common-sdk` and modules:
   `include ':common-sdk', ':banner-sdk', ':sdk-sample-app', ':native-sdk'`

4. Modify your project's **`build.gradle`** file and add the `common-sdk` and other required ad format modules as dependencies.
   Please refer sdk-sample-app for sample implementation.

Example: Adding Dependencies

```
dependencies

{
    // Add PubMatic common-sdk as Module dependency
    compile project(':common-sdk')
    // Add PubMatic banner-sdk as Module dependency
    compile project(':banner-sdk')
    // Add PubMatic native-sdk as Module dependency
    compile project(':native-sdk')
}
```

Integration with Eclipse-based Projects

If you are using the Eclipse IDE for your Android projects, you can include the PubMatic SDK as JAR library files.

Note: when you add SDK JAR files, make sure that you add **`common-sdk.jar`**.

Following is a list of SDK JAR files available with PubMatic SDK:

1. `common-sdk.jar` (Mandatory - Base SDK library)
2. `banner-sdk.jar` (For integrating Banner ads)
3. `native-sdk.jar` (For integrating Native ads)

While integrating SDK, you can add any one or multiple of the above SDK JAR files, per your requirement.

Generating JAR files to use with Eclipse projects

You can also generate JAR files by running a custom `gradle` task. You must have `gradle` installed to run a `gradle` task.

Run the following gradle task from PubMatic SDK source project folder:

```
# PubMatic-Android-SDK $ gradle exportJar
```

Running this task successfully generates the JAR files in **SDK-JAR/** folder: for example: PubMatic-Android-SDK/SDK-JAR/common-sdk.jar

Add SDK JAR files to Eclipse project

To integrate SDK with your Eclipse application, copy above generated JAR files to your projects `libs/` folder. Make sure that these libraries are present in your projects classpath.

Adding Permissions to the Manifest file

Declare the following security permissions in your manifest file (`AndroidManifest.xml`) to enable the SDK.

Table 1 Basic Permissions for the Manifest File

Permission	Description and Manifest file entry
Internet	Access the Internet. Required for ad-content download. `<uses-permission android:name="android.permission.INTERNET" />`
Location	Use the phone network to obtain location information. This permission is needed as location detection in enabled by default in SDK. `<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />`
Network State	Access the network state. Required for ad request parameter setting, and MRAID support. `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`

Depending on the ad content you display in your app, the following might also be needed.

Table 2 Advanced Permissions for the Manifest File

Permission	Description and Manifest file Entry
Fine Location	Use GPS to obtain location information. `<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>`
Phone State	Read state of phone data connection. Required for ad request parameter setting. `<uses-permission android:name="android.permission.READ_PHONE_STATE"/>`

Permission	Description and Manifest file Entry
Read Calendar	Read calendar events. Needed if the MRAID ad makes use of calendar features. <uses-permission android:name="android.permission.READ_CALENDAR"/>
Write Calendar	Write calendar events. Needed if the MRAID ad makes use of calendar features. <uses-permission android:name="android.permission.WRITE_CALENDAR"/>
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"/>
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"/>
External Storage	Access the SD card storage area. Required for photo, and file access to support MRAID features. <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

Integrating Banner Ads

PubMatic SDK supports Banner ad serving from PubMatic as well as Mocean Ad Server.

To integrate a banner ad:

1. Decide where to put an ad in your UI.
2. Create the ad view.

Use one of the following ways to add the `AdView` components to your activity:

- Dynamically by creating the view in code and adding it to a layout
- Using an XML layout definition.

Example of creating a banner ad with each approach follows. Note that these examples show a small set of the ad view properties that you can use to customize the appearance and behavior of the ad view. The full set of View properties like margins, padding, orientation etc are also available for use as needed. You can refer attributes of Android View class for more details.

Note: Before integrating any ad formats to your application, please make sure that you have added all necessary permissions in your applications manifest file.

Layout Based Ad View Creation

1. Open the layout XML file for your activity.
2. Add the `com.pubmatic.sdk.banner.PMBannerAdView` component into the XML view.

Example: Mocean Ad Request

If you are using the Mocean ad server:

1. Use **`channel="mocean"`**.
2. Add `placement` details using the **`zone`** parameter.

```
<com.pubmatic.sdk.banner.PMBannerAdView
        android:layout_width="match_parent"
        android:layout_height="50dp"
        channel="mocean"
        zone="<zoneId>" />
```

Example: PubMatic Ad Request

When you use the PubMatic ad server:

1. Specify **`channel=pubmatic`**.
2. Add `AdTag` details using the following parameters: **`pubId`**, **`siteId`**, **`adId`**, **`adWidth`**, and **`adHeight`**.

You can set and update these parameters from the code.

```
<com.pubmatic.sdk.banner.PMBannerAdView
        android:layout_width="match_parent"
        android:layout_height="50dp"
        channel="pubmatic"
        pubId=<pubID>
        siteId="<siteId>"
        adId="<adId>"
        adWidth="320"
        adHeight="50" />
```

Creating Code Based AdView

1. Import PubMatic's object definitions into your Java class.

Example: Code-based AdView

```
// Banner Ad View Class
import com.pubmatic.sdk.banner.PMBannerAdView;

// For making request to Mocean Ad server
import com.pubmatic.sdk.banner.mocean.MoceanBannerAdRequest;

// For making request to PubMatic Ad server
import com.pubmatic.sdk.banner.pubmatic.PubMaticBannerAdRequest;
```

2. Use code such as **Example: Mocean Ad Request** to create and setup a `PMBannerAdView` component

Example: Mocean Ad Request

```
PMBannerAdView banner = new PMBannerAdView(context);
RelativeLayout layout = (RelativeLayout)
findViewById(R.id.parent);
// Set layout height & width in pixels for banner ad view as per
requirement
```

```
LayoutParams params = new LayoutParams(LayoutParams.MATCH_PARENT,
100);
params.setLayoutDirection(RelativeLayout.ALIGN_PARENT_TOP);
layout.addView(banner, params);

MoceanBannerAdRequest adRequest = MoceanBannerAdRequest
  .createMoceanBannerAdRequest(this, "<zoneId>");
// Make the ad request to Server
banner.execute(adRequest);
```

Example: PubMatic Ad Request

```
PMBannerAdView banner = new PMBannerAdView(context);


RelativeLayout layout = (RelativeLayout)
findViewById(R.id.parent);
// Set layout height & width for banner ad view in pixels as per
requirement
LayoutParams params = new LayoutParams(LayoutParams.MATCH_PARENT,
100);
params.setLayoutDirection(RelativeLayout.ALIGN_PARENT_TOP);
layout.addView(banner, params);



PubMaticBannerAdRequest adRequest = PubMaticBannerAdRequest
  .createPubMaticBannerAdRequest(PubRuntimeBannerActivity.this,
      "<pubId>",
      "<siteId>",
      "<adId>");
adRequest.setAdSize(PUBAdSize.PUBBANNER_SIZE_320x50);

// Make the ad request to Server
banner.execute(adRequest);
```

Set update interval time

PubMatic Banner SDK provides an optional feature to auto refresh the banner ad after specified interval. Publisher can set the refresh interval from java code using the PMBannerAdView.setUpdateInterval(<value in sec>) before calling the execute() method. And by using the updateInterval field from the layout xml file. Expected value should be in range of 12 to 120.

Ads refresh behavior on setting update interval value:

Value (X) in seconds	SDK behavior
X<=0	Ad will not refresh
X > 0 & X <= 12	Ad will get refreshed after every 12 seconds.
X > 12 & X <=120	Ad will get refreshed after every X seconds
X > 120	Ad will get refreshed after every 120 seconds

Update interval can be set from either Java code or from XML layout file as shown below.

Set update interval in Java code

```
// Set update interval of say 15 seconds
banner.setUpdateInterval(15);
// Make the ad request to Server
banner.execute(adRequest);
```

Set update interval in XML layout file

```
<com.pubmatic.sdk.banner.PMBannerAdView
        android:layout_width="match_parent"
        android:layout_height="50dp"
        channel="mocean"
        zone="<zoneId>"
        updateInterval=”15” />
```

Listening to Banner Ad Request callbacks

PubMatic Banner SDK provides **`RequestListener`** delegate through which you can receive Ad received / Ad failed callbacks.

Note: Make sure to set `RequestListener` on the banner `AdView` instance before calling `execute()` method.

Example: implementation of RequestListener

```
// First initialize banner ad view...
// Set Ad request listener
banner.setRequestListener(new
PMBannerAdView.BannerAdViewDelegate.RequestListener() {
      @Override
      public void onFailedToReceiveAd(PMBannerAdView adView,
      Exception ex {
            // Ad failed callback
            // You can hide the ad view in this case
            runOnUiThread(new Runnable() {
                  @Override
                  public void run() {
                        banner.setVisibility(View.GONE);
                  }
            });
      }

      @Override
      public void onReceivedAd(PMBannerAdView adView) {
            // Ad received successfully
      }

      @Override
      public void onReceivedThirdPartyRequest(PMBannerAdView
      adView,
      Map<String, String> properties,
      Map<String, String> parameters) {
            // Third party ad response received
      }
});
```

```
// Make the ad request to Server
banner.execute(adRequest);
```

Handling Rotation Changes for Banner Ads

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 typically causes a full reload of all resources, and a refresh of screen content, including the ad view.

Sometimes this complete restart is not desired; application developers override the default behavior, so that only those resources that 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, ifrequently the ad view is one of the elements that should be reloaded after a screen orientation change or the related physical keyboard change.

PubMatic recommends the following value \for the activity tag configChanges attribute in the manifest for activities that contain **`PMBannerAdView`** instances:

`android:configChanges="keyboardHidden|orientation|screenSize"`

Integrating Interstitial Ads

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, and others. Interstitial ads always include a **Close** button. Optionally, you can configure them 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.

Create interstitial ads in code exclusively. Do not use an XML definition because they do not appear in layouts or need to be added to managers,

Note: Make sure to call either `showInterstitial()` or `showInterstitialWithDuration()` method after calling the `execute()` method on `AdView`. The interstitial ad is not shown until one of these methods are called.

For faster display of interstitial ads, call the **`showInterstitial()`** or **`showInterstitialWithDuration()`** method in **`onReceivedAd()`** callback. This ensures that a full-screen interstitial ad view is shown only when ad is successfully received from the server.

Example: Mocean Interstitial Ad Request

```
// In onCreate method of your Activity

PMInterstitialAdView interstitialAdView = new
PMInterstitialAdView(context);
MoceanBannerAdRequest adRequest = MoceanBannerAdRequest
.createMoceanBannerAdRequest(this, "88269");

// Make Ad request to server
interstitialAdView.execute(adRequest);

// Set Ad request listener
interstitialAdView.setRequestListener(new RequestListener() {

     @Override
     public void onReceivedAd(PMBannerAdView interstitialAdView)
```

```
      {
            // Call showInterstitial method in onReceivedAd
            callback
            interstitialAdView.showInterstitial();
            // Your code...
      }
      @Override
      public void onReceivedThirdPartyRequest(PMBannerAdView
      interstitialAdView,
      Map<String, String> properties, Map<String, String>
      parameters) {
            // Your code...
      }

      @Override
      public void onFailedToReceiveAd(PMBannerAdView
      interstitialAdView, Exception e) {
            // Your code...
      }
});
```

Native Ads Integration

Native advertising is a form of online advertising where the ad is rendered with the same look and feel like the page or application in which it is displayed.

To show Native ads:

1. Initialize **`PMNativeAd`**.
2. Use **`NativeRequestListener`** to get the callback and assets required for rendering the Native ad.

Example: Integration of a Native Ad with your Application

```
// First create some assets to add in the request
 List<PMAssetRequest> assets = new ArrayList<>();
PMTitleAssetRequest titleAsset = new PMTitleAssetRequest();
// Note: Unique assetId is mandatory for each asset
titleAsset.setAssetId(1001);
titleAsset.setLength(50);
titleAsset.setRequired(true); // Optional (Default: false)
assets.add(titleAsset);

PMImageAssetRequest imageAssetIcon = new PMImageAssetRequest();
imageAssetIcon.setAssetId(1003);
imageAssetIcon.setImageType(PMImageAssetTypes.icon);
assets.add(imageAssetIcon);

PMImageAssetRequest imageAssetMainImage = new
PMImageAssetRequest();
imageAssetMainImage.setAssetId(1004);
imageAssetMainImage.setImageType(PMImageAssetTypes.main);
assets.add(imageAssetMainImage);


PMDataAssetRequest dataAssetDesc = new PMDataAssetRequest();
dataAssetDesc.setAssetId(1002);
dataAssetDesc.setDataAssetType(PMDataAssetTypes.desc);
dataAssetDesc.setLength(25);
assets.add(dataAssetDesc);
```

Example: Mocean Ad Request

```
// Initialize PMNativeAd
PMNativeAd ad = new PMNativeAd(context);

// Create a Mocean ad request
MoceanNativeAdRequest adRequest = MoceanNativeAdRequest
.createMoceanNativeAdRequest(context, <zoneId>, assets);

// Set Native ad request listener
ad.setRequestListener(new AdRequestListener());

// Request for ads
ad.execute(adRequest);
```

Example: PubMatic Ad Request

```
// Initialize PMNativeAd
PMNativeAd ad = new PMNativeAd(context);

// Create a PubMatic ad request
PubMaticNativeAdRequest adRequest = PubMaticNativeAdRequest
.createPubMaticNativeAdRequest(context, "<pubId>", "<siteId>",
"<adId>", assets);

// Set Native ad request listener
ad.setRequestListener(new AdRequestListener());

// Request for ads
ad.execute(adRequest);
```

Native Response Handling

Example: implementation of AdRequestListener and Native Response Handling

```
private class AdRequestListener implements
PMNativeAd.NativeRequestListener {
```

```
@Override
public void onNativeAdFailed(PMNativeAd ad, Exception ex) {
      ex.printStackTrace();
}

@Override
public void onNativeAdReceived(final PMNativeAd ad) {
if (ad != null) {
      runOnUiThread(new Runnable() {

      @Override
      public void run() {
            List<PMAssetResponse> nativeAssets =
            ad.getNativeAssets();
            for (PMAssetResponse asset : nativeAssets) {
            try {
                  /*
                  * As per openRTB standard, assetId in
                  response
                  * must match that of in request.
                  */
                  switch (asset.getAssetId()) {
                  case 1001:
                        txtTitle.setText(((PMTitleAssetResponse)
                        asset)
                        .getTitleText());
                        break;
                  case 1003:
                        PMNativeAd.Image iconImage =
                        ((PMImageAssetResponse) asset)
                        .getImage();
                        if (iconImage != null) {
                        imgLogo.setImageBitmap(null);
                        ad.loadImage(imgLogo,
                        iconImage.getUrl());
                        }
                        break;
                  case 1004:
                        PMNativeAd.Image mainImage =
                        ((PMImageAssetResponse) asset)
                        .getImage();
                        if (mainImage != null) {
                        imgMain.setImageBitmap(null);
                        ad.loadImage(imgMain,
                        mainImage.getUrl());
                        }
                        break;
                  case 1002:
                        txtDescription
                        .setText(((PMDataAssetResponse) asset)
                        .getValue());
```

```
                              break;
                        default:
                              break;
                        }
                        } catch (Exception ex) {
                              // Skip asset if there is any error
                        }
                     }
                   }
                });

                /*
                Important: Must call this method when response
                rendering is complete. This method sets click
                listener on the ad container layout. This is
                required for firing click tracker when ad is
                clicked by the user.
                */
                ad.trackViewForInteractions(adContainerLayout);
          }
     }

     @Override
     public void onReceivedThirdPartyRequest(PMNativeAd
     mastNativeAd,
     Map<String, String> properties, Map<String, String>
     parameters) {
          // NOOP
     }

     @Override
     public void onNativeAdClicked(PMNativeAd ad) {
          // Ad clicked

     }
}
```

Track View for Interactions

You must call `trackViewForInteractions()` method when response rendering is complete.

Pass the instance of container layout (`ViewGroup`) in which native ads are rendered.

This method sets click listener on the ad container layout. This is required for firing a click tracker when an ad is clicked by the user.

Example: Track view for interaction

```
@Override
public void onNativeAdReceived(final PMNativeAd ad) {
if (ad != null) {

            /*
            Code to render native assets as shown in code
            snippet above...
            */

            /*
            Important: Must call this method when response
            rendering is complete. This method sets click
            listener on the ad container layout. This is
            required for firing click tracker when ad is clicked
            by the user.
            */
            ad.trackViewForInteractions(adContainerLayout);
      }
}
```

Deallocating Native Ads

PubMatic highly recommends deallocating the **`PMNativeAd`** instance when your activity is about to get destroyed.

Example: Deallocating Native Ad When Activity Ends

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

Passing Extra Targeting Information

You can pass additional targeting information in the ad request for better monetization.

Best Practice: Passing extra targeting information increases leads to demand partners usually bidding higher for such impressions. While it is not mandatory to pass extra targeting information in ad requests, PubMatic highly recommends passing this information to increase monetization

Pass User Information

To pass user information, set demographic parameters on `AdRequest` instances.

Example: Pass User information on AdRequest Instances

```
// Create instance of PMUserInfo and add user information to it
PMUserInfo userInfo = new PMUserInfo();
     userInfo.setAge("25");
     userInfo.setOver18(PMUserInfo.OVER_18.ALLOW_ALL);
     userInfo.setAreaCode("12345");
     userInfo.setCity("Pune");
     userInfo.setState("Maharashtra");
     userInfo.setCountry("India");
     userInfo.setZip("112233");
     userInfo.setDMA("1234");
     userInfo.setGender("M");
     userInfo.setEthnicity("asian");
     userInfo.setIncome("123456");
     userInfo.setYearOfBirth("1991");
// Set PMUserInfo to AdRequest
adRequest.setUserInformation(userInfo);
```

Pass Application Information

To pass application information, set suitable customer parameters on `AdRequest` instances.

Example: Set Parameters in Pass Application Information

```
// Send App details
adRequest.setStoreURL("http://play.google.com?id=com.test.app");
adRequest.setAppCategory("Games");
adRequest.setAppName("TestGameApp");
adRequest.setIABCategory("IAB-CAT001");
adRequest.setCoppa(false);
```

Detecting Location

Automatic location detection is enabled by default in PubMatic SDK. You can disable automatic location detection using **`setLocationDetectionEnabled(false)`** method as shown in the code snippet . You can also manually pass user location using methods as shown in code snippet.

- PubMatic recommends that you keep automatic location detection enabled in SDK for better monetization.
- For retrieving location, you must add necessary access-location permissions in your application's manifest file. For more details, refer to **Adding Permissions to the Manifest file**.

Example: Disable Location Detection

```
// Disable location detection
// Location detection is enabled by default
banner.setLocationDetectionEnabled(false);

// Manually set location
adRequest.setLocation(<custom-location>);
```