Innity Mobile Ads

Android SDK v1.10 Implementation Guide

Getting Started

Thank you for including Innity Mobile Ads as a part of your mobile app monetisation strategy.

This document serves as a guide for app developers to implement Innity Mobile ads in Android apps. You'll see in this document that implementing the SDK is fairly easy, showing ads requires nothing more but a few lines of code. Happy coding!

Requirements

This Innity Mobile Ads Android SDK requires:

- Android Studio v3.0 or newer
- Android 4.4 (API 19) or newer

Ad Format Overview

This SDK comes with support for several ad formats:

Engage Ads

- **Spin Lite**
  This banner is presented as a circular disc overlay at the bottom of the screen taking up a full width 9:4 space (e.g. 414 x 184) The disc can be rotated by the user.

- **Spin**
  This banner is presented as a circular disc overlay at the bottom of the screen taking up a full width 9:4 space (e.g. 414 x 184) The disc can be rotated and when tapped, a series of swipeable cards are presented.

- **Spin+**
  This banner is presented as a circular disc overlay at the bottom of the screen taking up a full width 9:4 space (e.g. 414 x 184) The disc can be rotated to reveal a full screen ad behind the disc.

- **Revolver**
  This banner is presented as a series of cards arranged in a circle in 3D space. The cards can be rotated by swiping. The cards would expand to a series of larger cards, upon the second swipe by the user, also arranged in a circle in 3D space covering the full screen.

- **Pull**
  This banner is presented as a 320x50 unit that needs to be anchored to the bottom of the screen. The 320x50 unit can be "pulled" or swiped up to reveal a fullscreen banner. The fullscreen banner includes a close button. (Only use ONE pull banner per screen)

- **Interstitial**
  An interstitial banner is presented as a fullscreen unit with a close button.

- **Mobile Leaderboard (320x50)**
  This banner is a standard 320x50 unit without expansion.

- **Medium Rectangle (300x250)**
  This banner is a standard 320x50 unit without expansion.

- **Medium Rectangle Cube (300x250)**
  This banner is presented as a medium rectangle size banner that is periodically rotated like a cube.

- **Medium Rectangle Slider (300x250)**
  This banner is presented as a medium rectangle size slideshow that will periodically animated to the next slide. User can swipe the banner to navigate through the slideshow.

- **Mobile Cards**
  This banner is presented as a stack of cards. Only one card is visible to the user. The card at the front can be sent to the back by swiping the card to the left or right.

- **Splash Cards**
  This banner is presented with a brief splash screen. After the splash screen animation ended, the banner shrinks into a mobile cards banner.

- **Splash Spin Lite**
  This banner is presented with a brief splash screen. After the splash screen animation ended, the banner shrinks into a spin lite banner.

- **Reco**
  This banner is presented as a full-width carousel of cards and rounded thumbnails. The cards will automatically advance to the next card without intervention. User can swipe, or tap on the card of interest to expand the card.

IMPORTANT:

- All Engage ads served are for portrait orientation only.

- External links in banner ads will open in the embedded browser provided by the SDK for a less disruptive experience. The embedded browser includes the user to open the link in an external web browser.

Video Interstitial

The video interstitial unit is a fullscreen unit that presents an auto-played video typically lasting 15 to 30 seconds long. Once the video is played to completion, a button will appear allowing the user to learn more about the advertiser. Video units may be skippable or non-skippable or non-skippable only for a few seconds.

Native Ads

Unlike typical banner ad formats, native ads do not have have a fixed layout, components of the ad are returned for you to present the ad in a way that blends into the the look and feel of your app.

The components returned to you are the ad's title, description, advertiser's brand name, a thumbnail image and a larger portrait or landscape image.

Native Video Ads

Native Video ads are almost identical to native ads except you'll get a single video instead of images. As with native ads, you'll be able to present the ad with your app's look and feel as well.

If you need any assistance with choosing the right formats to implement in your app, feel free to contact your account manager at any time.

Adding the SDK to your App

1. Open your project in Android Studio
2. Select **File** -> **New** -> **New Module…**
3. Choose “**Import .JAR / .AAR Package**” from the popup window
4. Locate the Innity Mobile Ads AAR file and complete the import
5. Select **File** -> **Project Structure**
6. Highlight “**app**” under “**Modules**” section.
7. Go to “**Dependencies**” tab.
8. Select “**+**” to add a Module dependency
9. Select the Innity Mobile Ads AAR module added in the steps above
10. Add the following permission to your project:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Implementing Targeted Ads based on User’s Installed Apps

The Innity Advenue server has the ability to target ads based on the apps installed by users. This enables advertisers to utilise their marketing budgets more effectively by showing ads to users who are more relevant.

By calling the trackInstalledApps() function once when your app is launched, it enables our SDK to provide the Innity Advenue server with a list of user-installed apps.

```
@Override
public void onCreate()
{
       super.onCreate();
       imAdSdk = new IMAdSdkBuilder(this).build();
       imAdSdk.trackInstalledApps(this);
}
```

This method must be called during app launch, to track the installed apps on the user’s mobile.

Implementing Engage Ads

Here are the steps to add a banner to your app:

1. Create an instance of the IMAd SDK Core.

2. Create an instance of the IMAd Engage ad format and set the **publisher ID** assigned to you by Innity as well as any targeting properties your app is able to provide, for example, the user's gender and age.

3. Load a banner from a designated zone (Zone IDs are issued by Innity, the IDs are created according to how you intent to target ads. You may want to have a zone for each category of content your app provides, for example, any content related to Fashion may be considered to be in the same zone and hence share the same zone ID.)

4. Once you receive the notification that an ad is available and has been loaded, add the banner ad's view to your layout.

   There are various events that fire such as when the banner expands or closes that you may want to handle. If your app is playing a video, you may want to pause the video when the ad is expanded to cover the full screen.

5. Finally, you'll want to clean up/destroy the ad when you're done using it.

IMPORTANT:

All Engage format ads served are for portrait orientation only. It is also your responsibility to ensure that the banner's view is the top-most view in your hierarchy.

Here is some sample code to do the above:

1. Create an IMAd SDK Core Object

```
import com.appsploration.imadsdk.core.sdk.IMAdSdk;
import com.appsploration.imadsdk.core.sdk.IMAdSdkBuilder;

import com.appsploration.imadsdk.engage.IMAdEngage;

imAdSdk = new IMAdSdkBuilder(this).build();
```

2. Create an IMAdEngage ad and set targeting

Once you've created an IMAdEngage object, we recommend that you set targeting properties if you know your user. There are a list of standard properties provided, please see the **Audience Targeting Properties** table below, however, if there are certain properties you can provide which are useful for targeting, you may also set your own custom key, value pairs.

```
IMAdEngage engage = IMAdEngage.with(getApplication().getImAdSdk(), YOUR_PUBLISHER_ID);

TargetProperties properties = new TargetProperties();

properties.setProperty(TargetProperties.IMATargetPropKey_Gender, "M");

properties.setProperty(TargetProperties.IMATargetPropKey_Age, "21");

properties.setProperty(TargetProperties.IMATargetPropKey_Language, "EN");
```

3. Load a banner

Every banner needs to be loaded from the server with a Zone ID. The Zone IDs are allocated for your app by Innity. There are 6 zone IDs provided for the purpose of testing, each zone is for one of the 6 Engage Ad formats. *See table above for Test Zone IDs.*

You'll also need to handle banner load events and other events related to the banner ad's creative. Specify the object to handle the load events via the 3rd argument in **engage.load()** and implement the mandatory **adReady** & **adFailed** methods.

Also specify the object to handle the banner ad's creative events via the 4th argument in **engage.load()** and implement the various methods as required. Take a look at the "**Handling Engage Ad Banner Events**" section below for an explanation of what these events are and what they do.

NOTE: The test zone IDs are **alphabetical** while the live zone IDs are **numerical**. You would still pass the zone ID **as a string**, e.g. zoneId: "8234"

```
//engage.load(Activity activity, String id, TargetProperties targetProperties,
//             AdExecutor.AdLoadCallback callback, AdEventCallback eventListener)

engage.load(this, "interstitial", properties, this, this);
```

Once a banner has successfully loaded, the **adReady** delegate method will be called. This delegate method will only be called if there is an ad available for display in the requested zone.

The **adFailed** delegate method will be called whenever there is no ad available or a timeout occurred during loading. The timeout is set to 30 seconds.

4. Handling Banner Load Events

There are various banner events that the ad unit may trigger, at minimum, you'll need to handle **adReady** - when the ad has loaded and is ready for you to ad to your view and **adFailed** when the ad fails to load.

```
@Override
```

```
public void adReady(AdUnit adUnit)
{
   banner = (EngageAd) adUnit;

   ViewGroup container = (ViewGroup) findViewById(R.id.entryType);
   container.addView(banner.getView());
   banner.show();
}

@Override
public void adFailed(String id, Exception reason)
{
   // failed
}
```

5. Destroying the AdUnit

Remember to cleanup once you're done.

```
banner.destroy();
banner = null;
```

Zones

The following is a table of the test zone IDs. Remember that in a live environment, the zone IDs are numerical. Your Innity account manager will provide you with the live zone IDs.

Test Zone ID	Format
spin_lite	**Spin Lite** 9:4 (no expansion)
spin	**Spin** 9:4 expands to fullscreen
spin_plus	**Spin+** 9:4 expands to fullscreen
revolver	**Revolver** 9:4 expands to fullscreen
pull	**Pull** 320px x 50px expands to fullscreen
interstitial	**Interstitial** Fullscreen (no pre-expansion banner)
mobile_leaderboard	**Mobile Leaderboard** 320px x 50px (no expansion)
medium_rectangle	**Medium Rectangle** 300px x 250px (no expansion)
medium_rectangle_cube	**Medium Rectangle Cube** 300px x 250px (no expansion)
medium_rectangle_slider	**Medium Rectangle Slider** 300px x 250px (no expansion)
mobile_cards	**Mobile Cards** 9:4 expands to fullscreen
splash_cards	**Splash Cards** 9:4 expands to fullscreen
splash_spin_lite	**Splash Spin** 9:4 expands to fullscreen
reco	**Reco** Full-width x 405px (no expansion)

IMPORTANT:

Some ad formats such as the Spin family of formats may conflict with the gestures you wish to recognise in your app. The gestured required to spin the creative can conflict with the horizontal swipe gesture used to switch between tabs in a Tab Bar activity.

We recommend you take some time to understand how the gestures required for each format before implementing them in your app.

Premium Ad Zones

Premium ads are reserved for ads that have have a higher priority over standard ad zones. If you have been given premium ad zone IDs, please add the premium ad zone when requesting for ads. The SDK will prioritise premium ad zones when there are premium campaigns available.

If you have not received any premium ad zone IDs, please check with your account manager if you are eligible for premium ad zones.

The zone can be added with the following code:

```
IMAdEngage engage = IMAdEngage.with(getApplication().getImAdSdk(), YOUR_PUBLISHER_ID);

engage.setPremiumZone(YOUR_PREMIUM_ZONE_ID);

engage.load(this, YOUR_NORMAL_ZONE_ID, properties, this, this);
```

Handling Engage Ad Banner Events

In addition to the mandatory `adDidLoad` and `adDidFailLoad`, optional delegates are available to better manage your app. They are:

- `public void adDidBeginPreview(EngageAd ads)`

  When a **Pull** banner is tapped on, the ad is expanded upwards slightly to provide the user with a preview of the ad. The adDidBeginPreview delegate is called just before the banner expands slightly. If your app includes some animation or music, you may utilise this delegate to pause the animation or music when the preview begins. Also see `adDidEndPreview`.

- `public void adDidEndPreview(EngageAd ads)`

  This delegate is called when the preview of a **Pull** banner ends, that is, when the banner returns to it's original position after expanding slightly. Also see `adDidBeginPreview`

- `public void adDidExpand(EngageAd ads)`

  The delegate is called just after the the banner ad has expanded fully, for example when a **Pull, Spin** or **Revolver** banner is expanded. This delegate is also called when an **Interstitial** banner format appears.

- `public void adDidCancelExpand(EngageAd ads)`

  The delegate is called after an expandable ad returns to its initial position but **did not** fully expand. This is only applicable to the **Pull** banner.

- `public void adDidCloseExpanded(EngageAd ads)`

  The delegate is called after an expandable ad returns to its initial position from its **fully expanded** state. This delegate is also called when the user closes a **Pull**, **Interstitial, Spin** or **Revolver** banner.

- `func adDidUnload(EngageAd)`

  The delegate is called after **Spin Lite**, **Spin**, **Spin+** or **Revolver** is closed. The developer must remove the ad from the view hierarchy manually.

- `public void adActionWillBegin(EngageAd ads)`

  The delegate is called just before a the calendar event UI is invoked (some ads may have a button for the user to add a reminder to an event such as a concert date and time) and just before an external link is opened with the SDK's embedded web browser.

- `public void adActionDidEnd(EngageAd ads)`

  The delegate is called just after the calendar event UI has been dismissed.

- `public void adActionWillLeaveApplication(EngageAd ads)`

  The delegate is called just before the "do you want make a call" confirmation dialog appears or before a messaging app is invoked to compose an SMS.

- `public void adWasClicked(EngageAd ads)`

  The delegate is called when the user “clicks” on the ad and triggers a web browser to open the landing page.

Implementing Video Interstitial Ads

Here are the steps to add a banner to your app:

1. Create an instance of the IMAd SDK Core.
2. Create an instance of the IMAd Video Interstitial ad format and set any targeting properties your app is able to provide, for example, the user's gender and age.
3. Load a banner from a designated zone (Zone IDs are issued by Innity, the IDs are created according to how you intent to target ads. You may want to have a zone for each category of content your app provides, for example, any content related to Fashion may be considered to be in the same zone and hence share the same zone ID.)
4. Once you receive the notification that an ad is available and has been loaded, show the interstitial video when you're ready.
5. Finally, you'll want to clean up/destroy the ad when you're done using it.

1. Create an IMAd SDK Core Object

```
import com.appsploration.imadsdk.core.sdk.IMAdSdk;
import com.appsploration.imadsdk.core.sdk.IMAdSdkBuilder;

import com.appsploration.imadsdk.video_interstitial.IMAdVideoInterstitial;

imAdSdk = new IMAdSdkBuilder(this).build();
```

2. Create an IMAdVideoInterstitial ad and set targeting

Once you've created an IMAdVideoInterstitial object, we recommend that you set targeting properties if you know your user. There are a list of standard properties provided, please see the **Audience Targeting Properties** table below, however, if there are certain properties you can provide which are useful for targeting, you may also set your own custom key, value pairs.

```
IMAdVideoInterstitial video = IMAdVideoInterstitial.with(getApplication().getImAdSdk());

TargetProperties properties = new TargetProperties();

properties.setProperty(TargetProperties.IMATargetPropKey_Gender, "M");

properties.setProperty(TargetProperties.IMATargetPropKey_Age, "21");

properties.setProperty(TargetProperties.IMATargetPropKey_Language, "EN");
```

3. Load a banner

Every banner needs to be loaded from the server with a Zone ID. The Zone IDs are allocated for your app by Innity.

NOTE: The test zone IDs are **alphabetical** while the live zone IDs are **numerical**. You would still pass the zone ID **as a string**, e.g. zoneId: “8234”

```
video.load("video_interstitial", properties, this);
```

Once a banner has successfully loaded, the **adReady** delegate method will be called. This delegate method will only be called if there is an ad available for display in the requested zone.

The **adFailed** delegate method will be called whenever there is no ad available or a timeout occurred during loading. The timeout is set to 30 seconds.

4. Handling Banner Load Events

There are various banner events that the ad unit may trigger, at minimum, you’ll need to handle **adReady** - when the ad has loaded and is ready for you to ad to your view and **adFailed** when the ad fails to load.

```
@Override
public void adReady(AdUnit adUnit)
{
    this.adUnit = (VideoInterstitialAd) adUnit;
    this.adUnit.show(this);
}

@Override
public void adFailed(String id, Exception reason)
{
   // failed
}
```

5. Destroying the AdUnit

Remember to cleanup once you’re done.

```
adUnit.destroy();
adUnit = null;
```

Implementing Native Ads

Native Ads allow you to present ads according to the look and feel of your app. The SDK will return the following data to you for you to present in your app as you wish:

1. Title
2. Description
3. Advertiser's Brand Name
4. A Square Thumbnail Image
5. A Portrait Image
6. A Landscape Image

Once you present the ad, simply call the **displayed** method when the ad becomes visible the user and call the **clicked** method when the ad is tapped on.

The process is still very similar to Engage ads and Video Interstitial:

1. Create an instance of the IMAd SDK Core.

2. Create an instance of the IMAd Native Ad format and set any targeting properties your app is able to provide, for example, the user's gender and age.

3. Load an ad from a designated zone (Zone IDs are issued by Innity, the IDs are created according to how you intent to target ads. You may want to have a zone for each category of content your app provides, for example, any content related to Fashion may be considered to be in the same zone and hence share the same zone ID.)

4. Once you receive the notification that an ad is available and has been loaded, present the ad in your own style when you're ready and call the **displayed** method when the ad becomes visible to the user.

5. Respond to a click/tap by calling the **clicked** method.

6. Finally, you'll want to clean up/destroy the ad when you're done using it.

1. Create an IMAd SDK Core Object

```
import com.appsploration.imadsdk.core.sdk.IMAdSdk;
import com.appsploration.imadsdk.core.sdk.IMAdSdkBuilder;

import com.appsploration.imadsdk.native_ad.IMAdNative;

imAdSdk = new IMAdSdkBuilder(this).build();
```

2. Create an IMAdNative ad and set targeting

Once you've created an IMAdNative object, we recommend that you set targeting properties if you know your user. There are a list of standard properties provided, please see the **Audience Targeting Properties** table below, however, if there are certain properties you can provide which are useful for targeting, you may also set your own custom key, value pairs.

```
IMAdNative nativeAd = IMAdNative.with(getApplication().getImAdSdk());

TargetProperties properties = new TargetProperties();

properties.setProperty(TargetProperties.IMATargetPropKey_Gender, "M");

properties.setProperty(TargetProperties.IMATargetPropKey_Age, "21");

properties.setProperty(TargetProperties.IMATargetPropKey_Language, "EN");
```

3. Load a banner

Every banner needs to be loaded from the server with a Zone ID. The Zone IDs are allocated for your app by Innity.

NOTE: The test zone IDs are **alphabetical** while the live zone IDs are **numerical**. You would still pass the zone ID **as a string**, e.g. zoneId: “8234”

```
nativeAd.load(“native", properties, this);
```

Once a banner has successfully loaded, the **adReady** delegate method will be called. This delegate method will only be called if there is an ad available for display in the requested zone.

The **adFailed** delegate method will be called whenever there is no ad available or a timeout occurred during loading. The timeout is set to 30 seconds.

4. Handling Banner Load Events

There are various banner events that the ad unit may trigger, at minimum, you’ll need to handle **adReady** - when the ad has loaded and is ready for you to ad to your view and **adFailed** when the ad fails to load.

```
@Override
public void adReady(AdUnit adUnit)
{
       this.adUnit = (NativeAd) adUnit;

       // Present the ad
       viewHolder.title.setText(adUnit.getTitle());
       viewHolder.description.setText(adUnit.getDescription());
       viewHolder.icon.setImageBitmap(adUnit.getThumbnail());

       // Notify the SDK that the ad unit was displayed
       this.adUnit.displayed();
}

@Override
public void adFailed(String id, Exception reason)
{
   // failed
}
```

5. Respond to a Click/Tap

Once you have presented the ad, it is your responsibility to respond to the users' tap on the ad by calling the **clicked** method. This method will invoke an embedded web browser to display the advertiser's target URL.

```
adUnit.clicked(activity);
```

6. Destroying the IMAdNative and AdUnit

Remember to cleanup once you're done.

```
adUnit.destroy();
adUnit = null;
```

Implementing Native Video Ads

Native Video Ads are almost identical to Native Ads. You can present ads according to the look and feel of your app however there are no static images for Native Video Ads.

The SDK will return the follow data to you for you to present in your app as you wish:

1. Title
2. Description
3. Advertiser's Brand Name

You'll also have to call the **displayed** method once the ad is visible within your app and call the **clicked** method when the ad is tapped on.

Once again, the process is very similar to Native Ads:

1. Create an instance of the IMAd SDK Core.

2. Create an instance of the IMAd Native Video Ad format and set any targeting properties your app is able to provide, for example, the user's gender and age.

3. Load an ad from a designated zone (Zone IDs are issued by Innity, the IDs are created according to how you intent to target ads. You may want to have a zone for each category of content your app provides, for example, any content related to Fashion may be considered to be in the same zone and hence share the same zone ID.)

4. Once you receive the notification that an ad is available and has been loaded, present the ad in your own style when you're ready. Also invoke the **displayed** method when the ad comes into view and play the video.

5. Respond to a click/tap and call the **clicked** method.

6. Finally, you'll want to clean up/destroy the ad when you're done using it.

1. Create an IMAd SDK Core Object

```
import com.appsploration.imadsdk.core.sdk.IMAdSdk;
import com.appsploration.imadsdk.core.sdk.IMAdSdkBuilder;

import com.appsploration.imadsdk.native_video.IMAdNativeVideo;

imAdSdk = new IMAdSdkBuilder(this).build();
```

2. Create an IMAdNativeVideo ad and set targeting

Once you've created an IMAdNativeVideo object, we recommend that you set targeting properties if you know your user. There are a list of standard properties provided, please see the **Audience Targeting Properties** table below, however, if there are certain properties you can provide which are useful for targeting, you may also set your own custom key, value pairs.

```
IMAdNativeVideo nativeAd = IMAdNativeVideo.with(getApplication().getImAdSdk());

TargetProperties properties = new TargetProperties();

properties.setProperty(TargetProperties.IMATargetPropKey_Gender, "M");

properties.setProperty(TargetProperties.IMATargetPropKey_Age, "21");

properties.setProperty(TargetProperties.IMATargetPropKey_Language, "EN");
```

3. Load a banner

Every banner needs to be loaded from the server with a Zone ID. The Zone IDs are allocated for your app by Innity.

NOTE: The test zone IDs are **alphabetical** while the live zone IDs are **numerical**. You would still pass the zone ID **as a string**, e.g. zoneId: "8234"

```
nativeAd.load("native_video", properties, this);
```

Once a banner has successfully loaded, the **adReady** delegate method will be called. This delegate method will only be called if there is an ad available for display in the requested zone.

The **adFailed** delegate method will be called whenever there is no ad available or a timeout occurred during loading. The timeout is set to 30 seconds.

4. Handling Banner Load Events

There are various banner events that the ad unit may trigger, at minimum, you'll need to handle **adReady** - when the ad has loaded and is ready for you to ad to your view and **adFailed** when the ad fails to load.

```
@Override
public void adReady(AdUnit adUnit)
{
   this.adUnit = (NativeVideoAd) adUnit;
```

```
    // Present the ad
    viewHolder.title.setText(adUnit.getTitle());
    viewHolder.description.setText(adUnit.getDescription());
    adUnit.setVideoView(viewHolder.video); // set up the view to hold the video

    // Notify the SDK that the ad unit was displayed
    adUnit.displayed();
}

@Override
public void adFailed(String id, Exception reason)
{
    // failed
}
```

5. Respond to a Click/Tap

Once you have presented the ad, it is your responsibility to respond to the users' tap on the ad by calling the **clicked** method. This method will invoke an embedded web browser to display the advertiser's target URL.

```
adUnit.clicked(activity);
```

6. Destroying the AdUnit

Remember to cleanup once you're done.

```
adUnit.destroy();
adUnit = null;
```

Implementing Alternate Ads

It is recommended that you attempt to load an ad through the Innity Mobile Ads SDK first for the best premium rates before invoking other mobile ad SDKs.

If the Innity Mobile Ads SDK does not return an ad, you may proceed to display potential ads from other mobile ad SDKs.

The general process to implement alternate ads is as follows:

1. Load Innity Ads as usual
2. Create a function to load alternate ads (e.g. from Google Mobile Ads, AdMob, MoPub…etc)
3. Listen to the **adFailed** callback and call the function above if the callback is triggered
4. You may also create a timeout after **loadBanner** to call the above alternate ad loading function if both **adFailed** and **adReady** are not triggered within your preferred timeframe. Remember to cancel the timeout inside the **adReady** function.

Sample Code

```
@Override
protected void onCreate(Bundle savedInstanceState)
{

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_interstitial_ad_test);

    // Create IMAd Object with Activity Context
    timer = new Timer();
    String zoneId = getIntent().getStringExtra(KEY_AD_TYPE_STRING);

    TargetProperties properties = new TargetProperties();
    properties.setProperty(TargetProperties.IMATargetPropKey_Gender, "M");
    properties.setProperty(TargetProperties.IMATargetPropKey_Age, "21");
    properties.setProperty(TargetProperties.IMATargetPropKey_Language, "EN");

    // Load the banner
    IMAdEngage engage = IMAdEngage.with(((TestIMAdApplication)
                                getApplication()).getImAdSdk());
    engage.load(this, zoneId, properties, this, this);

    // set a timeout
    timer.schedule(new TimerTask()
    {
        @Override
        public void run()
        {
            loadAlternateAd();
        }
    }, 3000);
}

private void loadAlternateAd()
{
    // Your code to load an alternate ad here
    // load AdMob, MoPub, Google Ads here
}

@Override
public void adReady(AdUnit adUnit)
{
    // cancel the timeout
    timer.cancel();
```

```
}
    // Ad is loaded, no need to load alternate ad
    banner = (EngageAd) adUnit;

    ViewGroup container = (ViewGroup) findViewById(R.id.entryType);
    container.addView(banner.getView());
    banner.show();
}

@Override
public void adFailed(String id, Exception reason)
{
    // Cancel the timeout
    timer.cancel();

    // Call to load alternate ad
    loadAlternateAd();
}
```

Rate Limit for Loading Ads

There is a rate limit applied on ad loading. An app cannot request more than one ad per zone within 30 seconds.

, If an ad is requested from the same zone within 30 seconds, the **adFailed** callback will be fired with a **RateLimitException** object.

Example App

An example app has been included in this SDK. Explore the **IMAdSDKDemo** to see how the an app may create and display the available banner formats.

Right Audience, Right Message

The Innity Advenue server is capable of targeting audiences and serving banner ads with *dynamic creatives* to provide the most relevant ads and messaging.

With audience targeting, the Innity Advenue ad server can determine which ads to serve depending on your user's profile.

Audience Targeting Properties

The following table shows a list of standard properties provided for your use.

System Property Key	Key	Values
IMATargetPropKey_Gender	gender	Single character, for example: "**m**" or "**f**"
IMATargetPropKey_Age	age	Numeric string, for example: "**26**"
IMATargetPropKey_Latitude	lat	Numeric string, for example: "**37.0625**"
IMATargetPropKey_Longitude	lon	Numeric string, for example: "**-95.677068**"
IMATargetPropKey_Birthday	birthday	YYYY-MM-DD format, for example: "**1988-12-31**"
IMATargetPropKey_Language	lang	ISO-639-1 language code (see: http://www.loc.gov/standards/iso639-2/php/code_list.php), for example: "**en**"

Support & Feedback

If you encounter any technical issues implementing the SDK, please contact your account manager, we would be more than happy to assist you.

We would also love to hear your feedback on your experience or on the features of the SDK, please do let us know so we can improve.