Zapr SDK

Developer Guide Document

SDK Version 3.3.0

18 Nov 2018

Overview

Integrate ZAPR's SDK to drive brand ads and move away from low value app install ads on your inventory, go beyond generic mobile advertising to deliver high margin TV ad spends.

ZAPR's TV audience data now allows brands to differentiate your app users as personas they understand – from IPL fans to soccer fanatics, action movie buffs to rom-com lovers, news addicts to Masterchef patrons, Punjabi music lovers to WWE crazies.

Setup

Download / Install Zapr SDK

Zapr SDK is available as an AAR via jCenter repository. To add the `zapr-sdk` dependency, open `build.gradle` file of your application module and add `zapr-sdk` dependency as follows.

```
dependencies {
   compile('com.redbricklane.zapr:zapr-sdk:3.3.0@aar') {transitive = true;}
}
```

Requirements

Minimum SDK Version : Android 4.0 (API level 14)

Recommended target SDK version : Android 7.0 (API level 24)

Gradle Plugin version : 2.2.0 (or later)

Gradle wrapper version : 2.14.1 (or later)

Android Studio : 2.2

Google Play Services : **12.0.1**

Dependencies

Zapr SDK requires Google Play Services ads, identity and location API's as dependency.

These API's are used for retrieving device location and Advertising ID which are required for ad serving.

User can either add complete google play services library dependency or add individual three modules.

Either add complete Google play services

```
compile 'com.google.android.gms:play-services:12.0.1'
```

Or, add individual module dependencies (recommended)

```
For Google Play Services version 12.0.1

compile 'com.google.android.gms:play-services-basement-12.0.1

For above Google Play Services 12.0.1

compile 'com.google.android.gms:play-services-ads:12.xx:xx
```

Also add the Android support library appcompat

```
compile 'com.android.support:appcompat-v7:24.2.0' (or later)
```

For more information on setting Google play services and Android support libraries, please refer following links:

- Google Play Services
- Support libraries

Manifest file changes

Since the introduction of Gradle based build system in Android, Manifest file settings from Android library project's AAR file are automatically merged into your application's APK.

So you are not required to make any changes in your application's manifest file. All required permissions, Activities and Receiver entries will be automatically imported from zapr-sdk AAR file into your application build.

Please refer to Appendix for list of permissions, Activities and receivers that zapr-sdk uses.

For more details on Manifest merging and Manifest file conflict resolution, refer to this Android developer documentation.

Integration via mediation

If you are integrating Zapr SDK via mediation (Admob, DFP, Mopub)

Refer the below mentioned documents

Admob	Link
DFP	Link
Mopub	Link

Change Log

v3.3.0

- ANR fixes
- Bug fixes

v3.2.2

- ANR fixes
- Bug fixes

v3.2.1

- Added country in Registration and UpdateDetailsService
- Minor bug fixes

v3.2.0

- Removed Google Play Services Location dependency
- Added https secure connection for all requests
- Removed play-services-ads dependency and added play-services-basement till google-play-services version 12.0.1
- Minor bug fixes

v3.1.0

- Optimised logging mechanism
- Minor bug fixes

v3.0.0

- Added API to stop Data SDK
- Minor optimization and bug fixes

v2.2.4.1

- Minor Optimization and bug Fixes for Oreo

v2.2.4

- Optimization for Oreo

v2.2.2

- Fixed memory leak issue in Banner Ads

v2.2.1

- Fix for rare NPE crash in Banner Ads.

v2.2.0

- Added support for Android Oreo (8.0).
- Data SDK optimizations to use JobScheduler

v2.1.1

- Rewarded videos feature added.

v1.13.3

- Minor bug fix in Data SDK. Added some delay before starting Zapr service.

v1.13.2

- Added functionality for debug and crash events logging.
- Added functionality to run Zapr service for lower priority app if higher priority app does not have audio permission.

v1.12.1

- Minor code optimisation

v1.12.0

- Minor bug fixes
- Updated Native Code in Data SDK for better matching
- Added remote config feature in Ads SDK

v1.11.5

- Fix for frequent ad request
- Added Optional AdStatusListener interface
- Disable Zapr vast ad player feature is added
- Minor bug fixes

v1.11.4

- Fix for Data SDK rare crash issue

v1.11.3

- Video ad caching optimizations
- Fix for Video SDK close button issue

v1.11.2

- Bumped Google Play services dependency version from 9.4.0 to 9.8.0
- Internal SDK logger optimization
- Minor bug fixes

v1.11.1

- Major optimizations in Data SDK Native fingerprinting code for faster operation
- Native code libzaprdatajni.so file is updated
- Added Analytics SDK as optional module

v1.11

- Added VAST wrapper support in Zapr Video Ad SDK
- Added video ad cache holding functionality
- **Video ad listener callbacks are changed**
- Optimizations in Zapr Data SDK. Updated native code in data SDK

v1.9.9

- Added support for Android Nougat (API level 24)
- Upgraded dependency versions
    - Google Play services to v9.4.0

    - Support library appcompat v7:24.2.0
- Few optimizations in Zapr Data SDK
- Minor bug fixes in Video SDK caching mechanism

v 1.9.8

- Optimization in Video ad pre caching and cache eviction mechanism
- Video ad cache will now be maintained across multiple launches of same application to save user's data usage
- Added screen wake lock while playing video ad
- Optimization in device location detection functionality. If location information is not available using Play services location API, then Android legacy location API will be invoked as a fallback
- Added extra parameter "loc_age" in "device/geo/ext" ad request Json. It will pass the age of location fix in seconds. This can be utilized by adserver to ignore stale locations
- Optimization in Advertising ID (IFA) passing. If IFA is not available from cache in first request, then SDK will make a fallback call to fetch the Advertising ID while making ad request
- Passing SHA1 hashed Android ID as "dpidsha1" in OpenRTB ad request.
- Added functionality to change background color of full screen Interstitial ads. The background color can also be set to transparent.

  Eg: `zaprInterstitialAd.setInterstitialAdBackgroundColor(0xCCFFFFFF); // 0xAARRGGBB`
- Added functionality to pass device "orientation" in "imp/banner/ext" ad request Json for interstitial ads. (orientation : 1=portrait, 2=landscape)

Upgrading from Older SDK

Upgrading from Zapr SDK version 2.1.1 (or older)

Changes in Data SDK

- Added support for Android Oreo in Data SDK. JobScheduler services are added to SDK. If you are manually integrating Zapr SDK, please ensure that all manifest entries are updated as per latest version. If you are integrating via Jcenter dependency, then manifest entries will be automatically updated.

Upgrading from Zapr SDK version 1.13.1 (or older)

Changes in Ads SDK

- Added Zapr's own vast player. So if you are manually integrating Zapr SDK, please ensure that all manifest entries are updated as per latest version. If you are integrating via Jcenter dependency, then manifest entries will be automatically updated.

Upgrading from Zapr SDK version 1.11.5 (or older)

Changes in Data SDK

- Native code files (libzaprdatajni.so) files in Data SDK are changed. If you are doing manual integration of SDK, please replace old .so files in jniLibs folder of your project with new ones.

Upgrading from Zapr SDK version 1.11.4 (or older)

Changes in Ads SDK

- Frequent Ad requests will now be blocked.
- Unless previous Video/Interstitial ad is shown, new ad request will not be made
- Before calling show ad, you can check ad status by calling `isInterstitialAdLoaded()` / `isVideoAdLoaded()` method.

Upgrading from Zapr SDK version 1.11 (or older)

Changes in Dependencies

- Google Play Services version is changed to 9.8.0

Upgrading from Zapr SDK version 1.9.9 (or older)

Changes in Zapr Data SDK

- New Native shared object library files are added (libzaprdatajni.so)
- Old libzaprscjni.so files are removed

If you are upgrading from old Zapr SDK versions (1.9.9 or old), then add updated dependency version in build.gradle and perform a gradle clean build.
This will remove old .so files and add new files to your project.

- Add ABI filter in your app build.gradle

Zapr Data SDK contains .so files for all ABI's. To reduce the final APK size, you can add ABI filter in your app modules build.gradle. Refer Data SDK Integration section for more details.

Changes in Zapr Video SDK

- Added video ad cache hold functionality in Video SDK.

Due to which the **ZaprVideoAdEventListener callbacks are changed**. Please implement new callback method **onResponseReceived()**.
If you are not using video cache hold functionality, then you can keep this method empty.

Dependency Changes

We have upgraded the dependencies used by Zapr SDK.

- Updated Google play services version to 9.4.0
- Updated Android support library appcompat to version v7:24.2.0

If you are upgrading your app to use new Zapr SDK version from older one, please update the dependencies versions as mentioned above.

It is highly recommended to refresh all your dependencies after you have upgraded the version in build.gradle file. To refresh dependencies, first upgrade dependency version in build.gradle file and run gradle build with --refresh-dependencies flag.

```
// Navigate to your project's root directory and build using command:

$ gradle build --refresh-dependencies
```

Integrating Zapr Data SDK

Setup

Add ABI filter

Zapr Data SDK uses native code (.so) files and ships with native code shared object library files (libzaprdatajni.so) for following ABI's:

- armeabi
- armeabi-v7a
- arm64-v8a
- x86
- x86_64
- mips

As you might not need to support all ABI's in your app, please add ABI filter in your app modules build.gradle file.

To add ABI filter, open build.gradle file for your app module and add ndk abiFilters in android-> defaultConfig section.
Eg: If you need to support only armeabi-v7a and x86 architectures, add abiFilters only for these two ABI's as shown below:

```
// app module's build.gradle file
android {
    defaultConfig {
        // Add following ndk->abiFilter config for supporting arm-v7a and x86 arch
        ndk {
            abiFilters "armeabi-v7a", "x86"
        }
    }
}
```

If you are not sure about which ABI's to support, please add default ABI support for x86 and arm-v7a arch as shown below.

```
// app module's build.gradle file
android {
    defaultConfig {
        // Add following ndk->abiFilter config for default arm-v7a and arm arch
        ndk {
            abiFilters "armeabi-v7a", "x86"
        }
    }
}
```

Note: *If you are using any other third party SDK in your app which also uses native code (.so files), then you should add filters for exactly those ABI's which are supported by third party libraries.*

Proguard Configuration

If you are minifying your apk build, you must add following lines in your application's proguard-rules.pro file. Adding following lines will ensure normal operation of Zapr SDK.

Add following lines in your apps proguard-rules.pro file which is located at app module's root folder.

```
// Append following lines at the end of proguard-rules.pro file
-keep class com.redbricklane.zaprSdkBase.** {
  public protected private *;
}
-keep class com.redbricklane.zapr.** {
  public protected private *;
}
```

Starting Zapr service

Once you have completed the Setup, you can start Zapr service just by calling `Zapr.start()` method.

After calling `start()` method, the Zapr services run in the background and periodically gathers necessary data. All Zapr services are designed to run in most optimized way without affecting device performance or battery consumption.

```
// Import Zapr
import com.redbricklane.zaprSdkBase.Zapr;

// In onCreate() method of your main Activity, call start method
public void onCreate(Bundle bundle){
    // Your activity initialization code ...
    // Start Zapr service. Pass activity context as parameter
    Zapr.start(this);
}
```

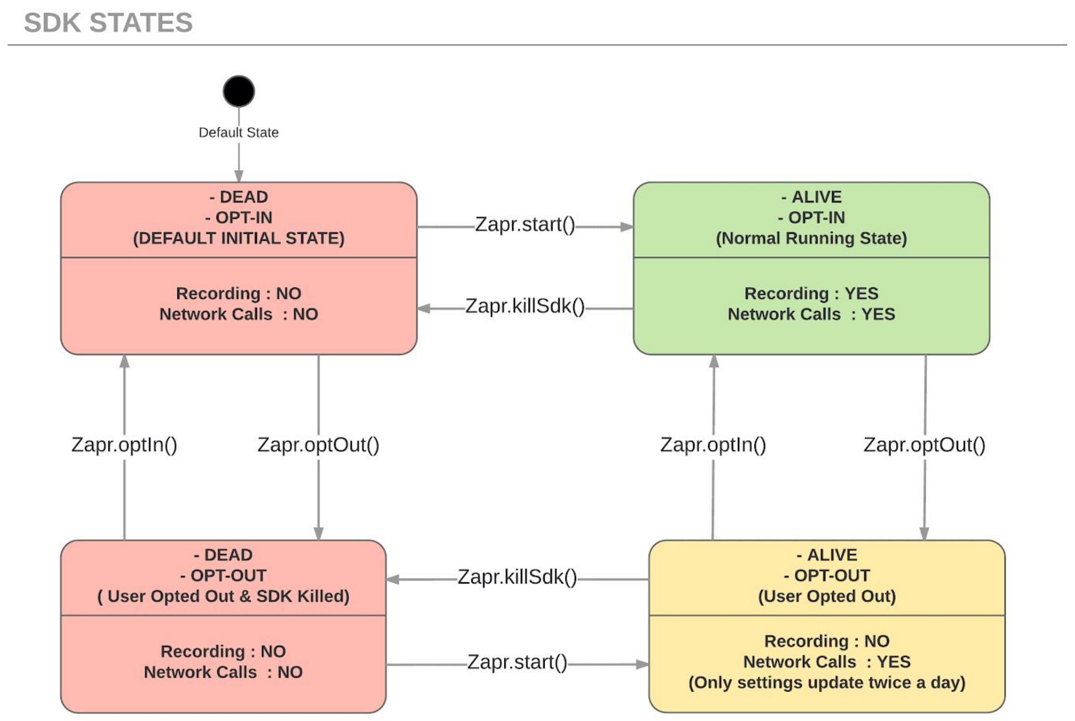
Stopping Zapr SDK completely

To stop/kill Zapr SDK completely, you can call `Zapr.killSdk()` API. Once `killSdk()` method is called, Zapr SDK will completely stop working, and will not collect any data, or will not make any network calls.

Note:
- Use `Zapr.killSdk()` API only if you want to stop SDK completely. In normal scenarios, you should not call this API.
- The default state of SDK is always dead/stopped. Until you call `Zapr.start()`

```
// Stop/kill Zapr SDK completely. Pass activity context as parameter
Zapr.killSdk(this);
```

Controlling SDK state remotely



If you wish to control SDK's start/stop state remotely, you can get a boolean flag from your config server (Eg Firebase Remote Config), and based on this flag value, you can either call `start()` or `killSdk()` APIs in SDK.

```
// Import Zapr Class
import com.redbricklane.zaprSdkBase.Zapr;

// In onCreate() method of your main Activity, call start method
public void onCreate(Bundle bundle){
    // Your activity initialization code …
    // Get a boolean flag value from your remote config
    boolean shouldZaprSDkRun = getFlagFromYourRemoteConfig();
    if(shouldZaprSDkRun) {
      // Start Transform service. Pass activity context as parameter
      Zapr.start(this);
    } else {
      // Stop/kill Zapr SDK. Pass activity context as parameter
      Zapr.killSdk(this);
    }
}
```

Opt-out of Zapr service (Optional)

Although Zapr services does not have any adverse effect on device performance or battery, still if user needs to opt out of Zapr services, you can use following API.

You can integrate this API into a toggle setting in the settings section of the application.

```
// To opt-out of Zapr services, call optOut() method and pass Activity context as
parameter.
Zapr.optOut(this);
```

To opt-in again to use Zapr services, you can use use `optIn()` API.

```
// To opt-in of Zapr services, call optIn() method and pass Activity context as
parameter.
Zapr.optIn(this);
```

Customization

Runtime permissions check for Android M+ devices

Zapr Data SDK needs various permissions for its normal operation. Since the introduction of runtime permissions in Android M version, user needs to grant these permissions at runtime.
Zapr Data SDK checks runtime permissions while starting Zapr services when `Zapr.start()` is called. And if permissions are not available, then Zapr Data SDK will request user to grant these permissions.

Disabling Runtime permissions check

Zapr Data SDK checks and requests runtime permissions while starting Zapr services when Zapr.start() is called.

If you wish to implement runtime permission request in your app yourself, then you can disable Zapr SDK's permission check and permission request functionality using following method. ques

Note that if you are disabling Zapr SDK's permission check, then you must ensure that necessary permissions are requested and granted before calling `Zapr.start()`.

```
// Disable Runtime permission request for Android M+ devices
Zapr.setRequestForPermissionsEnabled(false);
```

Note: *Make sure to call above method before calling* `Zapr.start()`.

Show Privacy policy dialog to user

Zapr Data SDK also provides optional functionality to show privacy policy dialog with your privacy policy to user when Zapr service is started. You can choose to add either a privacy policy web-page link or privacy policy text in the dialog.

You can enable Policy dialog using following method. Make sure to call this method before calling `Zapr.start()`.

Note that privacy policy dialog is disabled by default and Zapr service starts directly when `Zapr.start()` is called. But if enabled, then policy dialog will be show to user when `Zapr.start()` is called and Zapr service will start only if user accepts privacy policy.

1. Enable policy dialog with link

```
// Enable privacy policy dialog with web-page link
Zapr.enablePolicyDialogWithPolicyLink("http://<your-page-link>");
// NOTE : As per the new google guidelines, you should not use this API
// Start Zapr service
Zapr.start(this);
```

Note: *The webpage link for your privacy policy page must be a valid http or https link. If link is invalid, then Privacy policy dialog will be disabled.*

2. Enable policy dialog with policy text

```
// Enable privacy policy dialog with policy text
```

```
Zapr.enablePolicyDialogWithMessage("<your privacy policy text>");
// NOTE : As per the new google guidelines, you should not use this API
// Start Zapr service
Zapr.start(this);
```

***Note:** You can use basic html formatting for policy text.*
If policy text is null or empty, then Privacy policy dialog will be disabled.

Capture device IMEI number

***Important Note:** Google Play Developer Content Policy strongly recommends NOT to use IMEI number in the application without explicit consent of user.*
If you are enabling this functionality in Zapr SDK to capture IMEI number, then you must get explicit consent from the user in your app.

Zapr Data SDK also provides optional functional to capture device IMEI number. This functionality is disabled by default in SDK. however, to enable it, you can use enableImeiCapture() method. And add READ_PHONE_STATE permission in your application manifest file.

```
// Enable IMEI number capture
Zapr.enableImeiCapture(true);
```

***Note:** Make sure to call above method before calling `Zapr.start()`.*

Add READ_PHONE_STATE permission in manifest . This permission is required only if you need to capture IMEI number.

```
// Add following permission in manifest
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

APK Split

To optimize your apps final APK size, we strongly recommend you to use APK split functionality provided by gradle build system.

If you are supporting multiple ABIs in your app, then you can split your app APK per ABI and effectively reduce the APK size.
Please refer Android developer documentation for more details about ABI APK splits here :
https://developer.android.com/studio/build/configure-apk-splits.html#configure-abi-split

Integrating Banner Ads

Banner ads are usually rectangular graphic ad creatives that you can display in your application screen.

It takes following easy steps to add a Banner Ad unit to your application.

1. Define Zapr Banner Ad view in layout XML
2. Initialize and load Ad from your Activity
3. Auto reload banner ads
4. Banner Ad event listener
5. Destroying Banner Ad view
   Optional -
6. Passing user information
7. XML-Only initialization for Banner Ad
8. Banner Ad customization

1. Define Zapr Banner view in layout XML

Zapr SDK provides an Android custom View implementation `ZaprBannerAd`. To show Banner ad in your application Activity, add this view to your Activity layout file at required position.
Make sure to set necessary width and height to the view as per the ad unit you have created on Zapr portal. E.g. For displaying a `320x50` Banner ad, you can set `ZaprBannerAd` view's width as `match_parent` and height as `50dp`.

```
<!-- Other layout views... -->
<com.redbricklane.zapr.bannersdk.ZaprBannerAd
        android:id="@+id/zaprAdView"
        android:layout_width="match_parent"
        android:layout_height="50dp" />
<!-- Other layout views... -->
```

2. Initialize and load Banner Ad

Once you have defined the ZaprAdView in layout XML, declare an instance variable in your Activity and assign the same view to it using view id.

Set the Ad unit id received from Zapr portal, to your Banner ad instance.

Note*: Each Banner ad slot in your app should use different Ad unit id's.*

```
// Import ZaprBannerAd
import com.redbricklane.zapr.bannersdk.ZaprBannerAd;
...
// In your Activity, declare ZaprBannerAd instance
ZaprBannerAd mBannerAd;
...

// In onCreate() method...
mBannerAd = (ZaprBannerAd) findViewById(R.id.zaprAdView);
// Set Ad unit id
mBannerAd.setAdUnitId("<your-ad-unit-id>");
// Load Banner Ad
mBannerAd.loadAd();
```

Note:

1. *Calling loadAd() without setting mandatory parameter like adUnitId, will block the ad request and give* `onFailedToLoadBannerAd`*() callback.*
2. *Next banner ad request should be sent after 10 seconds.*
3. *Frequent ad request is blocked if previous ad request is already in process*
4. *In case of* `onFailedToLoadBannerAd`*() callback, next ad request should made after some time.*

3. Auto Reload Banner ads

Zapr Banner Ad SDK also provides mechanism to auto reload banner ads after specific time interval. To enable auto reloading of banner ads, set the ad refresh time as shown below.

```
// Set Ad refresh time (in seconds)
mBannerAd.setAdRefreshTime(15);
```

Note: Valid Ad refresh times are in the range of 10 sec to 300 sec.
Setting Ad refresh time to 0 disables auto reloading mechanism.
By Default, Banner ad auto refresh mechanism is disabled.

Auto retry after error

If Zapr SDK encounters any error during Ad request, then SDK stops banner Ad auto reload functionality and gives Banner Ad event listener's `onFailedToLoadBannerAd()` callback to application.
This gives application complete flexibility to take necessary action like hiding banner ad view.
You can disable this functionality and enable auto retry mechanism by setting `enableAutoRetryOnError(true)` as shown below.

```
// Enable auto retry mechanism after ad failed
mBannerAd.enableAutoRetryOnError(true);
```

4. Banner Ad event listener

Zapr Banner SDK also provides Ad event listener interface using which you can listen to Banner ad lifecycle events.

To listen to Banner ad lifecycle events, implement `ZaprBannerAdEventListener` and set this instance to ZaprBannerAdView.

```
// Implement ZaprBannerAdEventListener
public class YourActivity extends Activity implements
ZaprBannerAdEventListener{


// Implement listener methods
@Override
public void onBannerAdLoaded() {
    // Banner Ad Loaded
}


@Override
public void onBannerAdClicked() {
   // Banner Ad Clicked
}

@Override
public void onFailedToLoadBannerAd(int errorCode, String errorMessage) {
    // Banner Ad Failed.
    // You can take corrective measures here, like hiding Banner Ad view
}
    // Pass the instance of your Activity to Banner Ad view
    mBannerAd.setBannerAdEventListener(this);
}
```

5. Destroying Banner Ad view

It is very important to destroy instance of Banner Ad view when the hosting Activity or Fragment is destroyed. Destroying Banner Ad view instance clears up resources used Banner ads and also unregisters broadcast receivers and location receivers if used.
To destroy Banner ad view instance, call `destroy()` method on Banner ad instance from `onDestroy()` callback of your hosting Activity or Fragment.

```
// Destroy the instance of Banner Ad View
@Override
protected void onDestroy() {
  super.onDestroy();
  if (mBannerAd != null) {
    mBannerAd.destroy();
  }
}
```

6. Passing user information

You can optionally pass extra information in Banner ad for better ad targeting which fetches more eCPM and revenue.

```
// Import UserInfo
import com.redbricklane.zapr.basesdk.model.UserInfo;
```

```
// Pass User Information like gender and age (Example shown below)
UserInfo userInfo = new UserInfo();
userInfo.setAge(25);
userInfo.setGender("M");
// Set user info to banner ad view instance
mBannerAd.setUserInfo(userInfo);
```

7. XML-Only initialization for Banner Ad

If you don't need to do any any customization or pass any extra user information, you can also integrate Zapr Banner Ad view just by adding following XML block to your Activity or Fragment's layout. Using XML only initialization, you don't need to make any change in your Activity or Fragment class code.

```
<!-- Other layout views... -->
<com.redbricklane.zapr.bannersdk.ZaprBannerAd
    android:id="@+id/zaprAdView"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    adUnitId="<your-ad-unit-id>"
    autoLoad="true" />
<!-- Other layout views... -->
```

Note: *You must set **adUnitId** and **autoLoad="true"** in the XML. Without setting autoLoad="true", Banner ad will not load.*
Also note that using XML only initialization, you won't be able to listen to Banner Ad Event callbacks. So if you need to take any actions on Banner ad event callbacks, it is recommended to use normal integration as explained in steps 1-5 above.

8. Banner Ad customization

A. Runtime permissions check for Android M+ devices

Zapr Ad SDK needs permissions of Location and Phone state for detecting device information like location etc. Since the introduction of runtime permissions in Android M version, user needs to grant these permissions at runtime.
Zapr SDK checks runtime permissions when `loadAd()` is called. And if permissions are not available, then Zapr SDK will request user to grant these permissions.

If application wants to handle runtime permission request itself, and if application do not want Zapr SDK to ask for runtime permissions to the user, you can disable this feature by calling `setRequestForPermissionsEnabled(false)`.

```
// Disable request for runtime permissions (Not-Recommended)
mBannerAd.setRequestForPermissionsEnabled(false);
```

B. Location detection

Zapr SDK automatically fetches necessary information like device location etc, which is required for Ad serving. You can optionally disable this by using following API.

```
// Disable automatic location detection (Not-Recommended)
mBannerAd.setLocationDetectionEnabled(false);
```

***Note:** It is highly recommended to keep location detection enabled for Ad serving. Ad requests with proper device location and user information usually fetches higher eCPM and revenue.*

C. Use of In App Browser

On clicking any banner ad, the landing page opens in an In App Browser provided by Zapr SDK. If required, you can change this behaviour to open Ad's landing page in device default native web browser. Use following API to use native browser instead of In App Browser.

```
// If required, disable the use of In App Browser
mBannerAd.setUseInAppBrowser(false);
```

D. Serving Ads in test mode

During initial setup and testing phase, it is highly recommended to run the ads in test mode. Use the following API to enable test mode. In test mode, server may return test ads which will not be monetized.

```
// Enable test mode for ad serving
mBannerAd.setTestModeEnabled(true);
```

E. Enable Ad SDK logging

While integrating Zapr Ad SDK with your application, you may need to enable logging in SDK for debugging and testing purposes.
Various logging levels available are verbose, debug, info, warn, error and none.

Note: *Make sure to disable logging before releasing your app to Play store.*

```
// Enable Ad SDK logging at debug level
com.redbricklane.zapr.basesdk.Log.setLogLevel(LOG_LEVEL.debug);
```

Various logging levels available are verbose, debug, info, warn, error and none.

F. Set AdStatusListener

While integrating Zapr Banner Ads with your application, you may need to get the ad status related callbacks i.e. *adAlreadyLoaded()* and *adRequestBlocked(int seconds)*, then you have to set this AdStatusListener with your banner ad object.

```
// Set AdStatusListener
mBannerAd.setAdStatusListener(new AdStatusListener(){
      @Override
      public void adAlreadyLoaded() {
          // You will get this callback if ad is already loaded
      }
      @Override
      public void adRequestBlocked(int seconds) {
      // You will get this callback in case of frequent ad request
      // int seconds : This parameter will give you the time in seconds, after
      that next ad request will be sent to server
```

```
        }
};)
```

Integrating Interstitial Ads

Interstitial ads are full screen ads that cover the entire screen of your application. Interstitial ads are usually displayed at transition points of the application, such as between switching activities, during Viewpager switch or between levels in a game. The larger size of interstitials make them one of the most effective form of advertising.

Because of their larger size, interstitial ads require more bandwidth than traditional banners. To accommodate this, applications are required to load them asynchronously, in advance of when they will be displayed. This helps ensure that an ad is ready to be displayed when required.

Zapr SDK provides easy way to integrate and show full screen Interstitial ads in your application.

It takes following easy steps to add a Interstitial Ads to your application.

1. Initialize `ZaprInterstitialAd` in your application Activity
2. Load interstitial ad asynchronously in background
3. Listen to Interstitial ad callbacks
4. Show full screen Interstitial ad when ready
5. Set optional Ad request parameters

1. Initialize Zapr Interstitial Ad

Initialize `ZaprInterstitialAd` and set mandatory parameters like Ad Unit Id to this instance.

```
// Import ZaprInterstitialAd
import com.redbricklane.zapr.bannersdk.ZaprInterstitialAd;

// Initialize ZaprInterstitialAd in onCreate() method of your Activity
// Pass your activity context in the constructor
ZaprInterstitialAd mInterstitialAd = new ZaprInterstitialAd(context);
// Set Ad unit ID
mInterstitialAd.setAdUnitId("<ad-unit-id>");
```

Note: Calling loadInterstitialAd() without setting mandatory parameter like adUnitId, will block the ad request and give onFailedToLoadInterstitialAd() callback.

2. Load Interstitial Ad

To request Interstitial ad from server call `loadInterstitialAd()` method.

Note: Loading interstitial ad does not directly display the ad to user without calling `showInterstitialAd()` method as described below.

```
// Request for Interstitial ad from server
mInterstitialAd.loadInterstitialAd();
```

Note:

1. *Calling loadInterstitialAd() without setting mandatory parameter like adUnitId, will block the ad request and give* `onFailedToLoadInterstitialAd`*() callback.*
2. *Next interstitial ad request will send only the current received ad is consumed.*
3. *Frequent ad request is blocked if previous ad request is already in process.*
4. *In case of* `onFailedToLoadInterstitialAd`*() callback, next ad request should made after some time.*

3. Listen to Interstitial Ad callbacks

You can listen to Interstitial Ad event callbacks by implementing the ZaprInterstitialAdEventListener interface.

```
// Implement ZaprVideoAdEventListener
public class YourActivity extends Activity implements
ZaprInterstitialAdEventListener {


      @Override
      public void onInterstitialAdLoaded() {
          // Ad loaded. You can call showInterstitialAd() now
      }


      @Override
      public void onInterstitialAdShown() {
          // Interstitial Ad Shown to user
      }


      @Override
      public void onInterstitialAdClicked() {
          // Interstitial Ad is Clicked
      }


      @Override
      public void onInterstitialAdClosed() {
          // Interstitial Ad is Closed
      }


      @Override
      public void onFailedToLoadInterstitialAd(int errorCode, String
      errorMessage) {
          // Failed to load Interstitial Ad
      }
}
```

Set this interstitial ad listener on Interstitial ad instance.

```
// Set Interstitial ad event listener to Ad instance
mInterstitialAd.setInterstitialAdEventListener(this);
```

4. Show Interstitial Ad

You can show Interstitial ad to user only after receiving `onInterstitialAdLoaded()` callback.

To show video ad to user, call `showInterstitialAd()` method after `onInterstitialAdLoaded()` callback whenever you want to show ad to user.

```
private boolean isInterstitailAdReady = false;
 @Override
 public void onInterstitialAdLoaded() {
   // Interstitial ad is ready to display
   // Call showInterstitialAd() whenever you want to show ad to user
 }
```

```
...
// On some transaction event like before activity change or game level change,
// show Interstitial ad.
if(mInternstitialAd.isInterstitailAdReady()) {
    mInterstitialAd.showInterstitialAd();
}
```

5. Destroying Interstitial Ad view

It is very important to destroy instance of Interstitial Ad view when the hosting Activity or Fragment is destroyed. Destroying Interstitial Ad view instance clears up resources used and also unregisters broadcast receivers and location receivers if used.
To destroy Interstitial ad view instance, call `destroy()` method on Interstitial ad instance from `onDestroy()` callback of your hosting Activity or Fragment.

```
// Destroy the instance of Interstitial Ad View
@Override
protected void onDestroy() {
  super.onDestroy();
  if (mInterstitialAd != null) {
    mInterstitialAd.destroy();
  }
}
```

6. Interstitial Ad Customization

A. Passing User Information

You can optionally pass extra information in Interstitial ad for better ad targeting which may fetch more eCPM and revenue.

```
// Import UserInfo
import com.redbricklane.zapr.basesdk.model.UserInfo;

// Pass User Information like gender and age
UserInfo userInfo = new UserInfo();
userInfo.setAge(25);
userInfo.setGender("M");
```

```
// Set user info to Interstitial ad view instance
mInterstitialAd.setUserInfo(userInfo);
```

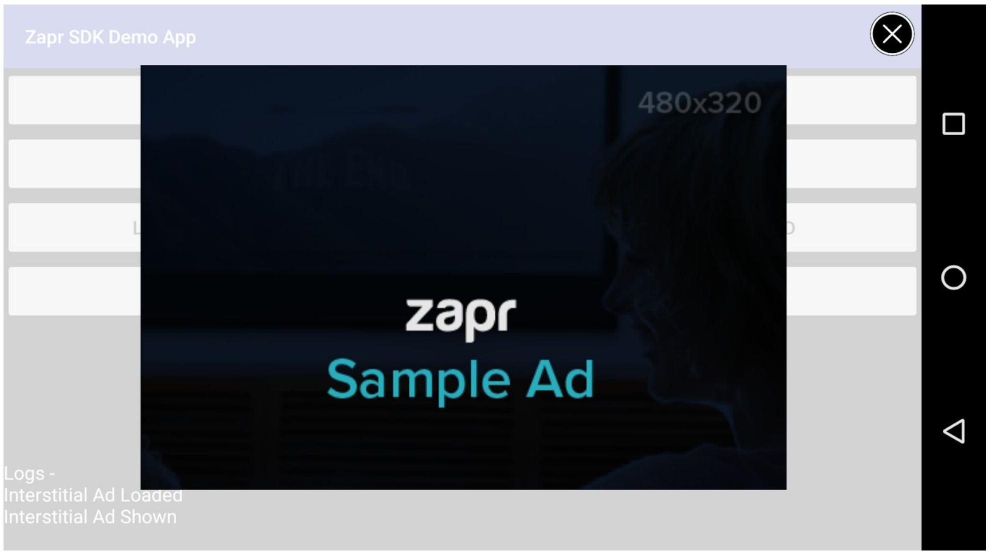
B. Changing background color of interstitial ads

Zapr Interstitial ads also provide optional functionality to change the background color of full screen interstitial ads. To change the background color, call `setInterstitialAdBackgroundColor()` on interstitial ad instance with color value.

The color can also be transparent or translucent as per alpha value set.
Note that the color value should be in to format `0xAARRGGBB`

For example: Following example changes the background color of interstitial ad to semi-transparent white background (0xCCFFFFFF).

```
// Eg: Change background color of interstitial ad (Format : 0xAARRGGBB)
mInterstitialAd.setInterstitialAdBackgroundColor(0xCCFFFFFF);
```



Logs -
Interstitial Ad Loaded
Interstitial Ad Shown

C. Runtime permissions check for Android M+ devices

Zapr Ad SDK needs permissions of Location and Phone state for detecting device information like location etc. Since the introduction of runtime permissions in Android M version, user needs to grant these permissions at runtime.

Zapr SDK checks runtime permissions when `loadInterstitialAd()` is called. And if permissions are not available, then Zapr SDK will request user to grant these permissions.

If application wants to handle runtime permission request itself, and if application do not want Zapr SDK to ask for runtime permissions to the user, you can disable this feature by calling `setRequestForPermissionsEnabled(false).`

```
// Disable request for runtime permissions (Not-Recommended)
mInterstitialAd.setRequestForPermissionsEnabled(false);
```

D. Location detection

Zapr SDK automatically fetches necessary information like device location etc, which is required for Ad serving. You can optionally disable this by using following API.

```
// Disable automatic location detection (Not-Recommended)
mInterstitialAd.setLocationDetectionEnabled(false);
```

Note: *It is highly recommended to keep location detection enabled for Ad serving. Ad requests with proper device location and user information usually fetches higher eCPM and revenue.*

E. Use of In App Browser

On clicking any Interstitial ad, the landing page opens in an In App Browser provided by Zapr SDK. If required, you can change this behaviour to open Ad's landing page in device default native web browser. Use following API to use native browser instead of In App Browser.

```
// If required, disable the use of In App Browser
mInterstitialAd.setUseInAppBrowser(false);
```

F. Serving Ads in test mode

During initial setup and testing phase, it is highly recommended to run the ads in test mode. Use the following API to enable test mode. In test mode, server may return test ads which will not be monetized.

```
// Enable test mode for ad serving
mInterstitialAd.setTestModeEnabled(true);
```

G. Enable Ad SDK logging

While integrating Zapr Ad SDK with your application, you may need to enable logging in SDK for debugging and testing purposes.
Various logging levels available are verbose, debug, info, warn, error and none.

```
// Enable Ad SDK logging at debug level
com.redbricklane.zapr.basesdk.Log.setLogLevel(LOG_LEVEL.debug);
```

Note: *Make sure to disable logging before releasing your app to Play store. You can disable logging by setting log level as LOG_LEVEL.none*

H. Set AdStatusListener

While integrating Zapr Interstitial Ads with your application, you may need to get the ad status related callbacks i.e. *adAlreadyLoaded()* and *adRequestBlocked(int seconds)*, then you have to set this AdStatusListener with your Interstitial ad object.

```
// Set AdStatusListener
mInterstitialAd.setAdStatusListener(new AdStatusListener(){
      @Override
      public void adAlreadyLoaded() {
          // You will get this callback if ad is already loaded
      }
      @Override
      public void adRequestBlocked(int seconds) {
      // You will get this callback in case of frequent ad request
      // int seconds : This parameter will give you the time in seconds, after
      that next ad request will be sent to server
      }
};)
```

Integrating Video Ads

Video ads usually fetch much higher eCPM than conventional Banner ads and thus integrating Video ads in your application can increase your application revenue.

Zapr SDK provides easy way to integrate and show full screen Video ads in your application.

Proguard Configuration

If you are using Zapr Video Ads in your application, you must add following lines in your application's `proguard-rules.pro` file. Adding following lines will ensure normal operation of SDK while creating final minified build of your app.

Add following lines in your apps `proguard-rules.pro` file which is located at app module's root folder.

```
// Append following lines at the end of proguard-rules.pro file
-keep class javax.xml.xpath.** {
  public protected private *;
}
-keep class org.w3c.dom.** {
  public protected private *;
}
```

Integrating Video Ads

It takes following easy steps to add a Video Ads to your application.

1. Initialize `ZaprVideoAd` in your application Activity
2. Load video ad
3. Listen to video ad callbacks
4. Show video ad when ready
5. Set optional video ad parameters

1. Initialize Zapr Video Ad

Initialize `ZaprVideoAd` and set mandatory parameters like Ad unit ID to this instance.

```
// Import ZaprVideoAd
import com.redbricklane.zapr.videosdk.ZaprVideoAd;
```

```
// Initialize ZaprVideoAd in onCreate() method of your Activity
// Pass your activity context in the constructor
ZaprVideoAd mVideoAd = new ZaprVideoAd(context);
// Set Ad unit ID
mVideoAd.setAdUnitId("<ad-unit-id>");
```

***Note**: Calling loadAd() without setting mandatory parameter like adUnitId, will block the ad request and give onVideoAdError() callback.*

2. Load Video Ad

To request video ad from server call `loadAd()` method.

***Note**: Loading video ad does not directly display the video to user without calling `showAd()` method as described below.*

```
// Request for video ad from server
mVideoAd.loadAd();
```

Note:

1. *Calling loadAd() without setting mandatory parameter like adUnitId, will block the ad request and give `onVideoAdError`() callback.*
2. *Next video ad request will send only the current received ad is consumed.*
3. *Frequent ad request is blocked if previous ad request is already in process.*
4. *In case of `onVideoAdError`() callback, next ad request should made after some time.*

3. Listen to Video Ad callbacks

You can listen to Video Ad event callbacks by implementing `ZaprVideoAdEventListener` interface.

```
// Implement ZaprVideoAdEventListener
public class YourActivity extends Activity implements
ZaprVideoAdEventListener{


    @Override
    public void onVideoAdError(int errorCode, String errorMessage) {
        // Video Ad error
    }
    @Override
    public void onResponseReceived(VideoAdResponse adResponse) {
        // Video ad response received from server successfully.
        // This callback is fired before video ad caching is started.
    }

    @Override
    public void onAdReady(VideoAdResponse adResponse, String vastXML) {
        // This callback is fired after video ad caching is complete and
        // video ad is ready to play.
        // Call showAd() to show video ad
    }

    @Override
    public void onVideoAdStarted() {
        // Video ad playing started
    }

    @Override
    public void onVideoAdClicked() {
        // Video ad is clicked
    }

    @Override
    public void onVideoAdFinished() {
        // Video Ad Playing Finished
    }

    @Override
    public void onVideoPlayerClosed() {
        // Video Ad Player closed
    }
// Activity code...
// Pass the instance of your activity to Zapr Video Ad
mVideoAd.setZaprVideoAdEventListener(this);
}
```

4. Show Video Ad

You can show video ad to user only after receiving onReady() callback.

To show video ad to user, call showVideoAd() method after onReady() callback whenever you want to show video ad to user.

```
   @Override
    public void onAdReady(VideoAdResponse adResponse, String vastXML) {
        // Video ad is ready to play
    }
      // Check for video ad is ready or not using below api
   if(mVideoAd.isVideoAdReady()){
       // Call showVideoAd now
       mVideoAd.showVideoAd();
    }
```

5. Destroying Video Ad instance

It is very important to destroy instance of Video Ad when the hosting Activity or Fragment is destroyed. Destroying Video Ad view instance clears up resources used and also unregisters broadcast receivers and location receivers if used.
To destroy Video ad view instance, call `destroy()` method on video ad instance from `onDestroy()` callback of your hosting Activity or Fragment.

```
// Destroy the instance of Video Ad View
@Override
protected void onDestroy() {
  super.onDestroy();
  if (mVideoAd != null) {
    mVideoAd.destroy();
  }
}
```

6. Set optional video ad parameters

Passing Video Ad specific request parameters

You can optionally set extra parameters, like video ad duration, while requesting video ads.

```
// Set Video ad minimum duration
mVideoAd.setMinDuration(5);

// Set Video ad maximum duration
mVideoAd.setMaxDuration(300);

// Request to block skippable video ads
mVideoAd.setBlockSkippableAds(true);
```

Passing User Information

You can optionally pass extra information in Video ad for better ad targeting which may fetch more eCPM and revenue.

```
// Import UserInfo
import com.redbricklane.zapr.basesdk.model.UserInfo;

// Pass User Information like gender and age (Example shown below)
UserInfo userInfo = new UserInfo();
userInfo.setAge(25);
userInfo.setGender("M");
// Set user info to video ad view instance
mVideoAd.setUserInfo(userInfo);
```

7. Disable auto pre-caching of video ad (Optional)

Zapr Video SDK automatically starts pre-caching of video ad when a successful video ad response is received (ie after `onResponseReceived()` callback). And once ad is cached, `onAdReady()` callback is fired.

But if you need to disable this auto pre-caching of video ad you can call `setPreCachingEnabled(false);`
With auto pre-caching disabled, you will have to call `cacheVideoAd()` manually after receiving `onResponseReceived()` callback. Once `cacheVideoAd()` is manually called, Zapr SDK will start caching of video ad and will fire `onAdReady()` callback once ad is cached completely.

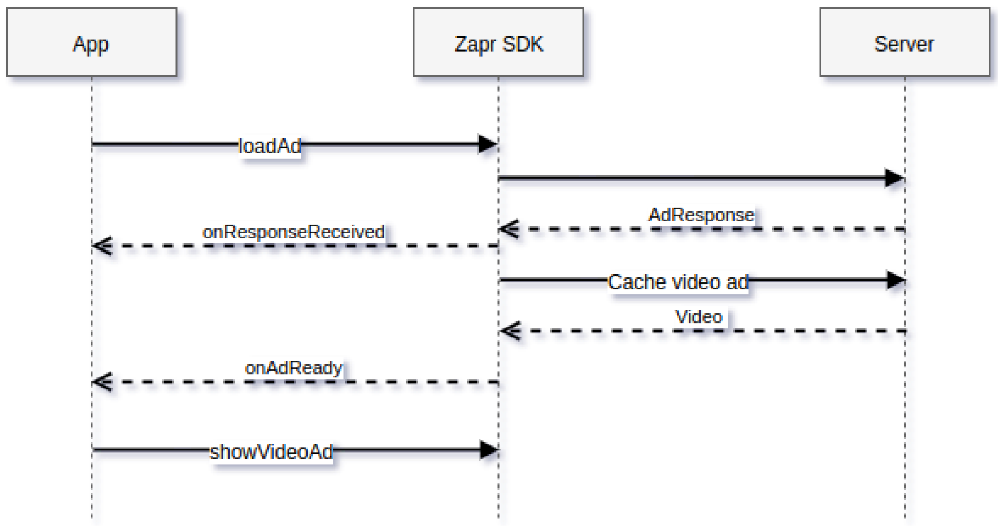
```
// Disable pre-caching before calling loadAd()
mVideoAd.setPreCachingEnabled(false);
// Call Load ad
mVideoAd.loadAd();
```

Listen for `onResponseReceived()` callback. And call `cacheVideoAd()` after receiving onResponseReceived() callback.

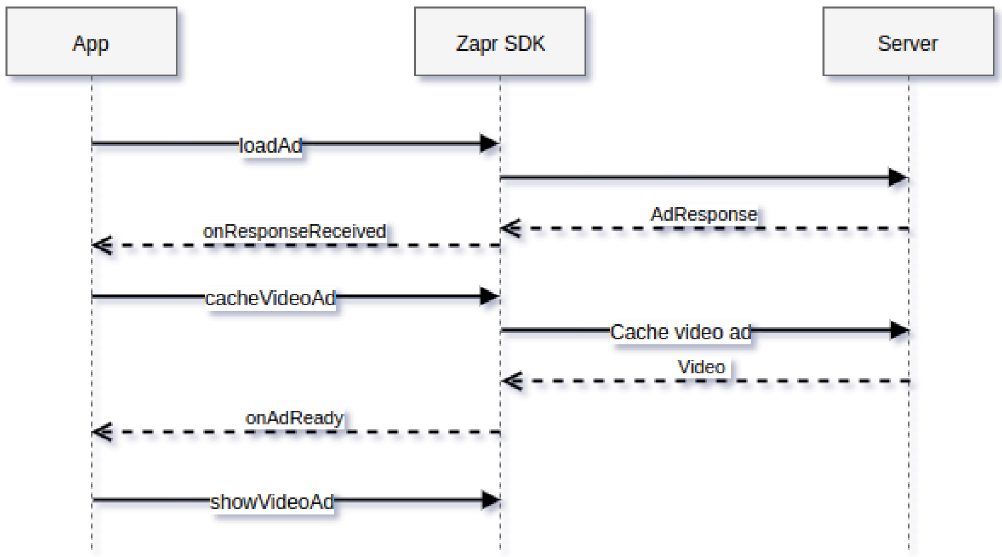
```
@Override
public void onResponseReceived(VideoAdResponse adResponse) {
    // Video ad response received from server successfully.
    // Call to cache video ad when required after receiving this callback
    mVideoAd.cacheVideoAd();
}
```

Note: *Auto pre-caching is enabled by default in Zapr SDK.*
Following sequence diagram shows method calls and callbacks with and without pre caching.

With Pre-Caching enabled -



With Pre-Caching disabled -



8. Video Ad customization

A. Runtime permissions check for Android M+ devices

Zapr Ad SDK needs permissions of Location and Phone state for detecting device information like location etc. Since the introduction of runtime permissions in Android M version, user needs to grant these permissions at runtime.
Zapr SDK checks runtime permissions when `loadAd()` is called. And if permissions are not available, then Zapr SDK will request user to grant these permissions.

If application wants to handle runtime permission request itself, and if application do not want Zapr SDK to ask for runtime permissions to the user, you can disable this feature by calling `setRequestForPermissionsEnabled(false)`.

```
// Disable request for runtime permissions (Not-Recommended)
mVideoAd.setRequestForPermissionsEnabled(false);
```

B. Location detection

Zapr SDK automatically fetches necessary information like device location etc, which is required for Ad serving. You can optionally disable this by using following API.

```
// Disable automatic location detection (Not-Recommended)
mVideoAd.setLocationDetectionEnabled(false);
```

C. Serving Ads in test mode

During initial setup and testing phase, it is highly recommended to run the ads in test mode. Use the following API to enable test mode.

```
// Enable test mode for ad serving
mVideoAd.setTestModeEnabled(true);
```

D. Set AdStatusListener

While integrating Zapr Video Ads with your application, you may need to get the ad status related callbacks i.e. *adAlreadyLoaded()* and *adRequestBlocked(int seconds)*, then you have to set this AdStatusListener with your Video ad object.

```
// Set AdStatusListener
mVideoAd.setAdStatusListener(new AdStatusListener(){
      @Override
      public void adAlreadyLoaded() {
          // You will get this callback if ad is already loaded
      }
      @Override
      public void adRequestBlocked(int seconds) {
      // You will get this callback in case of frequent ad request
      // int seconds : This parameter will give you the time in seconds, after
      that next ad request will be sent to server
      }
};)
```

E. Disable zapr vast ad player

If you wish to disable the Zapr's VAST player and want to play VAST response in your own player, then you can achieve this by using below api.

```
// Disable zapr's default vast ad player
// Default value of this api is false
mVideoAd.disableZaprVastAdPlayer(true)
```

Note:

1. *Once this api is used, zapr sdk will not play vast ad by using mVideoAd.showVideoAd().*
2. *You will get vast xml response in onAdReady() and onResponseReceived() callbacks.*
3. *To play zapr video ad in zapr's vast player, you have to reset the disableZaprVastAdPlayer api by passing false.*

F. Enable Ad SDK logging

While integrating Zapr Ad SDK with your application, you may need to enable logging in SDK for debugging and testing purposes.
Various logging levels available are verbose, debug, info, warn, error and none.

```
// Enable Ad SDK logging at debug level
com.redbricklane.zapr.basesdk.Log.setLogLevel(LOG_LEVEL.debug);
```

Note: *Make sure to disable logging before releasing your app to Play store. You can disable logging by setting log level as* LOG_LEVEL.none.

Integrating Rewarded Video Ads

Zapr SDK also provides you with a new ad format, namely Rewarded video ad which helps game developers to deal with rewards to be given when video ad rendering completes and the video player is closed.

SDK provides a very simplistic way to integrate the Rewarded video ads in your application.

Proguard Configuration

If you are using Zapr Rewarded Video Ads in your application, you must add following lines in your application's `proguard-rules.pro` file. Adding following lines will ensure normal operation of SDK while creating final minified build of your app.

Add following lines in your apps `proguard-rules.pro` file which is located at app module's root folder.

```
// Append following lines at the end of proguard-rules.pro file
-keep class javax.xml.xpath.** {
  public protected private *;
}
-keep class org.w3c.dom.** {
  public protected private *;
}
```

Integrating Rewarded Video Ads

It takes following easy steps to add a Video Ads to your application.

1. Initialize `ZaprRewardedVideoAd` in your application Activity
2. Load rewarded video ad
3. Listen to rewarded video ad callbacks
4. Show the rewarded video ad when ready
5. Set optional rewarded video ad parameters
6. Listen for rewarded video ad callback for gratification of reward
7. Take necessary action once the Reward gratification callback is received

1. Initialize Rewarded Zapr Video Ad

Initialize `ZaprRewardedVideoAd` and set mandatory parameters like Ad unit ID to this instance.

```
// Import ZaprRewardedVideoAd
import com.redbricklane.zapr.videosdk.ZaprRewardedVideoAd;

// Initialize ZaprVideoAd in onCreate() method of your Activity
// Pass your activity context in the constructor
ZaprRewardedVideoAd mRewardedVideoAd = new ZaprRewardedVideoAd(context);
// Set Ad unit ID
mRewardedVideoAd.setAdUnitId("<ad-unit-id>");
```

Note*: Calling loadAd() without setting mandatory parameter like adUnitId, will block the ad request and give onVideoAdError() callback.*

2. Load Rewarded Video Ad

To request rewarded video ad from server call `loadAd()` method.

Note*: Loading video ad does not directly display the video to user without calling `showAd()` method as described below.*

```
// Request for video ad from server
mRewardedVideoAd.loadAd();
```

Note:

5. *Calling loadAd() without setting mandatory parameter like adUnitId, will block the ad request and give `onVideoAdError`() callback.*
6. *Next rewarded video ad request will send only the current received ad is consumed.*
7. *Frequent consequtive ad requests are blocked if any previous ad request is already in process.*
8. *In case of `onVideoAdError`() callback, next ad request should made after some time.*

3. Listen to Rewarded Video Ad callbacks

You can listen to Rewarded Video Ad event callbacks by implementing `ZaprRewardedVideoAdEventListener` interface.

```
// Implement ZaprRewardedVideoAdEventListener
public class YourActivity extends Activity implements
ZaprRewardedVideoAdEventListener{


    @Override
```

```
public void onVideoAdError(int errorCode, String errorMessage) {
    // Video Ad error
}

@Override
public void onResponseReceived(VideoAdResponse adResponse) {
    // Video ad response received from server successfully.
    // This callback is fired before video ad caching is started.
}

@Override
public void onAdReady(VideoAdResponse adResponse, String vastXML) {
    // This callback is fired after video ad caching is complete and
    // video ad is ready to play.
    // Call showAd() to show video ad
}

@Override
public void onVideoAdStarted() {
    // Video ad playing started
}

@Override
public void onVideoAdClicked() {
    // Video ad is clicked
}

@Override
public void onVideoAdFinished() {
    // Video Ad Playing Finished
}

@Override
public void onVideoPlayerClosed() {
        // Video Ad Player closed
    }

    @Override
    public void onVideoAdRewardGratified(String rewardType, double
rewardAmount) {
        // Video Ad Reward Gratified
    }
// Activity code...

// Pass the instance of your activity to Zapr Rewarded Video Ad
mRewardedVideoAd.setZaprRewardedVideoAdEventListener(this);
}
```

4. Show Rewarded Video Ad

You can show video ad to user only after receiving onReady() callback.

To show video ad to user, call showVideoAd() method after onReady() callback whenever you want to show video ad to user.

```
   @Override
    public void onAdReady(VideoAdResponse adResponse, String vastXML) {
        // Rewarded Video ad is ready to play
    }
      // Check for video ad is ready or not using below api
   if(mRewardedVideoAd.isVideoAdReady()){
       // Call showVideoAd now
       mRewardedVideoAd.showVideoAd();
    }
```

5. Destroying Rewarded Video Ad instance

It is very important to destroy instance of Rewarded Video Ad when the hosting Activity or Fragment is destroyed. Destroying Rewarded Video Ad view instance clears up resources used and also unregisters broadcast receivers and location receivers if used.
To destroy Rewarded Video ad view instance, call `destroy()` method on rewarded video ad instance from `onDestroy()` callback of your hosting Activity or Fragment.

```
// Destroy the instance of Video Ad View

@Override
protected void onDestroy() {
  super.onDestroy();
  if (mRewardedVideoAd != null) {
    mRewardedVideoAd.destroy();
  }
}
```

6. Set optional video ad parameters

Passing Rewarded Video Ad specific request parameters

You can optionally set extra parameters, like video ad duration, while requesting video ads.

```
// Set Rewarded Video ad minimum duration

mRewardedVideoAd.setMinDuration(5);


// Set Rewarded Video ad maximum duration

mRewardedVideoAd.setMaxDuration(300);
```

Passing User Information

You can optionally pass extra information in Rewarded Video ad for better ad targeting which may fetch more eCPM and revenue.

```
// Import UserInfo

import com.redbricklane.zapr.basesdk.model.UserInfo;


// Pass User Information like gender and age (Example shown below)

UserInfo userInfo = new UserInfo();

userInfo.setAge(25);
userInfo.setGender("M");
// Set user info to rewarded video ad view instance
mRewardedVideoAd.setUserInfo(userInfo);
```

7. Disable auto pre-caching of rewarded video ad (Optional)

Zapr Rewarded Video SDK automatically starts pre-caching of video ad when a successful rewarded video ad response is received (ie after `onResponseReceived()` callback). And once ad is cached, `onAdReady()` callback is fired.

But if you need to disable this auto pre-caching of the rewarded video ad you can call `setPreCachingEnabled(false)`;
With auto pre-caching disabled, you will have to call `cacheVideoAd()` manually after receiving `onResponseReceived()` callback. Once `cacheVideoAd()` is manually called, Zapr SDK will start caching of video ad and will fire `onAdReady()` callback once ad is cached completely.

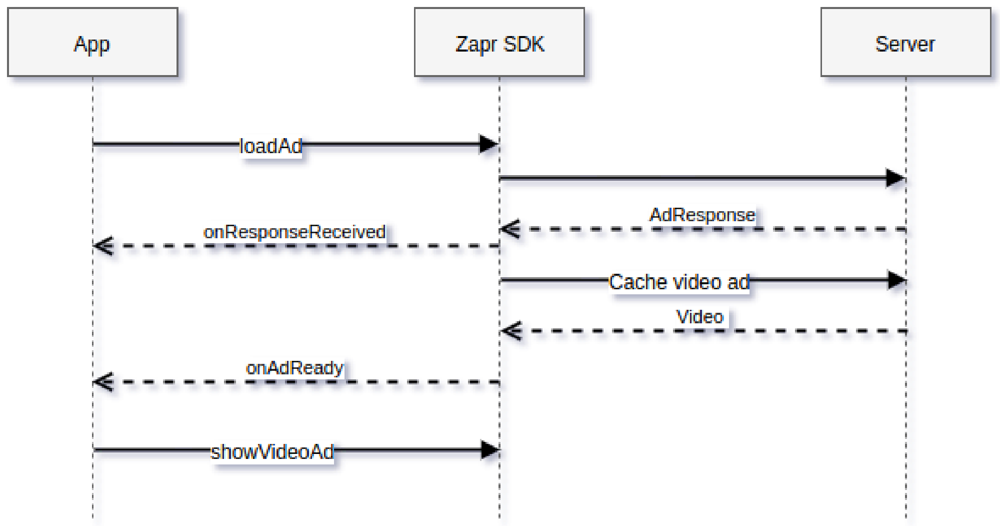
```
// Disable pre-caching before calling loadAd()
mRewardedVideoAd.setPreCachingEnabled(false);
// Call Load ad
mRewardedVideoAd.loadAd();
```

Listen for `onResponseReceived()` callback. And call `cacheVideoAd()` after receiving onResponseReceived() callback.

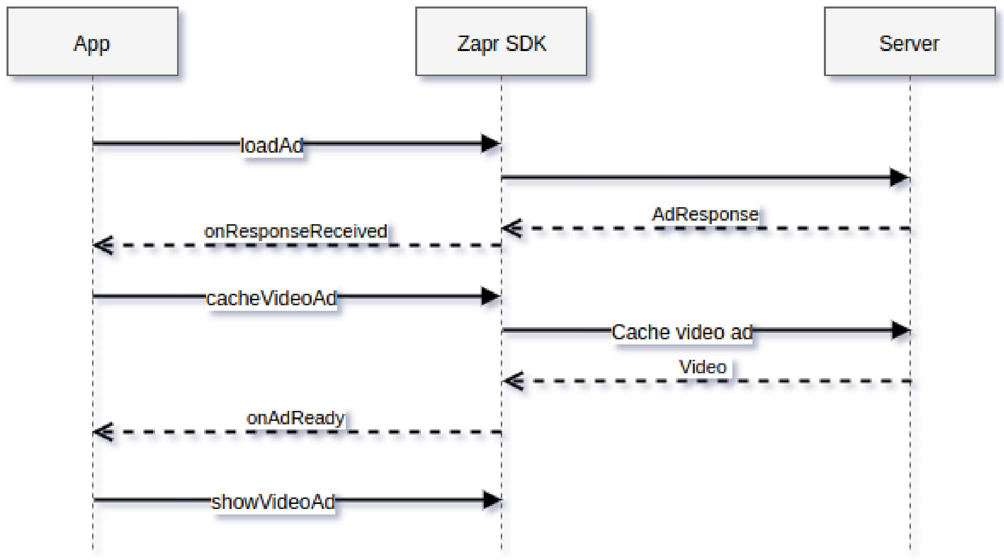
```
@Override
public void onResponseReceived(VideoAdResponse adResponse) {
    // Video ad response received from server successfully.
    // Call to cache video ad when required after receiving this callback
    mRewardedVideoAd.cacheVideoAd();
}
```

Note: *Auto pre-caching is enabled by default in Zapr SDK.*
Following sequence diagram shows method calls and callbacks with and without pre caching.

With Pre-Caching enabled -



With Pre-Caching disabled -



8. Rewarded Video Ad customization

A. Runtime permissions check for Android M+ devices

Zapr Ad SDK needs permissions of Location and Phone state for detecting device information like location etc. Since the introduction of runtime permissions in Android M version, user needs to grant these permissions at runtime.

Zapr SDK checks runtime permissions when `loadAd()` is called. And if permissions are not available, then Zapr SDK will request user to grant these permissions.
If application wants to handle runtime permission request itself, and if application do not want Zapr SDK to ask for runtime permissions to the user, you can disable this feature by calling `setRequestForPermissionsEnabled(false)`.

```
// Disable request for runtime permissions (Not-Recommended)
mRewardedVideoAd.setRequestForPermissionsEnabled(false);
```

B. Location detection

Zapr SDK automatically fetches necessary information like device location etc, which is required for Ad serving. You can optionally disable this by using following API.

```
// Disable automatic location detection (Not-Recommended)
mRewardedVideoAd.setLocationDetectionEnabled(false);
```

C. Serving Ads in test mode

During initial setup and testing phase, it is highly recommended to run the ads in test mode. Use the following API to enable test mode.

```
// Enable test mode for ad serving
mRewardedVideoAd.setTestModeEnabled(true);
```

D. Set AdStatusListener

While integrating Zapr Rewarded Video Ads with your application, you may need to get the ad status related callbacks i.e. *adAlreadyLoaded()* and *adRequestBlocked(int seconds)*, then you have to set this AdStatusListener with your Video ad object.

```
// Set AdStatusListener
mRewardedVideoAd.setAdStatusListener(new AdStatusListener(){
      @Override
      public void adAlreadyLoaded() {
          // You will get this callback if ad is already loaded
      }
      @Override
      public void adRequestBlocked(int seconds) {
      // You will get this callback in case of frequent ad request
      // int seconds : This parameter will give you the time in seconds, after
      that next ad request will be sent to server
      }
};)
```

E. Enable Ad SDK logging

While integrating Zapr Ad SDK with your application, you may need to enable logging in SDK for debugging and testing purposes.
Various logging levels available are verbose, debug, info, warn, error and none.

```
// Enable Ad SDK logging at debug level
com.redbricklane.zapr.basesdk.Log.setLogLevel(LOG_LEVEL.debug);
```

Note: *Make sure to disable logging before releasing your app to Play store. You can disable logging by setting log level as* LOG_LEVEL.none.

General Data Protection Regulation

Zapr supports new *GDPR*

(General Data Protection Regulation) for EU users

Zapr helps publishers to be compliant with GDPR by not collecting, sharing or storing any data from smartphone users located in any country falling within the European Union.

Zapr has deleted deletes ALL historical data of EU users from all SDK versions integrated with publishers.

EU users are detected by SIM country API and/or IP address. Zapr does not collect or store any data of these users.

For Zapr SDK version 3.x onwards, we strongly recommend publisher apps not to call `Zapr.start()` API for EU users, or use `Zapr.killSdk()` API to stop Zapr Data SDK completely. Using the `Zapr.killSdk()` API, publishers can proactively stop the Data SDK from running on the devices of EU users, and ensure that Zapr cannot collect or store any user data.

Please refer Zapr's privacy policy for further details.

If you need any more clarification regarding Zapr's role in your compliance with data regulations, write to us at privacy[at]zapr[dot]in.

Appendix

Manifest file permissions

Following permissions are used by zapr-sdk.

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

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

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

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

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

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

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

In case of any issue with automatic manifest merging, please add above permissions in your application's Manifest file.

Activity, Service and Broadcast Receiver Entries

Following are the entries of Activity, Services and Broadcast Receivers used by zapr-sdk.

In case of any issue with automatic manifest merging, please add below entries inside `<application>` tag your application's Manifest file.

```
<activity
android:name="com.redbricklane.zapr.videosdk.vastplayer.activity.ZaprVASTActivity"
      android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
      android:configChanges=”orientation|screenSize|keyboardHidden”/>


<service android:name="com.redbricklane.zaprSdkBase.services.zaprProcess.Ariel"
    android:enabled="true"
    android:process=":zapr_process" />
<service
android:name="com.redbricklane.zaprSdkBase.services.zaprProcess.ConfigUpdateService"
    android:enabled="true"
    android:process=":zapr_process" />
<service
```

```
android:name="com.redbricklane.zaprSdkBase.services.appProcess.ZaprCredentialsServic
e"
    android:enabled="true" />
<service
    android:name="com.redbricklane.zaprSdkBase.services.appProcess.LocationService"
    android:enabled="true" />
<service
android:name="com.redbricklane.zaprSdkBase.services.appProcess.UpdateDetailsService"
    android:enabled="true" />

<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />

<receiver
android:name="com.redbricklane.zaprSdkBase.broadcastReceivers.implicit.StartServiceR
eceiver">
    <intent-filter>
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
        <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED" />
        <action android:name="android.intent.action.PACKAGE_REPLACED" />
        <action android:name="android.intent.action.MY_PACKAGE_REPLACED" />
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<receiver
android:name="com.redbricklane.zaprSdkBase.broadcastReceivers.explicit.RegisterServe
rReceiver">
    <intent-filter>
        <action android:name="REGISTER_BR" />
    </intent-filter>
</receiver>
<receiver
android:name="com.redbricklane.zaprSdkBase.broadcastReceivers.explicit.PriorityRecei
ver">
    <intent-filter>
        <action android:name="com.redbricklane.zaprSdkBase.PRIORITY_ACTION" />
    </intent-filter>
</receiver>
<!-- JobServices -->
```

```
        <service
android:name="com.redbricklane.zaprSdkBase.services.zaprProcess.ArielJobService"
            android:enabled="true"
            android:exported="true"
            android:permission="android.permission.BIND_JOB_SERVICE"
            android:process=":zapr_process" />
        <service
android:name="com.redbricklane.zaprSdkBase.services.zaprProcess.ConfigUpdateJobServi
ce"
            android:enabled="true"
            android:exported="true"
            android:permission="android.permission.BIND_JOB_SERVICE"
            android:process=":zapr_process" />
        <service
android:name="com.redbricklane.zaprSdkBase.services.appProcess.LocationJobService"
            android:enabled="true"
            android:exported="true"
            android:permission="android.permission.BIND_JOB_SERVICE" />
    <!-- JobServices -->
```

Troubleshooting

FirebaseApp is not initialized (in process)

It has been observed that if your app is using Firebase SDK and app has multiple processes defined in Manifest, then in some version of Firebase library, Firebase SDK gives error `‘Default FirebaseApp is not initialized in this process xyz_process. Make sure to call FirebaseApp.initializeApp(Context)’.`
Suggested workaround for this issue is to initialize Firebase App in `onCreate()` method of `Application` class in your app. For doing so, first create a new class extending `Application` class of Android. And specify this class as `android:name` property in the the `<application>` tag in your app’s `AndroidManifest.xml`

```
<application
   android:name=".XyzApplicationClass"
   android:icon="@drawable/icon"
   android:label="@string/app_name" … >
```

Now, in the onCreate method of your class, initialize Firebase App.

```
import android.app.Application;
import com.google.firebase.FirebaseApp;

public class XyzApplicationClass extends Application {
  @Override
  public void onCreate() {
    super.onCreate();
    // Init firebase here
    try {
      FirebaseApp.initializeApp(this);
    }
    catch (Exception e) {
      // Error initializing Firebase.
    }
  }
}
```

Dex Error : Enable Multidexing

If adding third party sdk libraries to your project causes dex error. Enable multidex support for your app. Refer following link to enable multidex support in app.
https://developer.android.com/tools/building/multidex.html

Manifest merge conflict

In case of any issues during automatic Manifest file merging, please refer Android developer documentation on Manifest merge.