PubMatic

PubMatic Android SDK

Developer Guide

For Android SDK Version 4.3.3

Dec 9, 2015

CORPORATE HEADQUARTERS

PubMatic, Inc.
901 Marshall Street, Suite 100
Redwood City, CA 94063
USA

www.pubmatic.com

Table of Contents

Overview

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

Setup

What Changed In 4.3.3

- Added support for Android Marshmallow (6.0).
- Updated minimum Android SDK support from API level 8 to API level 9.
- Removed default functionality to disable click on all sub-views of native ad.
- Added optional method to disable click listeners on all sub-views of native ad.

> **Note**: See SDK/MoceanNativeSDK/ReadMe.txt document for latest build release notes.

Upgrading From 4.3.2

- While upgrading, remove older 4.3.2 SDK library project and import new 4.3.3 SDK library project.
- Optionally, if you need to disable click listeners on all sub-views of a native ad, you can call method setSubViewsClickable(false).
- Right click on Sample application, Go to Properties > Android > Library section and add MoceanNativeSDK as a dependent library project.

System Requirements

- Android SDK (API level 9, platform version 2.3 or later)
- Eclipse 4.2 (Juno) or later
- 10 Mb free disk space

SDK Contents

- SDK/MoceanNativeSDK -PubMatic SDK library files
- SDK/MoceanNativeSample - Sample usage/test app
- SDK/Documentation - Developer guide document

Installation Guidelines

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:

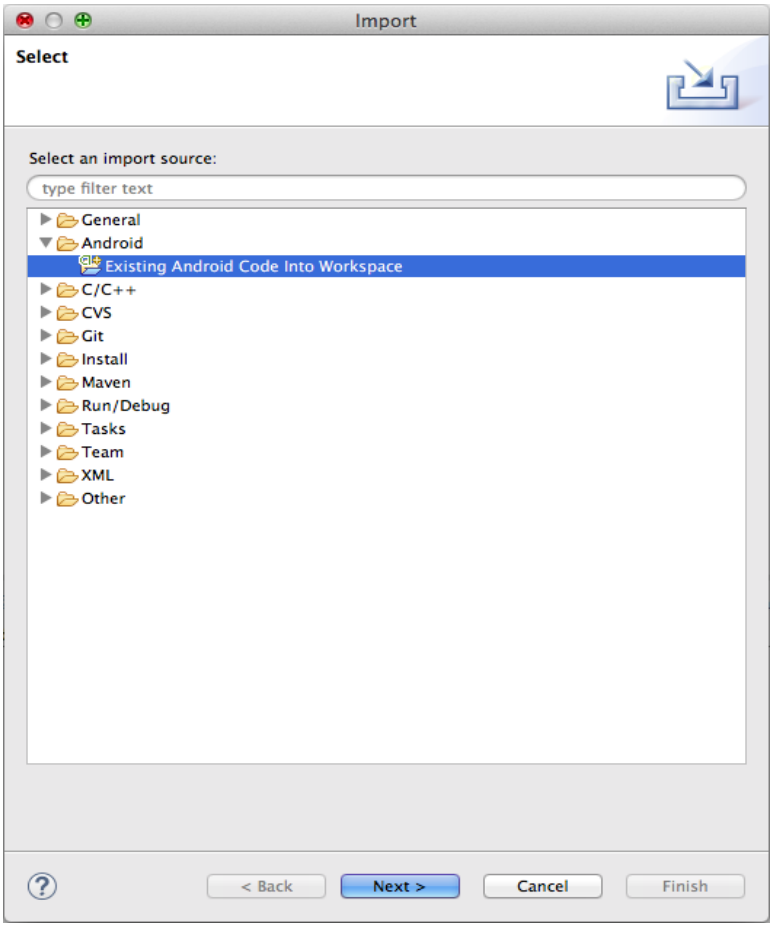
http://developer.android.com/guide/index.html

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

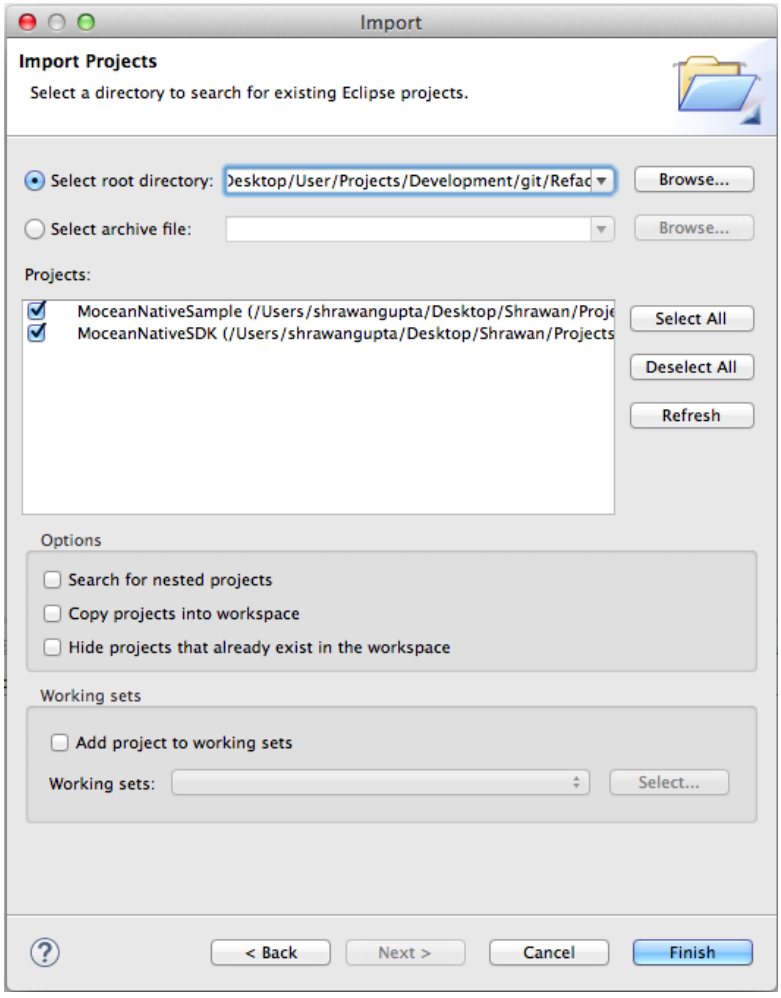
Installing PubMatic Android SDK

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. Unzip 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.
    a. Choose **Import** from the **File** menu, then Existing Android Code Into Workspace item under the Android heading, as show in Figure below:



b. Browse to the location where you unpacked the SDK file and import the com.moceanmobile.mast.MoceanNativeSDK project; you can also optionally import the MoceanNativeSample project if you want to work with the SDK sample application.
See Figure below for an example:



c. To add the SDK as library project, link the imported SDK project in your application project as android library project.

d. IMPORTANT: If using release 18 or later of the Android SDK tools, and if you are using android SDK as jar library, choose the Order and Export tab of the project, and check the box to export the SDK moceannativesdk.jar (if present).

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.

e. Updating the manifest file.

Add "minSdkVersion" parameter in project manifest file - AndroidManifest.xml

```
<uses-sdk android:minSdkVersion="9" />
```

f. Adding permission in manifest file:

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. <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>

Getting Started with Development

Native Ads Integration

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

Add the following in your activity file:

1. **Initialize MASTNativeAd**

```
private MASTNativeAd ad = null;

ad = new MASTNativeAd(this);
// Register your activity to receive notification events
ad.setRequestListener(this);
// ZONE_ID received from PubMatic Portal
ad.setZone(ZONE_ID);
// AD_URL received from PubMatic Portal
ad.setAdNetworkURL(AD_URL);

// If requesting ads in test mode, use this.
// Do not use this in production
ad.setTest(true);
```

2. **Request for native assets**

```
List<AssetRequest> assets = new ArrayList<AssetRequest>();

TitleAssetRequest titleAsset = new TitleAssetRequest();
titleAsset.setAssetId(1); // Unique assetId is mandatory for each asset
titleAsset.setLength(50);
titleAsset.setRequired(true); // Optional (Default: false)
assets.add(titleAsset);

ImageAssetRequest imageAssetIcon = new ImageAssetRequest();
imageAssetIcon.setAssetId(2);
imageAssetIcon.setImageType(ImageAssetTypes.icon);
imageAssetIcon.setWidth(60); // Optional
imageAssetIcon.setHeight(60); // Optional
assets.add(imageAssetIcon);

ImageAssetRequest imageAssetLogo = new ImageAssetRequest();
```

```
imageAssetLogo.setAssetId(3);
imageAssetLogo.setImageType(ImageAssetTypes.logo);
assets.add(imageAssetLogo);

ImageAssetRequest imageAssetMainImage = new ImageAssetRequest();
imageAssetMainImage.setAssetId(4);
imageAssetMainImage.setImageType(ImageAssetTypes.main);
assets.add(imageAssetMainImage);

DataAssetRequest dataAssetDesc = new DataAssetRequest();
dataAssetDesc.setAssetId(5);
dataAssetDesc.setDataAssetType(DataAssetTypes.desc);
dataAssetDesc.setLength(25);
assets.add(dataAssetDesc);

DataAssetRequest dataAssetRating = new DataAssetRequest();
dataAssetRating.setAssetId(6);
dataAssetRating.setDataAssetType(DataAssetTypes.rating);
assets.add(dataAssetRating);

// Request for native assets
ad.addNativeAssetRequestList(assets);
```

3. ## Make the ad request

```
// Request native ad
ad.update();
```

4. ## Receiving Notification from MASTNativeAd

```
@Override
public void onNativeAdReceived(final MASTNativeAd ad) {
if (ad != null) {
        // Code to render native assets on UI.
        // Refer Samples app NativeActivity for sample implementation.
        /* Use this method to tell PubMatic SDK to handle clicks and send the impression
trackers as well as click trackers
         */
        ad.trackViewForInteractions(mLayout);
}
}@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) {
}
```

5. Rendering native ad response assets:

Assets received in ad response can be rendered in onNativeAdReceived callback. Get the list of assets received in response using *ad.getNativeAssets()* method in onNativeAdReceived callback.

> **NOTE**: As per openRTB Native ad specification, the assetId passed in native ad request must match the assetId in response. So you can get and render the assets based on the asset id's that you have passed during the ad request.
> But in case of mediation response the mediation SDK's do not support openRTB protocol. So the assets are not mapped by assetId's. Hence in case of mediation resposne, you can render assets by checking asset type and subtype.

```
public void onNativeAdReceived(final MASTNativeAd ad) {

 if (ad != null) {
  runOnUiThread(new Runnable() {@Override
   public void run() {
    List < AssetResponse > nativeAssets = ad.getNativeAssets();
    for (AssetResponse asset: nativeAssets) {
     try {
       /*
        * As per openRTB standard, assetId in
        * response must match that of in request,
        * Except in case of mediation response.
        */
       switch (asset.getAssetId()) {
        case 1:
         txtTitle.setText(((TitleAssetResponse) asset).getTitleText());
         break;
        case 2:
         // Code to render icon image ...
         break;
       case 3:
        Image logoImage = ((ImageAssetResponse) asset).getImage();
        if (logoImage != null) {
        imgLogo.setImageBitmap(null); // Clear old image
        ad.loadImage(imgLogo, logoImage.getUrl());
        }
        break;
        case 4:
        Image mainImage = ((ImageAssetResponse) asset).getImage();
         if (mainImage != null) {
          imgMain.setImageBitmap(null);
          ad.loadImage(imgMain, mainImage.getUrl());
         }
         break;
        case 5:
         txtDescription.setText(((DataAssetResponse) asset).getValue());
         break;
        case 6:
         String ratingStr = ((DataAssetResponse) asset).getValue();
```

```
                    try {
                     float rating = Float.parseFloat(ratingStr);
                     if (rating > 0f) {
                      ratingBar.setRating(rating);
                      ratingBar.setVisibility(View.VISIBLE);
                     } else {
                      ratingBar.setRating(rating);
                      ratingBar.setVisibility(View.GONE);
                     }
                    } catch (Exception e) {
                     // Invalid rating string
                     Log.e("NativeActivity", "Error parsing 'rating'");
                    }
                    break;

                  default:
                   // NOOP
                   break;
                 }
               } catch (Exception ex) {
                // ERROR in rendering asset. Skipping asset
               }
              }
             }
            });
           }
```

6. Track view for interactions

You must call *trackViewForInteractions()* method when response rendering is complete. Pass the instance of container layout (ViewGroup) in which native ad is rendered. This method sets click listener on the ad container layout. This is required for firing click tracker when ad is clicked by the user.

```
public void onNativeAdReceived(final MASTNativeAd ad) {

 if (ad != null) {
  runOnUiThread(new Runnable() {
   public void run() {
    // Code to render native ad
   }
  });
   // Ad rendering complete

  /*
   * 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(mLayout);
 }
}
```

7. Using Js tracker

As per OpenRTB native ad specifications, jstracker (if present) contains valid javascript wrapped in <script> tags. It should be executed at impression time where it can be supported.

User should execute this javascript whenever rendering of native ad is complete.

```
public void onNativeAdReceived(final MASTNativeAd ad) {

 if (ad != null) {
  runOnUiThread(new Runnable() {

   @Override
   public void run() {
    /* Code to render native ad  */
   }
  });
  if (ad.getJsTracker() != null) {
   // Code to execute Js tracker..
   /*
    * Note: Publisher should execute the javascript tracker
    * whenever possible.
    */
  }
  ad.trackViewForInteractions(mLayout);
 }
}
```

8. (Optional) Disable click listeners on all sub views of Native ad

If you wish to disable click events of clickable views in a Native ad, then you can call setSubViewsClickable(false). This method will remove click listeners on all sub views of a native ad. Only native ad parent container view will be clickable.
By default, all sub views in native ad are clickable.

```
// Disable click on all sub views (Optional)
ad.setSubViewsClikable(false);
```

9. Deallocating MASTNativeAd

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

Using Mediation for Native ad serving

Publishers can use mediation support in Mocean SDK to integrate third party SDK as client side mediation for Native ads. On receiving third party response, application developer can get the required information from Mocean SDK to load and invoke third party SDK.

User can get parameters which are required to initialize thirdparty SDK in onReceivedThirdpartyRequest callback using getMediationData() method. The MASTMediationData object contains networkId, networkName, adUnitId and trackers.

```
public void onReceivedThirdPartyRequest (MASTNativeAd ad,
Map < String, String > properties, Map < String, String > parameters){
   MASTMediationData mediationData = ad.getMediationData();
  if(mediationData != null) {
// Indicates mediation response received from thirdparty ad server
String adId = mediationData.getAdId();
// Use this adId as placementId for initializing third party SDK
  }
  else {
/* mediationData=null indicates: mediation response received directly from mocean adserver */
 }
```

Where To Go Next

More thorough, complex examples and additional use cases in the sample application distributed with the SDK. Both the sample app and the SDK itself are available in source code. Additional documentation, information, and other supported platforms on our developer wiki at: http://developer.moceanmobile.com/SDKs.