Flurry Advertising

Android SDK Instructions

SDK version 5.5.0
Updated: 04/29/2015

Welcome to Flurry Advertising!

This README contains:

---

1. Introduction

Flurry Advertising is a flexible ad serving solution that allows you to easily manage the complete monetization of your mobile applications. Through integration with Flurry Analytics and various mobile ad networks, publishers can easily generate the maximum value from ad campaigns. Both FFA (Flurry For Advertisers) and FFP (Flurry For Publishers - direct and custom network) ad campaigns are created and managed in the same interface.

With FFP you will be able to:

1. Define your inventory
2. Traffic your ad campaigns
3. Track & optimize your performance

The Flurry SDK contains all the existing Flurry Analytics functionality as well as the new Advertising functionality. It is designed to be as easy as possible with a basic setup completed in under 5 minutes.

These instructions assume that you have already integrated Flurry Analytics into your application. If you have not done so, please refer to **Analytics-README** to get started.

For ad space setup and more information on Flurry advertising, please visit
https://developer.yahoo.com/flurry/docs/publisher/

Please note, it is **required** that you create ad spaces before retrieving ads. Adspaces can be created on the Flurry Developer Portal or in the application code. If Adspaces are created in the code, they will appear in the dev portal designated as “Determined by SDK” in the Adspace setup page.

Flurry Advertising uses the Android Advertising ID provided by Google Play Services and will check for and respect the user's ad tracking preference. For more information, please visit https://developer.android.com/google/play-services/id.html

---

2. Basic Integration

Flurry Ads requires minimum Android API level 10. To integrate Flurry Advertising into your Android application, just complete two steps:

Step 1. Include the appropriate jar

Flurry Ads work on top of Flurry analytics. Add FlurryAds_5.5.0.jar file (in addition to the the FlurryAnalytics_5.5.0.jar and) to your classpath.

Using Android Studio:

1. Add FlurryAnalytics_5.5.0.jar and FlurryAds_5.5.0.jar to your project's libs folder.
2. Navigate to File > Project Structure > Module > Dependencies. Click the ‘+’ button in the bottom of the ‘Project Structure’ popup to add dependencies. Select ‘File dependency’ and add libs/FlurryAnalytics_5.5.0.jar and libs/FlurryAds_5.5.0.jar and.
3. Add Google Play Services library. Please follow instructions at http://developer.android.com/google/play-services/setup.html#Setup
4. Add v4 support library (or greater). Please follow instructions at https://developer.android.com/tools/support-library/setup.html#add-library

Using Eclipse:

1. Add FlurryAnalytics_5.5.0.jar and FlurryAds_5.5.0.jar to your project’s libs folder. Right click on each JAR file and select Build Path > Add to Build Path.
2. Add Google Play Service library. Please follow instructions at http://developer.android.com/google/play-services/setup.html#Setup
3. Add v4 Support Library (or greater). Please follow instructions at https://developer.android.com/tools/support-library/setup.html#add-library.

Step 2. Declare our Activity in your manifest

Declare the Activity “FlurryFullscreenTakeoverActivity” in your AndroidManifest.xml file:

```
<activity
android:name="com.flurry.android.FlurryFullscreenTakeoverActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|u
iMode|screenSize|smallestScreenSize"
</activity>
```

This is required to show interstitial ads.

Optional Permission (Highly Recommended):

`android.permission.WRITE_EXTERNAL_STORAGE`

External storage is used for pre-caching features if available.

Step 3. Requesting an Ad

3.1 Banner Ad

In order to serve banner ads please use `FlurryAdBanner` and `FlurryAdBannerListener`:

```
   public class Example extends Activity {
        private RelativeLayout mBanner;
        private FlurryAdBanner mFlurryAdBanner = null;
        private String mAdSpaceName = "BANNER_ADSPACE_NAME";

        public void onCreate(Bundle bundle) {
            super.onCreate(bundle);
            setContentView(R.layout.example);
            mBanner = (RelativeLayout)findViewById(R.id.banner);
            mFlurryAdBanner = new FlurryAdBanner(this, mBanner, mAdSpaceName);

            // allow us to get callbacks for ad events
            mFlurryAdBanner.setListener(bannerAdListener);
        }
        public void onStart() {
            super.onStart();
            FlurryAgent.onStartSession(this);
            // fetch and prepare ad for this ad space. won't render one yet
            mFlurryAdBanner.fetchAd();
        }
        public void onStop() {
            super.onStop();
            FlurryAgent.onEndSession(this);
            mFlurryAdBanner.destroy();
        }

       FlurryAdBannerListener bannerAdListener = new FlurryAdBannerListener() {

       @Override
       public void onFetched(FlurryAdBanner adBanner) {
             adBanner.displayAd();
       }

       @Override
       public void onError(FlurryAdBanner adBanner, FlurryAdErrorType
adErrorType, int errorCode) {
             adBanner.destroy();
```

```
    }

    @Override
    public void onRendered(FlurryAdBanner adBanner) {}

    @Override
    public void onShowFullscreen(FlurryAdBanner adBanner) {}

    @Override
    public void onCloseFullscreen(FlurryAdBanner adBanner) {}

    @Override
    public void onAppExit(FlurryAdBanner adBanner) {}

    @Override
    public void onClicked(FlurryAdBanner adBanner) {}

    @Override
    public void onVideoCompleted(FlurryAdBanner adBanner) {}
    };
}
```

Implementing FlurryAdBannerListener

To be notified of certain events during the full lifecycle of the Ad, you shall implement the `FlurryAdBannerListener` interface and then call the setListener method to attach your implementation of `FlurryAdBannerListener` to the Flurry SDK. You will need to implement the following callback methods:

- `onClicked(FlurryAdBanner adBanner)`
  - This method will be called when the user has clicked on the ad.
- `onCloseFullscreen(FlurryAdBanner adBanner)`
  - This method will be called when the user dismisses the current Ad for the provided Ad Space name.
- `onShowFullscreen(FlurryAdBanner adBanner)`
  - This method will be called when the user has opened the ad.
- `onAppExit(FlurryAdBanner adBanner)`
  - This method will be called when the user is leaving the application after following events associated with the current Ad in the provided Ad Space name.
- `onRendered(FlurryAdBanner adBanner)`
  - This method will be called when ad is successfully rendered.
- `onError(FlurryAdBanner adBanner, FlurryAdErrorType adErrorType, int errorCode)`
  - This method will be called when fetch, render or click failed.
- `onFetched(FlurryAdBanner adBanner)`
  - This method will be called when the ad has been received from the server
- `onVideoCompleted(FlurryAdBanner adBanner)`

- This method is present only In case the ad served is a video clip and adspace is marked as client side rewarded, or rewarded.

Note: please see **Analytics-README** for details on `FlurryAgent.onStartSession()`

Here are the details on the parameters to `FlurryAdBanner`:

- **`Context context`** - We advise using the current Activity.
- **`String adSpace`** - Unique name of your ad placement defined on the Flurry website.
- **`ViewGroup viewSpace`** - ViewGroup inside the Context that you want the ad to reside in. We suggest using a `RelativeLayout`
- **`FlurryAdErrorType`** `adErrorType` – Ad error types.

In order to specify the format and size, please visit Flurry's developer portal at https://dev.flurry.com/FFPSetupAdSpaces.do

3.2 Interstitial Ad

In order to serve interstitial ads please use `FlurryAdInterstitial` and `FlurryAdInterstitialListener`:

```
   public class Example extends Activity {

        private FlurryAdInterstitial mFlurryAdInterstitial = null;
        private String mAdSpaceName = "INTERSTITIAL_ADSPACE_NAME";

        public void onCreate(Bundle bundle) {
            super.onCreate(bundle);
            setContentView(R.layout.example);
            mFlurryAdInterstitial = new FlurryAdInterstitial(this,
mAdSpaceName);

            // allow us to get callbacks for ad events
            mFlurryAdInterstitial.setListener(interstitialAdListener);
        }
        public void onStart() {
            super.onStart();
            FlurryAgent.onStartSession(this);
            // fetch and prepare ad for this ad space. won't render one yet
            mFlurryAdInterstitial.fetchAd();
        }
        public void onStop() {
            super.onStop();
            FlurryAgent.onEndSession(this);
            mFlurryAdInterstitial.destroy();
        }

FlurryAdInterstitialListener interstitialAdListener = new
FlurryAdInterstitialListener() {
```

```
	@Override
	public void onFetched(FlurryAdInterstitial adInterstitial) {
		adInterstitial.displayAd();
	}

	@Override
	public void onError(FlurryAdInterstitial adInterstitial,
FlurryAdErrorType adErrorType, int errorCode) {
		adInterstitial.destroy();
	}

	@Override
	public void onRendered(FlurryAdInterstitial adInterstitial) {}

	@Override
	public void onDisplay(FlurryAdInterstitial adInterstitial) {}

	@Override
	public void onClose(FlurryAdInterstitial adInterstitial) {}

	@Override
	public void onAppExit(FlurryAdInterstitial adInterstitial) {}

	@Override
	public void onClicked(FlurryAdInterstitial adInterstitial) {}

	@Override
	public void onVideoCompleted(FlurryAdInterstitial adInterstitial) {}
	};
}
```

Implementing FlurryAdInterstitialListener

To be notified of certain events during the full lifecycle of the Ad, you shall implement the `FlurryAdInterstitialListener` interface and then call the setListener method to attach your implementation of `FlurryAdInterstitialListener` to the Flurry SDK. You will need to implement the following callback methods:

- `onClicked(FlurryAdInterstitial adInterstitial)`
  - This method will be called when the user has clicked on the ad.
- `onClose(FlurryAdInterstitial adInterstitial)`
  - This method will be called when the user dismisses the current Ad for the provided Ad Space name.
- `onDisplay(FlurryAdInterstitial adInterstitial)`
  - This method will be called when the user has opened the ad.
- `onAppExit(FlurryAdInterstitial adInterstitial)`
  - This method will be called when the user is leaving the application after following events associated with the current Ad in the provided Ad Space name.

- `onRendered(FlurryAdInterstitial adInterstitial)`
    - This method will be called when ad is successfully rendered.
- `onError(FlurryAdInterstitial adInterstitial, FlurryAdErrorType adErrorType, int errorCode)`
    - This method will be called when fetch, render or click failed. There can be many reasons for the failure.
- `onFetched(FlurryAdInterstitial adInterstitial)`
    - This method will be called when the ad has been received from the server
- `onVideoCompleted(FlurryAdInterstitial adInterstitial)`
    - In case the ad served is a video clip and adspace is marked as client side rewarded, this method

Note: please see **Analytics-README** for details on `FlurryAgent.onStartSession()`

Here are the details on the parameters to `fetchAd` and `displayAd`:

- **`Context context`** - We advise using the current Activity.
- **`String adSpace`** - Unique name of your ad placement, may defined on the Flurry website, but does not have to be
- **`FlurryAdErrorType`** `adErrorType` – Ad error types.

3.3 Native Ad

In order to serve native ads please use `FlurryAdNative` and `FlurryAdNativeListener`:

```
public class Example extends Activity {
    private FlurryAdNative mFlurryAdNative = null;
    private String mAdSpaceName = "NATIVE_ADSPACE_NAME";

    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.example);
        mFlurryAdNative = new FlurryAdNative(this, mAdSpaceName);
        // allow us to get callbacks for ad events
        mFlurryAdNative.setListener(nativeAdListener);
    }
    public void onStart() {
        super.onStart();
        FlurryAgent.onStartSession(this);

         // fetch and prepare ad for this ad space. won't render one yet
        mFlurryAdNative.fetchAd();
    }
    public void onStop() {
        super.onStop();
        FlurryAgent.onEndSession(this);

         // destroy the ad object
         mFlurryAdNative.destroy();
```

```
 }

FlurryAdNativeListener nativeAdListener = new FlurryAdNativeListener() {

@Override
public void onFetched(FlurryAdNative adNative) {

      // Fetch all native ad assets and create your native ad view.
      FlurryAdNativeAsset headlineAsset = adNative.getAsset("headline");
      FlurryAdNativeAsset summaryAsset = adNative.getAsset("summary");
      FlurryAdNativeAsset sourceAsset = adNative.getAsset("source");
      FlurryAdNativeAsset sponsoredImageUrl =
adNative.getAsset("secHqBrandingLogo");
      FlurryAdNativeAsset secHqImageAsset =
adNative.getAsset("secHqImage");
      FlurryAdNativeAsset secImageAsset = adNative.getAsset("secImage");
      FlurryAdNativeAsset sourceAsset = adNative.getAsset("source");

      // Get the layout and add the assets
      RelativeLayout mainLayout = (RelativeLayout)
findViewById(R.id.nativeAdLayout);

      // Create a view and show these assets
      TextView headlineView = (TextView) findViewById(R.id.headline);
      headlineView.setText(headlineAsset.getValue());

      // Note: It is mandatory to show sponsored text and image in the
native ad.
      TextView sponsoredText = (TextView) findViewById(R.id.sponsored);
      sponsoredText.setText("SPONSORED");

      // getAssets() for "secHqBrandingLogo", "secHqImage" and "secImage"
returns the url to the image. If the image is downloaded by the SDK
(depending upon your pre-caching settings) this url will be the file path
on the device otherwise an http url to the image.

      // set the tracking view once your view is created. We recommend
setting the entire ad view as tracking view.
      // If you plan to re-use the view to show different ads, do call
      // adNative.removeTrackingView() before setting the new tracking
view.
      adNative.setTrackingView(mainLayout);
}

@Override
public void onError(FlurryAdNative adNative, FlurryAdErrorType
adErrorType, int errorCode) {
```

```
	switch (adErrorType) {
		case FETCH:{
			if (maxFetchTries < 3) { // maxFetchTries is the counter to
			  // check how many times we have tried fetching an ad. If
			 // fetchAd fails for 3 times, destroy the native ad object.

				adNative.fetchAd();
			} else {
				adNative.destroy();
			}
			break;
		}
		case RENDER:{
			adNative.destroy();
			break;
		}
		case CLICK:{
			adNative.destroy();
			break;
		}
		default: {
			fAdStateText = "unknownError";
			break;
		}
		}
	}


	@Override
	public void onShowFullscreen(FlurryAdNative adNative) {}

	@Override
	public void onCloseFullscreen(FlurryAdNative adNative) {}

	@Override
	public void onClicked(FlurryAdNative adNative) {}

	@Override
	public void onImpressionLogged(FlurryAdNative adNative) {}

	@Override
	public void onAppExit(FlurryAdNative adNative) {}
	};
}
```

Implementing FlurryAdNativeListener

To be notified of certain events during the full lifecycle of the Ad, you shall implement the `FlurryAdNativeListener` interface and then call the setListener method to attach your implementation of `FlurryAdNativeListener` to the Flurry SDK. You will need to implement the following callback methods:

- `onClicked(FlurryAdNative adNative)`
    - This method will be called when the user has clicked on the ad.
- `onImpressionLogged(FlurryAdNative adNative)`
    - This method will be called when the user views the ad.
- `onCloseFullscreen(FlurryAdNative adNative)`
    - This method will be called when the user dismisses the current Ad for the provided Ad Space name.
- `onShowFullscreen(FlurryAdNative adNative)`
    - This method will be called when the user has opened the ad.
- `onAppExit(FlurryAdNative adNative)`
    - This method will be called when the user is leaving the application after following events associated with the current Ad in the provided Ad Space name.
- `onRendered(FlurryAdNative adNative)`
    - This method will be called when ad is successfully rendered.
- `onError(FlurryAdNative adNative, FlurryAdErrorType adErrorType, int errorCode)`
    - This method will be called when there is failure with fetch, render or click. There can be many reasons for the failure. Developer may want to handle each error differently and provide an input to user for some cases.
- `onFetched(FlurryAdNative adNative)`
    - This method will be called when the ad has been received from the server

Note: please see **Analytics-README** for details on `FlurryAgent.onStartSession()`

Here are the details on the parameters to ad apis:

- **`Context context`** - We advise using the current Activity.
- **`String adSpace`** - Unique name of your ad placement, may defined on the Flurry website, but does not have to be
- **`FlurryAdErrorType`** `adErrorType` – Ad error types.

Native Ad Tracking

To enable impression and click tracking, call setTrackingView() on the main ad container layout.

```
adNative.setTrackingView(mainLayout);
```

If you plan to re-use the view to show different ads, do call removeTrackingView() before setting the new ad tracking view.

```
adNative.removeTrackingView();
```

Native Ad Expiry

After an ad is fetched, you can explicitly check if the ad is expired by using the `adNative.isExpired()` `api.` If the ad is expired, destroy the ad object and fetch another ad.

Native Ads precaching

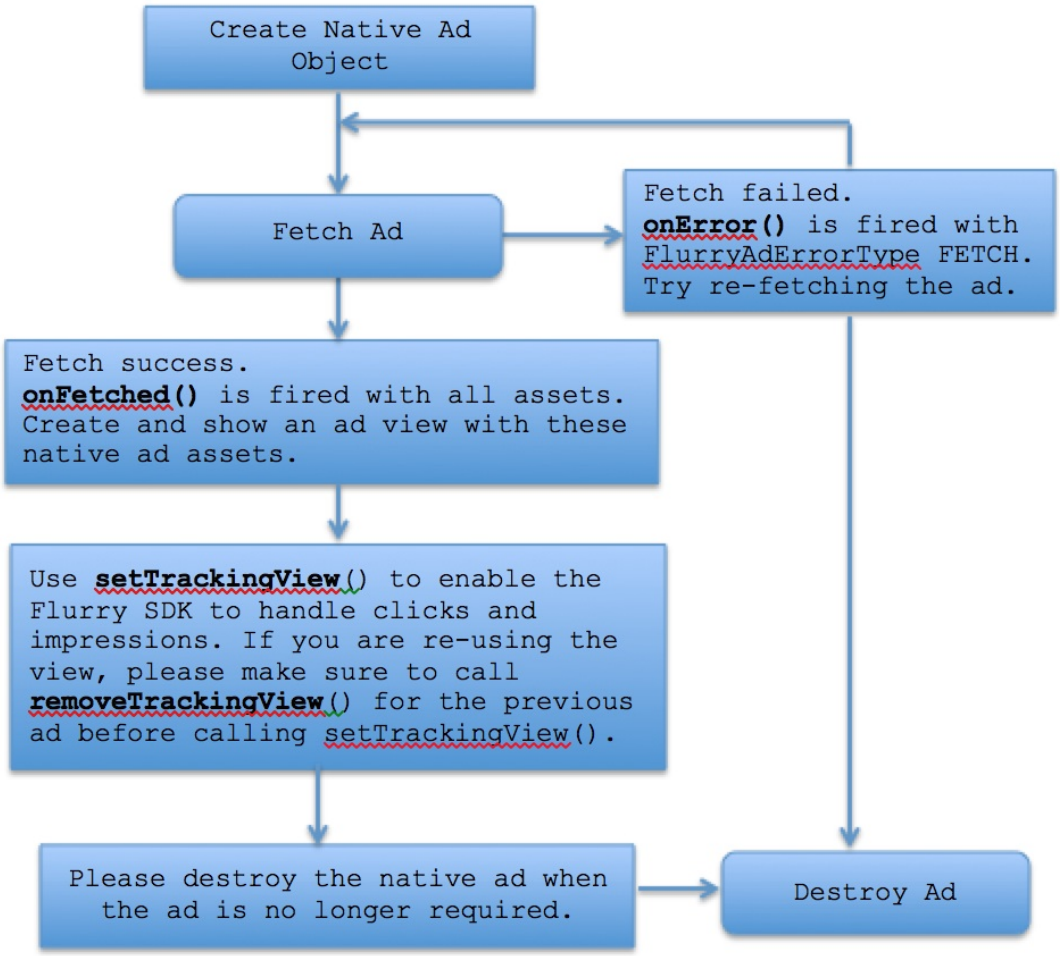
If precaching is enabled for a given ad space, all precacheable assets will be cached BEFORE the ad is ready and will available until the ad is destroyed. Assets are only cached on disk; it is up to the developer to manage an in-memory cache. Non-cached assets will be loaded directly from the network.

If you wish to implement a memory cache it is recommended to load the assets directly via URL instead of using loadAssetIntoView / getAssetView. Please note that precached asset urls in FlurryAdNativeAsset will be file:// and non-precached assets will be regular https:// urls.

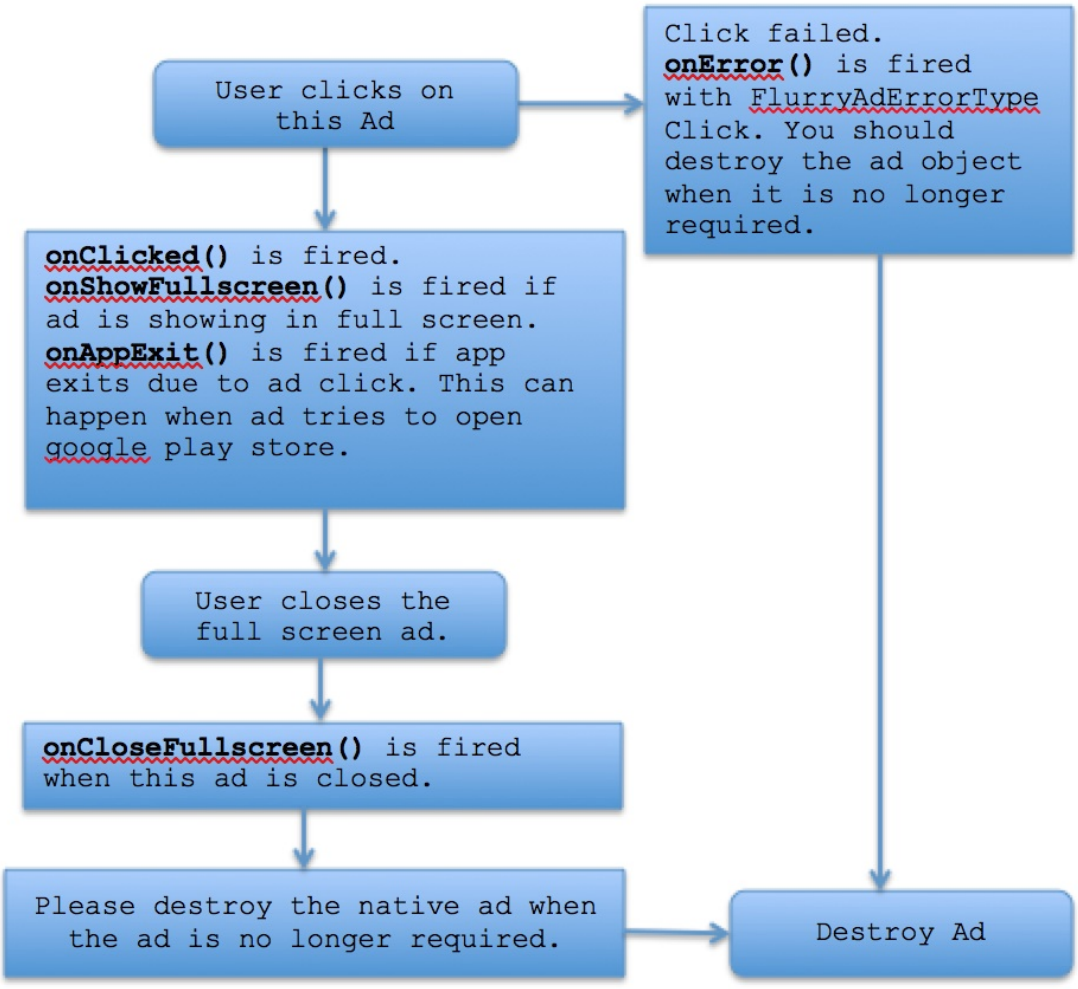
---

3. Native ad object lifecycle

Fetch Cycle



Click Cycle



---

4. Additional Ad Controls (Optional)

Keyword Targeting

Add a call to specify keywords to be used when targeting ads. Targeting allows partners to supply information they track internally, which is not available to Flurry's targeting system. If the targeting settings are matched on the ad server, a subset of targeted ads will be delivered to the user.
This api is available in all ad types - `FlurryAdBanner, FlurryAdInterstitial and FlurryAdNative.`

```
setTargeting(FlurryAdTargeting targeting)
```

Video Ads

Starting with version 4.0.0, Flurry Android SDK supports precaching of the video ads for enhanced user experience and better completion rate. Precaching is done for video ads served by Flurry marketplace and by Flurry FFA.

Since version 4.0.0, Flurry SDK supports displaying VAST Linear (including VAST Wrapper) ads from Flurry marketplace.

These features are supported automatically without any additional integration.

---

5. Enabling Ad Network Mediation (Optional)

Once you have your Ad Spaces set up, you will have the option of selecting 3rd party ad networks to serve ads into your Ad Spaces using the Flurry website (in addition to FFA, FFP and your own ads). You can change which ad networks serve ads at any time on the Flurry website, but in order to enable them you need to add the ad network SDKs into your application and configure them. The list of currently supported Ad Networks contains:

- Google Mobile Ads - included automatically in Google Play Services SDK
- Millennial - SDK Version 5.3.0
- InMobi - SDK Version 4.5.3

To implement an Ad Network you must perform the following steps:

1. Include the Ad Network Android SDK with your app and add it to the build path. Follow the instructions from the Ad network on how to complete this step.
2. Create the proper “activity” and “meta-data” tags in AndroidManifest.xml
    a. the first meta-data tag instructs the SDK about how to find the API_KEY
    b. the second meta-data tag instructs the SDK whether to request test ads
3. Add your API_KEY in strings.xml

Here is the example of implementing the Google Mobile Ads SDK:

AndroidManifest.xml

```
<activity android:name="com.google.android.gms.ads.AdActivity"
      android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|u
      iMode|screenSize|smallestScreenSize"/>
```

```
<meta-data android:name="com.flurry.gms.ads.MY_AD_UNIT_ID"
      android:value="@string/ffp_gms_apikey"/>

<meta-data android:name="com.flurry.gms.ads.MYTEST_AD_DEVICE_ID"
      android:value="@string/ffp_gms_testdevicekey"/>
<meta-data android:name="com.flurry.gms.ads.test"
      android:value="true"/>
```

strings.xml

```
<string name="ffp_gms_apikey">YOUR_API_KEY</string>
<string name="ffp_gms_testdevicekey">(add device id that shows in
logcat</string>

Note: admob.test value needs to be set to true above. This will enable admob
test ads to show up in the emulator and on a device.
```

Here is an example of implementing the Millennial SDK:

AndroidManifest.xml

```
<activity android:name="com.millennialmedia.android.MMActivity"
      android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|u
      iMode|screenSize|smallestScreenSize"
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />

<meta-data
      android:name="com.flurry.millennial.MYAPID"
      android:value="@string/ffp_millennial_apikey" />
<meta-data
    android:name="com.flurry.millennial.MYAPIDINTERSTITIAL"
    android:value="@string/ffp_millennial_apikey_interstitial" />
<meta-data
    android:name="com.flurry.millennial.MYAPIDRECTANGLE"
    android:value="@string/ffp_millennial_apikey_rectangle" />
```

strings.xml

```
<string name="ffp_millennial_apikey">YOUR_MILLENNIAL_API_KEY</string>
<string
name="ffp_millennial_apikey_interstitial">YOUR_MILLENNIAL_INTERSTITIAL_API_KEY<
/string>
<string
name="ffp_millennial_apikey_rectangle">YOUR_MILLENNIAL_INTERSTITIAL_API_RECTANG
LE</string>
```

Here is an example of implementing the Inmobi SDK:

AndroidManifest.xml

```
<activity android:name="com.inmobi.androidsdk.IMBrowserActivity”
```

```
      android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|u
iMode|screenSize|smallestScreenSize"
    android:hardwareAccelerated="true" />

<meta-data android:name="com.flurry.imobi.APP_ID"
    android:value="@string/ffp_inmobi_apikey"/>
```

strings.xml

```
<string name="ffp_inmobi_apikey">YOUR_INMOBI_API_KEY</string>
```

---

6. Ad Targeting Options (Optional)

There are a number of configuration parameters that you can use to modify the behavior of your ad spaces.

Option 1. Set Location

Call this method to set the current location (used with geographical targeting)

```
setLocation(float latitude, float longitude)
```

To remove current location.

```
clearLocation();
```

Option 2. User Cookies

Add a call to identify any user specific information you want associated with the ad request:

```
setUserCookies(Map<String, String> cookies)
```

To remove any user cookies call:

```
clearUserCookies()
```

Option 3. Set Keywords

Add a keywords to specify information on a user executing an ad action for the purposes of targeting.

```
setKeywords(Map<String, String> targetingKeywords)
```

To remove keyword selection.

```
clearKeywords();
```

Option 4. Set Age

Set the age of the user.

```
setAge(int age);
```

To remove the age.

```
clearAge();
```

Option 5. Set Gender

Set the gender of the user.

```
setGender(FlurryGender gender);
```

To remove the gender.

```
clearGender();
```

---

7. Using ProGuard (Optional)

If you plan to run ProGuard on your APK before releasing your app, you will need to add the following to your “proguard.cfg” file:

```
-keep class com.flurry.** { *; }
-dontwarn com.flurry.**
-keepattributes *Annotation*,EnclosingMethod
-keepclasseswithmembers class * {
public <init>(android.content.Context, android.util.AttributeSet, int);
}

# Google Play Services library
-keep class * extends java.util.ListResourceBundle {
  protected Object[][] getContents();
}

-keep public class
com.google.android.gms.common.internal.safeparcel.SafeParcelable {
  public static final *** NULL;
}

-keepnames @com.google.android.gms.common.annotation.KeepName class *
-keepclassmembernames class * {
  @com.google.android.gms.common.annotation.KeepName *;
}

-keepnames class * implements android.os.Parcelable {
  public static final ** CREATOR;
}
```

If you are using the Google Mobile Ads SDK, add the following:

```
# Preserve GMS ads classes
-keep class com.google.android.gms.ads.** { *; }
-dontwarn com.google.android.gms.ads.**
```

If you are using the InMobi SDK, add the following:

```
# Preserve InMobi Ads classes
-keep class com.inmobi.** { *; }
-dontwarn com.inmobi.**
```

If you are using the Millennial Media SDK, add the following:

```
# Preserve Millennial Ads classes
-keepclassmembers class com.millennialmedia.android.* {
public *;
}
-keep class com.millennialmedia.android.*
-dontwarn com.millennialmedia.**
```