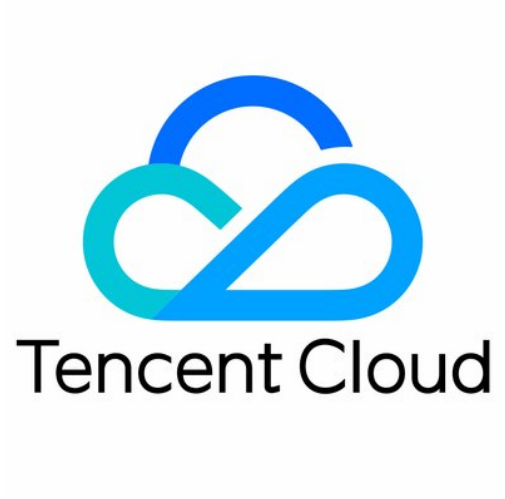
# Mobile Live Video Broadcasting

# Android-based Integration

# Product Introduction



# Contents

Android-based Integration
- Basic Features
  - Getting Started
  - Push Feature
  - TXLivePlayer
  - Gaming Screen Recording
  - Effect Feature
- realtime
  - LiveRoom
  - RTCRoom

# Android-based Integration
# Basic Features
# Getting Started

Last updated：2018-08-22 15:16:14

## Download SDK

You can download the LVB SDK for mobile devices from Tencent Cloud's official website. Decompress the downloaded file to acquire the libs directory, which mainly includes "so" and "jar" files. Files are listed below:

| File | Description |
| --- | --- |
| txrtmpsdk.jar | SDK Java layer encapsulation |
| libtxrtmpsdk.so | SDK core components |

## Supported Platform

- Android 4.0 (API 14) systems and above

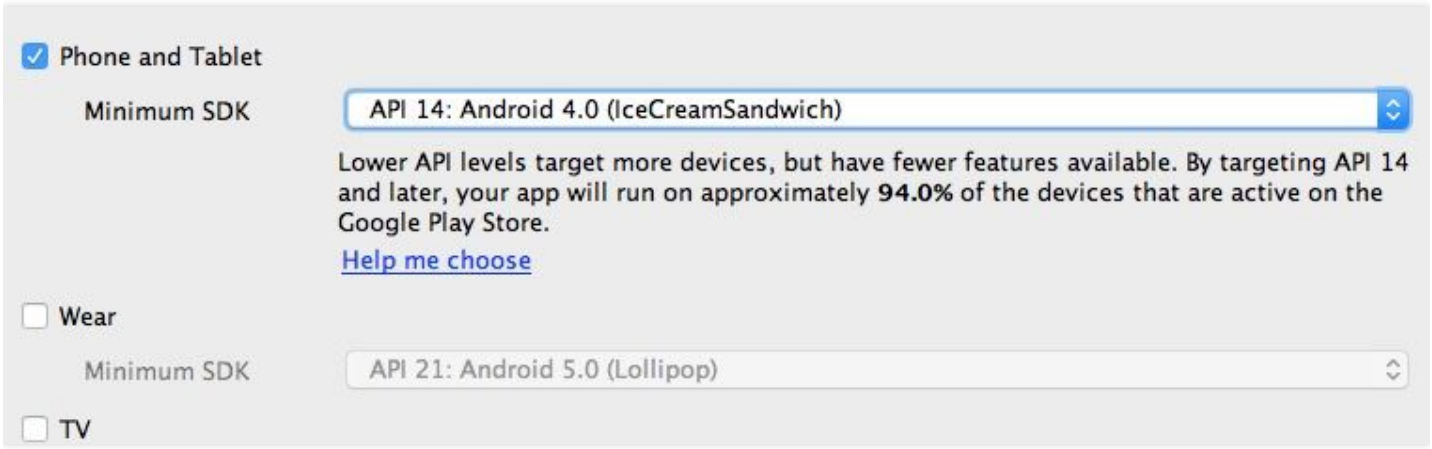
## Development Environment

The SDK development environment is described below. The App development environment does not need to be consistent with that of SDK, but they must be compatible:

- Android NDK: android-ndk-r10e
- Android SDK Tools: android-sdk_r21.1.2
  - minSdkVersion: 14
  - targetSdkVersion: 21
- Android Studio (while Android Studio is recommended, you can also choose to use Eclipse + ADT)

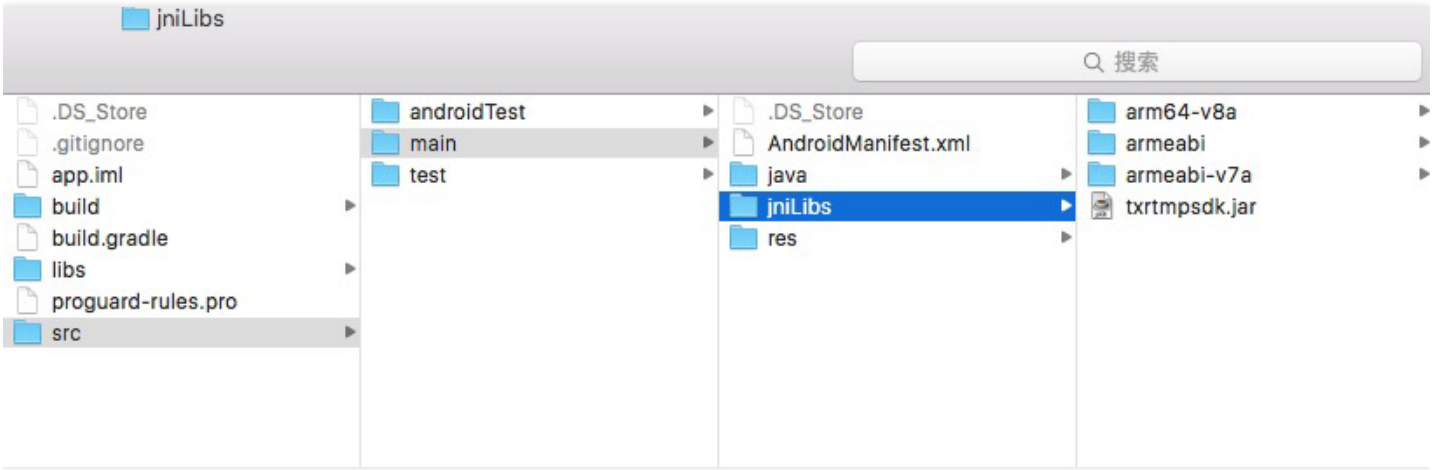
## Android Studio Environment Configuration

### 1. Create Android Project



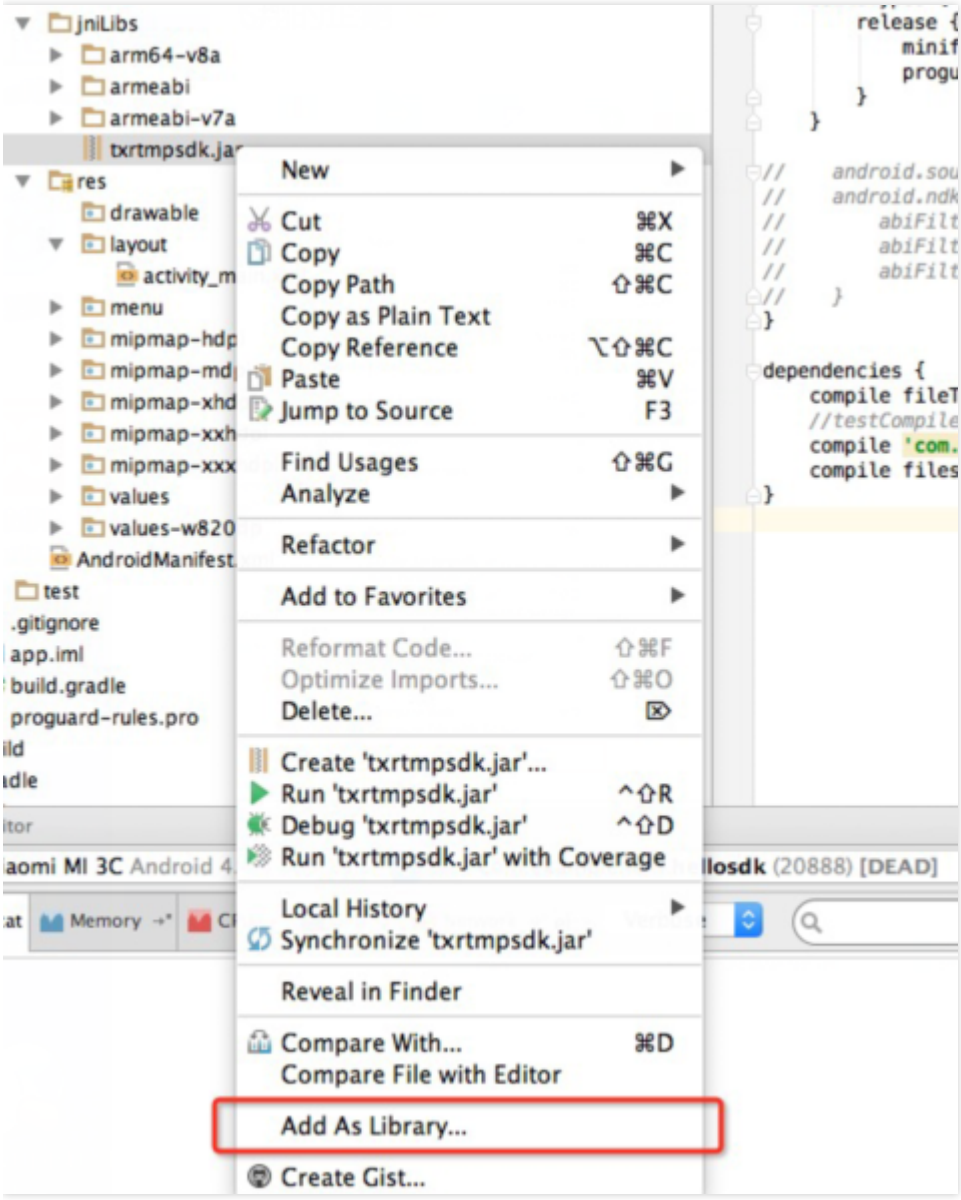

## 2. Copy Files

If your project doesn't have a previously specified jni loading path, we recommend that you put the files under the /src/main/jniLibs directory, which is the default jni loading directory of Android studio. **If you have specified the jni loading path (through gradle syntax: the sourceSets syntax or android.sources syntax), please copy the SDK related files mentioned above to this directory.**




## 3. Import jar Package

Find the newly created jniLibs directory in the Android Studio project. Expand the directory, and you will see txrtmpsdk.jar. Right-click on it and select "Add As Library..."




Once the package is imported, you will find that the following line of script is generated automatically in build.gradle:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    //testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:20.0.0'
    compile files('src/main/jniLibs/txrtmpsdk.jar')
}
```

## 4. Configure APP Permissions

Configure App permissions in AndroidManifest.xml. Generally, audio and video Apps require 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.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.Camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

## 5. Verify

Call the SDK API in the project to acquire SDK version information and verify if the project settings are correct.

**1. Reference the SDK**

Reference the class of SDK in MainActivity.java:

```
import com.tencent.rtmp.TXLivePusher;
```

**2. Call the getSDKVersion API in onCreate to acquire version number:**

```
int[] sdkver = TXLivePusher.getSDKVersion();
if (sdkver != null && sdkver.length >= 3) {
Log.d("rtmpsdk","rtmp sdk version is:" + sdkver[0] + "." + sdkver[1] + "." + sdkver[2]);
}
```

**3. Compile/Run**

The demo project can be compiled successfully if all of the above steps are correctly performed. If you run the project, you will see the following log information in logcat:

```
07-13 20:25:05.099 26119-26119/? D/rtmpsdk: rtmp sdk version is:1.5.188
```

## 6. Troubleshoot

If the following errors occur when you compile/run the project after importing the SDK into it:

**Caused by**: **android**.view.InflateException:
**Binary XML file #14**:Error **inflating class com**.tencent.rtmp.ui.TXCloudVideoView

Find the problem by following the steps below

**1**. Check if you have placed the "jar" package and "so" library into the jnilib directory.

**2**. If you're using the full version, check if the x64 "so" library has been filtered out. This is because the joint broadcasting feature in full version does not support mobile phones with x64 architecture at the moment.

```
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

ndk {
// filter out armeabi-x64 library
abiFilters "armeabi", "armeabi-v7a"
}
}
}
```

**3**. Look at the proguard rules and check if you have obfuscated RTMP SDK related classes too.

# Push Feature

Last updated：2018-08-10 16:21:46

## Basics

**Push** means to collect, encode audio/video data and push the data to your specified cloud video platform. The process involves a large amount of basic audio/video-related knowledge, you can only achieve desired results after lots of refining and optimizing.

Tencent Video Cloud SDK mainly helps you push videos on smart phones. The SDK comes with easy-to-use APIs and can be driven by using a single push URL:


## Notes

**You can push to non-Tencent Cloud addresses** using this SDK.

To solve the inaccurate DNS mapping problem within China, "Closest route selection" has been introduced starting from SDK 1.5.2. This feature selects the push route nearest to the VJ's location using

Tencent Cloud's close route selection server, which significantly improves push quality. However this also means that the route selection results only include Tencent server addresses. In addition, since a large number of customers use dedicated push domain names, SDK cannot determine whether the target is Tencent Cloud simply by using the URL text.

Therefore, if you wish to push videos to addresses of other cloud providers, contact our customer service for help to disable closest route selection feature for your account. This can be done through cloud control, thus there is no need to release new client versions.

# Preparation

- **Acquiring SDK**
  Download SDK and follow the instructions in Project Configuration to add the SDK into your APP development project.

- **Acquiring Test URL**
  After Activating the LVB service, use the "LVB Console" -> "LVB Code Access" -> "Push Generator" to generate push address. For more information, please see Acquiring Push Playback URL.

#### Code Interfacing

The guide mainly aims for **camera LVB** solution, which is mainly used for scenarios such as beauty show LVB, personal LVB, event LVB.

### Step 1: Add Interface Elements

To display the camera preview image, you need to add the following codes to your layout xml file. These codes will insert a TXCloudVideoView control, which is a specialized control we use to display the camera image, to your UI.

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

### Step 2: Create a Pusher Object

Create a **TXLivePusher** object, which will be used later to complete the push task.

Before you create a LivePush object, you need to specify a **LivePushConfig** object to determine the configuration parameters of various LivePush aspects, such as push resolution, frames per second (FPS) and GOP (seconds per one I-frame).

The LivePushConfig object has been equipped with some parameters that we have repeatedly tuned upon creation. If you do not wish to customize these parameters, you can simply assign them to the LivePush object. If you have experience in the related field and want to adjust the default configuration, you can read the **Advance Guide**.

```
TXLivePusher mLivePusher = new TXLivePusher(getActivity());
mLivePushConfig = new TXLivePushConfig();
mLivePusher.setConfig(mLivePushConfig);
```

### Step 3: Launch Push

After the preparations in Step 1 and Step 2, you can use the following codes to start the push:

```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startPusher(rtmpUrl);

TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mLivePusher.startCameraPreview(mCaptureView);
```

- **startPusher** is used to tell the SDK that which push URL the audio/video streams are being pushed to.
- **startCameraPreview** is used to associate the interface elements with the Pusher object, in order to render the picture captured by the mobile phone camera onto the screen.

### Step 4: Configure Video Definition

Configure video definition by using **setVideoQuality**, you can also configure by using the video quality options in TXLivePushConfig. However we still recommend that you use the options below:

| Definition | Configuration Parameter | Resolution | Applicable Scenario |
| --- | --- | --- | --- |
| **High Definition** | VIDEO_QUALITY_HIGH_DEFINITION | **540P** | This is a recommended definition which allows most mainstream mobile phones to present clear pictures. |

| Definition | Configuration Parameter | Resolution | Applicable Scenario |
| --- | --- | --- | --- |
| **Standard Definition** | VIDEO_QUALITY_STANDARD_DEFINITION | **360P** | Choose this definition if you have consideration about bandwidth cost. It brings average video quality but reduces bandwidth cost by 60%, compared to high definition. |
| **Dynamic** | VIDEO_QUALITY_QOS_DEFINITION | **Dynamic** | This will automatically adjust video resolution among three levels (192 * 336 - 540 * 960) based on network condition, in order to cope with network fluctuations. Suitable for scenarios with inconsistent network such as overseas LVB. Note: This type of video streams may be incompatible with certain players. |
| **Ultra High Definition** | VIDEO_QUALITY_SUPER_DEFINITION | **720P** | Note: This is not recommended for scenarios where videos are mostly viewed in small screens. You can consider using this definition if videos are viewed in large screens and the VJ has a great network. |

### Step 5: Beauty Filter

- **Beautify**
  The setBeautyFilter API can be used to configure beautify and whitening levels (0-9). 0 means to disable beautify feature. The beautify feature has been significantly improved since 1.9.1, now you can obtain optimal video quality if you use the feature together with 540 * 960 resolution (setVideoQuality - VIDEO_QUALITY_HIGH_DEFINITION).

```
mLivePusher.setBeautyFilter(7, 3);
```

- **Filter**
  The setFilter API can be used to configure filter effect. The filter is a histogram file, our designer group provided 8 materials which are packaged inside the Demo by default. You can use them as you like, without considering about copyright issues.

```
Bitmap bmp = null;
bmp = decodeResource(getResources(), R.drawable.langman);
if (mLivePusher != null) {
mLivePusher.setFilter(bmp);
}
```


Be sure to use PNG images if you wish to customize the filters. **Do NOT use JPG image.**

- **Exposure**
  setExposureCompensation is used to adjust exposure value. This option is not available on the iOS end (where we use the auto-exposure feature of the system). Considering that Android models can be greatly different from each other and most inexpensive phones may have poor exposure mechanisms, we recommend that you add an auto-exposure slider on the UI to allow VJs to adjust exposure value on

their own.


The parameter of setExposureCompensation is a floating-point value between -1 and 1: 0 means no adjustment; -1 means lowest exposure and 1 means highest exposure.

## Step 6: Camera Control

- **Switch front or rear camera**
  The **front** camera is used by default (this default value can be changed by modifying the configuration function setFrontCamera in TXLivePushConfig). The camera is switched each time switchCamera is called. Make sure both TXLivePushConfig and TXLivePusher objects have been initialized before switching camera.

```
// The front camera is used by default
mLivePusher.switchCamera();
```

- **Turn the flashlight on or off**
  Flashlight is only available for the rear camera. In addition, this API needs to be called after preview is started.

```
//Flashlight is turned on if mFlashTurnOn is set to true; otherwise it is turned off
if (!mLivePusher.turnOnFlashLight(mFlashTurnOn)) {
Toast.makeText(getActivity().getApplicationContext(),
"Failed to turn on flashlight: Most mobile phones do not support a front flashlight!", Toast.LENGTH
_SHORT).show();
}
```

- **Auto or Manual Camera Focus**
  In most cases, focusing is only supported for the rear camera. The SDK supports two focusing mode: **Manual Focus** and **Auto Focus**.
  Auto Focus is a capability provided by the system, but some models do not support Auto Focus. Manual Focus and Auto Focus are mutually exclusive. If Auto Focus is enabled, Manual Focus will not work.
  The SDK uses Manual Focus by default. You can switch it through the configuration function setTouchFocus API in TXLivePushConfig:

```
mLivePushConfig.setTouchFocus(mTouchFocus);
mLivePusher.setConfig(mLivePushConfig);
```

### Step 7: Set Logo Watermark

Recent policies require that LVB videos must be marked with watermarks. With that in mind, we will focus on this feature that had seemed insignificant before.
Tencent Video Cloud currently supports two watermark settings. One is to set watermark in the push SDK, where the videos are marked with watermarks in the SDK before being encoded. Another is applying watermarks in the cloud. That is, the cloud resolves videos and adds Logo watermarks to them.

We suggest that you **add watermarks with the SDK**, because there are three major problems when watermarking in the cloud:
(1) This service increases the load on the cloud machine and is not free, which will increase your cost;
(2) It is not ideally compatible with certain situations such as resolution switching during the push process. This may cause problems like blurred screen.
(3) It may cause an additional 3-second video delay, which is caused by the transcode service.

SDK requires that watermark images are PNG format, because such images contain transparency information, which helps processes such as anti-aliasing. (Do not just change the extension of a JPG image to PNG in Windows and put it in. Professional PNG logos need to be processed by professional art designers)

```
//Set video watermark
mLivePushConfig.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 10, 10);
mLivePusher.setConfig(mLivePushConfig);
```

### Step 8: Hardware Encoding

The **setHardwareAcceleration** configuration API in TXLivePushConfig can be used to enable or disable hardware encoding.

```
if (!HWSupportList.isHWVideoEncodeSupport()){
Toast.makeText(getActivity().getApplicationContext(),
"The current mobile phone model is not whitelisted or the API level is too low (the minimum level is 18). Please think carefully before enabling hardware encoding.",
Toast.LENGTH_SHORT).show();
}
mLivePushConfig.setHardwareAcceleration(mHWVideoEncode);
mLivePusher.setConfig(mLivePushConfig);
```

mHWVideoEncode includes the following options.

| Hardware Encoding Option | Description |
| --- | --- |
| ENCODE_VIDEO_HARDWARE | Enable hardware acceleration |
| ENCODE_VIDEO_SOFTWARE | Disable hardware acceleration (default) |
| ENCODE_VIDEO_AUTO | Automatically determines whether to enable hardware acceleration |

- **Compatibility Assessment**
  Android phones now provide better support for hardware acceleration, compared to previous years. However, certain models still have incompatibility problems. Currently, RTMP SDK controls the use of hardware acceleration by using an internal blacklist in order to prevent such problems on models with incompatibility issues. If you failed to use hardware encoding, the SDK will automatically switch to software encoding.

- **Differences**
  If you enable hardware acceleration, the phone's battery consumption will significantly decrease, while maintaining an ideal temperature. But there will be more obvious mosaic when there are large movements in the image, compared to software encoding. The mosaic will get worse for earlier, low-end phones. Hardware acceleration is not recommended for users who require high video quality.

- **Recommended Design**
  We recommend that you use software acceleration when setVideoQuality is configured as High Definition (recommended definition) or Standard Definition. If you're worrying about CPU usage and high temperature, you can create a simply protection logic:

  > If the **FPS** of the push process stays low for a certain period of time (this can be detected through the NET_STATUS_VIDEO_FPS event of TXLivePushListener), for example, FPS stays below 10 frames/sec in 30 seconds, which means the CPU is overloaded, switch to hardware encoding.

### Step 9: Background Push

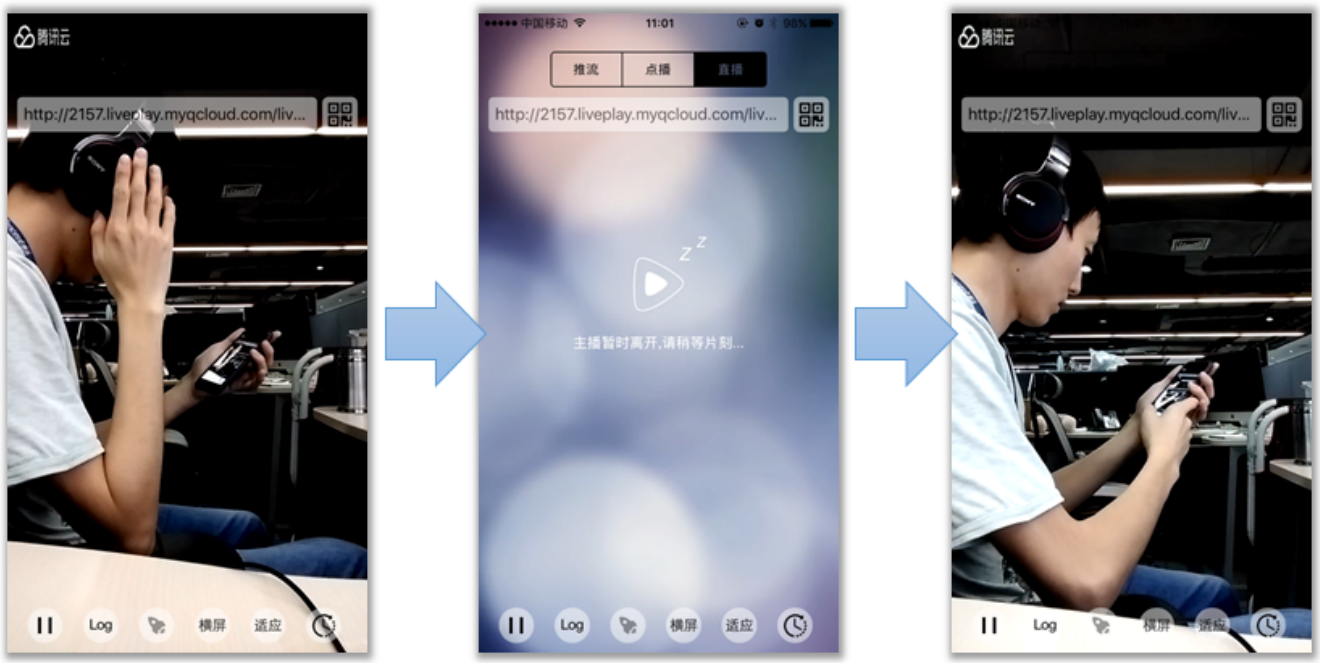
Usually, once the App switches to background, the camera's capture function will be disabled by the Android system, which means the SDK can no longer continue to capture and encode audio/video data. This is what happens if we don't do anything:

- Phase 1 (from switching to background -> 10 seconds later) - CDN cannot provide video streams to viewers because it doesn't have any data, and the viewers will see a frozen display.
- Phase 2 (10 seconds -> 70 seconds) - The player at the viewer side exits because it cannot receive LVB stream in a long time. Everyone leaves the studio.
- Phase 3 (after 70 seconds) - The RTMP linkage of the push is disconnected by the server. The VJ needs to restart LVB to continue.

Even answering a short emergency call will cause all the viewers to leave the studio, with such an interaction experience. So how can we optimize it?

Actually, we can some irregular approaches, e.g., create a Service, and use a SurfaceView with 1*1 pixel to collect Camera data continuously. However, if you are a VJ and find that the App can still access your camera after it is switched to background, are you really going to use it?

We need to achieve a perfect balance between privacy protection and the viewers' experience, thus we introduced a solution since SDK 1.6.1 version:




- **9.1) Set pauseImg**
  Before the push, use the setPauseImg API in TXLivePushConfig to set a waiting picture. It is recommended to use a picture such as: "The VJ will come back soon".

- **9.2) Set setPauseFlag**
  Before the push, use the setPauseFlag API in TXLivePushConfig to configure which collections are to be stopped when the push is paused. The default picture set with pauseImg will be pushed when video collection is stopped, while mute data will be pushed when audio collection is stopped.

  ```
  setPauseFlag(PAUSE_FLAG_PAUSE_VIDEO|PAUSE_FLAG_PAUSE_AUDIO);// means to stop both video collection and audio collection, then push filler audio/video stream;

  setPauseFlag(PAUSE_FLAG_PAUSE_VIDEO);// means to stop camera video collection while the microphone still collects audio as usual. This is used for scenarios such as when VJ is dressing up;
  ```

- **9.3) Switch to background process**
  If the App is switched to background during the push process and the pausePush API function in TXLivePusher is called, the SDK will no longer be able to collect camera video but you can still use the PauseImg you previously set to keep up the push.

```
// onStop life cycle function of activity
@Override
public void onStop(){
super.onStop();
mCaptureView.onPause(); // mCaptureView is the image rendering view of the camera
mLivePusher.pausePusher(); // To inform the SDK that "Background Push Mode" has started
}
```

- **9.4) Switch to foreground process**
  After the App is switched back to the foreground and the resumePush API function of TXLivePusher is called, the SDK will continue to collect camera pictures to push.

```
// onStop life cycle function of activity
@Override
public void onResume() {
super.onResume();
mCaptureView.onResume(); // mCaptureView is the image rendering view of the camera
mLivePusher.resumePusher(); // Inform the SDK to return to foreground to push
}
```

### Step 10: Remind the VJ "network is poor"

Step 13 will discuss how to handle SDK push events. **PUSH_WARNING_NET_BUSY** is very useful, it means: **the uplink network of the VJ is poor, and stutters have occurred in the viewer end.**

When you receive this WARNING, you can use the UI to remind VJ to change network egress, or get closer to the WiFi. Or tell her to say something like this: "Hello dear, I'm streaming! Stop reading eBay! What? Not eBay? Then stop downloading movies!"

### Step 11: Push in landscape mode

In most cases, VJs push videos in an LVB by holding the screen in a portrait orientation so that the viewers can get portrait images. However, sometimes VJs may need to hold the screen in a landscape orientation to allow the viewers to get landscape images with a wider view. In this case, push in landscape mode is required. The figures below show the difference between landscape mode and portrait mode in terms of

the images at the viewer end.


**Note:** The aspect ratios of images at viewer end are different between landscape mode and portrait mode. In portrait mode, the aspect ratio is 9:16, while in landscape mode, 16:9.

### Adjust Viewer-end Performance

This is done by configuring the **setHomeOrientation** option in LivePushConfig. It controls whether the video aspect ratio seen by the viewers is **16:9** or **6:19**, you can check the result by using your player to see if the aspect ratio is adjusted as expected.

| Settings | Description |
| --- | --- |
| VIDEO_ANGLE_HOME_RIGHT | The Home key is on the right |
| VIDEO_ANGLE_HOME_DOWN | The Home key is at the bottom |
| VIDEO_ANGLE_HOME_LEFT | The Home key is on the left |

| Settings | Description |
| --- | --- |
| VIDEO_ANGLE_HOME_UP | The Home key is at the top |

- **Adjust VJ-end Performance**
  When viewers are able to see expected display, you can then adjust the preview display at the VJ side. Now, the picture seen by the VJ can be rotated through the setRenderRotation API in TXLivePusher. This API provides four parameters (**0, 90, 180 and 270**) for setting the rotation angle.

- **Activity Auto-rotation**
  The Activity of Android system can rotate according to the feedback of the phone's gravity sensor (configure android:configChanges). How can we make the push switch between landscape mode and portrait mode based on gravity sensor feedback as shown below?


```
@Override
public void onConfigurationChanged(Configuration newConfig) {
// When auto-rotation is enabled, as Activity rotates with the mobile phone, you simply need to ch
ange the push direction
int mobileRotation = this.getActivity().getWindowManager().getDefaultDisplay().getRotation();
int pushRotation = TXLiveConstants.VIDEO_ANGLE_HOME_DOWN;
switch (mobileRotation) {
case Surface.ROTATION_0:
pushRotation = TXLiveConstants.VIDEO_ANGLE_HOME_DOWN;
break;
case Surface.ROTATION_90:
pushRotation = TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT;
```

```
break;
case Surface.ROTATION_270:
pushRotation = TXLiveConstants.VIDEO_ANGLE_HOME_LEFT;
break;
default:
break;
}

//Make the configuration effective by setting "config" (there is no need to restart the push process because Tencent Cloud is one of the few cloud providers that support resolution hot-switch during LVB)
mLivePusher.setRenderRotation(0);
mLivePushConfig.setHomeOrientation(pushRotation);
mLivePusher.setConfig(mLivePushConfig);
}
```

### Step 12: Background Music Mix

Background music mixing is supported starting from SDK 1.6.1. The VJ can choose headset mode or no-headset mode. You can achieve background music mixing feature by using the following APIs in TXLivePusher:

| API | Description |
| --- | --- |
| playBGM | Send a song via path. In Mini LVB Demo, we obtain music files from the iOS local media library |
| stopBGM | Stop background music |
| pauseBGM | Pause background music |
| resumeBGM | Resume background music |
| setMicVolume | Set the microphone volume when mixing music. It is recommended to add a slider in the UI to allow VJs to set volume on their own |
| setBGMVolume | Set the background music volume when mixing music. It is recommended to add a slider in the UI to allow VJs to set volume on their own |

### Step 13: End Push

It is simple to end a push process, but proper cleaning work is required. Since only one TXLivePusher object (used for pushing) and only one TXCloudVideoView object (used for displaying image) can run at a time, improper cleaning work may adversely affect the next LVB.

```
//End the push with proper cleanup work
public void stopRtmpPublish() {
mLivePusher.stopCameraPreview(true); //End camera preview
mLivePusher.stopPusher(); //End push
mLivePusher.setPushListener(null); //Unbind listener
}
```

# Event Handling

### 1. Event Listening

SDK listens to push related events using the TXLivePushListener proxy. Note that the TXLivePushListener only listens to push events with prefix **PUSH_**.

### 2. Normal Events

Events that are always prompted during a successful push. For example, receiving 1003 means that the system will start rendering the camera pictures

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_EVT_CONNECT_SUCC | 1001 | Successfully connected to Tencent Cloud push server |
| PUSH_EVT_PUSH_BEGIN | 1002 | Handshake with the server completed, everything is OK, ready to start push |
| PUSH_EVT_OPEN_CAMERA_SUCC | 1003 | The pusher has successfully started the camera (this will take 1-2 seconds on some Android phones) |

### 3. Error Notification

The push cannot continue as the SDK detected critical problems. For example, the user disabled camera permission for the APP so the camera cannot be started.

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_ERR_OPEN_CAMERA_FAIL | -1301 | Failed to start the camera |
| PUSH_ERR_OPEN_MIC_FAIL | -1302 | Failed to start the microphone |
| PUSH_ERR_VIDEO_ENCODE_FAIL | -1303 | Video encoding failed |

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_ERR_AUDIO_ENCODE_FAIL -1304 | Audio encoding failed | |
| PUSH_ERR_UNSUPPORTED_RESOLUTION | -1305 | Unsupported video resolution |
| PUSH_ERR_UNSUPPORTED_SAMPLERATE | -1306 | Unsupported audio sampling rate |
| PUSH_ERR_NET_DISCONNECT | -1307 | Network disconnected. Reconnection attempts have failed for three times, thus no more retries will be performed. Please restart the push manually |

### 4. Warning Events

SDK detected some reparable problems. Most warning events will trigger protection logics or recovery logics that involve retrying, and in most of the cases the problems can be recovered. Don't make a fuss.

- **WARNING_NET_BUSY**
  The VJ's network is poor. If you need UI prompts, this warning is relatively more useful (Step 10).

- **WARNING_SERVER_DISCONNECT**
  The push request is rejected by backend. This is usually caused by miscalculated txSecret in the push address, or because the push address is occupied by others (a push URL can only have one pushing end at a time).

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_WARNING_NET_BUSY | 1101 | Bad network condition: data upload is blocked because uplink bandwidth is too small |
| PUSH_WARNING_RECONNECT | 1102 | Network disconnected, automatic reconnection has started (auto reconnection will be stopped if it fails for three times) |
| PUSH_WARNING_HW_ACCELERATION_FAIL | 1103 | Failed to start hardware encoding. Software encoding is used |
| PUSH_WARNING_DNS_FAIL | 3001 | RTMP - DNS resolution failed (this will trigger retry process) |

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_WARNING_SEVER_CONN_FAIL | 3002 | Failed to connect to the RTMP server (this will trigger retry process) |
| PUSH_WARNING_SHAKE_FAIL | 3003 | RTMP server handshake failed (this will trigger retry process) |
| PUSH_WARNING_SERVER_DISCONNECT | 3004 | The RTMP server actively disconnected (this will trigger retry process) |

For the definition of all events, please see the header file "**TXLiveConstants.java**"

# TXLivePlayer

Last updated：2018-09-26 10:26:14

## Basics

This document describes the LVB playback feature of Tencent Video Cloud SDK. The following are the basics you must learn before getting started.

- **LVB and VOD**
  The video source of LVB (LIVE) is pushed by VJ in real time. When the VJ stops broadcasting, the video image on the playback device stops. In addition, the video is broadcasted in real time, no progress bar is displayed when the player is playing the LVB URL.

  The video source of Video On-demand (VOD) is a video file on cloud, which can be played at any time as long as it has not been deleted from the cloud. You can control the playback progress using the progress bar. The video playbacks on Tencent Video and Youku Tudou are typical VOD scenarios.

- **Supported Protocols**
  Commonly used LVB protocols are as follows. It is recommended to use an LVB URL based on the FLV protocol (starting with "http" and ending with ".flv") on Apps:

| 直播协议 | 优点 | 缺点 | 播放延迟 |
| --- | --- | --- | --- |
| FLV | 成熟度高、高并发无压力 | 需集成SDK才能播放 | 2s - 3s |
| RTMP | 优质线路下理论延迟最低 | 高并发情况下表现不佳 | 1s - 3s |
| HLS(m3u8) | 手机浏览器支持度高 | 延迟非常高 | 10s - 30s |

## Notes

- **Is there any restriction?**
  Tencent Cloud SDK **does not** impose any restrictions on the source of playback URLs, which means you can use the SDK to play videos from both Tencent Cloud and non-Tencent Cloud addresses. But the player in Tencent Video Cloud SDK only supports three LVB video address formats (FLV, RTMP and HLS (m3u8)) and three VOD address formats (MP4, HLS (m3u8) and FLV).

- **Historical factors**
  In earlier versions of the SDK, TXLivePlayer works as the only Class carrying both LVB and VOD features. With the expansion of the VOD features, we have made VOD a separate set of features carried by TXVodPlayer starting from SDK 3.5. For the compilation to be successful, VOD features such as seek are still visible in TXLivePlayer.

# Interfacing

### Step 1: Add a view

To display the video views in a player, you first need to add the following code into the layout xml file:

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

### Step 2: Create a player

The **TXLivePlayer** module in Tencent Video Cloud SDK carries the LVB playback feature. Use API **setPlayerView** to associate it with the **video_view** control we just added to the interface.

```
//mPlayerView is the view added in step 1.
TXCloudVideoView mView = (TXCloudVideoView) view.findViewById(R.id.video_view);

//Create a player object
TXLivePlayer mLivePlayer = new TXLivePlayer(getActivity());

//Key player object and interface view
mLivePlayer.setPlayerView(mView);
```

### Step 3: Start playback

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
mLivePlayer.startPlay(flvUrl, TXLivePlayer.PLAY_TYPE_LIVE_FLV); //FLV is recommended
```

| Option | Enumerated Value | Description |
| --- | --- | --- |

| Option | Enumerated Value | Description |
| --- | --- | --- |
| PLAY_TYPE_LIVE_RTMP | 0 | The URL passed in is an RTMP-based LVB URL |
| PLAY_TYPE_LIVE_FLV | 1 | The URL passed in is an FLV-based LVB URL |
| PLAY_TYPE_LIVE_RTMP_ACC | 5 | Low-latency URL (only for joint broadcasting scenarios) |
| PLAY_TYPE_VOD_HLS | 3 | The URL passed in is an HLS (m3u8)-based playback URL |

**About HLS (m3u8)**
Considering its high latency, HLS is not recommended as the playback protocol for playing LVB videos on your App (although it is suitable for playing VOD videos). Recommended playback protocols include LIVE_FLV and LIVE_RTMP.

### 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 "video_view" control added in step 1.

- **setRenderMode: Full Screen or Self-Adaption**

| Option | Description |
| --- | --- |
| RENDER_MODE_FULL_FILL_SCREEN | The full screen is filled with the image that is spread proportionally, with the excess parts cut out. In this mode, black edges will not appear in the screen, but the image may not be displayed completely because some areas are cut out. |
| RENDER_MODE_ADJUST_RESOLUTION | The image is scaled proportionally to adapt to the longest edge. Both the width and the height of the scaled image will not extend beyond the display area and the image is centered. In this mode, black edges maybe appear in the screen. |

- **setRenderRotation: Screen rotation**

| Option | Description |
| --- | --- |
| RENDER_ROTATION_PORTRAIT | Normal playback (The Home button is located directly below the image) |
| RENDER_ROTATION_LANDSCAPE | The image rotates 270° clockwise (the Home button is directly to the left of the image) |

```
// Set the filling mode
mLivePlayer.setRenderMode(TXLiveConstants.RENDER_MODE_ADJUST_RESOLUTION);
// Set image rendering orientation
mLivePlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_LANDSCAPE);
```


最长边填充


完全填充


横屏模式

### Step 5: Pauses playback.

Strictly speaking, you cannot pause LVB playback. The so-called pausing LVB playback means **freezing the image** and **turning off the sound**, but the video source keeps updating on the cloud. When you call resume, the video is resumed from the latest time, which works quite differently from VOD videos that are paused and resumed in the same way as local video files).

```
// Pause
mLivePlayer.pause();
// Resume
mLivePlayer.resume();
```

### Step 6: End playback

At the end of the playback, **be sure to terminate the View control**, especially before the next startPlay. Otherwise a number of memory leak and splash screen issues will occur.

At the same time, when exiting the playback interface, be sure to call the `onDestroy()` function of the rendering View, otherwise memory leak and the warning message "Receiver not registered" may occur.

```
@Override
public void onDestroy() {
super.onDestroy();
mLivePlayer.stopPlay(true); // "true" means to clear the last frame
mView.onDestroy();
}
```

The boolean parameter of stopPlay means whether to clear the last frame. The LVB players in the earlier versions of RTMP SDK did not have the concept of Pause, so the boolean value is used to control whether to clear the last frame.

If you want to stop the video at the last frame after the VOD playback ends, do nothing after receiving the playback end event, and it defaults to stop at the last frame.

### Step 7: Receive messages

This feature is used to deliver certain custom messages from the pusher end to the viewer end via audio/video lines. It is applicable to the following scenarios:
(1) Online quiz: The pusher end delivers the **questions** to the viewer end. Perfect "sound-image-question" synchronization can be achieved.
(2) Live show: The pusher end delivers **lyrics** to the viewer end. The lyric effect can be displayed on the viewer end in real time and its image quality is not affected by video encoding.
(3) Online education: The pusher end delivers the operations of **Laser pointer** and **Doodle pen** to the viewer end. The drawing can be performed at the viewer end in real time.

You can use this feature as follows:

- Switch the **setEnableMessage** toggle button in TXLivePlayConfig to **YES**.

- TXLivePlayer listens into messages by **TXLivePlayListener**, message No.: **PLAY_EVT_GET_MESSAGE (2012)**.

```
//Android sample code
mTXLivePlayer.setPlayListener(new ITXLivePlayListener() {
@Override
public void onPlayEvent(int event, Bundle param) {
if (event == TXLiveConstants.PLAY_ERR_NET_DISCONNECT) {
roomListenerCallback.onDebugLog("[AnswerRoom] Pull failed: network disconnected");
roomListenerCallback.onError(-1, "Network disconnected, pull failed");
}
else if (event == TXLiveConstants.PLAY_EVT_GET_MESSAGE) {
String msg = null;
try {
msg = new String(param.getByteArray(TXLiveConstants.EVT_GET_MSG), "UTF-8");
roomListenerCallback.onRecvAnswerMsg(msg);
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}
}
@Override
public void onNetStatus(Bundle status) {
}
});
```

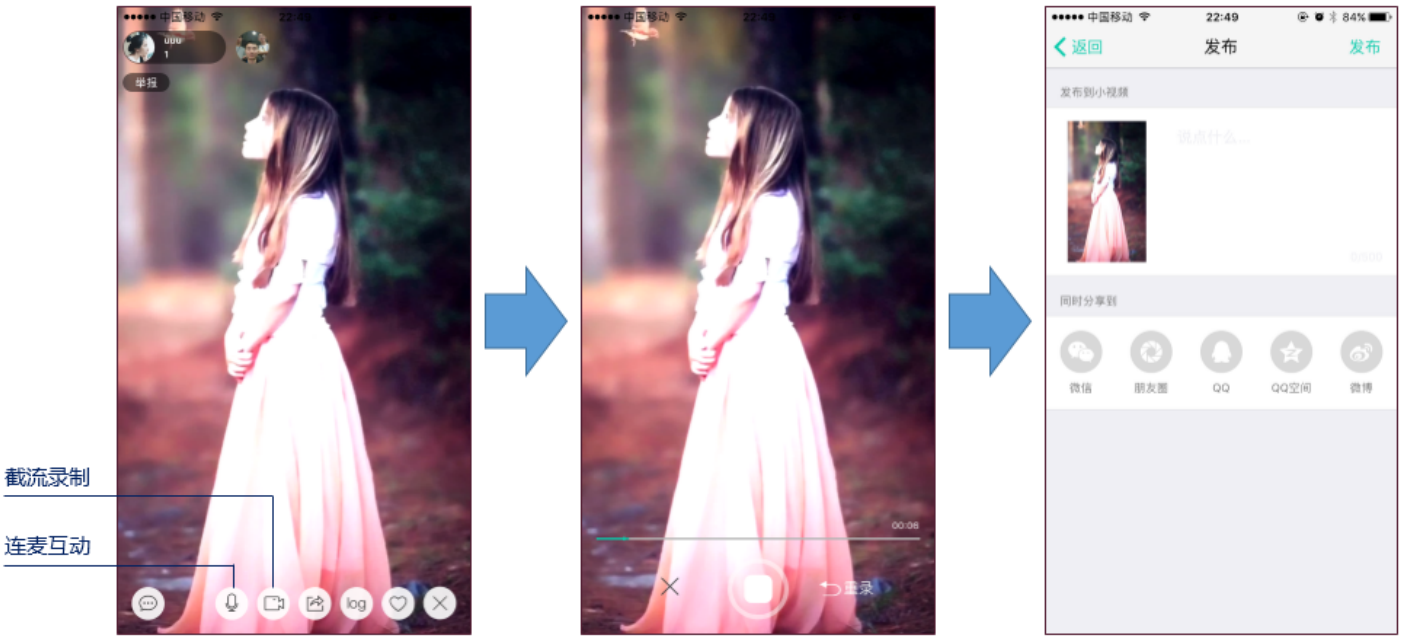
### Step 8: Screencap

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


```
mLivePlayer.snapshot(new ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
if (null != bmp) {
//Acquire screenshot bitmap
}
}
});
```

### Step 9: Recode the captured stream

As an extension in LVB playback scenarios, Recording Captured Stream means that during the LVB, the viewer can capture and record a segment of video by clicking the "Record" button and publish the recorded content via the video delivery platform (e.g. Tencent Cloud's VOD system) so that the content can be shared through UGC message on social platforms such as the "Moment" of WeChat.




```
//Specify an ITXVideoRecordListener to synchronize the progress and results of the recording
mLivePlayer.setVideoRecordListener(recordListener);
//Start the recording. It can be placed in the response function of the "Record" button. You can only record the video source, but not the other contents such as the on-screen comments.
mLivePlayer.startRecord(int recordType);
// ...
// ...
//End the recording. It can be placed in the response function of the "End" button
mLivePlayer.stopRecord();
```

- The progress of recording is indicated as a time value by ITXVideoRecordListener's onRecordProgress.
- The recorded file is in the format of MP4, and is indicated by ITXVideoRecordListener's onRecordComplete.
- TXUGCPublish is used to upload and publish videos. For more information on how to use TXUGCPublish, please see Short Video - Publish Files.

# Adjusting delay

The LVB feature of Tencent Cloud SDK, equipped with the self-developed playback engine, is not developed based on ffmped. Compared with open source players, it performs better in terms of LVB delay

control. We provide three delay adjusting modes, suitable for Live show, game LVB, and combined scenarios.

- **Performance comparison among the three modes**

| Control mode | Stutter rate | Average delay | Applicable scenarios | Principle description |
| --- | --- | --- | --- | --- |
| Speedy mode | High (relatively smooth) | 2s - 3s | Live show (online quiz) | It has the upper hand in delay control and is suitable for delay-sensitive scenarios. |
| Smooth mode | Lowest | >= 5s | Game LVB (Penguin e-Sports) | It is suitable for the LVB of ultra-high-bitrate games (such as battle royale games) |
| Auto mode | Network adaption | 2s - 8s | Combined scenarios | The better the viewers' network condition, the lower the latency, and vice versa. |

- **Interface codes of the three modes**

```
TXLivePlayConfig mPlayConfig = new TXLivePlayConfig();
//
//Auto mode
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(5);
//
//Speedy mode
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(1);
//
//Smooth mode
mPlayConfig.setAutoAdjustCacheTime(false);
mPlayConfig.setCacheTime(5);
//
mLivePlayer.setConfig(mPlayConfig);
//Launch the playback after the configuration
```

For more technical information on how to deal with stutter and delay problems, please see How to Deal with Stutter

# Ultra-low latency playback

Tencent Cloud LVB player supports ultra-low delay playback with a delay of about **400ms**, which can be used in scenarios that have high requirement for delay, such as **remote prize claw** and **joint broadcasting**. Notes about this feature:

- **You don't need to activate this feature**
  This feature does not need to be enabled in advance, but it requires that LVB streams reside in Tencent Cloud. Implementing low-delay linkage across cloud providers is difficult, in more than just technical terms.

- **Hotlink protection must be included in the URL**
  The playback URL cannot be a normal CDN URL. It must have a hotlink protection signature. For more information on how to calculate the hotlink protection signature, please see **txTime&txSecret**.

- **Specify the playback type as ACC**
  Specify the type as **PLAY_TYPE_LIVE_RTMP_ACC** when calling the startPlay function. The SDK pulls LVB streams using the RTMP-UDP protocol.

- **This feature has restrictions on concurrent playback**
  It supports 10 channels of concurrent playback at most. Instead of being set due to limited technical capabilities, this restriction is intended to encourage you to use this feature in interaction scenarios only (for example, for VJs only in joint broadcasting and for players only in prize claw scenarios), so that you do not incur any unnecessary costs in the mere pursuit of low delay (The price of low latency lines is higher than that of CDN lines).

- **The delay performance of OBS is unsatisfactory**
  If the push end is TXLivePusher, set `quality` to MAIN_PUBLISHER or VIDEO_CHAT using setVideoQuality. If the push end is Windows, use Windows SDK. Pushing with OBS leads to unsatisfactory delay due to the excessive accumulated data at the pusher end.

# Listening to SDK Events

You can bind a **TXLivePlayListener** to the TXLivePlayer object to receive notifications about the internal status of SDK through onPlayEvent (Event Notification) and onNetStatus (Quality Feedback).

### 1. Playback events

| Event ID | Value | Description |
| --- | --- | --- |
| PLAY_EVT_CONNECT_SUCC | 2001 | Connected to the server |
| PLAY_EVT_RTMP_STREAM_BEGIN | 2002 | Connected to the server and started to pull stream (thrown only if the playback URL is RTMP) |
| PLAY_EVT_RCV_FIRST_I_FRAME | 2003 | The network has received the first renderable video packet (IDR) |
| PLAY_EVT_PLAY_BEGIN | 2004 | Video playback begins. The "loading" icon stops flashing at this point |
| PLAY_EVT_PLAY_LOADING | 2007 | Video playback is being loaded. If video playback is resumed, this will be followed by a BEGIN event |
| PLAY_EVT_GET_MESSAGE | 2012 | Used to receive messages inserted into the audio/video stream. For details, please see Message Reception |

- **Do not hide the playback view after receiving PLAY_LOADING**
  Because the time length between PLAY_LOADING and PLAY_BEGIN can be different (sometimes 5 seconds, sometimes 5 milliseconds). Some customers consider hiding the view upon LOADING and displaying the view upon BEGIN, which will cause serious flickering (especially in LVB scenarios). It is recommended to place a translucent Loading animation on top of the video view.

### 2. Ending events

| Event ID | Value | Description |
| --- | --- | --- |
| PLAY_EVT_PLAY_END | 2006 | Video playback ends |
| PUSH_ERR_NET_DISCONNECT | -2301 | Network is disconnected. Too many failed reconnection attempts. Restart the playback for more retries |

- **How do I tell whether the LVB is over?**
  Because of the varying implementation principles of different standards, many LVB streams usually don't throw end events (2006) and it is expected that when the VJ stops pushing stream, the SDK will soon find that data stream pull fails (WARNING_RECONNECT) and attempt to retry until the PLAY_ERR_NET_DISCONNECT event is thrown after three failed attempts.

Therefore, you need to listen to both 2006 and -2301 and use the result as the events to determine the end of LVB.

### 3. Warning events

You don't need to consider the following events. We listed the information of these events for synchronization purposes, according to the SDK white-box design concept

| Event ID | Value | Description |
| --- | --- | --- |
| PLAY_WARNING_VIDEO_DECODE_FAIL | 2101 | Failed to decode the current video frame |
| PLAY_WARNING_AUDIO_DECODE_FAIL | 2102 | Failed to decode the current audio frame |
| PLAY_WARNING_RECONNECT | 2103 | Network disconnected, automatic reconnection has started (the PLAY_ERR_NET_DISCONNECT event will be thrown after three failed attempts) |
| PLAY_WARNING_RECV_DATA_LAG | 2104 | Unstable incoming packet from network: This may be caused by insufficient downstream bandwidth, or unstable outgoing stream at the VJ end |
| PLAY_WARNING_VIDEO_PLAY_LAG | 2105 | Stutter occurred during the current video playback |
| PUSH_WARNING_HW_ACCELERATION_FAIL | 2106 | Failed to start hard-decoding; Soft-decoding is used |
| PLAY_WARNING_VIDEO_DISCONTINUITY | 2107 | Current video frames are discontinuous and frame loss may occur |
| PLAY_WARNING_DNS_FAIL | 3001 | RTMP-DNS resolution failed (thrown only if the playback address is RTMP) |
| PLAY_WARNING_SEVER_CONN_FAIL | 3002 | Failed to connect to RTMP server (thrown only if the playback address is RTMP) |
| PLAY_WARNING_SHAKE_FAIL | 3003 | RTMP server handshaking failed (thrown only if the playback address is RTMP) |

## Video Width and Height

**What is the video resolution (in width and height)?**
This question cannot be figured out if SDK only obtains one URL string. To know the width and the height of a video image in pixels, SDK needs to access the cloud server until enough information is loaded to analyze the size of the video image. Therefore, SDK can only tell the video information to your application by notification.

The **onNetStatus** notification is triggered once per second to provide real-time feedback on the current status of the pusher. Like a car dashboard, it can offer you a picture about what is happening inside the SDK, so that you can keep track of current network conditions and video information.

| Evaluation Parameter | Description |
| --- | --- |
| NET_STATUS_CPU_USAGE | Current CPU utilization (instantaneous) |
| **NET_STATUS_VIDEO_WIDTH** | Video resolution - Width |
| **NET_STATUS_VIDEO_HEIGHT** | Video resolution - Height |
| NET_STATUS_NET_SPEED | Current speed at which network data is received |
| NET_STATUS_NET_JITTER | Network jitter status. A bigger jitter means a more unstable network |
| NET_STATUS_VIDEO_FPS | The video frame rate of the current stream media |
| NET_STATUS_VIDEO_BITRATE | Video bitrate of the current stream media (in Kbps) |
| NET_STATUS_AUDIO_BITRATE | Audio bitrate of the current stream media (in Kbps) |
| NET_STATUS_CACHE_SIZE | Buffer size (jitterbuffer). A buffer length of 0 means that stutter will occur in all probability |
| NET_STATUS_SERVER_IP | IP of the connected server |

# Gaming Screen Recording

Last updated：2018-08-10 16:21:58

## Mobile Phone Screencap

RTMP SDK 1.6.1 has begun to support screencap LVB on mobile phones, that is, the VJ's mobile phone screen can be used as the LVB source. Meanwhile, camera preview can be overlaid and used for scenarios that require mobile phone screens such as game LVB and mobile APP demo.


准备录屏直播　　录屏开始　　所见（主播）即所看（观众）

Implementation solutions to screencap are distinctly different on iOS and Android:

- **Android Platform**
  The feature is supported by Android 5.0 and later versions. The VJ only needs to install and start the LVB App before broadcasting, and then press the Home key to switch the App to the background. After that, all the contents on the VJ's screen can be used as LVB contents. This is how it works internally: The screencap API provided in the Android system is used to capture the screen, and the RTMP SDK underlying module performs encoding and RTMP push.

- **iOS platform**
  The feature is supported by iOS 10.0 and later versions, and is implemented based on the extension mode of iOS, that is, when a game LVB starts, the iOS will evoke the system extension (installed by the LVB App) which supports Screencap LVB, and transmit the screen images to this system extension which will in turn perform encoding and LVB push.

# Try out the Feature

In the Mini LVB Demo, we provide screencap on both mobile phone platforms based on Tencent Cloud RTMP SDK. You can scan the QR code below to install and try it out.


# Interfacing Guide

### Step 1: Add Activity

Paste an activity as follows to the manifest file

```
<activity
android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity"
android:theme="@android:style/Theme.Translucent"/>
```

### Step 2: Create a Pusher Object

Create a **TXLivePusher** object, which will be used later to complete the push task.

Before you create a LivePush object, you need to specify a **LivePushConfig** object to determine the configuration parameters of various LivePush aspects, such as push resolution, frames per second (FPS) and GOP (seconds per one I-frame).

The LivePushConfig object has been equipped with some parameters that we have repeatedly tuned upon creation. If you do not wish to customize these parameters, you can simply assign them to the LivePush object. If you have experience in the related field and want to adjust the default configuration, you can read the **Advance Guide**.

```
TXLivePusher mLivePusher = new TXLivePusher(getActivity());
mLivePushConfig = new TXLivePushConfig();
mLivePusher.setConfig(mLivePushConfig);
```

### Step 3: Launch Push

After the preparations in Step 1 and Step 2, you can use the following codes to start the push:

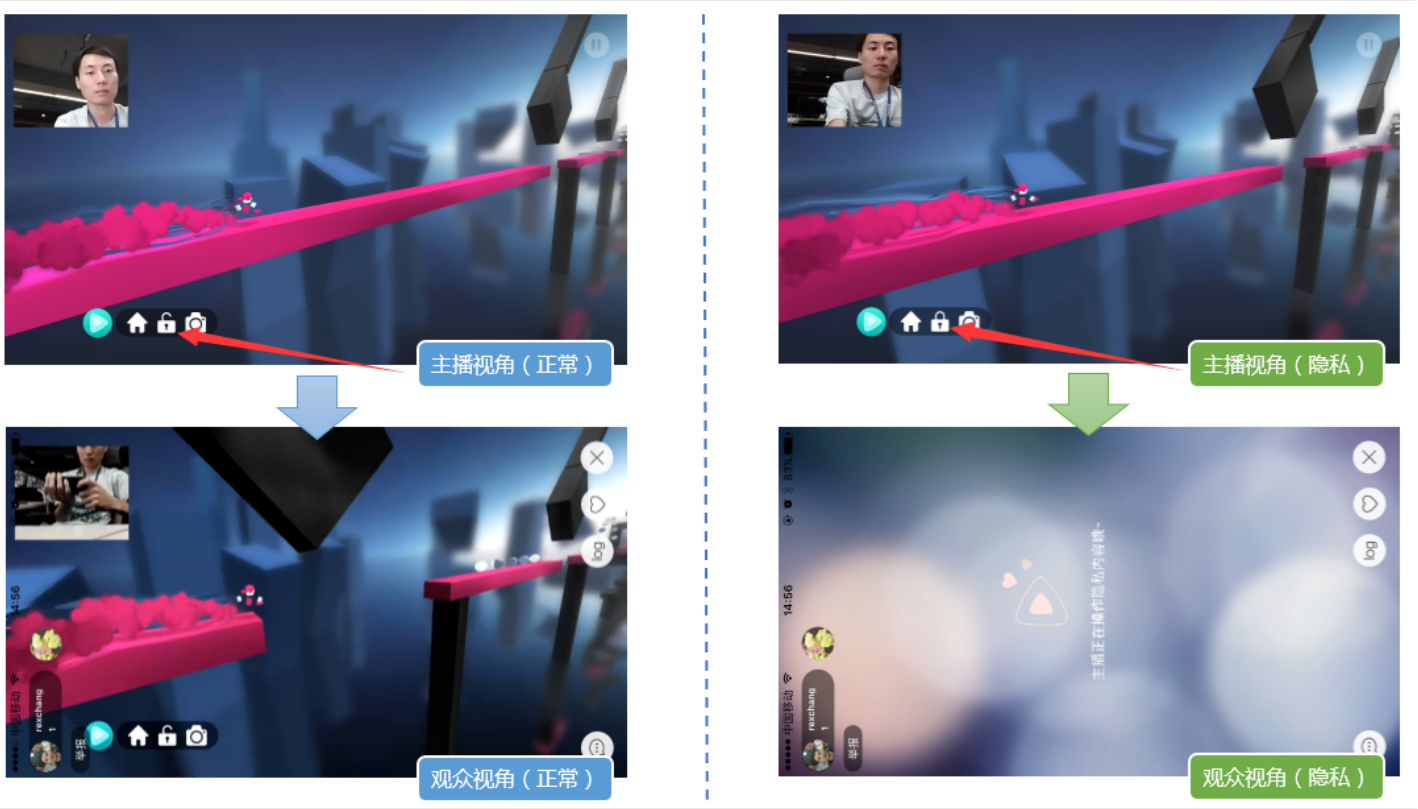
```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startPusher(rtmpUrl);
mLivePusher.startScreenCapture();
```

- **startPusher** is used to tell the RTMP SDK that which push URL the audio/video streams are being pushed to.
- **startScreenCapture** is used to start screencap. Since screencap is implemented based on the native capabilities of the Android system, for security reasons, Android will warn the user before the screencap is initiated by displaying a prompt: "an App will capture all the contents on your screen".

### Step 4: Privacy Mode

The privacy mode is a basic feature of screencap LVB: During the screencap LVB, if the VJ does not want certain operations (e.g., entering game account and password) to be seen by the audience, he/she can enable the **Privacy Mode**. While in privacy mode, the VJ's push stays available, and the screen keeps

being visible to the audience, only showing a waiting screen with prompt indicating that "the VJ is busy".




To realize such a feature, you can complete the interfacing process as follows:

- **4.1) Set pauseImg**
  Before the push, use the setPauseImg API of TXLivePushConfig to set a waiting image, e.g., "The VJ will switch back the screen soon...".

- **4.2) Privacy mode switch**
  On the floating window serving as the toolbar, add a button to enable/disable the privacy mode. The response logic of enabling the privacy mode is calling the TXLivePusher##pausePush API function; and the response logic of disabling the privacy mode is calling the TXLivePusher##resumePush API function.

```
public void triggerPrivateMode() {
if (mInPrivacy) {
Toast.makeText(getApplicationContext(), "Privacy mode enabled", Toast.LENGTH_SHORT).show();
mTVPrivateMode.setText(getString(R.string.private_mode_off));
mTVPrivateMode.setCompoundDrawables(mDrawableLockOn,null,null,null);
mPrivateBtn.setImageResource(R.mipmap.lock_off);
mTXLivePusher.resumePusher();
} else {
```

```
Toast.makeText(getApplicationContext(), "Privacy mode disabled", Toast.LENGTH_SHORT).show();
mTXLivePusher.pausePusher();
mPrivateBtn.setImageResource(R.mipmap.lock_on);
mTVPrivateMode.setText(getString(R.string.private_mode_on));
mTVPrivateMode.setCompoundDrawables(mDrawableLockOff,null,null,null);
}
mInPrivacy = !mInPrivacy;
}
```

### Step 5: Set Logo Watermark

Recent policies require that LVB videos must be marked with watermarks. With that in mind, we will focus on this feature that had seemed insignificant before.
Tencent Video Cloud currently supports two watermark settings. One is to set watermark in the push SDK, where the videos are marked with watermarks in the SDK before being encoded. Another is applying watermarks in the cloud. That is, the cloud resolves videos and adds Logo watermarks to them.

We suggest that you add watermarks with the SDK, because there are three major problems when watermarking in the cloud:
(1) This service increases the load on the cloud machine and is not free, which will increase your cost;
(2) It is not ideally compatible with certain situations such as resolution switching during the push process. This may cause problems like blurred screen.
(3) It may cause an additional 3-second video delay, which is caused by the transcode service.

SDK requires that watermark images are PNG format, because such images contain transparency information, which helps processes such as anti-aliasing. (Do not just change the extension of a JPG image to PNG in Windows and put it in. Professional PNG logos need to be processed by professional art designers)

```
//Set video watermark
mLivePushConfig.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 10, 10);
mLivePusher.setConfig(mLivePushConfig);
```

### Step 6: Recommended Definition

Three major factors affect video quality: **resolution**, **frame rate** and **bit rate**.

- **Resolution**
  Screencap LVB on mobile phones provides resolutions at three levels: 360*640, 540*960, and 720*1280. The API for relevant settings is setVideoResolution in TXLivePushConfig.

- **Frame Rate**
  You will feel significant stutter if FPS <=10. It is recommended to set the frame rate to 20 - 25 FPS for screencap LVB on mobile phones. API for this configuration is setVideoFPS in TXLivePushConfig.
- **Bit Rate**
  It refers to the size of data encoded by the encoder in each second (in kbps). For example, 800 kbps indicates that the encoder produces 800 kb (or 100 KB) of data per second. The API for this configuration is setVideoBitrate in TXLivePushConfig.

Compared to camera LVB, screencap LVB has many more uncertainties, the most significant one of which is the screencap scenario.
(1) At one extreme, the mobile phone screen remains unchanged, e.g., the desktop. In this case, the encoder can complete the task with very low bit rate output.

(2) At the other extreme, the mobile phone screen changes dramatically at all times, e.g., when the VJ is playing Temple Run. In this case, the bit rate must be at least 2 Mbps to ensure that there is no mosaic even for a resolution as ordinary as 540 * 960.

| Level | Resolution | FPS | Bit Rate-Game Screencap (Fishing Joy) | Bit Rate-Game Screencap (Temple Run) |
|---|---|---|---|---|
| SD | VIDEO_RESOLUTION_TYPE_360_640 | 20 | 800kbps | 1200kbps |
| HD | VIDEO_RESOLUTION_TYPE_540_960 | 20 | 1200kbps | 2000kbps |
| UHD | VIDEO_RESOLUTION_TYPE_720_1280 | 20 | 1600kbps | 3000kbps |

### Step 7: Remind the VJ "network is poor"

Step 9 will discuss how to handle RTMP SDK push events. **PUSH_WARNING_NET_BUSY** is very useful, it means: **the uplink network of the VJ is poor, and stutters have occurred in the viewer end.**

When you receive this WARNING, you can use the UI to remind VJ to change network egress, or get closer to the WiFi. Or tell him to say something like this: "Hello dear, I'm streaming! Stop reading eBay! What? Not eBay? Then stop downloading movies!"

### Step 8: Landscape/portrait screen adaption

Dynamic video switching logic between landscape/portrait screen has been realized in Tencent Cloud RTMP SDK, so there is no need to worry about this issue when using screencap LVB. When the VJ's mobile phone switches between landscape mode and portrait mode, the images seen at the viewer end stay consistent with the VJ end.

## Step 9: Add Custom Audio Data to SDK

If you wish to replace audio capture with your own logic, you need to add CUSTOM_MODE_AUDIO_CAPTURE to the CustomMode settings. Meanwhile, you also need to specify key information such as audio sampling rate and number of channels.

```
// (1) Set CustomMode as follows: Capture audio data by yourself; the SDK is responsible for encodi
ng and sending data only
_config.customModeType |= CUSTOM_MODE_AUDIO_CAPTURE;
//
// (2) Set audio encoding parameters: Audio sampling rate and number of channels
_config.audioSampleRate = 44100;
_config.audioChannels = 1;
```

Next, call **sendCustomPCMData** to insert your own PCM data to the SDK.

## Step 10: Event Handling

### Event Listening

RTMP SDK listens to push related events using the TXLivePushListener proxy. Note that the TXLivePushListener only listens to push events with prefix PUSH_.

### Normal Events

Events that are always prompted during a successful push. For example, receiving 1003 means that the system will start rendering the camera pictures

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_EVT_CONNECT_SUCC | 1001 | Successfully connected to Tencent Cloud push server |
| PUSH_EVT_PUSH_BEGIN | 1002 | Handshake with the server completed, everything is OK, ready to start push |
| PUSH_EVT_OPEN_CAMERA_SUCC | 1003 | The pusher has successfully started the camera (this will take 1-2 seconds on some Android phones) |

### Error Notification

The push cannot continue as the SDK detected critical problems. For example, the user disabled camera permission for the APP so the camera cannot be started.

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_ERR_OPEN_CAMERA_FAIL | -1301 | Failed to start the camera |

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_ERR_OPEN_MIC_FAIL | -1302 | Failed to start the microphone |
| PUSH_ERR_VIDEO_ENCODE_FAIL | -1303 | Video encoding failed |
| PUSH_ERR_AUDIO_ENCODE_FAIL -1304 | Audio encoding failed | |
| PUSH_ERR_UNSUPPORTED_RESOLUTION | -1305 | Unsupported video resolution |
| PUSH_ERR_UNSUPPORTED_SAMPLERATE | -1306 | Unsupported audio sampling rate |
| PUSH_ERR_NET_DISCONNECT | -1307 | Network disconnected. Reconnection attempts have failed for three times, thus no more retries will be performed. Please restart the push manually |

**Warning Events**

SDK detected some reparable problems. Most warning events will trigger protection logics or recovery logics that involve retrying, and in most of the cases the problems can be recovered. Don't make a fuss.

- PUSH_WARNING_NET_BUSY
  The VJ's network is busy. If you need UI prompts, this warning is relatively more useful (Step 10).

- PUSH_WARNING_SERVER_DISCONNECT
  The push request is rejected by the backend. This will trigger retry logic for a limited number of times and the push may succeed in a certain attempt. But in most cases, it is because the txSecret in the push address is miscalculated, or because the test address is occupied by others. Therefore, this warning is more conducive to your debugging.

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_WARNING_NET_BUSY | 1101 | Bad network condition: data upload is blocked because uplink bandwidth is too small |
| PUSH_WARNING_RECONNECT | 1102 | Network disconnected, automatic reconnection has started (auto reconnection will be stopped if it fails for three times) |

| Event ID | Value | Description |
| --- | --- | --- |
| PUSH_WARNING_HW_ACCELERATION_FAIL | 1103 | Failed to start hardware encoding. Software encoding is used |
| PUSH_WARNING_DNS_FAIL | 3001 | RTMP - DNS resolution failed (this will trigger retry process) |
| PUSH_WARNING_SEVER_CONN_FAIL | 3002 | Failed to connect to the RTMP server (this will trigger retry process) |
| PUSH_WARNING_SHAKE_FAIL | 3003 | RTMP server handshake failed (this will trigger retry process) |
| PUSH_WARNING_SERVER_DISCONNECT | 3004 | The RTMP server actively disconnected (this will trigger retry process) |

For the definition of all events, please see the header file **"TXLiveConstants.java"**

### Step 11: End Push

It is simple to end a push process, but proper cleaning work is required. Since only one TXLivePusher object can run at a time, improper cleaning work may adversely affect the next LVB.

```
//End the screencap LVB and perform proper cleanup work
public void stopPublish() {
mTXLivePusher.stopScreenCapture();
mTXLivePusher.setPushListener(null);
mTXLivePusher.stopPusher();
}
```

# Effect Feature

Last updated：2018-08-10 16:22:02

# Special Effects (Eye Enlarging, Face Slimming, Dynamic Effect and Green Screen)

## Feature Description

Special effects such as eye enlarging, face slimming, dynamic sticker and green screen, are privileged features developed based on face recognition technology of Tencent YouTu Lab team and makeup technology of Tencent Pitu team. By cooperating with the two teams, Tencent Cloud's Mini LVB team deeply integrates these special effects into the image processing process of RTMP SDK to achieve better video effects.

## Charges

The special effects use patented technology of Tencent YouTu Lab, with annual licensing fees being about **0.5 million CNY**(currently, the fees of similar image processing products in China are millions of CNY). If you need the feature, submit a ticket or call our customer service at 400-9100-100. Staff of the business department will provide a password for decoding the SDK package and apply to Tencent YouTu Lab for a trial license for you.

## Version Downloading

You can download the privileged SDK package at the bottom of the RTMP SDK page. The package is encrypted and you can get the password and the license file from our staff of the business department. After decompressing the package, you need to replace the non-privileged jar and so files in your project with the decompressed `txrtmpsdk.jar` , `libtxrtmpsdk.so` and other so files.

## Project Settings

### 1. Add the SDK

Copy txrtmpsdk.jar, libtxrtmpsdk.so and other so files in the SDK to the corresponding location in the project, such as in the libs folder

> Note: Privileged version only supports so files with armeabi architecture. Therefore, you should delete so files with other architectures in the App to avoid loading failure of so files.

### 2. Add resources

Copy the `camera` folder in the zip package to the assets directory of the project

> Note: The camera directory includes files such as resources for switching dynamic effects, and must be placed correctly under the assets directory, otherwise an error will occur

### 3. Import the license file

The features of the privileged version take effect only after the license verification is successful. You can apply to our staff of the business department for a 30-day free license for debugging.
After getting the license, you need to name it **YTFaceSDK.licence** and place it under the assets directory in the project.

> Each license is bound to a specific package name. Therefore, modifying the package name in the App can cause verification failure.
>
> The name of YTFaceSDK.license file is fixed, cannot be modified, and must be placed under the assets directory.
>
> For IOS and Android systems, you need to apply for only one license because one license can authorize the bundleid of an iOS system and the packageName of an Android system at the same time.

## Feature Calling

### 1. Dynamic effects

One dynamic effect template is included in one directory, which contains many resource files. Depending on the complexity of each dynamic effect, the number of directories and file sizes are different.

Download resources for dynamic effects from the background and decompress them to the Resource directory, and then you can get sample codes in the Mini LVB. In the codes, you can find download addresses of resources and thumbnails for dynamic effects. The format is as follows

```
https://st1.xiangji.qq.com/yunmaterials/{ID of the dynamic effect}Android.zip
https://st1.xiangji.qq.com/yunmaterials/{ID of the dynamic effect}.png
```

It is strongly recommended that you put the resources for dynamic effects on your own servers to prevent unnecessary impact caused by modifications of the Mini LVB.

After decompression, you can enable dynamic effects through the following API

```
/**
* setMotionTmpl is used to set the location of dynamic sticker files
* @param tmplPath
*/
public void setMotionTmpl(String tmplPath);
```

### 2. Green screen

Prepare a mp4 file for playback, and then you can enable the green screen effect by calling the following API

```
/**
* Set the green screen file: Currently, the supported pictures are in jpg/png format, and the supported videos are in mp4, 3gp and other Android-supported format
* API requirement 18
* @param path : Location of the green screen file. It supports two ways:
* 1. The resource file is put under the assets directory, and path is the file name
* 2. path is the absolute path of the file
*/
@TargetApi(18)
public void setGreenScreenFile(String path);
```

### 3. Eye enlarging and face slimming

Eye enlarging and face slimming features for SDK 2.0.0 are still under tense development and will be released as soon as possible.

# realtime LiveRoom

Last updated：2018-07-23 10:53:27

## Feature Description

**LVB+Joint Broadcasting** is an LVB mode commonly used in the **Live Show** and **Online Education** scenarios. With a good applicability to many scenarios, it supports online live broadcasting featuring both high concurrency and low cost, but also enables video chats between VJs and viewers via joint broadcasting.


Tencent Cloud provides "LVB+Joint Broadcasting" by using **LiveRoom**, a component consisting of Client and Server (both open source). For more information on how to interface with it, please see DOC. This document displays the API list for Client:

## LiveRoom

| Name | Description |
| --- | --- |
| setLiveRoomListener(ILiveRoomListener listener) | Sets liveroom callback |
| login(serverDomain, loginInfo, loginCallback) | Logs in to the RoomService backend |
| logout() | Logs out of the RoomService backend |
| getRoomList(int index, int count, GetRoomListCallback callback) | Gets the room list (optional, you can select your room list.) |
| getAudienceList(String roomID, final GetAudienceListCallback callback) | Gets a list of viewers in a room (a maximum of the last 30 viewers who entered the room are returned.) |
| createRoom(String roomID, String roomInfo, CreateRoomCallback cb) | VJ: Creates a room (roomID can be left blank.) |
| enterRoom(String roomID, TXCloudVideoView videoView, EnterRoomCallback cb) | Viewer: Enters a room |
| exitRoom(ExitRoomCallback callback) | VJ or viewer: Exits a room |
| startLocalPreview(TXCloudVideoView videoView) | VJ or joint broadcasting viewer: Enables camera preview |
| stopLocalPreview() | Disables camera preview |
| requestJoinPusher(int timeout, RequestJoinPusherCallback callback) | Viewer: Sends a joint broadcasting request |
| joinPusher(final JoinPusherCallback cb) | Viewer: Joins joint broadcasting |
| quitPusher(final QuitPusherCallback cb) | Viewer: Quits joint broadcasting |
| acceptJoinPusher(String userID) | VJ: Accepts a joint broadcasting request from the viewer |

| Name | Description |
| --- | --- |
| rejectJoinPusher(String userID, String reason) | VJ: Rejects a joint broadcasting request from the viewer |
| kickoutSubPusher(String userID) | VJ: Kicks a viewer out of joint broadcasting |
| getOnlinePusherList(final GetOnlinePusherListCallback callback) | VJ PK: Gets a list of online VJs |
| startPlayPKStream(final String playUrl, TXCloudVideoView videoView, final PKStreamPlayCallback callback) | VJ PK: Starts playing back each other's video streams |
| stopPlayPKStream() | VJ PK: Stops playing back each other's video streams |
| sendPKRequest(String userID, int timeout, final RequestPKCallback callback) | VJ PK: Sends a PK request |
| sendPKFinishRequest(String userID) | VJ PK: Sends a request to finish PK |
| acceptPKRequest(String userID) | VJ PK: Accepts a PK request |
| rejectPKRequest(String userID, String reason) | VJ PK: Rejects a PK request |
| addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback) | VJ: Plays back the remote video images of the joint broadcasting viewer |
| deleteRemoteView(PusherInfo pusherInfo) | VJ: Removes the remote video images of the joint broadcasting viewer |
| sendRoomTextMsg(String message, SendTextMessageCallback callback) | Sends a text (on-screen comment) message |
| sendRoomCustomMsg(String cmd, String message, SendCustomMessageCallback callback) | Sends a custom message (gives a "Like" or flower) |

| Name | Description |
| --- | --- |
| startScreenCapture() | Starts screen capturing (only for Android) |
| stopScreenCapture() | Stops screen capturing (only for Android) |
| switchToBackground() | App switches from foreground to background |
| switchToForeground() | App switches from background to foreground |
| setBeautyFilter(style, beautyLevel, whiteningLevel, ruddyLevel) | Sets beauty filter effects |
| switchCamera() | Switches between front and rear cameras. Dynamic switching is supported during push |
| setMute(mute) | Enables Mute |
| setMirror(enable) | Sets mirroring for images (this API works on the effect at the viewer end only; the VJ end always has the mirroring effect available) |
| playBGM(String path) | Starts background music ("path" indicates the path to the music file) |
| stopBGM() | Stops background music |
| pauseBGM() | Pauses background music |
| resumeBGM() | Resumes background music |
| setMicVolume(x) | Sets microphone volume for audio mixing |

| Name | Description |
| --- | --- |
| setBGMVolume(x) | Sets background music volume for audio mixing |
| getMusicDuration(fileName) | Gets background music duration |
| startRecord(recordType) | Starts video recording |
| stopRecord() | Stops video recording |
| setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener) | Sets video recording callback |
| incCustomInfo(fieldName, count) | Increases the custom value (fieldName) of a room |
| decCustomInfo(fieldName, count) | Decreases the custom value (fieldName) of a room |
| updateSelfUserInfo(userName, userAvatar) | Updates user's information of liveroom |
| setPauseImage(bitmap) | Sets the images to be pushed when switching to the background |

## ILiveRoomListener

| Name | Description |
| --- | --- |
| onGetPusherList(pusherList) | Notification: The list of existing pushers in the room (the number of remote video streams) |
| onPusherJoin(pusherInfo) | Notification: A new pusher joined the room (notifies you of addition of a remote video stream) |
| onPusherQuit(pusherInfo) | Notification: A pusher left the room (notifies you of subtraction of a remote video stream) |
| onRecvJoinPusherRequest(userID, userName, userAvatar) | Notification: VJ receives a joint broadcasting request from a viewer |

| Name | Description |
| --- | --- |
| onKickOut() | Viewer: Viewer gets notified of being kicked out by the primary VJ |
| onRecvPKRequest(String userID, String userName, String userAvatar, String streamUrl) | Receives a PK request |
| onRecvPKFinishRequest(String userID) | Receives a request to finish PK |
| onRecvRoomTextMsg(roomID, userID, userName, userAvatar, message) | Chat room: Receives a text message |
| onRecvRoomCustomMsg(roomID, userID, userName, userAvatar, cmd, message) | Chat room: Receives a custom message |
| onRoomClosed(roomID) | Notification: Notifies you of the room being closed |
| onDebugLog(log) | LOG: Log callback |
| onError(errorCode, errorMessage) | ERROR: Error callback |

# Details of LiveRoom APIs

### 1. setLiveRoomListener

- API definition: void setLiveRoomListener(ILiveRoomListener listener)
- API description: Sets the ILiveRoomListener callback. For the callback function, please see the ILiveRoomListener API description.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| listener | ILiveRoomListener | A liveroom callback API |

- Sample code:

```
mLiveRoom.setLiveRoomListener(new ILiveRoomListener() {
@Override
void onPusherJoin(PusherInfo pusherInfo) {
// ...
}
```

```
@Override
void onPusherQuit(PusherInfo pusherInfo) {
// ...
}

.....
});
```

## 2. login

- API definition: void login(String serverDomain, final LoginInfo loginInfo, final LoginCallback callback)
- API description: Logs in to the RoomService backend. You can specify whether to use the Tencent Cloud RoomService or the user-deployed RoomService.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| serverDomain | String | Server address of RoomService. For more information, please see DOC. |
| loginInfo | LoginInfo | The login parameter. For more information, please see DOC. |
| callback | LoginCallback | Callback to verify whether the login is successful |

- Sample code:

```
final String DOMAIN = "https://room.qcloud.com/weapp/live_room ";
LoginInfo loginInfo = new LoginInfo();
loginInfo.sdkAppID = sdkAppID;
loginInfo.userID = userID;
loginInfo.userSig = userSig;
loginInfo.accType = accType;
loginInfo.userName = userName;
loginInfo.userAvatar = userAvatar;
mLiveRoom.login(DOMAIN, loginInfo, new LiveRoom.LoginCallback() {
@Override
public void onError(int errCode, String errInfo) {
//
}

@Override
public void onSuccess(String userId) {
```

```
//The userId is returned upon successful login.
}
});
```

### 3. logout

- API definition: void logout();
- API description: Logs out of the RoomService backend
- Sample code:

```
mLiveRoom.logout();
```

### 4. getRoomList

- API definition: void getRoomList(int index, int count, GetRoomListCallback callback)
- API description: Pulls the room list. The parameters index and count are used to handle paging, which means you can pull "count" rooms from the room numbered "index". This API is not required to be called, and you can continue using it if you already have your own room list service modules.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| index | int | The index of the room from which pull starts |
| count | int | Number of rooms to be returned via RoomService |
| callback | GetRoomListCallback | Callback of pulling a room list |

- Sample code:

```
//The pulling begins from the number 0, and ends when 20 rooms are pulled.
mLiveRoom.getRoomList(0, 20, new LiveRoom.GetRoomListCallback() {
@Override
public void onSuccess(ArrayList<RoomInfo> data) {
//For information on each room, please see the definition of RoomInfo.
}

@Override
public void onError(int errCode, String e) {
```

```
}
});
```

### 5. getAudienceList

- API definition: void getAudienceList(String roomID, final GetAudienceListCallback callback)
- API description: Gets the list of viewers in a room and returns only the last 30 viewers who entered the room.

### 6. createRoom

- API definition: void createRoom(final String roomID, final String roomInfo, final CreateRoomCallback cb)
- API description: Creates a room at RoomService backend.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| roomID | String | You can specify an ID for a new room with the parameter roomID, or leave it unspecified. If you do not specify an ID for the room, RoomService will automatically create a new roomID and return it to you through CreateRoomCallback. |
| roomInfo | String | To be customized by the creator. This information is returned via getRoomList |
| cb | CreateRoomCallback | Creates room creation result callback |

- Sample code:

```
String roomInfo = mTitle;
try {
roomInfo = new JSONObject()
.put("title", mTitle)
.put("frontcover", mCoverPicUrl)
.put("location", mLocation)
.toString();
} catch (JSONException e) {
roomInfo = mTitle;
}
mLiveRoom.createRoom("", roomInfo, new LiveRoom.CreateRoomCallback() {
@Override
public void onSuccess(String roomId) {
```

```
Log.w(TAG,String.format("Room %s created successfully", roomId));
}

@Override
public void onError(int errCode, String e) {
Log.w(TAG,String.format("Error while creating the room, code=%s,error=%s", errCode, e));
}
});
```

### 7. enterRoom

- API definition: void enterRoom(String roomID, TXCloudVideoView videoView, EnterRoomCallback cb)
- API description: (Viewer) enters a room.
- Sample code:

```
mLiveRoom.enterRoom(mGroupId, mTXCloudVideoView, new LiveRoom.EnterRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "enter room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "enter room success ");
}
});
```

### 8. exitRoom

- API definition: void exitRoom(final ExitRoomCallback callback)
- API description: VJ (or viewer) exits the room.
- Sample code:

```
mLiveRoom.exitRoom(new LiveRoom.ExitRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
```

```
TXLog.w(TAG, "exit room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "exit room success ");
}
});
```

### 9. startLocalPreview

- API definition: void startLocalPreview(TXCloudVideoView view)
- API description: VJ (or joint broadcasting viewer) enables camera preview. The front camera is used by default, and switchCamera is used to switch between front and rear cameras.
- Sample code:

```
TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mLiveRoom.startLocalPreview(mCaptureView);
```

### 10. stopLocalPreview

- API definition: void stopLocalPreview()
- API description: VJ (or viewer) disables camera preview.
- Sample code:

```
mLiveRoom.stopLocalPreview();
```

### 11. requestJoinPusher

- API definition: void requestJoinPusher(int timeout, RequestJoinPusherCallback callback)
- API description: This API is called when (viewer) sends a request for joint broadcasting with the VJ.
- Sample code:

```
mLiveRoom.requestJoinPusher(10, new LiveRoom.RequestJoinPusherCallback() {
@Override
public void onAccept() {
mLiveRoom.startLocalPreview(videoView);
mLiveRoom.setPauseImage(BitmapFactory.decodeResource(getResources(), R.drawable.pause_publish));
mLiveRoom.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
```

```
mLiveRoom.joinPusher(new LiveRoom.JoinPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {
mLiveRoom.startLocalPreview(videoView);
Toast.makeText(LivePlayActivity.this, "joint broadcasting failed:" + errInfo, Toast.LENGTH_SHORT).sho
w();
}

@Override
public void onSuccess() {

}
});
}

@Override
public void onReject(String reason) {
Toast.makeText(LivePlayActivity.this, "VJ rejected your joint broadcasting request", Toast.LENGTH_SH
ORT).show();
}

@Override
public void onTimeOut() {
Toast.makeText(LivePlayActivity.this, "Joint broadcasting request timed out, and the VJ made no resp
onse", Toast.LENGTH_SHORT).show();
}

@Override
public void onError(int code, String errInfo) {
Toast.makeText(LivePlayActivity.this, "Joint broadcasting request failed:" + errInfo, Toast.LENGTH_SH
ORT).show();
}
});
```

### 12. joinPusher

- API definition: void joinPusher(final JoinPusherCallback cb)
- API description: (Viewer) begins pushing, and joins joint broadcasting, but whether the joint broadcasting can be successful depends on the callback result of JoinPusherCallback.
- Sample code:

```
mLiveRoom.joinPusher(new LiveRoom.JoinPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {
```

```
mLiveRoom.startLocalPreview(videoView);
Toast.makeText(LivePlayActivity.this, "Joint broadcasting failed:" + errInfo, Toast.LENGTH_SHORT).sho
w();
}

@Override
public void onSuccess() {

}
});
```

### 13. quitPusher

- API definition: void quitPusher(final QuitPusherCallback cb)
- API description: (Viewer) quits joint broadcasting.
- Sample code:

```
mLiveRoom.quitPusher(new LiveRoom.QuitPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {

}

@Override
public void onSuccess() {

}
});
```

### 14. acceptJoinPusher

- API definition: void acceptJoinPusher(String userID)
- API description: (VJ) accepts viewer's request for joint broadcasting.
- Sample code:

```
mLiveRoom.acceptJoinPusher(userId);
```

### 15. rejectJoinPusher

- API definition: void rejectJoinPusher(String userID, String reason)
- API description: (VJ) rejects viewer's request for joint broadcasting.
- Sample code:

```
mLiveRoom.rejectJoinPusher(userId, "");
```

### 16. kickoutSubPusher

- API definition: void kickoutSubPusher(String userID)
- API description: VJ kicks a viewer out of joint broadcasting.
- Sample code:

```
mLiveRoom.kickoutSubPusher(userId);
```

### 17. getOnlinePusherList

- API definition: void getOnlinePusherList(final GetOnlinePusherListCallback callback)
- API description: VJ PK: Gets a list of online VJs.
- Sample code:

```
mLiveRoom.getOnlinePusherList(new LiveRoom.GetOnlinePusherListCallback() {
@Override
public void onError(int errCode, String errInfo) {

}

@Override
public void onSuccess(final ArrayList<PusherInfo> pusherList) {

}
});
```

### 18. startPlayPKStream

- API definition: void startPlayPKStream(final String playUrl, TXCloudVideoView videoView, final PKStreamPlayCallback callback)
- API description: VJ PK: Starts playing each other's streams
- Sample code:

```
mLiveRoom.startPlayPKStream(streamUrl, videoView, new LiveRoom.PKStreamPlayCallback() {
@Override
public void onPlayBegin() {

}
```

```
@Override
public void onPlayError() {

}
});
```

### 19. stopPlayPKStream

- API definition: void stopPlayPKStream()
- API description: VJ PK: Stops playing each other's streams
- Sample code:

```
mLiveRoom.stopPlayPKStream()
```

### 20. sendPKRequest

- API definition: void sendPKRequest(String userID, int timeout, final RequestPKCallback callback)
- API description: VJ PK: Sends a PK request
- Sample code:

```
mLiveRoom.sendPKRequest(userID, 10, new LiveRoom.RequestPKCallback() {
@Override
public void onAccept(String streamUrl) {

}

@Override
public void onReject(String reason) {

}

@Override
public void onTimeOut() {

}

@Override
public void onError(int code, String errInfo) {

}
});
```

### 21. sendPKFinishRequest

- API definition: void sendPKFinishRequest(String userID)
- API description: VJ PK: Sends a request to end PK
- Sample code:

```
mLiveRoom.sendPKFinishRequest(userID)
```

### 22. acceptPKRequest

- API definition: void acceptPKRequest(String userID)
- API description: VJ PK: Accepts a PK request
- Sample code:

```
mLiveRoom.acceptPKRequest(userID);
```

### 23. rejectPKRequest

- API definition: void rejectPKRequest(String userID, String reason)
- API description: VJ PK: Rejects a PK request
- Sample code:

```
mLiveRoom.rejectPKRequest(userID, "");
```

### 24. addRemoteView

- API definition: void addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback)
- API description: (VJ) plays the remote video image of a viewer in joint broadcasting. This API is called when onPusherJoin (notification of new viewer joining joint broadcasting) is received.
- Sample code:

```
public void onPusherJoin(PusherInfo pusherInfo) {
......
mLiveRoom.addRemoteView(videoView, pusherInfo, new LiveRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
}
```

```
@Override
public void onPlayError() {
}
});
.....
}
```

### 25. deleteRemoteView

- API definition: void deleteRemoteView(final PusherInfo pusherInfo)
- API description: Stops pushing the video stream of a viewer in joint broadcasting. This API is called when onPusherQuit (a viewer quits the joint broadcasting) is received.
- Sample code:

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mLiveRoom.deleteRemoteView(pusherInfo);
.....
}
```

### 26. sendRoomTextMsg

- API definition: void sendRoomTextMsg(@NonNull String message, final SendTextMessageCallback callback)
- API description: Sends a text message. Other members in the room will receive a notification via onRecvRoomTextMsg.
- Sample code:

```
mLiveRoom.sendRoomTextMsg("hello", new LiveRoom.SendTextMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.d(TAG, "sendRoomTextMsg error:");
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomTextMsg success:");
}
});
```

### 27. sendRoomCustomMsg

- API definition: void sendRoomCustomMsg(@NonNull String cmd, @NonNull String message, final SendCustomMessageCallback callback)
- API description: Sends a custom message. Other members in the room will receive a notification via onRecvRoomCustomMsg.
- Sample code:

```
mLiveRoom.sendRoomCustomMsg(String.valueOf(TCConstants.IMCMD_DANMU),
"hello ", new LiveRoom.SendCustomMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.w(TAG, "sendRoomDanmuMsg error: "+errInfo);
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomDanmuMsg success");
}
});
```

### 28. startScreenCapture

- API definition: void startScreenCapture()
- API description: Enables screen capturing. Since screencap is implemented based on the native capabilities of the Android system, for security reasons, Android will warn the user before the screencap is initiated by displaying a prompt: "an App will capture all the contents on your screen".

Note: This API takes effect just on Android API 21. Screencap and camera preview are mutually exclusive, which means only one of them can be effective at a time.

### 29. stopScreenCapture

- API definition: void stopScreenCapture()
- API description: Stops screen capturing.

### 30. switchToBackground

- API definition: void switchToBackground()

- API description: Switches from foreground to background, stops collecting camera data, and pushes default pictures.

### 31. switchToForeground

- API definition: void switchToForeground()
- API description: Switches from background to foreground, and starts collecting camera data.

### 32. setBeautyFilter

- API definition: boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel)
- API description: Sets beauty filter style, dermabrasion level, whitening level, and blushing level.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| style | int | Dermabrasion style: 0: Smooth 1: Natural 2: Hazy |
| beautyLevel | int | Dermabrasion level: Value range: 0-9. 0 means disabling beauty filter. Default is 0, i.e., disabling beauty filter |
| whiteningLevel | int | Whitening level: Value range: 0-9. 0 means disabling whitening. Default is 0, i.e., disabling whitening |
| ruddyLevel | int | Blushing level: Value range: 0-9. 0 means disabling blushing. Default is 0, i.e., disabling blushing |

- Sample code:

```
mLiveRoom.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
```

### 33. switchCamera

- API definition: void switchCamera()
- API description: Switches between cameras. When the front camera is in use, calling this API enables a switch from the front camera to the rear camera, and vice versa. This API takes effect only if it is called after camera preview (startCameraPreview(TXCloudVideoView)) is enabled. The front camera is used by default when the SDK enables camera preview.

### 34. setMute

- API definition: void setMute(mute)

- API description: Enables Mute Once Mute is enabled, the SDK shifts from pushing microphone-collected sounds to pushing mute.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| mute | boolean | Whether to enable Mute |

### 35. setMirror

- API definition: void setMirror(boolean enable)
- API description: Sets horizontal mirroring at the viewer end. Note that this API only works on the viewer end, not the VJ (pusher) end. The mirroring effect is always seen from the pusher end. The image is seen as mirrored from the pusher end when the front camera is in use, and non-mirrored when the rear camera is in use.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| enable | boolean | "true" indicates the image is seen as mirrored, and "false" indicates the image is seen as non-mirrored. |

- Sample code:

```
//The image is seen as mirrored at the viewer end
mLiveRoom.setMirror(true);
```

### 36. playBGM

- API definition: boolean playBGM(String path)
- API description: Starts background music. This API is used for audio mixing, for example, mixing background music with sounds collected from the microphone for playback. If the playback is successful, a value of "true" is returned. If the playback fails, a value of "false" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| path | String | The background music file is located in the absolute path in the phone |

### 37. stopBGM

- API definition: boolean stopBGM()
- API description: Stops background music. If the playback ends successfully, a value of "true" is returned. If the playback fails to end, a value of "false" is returned.

### 38. pauseBGM

- API definition: boolean pauseBGM()
- API description: Pauses background music. If the playback pauses successfully, a value of "true" is returned. If the playback fails to pause, a value of "false" is returned.

### 39. resumeBGM

- API definition: boolean resumeBGM()
- API description: Resumes background music. If the playback resumes successfully, a value of "true" is returned. If the playback fails to resume, a value of "false" is returned.

### 40. setMicVolume

- API definition: boolean setMicVolume(float x)
- API description: Sets microphone volume for audio mixing. If the microphone volume is set successfully, a value of "true" is returned. If the microphone volume fails to be set, a value of "false" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| x | float | Volume: Normal volume is 1. The recommended value is 0-2. If you need to turn up the volume, you can set it to a larger value. It is recommended to add a slider in the UI to allow VJs to set volume on their own |

### 41. setBGMVolume

- API definition: boolean setBGMVolume(float x)
- API description: Sets background music volume for audio mixing. If the background music volume is set successfully, a value of "true" is returned. If the background music volume fails to be set, a value of "false" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |

| Parameter | Type | Description |
| --- | --- | --- |
| x | float | Volume: Normal volume is 1. The recommended value is 0-2. If you need to turn up the volume, you can set it to a larger value. It is recommended to add a slider in the UI to allow VJs to set volume on their own |

### 42. getMusicDuration

- API definition: int getMusicDuration(String path)
- API description: Gets background music duration. The returned value is in seconds.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| path | String | Gets the duration of the current music if path == null and the duration of music under the path if path != null |

### 43. startRecord

- API definition: int startRecord(int recordType)
- API description: Starts recording video. This API is used at the viewer end to save the videos the viewers are watching as a local file in real time.
- Note: This API can be called only after the enterRoom operation is successful. Additionally, the generated video files are managed by your application layer code, and the SDK does not clean them.
- If the recording starts successfully, "0" is returned. If the recording is in progress, "-1" is returned. If the pushing has not started and the recording fails to start, "-2" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| recordType | int | Recording type, only video-only recording is supported TXRecordCommon.RECORD_TYPE_STREAM_SOURCE |

- Sample code:

```
mLiveRoom.startRecord(TXRecordCommon.RECORD_TYPE_STREAM_SOURCE);
```

### 44. stopRecord

- API definition: int stopRecord()

- API description: Stops recording video. The recording result is asynchronously notified by means of recording callback.

### 45. setVideoRecordListener

- API definition: void setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)
- API description: Sets video recording callback to receive the video recording progress and result.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| listener | TXRecordCommon.ITXVideoRecordListener | Video recording callback |

- Sample code:

```
mLiveRoom.setVideoRecordListener(new TXRecordCommon.ITXVideoRecordListener(){
@Override
public void onRecordEvent(int event, Bundle param) {
}

@Override
public void onRecordProgress(long milliSecond) {
Log.d(TAG, "record progress:" + milliSecond);
}

@Override
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
if (result.retCode == TXRecordCommon.RECORD_RESULT_OK) {
String videoFile = result.videoPath;
String videoCoverPic = result.coverPath;
} else {
Log.d(TAG, "record error:" + result.retCode + ", error msg:" + result.descMsg);
}
}
});
```

### 46. incCustomInfo

- API definition: void incCustomInfo(String fieldName, int count)
- API description: Increases the custom fieldName count. This API is used to count the number of "likes", gifts, and others of a room. The cumulative values can be obtained from the "custom" field of the "roominfo" parameter.

- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| fieldName | String | Field name that requires counting |
| count | int | Increment value for each count, which is typically 1 |

- Sample code:

```
mLiveRoom.incCustomInfo("praise",1);
```

### 47. decCustomInfo

- API definition: void decCustomInfo(String fieldName, int count)
- API description: Decreases the custom fieldName count. This API is used to count the number of "likes", gifts, and others of a room. The cumulative values can be obtained from the "custom" field of the "roominfo" parameter.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| fieldName | String | Field name that requires counting |
| count | int | Decrement value for each count, which is typically 1 |

### 48. updateSelfUserInfo

- API definition: void updateSelfUserInfo(String userName, String userAvatar)
- API description: Updates the nickname and profile photo of a new user. This API is used to update the liveroom information in real time after the nickname and profile photo of a user are modified

### 49. setPauseImage

- API definition: void setPauseImage(Bitmap bitmap)
- API description: Sets the images to be pushed when switching from foreground to background.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |

| Parameter | Type | Description |
| --- | --- | --- |
| bitmap | Bitmap | Background image bitmap |

# Details of ILiveRoomListener APIs

### 1. onGetPusherList

- API definition: void onGetPusherList(List pusherList)
- API description: The list of existing pushers in the room (the number of remote video streams). New viewers will receive this notification when they enter the room. In the callback, you can call addRemoteView to playback the video of an existing viewer in the room.
- Sample code:

```
public void onGetPusherList(List<PusherInfo> pusherList) {
.....
for (PusherInfo pusherInfo : pusherInfoList) {
mLiveRoom.addRemoteView(videoView, pusherInfo, new LiveRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
}
.....
}
```

### 2. onPusherJoin

- API definition: void onPusherJoin(PusherInfo pusherInfo)
- API description: When a new viewer enters a room, the primary VJ and other viewers will receive this notification. In the callback, you can call addRemoteView to playback the video of this new viewer.

- Sample code:

```
public void onPusherJoin(final PusherInfo pusherInfo) {
......
mLiveRoom.addRemoteView(videoView, pusherInfo, new LiveRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
......
}
```

### 3. onPusherQuit

- API definition: void onPusherQuit(PusherInfo pusherInfo)
- API description: The primary VJ and other viewers will receive this notification when a viewer exits the room. In the callback, you can call deleteRemoteView to stop playing back the video of this viewer.
- Sample code:

```
public void onPusherQuit(PusherInfo pusherInfo) {
......
mLiveRoom.deleteRemoteView(pusherInfo);
......
}
```

### 4. onRecvJoinPusherRequest

- API definition: void onRecvJoinPusherRequest(String userID, String userName, String userAvatar)
- API description: The VJ receives this notification when a viewer requests joint broadcasting with this VJ. The VJ can either accept (acceptJoinPusher) or reject (rejectJoinPusher) the request in the callback.
- Sample code:

```
public void onRecvJoinPusherRequest(final String userId, String userName, String userAvatar) {
final AlertDialog.Builder builder = new AlertDialog.Builder(mActivity)
.setCancelable(true)
```

```
.setTitle("Prompt")
.setMessage(userName + "initiated a joint broadcasting request with you")
.setPositiveButton("Accept", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.acceptJoinPusher(userId);
dialog.dismiss();
}
})
.setNegativeButton("Reject", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.rejectJoinPusher(userId, "VJ rejected your joint broadcasting request");
dialog.dismiss();
}
});
}
```

### 5. onKickOut

- API definition: void onKickOut()
- API description: The viewer will receive this notification when he/she is kicked out of joint broadcasting by the VJ. In the callback, you can stop local preview and exit the live room.
- Sample code:

```
public void onKickOut() {
......
mLiveRoom.stopLocalPreview();
mLiveRoom.quitPusher(new LiveRoom.QuitPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {

}

@Override
public void onSuccess() {

}
});
......
}
```

### 6. onRecvPKRequest

- API definition: void onRecvPKRequest(String userID, String userName, String userAvatar, String streamUrl)
- API description: When a VJ calls sendPKRequest to send a PK request to another VJ, the another VJ will receive this callback notification. In this callback, you can display a pop-up window indicating the reception of a PK request and asking the user whether he/she will accept or reject it.
- Sample code:

```
@Override
public void onRecvPKRequest(final String userID, final String userName, final String userAvatar, fina
l String streamUrl){
final AlertDialog.Builder builder = new AlertDialog.Builder(mActivity)
.setCancelable(true)
.setTitle("Prompt")
.setMessage(userName + "initiated a PK request with you")
.setPositiveButton("Accept", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.acceptPKRequest(userID);
mLiveRoom.startPlayPKStream(streamUrl, videoView, new LiveRoom.PKStreamPlayCallback() {
@Override
public void onPlayBegin() {

}

@Override
public void onPlayError() {

}
});
dialog.dismiss();
}
})
.setNegativeButton("Reject", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.rejectPKRequest(userID, "VJ rejected your PK request");
dialog.dismiss();
}
});

final AlertDialog alertDialog = builder.create();
alertDialog.setCancelable(false);
alertDialog.setCanceledOnTouchOutside(false);
```

```
alertDialog.show();
}
```

### 7. onRecvPKFinishRequest

- API definition: void onRecvPKFinishRequest(String userID)
- API description: When a VJ calls sendPKFinishRequest to notify another VJ that the PK has ended, the another VJ will receive this callback notification. In this callback, call stopPlayPKStream to end the PK and perform clean-up.
- Sample code:

```
@Override
public void onRecvPKFinishRequest(final String userID){
mLiveRoom.stopPlayPKStream();
}
```

### 8. onRecvRoomTextMsg

- API definition: void onRecvRoomTextMsg(String roomID, String userID, String userName, String userAvatar, String message)
- API description: When sendRoomTextMsg is called at the VJ or viewer end, the VJ or viewers in the room will receive this notification.
- Sample code:

```
public void onRecvRoomTextMsg(String roomid, String userid, String userName, String userAvatar, St
ring message) {
//do nothing
}
```

### 9. onRecvRoomCustomMsg

- API definition: void onRecvRoomCustomMsg(String roomID, String userID, String userName, String userAvatar, String cmd, String message)
- API description: When sendRoomCustomMsg is called at the VJ or viewer end, the VJ or viewers in the room will receive this notification.

### 10. onRoomClosed

- API definition: void onRoomClosed(String roomID)

- API description: When a room is terminated, viewers in the room will receive this notification. Exit the room in the callback.
- Sample code:

```
public void onRoomClosed(String roomId) {
......
mLiveRoom.exitRoom(new LiveRoom.ExitRoomCallback() {
@Override
public void onSuccess() {
Log.i(TAG, "exitRoom Success");
}

@Override
public void onError(int errCode, String e) {
Log.e(TAG, "exitRoom failed, errorCode = " + errCode + " errMessage = " + e);
}
});
......
}
```

### 11. onDebugLog

- API definition: void onDebugLog(String log)
- API description: Live room log callback. You can save the logs as a file in the callback, so as to make it easy to analyze problems.
- Sample code:

```
public void onDebugLog(String line) {
Log.i(TAG,line);
}
```

### 12. onError

- API definition: void onError(int errorCode, String errorMessage)
- API description: Live room error callback
- Sample code:

```
public void onError(final int errorCode, final String errorMessage) {
mLiveRoom.exitRoom(null);
new AlertDialog.Builder(mActivity)
.setTitle("Live room error")
```

```
.setMessage(errorMessage + "[" + errorCode + "]")
.setNegativeButton("OK", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

}
}).show();
}
```

# RTCRoom

Last updated：2018-07-23 10:57:25

## RTCRoom

| Name | Description |
| --- | --- |
| setRTCRoomListener(IRTCRoomListener listener) | Sets rtcroom callback |
| login(serverDomain, loginInfo, loginCallback) | Logs in to the RoomService backend |
| logout() | Logs out of the RoomService backend |
| getRoomList(int index, int count, GetRoomListCallback callback) | Gets the room list (optional, you can select your room list.) |
| createRoom(String roomID, String roomInfo, CreateRoomCallback cb) | Meeting initiator: Creates a room (roomID can be left blank.) |
| enterRoom(String roomID, EnterRoomCallback cb) | Meeting participant: Enters a room |
| exitRoom(ExitRoomCallback callback) | Meeting initiator or meeting participant: Exits a room |
| startLocalPreview(TXCloudVideoView videoView) | Meeting initiator or meeting participant: Enables camera preview |
| stopLocalPreview() | Disables camera preview |
| addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback) | Plays back the remote video images of the meeting participant |
| deleteRemoteView(PusherInfo pusherInfo) | Stops playing back the remote video images of the meeting participant |
| sendRoomTextMsg(String message, SendTextMessageCallback callback) | Sends a text (on-screen comment) message |
| sendRoomCustomMsg(String cmd, String message, SendCustomMessageCallback callback) | Sends a custom message (gives a "Like" or flower) |

| Name | Description |
| --- | --- |
| switchToBackground() | App switches from foreground to background |
| switchToForeground() | App switches from background to foreground |
| setBeautyFilter(style, beautyLevel, whiteningLevel, ruddyLevel) | Sets beauty filter effects |
| switchCamera() | Switches between front and rear cameras. Dynamic switching is supported during push |
| setMute(mute) | Enables Mute |
| setMirror(enable) | Sets mirroring for images (this API works on the effect at the viewer end only; the pusher end always has the mirroring effect available) |
| playBGM(String path) | Starts background music ("path" indicates the path to the music file) |
| stopBGM() | Stops background music |
| pauseBGM() | Pauses background music |
| resumeBGM() | Resumes background music |
| setMicVolume(x) | Sets microphone volume for audio mixing |
| setBGMVolume(x) | Sets background music volume for audio mixing |
| getMusicDuration(fileName) | Gets background music duration |
| setBitrateRange(minBitrate, maxBitrate) | Sets video bitrate range |
| setPauseImage(bitmap) | Sets the images to be pushed when switching to the background |

## IRTCRoomListener

| Name | Description |
| --- | --- |
| onGetPusherList(pusherList) | Notification: The list of existing pushers in the room (the number of remote video streams) |

| Name | Description |
| --- | --- |
| onPusherJoin(pusherInfo) | Notification: A new pusher joined the room (notifies you of addition of a remote video stream) |
| onPusherQuit(pusherInfo) | Notification: A pusher left the room (notifies you of subtraction of a remote video stream) |
| onRecvRoomTextMsg(roomID, userID, userName, userAvatar, message) | Chat room: Receives a text message |
| onRecvRoomCustomMsg(roomID, userID, userName, userAvatar, cmd, message) | Chat room: Receives a custom message |
| onRoomClosed(roomID) | Notification: Notifies you of the room being closed |
| onDebugLog(log) | LOG: Log callback |
| onError(errorCode, errorMessage) | ERROR: Error callback |

# Details of RTCRoom APIs

### 1. setRTCRoomListener

- API definition: void setRTCRoomListener(IRTCRoomListener listener)
- API description: Sets IRTCRoomListener callback. For the callback function, please see the IRTCRoomListener API description.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| listener | IRTCRoomListener | An rtcroom callback API |

- Sample code:

```
mRTCRoom.setRTCRoomListener(new IRTCRoomListener() {
@Override
void onPusherJoin(PusherInfo pusherInfo) {
// ...
}

@Override
```

```
void onPusherQuit(PusherInfo pusherInfo) {
// ...
}

.....
});
```

## 2. login

- API definition: void login(String serverDomain, final LoginInfo loginInfo, final LoginCallback callback)
- API description: Logs in to the RoomService backend. You can specify whether to use the Tencent Cloud RoomService or the user-deployed RoomService.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| serverDomain | String | Server address of RoomService. For more information, please see DOC. |
| loginInfo | LoginInfo | The login parameter. For more information, please see DOC. |
| callback | LoginCallback | Callback to verify whether the login is successful |

- Sample code:

```
final String DOMAIN = "https://room.qcloud.com/weapp/rtc_room ";
LoginInfo loginInfo = new LoginInfo();
loginInfo.sdkAppID = sdkAppID;
loginInfo.userID = userID;
loginInfo.userSig = userSig;
loginInfo.accType = accType;
loginInfo.userName = userName;
loginInfo.userAvatar = userAvatar;
mRTCRoom.login(DOMAIN, loginInfo, new RTCRoom.LoginCallback() {
@Override
public void onError(int errCode, String errInfo) {
//
}

@Override
public void onSuccess(String userId) {
//The userId is returned upon successful login.
```

```
}
});
```

### 3. logout

- API definition: void logout();
- API description: Logs out of the RoomService backend
- Sample code:

```
mRTCRoom.logout();
```

### 4. getRoomList

- API definition: void getRoomList(int index, int count, GetRoomListCallback callback)
- API description: Pulls the room list. The parameters index and count are used to handle paging, which means you can pull "count" rooms from the room numbered "index". This API is not required to be called, and you can continue using it if you already have your own room list service modules.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| index | int | The index of the room from which pull starts |
| count | int | Number of rooms to be returned via RoomService |
| callback | GetRoomListCallback | Callback of pulling a room list |

- Sample code:

```
//The pulling begins from the number 0, and ends when 20 rooms are pulled.
mRTCRoom.getRoomList(0, 20, new RTCRoom.GetRoomListCallback() {
@Override
public void onSuccess(ArrayList<RoomInfo> data) {
//For information on each room, please see the definition of RoomInfo.
}

@Override
public void onError(int errCode, String e) {
```

```
}
});
```

### 5. createRoom

- API definition: void createRoom(final String roomID, final String roomInfo, final CreateRoomCallback cb)
- API description: Creates a room at RoomService backend.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| roomID | String | You can specify an ID for a new room with the parameter roomID, or leave it unspecified. If you do not specify an ID for the room, RoomService will automatically create a new roomID and return it to you through CreateRoomCallback. |
| roomInfo | String | To be customized by the creator. This information is returned via getRoomList |
| cb | CreateRoomCallback | Creates room creation result callback |

- Sample code:

```
String roomInfo = mTitle;
try {
roomInfo = new JSONObject()
.put("title", mTitle)
.put("frontcover", mCoverPicUrl)
.put("location", mLocation)
.toString();
} catch (JSONException e) {
roomInfo = mTitle;
}
mRTCRoom.createRoom("", roomInfo, new RTCRoom.CreateRoomCallback() {
@Override
public void onSuccess(String roomId) {
Log.w(TAG,String.format("Room %s created successfully", roomId));
}

@Override
public void onError(int errCode, String e) {
Log.w(TAG,String.format("Error while creating the room, code=%s,error=%s", errCode, e));
```

```
}
});
```

### 6. enterRoom

- API definition: void enterRoom(String roomID, EnterRoomCallback cb)
- API description: (Meeting participant) enters a room.
- Sample code:

```
mRTCRoom.enterRoom(mRoomId, new RTCRoom.EnterRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "enter room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "enter room success ");
}
});
```

### 7. exitRoom

- API definition: void exitRoom(final ExitRoomCallback cb))
- API description: (Meeting initiator or meeting participant) exits a room.
- Sample code:

```
mRTCRoom.exitRoom(new RTCRoom.ExitRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "exit room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "exit room success ");
```

```
}
});
```

### 8. startLocalPreview

- API definition: void startLocalPreview(TXCloudVideoView view)
- API description: (A meeting initiator or meeting participant) enables camera preview. The front camera is used by default, and switchCamera is used to switch between front and rear cameras.
- Sample code:

```
TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mRTCRoom.startLocalPreview(mCaptureView);
```

### 9. stopLocalPreview

- API definition: void stopLocalPreview(boolean isNeedClearLastImg)
- API description: (Meeting initiator or meeting participant) disables camera preview.
- Sample code:

```
mRTCRoom.stopLocalPreview();
```

### 10. addRemoteView

- API definition: void addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback)
- API description: (A meeting initiator or meeting participant) plays the remote video image of a meeting participant. This API is called when onPusherJoin (notification of new meeting participant entering the room) is received.
- Sample code:

```
public void onPusherJoin(PusherInfo pusherInfo) {
.....
mRTCRoom.addRemoteView(videoView, pusherInfo, new RTCRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
}

@Override
public void onPlayError() {
}
```

```
});
.....
}
```

### 11. deleteRemoteView

- API definition: void deleteRemoteView(final PusherInfo pusherInfo)
- API description: Stops playing the video of a meeting participant. This API is called when onPusherQuit (a meeting participant leaves) is received.
- Sample code:

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mRTCRoom.deleteRemoteView(pusherInfo);
.....
}
```

### 12. sendRoomTextMsg

- API definition: void sendRoomTextMsg(@NonNull String message, final SendTextMessageCallback callback)
- API description: Sends a text message. Other members in the room will receive a notification via onRecvRoomTextMsg.
- Sample code:

```
mRTCRoom.sendRoomTextMsg("hello", new RTCRoom.SendTextMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.d(TAG, "sendRoomTextMsg error:");
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomTextMsg success:");
}
});
```

### 13. sendRoomCustomMsg

- API definition: void sendRoomCustomMsg(@NonNull String cmd, @NonNull String message, final SendCustomMessageCallback callback)

- API description: Sends a custom message. Other members in the room will receive a notification via onRecvRoomCustomMsg.
- Sample code:

```
mRTCRoom.sendRoomCustomMsg(String.valueOf(TCConstants.IMCMD_DANMU),
"hello ", new RTCRoom.SendCustomMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.w(TAG, "sendRoomDanmuMsg error: "+errInfo);
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomDanmuMsg success");
}
});
```

### 14. switchToBackground

- API definition: void switchToBackground()
- API description: Switches from foreground to background, stops collecting camera data, and pushes default pictures.

### 15. switchToForeground

- API definition: void switchToForeground()
- API description: Switches from background to foreground, and starts collecting camera data.

### 16. setBeautyFilter

- API definition: boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel)
- API description: Sets beauty filter style, dermabrasion level, whitening level, and blushing level.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| style | int | Dermabrasion style: 0: Smooth 1: Natural 2: Hazy |
| beautyLevel | int | Dermabrasion level: Value range: 0-9. 0 means disabling beauty filter. Default is 0, i.e., disabling beauty filter |
| whiteningLevel | int | Whitening level: Value range: 0-9. 0 means disabling whitening. Default is 0, i.e., disabling whitening |

| Parameter | Type | Description |
| --- | --- | --- |
| ruddyLevel | int | Blushing level: Value range: 0-9. 0 means disabling blushing. Default is 0, i.e., disabling blushing |

- Sample code:

```
mRTCRoom.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
```

### 17. switchCamera

- API definition: void switchCamera()
- API description: Switches between cameras. When the front camera is in use, calling this API enables a switch from the front camera to the rear camera, and vice versa. This API takes effect only if it is called after camera preview (startCameraPreview(TXCloudVideoView)) is enabled. The front camera is used by default when the SDK enables camera preview.

### 18. setMute

- API definition: void setMute(mute)
- API description: Enables Mute Once Mute is enabled, the SDK shifts from pushing microphone-collected sounds to pushing mute.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| mute | boolean | Whether to enable Mute |

### 19. setMirror

- API definition: void setMirror(boolean enable)
- API description: Sets horizontal mirroring at the viewer end. Note that this API only works on the viewer end, not the pusher end. The mirroring effect is always seen from the pusher end. The image is seen as mirrored from the pusher end when the front camera is in use, and non-mirrored when the rear camera is in use.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |

| Parameter | Type | Description |
| --- | --- | --- |
| enable | boolean | "true" indicates the image is seen as mirrored, and "false" indicates the image is seen as non-mirrored. |

- Sample code:

```
//The image is seen as mirrored at the viewer end
mRTCRoom.setMirror(true);
```

### 20. playBGM

- API definition: boolean playBGM(String path)
- API description: Starts background music. This API is used for audio mixing, for example, mixing background music with sounds collected from the microphone for playback. If the playback is successful, a value of "true" is returned. If the playback fails, a value of "false" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| path | String | The background music file is located in the absolute path in the phone |

### 21. stopBGM

- API definition: boolean stopBGM()
- API description: Stops background music. If the playback ends successfully, a value of "true" is returned. If the playback fails to end, a value of "false" is returned.

### 22. pauseBGM

- API definition: boolean pauseBGM()
- API description: Pauses background music. If the playback pauses successfully, a value of "true" is returned. If the playback fails to pause, a value of "false" is returned.

### 23. resumeBGM

- API definition: boolean resumeBGM()
- API description: Resumes background music. If the playback resumes successfully, a value of "true" is returned. If the playback fails to resume, a value of "false" is returned.

### 24. setMicVolume

- API definition: boolean setMicVolume(float x)
- API description: Sets microphone volume for audio mixing. If the microphone volume is set successfully, a value of "true" is returned. If the microphone volume fails to be set, a value of "false" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| x | float | Volume: Normal volume is 1. The recommended value is 0-2. If you need to turn up the volume, you can set it to a larger value. It is recommended to add a slider in the UI to allow VJs to set volume on their own |

### 25. setBGMVolume

- API definition: boolean setBGMVolume(float x)
- API description: Sets background music volume for audio mixing. If the background music volume is set successfully, a value of "true" is returned. If the background music volume fails to be set, a value of "false" is returned.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| x | float | Volume: Normal volume is 1. The recommended value is 0-2. If you need to turn up the volume, you can set it to a larger value. It is recommended to add a slider in the UI to allow VJs to set volume on their own |

### 26. getMusicDuration

- API definition: int getMusicDuration(String path)
- API description: Gets background music duration. The returned value is in seconds.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| path | String | Gets the duration of the current music if path == null and the duration of music under the path if path != null |

### 27. setBitrateRange

- API definition: void setBitrateRange(int minBitrate, int maxBitrate)
- API description: Sets video bitrate range, which is 400-800 for two persons, and 200-400 for more than two persons.

- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| minBitrate | int | The minimum video bitrate |
| maxBitrate | int | The maximum video bitrate |

### 28. setPauseImage

- API definition: void setPauseImage(Bitmap bitmap)
- API description: Sets the images to be pushed when switching from foreground to background.
- Parameter description:

| Parameter | Type | Description |
| --- | --- | --- |
| bitmap | Bitmap | Background image bitmap |

## Details of IRTCRoomListener APIs

### 1. onGetPusherList

- API definition: void onGetPusherList(List\ pusherList)

- API description: A new meeting participant will receive the current list of meeting participants when entering a room. In the callback, you can call addRemoteView to playback the video of other meeting participants.

- Sample code:

```
public void onGetPusherList(List<PusherInfo> pusherInfoList) {
for (PusherInfo pusherInfo : pusherInfoList) {
.....
mRTCRoom.addRemoteView(videoView, pusherInfo, new RTCRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
```

```
public void onPlayError() {

}
});
}
}
```

### 2. onPusherJoin

- API definition: void onPusherJoin(PusherInfo pusherInfo)
- API description: When a new meeting participant enters a room, the other meeting participants in the room will receive this notification. In the callback, you can call addRemoteView to playback the video of this new meeting participant.
- Sample code:

```
public void onPusherJoin(final PusherInfo pusherInfo) {
......
mRTCRoom.addRemoteView(videoView, pusherInfo, new RTCRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
......
}
```

### 3. onPusherQuit

- API definition: void onPusherQuit(PusherInfo pusherInfo)
- API description: The other meeting participants in the room will receive this notification when a meeting participant leaves a room. In the callback, you can call deleteRemoteView to stop the video of this meeting participant.
- Sample code:

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mRTCRoom.deleteRemoteView(pusherInfo);
.....
}
```

### 4. onRecvRoomTextMsg

- API definition: void onRecvRoomTextMsg(String roomID, String userID, String userName, String userAvatar, String message)
- API description: When a meeting participant calls sendRoomTextMsg, the other meeting participants in the room will receive this notification.
- Sample code:

```
public void onRecvRoomTextMsg(String roomid, String userid, String userName, String userAvatar, St
ring message) {
//do nothing
}
```

### 5. onRecvRoomCustomMsg

- API definition: void onRecvRoomCustomMsg(String roomID, String userID, String userName, String userAvatar, String cmd, String message)
- API description: When a meeting participant calls sendRoomCustomMsg, the other meeting participants in the room will receive this notification.

### 6. onRoomClosed

- API definition: void onRoomClosed(String roomID)
- API description: When a room is terminated, meeting participants in the room will receive this notification. Exit the room in the callback.
- Sample code:

```
public void onRoomClosed(String roomId) {
.....
mRTCRoom.exitRoom(new RTCRoom.ExitRoomCallback() {
@Override
public void onSuccess() {
Log.i(TAG, "exitRoom Success");
}
```

```
@Override
public void onError(int errCode, String e) {
Log.e(TAG, "exitRoom failed, errorCode = " + errCode + " errMessage = " + e);
}
});
.....
}
```

### 7. onDebugLog

- API definition: void onDebugLog(String log)
- API description: Live room log callback. You can save the logs as a file in the callback, so as to make it easy to analyze problems.
- Sample code:

```
public void onDebugLog(String line) {
Log.i(TAG,line);
}
```

### 8. onError

- API definition: void onError(int errorCode, String errorMessage)
- API description: Live room error callback
- Sample code:

```
public void onError(final int errorCode, final String errorMessage) {
mRTCRoom.exitRoom(null);
new AlertDialog.Builder(mActivity)
.setTitle("Live room error")
.setMessage(errorMessage + "[" + errorCode + "]")
.setNegativeButton("OK", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

}
}).show();
}
```