Verve Ad Library SDK

Overview

Verve's mobile SDK allows you to monetize your Android app with Verve's network of high quality advertisers.

This introduction assumes familiarity with the Android platform and how to incorporate 3rd party libraries. See the sample project for implementation examples.

To display banners in your Android app, simply incorporate the SDK into your Android project, add a com.vervewireless.advert.AdView to your UI and pass it an AdRequest.

Getting Started

SDK Files

- **AndroidSDKDocumentation1.3.17.rtf** - This is the file you are currently reading.
- **Verve_AdSDK_1.3.17.jar** - This is the Java library to include in your applications.
- **Verve_AdSDK_Sample_1.3.17.apk** - This is an Android sample app built using the example code.
- **samples** - This is the directory containing sample code for the Android AdSDK.
    - **vervesdk_sample** - Sample project for Verve Banner and Interstitial Ads.
    - **dfp_banner_sample** - Sample project for DFP Mediation with Banner Ads.
    - **dfp_interstitial_sample** - Sample project for DFP Mediation with Interstital Ads.
    - **mopub_banner_sample** - Sample project for MoPub Mediation with Banner Ads.
    - **mopub_interstitial_sample** - Sample project for MoPub Mediation with Interstitial Ads.

System Requirements

The Verve Ad SDK requires a run-time of Android 2.3 or later (set minSdkVersion to at least 9).

Project Setup

1. Obtain Partner (Ad) Keyword

Obtain a partner (ad) keyword from your Verve Mobile Account Manager to allow you to retrieve Vere Ads. A temporary partner (ad) keyword of “adsdkexample” has been included in the samples for your convenience. This keyword will allow you to see ads filled 100% of the time (banner and interstitial) and can be used for testing.

2. Add Verve Ad SDK To Project

Add the Verve Ad SDK (Verve_AdSDK_1.3.17.jar file) to your Android project.

3. Configure Dependencies for IDE

3.1 Eclipse

1. Add the Android Support Library (android-support-v4.jar file) to your Eclipse project. See the instructions for how to download and add the Support Library without resources to your project: https://developer.android.com/tools/support-library/setup.html
2. Add and reference the Google Play services library project in your Eclipse workspace. See the instructions for how to set up the Google Play services SDK: https://developer.android.com/google/play-services/setup.html

3.2 Android Studio

1. Open your module's build.gradle file and add the following dependencies:

```
dependencies {
    compile 'com.android.support:appcompat-v7:21.+'
    compile 'com.google.android.gms:play-services-ads:7.0.0'
    compile 'com.google.android.gms:play-services-location:7.0.0'
    compile files('libs/Verve_AdSDK_1.3.17.jar');
}
```

2. Update the module's gradle build tools version to be 22.0.0. **Note: Compiling with 23.0.0 rc1 may cause errors.**

```
buildToolsVersion "22.0.0"
```

3. Add the following permissions to your app's manifest file:

```
 <uses-permission android:name="android.permission.INTERNET"/>
 <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
 <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
 <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
 <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Note: You can use ad functionality without the ACCESS_FINE_LOCATION permission, but when using the Advertising Attribution functionality this permission is mandatory.

4. Declare AdActivity and FullscreenAdActivity in your app's manifest file within the application tag.

```
 <application>

     <activity android:name="com.vervewireless.advert.AdActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenSize|sm
allestScreenSize|screenLayout|uiMode"
android:theme="@android:style/Theme.Translucent"/>

     <activity
android:name="com.vervewireless.advert.FullscreenAdActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenSize|sm
allestScreenSize|screenLayout|uiMode"/>

 </application>
```

5. Add the Verve Local Notification Receiver to your app's manifest file within the application tag.

```
 <application>

     <receiver
android:name="com.vervewireless.advert.LocalNotificationReceiver">
         <intent-filter>
             <action
android:name="android.intent.action.BOOT_COMPLETED"/>
         </intent-filter>
     </receiver>

 </application>
```

Banner Ads

To display banners in your Android app, simply add a com.vervewireless.advert.AdView to your UI. Like any View, an AdView may be created either purely in code or largely in XML.

Option 1: Create your banner in code and request the ad

```
import com.vervewireless.advert.*;

public class ExampleActivity extends Activity {
    private AdView mAdView;

    //update this value with your Verve Mobile supplied keyword
    private String MY_AD_KEYWORD = "adsdkexample";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Create the adView.
        mAdView = new AdView(this);
        mAdView.setAdKeyword(MY_AD_KEYWORD);
        mAdView.setAdSize(AdSize.BANNER);

        // Add a listener to the ad view to override
        // the ad methods (optional).
        mAdView.setAdListener(new AdListener() {
            @Override
            public void onAdLoaded(AdResponse response) {
                Log.d("Test", "AdView loaded");
```

```
            }

            @Override
            public void onAdError(AdError e) {
                Log.e("Test", "AdView error:" + e);
                }

            @Override
            public void onNoAdReturned(AdResponse response) {
                Log.d("Test", "AdView no ad");
            }

            @Override
            public void onAdPageFinished() {
                Log.d("Test", "AdView page finished loading");
            }
        });

        // Lookup your RelativeLayout assuming it's been given
        // the attribute android:id="@+id/mainLayout".
        RelativeLayout layout = (RelativeLayout)findViewById(R.id.mainLayout);

        // Add the adView to it.
        layout.addView(mAdView);

        // Create a new AdRequest.
        AdRequest adRequest = new AdRequest();

        // Add a category to the ad request.
        adRequest.setCategory(Category.NEWS_AND_INFORMATION);

        // Request the ad.
        mAdView.requestAd(adRequest);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mAdView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        mAdView.onPause();
    }
}
```

Option 2: Create your banner in XML and request the ad

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:verve="http://schemas.android.com/apk/lib/com.vervewireless.advert"
    android:id="@+id/mainLayout"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <com.vervewireless.advert.AdView
        verve:ad_keyword="adsdkexample"
        verve:ad_size="BANNER"
        android:id="@+id/adView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Note: The inclusion of the verve namespace (xlms) is referenced when specifying ad_keyword and ad_size.

To request an ad, look up the AdView as a resource via findViewById, create an AdRequest and pass it to the AdView.

```
import com.vervewireless.advert.*;

public class ExampleActivity extends Activity {
    private AdView mAdView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Find the adView.
        mAdView = (AdView) findViewById(R.id.adView);

        // Add a listener to the ad view to override
        // the ad methods (optional).
        mAdView.setAdListener(new AdListener() {
            @Override
            public void onAdLoaded(AdResponse response) {
                Log.d("Test", "AdView loaded");
```

```
            }

            @Override
            public void onAdError(AdError e) {
                Log.e("Test", "AdView error:" + e);
            }

            @Override
            public void onNoAdReturned(AdResponse response) {
                Log.d("Test", "AdView no ad");
            }

            @Override
            public void onAdPageFinished() {
                Log.d("Test", "AdView page finished loading");
            }
        });

        // Create a new AdRequest.
        AdRequest adRequest = new AdRequest();

        // Add a category to the ad request.
        adRequest.setCategory(Category.NEWS_AND_INFORMATION);

        // Request the ad.
        mAdView.requestAd(adRequest);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mAdView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        mAdView.onPause();
    }
}
```

Interstitial Ads

Verve publishers can also display full screen interstitial ads to their users at appropriate points in the application flow. **Note: This is an activity and therefore cannot be created in the**

resource layout file.

```
import com.vervewireless.advert.*;

public class ExampleActivity extends Activity {
    private InterstitialAd mInterstitialAd;

    //update this value with your Verve Mobile supplied keyword
    private String MY_AD_KEYWORD = "adsdkexample";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Instantiate the interstitial ad.
        mInterstitialAd = new InterstitialAd(this);
        mInterstitialAd.setAdKeyword(MY_AD_KEYWORD);

        // Add a listener to the ad view to override
        // the onAdReady and onAdFailed methods (optional).
        mInterstitialAd.setInterstitialAdListener(new InterstitialAdListener()
{
            @Override
            public void onAdReady() {
                mInterstitialAd.display();
            }
            @Override
            public void onAdFailed() {
            }
        });

        // Create a new AdRequest.
        AdRequest adRequest = new AdRequest();

        // Add a category to the ad request.
        adRequest.setCategory(Category.NEWS_AND_INFORMATION);

        // Request the ad.
        mInterstitialAd.requestAd(adRequest);
    }
}
```

Advanced Implementation

Verve's Ad SDK also contains optional functionality to help you monetize your application.

Ad Categories

Add a category to your requests to further specify the most relevant ads for your application.

To add a category, call adRequest.setCategory and pass a member of the Category enum:

```
adRequest.setCategory(Category.HOME_PAGE);
```

Ad Sizes

We supports two banner ad sizes for phones and three ad sizes for tablets:

```
BANNER("320x50"),
BANNER_FULL_WIDTH("FULLx50"),
TABLET_BANNER("728x90"),
TABLET_BANNER_FULL_WIDTH("FULLx90"),
TABLET_RECTANGLE("300x250")
```

To set an ad size, call mAdView.setAdSize and pass a member of the AdSize enum:

```
mAdView.setAdSize(AdSize.BANNER);
```

Limit User Tracking

You can decide to limit tracking on your users when doing ad requests. To change tracking, call adRequest.setLimitUserTrackingEnabled and pass true or false value:

```
adRequest.setLimitUserTrackingEnabled(true);
```

Arbitrary key/value parameters

The SDK provides means of specifying custom key/value parameters on ad requests. You can set custom parameter by calling adRequest.addCustomParameter method. Method returns Boolean value you can inspect to confirm that setting was successful. It will fail if you try to set a value for the key that is being used internally by the SDK.

```
adRequest.addCustomParameter("sample_key", "sample_value");
```

AdListener Interface

Implement the AdListener interface to have your activity notified when AdRequest events occur.

```
    mAdView.setAdListener(new AdListener() {
        @Override
        public void onAdLoaded(AdResponse response) {
            Log.d("Test", "AdView loaded");
        }

        @Override
        public void onAdError(AdError e) {
            Log.e("Test", "AdView error:" + e);
        }

        @Override
        public void onNoAdReturned(AdResponse response) {
            Log.d("Test", "AdView no ad");
        }

        @Override
        public void onAdPageFinished() {
            Log.d("Test", "AdView page finished loading");
        }
    });
```

AdClickedListener Interface

Implement the AdClickedListener interface to have your activity notified when user has clicked the ad. If you want to handle clicks on ad and override the default behavior, return true, otherwise return false.

```
mAdView.setAdClickedListener(new AdClickedListener() {
        @Override
        public boolean onAdClicked(Ad ad, Uri uri) {
            Log.d("Test", "AdView link clicked - uri: " + uri.toString());

            /*
             * If you want to handle clicks on ad
             * and override the default behavior,
             * return true, otherwise return false.
             */
            return false;
        }
    });
```

Advertising Attribution

The SDK leverages Android's ability to monitor a small set of regions (geographic or bluetooth low energy beacon) on the device. These regions are increasingly used by retail advertisers to determine if users exposed to their ads end up going to actual stores.

To enable advertising attribution feature, add these additional declarations to your app's manifest file within the application tag:

```
<application>

    <receiver android:name="com.vervewireless.advert.geofence.OnBootReceiver"
>
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>

    <service
android:name="com.vervewireless.advert.geofence.GeofenceBackgroundService"
android:exported="false" />

    <service
android:name="com.vervewireless.advert.geofence.GeofenceTransitionService"
android:exported="false" />

</application>
```

This feature is enabled by default. If you want to disable it you need to explicitly call:

```
GeofenceSettings.setActiveGeofencingEnabled(context, false);
```

If you want to use Advertising Attribution without the ad features you must explicitly call:

```
GeofenceSettings.setActiveGeofencingEnabled(context, true);
```

in your Activity onResume() method.

DoubleClick for Publishers (DFP) Network Mediation

See this article for a walk-through of what mediation is and how to set it up in DFP: https://support.google.com/dfp_sb/answer/2675711

Initial configuration

1. To enable mediation, log into the DFP site: https://www.google.com/dfp/
2. Create an Ad Unit

   2.1. Click on "Inventory" tab.

   2.2. Create a new ad unit or click an existing ad unit.

   2.3. Click on "Generate tags" and choose "Mobile applications". Here you will see "Ad Unit ID". You will need this in the application code to reference it.
3. Create an Order and Line Item

   3.1. Click on "Orders" tab.

   3.2. Create a new order or click an existing order in the table.

   3.3. Click "New line item".

   3.4. Enter a line item name and inventory sizes.

   3.5. Enter the line item type, dates, quantity and cost.

   3.6. Under "Adjust delivery", configure your delivery settings (optional).

   3.7. Select the inventory you want to target (ad unit that you previously created).

3.8. Click "Save".

4. Add a Creative

   4.1. Click an existing line item where you'd like to add the creative.

   4.2. Click "New creative" and select the creative's dimensions.

   4.3. Choose "Mobile App" and "SDK mediation".

   4.4. Under "Creative settings", configure creative's name and input the following information:

   Banner Ads

```
Select network: Custom Event
Parameter: <empty>
Label: Verve Ad Network
Class Name: com.vervewireless.advert.mediation.VerveCustomEventBanner
```

   Interstitial Ads

```
Select network: Custom Event
Parameter: <empty>
Label: Verve Ad Network
Class Name:
com.vervewireless.advert.mediation.VerveCustomEventInterstitial
```

   4.5. Click "Save".

   4.6. Configure your project by following steps at the beginning of this document (see "Getting Started" section).

   4.7 Add the following activity to your app's manifest file within the application tag to enable DFP mediation:

```
 <application>

     <activity android:name="com.google.android.gms.ads.AdActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenSize|
smallestScreenSize|screenLayout|uiMode"
android:theme="@android:style/Theme.Translucent" />

 </application>
```

DFP Banner Ads

To display a DFP banner in your application, add a com.google.android.gms.ads.doubleclick.PublisherAdView to your UI and request the ad.

```
public class BannerSample extends Activity implements VerveAdHandler
{
    /** The view to show the ad. */
    private PublisherAdView adView;

    /** The listener to be notified about activity lifecycle. */
    private VerveActivityListener verveActivityListener;

    //update this value with your DFP Ad Unit ID
    private static final String MY_AD_UNIT_ID = "";

    //update this value with your Verve Mobile supplied keyword
    private static final String MY_PARTNER_KEYWORD = "adsdkexample";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Create the adView.
        adView = new PublisherAdView(this);
        adView.setAdUnitId(MY_AD_UNIT_ID);
        adView.setAdSizes(AdSize.BANNER);

        // Lookup your RelativeLayout assuming it's been given
        // the attribute android:id="@+id/mainLayout"
        RelativeLayout layout = (RelativeLayout)
findViewById(R.id.mainLayout);
```

```
        // Add the adView to it.
        RelativeLayout.LayoutParams params = new
RelativeLayout.LayoutParams(LayoutParams.MATCH_PARENT,
LayoutParams.WRAP_CONTENT);
        params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);

        // Add the adView to it.
        layout.addView(adView, params);

        // Request the ad.
        adView.loadAd(buildRequest());
    }

    private PublisherAdRequest buildRequest() {
        // Verve extra parameters.
        VerveExtras verveExtras = new VerveExtras();
        verveExtras.setPartnerKeyword(MY_PARTNER_KEYWORD);
        verveExtras.setCategory(Category.NEWS_AND_INFORMATION);

        CustomEventExtras customExtras = new CustomEventExtras();
        customExtras.setExtra(VerveExtras.EXTRAS_LABEL, verveExtras);

        // Create the ad request with these extra parameters.
        PublisherAdRequest request = new PublisherAdRequest.Builder()
            .addNetworkExtras(customExtras)
            .build();

        return request;
    }

    @Override
    public void setVerveActivityListener(VerveActivityListener listener) {
        verveActivityListener = listener;
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (verveActivityListener != null) {
            verveActivityListener.onResume();
        }
        adView.resume();
    }

    @Override
    protected void onPause() {
        super.onPause();
```

```
        if (verveActivityListener != null) {
            verveActivityListener.onPause();
        }
        adView.pause();
    }

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

DFP Interstitial Ads

You can also display DFP interstitials at appropriate points in the application flow. Create an instance of com.google.android.gms.ads.doubleclick.PublisherInterstitialAd and request the ad.

```
public class InterstitialSample extends Activity {
    private PublisherInterstitialAd interstitialAd;

    //update this value with your DFP Ad Unit ID
    private String MY_AD_UNIT_ID = "";

    //update this value with your Verve Mobile supplied keyword
    private String MY_PARTNER_KEYWORD = "adsdkexample";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Instantiate the interstitial ad.
        interstitialAd = new PublisherInterstitialAd(this);
        interstitialAd.setAdUnitId(MY_AD_UNIT_ID);

        // Set the AdListener and override
        // the onAdLoaded method to show the ad.
        interstitialAd.setAdListener(new AdListener() {

            /** Called when an ad is loaded. */
            @Override
            public void onAdLoaded() {
                Log.d(InterstitialSample.LOG_TAG, "onAdLoaded");
                interstitialAd.show();
```

```
            }

            /** Called when an ad failed to load. */
            @Override
            public void onAdFailedToLoad(int errorCode) {
                Log.d(InterstitialSample.LOG_TAG, "onAdFailedToLoad");
            }

            /**
             * Called when an Activity is created in front of the app (e.g. an
interstitial is shown, or an
             * ad is clicked and launches a new Activity).
             */
            @Override
            public void onAdOpened() {
                Log.d(InterstitialSample.LOG_TAG, "onAdOpened");
            }

            /** Called when an ad is closed and about to return to the
application. */
            @Override
            public void onAdClosed() {
                Log.d(InterstitialSample.LOG_TAG, "onAdClosed");
            }

            /**
             * Called when an ad is clicked and going to start a new Activity
that will leave the
             * application (e.g. breaking out to the Browser or Maps
application).
             */
            @Override
            public void onAdLeftApplication() {
                Log.d(InterstitialSample.LOG_TAG, "onAdLeftApplication");
            }
        });

        // Request the ad.
        interstitialAd.loadAd(buildRequest());
    }

    private PublisherAdRequest buildRequest() {
        // Verve extra parameters.
        VerveExtras verveExtras = new VerveExtras();
        verveExtras.setPartnerKeyword(MY_PARTNER_KEYWORD);
        verveExtras.setCategory(Category.NEWS_AND_INFORMATION);
```

```
        CustomEventExtras customExtras = new CustomEventExtras();
        customExtras.setExtra(VerveExtras.EXTRAS_LABEL, verveExtras);

        // Create the ad request with these extra parameters.
        PublisherAdRequest request = new PublisherAdRequest.Builder()
            .addNetworkExtras(customExtras)
            .build();

        return request;
    }
}
```

MoPub Network Mediation

The MoPub SDK can mediate Verve ads through custom events.

Initial configuration

1. To enable mediation, log into MoPub site: https://app.mopub.com/
2. Create an Ad Unit

   2.1. Click on "Inventory" tab.

   2.2. Create a new app for Android platform.

   2.3. In next step, create an Ad Unit for this app.

   2.4. Click on "Save and View Code Integration".

   2.5. Here you can see "Your Ad Unit ID". You will need this in the application code to reference it.
3. Create a Network

   3.1. Click on "Networks" tab.

   3.2. Click "Add a Network" and select "Custom Native Network".

   3.3 Enter title (Verve Ad Network).

   3.4. Under "Set Up Your Inventory", input the following information:

   Banner Ads:

```
Custom Event Method: <empty>
Custom Event Class: com.mopub.mobileads.VerveBanner
Custom Event Class Data: {"ad_keyword":"<value>", "ad_width":320,
"ad_height":50}
```

Interstitial Ads:

```
Custom Event Method: <empty>
Custom Event Class: com.mopub.mobileads.VerveInterstitial
Custom Event Class Data: {"ad_keyword":"<value>"}
```

Note: The value for "ad_keyword" is the "partner keyword"" that you obtained from your Verve Account Manager.

3.5. Click "Save and Continue". Don't forget to click the "Run" button (a new network is paused by default).

3.5. After you are done on MoPub site, configure your project by following steps at the beginning of this document (see "Getting Started" section).

3.6. Add the following additional dependencies to the module gradle file to support MoPub Mediation

```
dependencies {

    compile 'com.android.support:support-annotations:22.0.0'
    compile 'com.mopub.volley:mopub-volley:1.1.0'
    compile (name:'mopub-sdk-3.8.0', ext:'aar')
}
```

3.7. Add the following repository and flat directory to the module gradle file to support MoPub Mediation

```
repositories {
    flatDir {
        dirs 'libs'
    }
    jcenter()
}
```

3.8. Add a reference the MoPub SDK library project. See the instructions for how to download and set up the MoPub SDK: https://github.com/mopub/mopub-android-sdk/wiki/Getting-Started

Note: This version of the Verve Ad SDK was tested with the standard distribution of MoPub v3.8.0. Verve cannot guarantee functionality if other distributions of the MoPub SDK are utilized or if modifications to the standard distribution are made.

Note: MoPub includes a proguard.cfg file in both the mopub-sdk and mopub-sample directories. The contents of these identical files should be included in your Proguard config when using the MoPub SDK in a Proguarded project. https://github.com/mopub/mopub-android-sdk/blob/master/mopub-sample/proguard.cfg

MoPub Banner Ads

To display a MoPub banner in your application, add a com.mopub.mobileads.MoPubView to your UI and request the ad. To allow MoPub to request ads from the Verve ad SDK, implement the VerveAdHandler for your Activity.

```
public class BannerSample extends Activity implements VerveAdHandler
{
    private MoPubView adView;
    private VerveActivityListener verveActivityListener;

    //update this value with your MoPub Ad Unit ID
    private static final String MY_AD_UNIT_ID = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Create the adView.
        adView = new MoPubView(this);
        adView.setAdUnitId(MY_AD_UNIT_ID);
```

```
        // Lookup your RelativeLayout assuming it's been given
        // the attribute android:id="@+id/mainLayout"
        RelativeLayout layout = (RelativeLayout)
findViewById(R.id.mainLayout);

        // Add the adView to it.
        layout.addView(adView);

        // Request the ad.
        loadAd();
    }

    public void loadAd() {
        // Verve extra parameters.
        VerveExtras verve = new VerveExtras();
        verve.setCategory(Category.NEWS_AND_INFORMATION);

        Map<String, Object> localExtras = new HashMap<String, Object>();
        localExtras.put(VerveExtras.EXTRAS_LABEL, verve);

        adView.setLocalExtras(localExtras);
        adView.loadAd();
    }

    @Override
    public void setVerveActivityListener(VerveActivityListener listener) {
        this.verveActivityListener = listener;
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (verveActivityListener != null) {
            verveActivityListener.onResume();
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (verveActivityListener != null) {
            verveActivityListener.onPause();
        }
    }

    @Override
```

```
    protected void onDestroy() {
        super.onDestroy();
        adView.destroy();
    }
}
```

MoPub Interstitial Ads

You can also display MoPub interstitials at appropriate points in the application flow. Create an instance of com.mopub.mobileads.MoPubInterstitial and request the ad.

```
public class InterstitialSample extends Activity {
    private MoPubInterstitial interstitialAd;

    //update this value with your MoPub Ad Unit ID
    private static final String MY_AD_UNIT_ID = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Instantiate the interstitial ad.
        interstitialAd = new MoPubInterstitial(this, MY_AD_UNIT_ID);

        // Set the InterstitialAdListener and override
        // the onInterstitialLoaded method to show the ad.
        interstitialAd.setInterstitialAdListener(new InterstitialAdListener()
{
            /** Called when an ad is loaded. */
            @Override
            public void onInterstitialLoaded(MoPubInterstitial interstitial)
{
                interstitialAd.show();
                Log.d(InterstitialSample.LOG_TAG, "onInterstitialLoaded");
            }

            /** Called when an ad failed to load. */
            @Override
            public void onInterstitialFailed(MoPubInterstitial interstitial,
MoPubErrorCode errorCode) {
                Log.d(InterstitialSample.LOG_TAG, "onInterstitialFailed");
            }

            /**
```

```
             * Called when an Activity is created in front of the app (an
interstitial is shown).
             */
            @Override
            public void onInterstitialShown(MoPubInterstitial interstitial)
{
                Log.d(InterstitialSample.LOG_TAG, "onInterstitialShown");
            }

            /**
             * Called when an ad is clicked and going to start a new Activity
that will leave the
             * application (e.g. breaking out to the Browser or Maps
application).
             */
            @Override
            public void onInterstitialClicked(MoPubInterstitial
interstitial) {
                Log.d(InterstitialSample.LOG_TAG, "onInterstitialClicked");
            }

            /** Called when an ad is closed and about to return to the
application. */
            @Override
            public void onInterstitialDismissed(MoPubInterstitial
interstitial) {
                Log.d(InterstitialSample.LOG_TAG, "onInterstitialDismissed");
            }
        });

        // Request the ad.
        loadInterstitial();
    }

    public void loadInterstitial() {
        // Verve extra parameters.
        VerveExtras verve = new VerveExtras();
        verve.setCategory(Category.NEWS_AND_INFORMATION);

        Map<String, Object> localExtras = new HashMap<String, Object>();
        localExtras.put(VerveExtras.EXTRAS_LABEL, verve);

        interstitialAd.setLocalExtras(localExtras);
        interstitialAd.load();
    }

    @Override
```

```
    protected void onDestroy() {
        super.onDestroy();
        interstitialAd.destroy();
    }
}
```

Release Notes

Version 1.3.17 (6/18/2015)

- Using MoPub SDK version 3.8.0
- Using Google Play Service Version 7.0.0
- Fixed Ad banners become unresponsive when Verve SDK does a refresh to an ad which was preceded by a no-fill

Version 1.3.16 (05/05/2015)

- Advertising Attribution feature is turned on by default.
- Changes in ad request parameters.
- Fixed issue for using Advertising Attribution without ad features
- Ad functionality can be used without the ACCESS_FINE_LOCATION permission
- Fixed crash if meta-data tag for GooglePlayServices is missing

Version 1.3.15 (02/11/2015)

- Sample projects moved to Android Studio environment.
- Fixed integration issue for MoPub Network Mediation.

Version 1.3.14 (12/30/2014)

- Added option to set multiple categories for ad request.
- Additional changes in ad request parameters.
- Added Advertising Attribution feature (optional).
- Added support for latest Android SDK (5.0 or later).
- Removed non-standard files from library jar.

Version 1.3.13 (09/15/2014)

- Added support for attaching custom key/value pair to the ad request.
- Fixed minor issue related to MoPub Network Mediation.

Version 1.3.12 (08/29/2014)

- Added support for MoPub Network Mediation by using custom events.

- Added VRVSDK JS Bridge (top-level JavaScript object in ad container WebView). It can be used to access native functionality of OS.

Version 1.3.11 (08/08/2014)

- Fixed issue when click on banner is loading URL within banner slot when target="_blank" (on devices runing Android 4.4 or later).

Version 1.3.10 (07/14/2014)

- Fixed issue when click on banner is not working if ad contains nested iframes.

Version 1.3.9 (06/06/2014)

- Added support for DFP Network Mediation by using custom events.
- Fixed issues when using latest Android SDK (4.4 or later).

Version 1.3.8 (04/04/2014)

- Added two extra params to the ad request for easy migration of users from ANDROID_ID to Google Play Services Advertising ID without loosing track of them.

Version 1.3.7 (03/21/2014)

- Verve Ad SDK is compliant with the MRAID v2.0 specification.

Version 1.3.6 (03/18/2014)

- Removed need for setting "base_url" ("ad_keyword" is now required).
- Added two new "full width" ad sizes.
- Using Google Play Services Advertising ID instead of the ANDROID_ID for user tracking.
- Added switch to limit user tracking.

Version 1.3.5 (12/18/2013)

- Added two tablet ad sizes in addition to the 320x50 banner for phones.
- Added ability to scale up ad.
- Added extra information params to the ad request.
- Added default value for "p" parameter in base_url ("anap" for phone and "apad" for tablet).
- Link "tel:1234567890" will bring up the phone dialer to call from Interstitial Ad.

Version 1.3.4 (08/20/2013)

- Enabled HTML5 videos in Interstitial Ads (Android >= 3.0).
- Fixed an issue with Interstial Ads firing duplicate event calls.

Version 1.3.3 (06/06/2013)

- Library version number passed with Ad requests.
- Device ID source information passed with ad requests.
- Application and device information passed with Ad requests.
- Banner position information passed with Ad requests.
- Fixed an issue with negative value of location age passed with Ad requests.
- Updated button images for interstitial Ad.
- Updated html templates for Ad content.
- Some minor bugs fixed.

Version 1.3.2 (01/04/2013)

- Location precision parameter renamed to "lacc". We are passing the accuracy of the location (not the precision) so that makes more sense.

Version 1.3.1 (12/14/2012)

- Location age and precision passed with Ad requests.
- Fixed an issue with AdView so that when it appears on the screen it doesn't flicker and show the previous ad.

Version 1.3.0 (06/11/2012)

- Includes interstitial Ad support.

Version 1.2.6 (05/07/2012)

- Updated the project to support the latest Android SDK and tools (r19).
- Fixed an issue which would cause Ads to not be visible (under certain configurations).

Version 1.2.5 (03/29/2012)

- Removed locationListener code to avoid orphaned listeners.
- Added onNoAdReturned callback for empty response handling.

Version 1.2.4 (01/27/2012)

- Fixed an issue where certain Verve Ads with iFrames were not loading.

Version 1.2.3 (01/11/2012)

- Removes the requirement that apps using the Ad SDK must have a GPS.

Version 1.2.2 (11/28/2011)

- Ads are centered.

- Ad requests are now made with a browser-based user-agent.
- Fixes issues with query parameters on images.
- Ads do not queue up with empty responses and properly timeout.
- Fixed Ad click behavior which sometimes did not work.

Version 1.2.1 (08/12/2011)

- Fixed an issue with Ads clicks not registering