KingSoft Ksyun Mobile Advertising Android SDK Quick Access Document V4.0.3

ChangeLog

Build 4.0.3 [2018/3/9]
1.Renamed the preloadAd interface to loadAd,the same as the name of related callback methods.
2.Removed the hasLocalAd interface,now you can use hasAd for all cases.

Build 4.0.2 [2018/3/5]
1.Fixed the issue that the close button was occasionally missing
2.Fixed the issue that the home key press may occasionally cause the destruction of the video activity.
3.Modified the package name of FileProvider to avoid name confliction

Build 4.0.1 [2018/1/26]
1.Add new interface of hasLocalAd
2.Add support for the sandbox environment

Build 4.0.0 [2017/12/15]
1.First commit

Catalog

1. List of SDK package contents

The SDK contains the files as follows:

- SDK demo project
- Jar package SDK, and assets folder (contains SDK initial plug-in apk)
- AAR-style SDK (SDK initial plug-in apk)
- SDK Quick Guide Document

2. Integration of SDK

The SDK supports two ways for integration:

- AAR file (**recommended**)
- Jar package + Asset resource

Customers can choose either one to integrate the SDK according for your own convenience.

3. SandBox for testing integration

The SDK supports two working environments: the sandbox (SANDBOX_ENV) and the online (RELEASE_ENV). The default is the Sandbox. Before initialization, you can change the working environment by means of the SDK configuration items.
It is recommended that customers first use the sandbox environment and the test AppId for developing and testing. After confirm that the interface can work and the

data is correct, customers are suggested to switch to the online environment and the corresponding online AppId for the real production.

4. Instructions for Unity export Android project

If your exported Android project for Android Studio

- AAR integration is recommended
- Additional Android Support V4 support library required

If your exported Android project for Eclipse

- Recommend the use of Jar+Assets for import (AAR file NOT support by Eclipse)
- An additional Android Support V4 support library needs to be added and the version of the V4 library added should match the version of Target SDK that you are compiling in Eclipse

5. Android runtime permissions

By default in Android 6.0 or above system, when initialization, the SDK would request the host app for the following runtime permissions.

- Manifest.permission.READ_PHONE_STATE (**Required**，for generating a unique ID)
- Manifest.permission.ACCESS_COARSE_LOCATION,
- Manifest.permission.ACCESS_FINE_LOCATION (**Optional**，for geo-location related)
  If the APP does not want the SDK to apply for runtime permissions, you can set the corresponding SDK configuration item.

6. Import SDK

SDK library file

The first way, based on the AAR file (recommended)

1.Put the AAR package in the libs folder of the app project's root folder (if it does not exist, you can create one).
2.Add the following code in your app build.gradle file

```
android {
    ...
    repositories {
        flatDir {
            dirs 'libs'
        }
    }
}

dependencies {
     //  you should fill in the real name of the aar file, here only to sdk-xxx.aar as an example
     compile (name: 'sdk-xxx.aar', ext: 'aar')
}
```

3.**Add Manifest permissions**

```
<uses-permission android: name = "android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android: name = "android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android: name = "android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android: name = "android.permission.ACCESS_WIFI_STATE" />
<uses-permission android: name = "android.permission.READ_PHONE_STATE" />
<uses-permission android: name = "android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android: name = "android.permission.CHANGE_WIFI_STATE" />
<uses-permission android: name = "android.permission.INTERNET" />
<uses-permission android: name = "android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
<uses-permission android: name = "android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android: name = "android.permission.ACCESS_FINE_LOCATION" />
```

4.**Register SDK components**

```
// Add provider here, for compatible of the automatic installation in Android 7.0 or above
<provider
    android: name = "com.ksc.ad.sdk.util.KsyunFileProvider"
    android: authorities = "${applicationId}.fileprovider"
    android: exported = "false"
    android: grantUriPermissions = "true">
    <meta-data
        android: name = "android.support.FILE_PROVIDER_PATHS"
        android: resource = "@xml/file_paths" />
</ provider>
```

5.**Copy the xml folder within the AAR folder of the SDK to the appropriate folder of the app module project. Modify the package name in the xml file as followings:**

```
    // pay attention to the value of the path below, you need to fill in the user's own package name
    <external-path path = "Android/data/com.xxx.xxx.xxx/"
            name = "files_root"/>
    <external-path path = "cache/apk/." name = "external_storage_root" />
```

The second way, based on the Jar package + Asset resource

1.**Place the corresponding jar package in the libs folder in the root folder of the app module project(if it does not exist, create one).**

2.**Add Manifest permissions**

```
<uses-permission android: name = "android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android: name = "android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android: name = "android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android: name = "android.permission.ACCESS_WIFI_STATE" />
```

```
<uses-permission android: name = "android.permission.READ_PHONE_STATE" />
<uses-permission android: name = "android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android: name = "android.permission.CHANGE_WIFI_STATE" />
<uses-permission android: name = "android.permission.INTERNET" />
<uses-permission android: name = "android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
<uses-permission android: name = "android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android: name = "android.permission.ACCESS_FINE_LOCATION" />
```

3.**Register SDK components**

```
// Rewards video show Activity
<activity
    android: name = "com.ksc.ad.sdk.ui.AdProxyActivity"
     android: hardwareAccelerated = "true"
    android: theme = "@android:style/Theme.Black.NoTitleBar.Fullscreen"
    android: configChanges = "keyboardHidden|orientation|screenSize" />
// Runtime permissions, transparent floating layer Activity
<activity
    android: name = "com.ksc.ad.sdk.ui.AdPermissionProxyActivity"
    android: configChanges = "keyboardHidden|orientation|screenSize"
    android: theme = "@android:style/Theme.Translucent.NoTitleBar.Fullscreen" />
<service android: name = "com.ksc.ad.sdk.service.AdProxyService" />
// Add provider here, for the automatic installation in android 7.0 and above
<provider
    android: name = "com.ksc.ad.sdk.util.KsyunFileProvider"
    // Attention to the value of the com.xxx part of the authorities below, you need to fill in the user's own package name
    android: authorities = "com.xxx.xxx.xxx.fileprovider"
    android: exported = "false"
    android: grantUriPermissions = "true">
    <meta-data
        android: name = "android.support.FILE_PROVIDER_PATHS"
        android: resource = "@xml/file_paths" />
</ provider>
```

4.**Copy the xml folder within the Jar folder of the SDK to the appropriate folder of the app module project. Modify the package name in the xml file as followings:**

```
    // Attention to the value of the path below, you need to fill in y
our own package name
    <external-path path = "Android/data/com.xxx.xxx.xxx/"
            name = "files_root" />
    <external-path path = "cache/apk/." name = "external_storage_root"
/>
```

5.**Copy the content of the assets folder in the SDK folder to the your app's folder of src/main/assets**

7. Quick start guide for the SDK

7.1. Initialization and Ad loading

It is recommended to start the initialization when lauching your app's first page (OnCreate).If you do not call the method of setSdkEnvironment () to set the SDK environment, the default is the Sandbox.

Please call the loadAd interface in reasonable time after the callback of init success.

```
class MainActivity extends Activity {

    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);

        KsyunAdSdkConfig config = new KsyunAdSdkConfig ();
        // Set the SDK explicitly to the online environment. if you do
not set the config, the default is the Sandbox
        config.setSdkEnvironment (KsyunAdSdkConfig.RELEASE_ENV);
        KsyunAdSdk.getInstance (). Init (MainActivity.this,
"your_release_app_id", config, new IKsyunAdInitResultListener () {
            @Override
            public void onSuccess (Map <String, String> map) {
                // Set the callback method of related ad event.
                KsyunAdSdk.getInstance().setAdListener(this);
```

```
                KsyunAdSdk.getInstance().setRewardVideoAdListener(this);
            }

            @Override
            public void onFailure (int errCode, String errMsg) {
                // SDK initialization failed
            }
        });
    }

    //Load the ad in reasonable time after the callback of init succes
s.
    public void onLevelStart(){
        // This call will try to load ads for all the ad slots in your
app
        KsyunAdSdk.getInstance ().loadAd (new IKsyunAdLoadListener () {
            @Override
            public void onAdInfoSuccess () {
            // succeeded to load ad configuration
            }

            @Override
            public void onAdInfoFailed (final int errCode, final String
errMsg) {
            // Failed to load ad configuration
            }

            @Override
            public void onAdLoaded (final String adSlotId) {
            // This method may be called multiple times depending on th
e number of ads loaded for all the ad slots
            }
        });
    }

}
```

7.2. Showing Ad

Before you are going to show your UI item for watching the reward video, we suggest that you first call the hasAd() to check whether the current Ad slot has an advertisement ready for displaying. If the Ad is ready, you can call showAd() to start playing the video for the user.

```
1.
2.       //Before the enterance of reward video.
3.      public void onGameOver(){
4.       // check whether there is an Ad exist
5.       boolean isExist = KsyunAdSdk.getInstance ().hasAd (adslot_id);
6.       if (isExist) {
7.          // The Ad exists, call showAd()
8.          //KsyunAdSdk.getInstance().showAd(MainActivity.this,"YOUR_ADSLOT_
    ID");
9.       } else {
10.         // The Ad is not ready, you can call loadAd() to trigger a loa
    d
11.         // This call will try to load ads for all the ad slots in your
    app
12.         KsyunAdSdk.getInstance ().loadAd (adslot_id, new
    IKsyunAdLoadListener () {
13.             @Override
14.             public void onAdInfoSuccess () {
15.             // succeeded to load ad configuration
16.             }
17.
18.             @Override
19.             public void onAdInfoFailed (final int errCode, final String
    errMsg) {
20.             // Failed to load ad configuration
21.             }
22.
23.             @Override
24.             public void onAdLoaded (final String adSlotId) {
25.             // This method may be called multiple times depending on th
    e number of ads loaded for all the ad slots
26.             }
27.         });
28.       }
29.   }
```

8. Advanced usage

8.1. SDK configuration

Before calling the method of init(), you can change the environment and some other options by setting the SDK configuration items

```
        KsyunAdSdkConfig config = new KsyunAdSdkConfig ();
        // Set SDK to the online environment. the default is the Sandbox
        config.setSdkEnvironment (KsyunAdSdkConfig.RELEASE_ENV);
        // Allow the close button to appear during the playback of your reward video
        config.setShowCloseBtnOfRewardVideo (true);
        // Set the waiting time period for showing the close button after starting the ad video playback
        config.setCloseBtnComingTimeOfRewardVideo (5);

        KsyunAdSdk.getInstance().Init (MainActivity.this, appId, config, new IKsyunAdInitResultListener () {
            @Override
            public void onSuccess (Map <String, String> map) {

            }

            @Override
            public void onFailure (int errCode, String errMsg) {

            }
        });
```

8.2. Callback of Ad events

You can call setAdListener() to monitor the behavior of the user's Ad view.

```
public interface IKsyunAdListener {
    // Callback on successful ad display
    void onShowSuccess (String adSlotId);
    // Callback when ad display failed
    void onShowFailed (String adSlotId, int errCode, String errMsg);
    // Ad content is played, generally for video ads
    void onADComplete (String adSlotId);
    // Ad is clicked
    void onADClick (String adSlotId);
    // The ad is closed
    void onADClose (String adSlotId);
}
```

For ads with reward videos, set the setRewardVideoAdListener() interface to check the reward results.

```
public interface IKsyunRewardVideoAdListener {
    // Reward conditions reached
    void onAdAwardSuccess (String adSlotId);

    // Reward conditions not reached
    void onAdAwardFailed (String adSlotId, int errCode, String errMsg)
;
}
```

8.3. Callback of Ad loading events

By setting IKsyunAdloadListener, you can monitor corresponding events for Ad loading.

```
public interface IKsyunAdLoadListener {
    // Ad info loaded successfully, which indicates that we have obtai
ned the Ad's information such as name, urls..., but which does not mea
n that the completion of the download for all video resources
    void onAdInfoSuccess ();

    // Ad info get failed
    void onAdInfoFailed (int errCode, String errMsg);

    // Downloading for ad resources completed, the parameter is the ad
vertising Id
    void onAdLoaded (String adSlotId);
}
```