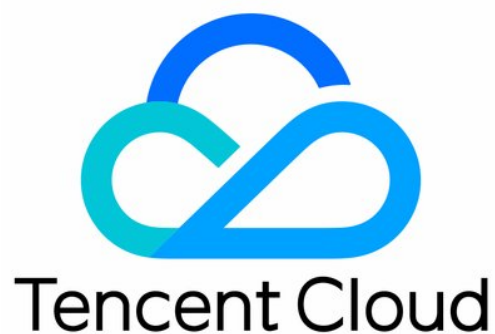


Mobile Live Video Broadcasting

Basic Features

Product Documentation



Copyright Notice

©2013-2019 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Basic Features

SDK Integration

iOS

Android

Camera Push

iOS

Android

Live Pull

iOS

Android

Co-anchoring

RTC

iOS & Android

Basic Features

SDK Integration

iOS

Last updated : 2021-08-13 20:36:48

Basics

This document introduces the live playback feature of the Video Cloud SDK.

Live streaming and video on demand

- In **live streaming**, the video streams published by hosts in real time are the source of streaming. When hosts stop publishing streams, the video at the playback end stops. Since video is streamed in real time, players do not have progress bars when they play live streaming URLs.
- In **video on demand (VOD)**, video files in the cloud are the source of streaming. Videos can be played at any time as long as they are not deleted from the cloud, and the playback progress can be adjusted using the progress bar. Video streaming websites such as Tencent Video and Youku Tudou are typical applications of VOD.

Supported protocols

The table below lists the common protocols used for live streaming. We recommend FLV URLs (which start with `http` and end with `flv`) for LVB and WebRTC for LEB. For more information, please see [Playback \(LEB\)](#).

Protocol	Pro	Con	Playback Latency
FLV	Mature, well adapted to high-concurrency scenarios	SDK integration is required.	2-3s
RTMP	Relatively low latency	Poor performance in high-concurrency scenarios	1-3s
HLS (M3U8)	Well supported on mobile browsers	High latency	10-30s
WebRTC	Lowest latency	SDK integration is required.	< 1s

Note :

LVB and LEB are priced differently. For details, please see [LVB Billing Overview](#) and [LEB Billing Overview](#).

Notes

The Video Cloud SDK **does not impose any limit on the sources of playback URLs**, which means you can use it to play both Tencent Cloud and non-Tencent Cloud URLs. However, the player of the SDK supports only live streaming URLs in FLV, RTMP, HLS (M3U8), and WebRTC formats and VOD URLs in MP4, HLS (M3U8), and FLV formats.

Integration

Step 1. Create a player object

The `V2TXLivePlayer` module in the Video Cloud SDK offers live playback capabilities.

```
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];
```

Step 2. Create a rendering view

In iOS, a view is used as a basic rendering unit. Therefore, you need to configure a view, whose size and position you can adjust, for the player to display video images on.

```
// Use setRenderView to bind a rendering view to the player
[_txLivePlayer setRenderView:_myView];
```

Technically, the player does not render video images directly on the view (`_myView` in the sample code) you provide. Instead, it creates a subview for OpenGL rendering over the view.

You can adjust the size of video images by changing the size and position of the view. The SDK will make changes to the video images accordingly.

How can I make animations?

You are allowed great flexibility in view animation, but note that you need to modify the `transform` rather than `frame` attribute of the view.

```
[UIView animateWithDuration:0.5 animations:^(
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); // Shrink by 1/3
```

```
}};
```

Step 3. Start playback

```
NSString* url = @"http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";  
[_txLivePlayer startPlay:url];
```

Step 4. Change the fill mode

- **setRenderFillMode: aspect fill or aspect fit**

Value	Description
V2TXLiveFillModeFill	Images are scaled to fill the entire screen, and the parts that don't fit are cropped. There are no black bars in this mode, but images may not be displayed in whole.
V2TXLiveFillModeFit	Images are scaled as large as the longer dimension can go. Neither dimension exceeds the screen after scaling. The images are centered, and there may be black bars.

- **setRenderRotation: clockwise rotation of video**

Value	Description
V2TXLiveRotation0	Original
V2TXLiveRotation90	Rotate 90 degrees clockwise
V2TXLiveRotation180	Rotate 180 degrees clockwise
V2TXLiveRotation270	Rotate 270 degrees clockwise

Step 5. Pause playback

Technically speaking, you cannot pause a live playback. In this document, by pausing playback, we mean **freezing video** and **disabling audio**. In the meantime, new video streams continue to be sent to the cloud. When you resume playback, the playback starts from the time of resumption. This is in contrast to VOD. With VOD, when you pause and resume playback, the player behaves the same way as it does when you pause and resume a local video file.

```
// Pause playback  
[_txLivePlayer pauseAudio];  
[_txLivePlayer pauseVideo];  
// Resume playback
```

```
[_txLivePlayer resumeAudio];
[_txLivePlayer resumeVideo];
```

Step 6. Stop playback

```
// Stop playback
[_txLivePlayer stopPlay];
```

Step 7. Take a screenshot

Call **snapshot** to take a screenshot of the live video streamed. You can get the screenshot taken in the [onSnapshotComplete](#) callback of `V2TXLivePlayerObserver`. This method captures a frame of the streamed video. To capture the UI, use the iOS system API.

```
...
[_txLivePlayer setObserver:self];
[_txLivePlayer snapshot];
...
- (void)onSnapshotComplete:(id<V2TXLivePlayer>)player image:(TXImage *)image {
    if (image != nil) {
        dispatch_async(dispatch_get_main_queue(), ^{
            [self handle:image];
        });
    }
}
```

Latency Control

The live playback feature of the SDK is not based on FFmpeg, but Tencent Cloud's proprietary playback engine, which is why the SDK offers better latency control than open-source players do. We provide three latency control modes, which can be used for showrooms, game streaming, and hybrid scenarios.

• Comparison of the three modes

Mode	Stutter	Average Latency	Scenario	Remarks
Speedy	More likely than the speedy mode	2-3s	Live showroom (Chongding Dahui)	The mode delivers low latency and is suitable for latency-sensitive scenarios.

Smooth	Least likely of the three	>= 5s	Game streaming (Penguin Esports)	Playback is least likely to stutter in this mode, which makes it suitable for ultra-high-bitrate streaming of games such as PUBG.
Auto	Self-adaptive to network conditions	2-8s	Hybrid	The better network conditions at the audience end, the lower the latency.

• Code to integrate the three modes

```
// Auto mode
[_txLivePlayer setCacheParams:1 maxTime:5];
// Speedy mode
[_txLivePlayer setCacheParams:1 maxTime:1];
// Smooth mode
[_txLivePlayer setCacheParams:5 maxTime:5];
// Start playback after configuration
```

Note :

For more information on stuttering and latency control, please see Video Stutter

Listening for SDK Events

You can bind a [V2TXLivePlayerObserver](#) to your `V2TXLivePlayer` object to receive callback notifications about the player status, playback volume, first audio/video frame, statistics, warning and error messages, etc.

Periodically triggered notifications

- The [onStatisticsUpdate](#) callback notification is triggered every 2 seconds to update you on the player's status in real time. Like a car's dashboard, the callback gives you information about network conditions, video parameters, etc.

Parameter	Description
appCpu	CPU usage (%) of the app
systemCpu	CPU usage (%) of the system

width	Video width
height	Video height
fps	Frame rate (FPS)
audioBitrate	Audio bitrate (Kbps)
videoBitrate	Video bitrate (Kbps)

- The [onPlayoutVolumeUpdate](#) callback, which notifies you of the player's volume, works only after you call [enableVolumeEvaluation](#) to enable the volume reminder. You can set the interval of the callback by specifying the `intervalMs` parameter when calling `enableVolumeEvaluation`.

Event-triggered notifications

Other callbacks are triggered when specific events occur.

Android

Last updated : 2021-08-13 20:33:14

This document describes how to quickly integrate Tencent Cloud LiteAVSDK for Android into your project.

Environment Requirements

- Android Studio 2.0 or above
- Android 4.1 (SDK API level 16) or above

Integrating the SDK (AAR)

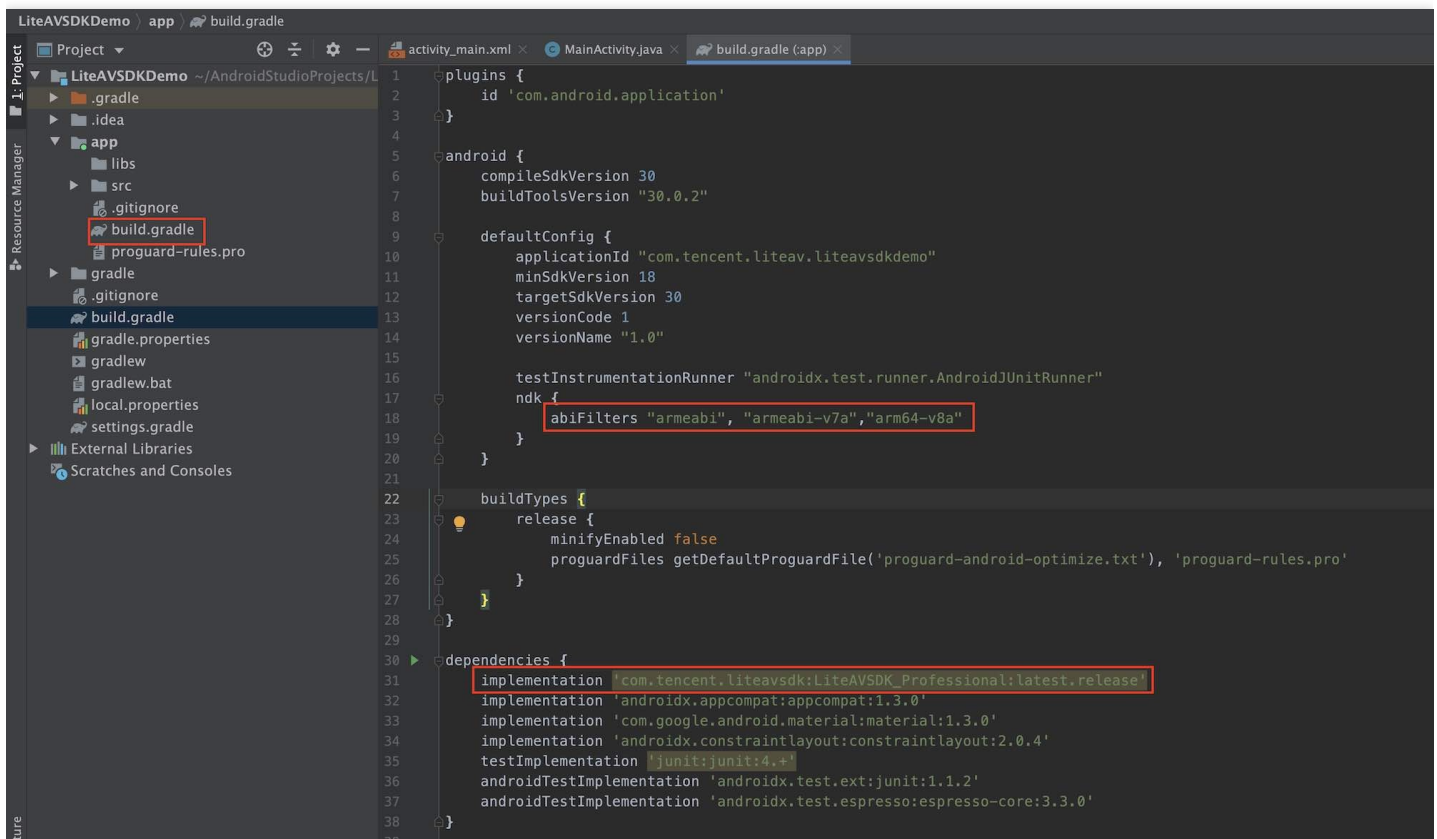
You can use Gradle to automatically load the AAR file or manually download the AAR file and import it into your project.

Method 1: automatic loading (AAR)

Since JCenter has been deprecated, you can configure a Maven Central repository in Gradle to automatically download and update LiteAVSDK.

Open your project with Android Studio and modify the `build.gradle` file as described below to

complete the integration.



1. Add the LiteAVSDK dependency to `dependencies`.

```
dependencies {
    implementation 'com.tencent.liteavsdemo:LiteAVSDK_Professional:latest.release'
}
```

Or

```
dependencies {
    implementation 'com.tencent.liteavsdemo:LiteAVSDK_Professional:latest.release@aar'
}
```

2. In `defaultConfig`, specify the CPU architecture to be used by the application. Currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a.

```
defaultConfig {
    ndk {
        abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
    }
}
```

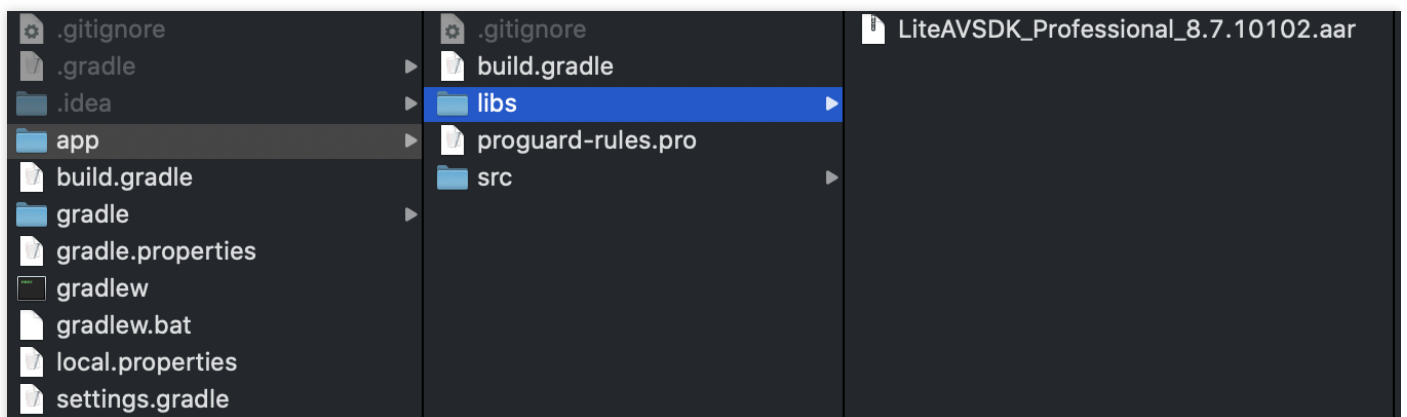


- Click the **Sync Now** button to sync the SDK. If you have no problem accessing Maven Central, the SDK will be downloaded and integrated into your project automatically.

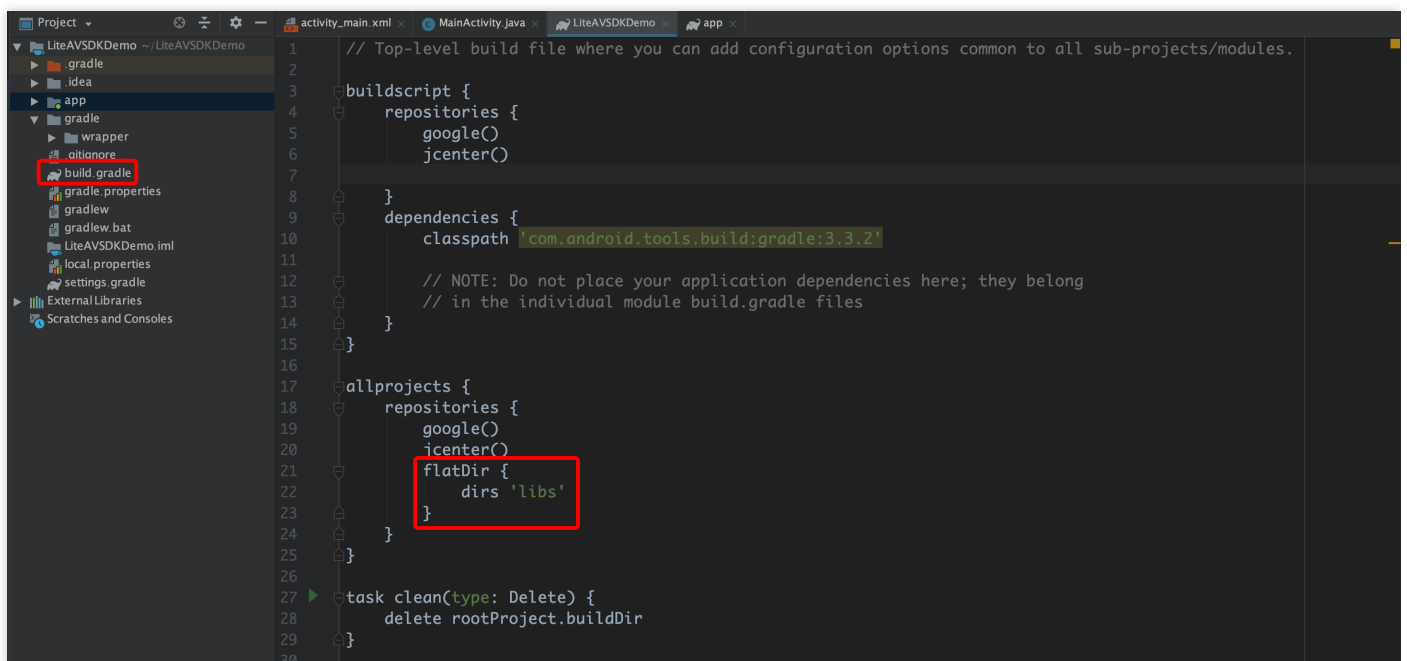
Method 2: manual download (AAR)

If you have problem accessing Maven Central, you can manually download the SDK and integrate it into your project.

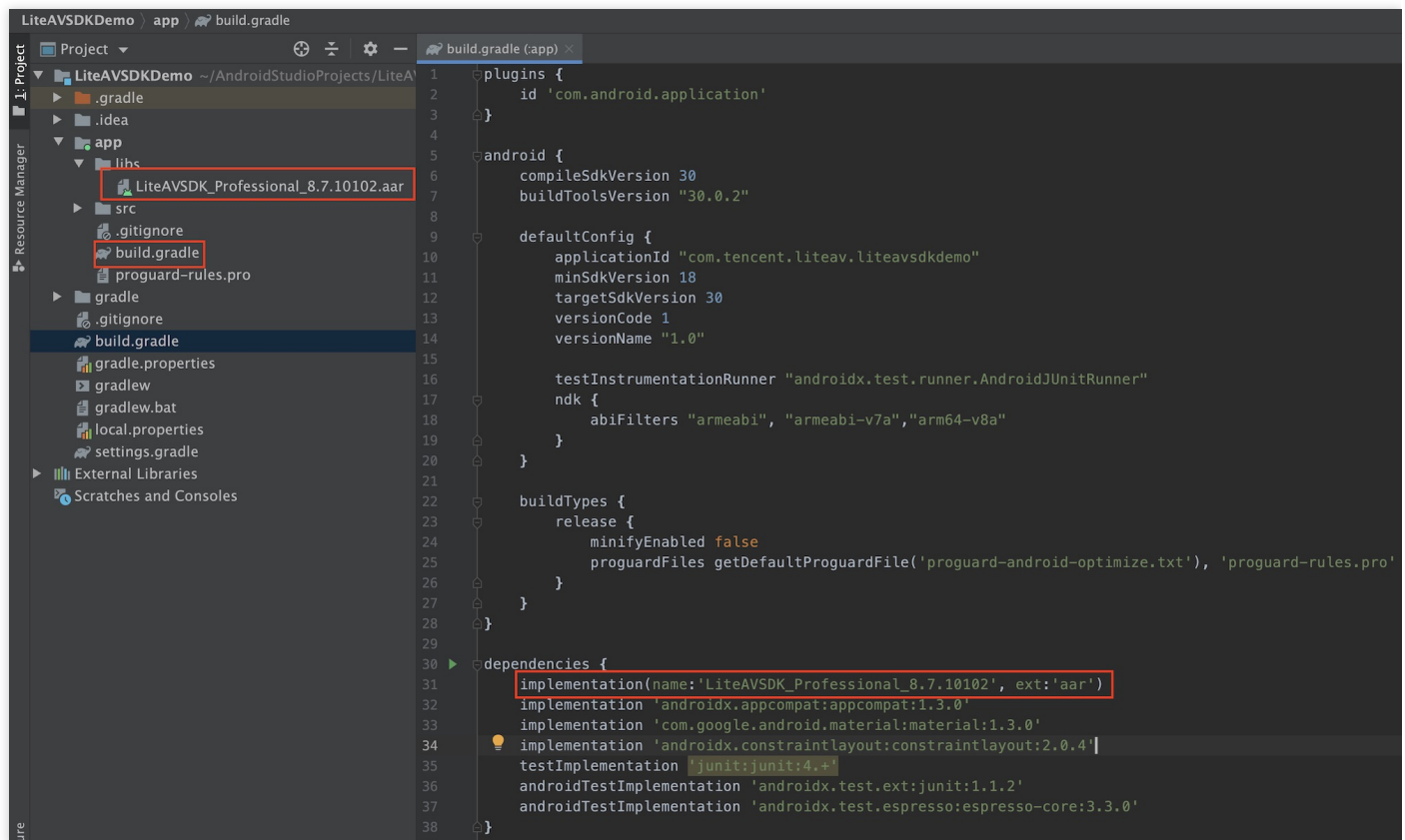
- Download [LiveAVSDK](#) and decompress the file.
- Copy the AAR file in the SDK directory to the **app/libs** directory of your project.



- Add **flatDir** to `build.gradle` under the project's root directory and specify a local path for the repository.



4. Add the LiteAVSDK dependency and, in `app/build.gradle`, add code that references the AAR file.



```
implementation(name:'LiteAVSDK_Professional_8.7.10102', ext:'aar')
```

5. In `defaultConfig` of `app/build.gradle`, specify the CPU architecture to be used by the application. Currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a.

```

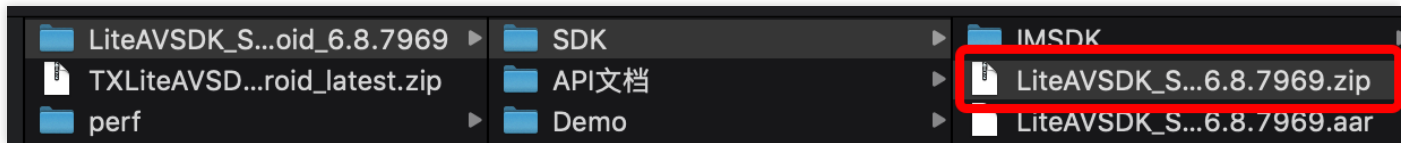
defaultConfig {
    ndk {
        abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
    }
}
  
```

6. Click **Sync Now** to complete the integration of LiteAVSDK.

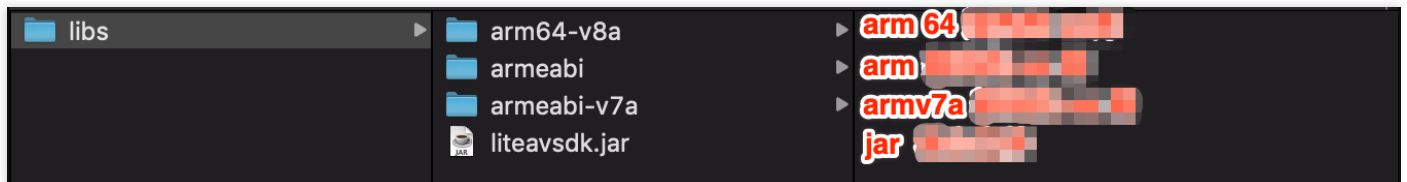
Integrating the SDK (JAR)

If you do not want to import the AAR library, you can also integrate LiteAVSDK by importing JAR and SO libraries.

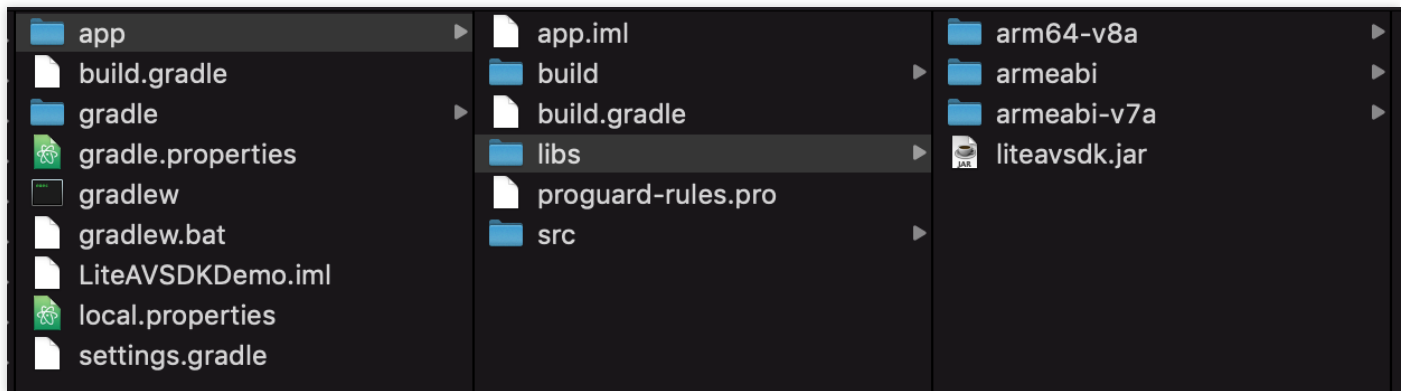
1. Download [LiveAVSDK](#) and decompress the file. In the SDK directory, find `LiteAVSDK_Smart_xxx.zip` (`xxx` indicates the version number of LiteAVSDK).



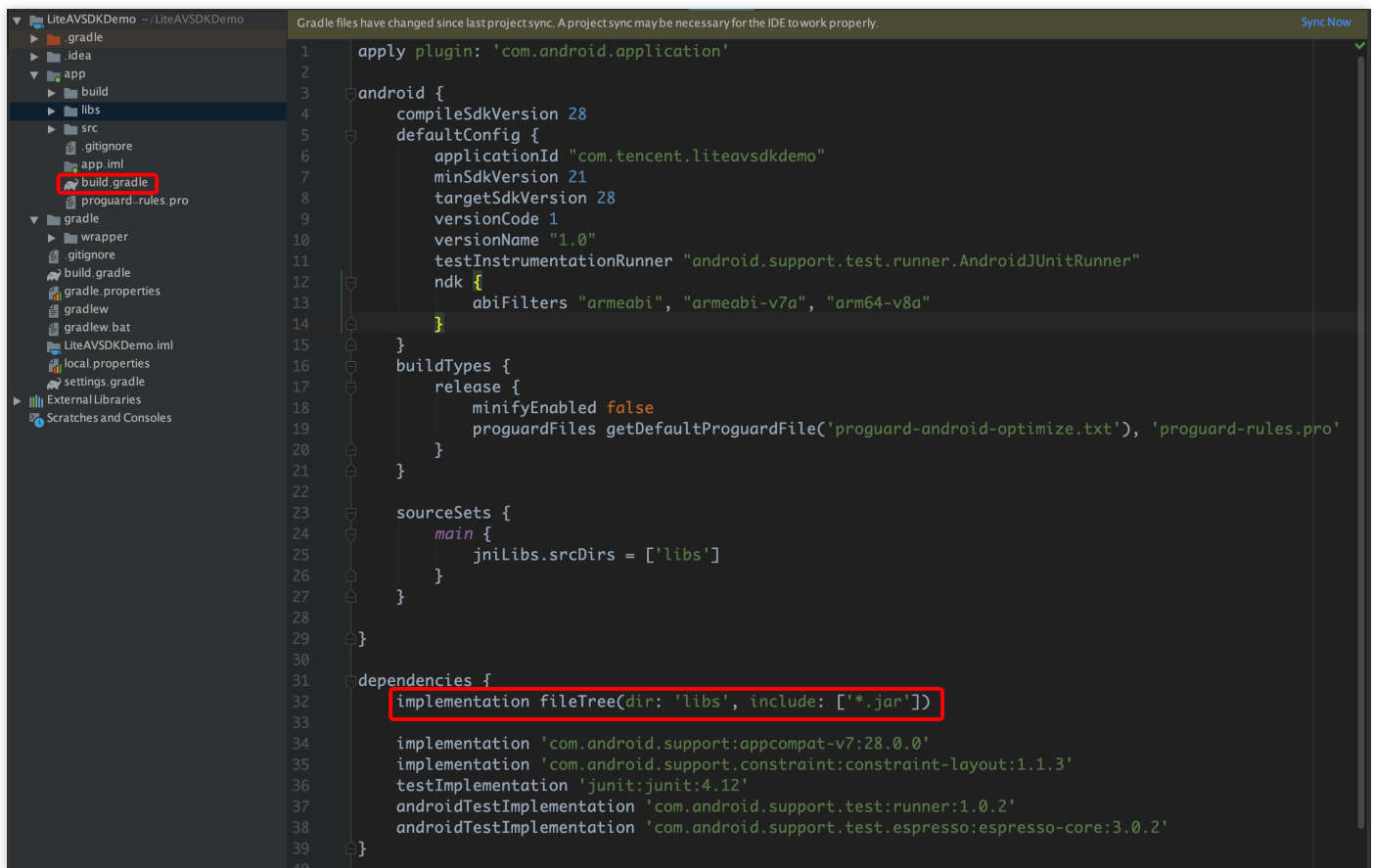
Decompress the file, and you will find a `libs` directory that contains a JAR file and several SO folders, as shown below:



2. Copy the JAR file and `armeabi` , `armeabi-v7a` , and `arm64-v8a` folders to the `app/libs` directory.

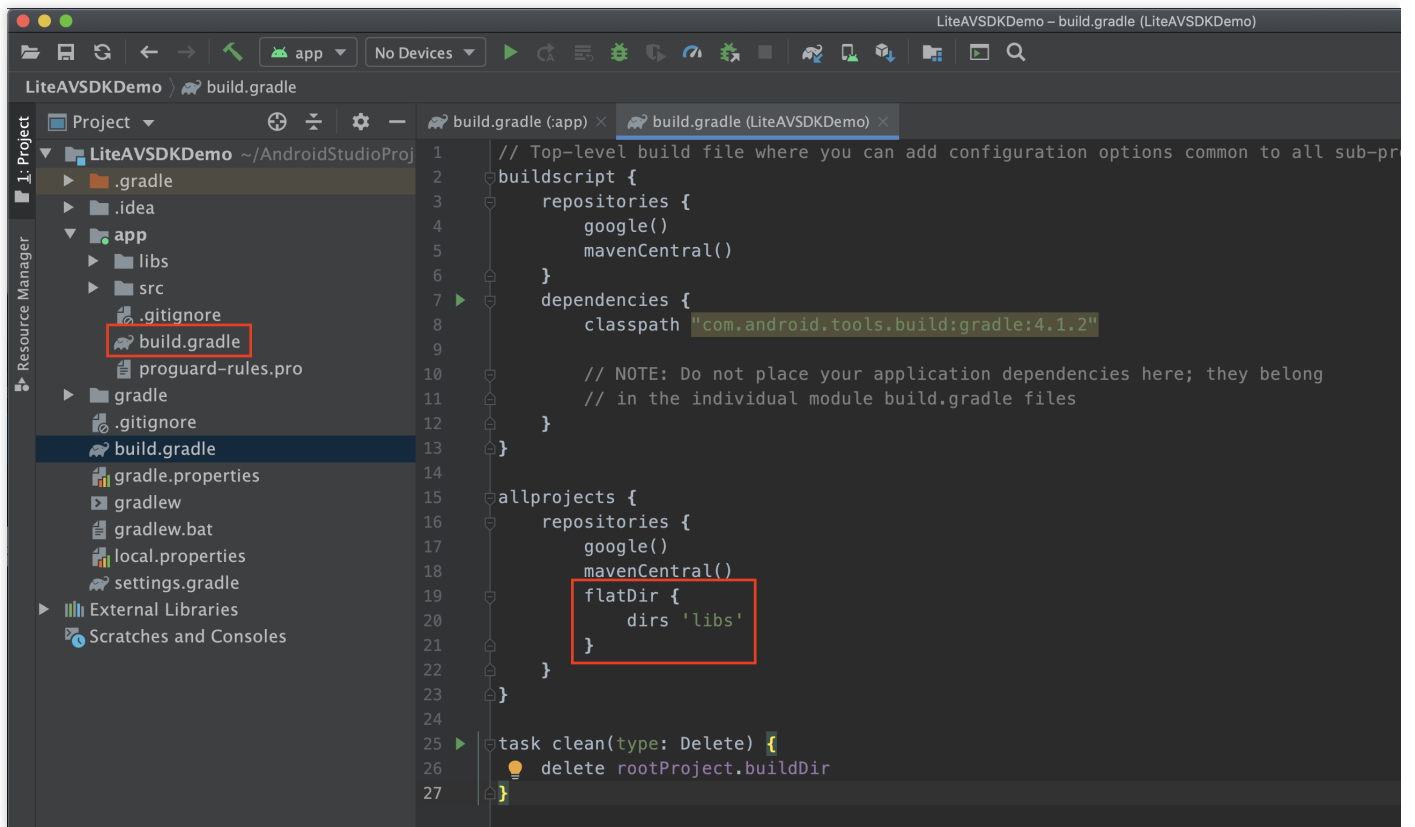


3. Add code that references the JAR library in `app/build.gradle`.

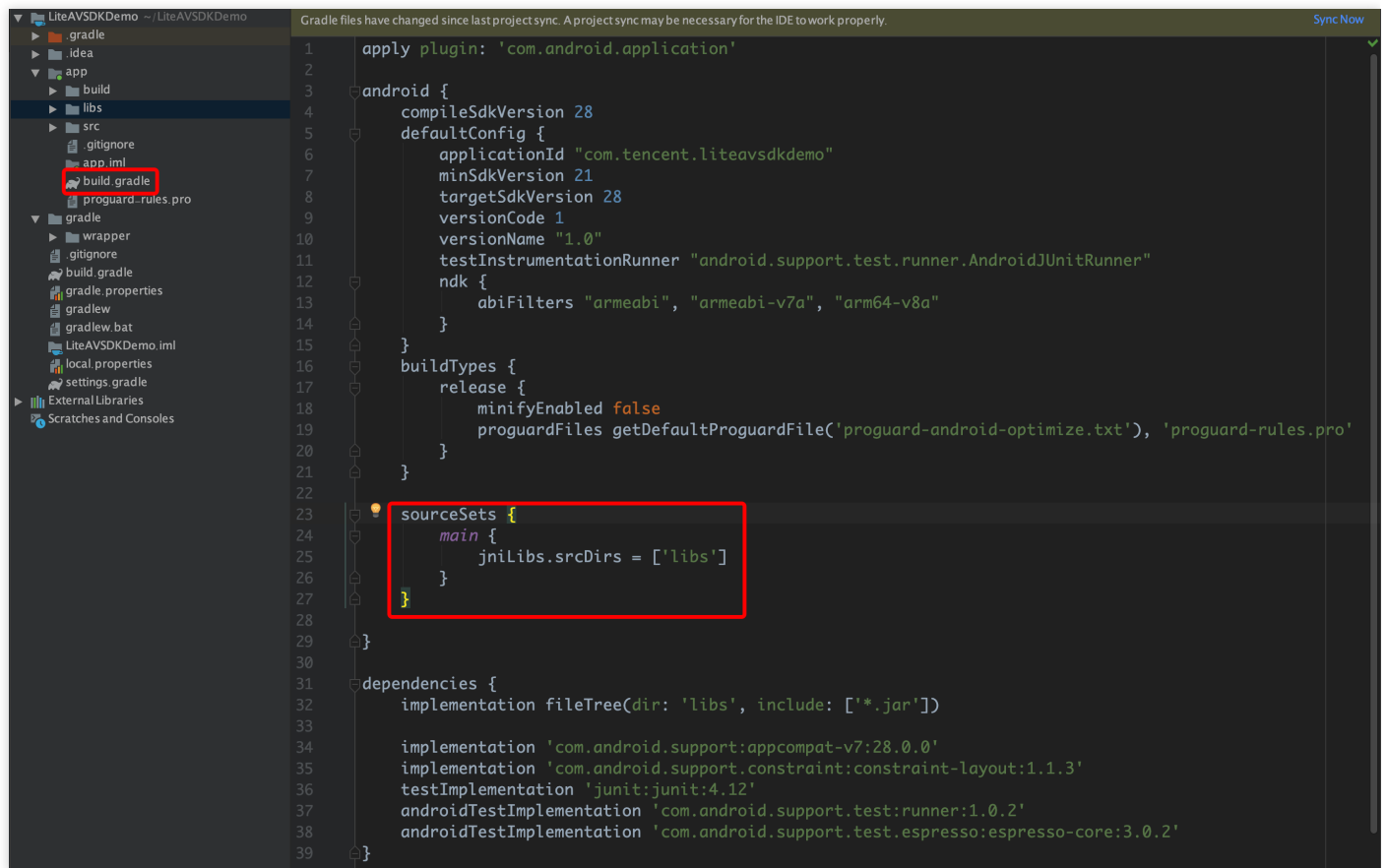


```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
}
```

4. Add **flatDir** to `build.gradle` under the project's root directory and specify a local path for the repository.



5. In `app/build.gradle`, add code that references the SO libraries.

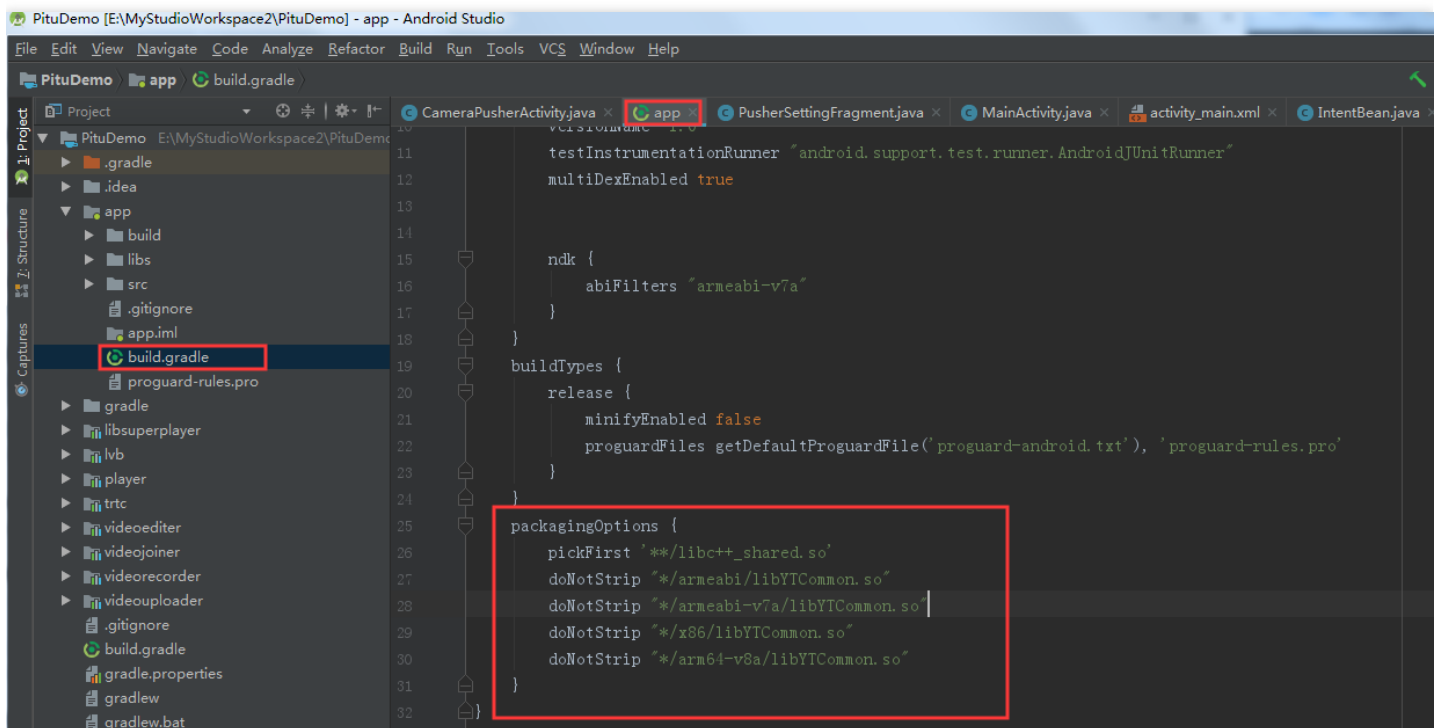


6. In `defaultConfig` of `app/build.gradle`, specify the CPU architecture to be used by the application. Currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a.

```
defaultConfig {
    ndk {
        abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
    }
}
```

7. Click **Sync Now** to complete the integration.

Setting Packaging Parameters



```
packagingOptions {
    pickFirst '**/libc++_shared.so'
    doNotStrip "**/armeabi/libYTCommon.so"
    doNotStrip "**/armeabi-v7a/libYTCommon.so"
    doNotStrip "**/x86/libYTCommon.so"
    doNotStrip "**/arm64-v8a/libYTCommon.so"
}
```

Configuring Permissions

Configure permissions for your application in `AndroidManifest.xml`. LiteAVSDK needs the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

```
<uses-feature android:name="android.hardware.Camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Configuring License

Click [Get License](#) to obtain a trial license. For more information, please see [Applying for a Trial License](#). You will get two strings: a license URL and a decryption key.

Before you use the features of the Enterprise Edition SDK in your application, complete the following configurations (preferably in the application class).

```
public class MApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // The license URL obtained
        String licenceKey = ""; // The license key obtained
        TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
    }
}
```

Configuring Obfuscation Rules

In the `proguard-rules.pro` file, add LiteAVSDK-related classes to the "do not obfuscate" list.

```
-keep class com.tencent.** { *; }
```

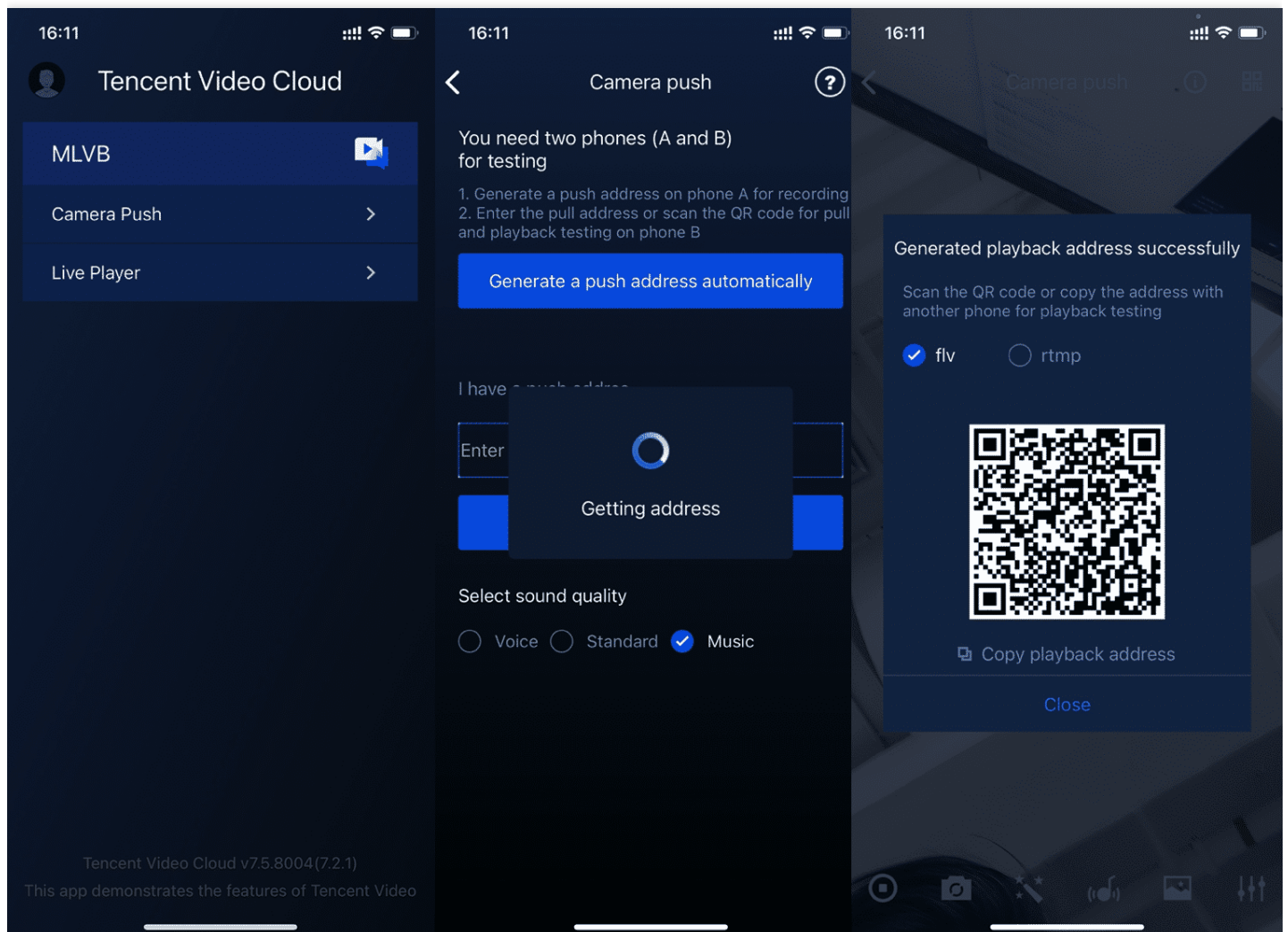
Camera Push

iOS

Last updated : 2020-12-09 11:10:06

Feature

Camera push refers to the process of collecting images from the camera of the mobile phone and voice from the microphone, encoding the video and audio data, and pushing the data to the livestreaming cloud platform. Tencent Cloud LiteAVSDK calls the V2TXLivePusher API to provide the camera push capability. The figure below shows the interfaces for camera push operations demonstrated in the LiteAVSDK demo.



Notes

• x86 simulator debugging

The SDK uses a lot of audio and video APIs of the iOS system. These APIs often cannot be used on the x86 simulator that comes with the Mac server. Therefore, if conditions allow, we recommend that you use the real Mac server for debugging.

Feature Interfacing

1. Download the SDK

[Download](#) the SDK and follow the instructions in the [SDK integration guide](#) to embed the SDK in your application project.

2. Configure a license for the SDK

Click [Apply for a License](#) to obtain the license for testing. You will receive two strings. One is the license URL, and the other is the decryption key.

Before you call the LiteAVSDK features in your app, we recommend that you complete the following configurations in `- [AppDelegate application:didFinishLaunchingWithOptions:]` :

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    NSString * const licenceURL = @"<Obtained license URL>";
    NSString * const licenceKey = @"<Obtained key>";

    //TXLiveBase is located in the "TXLiveBase.h" header file.
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

3. Initialize the V2TXLivePusher component

First, create a `V2TXLivePusher` instance. Later, this instance can be used to implement all the following push-related capabilities, including starting or stopping stream push, starting or stopping data collection from the camera, starting or stopping data collection from the microphone, starting or stopping headphone monitoring, setting the voice quality, and setting the image quality.

```
V2TXLivePusher *_pusher = [[V2TXLivePusher alloc] init];
[_pusher addObserver:self];
```

4. Enable camera preview

You can call the `setRenderView` API in `V2TXLivePusher` to enable the camera preview for the mobile phone. You must provide a view object for previewing images for the `setRenderView` API.

```
// Create a view object and insert it into the current interface.
UIView *localView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view insertSubview:localView atIndex:0];
localView.center = self.view.center;

// Enable the local camera preview.
[_pusher setRenderView:localView];
```

Note :

To add the animation effect for a view, you must modify the `transform` attribute of the view, instead of the `frame` attribute.

```
[UIView animateWithDuration:0.5 animations:^(
    _localView.transform = CGAffineTransformMakeScale(0.3, 0.3); // Shrink by 1/3
)];
```

5. Start and stop stream push

If you have called the `setRenderView` API to enable camera preview, you can call the `startPush` API of `V2TXLivePusher` to start pushing streams.

```
// Start pushing streams.
NSString* rtmpUrl = @"rtmp://test.com/live/xxxxxx"; // Enter your RTMP URL for stream push.
[_pusher startPush:rtmpUrl];
[_pusher startCamera]; // Start collecting data from the camera.
[_pusher startMicrophone]; // Start collecting data from the microphone.
```

After you complete stream push, you can call the `stopPush` API of `V2TXLivePusher` to stop stream push.

```
// Stop pushing streams.
[_pusher addObserver:nil];
```

```
[_pusher stopCamera];  
[_pusher stopMicrophone];  
[_pusher stopPush];
```

- **How can I obtain a valid push URL?**

After you activate the LVB service, you can choose LVB console > Auxiliary tools > [URL generator](#) to generate a push URL. For more information, see [Push/Pull URL](#).

Address Generator

Domain Type * Push domain 28251.livepush.myqcloud.com

If you select push domain, a push address will be generated; and if you select playback domain, a playback address will be generated. If there is no available domain, please [Add Domain](#)

AppName * live

Use "live" by default. Only letters, digits, and symbols are supported.

StreamName * Please enter StreamName

Only support letters, digits, and symbols.

Expiration Time 2020-10-27 17:29:36

The expiration time of playback address is the setting timestamp plus the playback authentication expiration time, and the push address expiration time is the setting time.

[Generate Address](#) [Address Resolution Sample](#)

- **Why is “V2TXLIVE_ERROR_INVALID_LICENSE” returned?**

If the `startPush` API returns “V2TXLIVE_ERROR_INVALID_LICENSE”, license verification has failed. In this case, check whether any problem occurred in [Step 2. Configure a license for the SDK](#).

6. Push pure audio streams

If you only need to push pure audio streams, perform the following operations:

```
// Start pushing streams.  
NSString* rtmpUrl = @"rtmp://test.com/live/xxxxxx"; // Enter your RTMP URL for stream push.  
[_pusher startPush:rtmpUrl];  
[_pusher startMicrophone]; // Start collecting data from the microphone.
```

7. Set the video resolution

You can call the `setVideoQuality:resolutionMode:` API of V2TXLivePusher to set the video resolution, aspect ratio, and portrait or landscape mode for the audience.

```
// Select the portrait mode and set the aspect ratio to 1280x720. The first parameter is the vide  
o resolution, and the second parameter is the portrait or landscape mode.
```

```
[_pusher setVideoQuality:V2TXLiveVideoResolution_1280x720 resolutionMode:V2TXLiveVideoResolutionMode_Portrait];
```

8. Set the beauty filter, whitening, and rosy skin effects.

The beauty filter SDK's `TCBeautyPanel.framework` can be introduced to pass in the `V2TXLivePusher` instance and display the beauty panel (`TCBeautyPanel`) on the interface. This allows users to set diversified beauty effects.

```
NSInteger controlHeight = [TCBeautyPanel getHeight];
CGRect frame = CGRectMake(0, 0, self.view.frame.size.width, controlHeight);
_beautyPanel = [TCBeautyPanel beautyPanelWithFrame:frame
SDKObject:_livePusher];
[ThemeConfigurator configBeautyPanelTheme:_beautyPanel];
_beautyPanel.pituDelegate = self; // Proxy method that is used to set and implement the beauty filter effects.
[_beautyPanel resetAndApplyValues];

#pragma mark - BeautyLoadPituDelegate

- (void)onLoadPituStart {
dispatch_async(dispatch_get_main_queue(), ^{
[self showInProgressText:@"Start loading resources"];
});
}

- (void)onLoadPituProgress:(CGFloat)progress {
dispatch_async(dispatch_get_main_queue(), ^{
[self showInProgressText:[NSString stringWithFormat:@"Loading resources %d %%", (int)(progress * 100)]];
});
}

- (void)onLoadPituFinished {
dispatch_async(dispatch_get_main_queue(), ^{
[self showText:@"Successfully loaded resources"];
});
}

- (void)onLoadPituFailed {
dispatch_async(dispatch_get_main_queue(), ^{
[self showText:@"Failed to load resources"];
});
}
```


9. Control camera behaviors

V2TXLivePusher provides an API to obtain the video device manager, TXVideoDeviceManager, which is used to control camera behaviors.

API Function	Description	Remarks
switchCamera	Switch between the front and rear cameras.	The corresponding function on the Mac platform is <code>selectCamera</code> .
enableCameraFlash	Enable or disable the flash.	This function is valid only when the current camera is a rear camera.
setCameraZoomRatio	Adjust the zoom ratio.	The valid range of the zoom ratio is 1 - 5. Default: 1.
setCameraFocusPosition	Set the focus position.	To use this setting, <code>enableCameraAutoFocus</code> must be used to disable auto focus.

10. Set the mirror effect on the audience side

You can call the `setEncoderMirror` API of V2TXLivePusher to set the mirror effect on the audience side. This mirror effect is different from that on the host side. When the host uses the front camera for livestreaming, the view seen by the host is inverted by the SDK by default. Therefore, the host's display is like looking in a mirror. `setEncoderMirror` only affects the mirror effect on the audience side.

11. Set the audio quality

```
// Pay attention to the calling sequence:  
[_pusher setAudioQuality: V2TXLiveAudioQuality_Music]; // You must set the audio quality before pushing streams. Otherwise, the audio quality setting does not take effect.  
[_pusher startPush]; // 2
```

12. Set the logo watermark

You can call the `setWatermark:position:scale:` API of V2TXLivePusher to allow the SDK to add a watermark to the pushed video stream. The watermark position is determined by the `position` parameter.

- The SDK requires that the watermark image be in png format, instead of jpg, because the png format provides the transparency information and allows the SDK to better address the image aliasing issue. If a jpg image is modified in the Windows system, its extension does not take effect.

- `position` is the normalized coordinates of the watermark image relative to the resolution of the pushed video. If the resolution of the pushed video is 540×960 , and `position` is set to (0.1, 0.1, 0.1, 0.0), the actual pixel coordinates of the watermark are: 540×0.1 , 960×0.1 , Watermark width $\times 0.1$, Automatically calculated watermark height.

```
UIImage *image = value?[UIImage imageNamed:@"watermark"]:nil;  
CGPoint pos = value?CGPointMake(10, 10):CGPointZero;  
[_pusher setWatermark:image position:pos scale:1.0];
```

Event Handling

1. Event listening

The SDK uses the `V2TXLivePusherObserver` proxy to set the pusher callback. By setting the callback, you can monitor some callback events of V2TXLivePusher, including the player status, volume callback, statistics, warnings, and error messages.

2. Normal events

The table below lists the events you will be notified of upon each successful push. If "0" is received, the related event is called successfully.

Event ID	Value	Description
V2TXLIVE_OK	0	Successfully called the push event.

3. Error notifications

The push cannot continue because the SDK detected a critical problem. For example, when the user revokes the camera permission for the app, the camera cannot be started.

Event ID	Value	Description
V2TXLIVE_ERROR_FAILED	-1	Failed.
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	Invalid parameter.
V2TXLIVE_ERROR_REFUSED	-3	Refused.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	Not supported.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Invalid license.

4. Warning events

The SDK detects some warning events. These events can trigger tentative protection logic or restoration logic. Warnings can often be recovered from.

Event ID	Value	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Poor network connection. Data upload is blocked because the upstream bandwidth is too low.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Video lag occurs.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to start the camera. This can occur because the camera configuration program (driver) on a Windows or Mac device is abnormal. To solve this problem, disable and then enable the device again, restart the device, or update the configuration program.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1308	Camera occupied. In this case, try to enable another camera.
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No permission to access the camera. This error often occurs on a mobile device when the permission is revoked by the user.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to start the microphone. This can occur because the microphone configuration program (driver) on a Windows or Mac device is abnormal. To solve this problem, disable and then enable the device again, restart the device, or update the configuration program.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	Microphone occupied. For example, if a mobile device has an ongoing call, the attempt to start the microphone will fail.

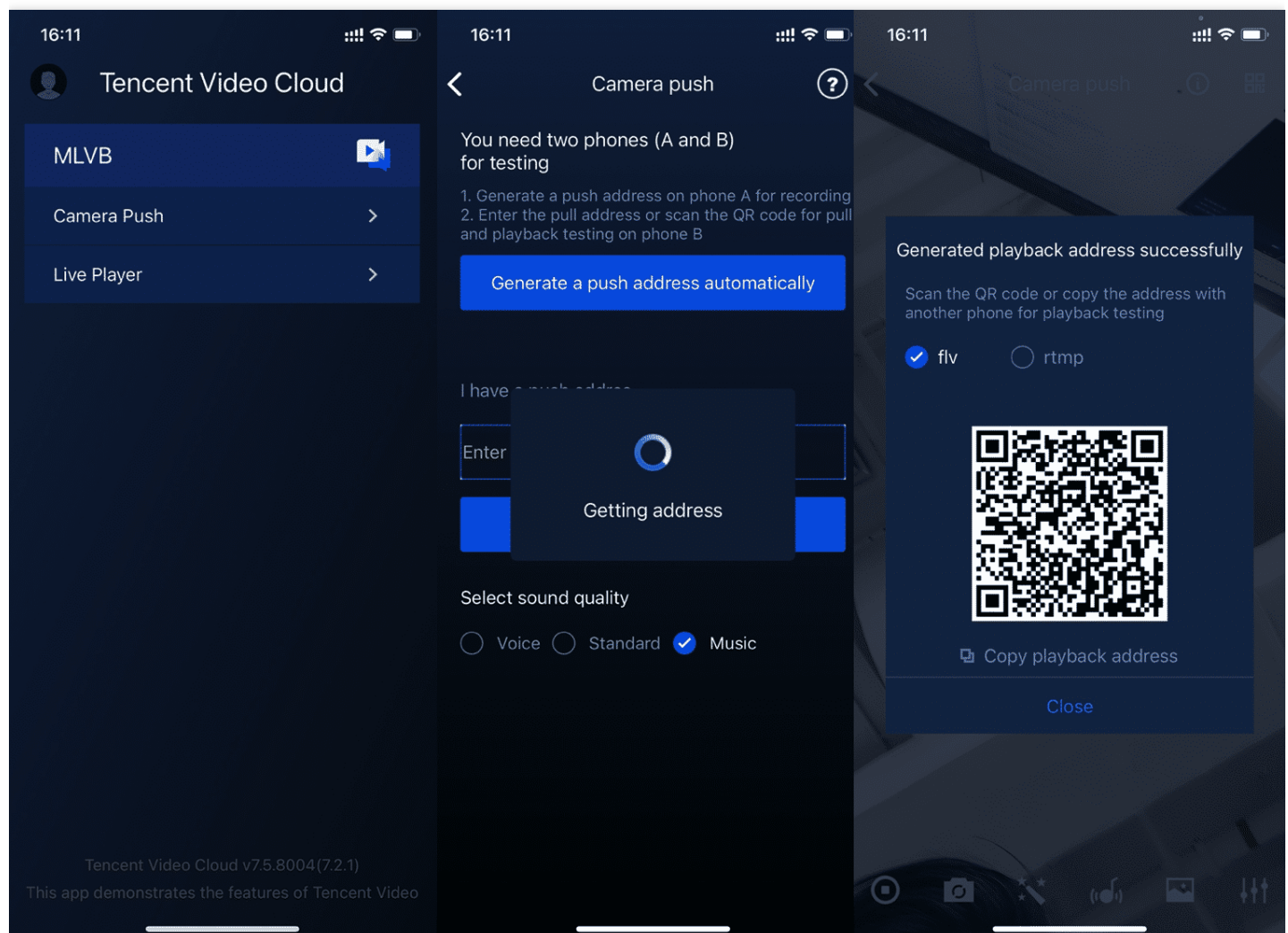
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No permission to access the microphone. This error often occurs on a mobile device when the permission is revoked by the user.
---	-------	--

Android

Last updated : 2020-12-09 11:14:33

Feature

Camera push refers to the process of collecting images from the camera of the mobile phone and voice from the microphone, encoding the video and audio data, and pushing the data to the livestreaming cloud platform. Tencent Cloud LiteAVSDK calls the `V2TXLivePusher` API to provide the camera push capability. The figure below shows the interfaces for camera push operations demonstrated in the LiteAVSDK demo.



Feature Interfacing

1. Download the SDK

Download the SDK and follow the instructions in the [SDK integration guide](#) to embed the SDK in your application project.

2. Configure a license for the SDK

Click [Apply for a License](#) to obtain the license for testing. You will receive two strings. One is the license URL, and the other is the decryption key.

Before you call the LiteAVSDK features in your app, we recommend that you complete the following configurations in `Application` :

```
public class MApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // Obtained licence URL
        String licenceKey = ""; // Obtained licence key
        TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
    }
}
```

3. Initialize the V2TXLivePusher component

First, create a `V2TXLivePusher` instance. Later, this instance can be used to implement all the following push-related capabilities, including starting or stopping stream push, starting or stopping data collection from the camera, starting or stopping data collection from the microphone, starting or stopping headphone monitoring, setting the voice quality, and setting the image quality.

```
V2TXLivePusher txLivePusher = new V2TXLivePusherImpl(mContext);
```

4. Enable camera preview

You can call the `setRenderView` API in `V2TXLivePusher` to enable the camera preview for the mobile phone. You must provide a view object for previewing images for the `setRenderView` API.

```
<com.tencent.rtmp.ui.TXCloudVideoView
    android:id="@+id/video_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Set render view:

```
TXCloudVideoView videoView = (TXCloudVideoView) findViewById(R.id.video_view);
txLivePusher.setRenderView(videoView);
```

5. Start and stop stream push

If you have called the `setRenderView` API to enable camera preview, you can call the `startPush` API of `V2TXLivePusher` to start pushing streams.

```
// Enter your RTMP URL for stream push.
String rtmpURL = "rtmp://test.com/live/xxxxxx";
// Start collecting data from the camera.
txLivePusher.startCamera();
// Start collecting data from the microphone.
txLivePusher.startMicrophone();
// Start push streams.
txLivePusher.startPush(rtmpURL);
```

After you complete stream push, you can call the `stopPush` API of `V2TXLivePusher` to stop stream push.

```
// Stop pushing streams
txLivePusher.stopPush();
```

• How can I obtain a valid push URL?

After you activate the LVB service, you can choose LVB console > Auxiliary tools > [URL generator](#) to generate a push URL. For more information, see [Push/Pull URL](#).

Address Generator

Domain Type *

Push domain ▼

28251.livepush.myqcloud.com ▼

If you select push domain, a push address will be generated; and if you select playback domain, a playback address will be generated. If there is no available domain, please [Add Domain](#)

AppName *

live

Use "live" by default. Only letters, digits, and symbols are supported.


StreamName *

Please enter StreamName

Only support letters, digits, and symbols.

Expiration Time

2020-10-27 17:29:36



The expiration time of playback address is the setting timestamp plus the playback authentication expiration time, and the push address expiration time is the setting time.

Generate Address

[Address Resolution Sample](#)

• Why is “-5” returned?

If the `startPush` API returns “-5”, license verification has failed. In this case, check whether any problem occurred in [Step 2. Configure a license for the SDK](#).

6. Push pure audio streams

If you only need to push pure audio streams, perform the following operations:

```
String rtmpURL = "rtmp://test.com/live/xxxxxx"; // Enter your RTMP URL for stream push.
// Start collecting data from the microphone.
txLivePusher.startMicrophone();
// Start push streams.
txLivePusher.startPush(rtmpURL);
```

7. Set the video resolution

You can call the `setVideoQuality` API of `TXLivePusherV2` to set the video resolution, aspect ratio, and portrait or landscape mode for the audience.

```
// Select the portrait mode and set the aspect ratio to 1280x720. The first parameter is the video resolution, and the second parameter is the portrait or landscape mode.
txLivePusher.setVideoQuality(V2TXLiveVideoResolution_1280x720, V2TXLiveVideoResolutionMode_Portrait];
```

8. Set the beauty filter, whitening, and rosy skin effects.

```
/*
 * With the beauty manager, you can use the following features:
 * - Set the following cosmetic effects: beauty style, whitening, ruddy, big eyes, slim face, V-shape face, chin, short face, small nose, bright eyes, white teeth, remove eye bags, remove wrinkles, remove laugh lines.
 * - Adjust the hairline, eye spacing, eye corners, mouth shape, nose wings, nose position, lip thickness, and face shape.
 * - Set animated effects such as face widgets (materials).
 * - Add makeup effects.
 * - Recognize gestures.
 */
TXBeautyManager beautyManager = txLivePusher.getBeautyManager();
beautyManager.setBeautyLevel(6);
```

9. Control camera behaviors

`TXLivePusherV2` provides an API to obtain the video device manager, `TXVideoDeviceManager`, which is used to control camera behaviors.

API Function	Description	Remarks
switchCamera	Switch between the front and rear cameras.	
enableCameraFlash	Enable or disable the flash.	This function is valid only when the current camera is a rear camera.
getCameraZoomMaxRatio	Get max zoom ratio.	
setCameraZoomRatio	Adjust the zoom ratio.	The valid range of the zoom ratio is 1 - <code>getCameraZoomMaxRatio</code> . Default: 1.
setCameraFocusPosition	Set the focus position.	To use this setting, <code>enableCameraAutoFocus</code> must be used to disable auto focus.

10. Set the mirror effect on the audience side

You can call the `setEncoderMirror` API of `V2TXLivePusher` to set the mirror effect on the audience side. This mirror effect is different from that on the host side. When the host uses the front camera for livestreaming, the view seen by the host is inverted by the SDK by default. Therefore, the host's display is like looking in a mirror. `setEncoderMirror` only affects the mirror effect on the audience side.

11. Set the audio quality

```
// Pay attention to the calling sequence :  
// You must set the audio quality before pushing streams. Otherwise, the audio quality setting does not take effect.  
txLivePusher.setAudioQuality(V2TXLiveDef.V2TXLiveAudioQuality.V2TXLiveAudioQuality_Music);  
txLivePusher.startPush();
```

12. Set the background music (BGM), voice changing effect, and reverb effect.

```
/**  
 * With the audio effect manager, you can use the following features:  
 * - Adjust the volume of human voice collected by the microphone.  
 * - Set the reverb and voice changing effects.  
 * - Start the headphone monitor, and set the volume of the headphone monitor.  
 * - Add the BGM, and adjust the playback effect of BGM.  
 */  
TXAudioEffectManager manager = txLivePusher.getAudioEffectManager();
```

13. Set the logo watermark

You can call the `setWatermark` API of `V2TXLivePusher` to allow the SDK to add a watermark to the pushed video stream.

- The SDK requires that the watermark image be in png format, instead of jpg, because the png format provides the transparency information and allows the SDK to better address the image aliasing issue. If a jpg image is modified in the Windows system, its extension does not take effect.
- `x, y` is the normalized coordinates of the watermark image relative to the resolution of the pushed video. If the resolution of the pushed video is 540×960 , and set to $(0.1, 0.1, 0.1, 0.0)$, the actual pixel coordinates of the watermark are: 540×0.1 , 960×0.1 , Watermark width $\times 0.1$, Automatically calculated watermark height.

```
Bitmap bitmap = decodeResource(getResources(), R.drawable.filter_water);
txLivePusher.setWatermark(bitmap, 0.1f, 0.1f, 0.1f);
```

Event Handling

1. Event listening

The SDK uses the `V2TXLivePusherObserver` proxy to set the pusher callback. By setting the callback, you can monitor some callback events of `V2TXLivePusher`, including the player status, volume callback, statistics, warnings, and error messages.

```
txLivePusher.setObserver(new V2TXLivePusherObserver());
```

2. Normal events

The table below lists the events you will be notified of upon each successful push. If "0" is received, the related event is called successfully.

Event ID	Value	Description
V2TXLIVE_OK	0	Successfully called the push event.

3. Error events

The push cannot continue because the SDK detected a critical problem. For example, when the user revokes the camera permission for the app, the camera cannot be started.

Error ID	Value	Description
----------	-------	-------------

V2TXLIVE_ERROR_FAILED	-1	Failed.
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	Invalid parameter.
V2TXLIVE_ERROR_REFUSED	-3	Refused.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	Not supported.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Invalid license.

```
@Override
public void onError(int code, String msg, Bundle extraInfo) {
    // doSomething
}
```

4. Warning events

The SDK detects some warning events. These events can trigger tentative protection logic or restoration logic. Warnings can often be recovered from.

Warning ID	Value	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Poor network connection. Data upload is blocked because the upstream bandwidth is too low.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Video lag occurs.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to start the camera.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	Camera occupied.
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No permission to access the camera. This error often occurs on a mobile device when the permission is revoked by the user.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to start the microphone.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	Microphone occupied. For example, if a mobile device has an ongoing call, the attempt to start the microphone will fail.

V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No permission to access the microphone. This error often occurs on a mobile device when the permission is revoked by the user.
---	-------	--

```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
    // doSomething
}
```

Live Pull

iOS

Last updated : 2020-10-30 16:53:30

Basics

This document introduces the livestream playback function of Video Cloud SDK.

Livestreaming

- The **livestreaming** video source is pushed by the host in real time. When the host stops pushing, the video view on the player stops. In addition, the video is broadcast in real time, and no progress bar is displayed when the player is playing the livestreaming URL.

Supported protocols

Common livestreaming protocols are listed below. We recommend that you use an FLV-based livestreaming URL that starts with "http" and ends with ".flv" on apps.

Livestreaming Protocol	Advantage	Disadvantage	Playback Delay
FLV	High maturity, high concurrency, and no pressure	The video can be played only after the SDK is integrated.	2s - 3s
RTMP	Lowest theoretical delay when using high-quality lines	The performance is poor in the case of high concurrency.	1s - 3s

Notes

- Are there any restrictions?**

The Tencent Video Cloud SDK **does not** impose any limits on the source of the playback URL, which means it is available for both Tencent Cloud and non-Tencent Cloud playback URLs. However, the player in the Tencent Video Cloud SDK only supports livestreaming URLs in FLV and RTMP formats.

Interfacing

Step 1: create a player

The V2TXLivePlayer module in the Tencent Video Cloud SDK implements the livestream playback feature.

```
V2TXLivePlayer *txLivePlayer = [[V2TXLivePlayer alloc] init];
```

Step 2: render a view

Find a place to display the video images in the player. In the iOS system, a view is used as the basic rendering unit. Therefore, you simply need to prepare a view and configure the layout.

```
[txLivePlayer setRenderView:_myView];
```

Technically, the player does not directly render the video image to the view (`_myView` in the sample code) you provide. Instead, it creates a subView used for OpenGL rendering on top of the view.

You can adjust the size of the rendered image simply by adjusting the size and position of the view. The SDK will automatically adapt the video images to the size and position of the view.

How can I make animations?

You can add animation effects for a view as desired. But note that you must modify the `transform` attribute of the view, instead of the `frame` attribute.

```
[UIView animateWithDuration:0.5 animations:^(
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); // Shrink by 1/3
)];
```

Step 3: start livestream playback

```
NSString* flvUrl = @"http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
[txLivePlayer startPlay:flvUrl];
```

Step 4: adjust the view

• view: size and position

You can modify the size and position of the view by adjusting the size and position of the parameter `view` of `setupVideoWidget`. The SDK will automatically adjust the size and position of the view based on your configuration.

• setRenderFillMode: Fill or Fit

Option	Description
V2TXLiveFillMode_Fill	Fill the image on the screen without leaving any black edges. If the aspect ratio of the view is different from that of the screen, part of the view content will be cropped.
V2TXLiveFillMode_Fit	Make the view fit the screen without cropping. If the aspect ratio of the view is different from that of the screen, black edges will appear.

- **setRenderRotation: view rotation**

Option	Description
V2TXLiveRotation_0	Do not rotate the view.
V2TXLiveRotation_90	Rotate 90 degrees clockwise.
V2TXLiveRotation_180	Rotate 180 degrees clockwise.
V2TXLiveRotation_270	Rotate 270 degrees clockwise.

Step 5: pause playback

Strictly speaking, you cannot pause livestream playback. Here, pausing livestream playback actually means **freezing the image** and **turning off the sound**, but the video source keeps updating on the cloud. When you resume the playback, the video is resumed from the latest time. This is the biggest difference between livestreaming and VOD. In the VOD service, the video is paused and resumed in the same way as a local video file.

```
// Pause the audio.  
[txLivePlayer pauseAudio];  
// Resume the audio.  
[txLivePlayer resumeAudio];  
  
// Pause the video.  
[txLivePlayer pauseVideo];  
// Resume the video.  
[txLivePlayer resumeVideo];
```

Step 6: stop livestream playback

Call **stopPlay** to stop livestream playback.

```
// Stop playback.  
[txLivePlayer stopPlay];
```

Step 7: take snapshots

You can capture the current image as a frame by calling **snapshot**. This feature can only capture frames from the current livestreaming video. To capture the entire UI, you must call the API of the iOS system.

```
// Callback notification of the Tencent Cloud video call feature  
- (V2TXLiveCode)snapshot:(id<V2TXLiveSnapshotObserver>)observer;  
  
// Snapshot callback  
@protocol V2TXLiveSnapshotObserver <NSObject>  
@optional  
- (void)onSnapshotComplete:(TXImage *)image;  
@end
```

Delay Adjustment

The LVB feature of the Tencent Cloud SDK, equipped with the self-developed playback engine, is not developed based on ffmpeg. Compared with open-source players, LVB delivers better performance in delay control. We provide three delay adjustment modes for live show, game, and hybrid scenarios, respectively.

• Performance comparison among the three modes

Control Mode	Lag Rate	Average Delay	Applicable Scenario	Principle
Speedy mode	High (relatively smooth)	2s - 3s	Live show (online quiz)	It has an advantage in delay control and is suitable for delay-sensitive scenarios.
Smooth mode	Lowest	>= 5s	Livestreaming game (Penguin e-Sports)	It is suitable for ultra-high-bitrate livestreaming games, such as PlayerUnknown's Battlegrounds.
Auto mode	Network adaption	2s - 8s	Hybrid scenario	The better the audience's network condition, the shorter the delay, and vice versa.

• Interfacing codes of the three modes

```
#define CACHE_TIME_FAST 1.0f
#define CACHE_TIME_SMOOTH 5.0f

// Speedy mode
[_player setCacheParams:CACHE_TIME_FAST maxTime:CACHE_TIME_FAST];

// Smooth mode
[_player setCacheParams:CACHE_TIME_SMOOTH maxTime:CACHE_TIME_SMOOTH];

// Auto mode
[_player setCacheParams:CACHE_TIME_FAST maxTime:CACHE_TIME_SMOOTH];
```

Listening to SDK Events

You can bind a **V2TXLivePlayerObserver** to the V2TXLivePlayer object. After that, you will be notified of SDK internal status information through **onPlayBegin**, **onLoading**, **onConnectionBroken** and **onStatisticsUpdate**.

Connection status update callback

```
/**
 * @brief Callback notification for the start of playback by the live player.
 */
- (void)onPlayBegin:(id<V2TXLivePlayer>)player;

/**
 * @brief Callback notification for the loading of the live player.
 */
- (void)onLoading:(id<V2TXLivePlayer>)player;

/**
 * @brief Callback notification for the disconnected live player.
 *
 * @param player Player object that calls back this notification
 * @param reason Disconnection reason. 0: normal disconnection
 */
- (void)onConnectionBroken:(id<V2TXLivePlayer>)player
reason:(NSInteger)reason;
```

Warning event callback

```
- (void)onWarning:(id<V2TXLivePlayer>)player  
code:(V2TXLiveCode)code  
message:(NSString *)msg  
extraInfo:(NSDictionary *)extraInfo;
```

Warning codes

Warning Code ID	Value	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Data upload was jammed because the upstream bandwidth was too low.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Lagging occurred during video playback.

Error event callback

```
- (void)onError:(id<V2TXLivePlayer>)player  
code:(V2TXLiveCode)code  
message:(NSString *)msg  
extraInfo:(NSDictionary *)extraInfo;
```

Error codes

Error Code ID	Value	Description
V2TXLIVE_ERROR_FAILED	-1	Unclassified error.
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was input during the API call.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The current API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API because the license was invalid.

Obtaining the video resolution

Through the `onVideoResolutionChanged` callback, you can obtain the aspect ratio of the current video. This is the quickest way to obtain the video resolution, which takes about 100 ms to 200 ms after playback starts.

```
- (void)onVideoResolutionChanged:(id<V2TXLivePlayer>)player  
width:(NSInteger)width  
height:(NSInteger)height;
```

Parameter	Description	Value
width	Video width	Resolution value, such as 1920
height	Video height	Resolution value, such as 1080

Periodically Triggered Status Notification

The **onStatisticsUpdate** notification is triggered every second to provide real-time feedback on the current status of the pusher. Like a car dashboard, it can inform you of what is happening inside the SDK so that you can see the current network conditions and video information.

```
- (void)onStatisticsUpdate:(id<V2TXLivePlayer>)player  
statistics:(V2TXLivePlayerStatistics *)statistics;
```

Evaluation Parameter	Description
appCpu	CPU utilization of the current app as a percentage (%)
systemCpu	int
width	Video width
height	Video height
fps	Frame rate in fps
videoBitrate	Video bitrate in Kbps
audioBitrate	Audio bitrate in Kbps

Android

Last updated : 2020-10-30 16:53:37

Basics

This document introduces the livestream playback function of Video Cloud SDK.

Livestreaming

- The **livestreaming** video source is pushed by the host in real time. When the host stops pushing, the video view on the player stops. In addition, the video is broadcast in real time, and no progress bar is displayed when the player is playing the livestreaming URL.

Supported protocols

Common livestreaming protocols are listed below. We recommend that you use an FLV-based livestreaming URL that starts with "http" and ends with ".flv" on apps.

Livestreaming Protocol	Advantage	Disadvantage	Playback Delay
FLV	High maturity, high concurrency, and no pressure	The video can be played only after the SDK is integrated.	2s - 3s
RTMP	Lowest theoretical delay when using high-quality lines	The performance is poor in the case of high concurrency.	1s - 3s
HLS (m3u8)	Good support by mobile browsers	There is a long delay.	10s - 30s

Notes

- Are there any restrictions?**

The Tencent Video Cloud SDK **does not** impose any limits on the source of the playback URL, which means it is available for both Tencent Cloud and non-Tencent Cloud playback URLs. However, the player in the Tencent Video Cloud SDK only supports livestreaming URLs in FLV and RTMP formats.

Interfacing

step1: setup a view

```
<com.tencent.rtmp.ui.TXCloudVideoView  
    android:id="@+id/video_video"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

step2: create a player

The `V2TXLivePlayer` module in the Tencent Video Cloud SDK implements the livestream playback feature, and using `setRenderView` to bind view.

```
TXCloudVideoView videoView = (TXCloudVideoView) view.findViewById(R.id.video_view);  
V2TXLivePlayer txLivePlayer = new V2TXLivePlayerImpl(mContext);  
txLivePlayer.setRenderView(videoView);
```

step3: start livestream playback

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";  
txLivePlayer.startPlay(flvUrl);
```

step4: adjust the view

- **setRenderFillMode: Fill or Fit**

Option	Description
V2TXLiveFillMode_Fill	Fill the image on the screen without leaving any black edges. If the aspect ratio of the view is different from that of the screen, part of the view content will be cropped.
V2TXLiveFillMode_Fit	Make the view fit the screen without cropping. If the aspect ratio of the view is different from that of the screen, black edges will appear.

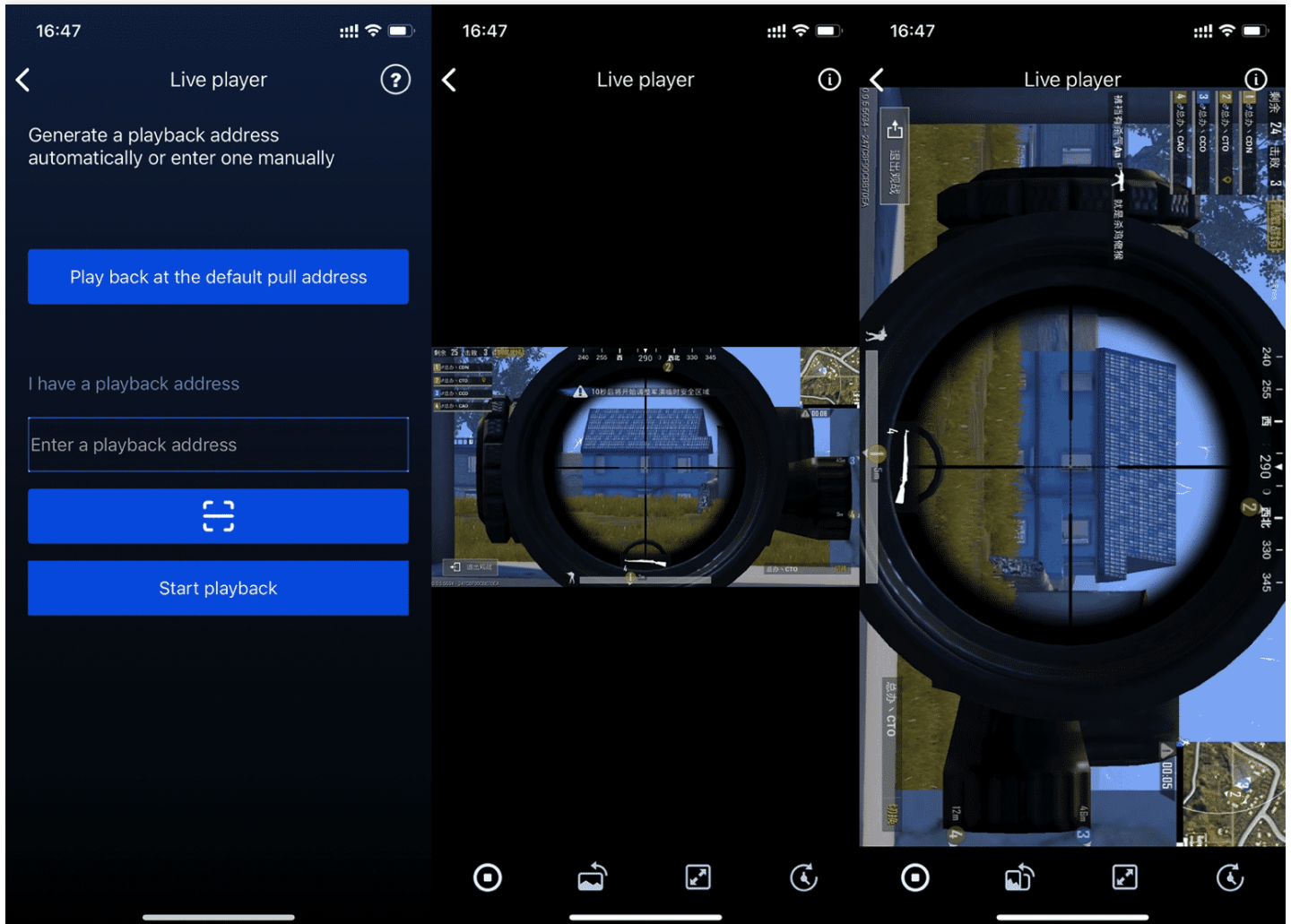
- **setRenderRotation: view rotation**

Option	Description
V2TXLiveRotation_0	Do not rotate the view.
V2TXLiveRotation_90	Rotate 90 degrees clockwise.
V2TXLiveRotation_180	Rotate 180 degrees clockwise.

V2TXLiveRotation_270

Rotate 270 degrees clockwise.

```
txLivePlayer.setRenderRotation(V2TXLiveDef.V2TXLiveRotation.V2TXLiveRotation_0);  
txLivePlayer.setRenderFillMode(V2TXLiveDef.V2TXLiveFillMode.V2TXLiveFillMode_Fit);
```



step5: pause playback

Strictly speaking, you cannot pause livestream playback. Here, pausing livestream playback actually means **freezing the image** and **turning off the sound**, but the video source keeps updating on the cloud. When you resume the playback, the video is resumed from the latest time. This is the biggest difference between livestreaming and VOD. In the VOD service, the video is paused and resumed in the same way as a local video file.

```
// Pause the audio.  
txLivePlayer.pauseAudio();  
// Resume the audio.
```

```
txLivePlayer.resumeAudio();

// Pause the video.
txLivePlayer.pauseVideo();
// Resume the video.
txLivePlayer.resumeVideo();
```

step6: stop livestream playback

Call **stopPlay** to stop livestream playback.

```
// Stop playback.
txLivePlayer.stopPlay();
```

step7: take snapshots

You can capture the current image as a frame by calling **snapshot**. This feature can only capture frames from the current livestreaming video. To capture the entire UI, you must call the API of the Android system.

```
txLivePlayer.snapshot(new V2TXLiveSnapshotObserver() {
    @Override
    public void onSnapshotComplete(Bitmap bitmap) {
        if (null != bitmap) {
            //doSomething
        }
    }
});
```

Delay Adjustment

The LVB feature of the Tencent Cloud SDK, equipped with the self-developed playback engine, is not developed based on ffmpeg. Compared with open-source players, LVB delivers better performance in delay control. We provide three delay adjustment modes for live show, game, and hybrid scenarios, respectively.

- **Performance comparison among the three modes**

Control Mode	Lag Rate	Average Delay	Applicable Scenario	Principle
Speedy	High	2s - 3s	Live show	It has an advantage in delay control and

mode	(relatively smooth)		(online quiz)	is suitable for delay-sensitive scenarios.
Smooth mode	Lowest	$\geq 5s$	Livestreaming game (Penguin e-Sports)	It is suitable for ultra-high-bitrate livestreaming games, such as PlayerUnknown's Battlegrounds.
Auto mode	Network adaption	2s - 8s	Hybrid scenario	The better the audience's network condition, the shorter the delay, and vice versa.

• Interfacing codes of the three modes

```
private float CACHE_TIME_FAST = 1.0f;
private float CACHE_TIME_SMOOTH = 5.0f;

// Speedy mode
txLivePlayer.setCacheParams(CACHE_TIME_FAST, CACHE_TIME_FAST);
// Smooth mode
txLivePlayer.setCacheParams(CACHE_TIME_SMOOTH, CACHE_TIME_SMOOTH);
// Auto mode
txLivePlayer.setCacheParams(CACHE_TIME_FAST, CACHE_TIME_SMOOTH);
```

Listening to SDK Events

You can bind a `V2TXLivePlayerObserver` to the `V2TXLivePlayer` object. After that, you will be notified of SDK internal status information through `onConnectionStateUpdate` and `onStatisticsUpdate`.

Warning codes

Warning Code ID	Value	Description
V2TXLIVE_OK	0	Playback successful
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Network busy
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Failed to obtain the video

```
@Override
public void onWarning(V2TXLivePlayer player, int code, String msg, Bundle extraInfo) {
    // doSomething
}
```


Error codes

Error Code ID	Value	Description
V2TXLIVE_ERROR_FAILED	-1	Unclassified error
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	Invalid parameter
V2TXLIVE_ERROR_REFUSED	-3	Refused
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	Not supported
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Invalid license

```
@Override
public void onError(V2TXLivePlayer player, int code, String msg, Bundle extraInfo) {
    // doSomething
}
```

Obtaining the video resolution

Through the `onVideoResolutionChanged` callback, you can obtain the aspect ratio of the current video. This is the quickest way to obtain the video resolution, which takes about 100 ms to 200 ms after playback starts.

```
@Override
public void onVideoResolutionChanged(V2TXLivePlayer player, int width, int height) {
    // doSomething
}
```

Parameter	Description	Value
width	Video width	Resolution value, such as 1920
height	Video height	Resolution value, such as 1080

Periodically Triggered Status Notification

The `onStatisticsUpdate` notification is triggered every second to provide real-time feedback on the current status of the pusher. Like a car dashboard, it can inform you of what is happening inside the SDK so that you can see the current network conditions and video information.

@Override

```
public void onStatisticsUpdate(V2TXLivePlayer player, V2TXLiveDef.V2TXLivePlayerStatistics statistics) {  
    // doSomething  
}
```

Parameter	Description
appCpu	CPU utilization of the current app as a percentage (%)
systemCpu	CPU utilization of the current system as a percentage (%)
width	Video width
height	Video height
fps	Frame rate in fps
videoBitrate	Video bitrate in Kbps
audioBitrate	Audio bitrate in Kbps

Co-anchoring RTC iOS & Android

Last updated : 2021-07-26 20:14:41

RTC-based Co-anchoring

In RTMP-based co-anchoring, the MLVB SDK offers the co-anchoring component `MLVBLiveRoom` to help you quickly implement the co-anchoring feature. To better cater to your co-anchoring needs, Tencent Cloud has launched an RTC-based co-anchoring scheme and offered simpler and more flexible V2 APIs.



MLVB's V2 APIs support publishing/co-anchoring via RTMP as well as RTC. You can choose whichever scheme fits your needs. Below is a comparison of the two schemes.

Item	RTMP	RTC
Protocol	Based on TCP	Based on UDP (more suitable for streaming)
QoS	Low adaptability to poor network connection	Streaming unaffected with 50% of packets loss; co-anchoring unaffected with 70% of packets loss
Region	Chinese mainland	Worldwide
Tencent Cloud products used	MLVB, CSS	MLVB, CSS, TRTC
Price	Contact sales	Tiered pricing. See Billing .

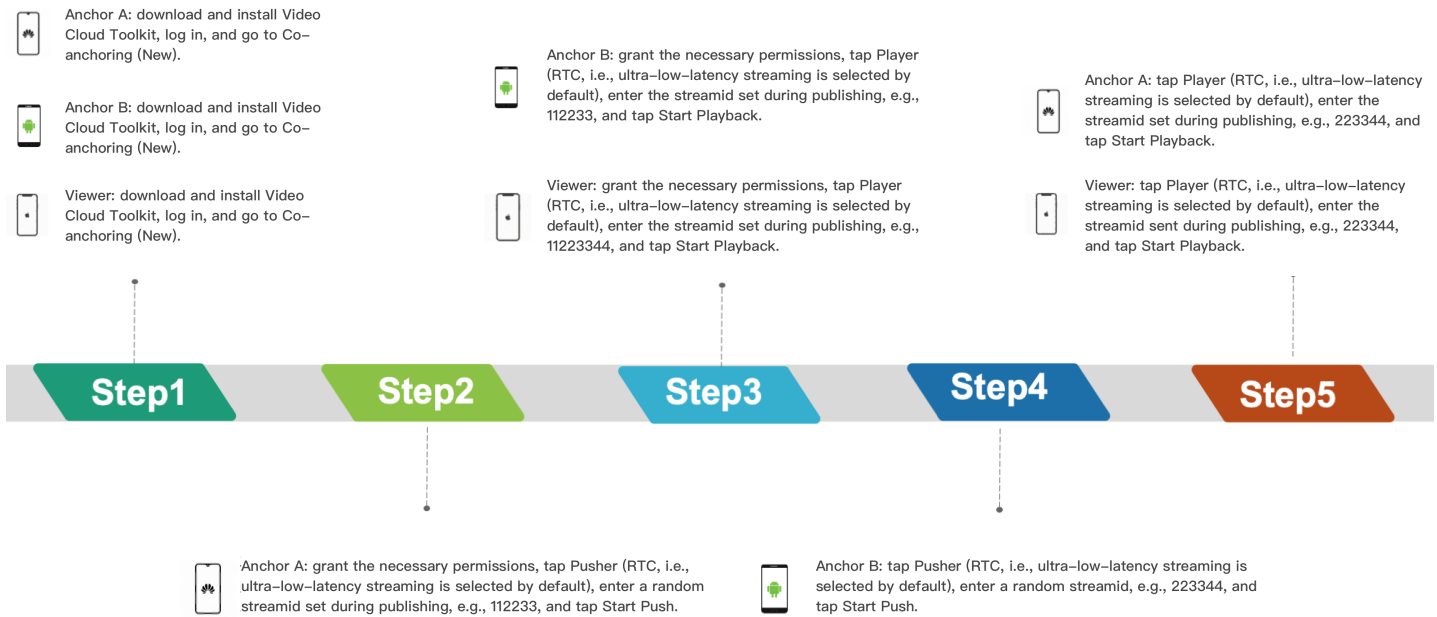
Trying out RTC-based Co-anchoring

Video Cloud Toolkit is a comprehensive audio-video service solution developed by Tencent Cloud. It allows you to try out the features of the TRTC, MLVB and UGC SDKs, including RTC-based co-anchoring: **Co-anchoring (New)**.

GitHub address

Platform	Demo	Source code	Folder
Android		GitHub	Demo/livelinkmicdemonew
iOS		GitHub	Demo/TXLiteAVDemo/LiveLinkMicDemoNew

Directions



Note :

When you try out the demo, please note that due to the use of the ultra-low-latency streaming protocol, in RTC-based co-anchoring, you cannot **use the same streamid for ultra-low-latency publishing and playback on the same device.**

Integration

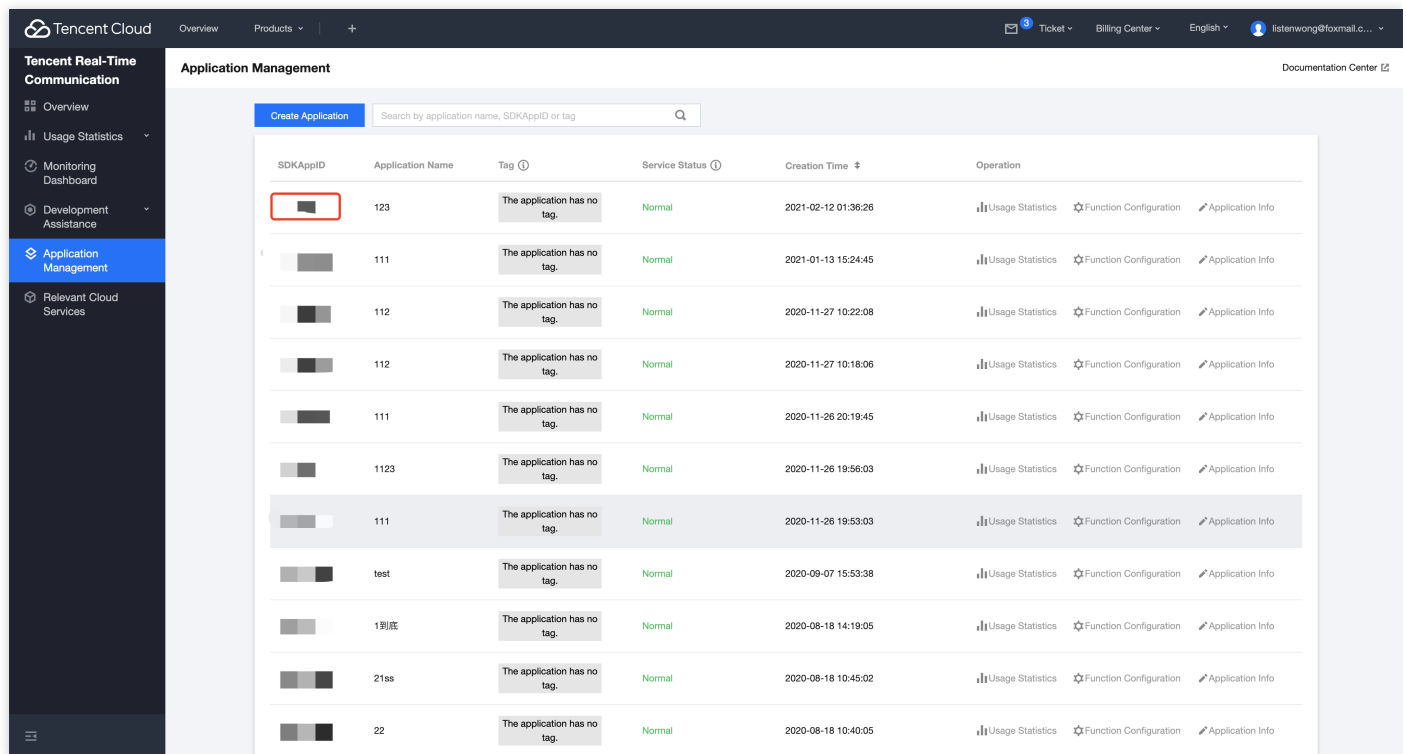
The latest MLVB SDK offers new V2 APIs `V2TXLivePusher` (publishing) and `V2TXLivePlayer` (playback) to power live streaming scenarios with **greater flexibility, lower latency, and larger scale**. See below for how to quickly implement interactive features such as co-anchoring and anchor competition using `V2TXLivePusher` and `V2TXLivePlayer`.

Step 1: Activate TRTC.

Before you start ultra-low-latency playback, follow the steps bellow to activate **TRTC**.

1. [Sign up](#) for a Tencent Cloud account and complete [Identity Verification](#).
2. Log in to the TRTC console and click [Application Management](#).

3. Click Create Application, enter an application name, e.g., `V2Demo` , and click confirm.



4. Find the application you created, and click **Application Info** on the right to view its `SDKAppID` .

5. Click **Quick Start**, wait for the information to load, and note the **key needed to issue UserSig**.

Note :

- The method for generating `UserSig` described in this document involves configuring `UserSig` in client code. In this method, `UserSig` may be easily decompiled and reversed, and if your key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of the demo.**
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see [How do I calculate UserSig on the server?](#).

Note :

After activating TRTC, you are advised to compile the SimpleCode (a simplified demo) we provide to try out the service and learn to use the demo's APIs with the help of the following documents.

- [Android](#)
- [iOS](#)

Step 2. Learn about publishing and playback protocols.

In live streaming scenarios, URLs are required for both publishing and playback. Below are examples of the URLs used for ultra-low-latency streaming.

• Publish

```
trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userid=A&usersig=xxxxx
```

-Playback

```
trtc://cloud.tencent.com/play/streamid?sdkappid=1400188888&userid=A&usersig=xxx
```

The table below lists the key fields in the URLs and their meanings.

Field	Description
trtc://	Prefix of the URL for low-latency publishing
cloud.tencent.com	Dedicated domain name for low-latency streaming, which must not be modified
push	Identifier, which indicates publishing
play	Identifier, which indicates playback
sdkappid	The SDKAppID generated in Activate TRTC
userid	Anchor's ID, which is set by you
usersig	The UserSig key obtained in Activate TRTC

Step 3. Learn about V2TXLivePusher publishing.

Splicing URLs

You need to splice your own URLs in the project code according to the rules described above.

Sample code

- [Java](#)
- [Objective-C](#)

```
// Create a V2TXLivePusher object and set the mode to TXLiveMode_RTC.
V2TXLivePusher pusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTC);
pusher.setObserver(new MyPusherObserver());
pusher.setRenderView(mSurfaceView);
pusher.startCamera(true);
pusher.startMicrophone();
// Pass in the low-latency publishing URL to start publishing.
pusher.startPush("trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=finnguan&
&usersig=xxxxx");
```

Step 4. Learn about V2TXLivePlayer playback.

Splicing URLs

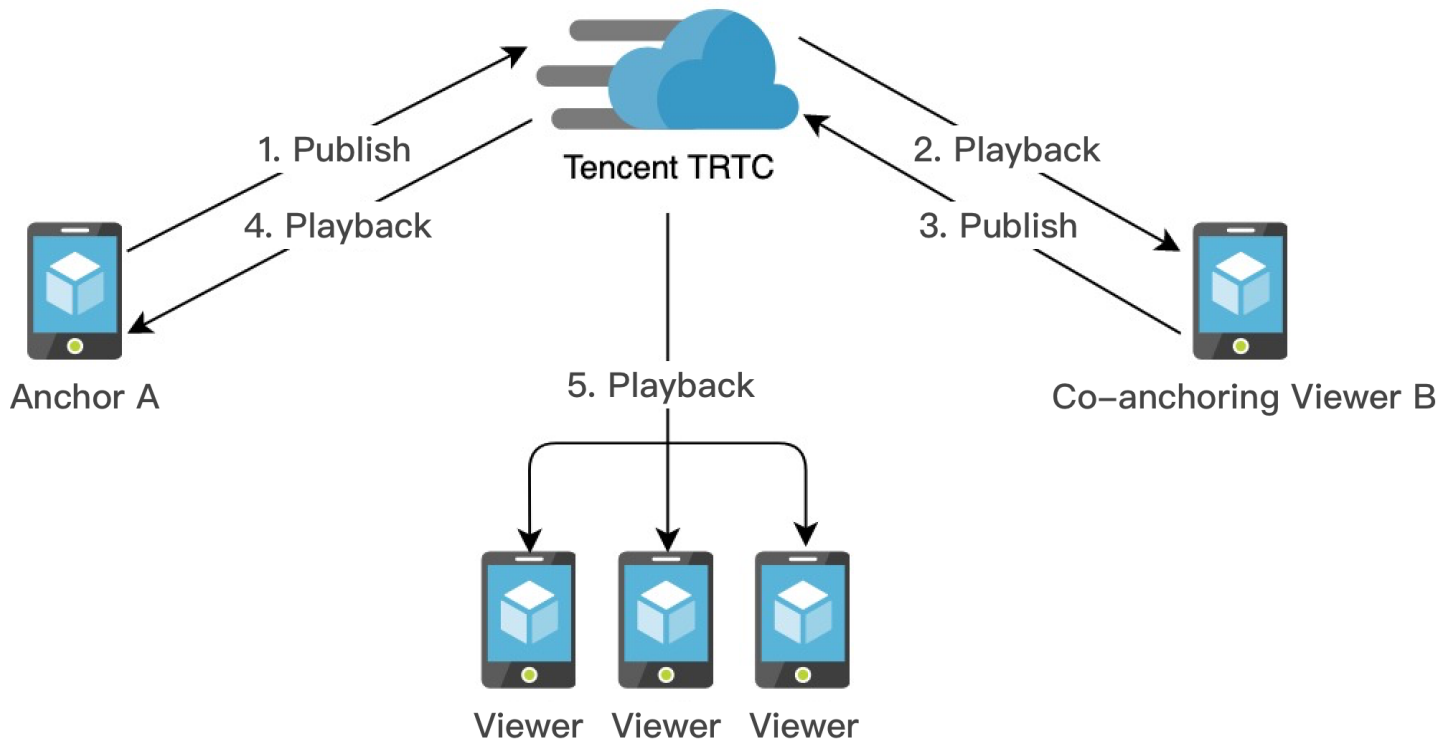
You need to splice your own URLs in the project code according to the publishing URL and the rules described above.

Sample code

- [Java](#)
- [Objective-C](#)

```
// Create a V2TXLivePlayer object.
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// Pass in the low-latency playback URL to start playback.
player.startPlay("trtc://cloud.tencent.com/play/streamid?sdkappid=1400188366&userId=A&use
rsig=xxx");
```

Step 5. Start co-anchoring.



1. Anchor A starts streaming by calling `V2TXLivePusher` .

- [Java](#)
- [Objective-C](#)

```
V2TXLivePusher pusherA = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
pusherA.startPush(pushURLA);
```

2. All viewers play back anchor A's stream by calling `V2TXLivePlayer` .

- [Java](#)
- [Objective-C](#)

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
playerA.startPlay(playURLA);
```

3. **Start co-anchoring:** viewer B (referred to as co-anchoring viewer B below) calls `V2TXLivePusher` to start publishing.

- [Java](#)
- [Objective-C](#)

```
V2TXLivePusher pusherB = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
pusherB.startPush(pushURLB);
```

4. **After receiving the co-anchoring notification**, anchor A calls `V2TXLivePlayer` to play back **co-anchoring viewer B**'s stream and start ultra-low-latency interaction with **co-anchoring viewer B**.

- [Java](#)
- [Objective-C](#)

```
V2TXLivePlayer playerB = new V2TXLivePlayerImpl(mContext);  
...  
playerB.startPlay(playURLB);
```

5. **After the co-anchoring starts**, other viewers call `V2TXLivePlayer` to play back **co-anchoring viewer B**'s stream.

Step 6. Start anchor competition.

1. Anchor A starts streaming by calling `V2TXLivePusher` .

- [Java](#)
- [Objective-C](#)

```
V2TXLivePusher pusherA = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);  
...  
pusherA.startPush(pushURLA);
```

2. Anchor B starts streaming by calling `V2TXLivePusher` .

- [Java](#)
- [Objective-C](#)

```
V2TXLivePusher pusherB = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);  
...  
pusherB.startPush(pushURLB);
```

3. **Start anchor competition**: anchor A and B call `V2TXLivePlayer` to play back each other's stream and interact with ultra low latency.

- [Java](#)
- [Objective-C](#)

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
playerA.startPlay(playURLA);

V2TXLivePlayer playerB = new V2TXLivePlayerImpl(mContext);
...
playerB.startPlay(playURLB);
```

4. **After the anchor competition starts**, anchor A's viewers call `V2TXLivePlayer` to play back anchor B's stream, and anchor B's viewers call `V2TXLivePlayer` to play back anchor A's stream.

Note :

Since you need to maintain a room and users yourself, you may think that the new RTC-based co-anchoring scheme is more complicated than the old one. In fact, **there isn't an absolutely better scheme, only one that better suits your needs.**

- You can stick to the old co-anchoring scheme if your application scenarios are not demanding on latency or concurrency.
- If you want to use V2 APIs without having to manage a room and users, try using [Tencent Cloud's IM SDK](#) to implement the necessary logic.

Billing

The RTC-based co-anchoring scheme is enabled by TRTC, which charges you based on **mic-on duration**. The table below lists the types of durations and their list prices.

Duration Type	Playback Resolution	Unit Price (USD/1,000 Min)
Audio	-	0.99
SD	$\leq 640 \times 480$	1.99
HD	$640 \times 480 - 1280 \times 720$	3.99
UHD	$> 1280 \times 720$	14.99

Note :

- Mic-on duration is the duration of playback by the anchor and co-anchoring viewers. The type of a duration depends on the playback resolution.
- If a user plays an audio-video stream, the duration will be charged only once as a video duration rather than twice as a video and audio duration.
- If a mic-on user publishes streams but does not play back streams, the user's duration will be billed as an audio duration, whose length is the same as the duration of publishing.

FAQs

1. Why is publishing and playback using the same `streamid` on the same device possible with `TXLivePusher` and `TXLivePlayer` but not with `V2TXLivePusher` and `V2TXLivePlayer` ?

`V2TXLivePusher` and `V2TXLivePlayer` are based on Tencent Cloud's TRTC protocol. The UDP-based ultra-low latency private protocol does not support communication using the same `streamid` on the same device. Given the current use case, we have not worked to support the feature, but may consider enabling it in the future.

2. What do the parameters generated in **Activate TRTC** mean?

`SDKAppID` is used to identify your application, and `UserID` your user. `UserSig` is a security signature calculated based on the two parameters using the **HMAC SHA256** encryption algorithm. Attackers cannot use your Tencent Cloud traffic without authorization as long as they cannot forge a `UserSig`. `UserSig` calculation involves hashing crucial information such as `SDKAppID`, `UserID`, and `ExpireTime`, as shown below.

```
// `UserSig` calculation formula, in which `secretkey` is the key used to calculate `UserSig`.
usersig = hmacsha256(secretkey, (userid + sdkappid + currtime + expire +
base64(userid + sdkappid + currtime + expire)))
```

3. How can I set the audio or video quality using `V2TXLivePusher` and `V2TXLivePlayer` ?

We provide APIs for the setting of audio and video quality. For details, please see [setAudioQuality\(\)](#) and [setVideoQuality:resolutionMode\(\)](#).

4. What does the error code `-5` mean?

The error code `-5` means failure to call an API due to invalid license. The enumerated value is `V2TXLIVE_ERROR_INVALID_LICENSE`. For other error codes, please see [V2TXLiveCode](#).

5. What is the typical latency of RTC-based co-anchoring?

In the new RTC-based co-anchoring scheme, the co-anchoring latency is lower than 200 ms, and the latency for anchors and viewers is 100-1,000 ms.