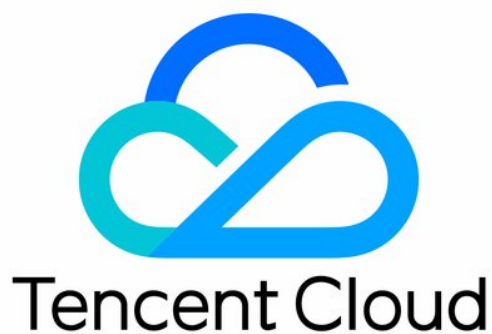


# **Tencent Real-Time Communication Basic Features Product Documentation**



## Copyright Notice

©2013-2019 Tencent Cloud. All rights reserved.

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

## Trademark Notice



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

## Service Statement

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

# Contents

## Basic Features

### Call Mode

- iOS and macOS

- Android

- Windows

- Electron

- Web

### Real-Time Screen Sharing

- iOS

- Android

- Windows

- macOS

- Web

- Flutter

### Live Streaming Mode

- iOS and macOS

- Android

- Windows

- Electron

- Web

### On-Cloud Recording

# Basic Features

## Call Mode

### iOS and macOS

Last updated : 2022-03-09 16:43:44

## Application Scenarios

TRTC supports four room entry modes. Video call ( `VideoCall` ) and audio call ( `AudioCall` ) are the call modes, and interactive video streaming ( `Live` ) and interactive audio streaming ( `VoiceChatRoom` ) are the [live streaming modes](#).

The call modes allow a maximum of 300 users in each TRTC room, and up to 50 of them can speak at the same time. The call modes are suitable for scenarios such as one-to-one video calls, video conferencing with up to 300 participants, online medical consultation, remote interviews, video customer service, and online Werewolf playing.

## How It Works

TRTC services use two types of server nodes: access servers and proxy servers.

- **Access server**

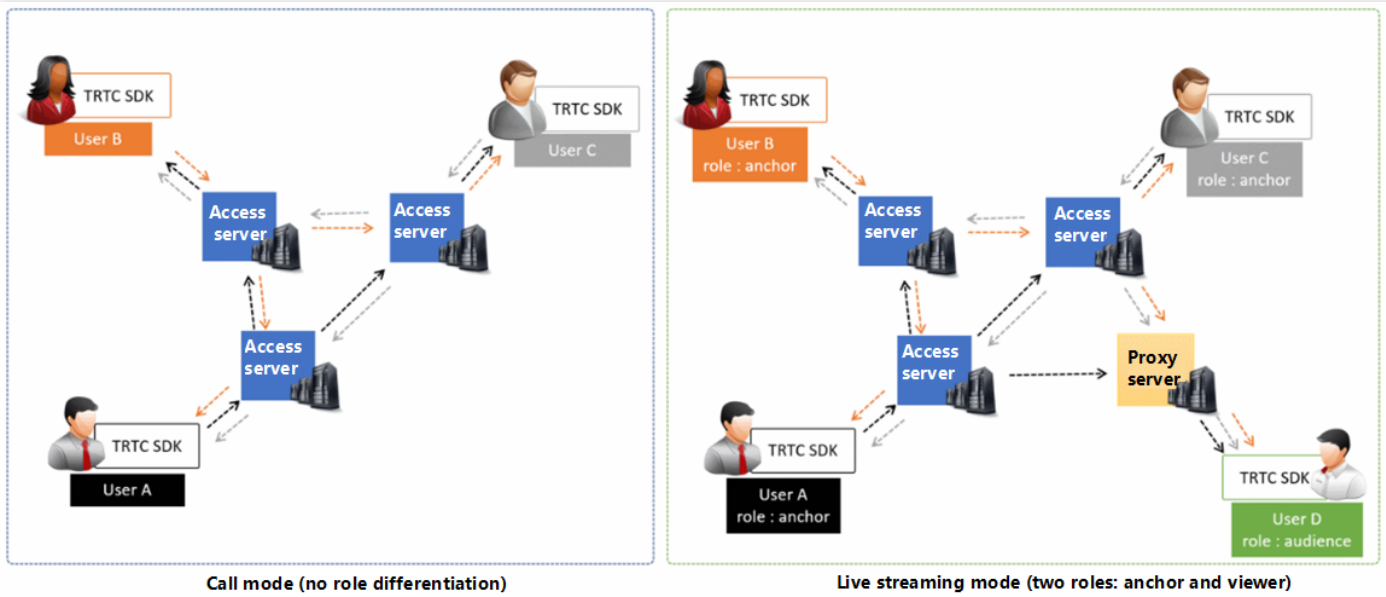
This type of nodes use high-quality lines and high-performance servers and are better suited to drive low-latency end-to-end calls, but the unit cost is relatively high.

- **Proxy server**

This type of servers use mediocre lines and average-performance servers and are better suited to power high-concurrency stream pulling and playback. The unit cost is relatively low.

In the call modes, all users in a TRTC room are assigned to access servers and are in the role of “anchor”. This means the users can speak to each other at any point during the call (up to 50 users can send data at the same time). This makes the call modes suitable for use cases such as online

conferencing, but the number of users in each room is capped at 300.



## Sample Code

You can visit [GitHub](#) to obtain the sample code used in this document.

master TRTCSDK / iOS / TRTC-API-Example-OC / Basic /

Go to file Add file ...

garyxgwang Update iOS TRTC-API-Example-OC ✓ c45668f 7 minutes ago History

AudioCall	Update iOS TRTC-API-Example-OC	7 minutes ago
Live	Update iOS TRTC-API-Example-OC	7 minutes ago
ScreenShare	Update iOS TRTC-API-Example-OC	7 minutes ago
<b>VideoCall</b>	Update iOS TRTC-API-Example-OC	7 minutes ago
VoiceChatRoom	Update iOS TRTC-API-Example-OC	7 minutes ago

Note :

If your access to GitHub is slow, download the ZIP file [here](#).

## Directions

### Step 1. Integrate the SDKs

You can integrate the **TRTC SDK** into your project in the following ways:

### Method 1: integrating through CocoaPods

1. Install **CocoaPods**. For detailed directions, please see [Getting Started](#).
2. Open the `Podfile` file in the root directory of your project and add the code below.

Note :

If you cannot find a `Podfile` file in the directory, run the `pod init` command to create one and add the code below.

```
target 'Your Project' do
  pod 'TXLiteAVSDK_TRTC'
end
```

3. Run the command below to install the **TRTC SDK**.

```
pod install
```

After successful installation, an **XCWORKSPACE** file will be generated in the root directory of your project.

4. Open the **XCWORKSPACE** file.

### Method 2: manual integration

If you do not want to install CocoaPods, or your access to CocoaPods repositories is slow, you can download the [ZIP file](#) of the SDK and integrate it into your project as instructed in [SDK Quick Integration > iOS](#).

### Step 2. Add device permission requests

Add camera and mic permission requests in the `Info.plist` file.

Key	Value
Privacy - Camera Usage Description	The reason for requesting camera permission, for example, "camera access is required to capture video"

Key	Value
Privacy - Microphone Usage Description	The reason for requesting mic permission, for example, "mic access is required to capture audio"

### Step 3. Initialize an SDK instance and configure event callbacks

1. Call `sharedInstance()` to create a `TRTCCloud` instance.

```
// Create a `TRTCCloud` instance
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;
```

2. Set the attributes of `delegate` to subscribe to event callbacks and listen for event and error notifications.

```
// Error events must be listened for and captured, and error messages should be sent to users.
- (void)onError:(TXLiteAVError)errCode errMsg:(NSString *)errMsg extInfo:(NSDictionary *)extInfo {
    if (ERR_ROOM_ENTER_FAIL == errCode) {
        [self toastTip:@"Failed to enter room"];
        [self.trtcCloud exitRoom];
    }
}
```

### Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom()` API, you need to pass in a key parameter `TRTCParams`, which includes the following required fields:

Parameter	Type	Description	Example
sdkAppId	Number	Application ID, which you can view in the <a href="#">TRTC console</a> .	1400000123
userId	String	Can contain only letters (a-z and A-Z), digits (0-9), underscores, and hyphens. We recommend you set it based on your business account system.	test_user_001
userSig	String	<code>userSig</code> is calculated based on <code>userId</code> . For the calculation method, see <a href="#">UserSig</a> .	eJyrVareCeYrSy1Ssll...

Parameter	Type	Description	Example
roomId	Number	Numeric room ID. For string-type room ID, use <code>strRoomId</code> in <code>TRTCParams</code> .	29834

Note :

In TRTC, users with the same `userId` cannot be in the same room at the same time as it will cause a conflict.

## Step 5. Create and enter a room

1. Call `enterRoom()` to enter the room specified by the `roomId` field in `TRTCParams`. If the room does not exist, the SDK will create a room whose room number is the value of `roomId`.
2. Set the `appScene` parameter according to your actual application scenario. Inappropriate `appScene` values may lead to increased lag or decreased clarity.
  - For video calls, set the parameter to `TRTCAppScene.videoCall`.
  - For audio calls, set the parameter to `TRTCAppScene.audioCall`.
3. You will receive the `onEnterRoom(result)` callback. If `result` is greater than 0, room entry succeeds, and the value of `result` indicates the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.

```
- (void)enterRoom() {
    TRTCParams *params = [TRTCParams new];
    params.sdkAppId = SDKAppID;
    params.roomId = _roomId;
    params.userId = _userId;
    params.role = TRTCRoleAnchor;
    params.userSig = [GenerateTestUserSig genTestUserSig:params.userId];
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneVideoCall];
}

- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Entered room successfully"];
    } else {
        [self toastTip:@"Failed to enter room"];
    }
}
```



**Note :**

- If room entry fails, you will also receive the `onError` callback, which contains `errCode` (error code), `errMsg` (error message), and `extraInfo` (reserved parameter).
- If you are already in a room, you must call `exitRoom` to exit the room before entering another room.
- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## Step 6. Subscribe to remote audio/video streams

The SDK supports automatic subscription and manual subscription.

### Automatic subscription (default)

In the automatic subscription mode, after room entry, the SDK will automatically pull audio streams from other users in the room. This enables instant streaming.

1. If other users in the room are sending audio data, you will receive the `onUserAudioAvailable()` notification, and the SDK will automatically play back the users' audio.
2. Call `muteRemoteAudio(userId, mute: true)` to mute a specified user ( `userId` ), or `muteAllRemoteAudio(true)` to mute all remote users. The SDK will stop pulling the audio data of the user(s).
3. If a remote user in the room is sending video data, you will receive the `onUserVideoAvailable()` notification, but since the SDK has not received instructions on how to display the video, it will not process the video data. You must call `startRemoteView(userId, view: view)` to associate the remote user's video data with `view`.
4. Call `setRemoteViewFillMode()` to specify the display mode of a remote video.
  - `Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
5. Call `stopRemoteView(userId)` to block the video data of a specified user ( `userId` ) or `stopAllRemoteView()` to block the video data of all remote users. The SDK will stop pulling the video data of the user(s).

```
// Sample code: subscribe to or unsubscribe from the video image of a remote user based on the notification received
- (void)onUserVideoAvailable:(NSString *)userId available:(BOOL)available {
```

```
UIView* remoteView = remoteViewDic[userId];  
if (available) {  
    [_trtcCloud startRemoteView:userId streamType:TRTCVideoStreamTypeSmall view:remoteView];  
} else {  
    [_trtcCloud stopRemoteView:userId streamType:TRTCVideoStreamTypeSmall];  
}  
}
```

#### Note :

If you do not call `startRemoteView()` to subscribe to the video stream immediately after receiving the `onUserVideoAvailable()` event callback, the SDK will stop pulling the remote video within 5 seconds.

## Manual subscription

You can call `setDefaultStreamRecvMode()` to switch the SDK to the manual subscription mode. In this mode, the SDK will not pull the data of other users in the room automatically. You have to start the process manually via APIs.

1. **Before you enter a room**, call the `setDefaultStreamRecvMode(false, video: false)` API to switch the SDK to the manual subscription mode.
2. If other users in the room are sending audio data, you will receive the `onUserAudioAvailable()` notification, and you need to call `muteRemoteAudio(userId, mute: false)` to manually subscribe to the users' audio. The SDK will decode and play the audio data received.
3. If a remote user in the room is sending video data, you will receive the `onUserVideoAvailable()` notification, and you need to call `startRemoteView(userId, view: view)` to manually subscribe to the user's video data. The SDK will decode and play the video data received.

## Step 7. Publish the local stream

1. Call `startLocalAudio()` to enable local mic capturing and encode and send the audio captured.
2. Call `startLocalPreview()` to enable local camera capturing and encode and send the video captured.
3. Call `setLocalViewFillMode()` to set the display mode of the local video:
  - **Fill** : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - **Fit** : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.

4. Call `setVideoEncoderParam()` to set the encoding parameters for the local video, which determine the [quality of your video](#) seen by other users in the room.

```
// Sample code: publish the local audio/video stream
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

Note :

The SDK for macOS uses the default camera and mic. You can call `setCurrentCameraDevice()` and `setCurrentMicDevice()` to switch to a different camera and mic.

## Step 8. Exit the room

Call `exitRoom()` to exit the room. The SDK disables and releases devices such as cameras and mics during room exit. Therefore, room exit is not an instant process. It completes only after the `onExitRoom()` callback is received.

```
// Please wait for the `onExitRoom` callback after calling the room exit API.
[self.trtcCloud exitRoom];

- (void)onExitRoom:(NSInteger)reason {
    NSLog(@"Exited room: reason: %ld", reason)
}
```

Note :

If your application integrates multiple audio/video SDKs, please wait after you receive the `onExitRoom` callback to start other SDKs; otherwise, the device busy error may occur.

# Android

Last updated : 2022-03-09 16:35:03

## Application Scenarios

TRTC supports four room entry modes. Video call ( `VideoCall` ) and audio call ( `AudioCall` ) are the call modes, and interactive video streaming ( `Live` ) and interactive audio streaming ( `VoiceChatRoom` ) are the [live streaming modes](#).

The call modes allow a maximum of 300 users in each TRTC room, and up to 50 of them can speak at the same time. The call modes are suitable for scenarios such as one-to-one video calls, video conferencing with up to 300 participants, online medical consultation, remote interviews, video customer service, and online Werewolf playing.

## How It Works

TRTC services use two types of server nodes: access servers and proxy servers.

- **Access server**

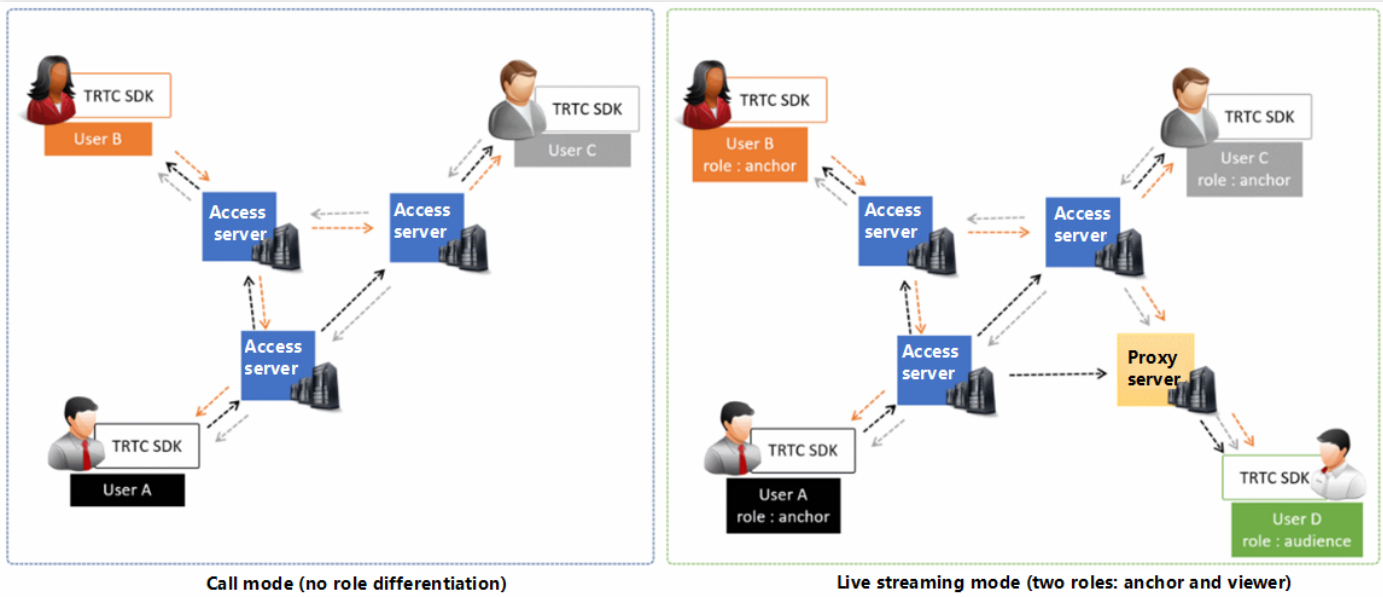
This type of nodes use high-quality lines and high-performance servers and are better suited to drive low-latency end-to-end calls, but the unit cost is relatively high.

- **Proxy server**

This type of servers use mediocre lines and average-performance servers and are better suited to power high-concurrency stream pulling and playback. The unit cost is relatively low.

In the call modes, all users in a TRTC room are assigned to access servers and are in the role of “anchor”. This means the users can speak to each other at any point during the call (up to 50 users can send data at the same time). This makes the call modes suitable for use cases such as online

conferencing, but the number of users in each room is capped at 300.



## Sample Code

You can visit [GitHub](#) to obtain the sample code used in this document.

master TRTCSDK / Android / TRTC-API-Example / Basic / Go to file Add file ...

garyxgwang Update Android TRTC-API-Example 6444d46 3 hours ago History

AudioCall	Update Android TRTC-API-Example	3 hours ago
Live	Update Android TRTC-API-Example	3 hours ago
ScreenShare	Update Android TRTC-API-Example	3 hours ago
<b>VideoCall</b>	Update Android TRTC-API-Example	3 hours ago
VoiceChatRoom	Update Android TRTC-API-Example	3 hours ago

Note :

If your access to GitHub is slow, download the ZIP file [here](#).

## Directions

### Step 1. Integrate the SDKs

You can integrate the **TRTC SDK** into your project in the following ways:

### Method 1: automatic loading (AAR)

The TRTC SDK has been released to the mavenCentral repository, and you can configure Gradle to download updates automatically.

The TRTC SDK has integrated `TRTC-API-Example`, which offers sample code for your reference. Use Android Studio to open your project and follow the steps below to modify the `app/build.gradle` file.

1. Add the TRTC SDK dependency to `dependencies`.

```
dependencies {  
    compile 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release'  
}
```

2. In `defaultConfig`, specify the CPU architecture to be used by your application.

Note :

Currently, the TRTC SDK supports armeabi, armeabi-v7a, and arm64-v8a.

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

3. Click **Sync Now** to sync the SDK.

If you have no problem connecting to mavenCentral, the SDK will be downloaded and integrated into your project automatically.

### Method 2: manual integration

You can directly download the [ZIP package](#) and integrate the SDK into your project as instructed in [Quick Integration \(Android\)](#).

### Step 2. Configure app permissions.

Add camera, mic, and network permission requests in `AndroidManifest.xml`.

```

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

```

### Step 3. Initialize an SDK instance and configure event callbacks

1. Call the `sharedInstance()` API to create a `TRTCCloud` instance.

```

// Create a `TRTCCloud` instance
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener(){
// Processing callbacks
...
});

```

2. Set the attributes of `setListener` to subscribe to event callbacks and listen for event and error notifications.

```

// Error notifications indicate that the SDK has stopped working and therefore must be listened for
@Override
public void onError(int errCode, String errMsg, Bundle extraInfo) {
Log.d(TAG, "sdk callback onError");
if (activity != null) {
Toast.makeText(activity, "onError: " + errMsg + "[" + errCode + "]", Toast.LENGTH_SHORT).show();
}
if (errCode == TXLiteAVCode.ERR_ROOM_ENTER_FAIL) {
activity.exitRoom();
}
}
}
}

```

## Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom()` API, you need to pass in a key parameter `TRTCParams`, which includes the following required fields:

Parameter	Type	Description	Example
<code>sdkAppId</code>	Number	Application ID, which you can view in the <a href="#">TRTC console</a> .	1400000123
<code>userId</code>	String	Can contain only letters (a-z and A-Z), digits (0-9), underscores, and hyphens. We recommend you set it based on your business account system.	test_user_001
<code>userSig</code>	String	<code>userSig</code> is calculated based on <code>userId</code> . For the calculation method, please see <a href="#">UserSig</a> .	eJyrVareCeYrSy1Ssll...
<code>roomId</code>	Number	Numeric room ID. For string-type room ID, use <code>strRoomId</code> in <code>TRTCParams</code> .	29834

### Note :

In TRTC, users with the same `userID` cannot be in the same room at the same time as it will cause a conflict.

## Step 5. Create and enter a room

1. Call `enterRoom()` to enter the audio/video room specified by `roomId` in the `TRTCParams` parameter. If the room does not exist, the SDK will automatically create it with the `roomId` value as the room number.
2. Set the `appScene` parameter according to your actual application scenario. Inappropriate `appScene` values may lead to increased lag or decreased clarity.
  - For video calls, please set `TRTC_APP_SCENE_VIDEOCALL`.
  - For audio calls, please set `TRTC_APP_SCENE_AUDIOCALL`.
3. You will receive the `onEnterRoom(result)` callback. If `result` is greater than 0, room entry succeeds, and the value of `result` indicates the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.



```
public void enterRoom() {
    TRTCCloudDef.TRTCCParams trtcParams = new TRTCCloudDef.TRTCCParams();
    trtcParams.sdkAppId = sdkappid;
    trtcParams.userId = userid;
    trtcParams.roomId = 908;
    trtcParams.userSig = usersig;
    mTRTCCloud.enterRoom(trtcParams, TRTC_APP_SCENE_VIDEOCALL);
}
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        toastTip("Entered room successfully; the total time used is [¥(result)] ms")
    } else {
        toastTip("Failed to enter the room; the error code is [¥(result)]")
    }
}
```

#### Note :

- If the room entry fails, the SDK will also call back the `onError` event and return the parameters `errCode` (error code), `errMsg` (error message), and `extraInfo` (reserved parameter).
- If you are already in a room, you must call `exitRoom` to exit the room before entering another room.
- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## Step 6. Subscribe to remote audio/video streams

The SDK supports automatic subscription and manual subscription.

### Automatic subscription (default)

In automatic subscription mode, after room entry, the SDK will automatically receive audio streams from other users in the room to achieve the best "instant broadcasting" effect:

1. When another user in the room is upstreaming audio data, you will receive the `onUserAudioAvailable()` event notification, and the SDK will automatically play back the audio of the remote user.
2. You can call `muteRemoteAudio(userId, true)` to mute a specified user ( `userId` ), or `muteAllRemoteAudio(true)` to mute all remote users. The SDK will stop pulling the audio data of the user(s).

- When another user in the room is upstreaming video data, you will receive the `onUserVideoAvailable()` event notification; however, since the SDK has not received instructions on how to display the video data at this time, video data will not be processed automatically. You need to associate the video data of the remote user with the display `view` by calling the `startRemoteView(userId, view)` method.
- Call `setRemoteViewFillMode()` to specify the display mode of a remote video.
  - `Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
- Call `stopRemoteView(userId)` to block the video data of a specified user ( `userId` ) or `stopAllRemoteView()` to block the video data of all remote users. The SDK will stop pulling the video data of the user(s).

```
@Override
public void onUserVideoAvailable(String userId, boolean available) {
    TXCloudVideoView remoteView = remoteViewDic[userId];
    if (available) {
        mTRTCCloud.startRemoteView(userId, remoteView);
        mTRTCCloud.setRemoteViewFillMode(userId, TRTC_VIDEO_RENDER_MODE_FIT);
    } else {
        mTRTCCloud.stopRemoteView(userId);
    }
}
```

#### Note :

If you do not call `startRemoteView()` to subscribe to the video stream immediately after receiving the `onUserVideoAvailable()` event callback, the SDK will stop pulling the remote video within 5 seconds.

## Manual subscription

You can call `setDefaultStreamRecvMode()` to switch the SDK to the manual subscription mode. In this mode, the SDK will not pull the data of other users in the room automatically. You have to start the process manually via APIs.

- Before room entry**, call the `setDefaultStreamRecvMode(false, false)` API to set the SDK to manual subscription mode.

2. If other users in the room are sending audio data, you will receive the `onUserAudioAvailable()` notification, and you need to call `muteRemoteAudio(userId, false)` to manually subscribe to the users' audio. The SDK will decode and play the audio data received.
3. If a remote user in the room is sending video data, you will receive the `onUserVideoAvailable()` notification, and you need to call `startRemoteView(userId, remoteView)` to manually subscribe to the user's video data. The SDK will decode and play the video data received.

## Step 7. Publish the local stream

1. Call `startLocalAudio()` to enable local mic capturing and encode and send the audio captured.
2. Call `startLocalPreview()` to enable local camera capturing and encode and send the video captured.
3. Call `setLocalViewFillMode()` to set the display mode of the local video:
  - `Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
4. Call `setVideoEncoderParam()` to set the encoding parameter of the local video, which determines the `image quality` of the video watched by other users in the room.

```
// Sample code: publish the local audio/video stream
mTRTCCloud.setLocalViewFillMode(TRTC_VIDEO_RENDER_MODE_FIT);
mTRTCCloud.startLocalPreview(mIsFrontCamera, mLocalView);
mTRTCCloud.startLocalAudio();
```

## Step 8. Exit the room

Call `exitRoom()` to exit the room. The SDK disables and releases devices such as cameras and mics during room exit. Therefore, room exit is not an instant process. It completes only after the `onExitRoom()` callback is received.

```
// Please wait for the `onExitRoom` callback after calling the room exit API.
mTRTCCloud.exitRoom()
@Override
public void onExitRoom(int reason) {
    Log.i(TAG, "onExitRoom: reason = " + reason);
}
```

Note :

If your application integrates multiple audio/video SDKs, please wait after you receive the `onExitRoom` callback to start other SDKs; otherwise, the device busy error may occur.

# Windows

Last updated : 2021-11-22 10:12:21

## Overview

This document describes how to use the TRTC SDK to implement the simple video call feature. It covers only the most used APIs. To learn about other APIs, please see the [API documentation](#).

## Sample Code

Platform	Sample Code
Windows (MFC)	<a href="#">TRTCMainViewController.cpp</a>
Windows (DuiLib)	<a href="#">TRTCMainViewController.cpp</a>
Windows (C#)	<a href="#">TRTCMainForm.cs</a>

## Video Call

### 1. Initialize the SDK

To use the TRTC SDK, the first step is obtaining the `ITRTCCloud*` pointer to a singleton object of `TRTCCloud` through the export API `getTRTCShareInstance`, and subscribing to the SDK's events.

- Inherit the `ITRTCCloudCallback` callback API class and rewrite the callback APIs for key events including room entry/exit by local user, room entry/exit by remote user, error event, and warning event.
- Call the `addCallback` API to subscribe to the SDK's events.

Note :

If `addCallback` is called N times, the SDK will trigger N callbacks for the same event. Therefore, you are advised to call `addCallback` only once.

- [C++](#)

- C#

```
// TRTCMainViewController.h

// Inherit the `ITRTCCloudCallback` callback API class
class TRTCMainViewController : public ITRTCCloudCallback
{
public:
    TRTCMainViewController();
    virtual ~TRTCMainViewController();

    virtual void onError(TXLiteAVError errCode, const char* errMsg, void* arg);
    virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void* arg);
    virtual void onEnterRoom(int result);
    virtual void onExitRoom(int reason);
    virtual void onRemoteUserEnterRoom(const char* userId);
    virtual void onRemoteUserLeaveRoom(const char* userId, int reason);
    virtual void onUserVideoAvailable(const char* userId, bool available);
    virtual void onUserAudioAvailable(const char* userId, bool available);

    ...
private:
    ITRTCCloud * m_pTRTCSdk = NULL;
    ...
}

// TRTCMainViewController.cpp

TRTCMainViewController::TRTCMainViewController()
{
    // Create a `TRTCCloud` instance
    m_pTRTCSdk = getTRTCShareInstance();

    // Subscribe to the SDK's events
    m_pTRTCSdk->addCallback(this);
}

TRTCMainViewController::~~TRTCMainViewController()
{
    // Unsubscribe from the SDK's events
    if(m_pTRTCSdk) {
        m_pTRTCSdk->removeCallback(this);
    }

    // Release the `TRTCCloud` instance
    if(m_pTRTCSdk != NULL) {
        destroyTRTCShareInstance();
    }
}
```

```
m_pTRTCSdk = null;
}
}

// Error notifications indicate that the SDK has stopped working and therefore must be listened for.
virtual void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
    if (errCode == ERR_ROOM_ENTER_FAIL) {
        LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF8Wide(errMsg).c_str());
        exitRoom();
    }
}
```

## 2. Assemble TRTCParams

TRTCParams is the most critical parameter in the SDK. It contains four required fields: SDKAppID , userId , userSig , and roomId .

- **SDKAppID**

Log in to the [TRTC console](#). If you don't have an application yet, create one, and you will see its SDKAppID .

- **userId**

A custom string, which you can keep in line with the naming of your account. Please note that **there cannot be users with identical userId in a room.**

- **userSig**

Calculated based on SDKAppID and userID . For details, see [UserSig](#).

- **roomId**

A custom number. Please note that **rooms under the same application cannot have identical roomId** . For string-type room ID, use strRoomId in TRTCParams .

## 3. Enter (or create) a room

Call enterRoom to enter the room specified by the roomId field in TRTCParams . If the room does not exist, the SDK will create one whose room number is the value of roomId .

The appScene parameter specifies the application scenario of the SDK. In this document, it is set to TRTCApSceneVideoCall (video call). In this scenario, the codec and network components give a higher priority to ensuring video smoothness and reducing call latency and stutter.

- After calling the room entry API, you will receive the `onEnterRoom` callback. If `result` is greater than 0, room entry succeeds, and the value represents the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.
  - If room entry fails, you will also receive the `onError` callback, which contains `errCode` (error code, whose value is `ERR_ROOM_ENTER_FAIL` ; for other error code values, please see `TXLiteAVCode.h` ), `errMsg` (error message), and `extraInfo` (reserved parameter).
  - If you are already in a room, you must call `exitRoom` to exit the room before entering another room.
- [C++](#)
  - [C#](#)

```
// TRTCMainViewController.cpp

void TRTCMainViewController::enterRoom()
{
    // For the definition of `TRTCParams`, please see the header file `TRTCCloudDef.h`.
    TRTCParams params;
    params.sdkAppId = sdkappid;
    params.userId = userid;
    params.userSig = usersig;
    params.roomId = 908; // Set it to the ID of the room you want to enter
    if(m_pTRTCSDK)
    {
        m_pTRTCSDK->enterRoom(params, TRTCApSceneVideoCall);
    }
}

...

void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
    if(errCode == ERR_ROOM_ENTER_FAIL)
    {
        LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
        // Check whether `userSig` is valid, network is normal, etc.
    }
}

...

void TRTCMainViewController::onEnterRoom(int result)
{
    LOGI(L"onEnterRoom result[%d]", result);
}
```



```
if(result >= 0)
{
    // Entered room successfully
}
else
{
    // Failed to enter room. Error code = result;
}
}
```

Note :

- Set the `appScene` parameter according to your actual application scenario. Inappropriate `appScene` values may lead to increased stutter or decreased clarity.
- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## 4. Play remote audio streams

The TRTC SDK pulls remote audio streams by default, so there is no need for extra code. If you do not want to play the audio stream of a specified user ( `userid` ), you can mute it by calling `muteRemoteAudio` .

## 5. Play remote video streams

The TRTC SDK does not pull remote video streams by default. If a user is sending video data, other users in the room can get his or her `userid` through the `onUserVideoAvailable` callback in `ITRTCCloudCallback` and call `startRemoteView` to display the user's video image.

Call `setRemoteViewFillMode` to set the video display mode to `Fill` or `Fit` . Video may be resized proportionally in both modes, but they differ in that:

- In the `Fill` mode, the image fills the entire screen. If the dimensions of the image do not match those of the screen after scaling, the excess parts are cropped.
- In the `Fit` mode, the image is displayed in whole. If the dimensions of the image do not match those of the screen after scaling, the blank area is filled with black bars.

- [C++](#)
- [C#](#)

```
// TRTCMainViewController.cpp
void TRTCMainViewController::onUserVideoAvailable(const char* userId, bool available){
    if (available) {
        // Get the handle of the rendering window
        CWnd *pRemoteVideoView = GetDlgItem(IDC_REMOTE_VIDEO_VIEW);
        HWND hwnd = pRemoteVideoView->GetSafeHwnd();

        // Set the rendering mode of the remote video
        m_pTRTCSdk->setRemoteViewFillMode(TRTCVideoFillMode_Fill);
        // Call the API below to play the remote video
        m_pTRTCSdk->startRemoteView(userId, hwnd);
    } else {
        m_pTRTCSdk->stopRemoteView(userId);
    }
}
```

## 6. Enable/Disable local audio capturing

Mic capturing is disabled by default. Call `startLocalAudio` to enable local audio capturing and send the data captured, and `stopLocalAudio` to disable audio capturing.

Note :

You can call `startLocalAudio` after `startLocalPreview` .

## 7. Enable/Disable local video capturing

Camera capturing is disabled by default. You can call `startLocalPreview` to turn the local camera on and enable preview, and `stopLocalPreview` to disable camera capturing and preview.

- Call `startLocalPreview` , specifying the window for local video rendering. **The SDK dynamically detects window size and renders the video in the window represented by `rendHwnd` .**
- Call the `setLocalViewFillMode` API to set the local video rendering mode to `Fill` or `Fit` . In both modes, video may be resized proportionally, but they differ in that:
  - In the `Fill` mode, the image fills the entire screen. If the dimensions of the image do not match those of the screen after scaling, the parts that do not fit are cropped.
  - In the `Fit` mode, the image is displayed in whole. If the dimensions of the image do not match those of the screen after scaling, the unoccupied space is painted black.

- C++
- C#

```
// TRTCMainViewController.cpp

void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
    ...

    // Get the handle of the rendering window
    CWnd *pLocalVideoView = GetDlgItem(IDC_LOCAL_VIDEO_VIEW);
    HWND hwnd = pLocalVideoView->GetSafeHwnd();

    if(m_pTRTCSdk)
    {
        // Call the APIs below to set the rendering mode and rendering window
        m_pTRTCSdk->setLocalViewFillMode(TRTCVideoFillMode_Fit);
        m_pTRTCSdk->startLocalPreview(hwnd);
    }

    ...
}
```

## 8. Block audio/video

- **Block local video data**

Call `MuteLocalVideo` to block local video data to other users in the room if you do not want them to see your video for privacy reasons.

- **Block local audio data**

Call `MuteLocalAudio` to block local audio data to other users in the room if you do not want them to hear your audio for privacy reasons.

- **Block remote video data**

Call `StopRemoteView` to block the video data of a specified user ( `userid` ).

Call `StopAllRemoteView` to block the video data of all remote users.

- **Block remote audio data**

Call `MuteRemoteAudio` to block the audio data of a specified user ( `userid` ).

Call `MuteAllRemoteAudio` to block the audio data of all remote users.

## 9. Exit the room

Call `ExitRoom` to exit the room. Regardless of whether the call has ended, the SDK will release all resources used by the call.

**Note :**

After `exitRoom` is called, the SDK will start a complex handshake process, which finishes only after the `onExitRoom` callback is received.

- C++
- C#

```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
    if(m_pTRTCSdk)
    {
        m_pTRTCSdk->exitRoom();
    }
}

....

void TRTCMainViewController::onExitRoom(int reason)
{
    // Exited room successfully. `reason` is a reserved parameter and is not used for the time being.

    ...
}
```

# Electron

Last updated : 2022-01-20 14:58:02

## Application Scenarios

TRTC supports four room entry modes. Video call ( `VideoCall` ) and audio call ( `VoiceCall` ) are the call modes, and interactive video streaming ( `Live` ) and interactive audio streaming ( `VoiceChatRoom` ) are the [live streaming modes](#).

The call modes allow a maximum of 300 users in each TRTC room, and up to 50 of them can speak at the same time. The call modes are suitable for scenarios such as one-to-one video calls, video conferencing with up to 300 participants, online medical consultation, remote interviews, video customer service, and online Werewolf playing.

## How It Works

TRTC services use two types of server nodes: access servers and proxy servers.

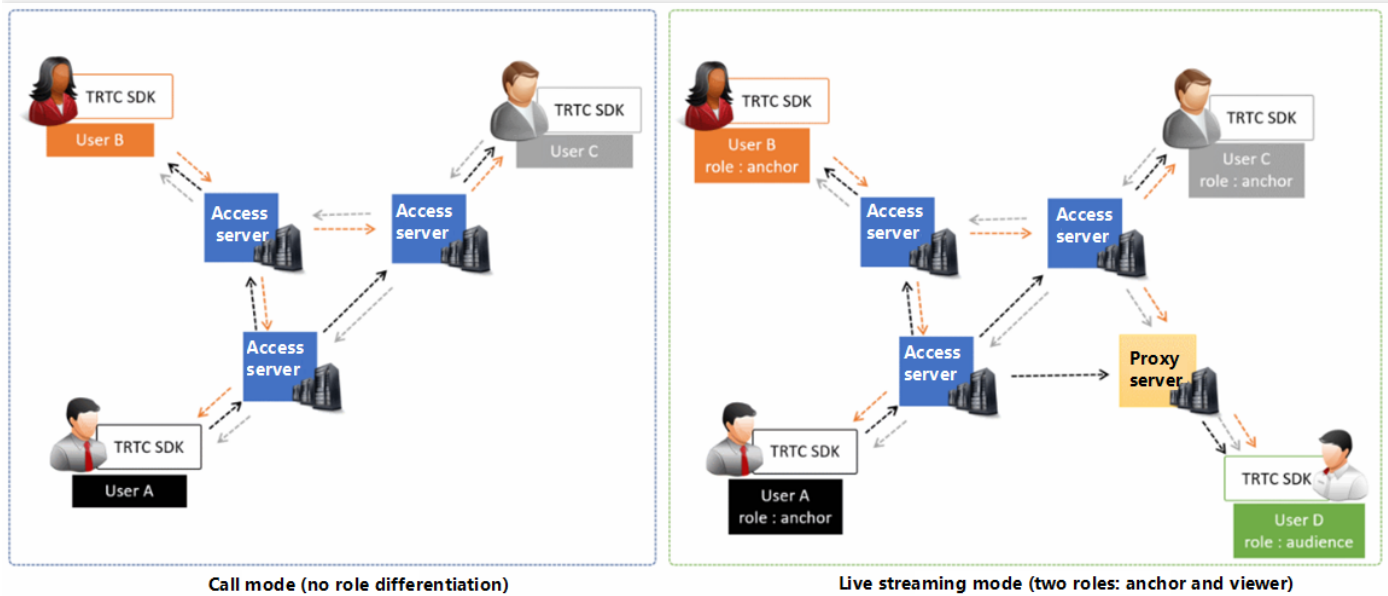
- **Access server**

This type of nodes use high-quality lines and high-performance servers and are better suited to drive low-latency end-to-end calls, but the unit cost is relatively high.

- **Proxy server**

This type of servers use mediocre lines and average-performance servers and are better suited to power high-concurrency stream pulling and playback. The unit cost is relatively low.

In the call modes, all users in a TRTC room are assigned to access servers and are in the role of “anchor”. This means the users can speak to each other at any point during the call (up to 50 users can send data at the same time). This makes the call modes suitable for use cases such as online conferencing, but the number of users in each room is capped at 300.



## Sample Code

You can obtain the sample code used in this document at [GitHub](#).

## Directions

### Step 1. Run the official SimpleDemo

We recommend that you read [Demo Quick Start > Electron](#) first and follow the instructions to run the official SimpleDemo.

- If you run SimpleDemo successfully, then you know how to install Electron in your project.
- If not, there may be a problem in the download or installation process. Try troubleshooting the problem by following the instructions in Electron's [installation document](#).

### Step 2. Integrate trtc-electron-sdk into your project

If you can [run SimpleDemo](#) successfully, then you know how to set up the Electron environment.

- You can develop your project based on the demo we provide to get started quickly.
- You can also run the following command to install trtc-electron-sdk in your project.

```
npm install trtc-electron-sdk --save
```

### Step 3. Initialize an SDK instance and configure event callbacks

1. Create a `trtc-electron-sdk` instance:

```
import TRTCCloud from 'trtc-electron-sdk';  
let trtcCloud = new TRTCCloud();
```

2. Listen for the `onError` event:

```
// Error events must be listened for and captured, and error messages should be sent to users.  
let onError = function(err) {  
  console.error(err);  
};  
trtcCloud.on('onError', onError);
```

### Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom()` API, you need to pass in a key parameter `TRTCParams`, which includes the following required fields:

Parameter	Type	Description	Example
<code>sdkAppId</code>	Number	Application ID, which can be found in <b>Application Management &gt; Application Info</b> in the <a href="#">console</a>	1400000123
<code>userId</code>	String	Can contain only letters (a-z and A-Z), digits (0-9), underscores, and hyphens. We recommend you set it based on your business account system.	test_user_001
<code>userSig</code>	String	<code>userSig</code> is calculated based on <code>userId</code> . For the calculation method, see <a href="#">UserSig</a> .	eJyrVareCeYrSy1Ssll...
<code>roomId</code>	Number	Numeric room ID. For string-type room ID, use <code>strRoomId</code> in <code>TRTCParams</code> .	29834

```
import {  
  TRTCParams,  
  TRTCRoleType  
} from "trtc-electron-sdk/liteav/trtc_define";
```

```
let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
```

Note :

- In TRTC, users with the same `userId` cannot be in the same room at the same time as it will cause a conflict.
- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## Step 5. Create and enter a room

1. Call `enterRoom()` to enter the room specified by the `roomId` field in `TRTCParams` . If the room does not exist, the SDK will create a room whose number is the value of `roomId` .
2. Set the `appScene` parameter according to your actual application scenario. Inappropriate `appScene` values may increase stutter or reduce video clarity.
  - For video calls, set it to `TRTCAppScene.TRTCAppSceneVideoCall` .
  - For audio calls, set it to `TRTCAppScene.TRTCAppSceneAudioCall` .

Note :

For more information about `TRTCAppScene` , see [TRTCAppScene](#) .

3. You will receive the `onEnterRoom(result)` callback. If `result` is greater than 0, room entry succeeds, and the value of `result` indicates the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.

```
import TRTCCloud from 'trtc-electron-sdk';
import { TRTCParams, TRTCAppScene } from "trtc-electron-sdk/liteav/trtc_define";
import TRTCCloud from 'trtc-electron-sdk';
let trtcCloud = new TRTCCloud();

let onEnterRoom = function (result) {
  if (result > 0) {
```



```
console.log(`onEnterRoom, room entry succeeded and took ${result} seconds`);
} else {
console.warn(`onEnterRoom: failed to enter room ${result}`);
}
};

// Subscribe to the room entry event
trtcCloud.on('onEnterRoom', onEnterRoom);

// Enter room. If the room does not exist, the TRTC will create one.
let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SsII...';
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneVideoCall);
```

## Step 6. Subscribe to remote audio/video streams

The SDK supports two subscription modes: automatic subscription and manual subscription. The former allows instant streaming and is suitable for calls with a small number of participants, while the latter reduces bandwidth usage and is suitable for conferences with a large number of participants.

### Automatic subscription (recommended)

After room entry, the SDK will automatically pull audio streams from other users in the room. This enables instant streaming.

1. If other users in the room are sending audio data, you will receive the `onUserAudioAvailable()` notification, and the SDK will play their audio automatically.
2. You can call `muteRemoteAudio(userId, true)` to mute a specified user ( `userId` ), or `muteAllRemoteAudio(true)` to mute all remote users. The SDK will stop pulling the audio data of the user(s).
3. If a remote user in the room is sending video data, you will receive the `onUserVideoAvailable()` notification, but since the SDK has not received instructions on how to display the video, it will not process the video data. You need to call `startRemoteView(userId, view, streamType)` to associate the remote user's video data with `view`.
4. You can call `setRemoteViewFillMode(userId, mode)` to set the display mode of a remote video.
  - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `TRTCVideoFillMode.TRTCVideoFillMode_Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.

5. You can call `stopRemoteView(userId)` to block the video of a specified `userId` or `stopAllRemoteView()` to block the video of all remote users. The SDK will stop pulling the video data of the user(s).

```
<div id="video-container"></div>

<script>
import TRTCCloud from 'trtc-electron-sdk';
const trtcCloud = new TRTCCloud();
const videoContainer = document.querySelector('#video-container');
const roomId = 29834;

/**
 * Whether camera video is enabled
 * @param {number} uid - user ID
 * @param {boolean} available - Whether video is enabled
 */

let onUserVideoAvailable = function (uid, available) {

console.log(`onUserVideoAvailable: uid: ${uid}, available: ${available}`);
if (available === 1) {
let id = `${uid}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (!view) {
view = document.createElement('div');
view.id = id;
videoContainer.appendChild(view);
}
trtcCloud.startRemoteView(uid, view);
trtcCloud.setRemoteViewFillMode(uid, TRTCVideoFillMode.TRTCVideoFillMode_Fill);
} else {
let id = `${uid}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (view) {
videoContainer.removeChild(view);
}
}
};

// Sample code: subscribe to or unsubscribe from the video image of a remote user based on the notification received
trtcCloud.on('onUserVideoAvailable', onUserVideoAvailable);

</script>
```

**Note :**

If you do not call `startRemoteView()` to subscribe to the video stream immediately after receiving the `onUserVideoAvailable()` callback, the SDK will stop pulling the remote video within 5 seconds.

**Manual subscription**

You can call `setDefaultStreamRecvMode(autoRecvAudio, autoRecvVideo)` to switch the SDK to the manual subscription mode. In this mode, the SDK will not pull the audio/video data of other users in the room automatically. You have to start the process manually via APIs.

1. **Before you enter a room**, call the `setDefaultStreamRecvMode(false, false)` API to switch the SDK to the manual subscription mode.
2. If other users in the room are sending audio data, you will receive the `onUserAudioAvailable()` notification, and you need to call `muteRemoteAudio(userId, false)` to manually subscribe to the users' audio. The SDK will decode and play the audio data received.
3. If a remote user in the room is sending video data, you will receive the `onUserVideoAvailable(userId, available)` notification, and you need to call `startRemoteView(userId, view)` to manually subscribe to the user's video data. The SDK will decode and play the video data received.

**Step 7. Publish the local stream**

1. Call `startLocalAudio()` to enable mic capturing and encode and publish the audio captured.
2. Call `startLocalPreview()` to enable camera capturing and encode and publish the video captured.
3. Call `setLocalViewFillMode()` to set the display mode of the local video:
  - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `TRTCVideoFillMode.TRTCVideoFillMode_Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
4. Call `setVideoEncoderParam()` to set the encoding parameters for the local video, which determine the **quality of your video** seen by other users in the room.

```
// Sample code: publish the local audio/video stream
trtcCloud.startLocalPreview(view);
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);
trtcCloud.startLocalAudio();
// Set local video encoding parameters
let encParam = new TRTCVideoEncParam();
```

```
encParam.videoResolution = TRTCVideoResolution.TRTCVideoResolution_640_360;
encParam.resMode = TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape;
encParam.videoFps = 25;
encParam.videoBitrate = 600;
encParam.enableAdjustRes = true;
trtcCloud.setVideoEncoderParam(encParam);
```

Note :

The SDK uses the default camera and mic. You can call [setCurrentCameraDevice\(\)](#) and [setCurrentMicDevice\(\)](#) to switch to a different camera and mic.

## Step 8. Exit the room

Call [exitRoom\(\)](#) to exit the room. The SDK disables and releases devices such as cameras and mics during room exit. Therefore, room exit is not an instant process. It completes only after the [onExitRoom\(\)](#) callback is received.

```
// Please wait for the `onExitRoom` event callback after calling the room exit API.
let onExitRoom = function (reason) {
  console.log(`onExitRoom, reason: ${reason}`);
};
trtcCloud.exitRoom();
trtcCloud.on('onExitRoom', onExitRoom);
```

Note :

If your Electron application integrates multiple audio/video SDKs, please wait after you receive the `onExitRoom` callback to start other SDKs; otherwise the device busy error may occur.

# Web

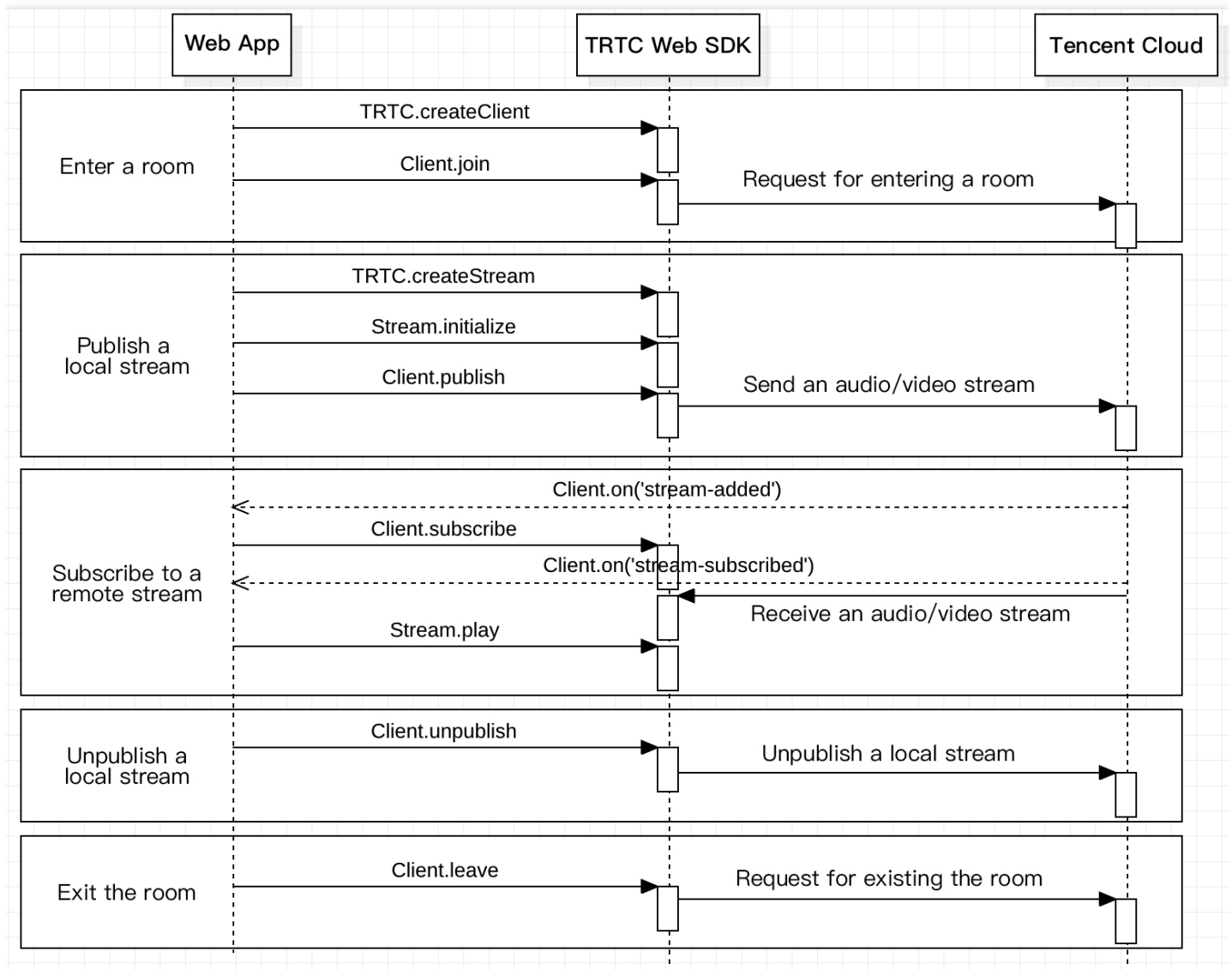
Last updated : 2022-01-20 14:58:02

This document describes the basic workflow of the TRTC SDK for Web and how to implement the real-time audio/video call feature.

You will deal with two types of objects in your use of the TRTC SDK for web.

- Client object, which represents a local client. The [Client](#) class provides APIs for room entry, local stream publishing, remote stream subscription, etc.
- Stream object, which represents an audio/video stream object. There are local stream objects ([LocalStream](#)) and remote stream objects ([RemoteStream](#)). The `Stream` class provides APIs for stream-related actions, including audio/video playback control.

Below are the APIs used in a basic audio/video call:



## Step 1. Create a client object

Create a `client` object using `TRTC.createClient()`. Set the parameters as follows:

- `mode` : TRTC mode, which should be set to `rtc`
- `sdkAppId` : the `sdkAppId` you obtain from Tencent Cloud
- `userId` : user ID
- `userSig` : user signature. For the calculation method, please see [UserSig](#)

```
const client = TRTC.createClient({
  mode: 'rtc',
```

```
sdkAppId,  
userId,  
userSig  
});
```

## Step 2. Enter a room

Call `Client.join()` to enter a TRTC room.

`roomId` : room ID

```
client  
  .join({ roomId })  
  .then(() => {  
    console.log('Entered room successfully');  
  })  
  .catch(error => {  
    console.error('Failed to enter room' + error);  
  });
```

## Step 3. Publish the local stream and subscribe to a remote stream

1. Call `TRTC.createStream()` to create a local audio/video stream.

In the example below, the local stream is captured by the camera and mic. The parameters are as follows:

- `userId` : ID of the user to whom the stream belongs
- `audio` : whether to enable audio
- `video` : whether to enable video

```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
```

2. Call `LocalStream.initialize()` to initialize the local audio/video stream.

```
localStream  
  .initialize()  
  .then(() => {  
    console.log('Local stream initialized successfully');  
  })
```

```
.catch(error => {  
  console.error('Failed to initialize local stream' + error);  
});
```

3. After the local stream is initialized, call `Client.publish()` to publish it.

```
client  
  .publish(localStream)  
  .then(() => {  
    console.log('Local stream published successfully');  
  })  
  .catch(error => {  
    console.error('Failed to publish local stream' + error);  
  });
```

4. After receiving the `Client.on('stream-added')` callback, which is used to listen for remote streams, call `Client.subscribe()` to subscribe to the stream.

Note :

- To ensure that you are notified when a remote user enters the room, please register the `Client.on('stream-added')` callback before you call `Client.join()` to enter the room.
- For other events such as the exit of a remote user, please see the [API documentation](#).

```
client.on('stream-added', event => {  
  const remoteStream = event.stream;  
  console.log('New remote stream:' + remoteStream.getId());  
  // Subscribe to the remote stream  
  client.subscribe(remoteStream);  
});  
client.on('stream-subscribed', event => {  
  const remoteStream = event.stream;  
  console.log('Subscribed to remote stream successfully:' + remoteStream.getId());  
  // Play the remote stream  
  remoteStream.play('remote_stream-' + remoteStream.getId());  
});
```

5. In the callback that indicates successful initialization of the local stream or subscription to a remote stream, call `Stream.play()` to play the stream on a webpage. The `play` method allows a



parameter that is a div element ID. The SDK will create an audio/video tag in the div element and play the stream on it.

- Play the local stream after successful initialization

```
localStream
  .initialize()
  .then(() => {
    console.log('Local stream initialized successfully');
    localStream.play('local_stream');
  })
  .catch(error => {
    console.error('Failed to initialize local stream' + error);
  });
```

- Play a remote stream after successful subscription

```
client.on('stream-subscribed', event => {
  const remoteStream = event.stream;
  console.log('Subscribed to remote stream successfully:' + remoteStream.getId());
  // Play the remote stream
  remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

## Step 4. Exit the room

Call `Client.leave()` to exit the room, and the call session ends.

```
client
  .leave()
  .then(() => {
    // Exited room. You can call `client.join` to start a new call.
  })
  .catch(error => {
    console.error('Failed to leave room' + error);
    // The error is unrecoverable. Please refresh the page.
  });
```

Note :

The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

# Real-Time Screen Sharing

## iOS

Last updated : 2022-03-20 11:04:03

TRTC supports two screen sharing schemes on iOS:

- **In-app sharing**

With in-app sharing, sharing is limited to the views of the current app. This feature is supported on iOS 13 and above. As content outside the current app cannot be shared, this feature is suitable for scenarios with high requirements on privacy protection.

- **Cross-app sharing**

Based on Apple's ReplayKit scheme, cross-app sharing allows the sharing of content across the system, but the steps required to implement this feature are more complicated than those for in-app sharing as an additional extension is needed.

## Supported Platforms

iOS	Android	macOS	Windows	Electron	Chrome
✓	✓	✓	✓	✓	✓

## In-App Sharing

You can implement in-app sharing simply by calling the [startScreenCaptureInApp](#) API of the TRTC SDK, passing in the encoding parameter `TRTCVideoEncParam`. If `TRTCVideoEncParam` is set to `nil`, the SDK will use the encoding parameters set previously.

We recommend the following encoding settings for screen sharing on iOS:

Item	Parameter	Recommended Value for Regular Scenarios	Recommended Value for Text-based Teaching
Resolution	videoResolution	1280 × 720	1920 × 1080
Frame rate	videoFps	10 fps	8 fps
Highest bitrate	videoBitrate	1600 Kbps	2000 Kbps

Item	Parameter	Recommended Value for Regular Scenarios	Recommended Value for Text-based Teaching
Resolution adaption	enableAdjustRes	NO	NO

- As screen content generally does not change drastically, it is not economical to use a high frame rate. We recommend setting it to 10 fps.
- If the screen you share contains a large amount of text, you can increase the resolution and bitrate accordingly.
- The highest bitrate ( `videoBitrate` ) refers to the highest output bitrate when a shared screen changes dramatically. If the shared content does not change a lot, the actual encoding bitrate will be lower.

## Cross-App Sharing

### Sample code

You can find the sample code for cross-app sharing in the **ScreenShare** directory of [this GitHub page](#), which contains the following files:

```

├── TRTC-API-Example-OC // TRTC API examples
│   ├── Basic // Demonstrates the cross-app screen sharing feature
│   │   ├── ScreenShare // Demonstrates the cross-app screen sharing feature
│   │   │   ├── ScreenAnchorViewController.h
│   │   │   ├── ScreenAnchorViewController.m // Screen recording view for anchor
│   │   │   ├── ScreenAnchorViewController.xib
│   │   │   ├── ScreenAudienceViewController.h
│   │   │   ├── ScreenAudienceViewController.m // Screen recording watching view for audience
│   │   │   ├── ScreenAudienceViewController.xib
│   │   │   ├── ScreenEntranceViewController.h
│   │   │   ├── ScreenEntranceViewController.swift // Screen sharing starting view
│   │   │   ├── ScreenEntranceViewController.xib
│   │   │   ├── TRTCBroadcastExtensionLauncher.h
│   │   │   ├── TRTCBroadcastExtensionLauncher.m // Supplementary code for starting screen recording
│   │   │   ├── TXReplayKit_Screen // Code for the screen recording process Broadcast Upload Extension. For details, see step 2.
│   │   │   │   ├── Info.plist
│   │   │   │   ├── SampleHandler.h
│   │   │   │   └── SampleHandler.m // Code for receiving screen recording data from the system

```

You can run the demo as instructed in [README](#).

## Directions

To enable cross-app screen sharing on iOS, you need to add the screen recording process Broadcast Upload Extension, which works with the host app to push streams. A Broadcast Upload Extension is created by the system when screen sharing is needed and is responsible for receiving the screen images captured by the system. For this, you need to do the following:

1. Create an App Group and configure it in Xcode (optional) to enable communication between the Broadcast Upload Extension and host app.
2. Create a target of Broadcast Upload Extension in your project and integrate into it `TXLiteAVSDK_ReplayKitExt.framework` from the SDK package, which is tailored for the extension module.
3. Make the host app wait to receive screen recording data from the Broadcast Upload Extension.
4. Use a helper class ( `RPSystemBroadcastPickerView` ) already implemented in the demo to make it possible to start screen sharing by tapping a button (optional), as in VooV Meeting for iOS.

### Note :

If you skip step 1, that is, if you do not configure an App Group (by passing in `nil` to the API), you can still enable screen sharing, but its stability will be compromised. Therefore, to ensure the stability of screen sharing, we suggest that you configure an App Group as described in this document.

## Step 1. Create an App Group

Log in to <https://developer.apple.com/> and do the following. **You need to download the provisioning profile again afterwards.**

1. Click **Certificates, IDs & Profiles**.
2. Click "+" next to **Identifiers**.
3. Select **App Groups** and click **Continue**.
4. In the form that pops up, fill in the **Description** and **Identifier** boxes. For **Identifier**, type the `AppGroup` value passed in to the API. After this, click **Continue**.

The image consists of three screenshots from the Apple Developer portal, illustrating the steps to register an App Group.   
1. The first screenshot shows the 'Certificates, Identifiers & Profiles' sidebar on the left. The 'Identifiers' link is highlighted with a red box and a red circle containing the number 1.   
2. The second screenshot shows the 'Identifiers' page. The 'Identifiers' tab is selected, and a red box with a red circle containing the number 2 highlights the '+ Add' button.   
3. The third screenshot shows the 'Register a New Identifier' page. The 'App Groups' option is selected with a radio button, and a red box with a red circle containing the number 3 highlights this option.   
4. The fourth screenshot shows the 'Register an App Group' form. A red box with a red circle containing the number 4 highlights the 'Register an App Group' button. The form includes fields for 'Description' and 'Identifier', with a 'Back' button and a 'Continue' button.

**Certificates, Identifiers & Profiles**

**Identifiers**

**Register a New Identifier**

**Certificates, Identifiers & Profiles**

**Register an App Group**

5. Select **Identifiers** on the top left sidebar, and click your App ID (you need to configure App ID for the host app and extension in the same way).
6. Select **App Groups** and click **Edit**.
7. In the form that pops up, select the App Group you created, click **Continue** to return to the edit page, and click **Save** to save the settings.

### Certificates, Identifiers & Profiles

Certificates

**Identifiers** +

App IDs

Identifiers

NAME	IDENTIFIER
liteavdemo	com.tencent.liteavdemo
liteavdemoReplayKitUpload	com.tencent.liteavdemo.ReplayKitUpload

### Certificates, Identifiers & Profiles

< All Identifiers

**Edit your App ID Configuration**

Platform

IOS, macOS, tvOS, watchOS

Description

liteavdemo

App ID Prefix

5GHU44CJHG (Team ID)

Bundle ID

com.tencent.liteavdemo (explicit)

Capabilities

ENABLED	NAME
<input type="checkbox"/>	Access WiFi Information
<input checked="" type="checkbox"/>	App Groups
<input type="checkbox"/>	Apple Pay Payment Processing

## App Group Assignment

Select the App Groups you wish to assign to the bundle.

☒ Select All

☒ RPLiveStreamShare

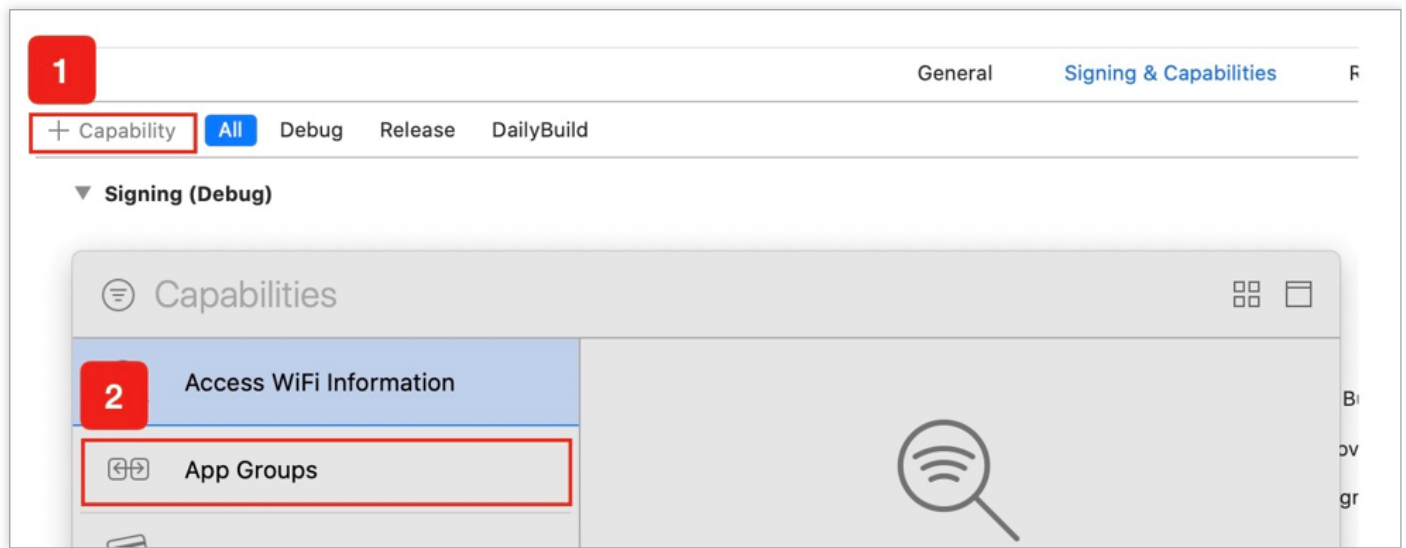
1 of 1 item(s) selected

8. Download the provisioning profile again and import it to Xcode.

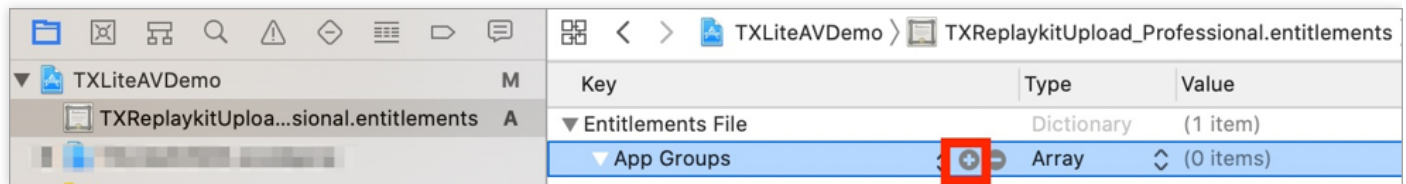
## Step 2. Create a Broadcast Upload Extension

1. In the Xcode menu, click **File > New > Target...**, and select **Broadcast Upload Extension**.
2. In the dialog box that pops up, enter the information required. You **don't need to** check **Include UI Extension**. Click **Finish** to complete the creation.
3. Drag `TXLiteAVSDK_ReplayKitExt.framework` in the SDK package into the project and select the target created.

4. Click **+ Capability**, and double-click **App Groups**, as shown below:



A file named `target name.entitlements` will appear in the file list as shown below. Select it, click "+", and enter the App Group created earlier.



5. Select the target of the host app **and configure it in the same way as described above.**
6. In the new target, Xcode will automatically create a file named `SampleHandler.h`. Replace the file content with the following code. You need to **change APPGROUP in the code to the App Group Identifier created earlier.**

```
#import "SampleHandler.h"
#import TXLiteAVSDK_ReplayKitExt;

#define APPGROUP @"group.com.tencent.liteav.RPLiveStreamShare"

@interface SampleHandler() <txreplaykitextdelegate>
@end

@implementation SampleHandler
// Note: replace `APPGROUP` with the App Group Identifier created earlier.
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject> *)setupInfo {
    [[TXReplayKitExt sharedInstance] setupWithAppGroup:APPGROUP delegate:self];
}

- (void)broadcastPaused {
```



```
// User has requested to pause the broadcast. Samples will stop being delivered.
}

- (void)broadcastResumed {
// User has requested to resume the broadcast. Samples delivery will resume.
}

- (void)broadcastFinished {
[[TXReplayKitExt sharedInstance] finishBroadcast];
// User has requested to finish the broadcast.
}

#pragma mark - TXReplayKitExtDelegate
- (void)broadcastFinished:(TXReplayKitExt *)broadcast reason:(TXReplayKitExtReason)reason
{
    NSString *tip = @"";
    switch (reason) {
        case TXReplayKitExtReasonRequestedByMain:
            tip = @"Screen sharing ended";
            break;
        case TXReplayKitExtReasonDisconnected:
            tip = @"Application disconnected";
            break;
        case TXReplayKitExtReasonVersionMismatch:
            tip = @"Integration error (SDK version mismatch)";
            break;
    }

    NSError *error = [NSError errorWithDomain:NSStringFromClass(self.class)
                                code:0
                                userInfo:@{
                                    NSLocalizedFailureReasonErrorKey:tip
                                }];
    [self finishBroadcastWithError:error];
}

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampl
eBufferType {
    switch (sampleBufferType) {
        case RPSampleBufferTypeVideo:
            [[TXReplayKitExt sharedInstance] sendVideoSampleBuffer:sampleBuffer];
            break;
        case RPSampleBufferTypeAudioApp:
            // Handle audio sample buffer for app audio
            break;
        case RPSampleBufferTypeAudioMic:
            // Handle audio sample buffer for mic audio
            break;
    }
}
```

```
default:
break;
}
}
@end
```

### Step 3. Make the host app wait to receive data

Before screen sharing starts, the host app must be on standby to receive screen recording data from the Broadcast Upload Extension. To achieve this, follow these steps:

1. Make sure that camera capturing has been disabled in `TRTCCloud` ; if not, call [stopLocalPreview](#) to disable it.
2. Call the [startScreenCaptureByReplaykit:appGroup:](#) API, passing in the `AppGroup` set in [step 1](#) to put the SDK on standby.
3. The SDK will then wait for a user to trigger screen sharing. If a "triggering button" is not added as described in [step 4](#), users need to press and hold the screen recording button in the iOS Control Center to start screen sharing.
4. You can call [stopScreenCapture](#) to stop screen sharing at any time.

```
// Start screen sharing. You need to replace `APPGROUP` with the App Group Identifier created earlier.
- (void)startScreenCapture {
    TRTCVideoEncParam *videoEncConfig = [[TRTCVideoEncParam alloc] init];
    videoEncConfig.videoResolution = TRTCVideoResolution_1280_720;
    videoEncConfig.videoFps = 10;
    videoEncConfig.videoBitrate = 2000;
    // You need to replace `APPGROUP` with the App Group Identifier created earlier.
    [[TRTCCloud sharedInstance] startScreenCaptureByReplaykit:videoEncConfig
    appGroup:APPGROUP];
}

// Stop screen sharing
- (void)stopScreenCapture {
    [[TRTCCloud sharedInstance] stopScreenCapture];
}

// Event notification for the start of screen sharing, which can be received through `TRTCCloudDelegate`
```

```
- (void)onScreenCaptureStarted  
[self showTip:@"Screen sharing started"];  
}
```

#### Step 4. Add a screen sharing triggering button (optional)

In [step 3](#), users need to start screen sharing manually by pressing and holding the screen recording button in the Control Center. To make it possible to start screen sharing by tapping a button in your app as in VooV Meeting, follow these steps:

1. Find the `TRTCBroadcastExtensionLauncher` class in the [demo](#) and add it to your project.
2. Add a button to your UI and call the `launch` function of `TRTCBroadcastExtensionLauncher` in the response function of the button to trigger screen sharing.

```
// Customize a response for button tapping  
- (IBAction)onScreenButtonTapped:(id)sender {  
    [TRTCBroadcastExtensionLauncher launch];  
}
```

#### Note :

- Apple added `RPSystemBroadcastPickerView` to iOS 12.0, which can show a picker view in apps for users to select whether to start screen sharing. Currently, `RPSystemBroadcastPickerView` does not support custom UI, and Apple does not provide an official triggering method.
- `TRTCBroadcastExtensionLauncher` works by going through the subviews of `RPSystemBroadcastPickerView`, finding the UI button, and triggering its tapping event.
- **Please note that this scheme is not recommended by Apple and may become invalid in its next update. We have therefore made [step 4](#) optional. You need to bear the risks of using the scheme yourself.**

## Watching Shared Screen

### • Watch screens shared by macOS/Windows users

When a macOS/Windows user in a room starts screen sharing, the screen will be shared through a substream, and other users in the room will be notified through `onUserSubStreamAvailable` in `TRTCCloudDelegate`.

Users who want to watch the shared screen can start rendering the substream image of the remote user by calling the `startRemoteSubStreamView` API.

- **Watch screens shared by Android/iOS users**

When an Android/iOS user starts screen sharing, the screen will be shared through the primary stream, and other users in the room will be notified through [onUserVideoAvailable](#) in

`TRTCCloudDelegate` .

Users who want to watch the shared screen can start rendering the primary stream of the remote user by calling the [startRemoteView](#) API.

# Android

Last updated : 2022-03-09 15:48:36

TRTC supports screen sharing on Android. This means a user can share the screen content of the local system with other users in the same room through the TRTC SDK. Pay attention to the following points regarding this feature:

- Unlike the desktop edition, for Android, SDK versions earlier than v8.6 do not support substream screen sharing. Therefore, video capturing by the camera must be stopped first before screen sharing can start. Substream screen sharing is supported on v8.6 and later versions, so there is no need to stop video capturing by the camera.
- Screen sharing consumes CPU. On Android, a background app consuming CPU continuously is very likely to be killed by the system. The solution to this problem is creating a floating window after screen sharing starts. As Android does not kill apps with foreground views, your app can share the screen continuously without being killed by the system.

## Supported Platforms

iOS	Android	macOS	Windows	Electron	Chrome
✓	✓	✓	✓	✓	✓

## Starting Screen Sharing

To start screen sharing on Android, simply call the [startScreenCapture\(\)](#) API in `TRTCCloud`. However, to ensure the stability and video quality of screen sharing, you need to do the following:

### Adding an activity

Copy the activity below and paste it in the manifest file. You can skip this if the activity is already included in your project code.

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

### Setting video encoding parameters

By setting the first parameter `encParams` in `startScreenCapture()`, you can specify the encoding quality of screen sharing. If `encParams` is set to `null`, the SDK will use the encoding parameters set previously. We recommend the following settings:

Item	Parameter	Recommended Value for Regular Scenarios	Recommended Value for Text-based Teaching
Resolution	<code>videoResolution</code>	1280 × 720	1920 × 1080
Frame rate	<code>videoFps</code>	10 fps	8 fps
Highest bitrate	<code>videoBitrate</code>	1600 Kbps	2000 Kbps
Resolution adaption	<code>enableAdjustRes</code>	NO	NO

- As screen content generally does not change drastically, it is not economical to use a high frame rate. We recommend setting it to 10 fps.
- If the screen you share contains a large amount of text, you can increase the resolution and bitrate accordingly.
- The highest bitrate ( `videoBitrate` ) refers to the highest output bitrate when a shared screen changes dramatically. If the shared content does not change a lot, the actual encoding bitrate will be lower.

## Displaying a floating window

Since Android 7.0, apps running in the background tend to be killed by the system if they consume CPU. To prevent your app from being killed when it is sharing the screen in the background, you need to create a floating window when screen sharing starts, which also serves the purpose of reminding the user to avoid displaying personal information as his or her screen is being shared.

### • Method 1: displaying a common floating window

The code in `FloatingView.java` offers an example of how to create a mini floating window similar to the one in VooV Meeting:

```
public void showView(View view, int width, int height) {
    mWindowManager = (WindowManager) mContext.getSystemService(Context.WINDOW_SERVICE);
    // `TYPE_TOAST` applies only to Android 4.4 and above. On earlier versions, use `TYPE_SYSTEM_ALERT` (the permission needs to be declared in `manifest`).
    // Android 7.1 and above set restrictions on `TYPE_TOAST`.
    int type = WindowManager.LayoutParams.TYPE_TOAST;
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.N) {
        type = WindowManager.LayoutParams.TYPE_PHONE;
    }
}
```

```
}
mLayoutParams = new WindowManager.LayoutParams(type);
mLayoutParams.flags = WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE;
mLayoutParams.flags |= WindowManager.LayoutParams.FLAG_WATCH_OUTSIDE_TOUCH;
mLayoutParams.width = width;
mLayoutParams.height = height;
mLayoutParams.format = PixelFormat.TRANSLUCENT;
mWindowManager.addView(view, mLayoutParams);
}
```

- **Method 2: displaying a camera preview window**

Unlike the desktop edition, for Android, SDK versions earlier than v8.6 do not support substream screen sharing (supported on v8.6 and later versions), so during screen sharing, the primary stream cannot be used to send camera video.

What if a user wants to share the screen and send camera video at the same time?

Just display a floating window of the camera preview on the screen, and the window will be captured during screen sharing and shared together with the screen.

## Watching Shared Screen

- **Watch screens shared by macOS/Windows users**

When a macOS/Windows user in a room starts screen sharing, the screen will be shared through a substream, and other users in the room will be notified through [onUserSubStreamAvailable](#) in

`TRTCCloudDelegate` .

Users who want to watch the shared screen can start rendering the substream image of the remote user by calling the [startRemoteSubStreamView](#) API.

- **Watch screens shared by Android/iOS users**

When an Android/iOS user starts screen sharing, the screen will be shared through the primary stream, and other users in the room will be notified through [onUserVideoAvailable](#) in

`TRTCCloudListener` .

Users who want to watch the shared screen can start rendering the primary stream of the remote user by calling the [startRemoteView](#) API.

```
//Sample code: watch the shared screen
@Override
public void onUserSubStreamAvailable(String userId, boolean available) {
    startRemoteSubStreamView(userId);
}
```

## FAQs

### **Can there be multiple channels of screen sharing streams in a room at the same time?**

Currently, each TRTC room can have only one channel of screen sharing stream.



# Windows

Last updated : 2021-11-08 11:39:29

TRTC supports screen sharing via the primary stream and substream on Windows:

- **Substream sharing**

In TRTC, you can share the screen via a dedicated stream, which is called the **substream**. In substream sharing, an anchor publishes camera video and screen sharing images at the same time. This is the scheme used by VooV Meeting. You can enable substream sharing by setting the `TRTCVideoStreamType` parameter to `TRTCVideoStreamTypeSub` when calling the `startScreenCapture` API. To play substream video, call `startRemoteSubStreamView`.

- **Primary stream sharing**

In TRTC, the channel via which camera images are published is the primary stream (**bigstream**). In primary stream sharing, an anchor publishes screen sharing images via the primary stream. As there is only one stream, an anchor cannot publish both camera video and screen sharing images. You can enable this mode by setting the `TRTCVideoStreamType` parameter to `TRTCVideoStreamTypeBig` when calling the `startScreenCapture` API.

## Supported Platforms

iOS	Android	macOS	Windows	Electron	Chrome
✓	✓	✓	✓	✓	✓

## APIs

Description	C++	C#	Electron
Selects a sharing source	<a href="#">selectScreenCaptureTarget</a>	<a href="#">selectScreenCaptureTarget</a>	<a href="#">selectScreenCaptureTa</a>
Starts screen sharing	<a href="#">startScreenCapture</a>	<a href="#">startScreenCapture</a>	<a href="#">startScreenCapture</a>

Description	C++	C#	Electron
Pauses screen sharing	<a href="#">pauseScreenCapture</a>	<a href="#">pauseScreenCapture</a>	<a href="#">pauseScreenCapture</a>
Resumes screen sharing	<a href="#">resumeScreenCapture</a>	<a href="#">resumeScreenCapture</a>	<a href="#">resumeScreenCapture</a>
Ends screen sharing	<a href="#">stopScreenCapture</a>	<a href="#">stopScreenCapture</a>	<a href="#">stopScreenCapture</a>

## Getting Sharable Sources

You can call `getScreenCaptureSources` to get a list of sharable sources, which is returned via the response parameter `sourceInfoList`.

### Note :

On Windows, the desktop also counts as a window. When two monitors are used, each monitor corresponds to a desktop window. The list returned via `getScreenCaptureSources` includes desktop windows.

Each `sourceInfo` object in `sourceInfoList` represents a sharable source, which is described by the following parameters:

Parameter	Type	Description
<code>type</code>	<code>TRTCScreenCaptureSourceType</code>	Capturing source type, which may be window or screen
<code>sourceId</code>	<code>HWND</code>	Capturing source ID. <ul style="list-style-type: none"><li>If a window is captured, the value of this parameter is the window handle.</li><li>If a screen is captured, the value of this parameter is the screen ID.</li></ul>

Parameter	Type	Description
sourceName	String	Window name. If a screen is captured, the value of this parameter is <code>Screen0</code> , <code>Screen1</code> , and so on.
thumbWidth	Int32	Window thumbnail width
thumbHeight	Int32	Window thumbnail height
thumbBGRA	Buffer	Window thumbnail binary buffer
iconWidth	Int32	Window icon width
iconHeight	Int32	Window icon height
iconBGRA	Buffer	Window icon binary buffer

Based on the information, you can display a list of sharable sources on the UI for users to choose from.

## Selecting Sharing Source

The TRTC SDK supports three screen sharing modes, which you can specify using

`selectScreenCaptureTarget` .

- **Share an entire screen:**

You can share an entire screen by selecting from `sourceInfoList` a source whose `type` is `TRTCScreenCaptureSourceTypeScreen` and setting `captureRect` to `{0, 0, 0, 0}`. This mode is supported when you split the screen onto multiple monitors.

- **Share a portion of a screen:**

You can share a specific portion of a screen by selecting from `sourceInfoList` a source whose `type` is `TRTCScreenCaptureSourceTypeScreen` and setting `captureRect` to a non-null value, such as `{100, 100, 300, 300}`.

- **Share a window:**

You can share a window by selecting from `sourceInfoList` a source whose `type` is `TRTCScreenCaptureSourceTypeWindow` and setting `captureRect` to `{0, 0, 0, 0}`.

Note :

Two additional parameters:

- `captureMouse` : specifies whether to capture the cursor.
- `highlightWindow` : specifies whether to highlight the window being shared and remind users to move the window when it is covered. The relevant UI design is implemented within the SDK.

## Starting Screen Sharing

- After selecting a sharing source, you can call the `startScreenCapture` API to start screen sharing.
- During screen sharing, you can call `selectScreenCaptureTarget` to change the sharing source.
- The difference between `pauseScreenCapture` and `stopScreenCapture` is that the former pauses screen capturing and displays the image at the moment of pausing. Remote users see the last frame of video before pausing until screen capturing is resumed.

```
/**
 * ¥brief 7.5 **Screen Sharing** Start screen sharing
 * ¥param: rendHwnd - HWND of the preview window
 */
void startScreenCapture(HWND rendHwnd);
/**
 * ¥brief 7.6 **Screen Sharing** Pause screen sharing
 */
void pauseScreenCapture();
/**
 * ¥brief 7.7 **Screen Sharing** Resume screen sharing
 */
void resumeScreenCapture();
/**
 * ¥brief 7.8 **Screen Sharing** Stop screen sharing
 */
void stopScreenCapture();
```

## Setting Video Quality

You can use the `setSubStreamEncoderParam` API to set the video quality of screen sharing, including resolution, bitrate, and frame rate. We recommend the following settings:

Clarity	Resolution	Frame Rate	Bitrate
FHD	1920 × 1080	10	800 Kbps
HD	1280 × 720	10	600 Kbps
SD	960 × 720	10	400 Kbps

## Watching Shared Screen

- **Watch screens shared by macOS/Windows users**

When a macOS/Windows user in a room starts screen sharing, the screen will be shared through a substream, and other users in the room will be notified through [onUserSubStreamAvailable](#) in

`TRTCCloudDelegate` .

Users who want to watch the shared screen can start rendering the substream image of the remote user by calling the [startRemoteSubStreamView](#) API.

- **Watch screens shared by Android/iOS users**

When an Android/iOS user starts screen sharing, the screen will be shared through the primary stream, and other users in the room will be notified through [onUserVideoAvailable](#) in

`TRTCCloudDelegate` .

Users who want to watch the shared screen can start rendering the primary stream of the remote user by calling the [startRemoteView](#) API.

*//Sample code: watch the shared screen*

```
void CTRTCCloudSDK::onUserSubStreamAvailable(const char * userId, bool available)
{
    LINFO(L"onUserSubStreamAvailable userId[%s] available[%d]\n", UTF8Wide(userId).c_str(), available);
    if (available) {
        startRemoteSubStreamView(userId, hWnd);
    } else {
        stopRemoteSubStreamView(userId);
    }
}
```

## FAQs

**Can there be multiple channels of screen sharing streams in the same room at the same time?**

Currently, a TRTC room can have only one screen sharing stream at a time.

**When a specified window ( `SourceTypeWindow` ) is shared, if the window size changes, will the resolution of the video stream change accordingly?**

By default, the SDK automatically adjusts encoding parameters according to the size of the shared window.

If you want a fixed resolution, call the `setSubStreamEncoderParam` API to set encoding parameters for screen sharing or specify the parameters when calling the `startScreenCapture` API.

# macOS

Last updated : 2021-11-08 11:39:29

On macOS, TRTC supports screen sharing via the primary stream and substream:

- **Substream sharing**

In TRTC, you can share the screen via a dedicated stream, which is called the **substream**. In substream sharing, an anchor publishes camera video and screen sharing images at the same time. This is the scheme used by VooV Meeting. You can enable substream sharing by setting the `TRTCVideoStreamType` parameter to `TRTCVideoStreamTypeSub` when calling the `startScreenCapture` API. To play substream video, call `startRemoteSubStreamView`.

- **Primary stream sharing**

In TRTC, the channel via which camera images are published is the primary stream (**bigstream**). In primary stream sharing, an anchor publishes screen sharing images via the primary stream. As there is only one stream, an anchor cannot publish both camera video and screen sharing images. You can enable this mode by setting the `TRTCVideoStreamType` parameter to `TRTCVideoStreamTypeBig` when calling the `startScreenCapture` API.

## Supported Platforms

iOS	Android	macOS	Windows	Electron	Chrome
✓	✓	✓	✓	✓	✓

## Getting Sharable Sources

You can call [getScreenCaptureSourcesWithThumbnailSize](#) to enumerate sharable sources. Each sharable source is a `TRTCScreenCaptureSourceInfo` object.

The desktop of macOS is also a sharable source. The type of sharable windows on macOS is `TRTCScreenCaptureSourceTypeWindow`, while that of the desktop is `TRTCScreenCaptureSourceTypeScreen`.

You can find the following information, including `type`, for each `TRTCScreenCaptureSourceInfo` object:

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

Parameter	Type	Description
type	TRTCScreenCaptureSourceType	Capturing source type, which may be window or screen
sourceId	NSString	Capturing source ID. If a window is captured, the value of this parameter is the window handle. If a screen is captured, the value of this parameter is the screen ID.
sourceName	NSString	Window name. If a screen is captured, the value of this parameter is <code>Screen0</code> , <code>Screen1</code> , and so on.
extInfo	NSDictionary	Extra information
Thumbnail	UIImage	Window thumbnail
Icon	UIImage	Window icon

Based on the information, you can display a list of sharable sources on the UI for users to choose from.

## Selecting Sharing Source

The TRTC SDK supports three sharing modes, which can be specified via [selectScreenCaptureTarget](#).

- **Share an entire screen:**

You can share an entire screen by selecting a source whose `type` is `TRTCScreenCaptureSourceTypeScreen` and setting `rect` to `{0, 0, 0, 0}`. This mode is supported when you split the screen onto multiple monitors.

- **Share a portion of a screen:**

You can share a specific portion of a screen by selecting a source whose `type` is `TRTCScreenCaptureSourceTypeScreen` and setting `rect` to a non-null value, such as `{100, 100, 300, 300}`.

- **Share a window:**

You can share a window by selecting a source whose `type` is `TRTCScreenCaptureSourceTypeWindow` and setting `rect` to `{0, 0, 0, 0}`.



Note :

Two additional parameters:

- `capturesCursor` : specifies whether to capture the cursor.
- `highlight` : specifies whether to highlight the window being shared and remind the user to move the window when it is covered. The relevant UI design is implemented within the SDK.

## Starting Screen Sharing

- After selecting a sharing source, you can call [startScreenCapture](#) to start screen sharing.
- The API [pauseScreenCapture](#) differs from [stopScreenCapture](#) in that it stops screen capturing and displays the image captured at the moment of pausing. As a result, remote users will see a still image until [resumeScreenCapture](#) is called.

```
/**
 * 7.6 Screen Sharing Start screen sharing
 * @param view Parent control of the rendering control
 */
- (void)startScreenCapture:(NSView *)view;
/**
 * 7.7 Screen Sharing Stop screen sharing
 * @return `0`: successful; negative number: failed
 */
- (int)stopScreenCapture;
/**
 * 7.8 Screen Sharing Pause screen sharing
 * @return `0`: successful; negative number: failed
 */
- (int)pauseScreenCapture;
/**
 * 7.9 Screen Sharing Resume screen sharing
 *
 * @return `0`: successful; negative number: failed
 */
- (int)resumeScreenCapture;
```

## Setting Video Quality

You can use [setSubStreamEncoderParam](#) to set the video quality of screen sharing, including resolution, bitrate, and frame rate. We recommend the following settings:

Clarity	Resolution	Frame Rate	Bitrate
FHD	1920 × 1080	10	800 Kbps
HD	1280 × 720	10	600 Kbps
SD	960 × 720	10	400 Kbps

## Watching Shared Screen

- **Watch screens shared by macOS/Windows users**

When a macOS/Windows user in a room starts screen sharing, the screen will be shared through a substream, and other users in the room will be notified through [onUserSubStreamAvailable](#) in `TRTCCloudDelegate`.

Users who want to watch the shared screen can start rendering the substream image of the remote user by calling the [startRemoteSubStreamView](#) API.

- **Watch screens shared by Android/iOS users**

When an Android/iOS user starts screen sharing, the screen will be shared through the primary stream, and other users in the room will be notified through [onUserVideoAvailable](#) in `TRTCCloudDelegate`.

Users who want to watch the shared screen can start rendering the primary stream of the remote user by calling the [startRemoteView](#) API.

```
//Sample code: watch the shared screen
- (void)onUserSubStreamAvailable:(NSString *)userId available:(BOOL)available {
    if (available) {
        [self.trtcCloud startRemoteSubStreamView:userId view:self.capturePreviewWindow.contentView];
    } else {
        [self.trtcCloud stopRemoteSubStreamView:userId];
    }
}
```

## FAQs

**Can more than one user in a room share their screens at the same time?**

Currently, a TRTC room can have only one screen sharing stream at a time.

**When a specified window ( `SourceTypeWindow` ) is shared, if the window size changes, will the resolution of the video stream change accordingly?**

By default, the SDK automatically adjusts encoding parameters according to the size of the shared window.

If you want a fixed resolution, call the `setSubStreamEncoderParam` API to set encoding parameters for screen sharing or specify the parameters when calling the `startScreenCapture` API.

# Web

Last updated : 2022-04-15 15:39:29

For a list of the browsers that support screen sharing, see [Browsers Supported](#). You can also use the [TRTC.isScreenShareSupported](#) API to check whether your current browser supports screen sharing. This document describes how to implement the screen sharing feature in the TRTC web SDK.

## Note :

- The TRTC web SDK does not support publishing substreams, and screen sharing streams are published as primary streams. Therefore, if a remote screen sharing stream is from a browser, the [RemoteStream.getType\(\)](#) API will return `main`. We recommend you set `userId` in such a way that you can tell from the ID that a user is sharing the screen from a browser.
- The TRTC SDKs for native applications (iOS, Android, macOS, Windows, etc.) support publishing substreams, and screen sharing streams are published as substreams. Therefore, if a remote screen sharing stream is from a native application, the [RemoteStream.getType\(\)](#) API will return `auxiliary`.

## Creating and Publishing a Screen Sharing Stream

### Note :

Follow the steps below to create a screen sharing stream and publish it.

### Step 1. Create a screen sharing stream

A screen sharing stream includes an audio track and a video track, and an audio track includes mic audio and system audio.

```
// Good:
// Capture only video
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// Capture mic audio and video
const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// Capture system audio and video
const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
```

```
// Bad:
const shareStream = TRTC.createStream({ camera: true, screen: true });
// or
const shareStream = TRTC.createStream({ camera: true, screenAudio: true });
```

Note :

- You cannot set both `audio` and `screenAudio` to `true` , nor can you set both `camera` and `screenAudio` to `true` . For more information about `screenAudio` , see "Capturing System Audio During Screen Sharing" below.
- You cannot set both `camera` and `screen` to `true` .

## Step 2. Initialize the screen sharing stream

During initialization, the browser will ask the user's permission to share the screen. If the user denies the permission or if the browser is not granted the permission by the system, the `NotReadableError` or `NotAllowedError` error will be returned. In such cases, you need to instruct the user to change the browser settings or grant the screen sharing permission and then initialize the screen sharing stream again.

Note :

For Safari, you need to initialize the screen sharing stream from an onclick callback. For details, see [FAQs](#).

```
try {
  await shareStream.initialize();
} catch (error) {
  // If the initialization of the screen sharing stream fails, notify the user and stop performing
  // subsequent steps including room entry and stream publishing.
  switch (error.name) {
    case 'NotReadableError':
      // Ask the user to check if the system has allowed the browser to record the screen.
      return;
    case 'NotAllowedError':
      if (error.message.includes('Permission denied by system')) {
        // Ask the user to check if the system has allowed the browser to record the screen.
      } else {
        // The user denies the permission or cancels screen sharing.
      }
    }
  }
}
```

```
return;
default:
  // An unknown error occurred during the initialization of the screen sharing stream. Ask the user
  // to try again.
  return;
}
}
```

### Step 3. Create a client object for screen sharing

We recommend you add the prefix `share` to the `userId` of the object to indicate that it is used for screen sharing.

```
const shareClient = TRTC.createClient({
  mode: 'rtc',
  sdkAppId,
  userId, // Example: 'share_teacher'
  userSig
});
// The client object enters the room.
try {
  await shareClient.join({ roomId });
  // ShareClient join room success
} catch (error) {
  // ShareClient join room failed
}
```

### Step 4. Publish the screen sharing stream

Use the client object created in step 1 to publish the stream. If it is successful, remote users will receive the stream.

```
try {
  await shareClient.publish(shareStream);
} catch (error) {
  // ShareClient failed to publish local stream
}
```

## Code

```
// We recommend you add the prefix `share` to the `userId` of the object to indicate that it is u
sed for screen sharing.
const userId = 'share_userId';
const roomId = 'roomId';
```

```
// Capture only video
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// Capture mic audio and video
// const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// Capture system audio and video
// const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
try {
  await shareStream.initialize();
} catch (error) {
  // If the initialization of the screen sharing stream fails, notify the user and stop performing
  // subsequent steps including room entry and stream publishing.
  switch (error.name) {
    case 'NotReadableError':
      // Ask the user to check if the system has allowed the browser to record the screen.
      return;
    case 'NotAllowedError':
      if (error.message.includes('Permission denied by system')) {
        // Ask the user to check if the system has allowed the browser to record the screen.
      } else {
        // The user denies the permission or cancels screen sharing.
      }
      return;
    default:
      // An unknown error occurred during the initialization of the screen sharing stream. Ask the user
      // to try again.
      return;
  }
}

const shareClient = TRTC.createClient({
  mode: 'rtc',
  sdkAppId,
  userId, // Example: 'share_teacher'
  userSig
});
// The client object enters the room.
try {
  await shareClient.join({ roomId });
  // ShareClient join room success
} catch (error) {
  // ShareClient join room failed
}

try {
  await shareClient.publish(shareStream);
} catch (error) {
  // ShareClient failed to publish local stream
}
```

## Configuring Screen Sharing Parameters

Screen sharing parameters include resolution, frame rate, and bitrate, which you can set using the `setScreenProfile()` API. Each profile value corresponds to a set of resolution, frame rate, and bitrate. The default value is `1080p`.

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });  
// For SetScreenProfile() to work, you must call it before you call initialize().  
shareStream.setScreenProfile('1080p');  
await shareStream.initialize();
```

You can also specify a custom value for the resolution, frame rate, and bitrate.

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });  
// For SetScreenProfile() to work, you must call it before you call initialize().  
shareStream.setScreenProfile({ width: 1920, height: 1080, frameRate: 5, bitrate: 1600 /* kbps */  
});  
await shareStream.initialize();
```

Recommended screen sharing settings:

Profile	Resolution (W x H)	Frame Rate (fps)	Bitrate (Kbps)
480p	640 x 480	5	900
480p_2	640 x 480	30	1000
720p	1280 x 720	5	1200
720p_2	1280 x 720	30	3000
1080p	1920 x 1080	5	1600
1080p_2	1920 x 1080	30	4000

Note :

Setting the parameters too high may cause unexpected results. We recommend you use the above settings.

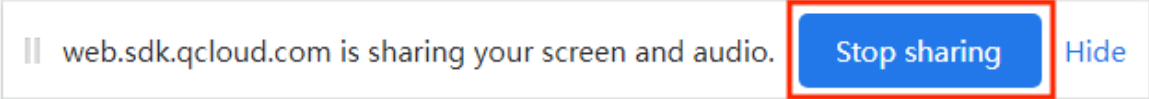
## Stopping Screen Sharing



```
// The screen sharing client stops publishing the stream.
await shareClient.unpublish(shareStream);
// Close the screen sharing stream.
shareStream.close();
// Leave the room.
await shareClient.leave();

// The above three steps are optional. You can determine what code to use according to the actual
// situation. Normally, you need to add code to determine whether the user has entered the room and
// whether the stream has been published. For more code samples, see the [demo source code](https://
// github.com/LiteAVSDK/TRTC_Web/blob/main/base-js/js/share-client.js).
```

A user may also stop screen sharing by clicking a built-in button in the browser, so it's necessary to listen for the screen sharing stopping event and, if the event occurs, take the necessary action.



```
// Listen for the screen sharing stopping event.
shareStream.on('screen-sharing-stopped', event => {
// Stop publishing the screen sharing stream.
await shareClient.unpublish(shareStream);
// Close the screen sharing stream.
shareStream.close();
// Leave the room.
await shareClient.leave();
});
```

## Publishing Both Camera and Screen Sharing Streams

A client can publish only one video track and one audio track. Therefore, to publish both the camera and screen sharing streams, you need to create two clients.

Below is an example:

- **client:** Publish the camera stream and subscribe to all remote streams except that of `shareClient`.
- **shareClient:** Publish the screen sharing stream and subscribe to no remote streams.

**Note :**

- You need to disable automatic subscription for `shareClient` so that it does not subscribe to remote streams. For details, see the [API document](#).
- For `client`, you need to unsubscribe from the stream of `shareClient`.

**Sample code:**

```
const client = TRTC.createClient({ mode: 'rtc', sdkAppId, userId, userSig });
// Set autoSubscribe to false to disable automatic subscription for shareClient.
const shareClient = TRTC.createClient({ mode: 'rtc', sdkAppId, `share_${userId}`, userSig, autoSubscribe: false });

// Unsubscribe from the stream of shareClient.
client.on('stream-added', event => {
  const remoteStream = event.stream;
  const remoteUserId = remoteStream.getUserId();
  if (remoteUserId === `share_${userId}`) {
    // Unsubscribe from the screen sharing stream.
    client.unsubscribe(remoteStream);
  } else {
    // Subscribe to other remote streams.
    client.subscribe(remoteStream);
  }
});

await client.join({ roomId });
await shareClient.join({ roomId });

const localStream = TRTC.createStream({ audio: true, video: true, userId });
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });

// The code for initialization and publishing is omitted. You can add the code based on your needs.
```

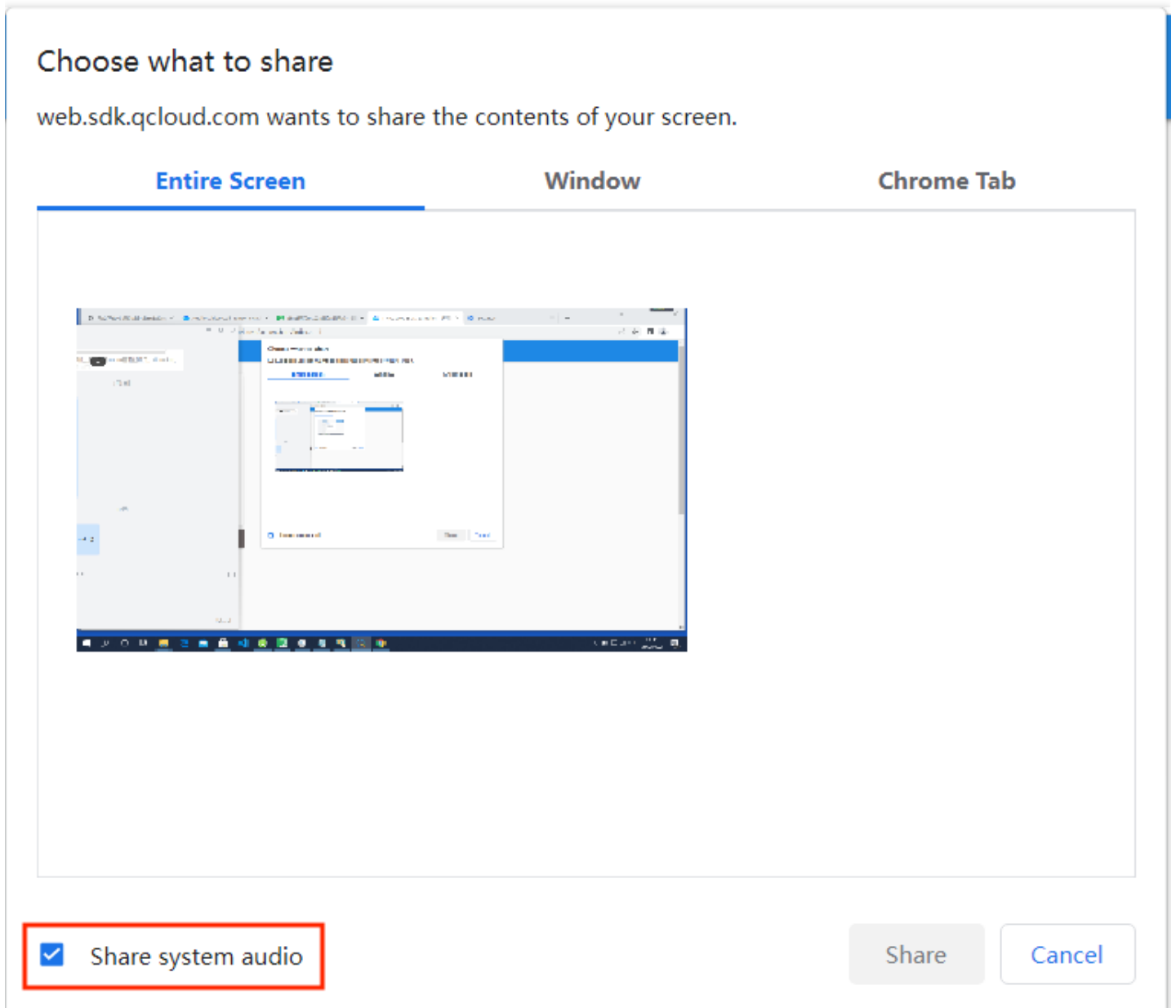
## Capturing System Audio During Screen Sharing

**System audio capturing is supported only on Chrome M74 and later versions. On Chrome for Windows and Chrome OS, you can capture the audio of the entire system, while on Chrome for Linux and macOS, you can only capture the audio of Chrome tabs. Other Chrome versions, OS, and browsers do not support system audio capturing.**

```
// Set screenAudio to true when creating the screen sharing stream. Don't set audio to true because you cannot capture mic and system audio at the same time.
```

```
const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });  
await shareStream.initialize();  
...
```

In the pop-up window, select **Share audio**, and the stream published will contain system audio.



## FAQs

## 1. What should I do if the error `getDisplayMedia must be called from a user gesture handler` occurs on Safari?

Safari does not support the screen capturing API `getDisplayMedia`. However, you can call it within one second of the callback for an onclick event. For details, see [this WebKit Bugzilla page](#).

```
// Good
async function onClick() {
  // We recommend you capture the stream first.
  const screenStream = TRTC.createStream({ screen: true });
  await screenStream.initialize();
  await client.join({ roomId: 123123 });
}

// Bad
async function onClick() {
  await client.join({ roomId: 123123 });
  // If it takes longer than one second for the client to enter the room, capturing will fail.
  const screenStream = TRTC.createStream({ screen: true });
  await screenStream.initialize();
}
```

## 2. For other questions, see [WebRTC Known Issues and Solutions](#).

# Flutter

Last updated : 2021-11-10 15:30:44

## Android

The TRTC SDK supports screen sharing on Android. This means you can share your screen with other users in the same room. Pay attention to the following points regarding this feature:

- Unlike the desktop edition, for Android, SDK versions earlier than v8.6 do not support substream screen sharing. Therefore, video capturing by the camera must be stopped first before screen sharing can start. Substream screen sharing is supported on v8.6 and later versions, so there is no need to stop video capturing by the camera.
- Screen sharing consumes CPU. On Android, a background app consuming CPU continuously is very likely to be killed by the system. The solution to this problem is creating a floating window after screen sharing starts. As Android does not kill apps with foreground views, your app can share the screen continuously without being killed by the system.

### Starting screen sharing

To start screen sharing on Android, simply call `startScreenCapture()` in `TRTCCloud`. However, to ensure the stability and video quality of screen sharing, you need to do the following.

### Adding an activity

Copy the activity below and paste it in the manifest file. You can skip this if the activity is already included in your project code.

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

### Setting video encoding parameters

By setting the first parameter `encParams` in `startScreenCapture()`, you can specify the encoding quality of screen sharing. If `encParams` is set to `null`, the SDK will use the encoding parameters set previously. We recommend the following settings:

Item	Parameter	Recommended Value for Regular Scenarios	Recommended Value for Text-based Teaching
Resolution	videoResolution	1280 × 720	1920 × 1080

Item	Parameter	Recommended Value for Regular Scenarios	Recommended Value for Text-based Teaching
Frame rate	videoFps	10 fps	8 fps
Highest bitrate	videoBitrate	1600 Kbps	2000 Kbps
Resolution adaption	enableAdjustRes	NO	NO

Note :

- As screen content generally does not change drastically, it is not economical to use a high frame rate. We recommend setting it to 10 fps.
- If the screen you share contains a large amount of text, you can increase the resolution and bitrate accordingly.
- The highest bitrate ( `videoBitrate` ) refers to the highest output bitrate when a shared screen changes dramatically. If the shared content does not change a lot, the actual encoding bitrate will be lower.

## Displaying a floating window

Since Android 7.0, apps running in the background tend to be killed by the system if they consume CPU. To prevent your app from being killed when it is sharing the screen in the background, you need to create a floating window when screen sharing starts, which also serves the purpose of reminding the user to avoid displaying personal information as his or her screen is being shared.

### Method: displaying a common floating window

The code in [tool.dart](#) offers an example of how to create a mini floating window similar to the one in VooV Meeting:

```
// Create a floating window when screen sharing starts to prevent the app from being killed when
running in the background
static void showOverlayWindow() {
  SystemWindowHeader header = SystemWindowHeader(
    title: SystemWindowText(
      text: "Screen being shared", fontSize: 14, textColor: Colors.black45),
    decoration: SystemWindowDecoration(startColor: Colors.grey[100]),
  );
  SystemAlertWindow.showSystemWindow(
    width: 18,
```

```
height: 95,
header: header,
margin: SystemWindowMargin(top: 200),
gravity: SystemWindowGravity.TOP,
);
}
```

## iOS

- **In-app sharing**

With in-app sharing, sharing is limited to the views of the current app. This feature is supported on iOS 13 and above. As content outside the current app cannot be shared, this feature is suitable for scenarios with high requirements on privacy protection.

- **Cross-app sharing**

Based on Apple's ReplayKit scheme, cross-app sharing allows the sharing of content across the system, but the steps required to implement this feature are more complicated than those for in-app sharing as an additional extension is needed.

### Scheme 1: in-app sharing on iOS

You can implement in-app sharing simply by calling the [startScreenCapture](#) API of the TRTC SDK, passing in the encoding parameter `TRTCVideoEncParam`, and setting the `appGroup` parameter to `''`. If `TRTCVideoEncParam` is set to `null`, the SDK will use the encoding parameters set previously.

We recommend the following encoding settings for screen sharing on iOS:

Item	Parameter	Recommended Value for Regular Scenarios	Recommended Value for Text-based Teaching
Resolution	videoResolution	1280 × 720	1920 × 1080
Frame rate	videoFps	10 fps	8 fps
Highest bitrate	videoBitrate	1600 Kbps	2000 Kbps
Resolution adaption	enableAdjustRes	NO	NO

Note :

- As screen content generally does not change drastically, it is not economical to use a high frame rate. We recommend setting it to 10 fps.
- If the screen you share contains a large amount of text, you can increase the resolution and bitrate accordingly.
- The highest bitrate ( `videoBitrate` ) refers to the highest output bitrate when a shared screen changes dramatically. If the shared content does not change a lot, the actual encoding bitrate will be lower.

## Scheme 2: cross-app sharing on iOS

### Sample code

You can find the sample code for cross-app sharing in the **ios** directory of the [TRTC demo](#). The directory contains the following files:

```
|—— Broadcast.Upload // Code for the screen recording process Broadcast Upload Extension. For
details, see step 2 below.
| |—— Broadcast.Upload.entitlements // Code for configuring an App Group to enable communicat
ion between processes
| |—— Broadcast.UploadDebug.entitlements // Code for configuring an App Group to enable commu
nication between processes (debug environment)
| |—— Info.plist
| |—— SampleHandler.swift // Code for receiving screen recording data from the system
|—— Resource // Resource file
|—— Runner // A simple TRTC demo
|—— TXLiteAVSDK_ReplayKitExt.framework //TXLiteAVSDK_ReplayKitExt SDK
```

You can run the demo as instructed in [README](#).

### Directions

To enable cross-app screen sharing on iOS, you need to add the screen recording process Broadcast Upload Extension, which works with the host app to push streams. A Broadcast Upload Extension is created by the system when a screen needs to be shared and is responsible for receiving the screen images captured by the system. For this, you need to do the following:

1. Create an App Group and configure it in Xcode (optional) to enable communication between the Broadcast Upload Extension and host app.
2. Create a target of Broadcast Upload Extension in your project and integrate into it `TXLiteAVSDK_ReplayKitExt.framework` from the SDK package, which is tailored for the extension module.
3. Make the host app wait to receive screen recording data from the Broadcast Upload Extension.



4. Edit the `pubspec.yaml` file and import the `replay_kit_launcher` plugin to make it possible to start screen sharing by tapping a button (optional), as in TRTC Demo Screen.

```
# Import the TRTC SDK and `replay_kit_launcher`  
dependencies:  
  tencent_trtc_cloud: ^0.2.1  
  replay_kit_launcher: ^0.2.0+1
```

Note :

If you skip [step 1](#), that is, if you do not configure an App Group (by passing `null` in the API), you can still enable the screen sharing feature, but its stability will be compromised. Therefore, to ensure the stability of screen sharing, we suggest that you configure an App Group as described in this document.

### Step 1. Create an App Group

Log in to <https://developer.apple.com/> and do the following. **You need to download the provisioning profile again afterwards.**

1. Click **Certificates, IDs & Profiles**.
2. Click "+" next to **Identifiers**.
3. Select **App Groups** and click **Continue**.
4. In the form that pops up, fill in the **Description** and **Identifier** boxes. For **Identifier**, type the `AppGroup` value passed in to the API. After this, click **Continue**.

The image consists of three screenshots from the Apple Developer portal, illustrating the process of registering an App Group. Red boxes and numbers highlight key steps:

- Screenshot 1:** The left sidebar of the Apple Developer portal. A red box with the number '1' highlights the 'Certificates, IDs & Profiles' link in the sidebar.
- Screenshot 2:** The 'Certificates, Identifiers & Profiles' page. A red box with the number '2' highlights the 'Identifiers' tab, and another red box with the number '3' highlights the '+ Add' button next to the 'Identifiers' tab.
- Screenshot 3:** The 'Register a New Identifier' page. A red box with the number '3' highlights the 'App Groups' option under the 'Cloud Containers' section.
- Screenshot 4:** The 'Register an App Group' form. A red box with the number '4' highlights the 'Register an App Group' button.

The 'Register an App Group' form includes fields for 'Description' and 'Identifier'. The 'Identifier' field has a note: 'We recommend using a reverse-domain name style string (i.e., com.domainname.appname).'

5. Select **Identifiers** on the top left sidebar, and click your App ID (you need to configure App ID for the host app and extension in the same way).
6. Select **App Groups** and click **Edit**.
7. In the form that pops up, select the App Group you created, click **Continue** to return to the edit page, and click **Save** to save the settings.

## Certificates, Identifiers & Profiles

Certificates

**Identifiers** +

App IDs

Identifiers

NAME	IDENTIFIER
liteavdemo	com.tencent.liteavdemo
liteavdemoReplayKitUpload	com.tencent.liteavdemo.ReplayKitUpload

## Certificates, Identifiers & Profiles

< All Identifiers

**Edit your App ID Configuration**

Platform

IOS, macOS, tvOS, watchOS

Description

liteavdemo

App ID Prefix

5GHU44CJHG (Team ID)

Bundle ID

com.tencent.liteavdemo (explicit)

Capabilities

ENABLED	NAME
<input type="checkbox"/>	Access WiFi Information
<input checked="" type="checkbox"/>	App Groups
<input type="checkbox"/>	Apple Pay Payment Processing

## App Group Assignment

Select the App Groups you wish to assign to the bundle.

☒ Select All

☒ RPLiveStreamShare

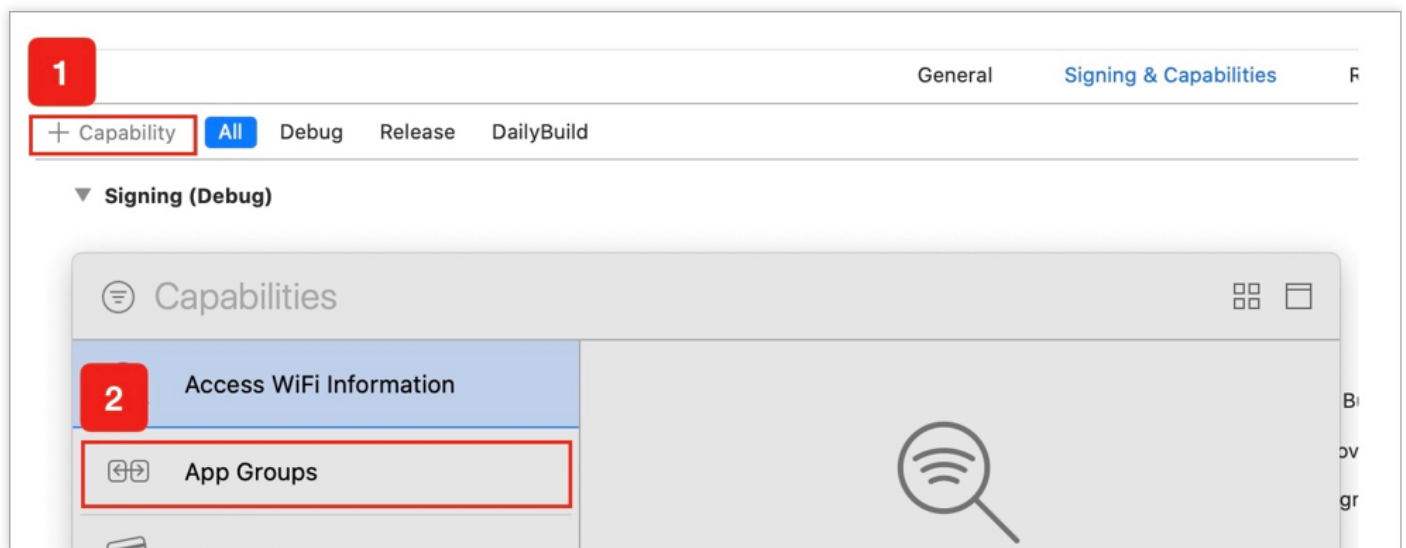
1 of 1 item(s) selected

8. Download the provisioning profile again and import it to Xcode.

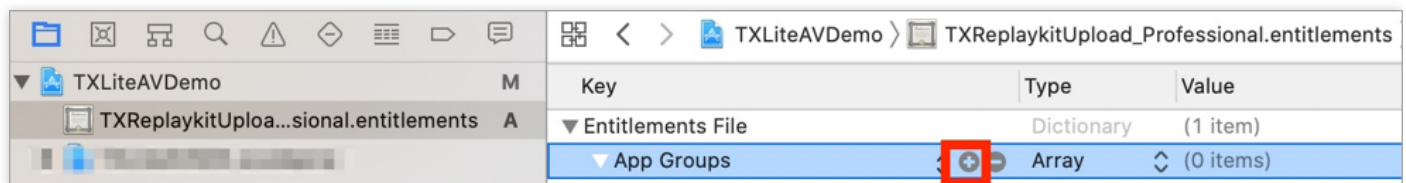
## Step 2. Create a Broadcast Upload Extension

1. In the Xcode menu, click **File > New > Target...**, and select **Broadcast Upload Extension**.
2. In the dialog box that pops up, enter the information required. You **don't need to** check **Include UI Extension**. Click **Finish** to complete the creation.
3. Drag `TXLiteAVSDK_ReplayKitExt.framework` in the SDK package into the project and select the target created.

4. Click **+ Capability**, and double-click **App Groups**, as shown below:



A file named `target name.entitlements` will appear in the file list as shown below. Select it, click "+", and enter the `App Group` created earlier.



5. Select the target of the host app **and configure it in the same way as described above.**
6. In the new target, Xcode will create a `SampleHandler.swift` file. Replace the file content with the following code. You need to **change APPGROUP in the code to the App Group Identifier created earlier.**

```
import ReplayKit
import TXLiteAVSDK_ReplayKitExt

let APPGROUP = "group.com.tencent.comm.trtc.demo"

class SampleHandler: RPBroadcastSampleHandler, TXReplayKitExtDelegate {

    let recordScreenKey = Notification.Name.init("TRTCRecordScreenKey")

    override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
        // User has requested to start the broadcast. Setup info from the UI extension can be supplied
        // but optional.
        TXReplayKitExt.sharedInstance().setup(withAppGroup: APPGROUP, delegate: self)
    }
}
```

```
override func broadcastPaused() {  
    // User has requested to pause the broadcast. Samples will stop being delivered.  
}  
  
override func broadcastResumed() {  
    // User has requested to resume the broadcast. Samples delivery will resume.  
}  
  
override func broadcastFinished() {  
    // User has requested to finish the broadcast.  
    TXReplayKitExt.sharedInstance().finishBroadcast()  
}  
  
func broadcastFinished(_ broadcast: TXReplayKitExt, reason: TXReplayKitExtReason) {  
    var tip = ""  
    switch reason {  
    case TXReplayKitExtReason.requestedByMain:  
        tip = "Screen sharing ended"  
        break  
    case TXReplayKitExtReason.disconnected:  
        tip = "App was disconnected"  
        break  
    case TXReplayKitExtReason.versionMismatch:  
        tip = "Integration error (SDK version mismatch)"  
        break  
    default:  
        break  
    }  
  
    let error = NSError(domain: NSStringFromClass(self.classForCoder), code: 0, userInfo: [NSLocalizedFailureReasonErrorKey:tip])  
    finishBroadcastWithError(error)  
}  
  
override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType: RPSampleBufferType) {  
    switch sampleBufferType {  
    case RPSampleBufferType.video:  
        // Handle video sample buffer  
        TXReplayKitExt.sharedInstance().sendVideoSampleBuffer(sampleBuffer)  
        break  
    case RPSampleBufferType.audioApp:  
        // Handle audio sample buffer for app audio  
        break  
    case RPSampleBufferType.audioMic:  
        // Handle audio sample buffer for mic audio  
        break  
    }
```

```
@unknown default:
// Handle other sample buffer types
fatalError("Unknown type of sample buffer")
}
}
}
```

### Step 3. Make the host app wait to receive data

Before screen sharing starts, the host app must be on standby to receive screen recording data from the Broadcast Upload Extension. To achieve this, follow these steps:

1. Make sure that camera capturing is disabled in `TRTCCloud` ; if not, call `stopLocalPreview` to disable it.
2. Call `startScreenCapture`, passing in the `AppGroup` set in [step 1](#) to put the SDK on standby.
3. The SDK will then wait for a user to trigger screen sharing. If a "triggering button" is not added as described in [step 4](#), users need to press and hold the screen recording button in the iOS Control Center to start screen sharing.
4. You can call `stopScreenCapture` to stop screen sharing at any time.

```
// Start screen sharing. You need to replace `APPGROUP` with the App Group created in the step
s above.
trtcCloud.startScreenCapture(
    TRTCVideoEncParam(
        videoFps: 10,
        videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720,
        videoBitrate: 1600,
        videoResolutionMode: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT,
    ),
    iosAppGroup,
);

// Stop screen sharing
await trtcCloud.stopScreenCapture();

// Event notification for the start of screen sharing, which can be received through `TRTCClou
dListener`
onRtcListener(type, param){
    if (type == TRTCCloudListener.onScreenCaptureStarted) {
        // Screen sharing starts.
```

```
}  
}
```

#### Step 4. Add a screen sharing triggering button (optional)

In [step 3](#), users need to start screen sharing manually by pressing and holding the screen recording button in the Control Center. To make it possible to start screen sharing by tapping a button in your app as in TRTC Demo Screen, follow these steps:

1. Add the `replay_kit_launcher` plugin to your project.
2. Add a button to your UI and call `ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);` in the response function of the button to activate the screen sharing feature.

```
// Customize a response for button tapping.  
onShareClick() async {  
  if (Platform.isAndroid) {  
    if (await SystemAlertWindow.requestPermissions) {  
      MeetingTool.showOverlayWindow();  
    }  
  } else {  
    // The screen sharing feature can only be tested on a real device.  
    ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);  
  }  
}
```

## Watching Shared Screen

- **Watch screens shared by Android/iOS users**

When an Android/iOS user starts screen sharing, the screen is shared via the primary stream, and other users in the room will be notified through `onUserVideoAvailable` in `TRTCCloudListener`.

Users who want to watch the shared screen can call the [startRemoteView](#) API to start rendering the primary stream of the remote user.

## FAQs

### Can there be multiple channels of screen sharing streams in a room at the same time?

Currently, each TRTC room can have only one channel of screen sharing stream.

# Live Streaming Mode

## iOS and macOS

Last updated : 2022-03-20 11:05:36

## Application Scenarios

TRTC supports four room entry modes. Video call ( `VideoCall` ) and audio call ( `VoiceCall` ) are the [call modes](#), and interactive video streaming ( `Live` ) and interactive audio streaming ( `VoiceChatRoom` ) are the live streaming modes.

The live streaming modes allow a maximum of 100,000 concurrent users in each room with smooth mic on/off. Co-anchoring latency is kept below 300 ms and watch latency below 1,000 ms. The live streaming modes are suitable for use cases such as low-latency interactive live streaming, interactive classrooms for up to 100,000 participants, video dating, online education, remote training, and mega-scale conferencing.

## How It Works

TRTC services use two types of server nodes: access servers and proxy servers.

- **Access server**

This type of nodes use high-quality lines and high-performance servers and are better suited to drive low-latency end-to-end calls.

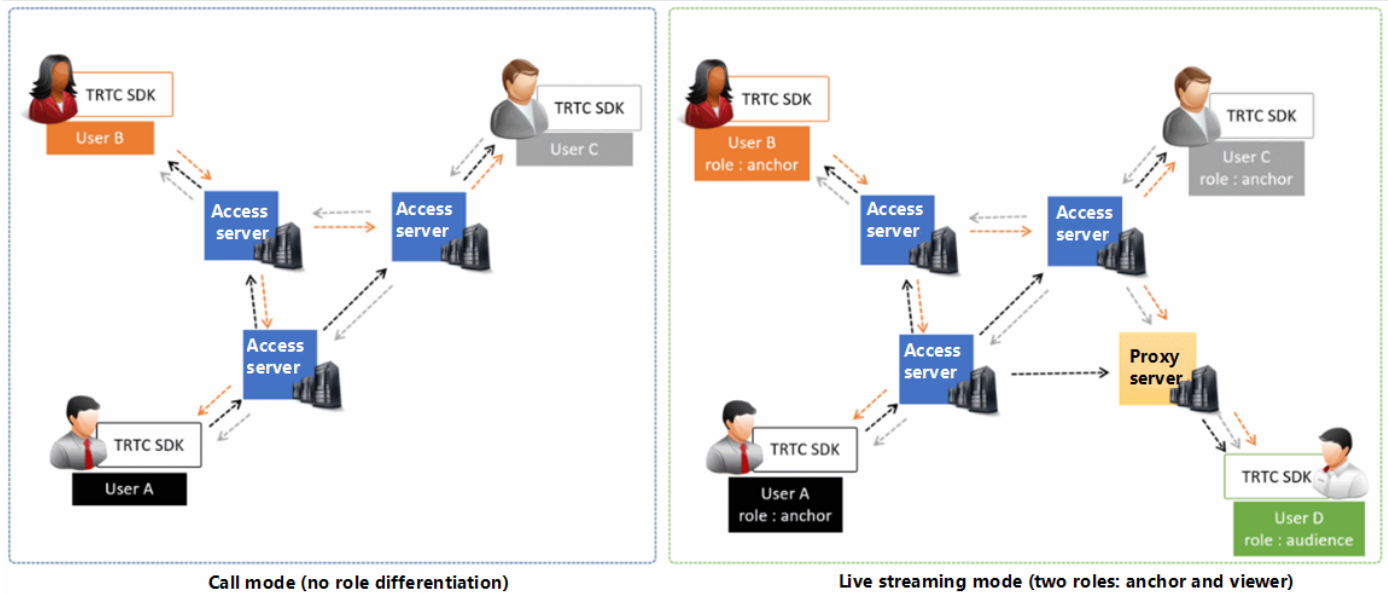
- **Proxy server**

This type of servers use mediocre lines and average-performance servers and are better suited to power high-concurrency stream pulling and playback.

In the live streaming modes, TRTC has introduced the concept of "role". Users are either in the role of "anchor" or "audience". Anchors are assigned to access servers, and audience to proxy servers. Each room allows up to 100,000 users in the role of audience.

For audience to speak, they must switch the role ( `switchRole` ) to "anchor". The switching process involves users being migrated from proxy servers to access servers. TRTC's low-latency streaming and smooth mic on/off technologies help keep this process short.





## Sample Code

You can visit [GitHub](#) to obtain the sample code used in this document.

master TRTCSDK / iOS / TRTC-API-Example-OC / Basic / Go to file Add file ...

garyxgwang Update iOS TRTC-API-Example-OC ✓ c45668f 11 minutes ago History

AudioCall	Update iOS TRTC-API-Example-OC	11 minutes ago
Live	Update iOS TRTC-API-Example-OC	11 minutes ago
ScreenShare	Update iOS TRTC-API-Example-OC	11 minutes ago
VideoCall	Update iOS TRTC-API-Example-OC	11 minutes ago
VoiceChatRoom	Update iOS TRTC-API-Example-OC	11 minutes ago

Note :

If your access to GitHub is slow, download the ZIP file [here](#).

## Directions

### Step 1. Integrate the SDKs

You can integrate the **TRTC SDK** into your project in the following ways:

### Method 1: integrating through CocoaPods

1. Install **CocoaPods**. For detailed directions, please see [Getting Started](#).
2. Open the `Podfile` file in the root directory of your project and add the code below.

Note :

If you cannot find a `Podfile` file in the directory, run the `pod init` command to create one and add the code below.

```
target 'Your Project' do
  pod 'TXLiteAVSDK_TRTC'
end
```

3. Run the command below to install the **TRTC SDK**.

```
pod install
```

After successful installation, an **XCWORKSPACE** file will be generated in the root directory of your project.

4. Open the **XCWORKSPACE** file.

### Method 2: manual integration

If you do not want to install CocoaPods, or your access to CocoaPods repositories is slow, you can download the [ZIP file](#) of the SDK and integrate it into your project as instructed in [SDK Quick Integration > iOS](#).

### Step 2. Add device permission requests

Add camera and mic permission requests in the `Info.plist` file.

Key	Value
Privacy - Camera Usage Description	The reason for requesting camera permission, for example, "camera access is required to capture video"

Key	Value
Privacy - Microphone Usage Description	The reason for requesting mic permission, for example, "mic access is required to capture audio"

### Step 3. Initialize an SDK instance and configure event callbacks

1. Call the `sharedInstance()` API to create a `TRTCCloud` instance.

```
// Create a `TRTCCloud` instance
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;
```

2. Set the attributes of `delegate` to subscribe to event callbacks and listen for event and error notifications.

```
// Error events must be listened for and captured, and error messages should be sent to users.
- (void)onError:(TXLiteAVError)errCode errMsg:(NSString *)errMsg extInfo:(NSDictionary *)extInfo {
    if (ERR_ROOM_ENTER_FAIL == errCode) {
        [self toastTip:@"Failed to enter room"];
        [self.trtcCloud exitRoom];
    }
}
```

### Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom()` API, you need to pass in a key parameter `TRTCParams`, which includes the following required fields:

Parameter	Type	Description	Example
sdkAppId	Number	Application ID, which you can view in the <a href="#">TRTC console</a> .	1400000123
userId	String	Can contain only letters (a-z and A-Z), digits (0-9), underscores, and hyphens. We recommend you set it based on your business account system.	test_user_001
userSig	String	<code>userSig</code> is calculated based on <code>userId</code> . For the calculation method, see <a href="#">UserSig</a> .	eJyrVareCeYrSy1Ssll...
roomId	Number	Numeric room ID. For string-type room ID, use <code>strRoomId</code> in <code>TRTCParams</code> .	29834

Note :

- In TRTC, users with the same `userId` cannot be in the same room at the same time as it will cause a conflict.
- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## Step 5. Enable camera preview and mic capturing

1. Call `startLocalPreview()` to enable preview of the local camera. The SDK will ask for camera permission.
2. Call `setLocalViewFillMode()` to set the display mode of the local video image:
  - `Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
3. Call `setVideoEncoderParam()` to set the encoding parameters for the local video, which determine the [quality of your video](#) seen by other users in the room.
4. Call `startLocalAudio()` to turn the mic on. The SDK will ask for mic permission.

```
// Sample code: publish the local audio/video stream
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];

// Set local video encoding parameters
TRTCVideoEncParam *encParams = [TRTCVideoEncParam new];
encParams.videoResolution = TRTCVideoResolution_640_360;
encParams.videoBitrate = 550;
encParams.videoFps = 15;

[self.trtcCloud setVideoEncoderParam:encParams];
```

## Step 6. Set beauty filters

1. Call `getBeautyManager()` to get the beauty filter management class `TXBeautyManager`.
2. Call `setBeautyStyle()` to set the beauty filter style.

- **Smooth** : smooth. This style features more obvious skin smoothing effect and is typically used by influencers.
  - **Nature** : natural. This style retains more facial details and is more natural.
  - **Pitu** : this style is supported only in the [Enterprise Edition](#).
3. Call `setBeautyLevel()` to set the skin smoothing strength ( `5` is recommended).
  4. Call `setWhitenessLevel()` to set the skin brightening strength ( `5` is recommended).
  5. Given the yellow tint of the iPhone camera, we recommended that you call `setFilter()` to apply the skin brightening filter to your video. You can download the file for the filter [here](#).

## Step 7. Create a room and push streams

1. Set the `role` field in `TRTCParams` to `TRTCRoleType.anchor` to take the role of “anchor”.
2. Call `enterRoom()`, specifying `appScene`, and a room whose ID is the value of the `roomId` field in `TRTCParams` will be created.
  - `TRTCAppScene.LIVE` : the interactive video streaming mode, which is used in the example of this document
  - `TRTCAppScene.voiceChatRoom` : the interactive audio streaming mode
3. After the room is created, start encoding and transferring audio/video data. The SDK will return the `onEnterRoom(result)` callback. If `result` is greater than 0, room entry succeeds, and the value indicates the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.

```
- (void)enterRoom() {
    TRTCParams *params = [TRTCParams new];
    params.sdkAppId = SDKAppID;
    params.roomId = _roomId;
    params.userId = _userId;
    params.role = TRTCRoleAnchor;
    params.userSig = [GenerateTestUserSig genTestUserSig:params.userId];
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Entered room successfully"];
    } else {
        [self toastTip:@"Failed to enter room"];
    }
}
```

## Step 8. Enter the room as audience

1. Set the `role` field in `TRTCParams` to `TRTCRoleType.audience` to take the role of “audience”.
2. Call `enterRoom()` to enter the room whose ID is the value of the `roomId` field in `TRTCParams`, specifying `appScene`.
  - `TRTCAppScene.LIVE`: the interactive video streaming mode, which is used in the example of this document
  - `TRTCAppScene.voiceChatRoom`: the interactive audio streaming mode
3. Watch the anchor’s video:
  - If you know the anchor’s `userId`, call `startRemoteView(userId, view: view)` with the anchor’s `userId` passed in to play the anchor's video.
  - If you do not know the anchor’s `userId`, find the anchor’s `userId` in the `onUserVideoAvailable()` callback, which you will receive after room entry, and call `startRemoteView(userId, view: view)` with the anchor’s `userId` passed in to play the anchor’s video.

## Step 9. Co-anchor

1. Call `switch(TRTCRoleType.TRTCRoleAnchor)` to switch the role to “anchor” (`TRTCRoleType.TRTCRoleAnchor`).
2. Call `startLocalPreview()` to enable preview of the local image.
3. Call `startLocalAudio()` to enable mic capturing.

```
// Sample code: start co-anchoring
[self.trtcCloud switchRole:TRTCRoleAnchor];
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];

// Sample code: end co-anchoring
[self.trtcCloud switchRole:TRTCRoleAudience];
[self.trtcCloud stopLocalAudio];
[self.trtcCloud stopLocalPreview]
```

## Step 10. Compete across rooms

Anchors from two rooms can compete with each other without exiting their current rooms.

1. Anchor A calls the `connectOtherRoom()` API. The API uses parameters in JSON strings, so anchor A needs to pass the `roomId` and `userId` of anchor B in the format of `{"roomId": 978, "userId": "userB"}` to the API.
2. After the cross-room call is set up, anchor A will receive the `onConnectOtherRoom()` callback, and all users in both rooms will receive the `onUserVideoAvailable()` and `onUserAudioAvailable()` callbacks.  
For example, after anchor A in room "001" uses `connectOtherRoom()` to call anchor B in room "002" successfully all users in room "001" will receive the `onUserVideoAvailable(B, available: true)` and `onUserAudioAvailable(B, available: true)` callbacks, and all users in room "002" will receive the `onUserVideoAvailable(A, available: true)` and `onUserAudioAvailable(A, available: true)` callbacks.
3. Users in both rooms can call `startRemoteView(userId, view: view)` to play the video of the anchor in the other room, and audio will be played back automatically.

```
// Sample code: cross-room competition
NSMutableDictionary * jsonDict = [[NSMutableDictionary alloc] init];
[jsonDict setObject:@([_otherRoomIdTextField.text intValue]) forKey:@"roomId"];
[jsonDict setObject:_otherUserIdTextField.text forKey:@"userId"];
NSData* jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:NSJSONWritingPrettyPrinted error:nil];
NSString* jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
[self.trtcCloud connectOtherRoom:jsonString];
```

## Step 11. Exit the room

Call `exitRoom()` to exit the room. The SDK disables and releases devices such as cameras and mics during room exit. Therefore, room exit is not an instant process. It completes only after the `onExitRoom()` callback is received.

```
// Please wait for the `onExitRoom` callback after calling the room exit API.
[self.trtcCloud exitRoom];

- (void)onExitRoom:(NSInteger)reason {
    NSLog(@"Exited room: reason: %ld", reason)
}
```

### Note :

If your application integrates multiple audio/video SDKs, please wait after you receive the `onExitRoom` callback to start other SDKs; otherwise, the device busy error may occur.





# Android

Last updated : 2022-03-20 11:04:50

## Application Scenarios

TRTC supports four room entry modes. Video call ( `VideoCall` ) and audio call ( `VoiceCall` ) are the [call modes](#), and interactive video streaming ( `Live` ) and interactive audio streaming ( `VoiceChatRoom` ) are the live streaming modes.

The live streaming modes allow a maximum of 100,000 concurrent users in each room with smooth mic on/off. Co-anchoring latency is kept below 300 ms and watch latency below 1,000 ms. The live streaming modes are suitable for use cases such as low-latency interactive live streaming, interactive classrooms for up to 100,000 participants, video dating, online education, remote training, and mega-scale conferencing.

## How It Works

TRTC services use two types of server nodes: access servers and proxy servers.

- **Access server**

This type of nodes use high-quality lines and high-performance servers and are better suited to drive low-latency end-to-end calls.

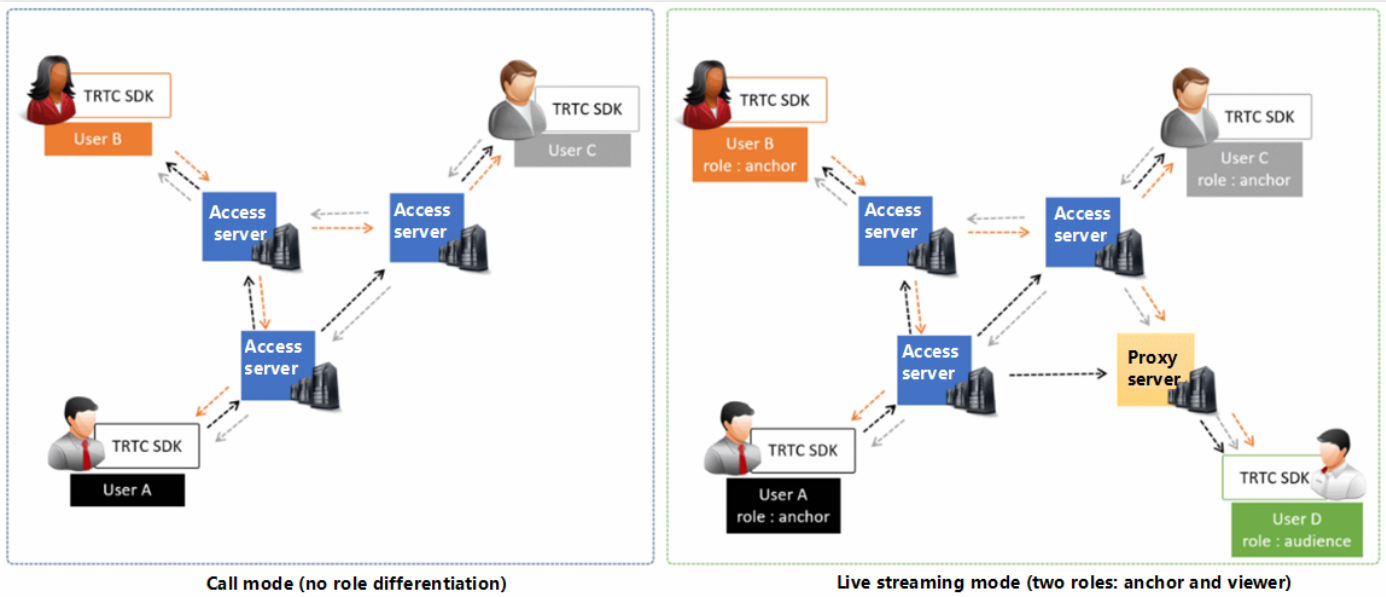
- **Proxy server**

This type of servers use mediocre lines and average-performance servers and are better suited to power high-concurrency stream pulling and playback.

In the live streaming modes, TRTC has introduced the concept of "role". Users are either in the role of "anchor" or "audience". Anchors are assigned to access servers, and audience to proxy servers. Each room allows up to 100,000 users in the role of audience.

For audience to speak, they must switch the role ( `switchRole` ) to "anchor". The switching process involves users being migrated from proxy servers to access servers. TRTC's low-latency streaming

and smooth mic on/off technologies help keep this process short.



## Sample Code

You can visit [GitHub](#) to obtain the sample code used in this document.

master ▾ TRTCSDK / Android / TRTC-API-Example / Basic /

Go to file Add file ▾ ...

garyxgwang Update Android TRTC-API-Example ✓ 6444d46 3 hours ago History

..		
AudioCall	Update Android TRTC-API-Example	3 hours ago
Live	Update Android TRTC-API-Example	3 hours ago
ScreenShare	Update Android TRTC-API-Example	3 hours ago
VideoCall	Update Android TRTC-API-Example	3 hours ago
VoiceChatRoom	Update Android TRTC-API-Example	3 hours ago

Note :

If your access to GitHub is slow, download the ZIP file [here](#).

## Directions

### Step 1. Integrate the SDKs

You can integrate the **TRTC SDK** into your project in the following ways:

### Method 1: automatic loading (AAR)

The TRTC SDK has been released to the mavenCentral repository, and you can configure Gradle to download updates automatically.

The TRTC SDK has integrated `TRTC-API-Example`, which offers sample code for your reference. Use Android Studio to open your project and follow the steps below to modify the `app/build.gradle` file.

1. Add the TRTC SDK dependency to `dependencies`.

```
dependencies {  
    compile 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release'  
}
```

2. In `defaultConfig`, specify the CPU architecture to be used by your application.

Note :

Currently, the TRTC SDK supports armeabi, armeabi-v7a, and arm64-v8a.

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

3. Click **Sync Now** to sync the SDKs.

If you have no problem connecting to mavenCentral, the SDK will be downloaded and integrated into your project automatically.

### Method 2: manual integration

You can download the [ZIP file](#) of the SDK and integrate it into your project as instructed in [SDK Quick Integration > Android](#).

### Step 2. Configure app permissions

Add camera, mic, and network permission requests in `AndroidManifest.xml`.

```

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

```

### Step 3. Initialize an SDK instance and configure event callbacks

1. Call the `sharedInstance()` API to create a `TRTCCloud` instance.

```

// Create a `TRTCCloud` instance
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener());

```

2. Set the attributes of `setListener` to subscribe to event callbacks and listen for event and error notifications.

```

// Error notifications indicate that the SDK has stopped working and therefore must be listened for
@Override
public void onError(int errCode, String errMsg, Bundle extraInfo) {
    Log.d(TAG, "sdk callback onError");
    if (activity != null) {
        Toast.makeText(activity, "onError: " + errMsg + "[" + errCode + "]", Toast.LENGTH_SHORT).show();
    }
    if (errCode == TXLiteAVCode.ERR_ROOM_ENTER_FAIL) {
        activity.exitRoom();
    }
}

```

### Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom()` API, you need to pass in a key parameter `TRTCParams`, which includes the following required fields:

Parameter	Type	Description	Example
<code>sdkAppId</code>	Number	Application ID, which you can view in the <a href="#">TRTC console</a> .	1400000123
<code>userId</code>	String	Can contain only letters (a-z and A-Z), digits (0-9), underscores, and hyphens. We recommend you set it based on your business account system.	test_user_001
<code>userSig</code>	String	<code>userSig</code> is calculated based on <code>userId</code> . For the calculation method, please see <a href="#">UserSig</a> .	eyJrVareCeYrSy1Ssll...
<code>roomId</code>	Number	Numeric room ID. For string-type room ID, use <code>strRoomId</code> in <code>TRTCParams</code> .	29834

Note :

-In TRTC, users with the same `userId` cannot be in the same room at the same time as it will cause a conflict.

- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## Step 5. Enable camera preview and mic capturing

1. Call `startLocalPreview()` to enable preview of the local camera. The SDK will ask for camera permission.
2. Call `setLocalViewFillMode()` to set the display mode of the local video image:
  - `Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
3. Call `setVideoEncoderParam()` to set the encoding parameters for the local video, which determine the [quality of your video](#) seen by other users in the room.
4. Call `startLocalAudio()` to turn the mic on. The SDK will ask for mic permission.

```
// Sample code: publish the local audio/video stream
mTRTCCloud.setLocalViewFillMode(TRTC_VIDEO_RENDER_MODE_FIT);
mTRTCCloud.startLocalPreview(mIsFrontCamera, localView);
// Set local video encoding parameters
TRTCCloudDef. TRTCVideoEncParam encParam = new TRTCCloudDef. TRTCVideoEncParam();
encParam.videoResolution = TRTCCloudDef. TRTC_VIDEO_RESOLUTION_960_540;
encParam.videoFps = 15;
encParam.videoBitrate = 1200;
encParam.videoResolutionMode = TRTCCloudDef. TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
mTRTCCloud.setVideoEncoderParam(encParam);
mTRTCCloud.startLocalAudio();
```

## Step 6. Set beauty filters

1. Call `getBeautyManager()` to get the beauty filter management class `TXBeautyManager`.
2. Call `setBeautyStyle()` to set the beauty filter style.
  - `Smooth` : smooth. This style features more obvious skin smoothing effect and is typically used by influencers.
  - `Nature` : natural. This style retains more facial details and is more natural.
  - `Pitu` : this style is supported only in the [Enterprise Edition](#).
3. Call `setBeautyLevel()` to set the skin smoothing strength ( `5` is recommended).
4. Call `setWhitenessLevel()` to set the skin brightening strength ( `5` is recommended).

## Step 7. Create a room and push streams

1. Set the `role` field in `TRTCParams` to `TRTCCloudDef. TRTCRoleAnchor` to take the role of “anchor”.
2. Call `enterRoom()`, specifying `appScene`, and a room whose ID is the value of the `roomId` field in `TRTCParams` will be created.
  - `TRTCCloudDef. TRTC_APP_SCENE_LIVE` : the interactive video streaming mode, which is used in the example of this document
  - `TRTCCloudDef. TRTC_APP_SCENE_VOICE_CHATROOM` : the interactive audio streaming mode
3. After the room is created, start encoding and transferring audio/video data. The SDK will return the `onEnterRoom(result)` callback. If `result` is greater than 0, room entry succeeds, and the value indicates the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.

```
public void enterRoom() {
    TRTCCloudDef.TRTCPParams trtcParams = new TRTCCloudDef.TRTCPParams();
    trtcParams.sdkAppId = sdkappid;
    trtcParams.userId = userid;
    trtcParams.roomId = 908;
    trtcParams.userSig = usersig;
    mTRTCCloud.enterRoom(trtcParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        toastTip("Entered room successfully; the total time used is [¥(result)] ms")
    } else {
        toastTip("Failed to enter the room; the error code is [¥(result)]")
    }
}
```

## Step 8. Enter the room as audience

1. Set the `role` field in `TRTCCloudDef.TRTCPParams` to `TRTCCloudDef.TRTCRoleAudience` to take the role of "audience".
2. Call `enterRoom()` to enter the room whose ID is the value of the `roomId` field in `TRTCCloudDef.TRTCPParams`, specifying `appScene`.
  - `TRTCCloudDef.TRTC_APP_SCENE_LIVE`: the interactive video streaming mode, which is used in the example of this document
  - `TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM`: the interactive audio streaming mode
3. Watch the anchor's video:
  - If you know the anchor's `userId`, call `startRemoteView(userId, view)` with the anchor's `userId` passed in to play the anchor's video.
  - If you do not know the anchor's `userId`, find the anchor's `userId` in the `onUserVideoAvailable()` callback, which you will receive after room entry, and call `startRemoteView(userId, view)` with the anchor's `userId` passed in to play the anchor's video.

## Step 9. Co-anchor

1. Call `switchRole(TRTCCloudDef.TRTCRoleAnchor)` to switch the role to "anchor"  
( `TRTCCloudDef.TRTCRoleAnchor` ).
2. Call `startLocalPreview()` to enable preview of the local image.
3. Call `startLocalAudio()` to enable mic capturing.

```
// Sample code: start co-anchoring
mTrtcCloud.switchRole(TRTCCloudDef. TRTCRoleAnchor);
mTrtcCloud.startLocalAudio();
mTrtcCloud.startLocalPreview(mIsFrontCamera, localView);
// Sample code: end co-anchoring
mTrtcCloud.switchRole(TRTCCloudDef. TRTCRoleAudience);
mTrtcCloud.stopLocalAudio();
mTrtcCloud.stopLocalPreview();
```

## Step 10. Compete across rooms

Anchors from two rooms can compete with each other without exiting their current rooms.

1. Anchor A calls the [connectOtherRoom\(\)](#) API. The API uses parameters in JSON strings, so anchor A needs to pass the `roomId` and `userId` of anchor B in the format of `{"roomId": 978, "userId": "userB"}` to the API.
2. After the cross-room call is set up, anchor A will receive the [onConnectOtherRoom\(\)](#) callback, and all users in both rooms will receive the [onUserVideoAvailable\(\)](#) and [onUserAudioAvailable\(\)](#) callbacks.  
For example, after anchor A in room "001" uses `connectOtherRoom()` to call anchor B in room "002" successfully, all users in room "001" will receive the `onUserVideoAvailable(B, true)` and `onUserAudioAvailable(B, true)` callbacks, and all users in room "002" will receive the `onUserVideoAvailable(A, true)` and `onUserAudioAvailable(A, true)` callbacks.
3. Users in both rooms can call [startRemoteView\(userId, view\)](#) to play the video of the anchor in the other room, and audio will be played automatically.

```
// Sample code: cross-room competition
mTRTCCloud.ConnectOtherRoom(String.format("{\"roomId\":%s,\"userId\":\"%s\"}", roomId, username));
```

## Step 11. Exit the room

Call [exitRoom\(\)](#) to exit the room. The SDK disables and releases devices such as cameras and mics during room exit. Therefore, room exit is not an instant process. It completes only after the [onExitRoom\(\)](#) callback is received.

```
// Please wait for the `onExitRoom` callback after calling the room exit API.
mTRTCCloud.exitRoom()
@Override
public void onExitRoom(int reason) {
    Log.i(TAG, "onExitRoom: reason = " + reason);
}
```



Note :

If your application integrates multiple audio/video SDKs, please wait after you receive the `onExitRoom` callback to start other SDKs; otherwise, the device busy error may occur.

# Windows

Last updated : 2022-01-05 11:57:22

## Overview

This document describes how to use the TRTC SDK to build a live streaming service that supports both co-anchoring and high-concurrency streaming to over 10,000 users. Only the most commonly used APIs are covered in this document. To learn about other APIs, please see the [API documentation](#).

## Sample Code

Platform	Sample Code
Windows (MFC)	<a href="#">TRTCMainViewController.cpp</a>
Windows (Duilib)	<a href="#">TRTCMainViewController.cpp</a>
Windows (C#)	<a href="#">TRTCMainForm.cs</a>

## Online Live Streaming

### 1. Initialize the SDK

The first step is to get a singleton object of `TRTCCloud` and subscribe to the SDK's event callbacks.

- Inherit the `ITRTCCloudCallback` callback API class and rewrite the callback APIs for key events including room entry/exit by local user, room entry/exit by remote user, error event, and warning event.
- Call the `addCallback` API to subscribe to the SDK's events.

Note :

If `addCallback` is called N times, the SDK will trigger N callbacks for the same event. Therefore, you are advised to call `addCallback` only once.

- C++
- C#

```
// TRTCMainViewController.h

// Inherit the `ITRTCCloudCallback` callback API class
class TRTCMainViewController : public ITRTCCloudCallback
{
public:
    TRTCMainViewController();
    virtual ~TRTCMainViewController();

    virtual void onError(TXLiteAVError errCode, const char* errMsg, void* arg);
    virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void* arg);
    virtual void onEnterRoom(uint64_t elapsed);
    virtual void onExitRoom(int reason);
    virtual void onRemoteUserEnterRoom(const char* userId);
    virtual void onRemoteUserLeaveRoom(const char* userId, int reason);
    virtual void onUserVideoAvailable(const char* userId, bool available);
    virtual void onUserAudioAvailable(const char* userId, bool available);
    ...
private:
    ITRTCCloud * m_pTRTCSdk = NULL ;
    ...
}

// TRTCMainViewController.cpp

TRTCMainViewController::TRTCMainViewController()
{
    // Create a `TRTCCloud` instance
    m_pTRTCSdk = getTRTCShareInstance();

    // Subscribe to the SDK's events
    m_pTRTCSdk->addCallback(this);
}

TRTCMainViewController::~~TRTCMainViewController()
{
    // Unsubscribe from the SDK's events
    if(m_pTRTCSdk) {
        m_pTRTCSdk->removeCallback(this);
    }

    // Release the `TRTCCloud` instance
    if(m_pTRTCSdk != NULL) {
```

```
destroyTRTCShareInstance();
m_pTRTCSDK = null;
}
}

// Error notifications indicate that the SDK has stopped working and therefore must be listened f
or.
virtual void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* ar
g)
{
    if (errCode == ERR_ROOM_ENTER_FAIL) {
        LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
        exitRoom();
    }
}
```

## 2. Assemble TRTCParams

TRTCParams is the most critical parameter in the SDK. It contains four required fields: sdkAppId , userId , userSig , and roomId .

- **SDKAppID**

Log in to the [TRTC console](#). If you don't have an application yet, create one, and you will see its SDKAppID .

- **userId**

A custom string, which you can keep in line with the naming of your account. Please note that **there cannot be users with identical userId in a room.**

- **userSig**

Calculated based on SDKAppID and userID . For details, see [UserSig](#).

- **roomId**

A custom number. Note that **you cannot assign the same roomId to two rooms under the same application.** For string-type room ID, use strRoomId in TRTCParams .

## 3. Enable preview of the local camera

Camera capturing is disabled by default. You can call startLocalPreview to turn the local camera on and enable preview, and stopLocalPreview to disable camera capturing and preview.

Before enabling preview of the local camera, you can call setLocalViewFillMode to set the video display mode to Fill or Fit . Video may be resized proportionally in both modes, but they differ in

that:

- In the `Fill` mode, the image fills the entire screen. If the dimensions of the image do not match those of the screen after scaling, the parts that do not fit are cropped.
- In the `Fit` mode, the image is displayed in whole. If the dimensions of the image do not match those of the screen after scaling, the unoccupied space is painted black.
- [C++](#)
- [C#](#)

```
void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
    // Get the handle of the rendering window
    CWnd *pLocalVideoView = GetDlgItem(IDC_LOCAL_VIDEO_VIEW);
    HWND hwnd = pLocalVideoView->GetSafeHwnd();

    if(m_pTRTCSdk)
    {
        // Call the APIs below to set the rendering mode and rendering window
        m_pTRTCSdk->setLocalViewFillMode(TRTCVideoFillMode_Fit);
        m_pTRTCSdk->startLocalPreview(hwnd);
    }
}
```

## 4. Enable mic capturing

Mic capturing is disabled by default. Call `startLocalAudio` to enable local audio capturing and send the data captured, and `stopLocalAudio` to disable audio capturing. You can call `startLocalAudio` after `startLocalPreview`.

Note :

After you call `startLocalAudio`, the SDK will check mic access and will ask for mic permission from the user if it does not have access.

## 5. Create a room and push streams

Call `enterRoom` to create a room, setting `role` to `TRTCRoleAnchor` (anchor) and specifying `roomId` in the `TRTCParams` parameter.

Specify `appScene` , which indicates the application scenario. `TRTCApSceneLIVE` (online live streaming) is used in the example of this document.

- If the room is created successfully, you will receive the `onEnterRoom` callback, in which the `elapsed` field represents the time (ms) room entry takes.
  - If room creation fails, you will receive the `onError` callback, which contains `errCode` (error code, whose value is `ERR_ROOM_ENTER_FAIL` ; for other error code values, please see `TXLiteAVCode.h` ), `errMsg` (error message), and `extraInfo` (reserved parameter).
- [C++](#)
  - [C#](#)

```
// TRTCMainViewController.cpp

void TRTCMainViewController::startBroadCasting()
{
    // For the definition of `TRTCParams`, please see the header file `TRTCCloudDef.h`.
    TRTCParams params;
    params.sdkAppId = sdkappid;
    params.userId = userid;
    params.userSig = usersig;
    params.roomId = 908; // Set it to the ID of the room you want to enter
    params.role = TRTCRoleAnchor; //Anchor
    if(m_pTRTCSDK)
    {
        m_pTRTCSDK->enterRoom(params, TRTCApSceneLIVE);
    }
}

void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
    if(errCode == ERR_ROOM_ENTER_FAIL)
    {
        LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
        // Check whether `userSig` is valid, network is normal, etc.
    }
}

...

void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
    LOGI(L"onEnterRoom elapsed[%lld]", elapsed);

    // Enable local video preview. For details, please see the sections below about encoding settings
```

```
and local video preview
}
```

## 6. Enable/Disable the privacy mode

At some points during a live stream, you may not want to publish your video or audio for privacy concerns. You can call `muteLocalVideo` to stop publishing local video and `muteLocalAudio` to stop publishing local audio.

## 7. Enter the room as audience

Call `enterRoom` to enter the room, specifying the room number via the `roomId` field in `TRTCParams`. Set `appScene` to `TRTCAppSceneLIVE` (online live streaming), and `role` to `TRTCRoleAudience` (audience).

- C++
- C#

```
void TRTCMainViewController::startPlaying()
{
    // For the definition of `TRTCParams`, please see the header file `TRTCCloudDef.h`.
    TRTCParams params;
    params.sdkAppId = sdkappid;
    params.userId = userid;
    params.userSig = usersig;
    params.roomId = 908; // Set it to the ID of the room you want to enter
    params.role = TRTCRoleAudience; // Viewer
    if(m_pTRTCSdk)
    {
        m_pTRTCSdk->enterRoom(params, TRTCAppSceneLIVE);
    }
}
```

If the anchor is in the room, you can find the anchor's `userid` in the `onUserVideoAvailable` callback in `TRTCCloudDelegate`, and then call `startRemoteView` to display the anchor's video.

Call `setRemoteViewFillMode` to set the video display mode to `Fill` or `Fit`. Video may be resized proportionally in both modes, but they differ in that:

- In the `Fill` mode, the image fills the entire screen. If the dimensions of the image do not match those of the screen after scaling, the excess parts are cropped.
- In the `Fit` mode, the image is displayed in whole. If the dimensions of the image do not match those of the screen after scaling, the blank area is filled with black bars.

- C++
- C#

```
void TRTCMainViewController::onUserVideoAvailable(const char* userId, bool available){
    if (available) {
        // Get the handle of the rendering window
        CWnd *pRemoteVideoView = GetDlgItem(IDC_REMOTE_VIDEO_VIEW);
        HWND hwnd = pRemoteVideoView->GetSafeHwnd();

        // Set the rendering mode of the remote video
        m_pTRTCSDK->setRemoteViewFillMode(TRTCVideoFillMode_Fill);
        // Call the API below to play the remote video
        m_pTRTCSDK->startRemoteView(userId, hwnd);
    } else {
        m_pTRTCSDK->stopRemoteView(userId);
    }
}
```

Note :

- In the `TRTCAppSceneLIVE` mode, there is no limit on the number of users in the role of “audience” ( `TRTCRoleAudience` ) in a room.
- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## 8. Co-anchor

Both audience and anchors can call the `switchRole` API of `TRTCCloud` to switch their roles. The most common application for the API is co-anchoring: audience call this API to switch their role to “anchor” so as to interact with the room owner.

## 9. Exit the room

Call `exitRoom` to exit the room. Whether the live streaming has ended or not, the SDK will start a complex handshake process where it releases all resources used by the live streaming. The process finishes only after you receive the `onExitRoom` callback.

- C++
- C#



```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
    if(m_pTRTCSDK)
    {
        m_pTRTCSDK->exitRoom();
    }
}

....
void TRTCMainViewController::onExitRoom(int reason)
{
    // Exited room successfully. `reason` is a reserved parameter and is not used for the time being.
}
```

# Electron

Last updated : 2022-03-20 11:06:11

## Application Scenarios

TRTC supports four room entry modes. Video call ( `VideoCall` ) and audio call ( `VoiceCall` ) are the [call modes](#), and interactive video streaming ( `Live` ) and interactive audio streaming ( `VoiceChatRoom` ) are the live streaming modes.

The live streaming modes allow a maximum of 100,000 concurrent users in each room with smooth mic on/off. Co-anchoring latency is kept below 300 ms and watch latency below 1,000 ms. The live streaming modes are suitable for use cases such as low-latency interactive live streaming, interactive classrooms for up to 100,000 participants, video dating, online education, remote training, and mega-scale conferencing.

## How It Works

TRTC services use two types of server nodes: access servers and proxy servers.

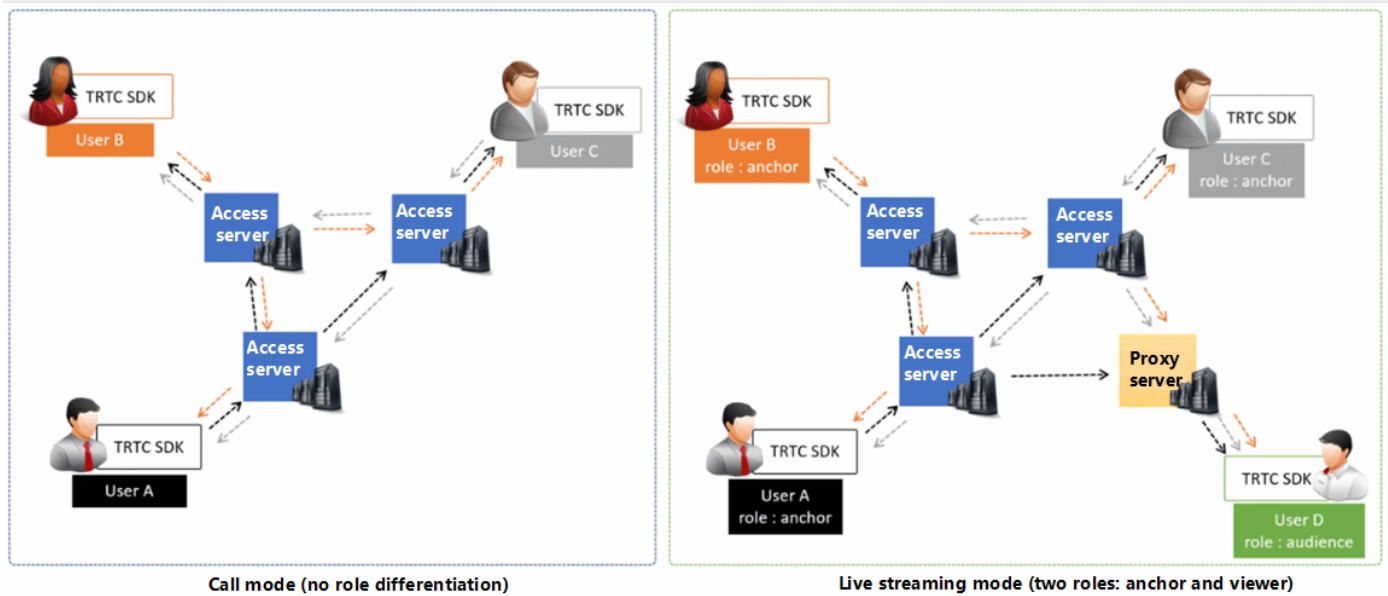
- **Access server**

This type of nodes use high-quality lines and high-performance servers and are better suited to drive low-latency end-to-end calls.

- **Proxy server**

This type of servers use mediocre lines and average-performance servers and are better suited to power high-concurrency stream pulling and playback.

In the call modes, all users in a TRTC room are assigned to access servers and are in the role of “anchor”. This means the users can speak to each other at any point during the call (up to 50 users can send data at the same time). This makes the call modes suitable for use cases such as online conferencing, but the number of users in each room is capped at 300.



## Sample Code

You can obtain the sample code used in this document at [GitHub](#).

## Directions

### Step 1. Run the official SimpleDemo

We recommend that you read [Demo Quick Start > Electron](#) first and follow the instructions to run the official SimpleDemo.

- If you run SimpleDemo successfully, then you know how to install Electron in your project.
- If not, there may be a problem in the download or installation process. Try troubleshooting the problem by following the instructions in Electron's [installation document](#).

### Step 2. Integrate trtc-electron-sdk into your project

If you can [run SimpleDemo](#) successfully, then you know how to set up the Electron environment.

- You can develop your project based on the demo we provide to get started quickly.
- You can also run the following command to install trtc-electron-sdk in your project.

```
npm install trtc-electron-sdk --save
```

### Step 3. Initialize an SDK instance and configure event callbacks

Create a `trtc-electron-sdk` instance:

```
import TRTCCloud from 'trtc-electron-sdk';  
let trtcCloud = new TRTCCloud();
```

Listen for the `onError` event:

```
// Error events must be listened for and captured, and error messages should be sent to users.  
let onError = function(err) {  
  console.error(err);  
}  
trtcCloud.on('onError', onError);
```

### Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom()` API, you need to pass in a key parameter `TRTCParams`, which includes the following required fields:

Parameter	Field Type	Description	Example
<code>sdkAppId</code>	Number	Application ID, which can be found in <b>Application Management &gt; Application Info</b> in the <a href="#">console</a>	1400000123
<code>userId</code>	String	Can contain only letters (a-z and A-Z), digits (0-9), underscores, and hyphens. We recommend you set it based on your business account system.	test_user_001
<code>userSig</code>	String	<code>userSig</code> is calculated based on <code>userId</code> . For the calculation method, see <a href="#">UserSig</a> .	eJyrVareCeYrSy1Ssll...
<code>roomId</code>	Number	Numeric room ID. For string-type room ID, use <code>strRoomId</code> in <code>TRTCParams</code> .	29834

```
import {  
  TRTCParams,  
  TRTCRoleType  
} from "trtc-electron-sdk/liteav/trtc_define";  
  
let param = new TRTCParams();  
param.sdkAppId = 1400000123;
```

```
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SsII...';
param.role = TRTCRoleType.TRTCRoleAnchor; // Set the role to "anchor"
```

Note :

-In TRTC, users with the same `userId` cannot be in the same room at the same time as it will cause a conflict.

- The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.

## Step 5. Enable camera preview and mic capturing

1. Call `startLocalPreview()` to enable preview of the local camera. The SDK will ask for camera permission.
2. Call `setLocalViewFillMode()` to set the display mode of the local video image.
  - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` : aspect fill. The image may be scaled up and cropped, but there are no black bars.
  - `TRTCVideoFillMode.TRTCVideoFillMode_Fit` : aspect fit. The image may be scaled down to ensure that it's displayed in its entirety, and there may be black bars.
3. Call `setVideoEncoderParam()` to set the encoding parameters of the local video, which determine the [quality of your video](#) seen by other users in the room.
4. Call `startLocalAudio()` to turn the mic on. The SDK will ask for mic permission.

```
// Sample code: publish the local audio/video stream
trtcCloud.startLocalPreview(view);
trtcCloud.startLocalAudio();
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);

// Set local video encoding parameters
let encParam = new TRTCVideoEncParam();
encParam.videoResolution = TRTCVideoResolution.TRTCVideoResolution_640_360;
encParam.resMode = TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape;
encParam.videoFps = 25;
encParam.videoBitrate = 600;
encParam.enableAdjustRes = true;
trtcCloud.setVideoEncoderParam(encParam);
```

## Step 6. Set beauty filters

1. Call `setBeautyStyle(style, beauty, white, ruddiness)` to enable filters.
2. Parameter description:
  - `style` : style, which may be smooth or natural. The smooth style features more obvious skin smoothing effect and is suitable for entertainment scenarios.
    - `TRTCBeautyStyle.TRTCBeautyStyleSmooth` : smooth, which features more obvious skin smoothing effect and is suitable for shows
    - `TRTCBeautyStyle.TRTCBeautyStyleNature` : natural, which retains more facial details and is more natural
  - `beauty` : strength of the beauty filter. Value range: 0-9. `0` indicates that the filter is disabled. The larger the value, the more obvious the effect.
  - `white` : strength of the skin brightening filter. Value range: 0-9. `0` indicates that the filter is disabled. The larger the value, the more obvious the effect.
  - `ruddiness` : strength of the rosy skin filter. Value range: 0-9. `0` indicates that the filter is disabled. The larger the value, the more obvious the effect. This parameter is unavailable on Windows currently.

```
// Enable beauty filters
trtcCloud.setBeautyStyle(TRTCBeautyStyle.TRTCBeautyStyleNature, 5, 5, 5);
```

## Step 7. Create a room and push streams

1. If `role` in `TRTCParams` is set to `TRTCRoleType.TRTCRoleAnchor`, the current user is in the role of an anchor.
2. Call `enterRoom()`, specifying the `appScene` parameter, and a room whose ID is the value of the `roomId` field in `TRTCParams` will be created.
  - `TRTCAppScene.TRTCAppSceneLIVE` : the interactive video streaming mode, which features smooth mic on/off and anchor latency below 300 ms. Up to 100,000 users can play the anchor's video at the same time, with playback latency as low as 1,000 ms. The example in this document uses this mode.
  - `TRTCAppScene.TRTCAppSceneVoiceChatRoom` : the interactive audio streaming mode, which features smooth mic on/off and anchor latency below 300 ms. Up to 100,000 users can play the anchor's audio at the same time, with playback latency as low as 1,000 ms.
  - For more information about `TRTCAppScene`, see [TRTCAppScene](#).
3. After the room is created, start encoding and transferring audio/video data. The SDK will return the `onEnterRoom(result)` callback. If `result` is greater than 0, room entry succeeds, and the value indicates the time (ms) room entry takes; if `result` is less than 0, room entry fails, and the value is the error code for the failure.

```

let onEnterRoom = function (result) {
  if (result > 0) {
    console.log(`onEnterRoom, room entry succeeded and took ${result} seconds`);
  } else {
    console.warn(`onEnterRoom: failed to enter room ${result}`);
  }
};

trtcCloud.on('onEnterRoom', onEnterRoom);

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SsII...';
param.role = TRTCRoleType. TRTCRoleAnchor;
trtcCloud.enterRoom(param, TRTCAppScene. TRTCAppSceneLIVE);

```

## Step 8. Enter the room as audience

1. Set the `role` field in `TRTCParams` to `TRTCRoleType. TRTCRoleAudience` to take the role of "audience".
2. Call `enterRoom()` to enter the room whose ID is the value of the `roomId` field in `TRTCParams` , specifying `appScene` :
  - `TRTCAppScene. TRTCAppSceneLIVE` : interactive video streaming
  - `TRTCAppScene. TRTCAppSceneVoiceChatRoom` : interactive audio streaming
3. Watch the anchor's video:
  - If you know the anchor's `userId` , call `startRemoteView(userId, view)` with the anchor's `userId` passed in to play the anchor's video.
  - If you do not know the anchor's `userId` , find the anchor's `userId` in the `onUserVideoAvailable()` callback, which you will receive after room entry, and call `startRemoteView(userId, view)` with the anchor's `userId` passed in to play the anchor's video.

```

<div id="video-container"></div>
<script>
const videoContainer = document.querySelector('#video-container');
const roomId = 29834;
// Callback for room entry
let onEnterRoom = function(result) {
  if (result > 0) {
    console.log(`onEnterRoom, room entry succeeded and took ${result} seconds`);
  } else {

```

```

console.warn(`onEnterRoom: failed to enter room ${result}`);
}
};

// This callback is triggered when the anchor publishes/unpublishes streams from the camera.
let onUserVideoAvailable = function(userId, available) {
  if (available === 1) {
    let id = `${userId}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
    let view = document.getElementById(id);
    if (!view) {
      view = document.createElement('div');
      view.id = id;
      videoContainer.appendChild(view);
    }
    trtcCloud.startRemoteView(userId, view);
    trtcCloud.setRemoteViewFillMode(userId, TRTCVideoFillMode.TRTCVideoFillMode_Fill);
  } else {
    let id = `${userId}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
    let view = document.getElementById(id);
    if (view) {
      videoContainer.removeChild(view);
    }
  }
};

trtcCloud.on('onEnterRoom', onEnterRoom);
trtcCloud.on('onUserVideoAvailable', onUserVideoAvailable);

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = roomId;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSyISsLI...';
param.role = TRTCRoleType.TRTCRoleAudience; // Set the role to "audience"
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
</script>

```

## Step 9. Co-anchor

1. Call `switchRole(TRTCRoleType.TRTCRoleAnchor)` to switch the role to "anchor" (`TRTCRoleType.TRTCRoleAnchor`).
2. Call `startLocalPreview()` to enable local camera preview.
3. Call `startLocalAudio()` to enable mic capturing.

```

// Sample code: start co-anchoring
trtcCloud.switchRole(TRTCRoleType.TRTCRoleAnchor);

```



```
trtcCloud.startLocalAudio();
trtcCloud.startLocalPreview(frontCamera, view);

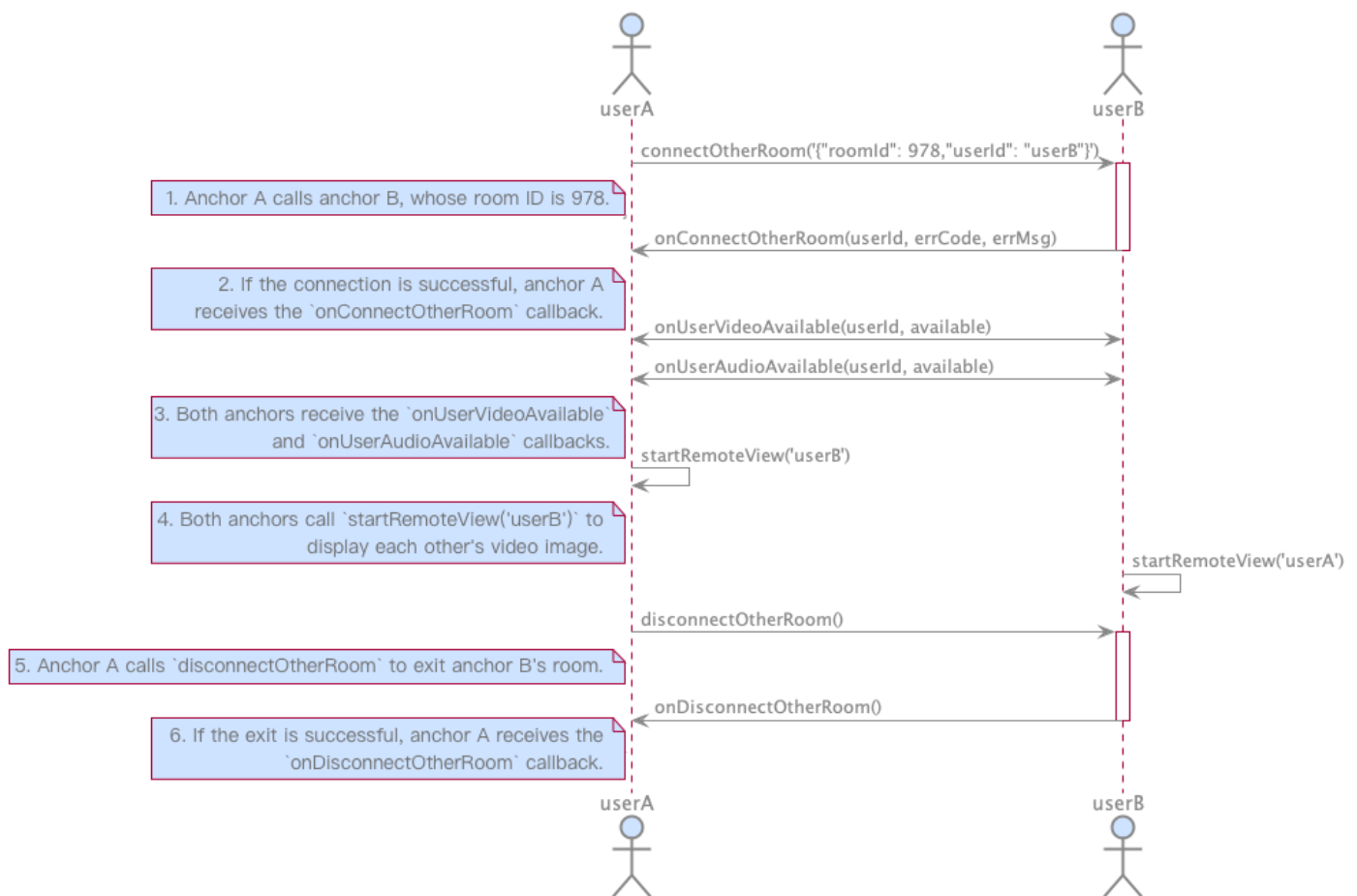
// Sample code: end co-anchoring
trtcCloud.switchRole(TRTCRoleType. TRTCRoleAudience);
trtcCloud.stopLocalAudio();
trtcCloud.stopLocalPreview()
```

## Step 10. Compete across rooms

Anchors from two rooms can compete with each other without exiting their current rooms.

1. Anchor A calls the [connectOtherRoom\(\)](#) API. The API uses parameters in JSON strings, so anchor A needs to pass the `roomId` and `userId` of anchor B in the format of `{"roomId": 978,"userId": "userB"}` to the API.
2. After the cross-room call is set up, anchor A will receive the [onConnectOtherRoom\(userId, errCode, errMsg\)](#) callback, and all users in both rooms will receive the [onUserVideoAvailable\(\)](#) and [onUserAudioAvailable\(\)](#) callbacks.  
For example, after anchor A in room "001" uses `connectOtherRoom()` to call anchor B in room "002" successfully, all users in room "001" will receive the `onUserVideoAvailable(B, true)` and `onUserAudioAvailable(B, true)` callbacks, and all users in room "002" will receive the `onUserVideoAvailable(A, true)` and `onUserAudioAvailable(A, true)` callbacks.
3. Users in both rooms can call [startRemoteView\(userId, view\)](#) to play the video of the anchor in the other room. Audio will be played automatically.

Event Timeline for Co-anchoring



```

// Sample code: cross-room competition
let onConnectOtherRoom = function(userId, errCode, errMsg) {
  if(errCode === 0) {
    console.log(`Connected to the room of anchor ${userId}`);
  } else {
    console.warn(`Failed to connect to the anchor's room: ${errMsg}`);
  }
};

const paramJson = '{"roomId": "978","userId": "userB"}';
trtcCloud.connectOtherRoom(paramJson);
trtcCloud.on('onConnectOtherRoom', onConnectOtherRoom);
  
```

## Step 11. Exit the room

Call `exitRoom()` to exit the room. The SDK disables and releases devices such as cameras and mics during room exit. Therefore, room exit is not an instant process. It completes only after the `onExitRoom()` callback is received.

```
// Please wait for the `onExitRoom` callback after calling the room exit API.  
let onExitRoom = function (reason) {  
  console.log(`onExitRoom, reason: ${reason}`);  
};  
trtcCloud.exitRoom();  
trtcCloud.on('onExitRoom', onExitRoom);
```

Note :

If your Electron application integrates multiple audio/video SDKs, please wait after you receive the `onExitRoom` callback to start other SDKs; otherwise the device busy error may occur.

# Web

Last updated : 2022-03-10 09:48:45

This document describes how to enter a room as audience and start co-anchoring. The process of entering a room and publishing streams as an anchor in live streaming scenarios is the same as that in call scenarios. Please see [Real-Time Audio/Video Call](#).

## Example

You can click [Demo](#) to try out the audio/video features, or log in to [GitHub](#) to get the sample code related to this document.

## Step 1. Create a client object

Create a [client](#) object using [TRTC.createClient\(\)](#). Set the parameters as follows:

- `mode` : TRTC mode, which should be set to `live`
- `sdkAppId` : the `sdkAppId` you obtain from Tencent Cloud
- `userId` : user ID
- `userSig` : user signature

```
const client = TRTC.createClient({
  mode: 'live',
  sdkAppId,
  userId,
  userSig
});
```

## Step 2. Enter a room as audience

Call [Client.join\(\)](#) to enter a TRTC room. Below are the request parameters:

- `roomId` : room ID
- `role` : role
  - `anchor` (default): users in the role of “anchor” can publish local streams and play remote streams.

- `audience` . Users in the role of “audience” can play remote streams but cannot publish local streams. To co-anchor and publish local streams, audience must switch the role to `anchor` using `Client.switchRole()`.

```
// Enter a room as audience
client
.join({ roomId, role: 'audience' })
.then(() => {
  console.log('Entered room successfully');
})
.catch(error => {
  console.error('Failed to enter room' + error);
});
```

## Step 3. Play a live stream

1. After receiving `client.on('stream-added')` , which is used to listen for remote streams, call `Client.subscribe()` to subscribe to the remote stream.

Note :

To ensure that you are notified when a remote user enters the room, please subscribe to the `client.on('stream-added')` callback before you call `Client.join()` to enter the room.

```
client.on('stream-added', event => {
  const remoteStream = event.stream;
  console.log('New remote stream:' + remoteStream.getId());
  // Subscribe to the remote stream
  client.subscribe(remoteStream);
});
client.on('stream-subscribed', event => {
  const remoteStream = event.stream;
  console.log('Subscribed to remote stream successfully:' + remoteStream.getId());
  // Play the remote stream
  remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

2. In the callback that indicates successful subscription to a remote stream, call `[Stream.play()]` (<https://web.sdk.qcloud.com/trtc/webtrtc/doc/zh-cn/Stream.html#play>) to play the stream on a

webpage. The `play` method allows a parameter that is a div element ID. The SDK will create an audio/video tag in the div element and play the stream on it.

```
client.on('stream-subscribed', event => {
  const remoteStream = event.stream;
  console.log('Subscribed to remote stream successfully:' + remoteStream.getId());
  // Play the remote stream
  remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

## Step 4. Co-anchor

### Step 4.1. Switch roles

Call `Client.switchRole()` to switch the role to `anchor`.

```
client
  .switchRole('anchor')
  .then(() => {
    // Role switched to "anchor" successfully
  })
  .catch(error => {
    console.error('Failed to switch role' + error);
  });
```

### Step 4.2. Co-anchor

1. Call `TRTC.createStream()` to create a local audio/video stream. In the example below, the audio/video stream is captured by the camera and mic. The parameters include:

- `userId` : ID of the user to whom the local stream belongs
- `audio` : whether to enable audio
- `video` : whether to enable video

```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
```

2. Call `LocalStream.initialize()` to initialize the local audio/video stream.

```
localStream
  .initialize()
  .then(() => {
    console.log('Local stream initialized successfully');
  })
  .catch(error => {
    console.error('Failed to initialize local stream' + error);
  });
```

### 3. Play the local stream after it is initialized

```
localStream
  .initialize()
  .then(() => {
    console.log('Local stream initialized successfully');
    localStream.play('local_stream');
  })
  .catch(error => {
    console.error('Failed to initialize local stream' + error);
  });
```

### 4. Call `Client.publish()` to publish the local stream and start co-anchoring.

```
client
  .publish(localStream)
  .then(() => {
    console.log('Local stream published successfully');
  })
  .catch(error => {
    console.error('Failed to publish local stream' + error);
  });
```

## Step 5. Exit the room

Call `Client.leave()` to exit the room. The live streaming session ends.

```
client
  .leave()
  .then(() => {
    // Exited room successfully
  })
  .catch(error => {
    console.error('Failed to leave room' + error);
  });
```

Note :

The value of `appScene` must be the same on each client. Inconsistent `appScene` may cause unexpected problems.



# On-Cloud Recording

Last updated : 2022-03-14 11:36:54

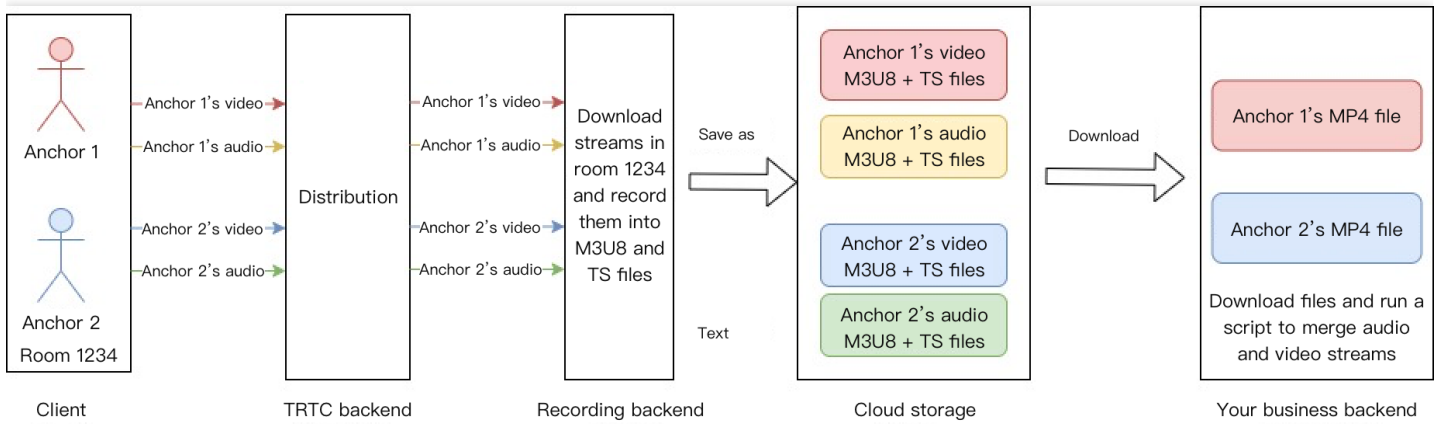
In applications such as online education, live showroom, video conferencing, online medical consultation, and remote banking, it is often necessary to record entire video calls or live streaming sessions for purposes including content moderation, archiving, and playback. The on-cloud recording feature of TRTC can help meet these demands.

## Overview

The on-cloud recording feature of TRTC allows you to record an audio/video stream in real time using a RESTful API. It is flexible, light, and easy-to-use, saving you the trouble of deploying servers and recording modules.

- **Recording mode:** Single-stream recording records the audio and video of each user in a room separately, while mixed-stream recording records all audios and videos in a room into one result.
- **Stream subscription:** You can determine whose streams you receive or do not receive using an allowlist/blocklist.
- **Transcoding parameters:** In the mixed-stream recording mode, you can determine the output video quality by specifying transcoding parameters.
- **Stream-mixing parameters:** For mixed-stream recording, we offer multiple auto-arranged layout templates. You can also customize a layout template.
- **File storage:** Currently, you can save recording files only in COS or VOD of Tencent Cloud. (We plan to add support for storage and video-on-demand services of third-party cloud vendors in the future. To save files to third-party platforms, you will need to provide your cloud service account and the storage parameters.)
- **Callback notification:** By configuring a callback domain in the console, you can receive notifications about on-cloud recording events via your callback server.

## Single-stream recording

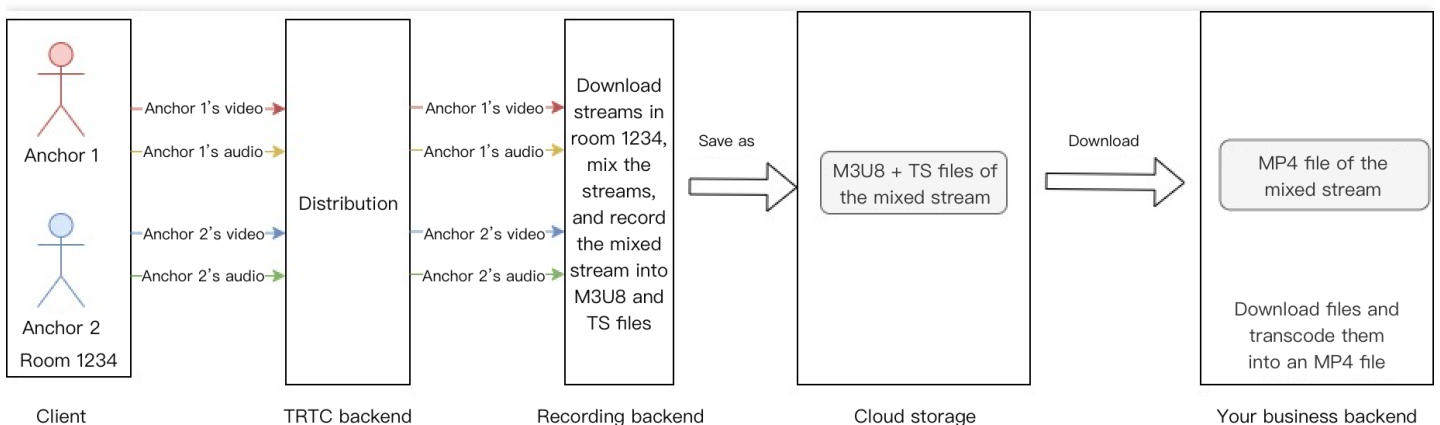


The diagram above shows the workflow of single-stream recording. In room 1234, anchor 1 and anchor 2 are publishing streams. If you subscribe to their streams and enable single-stream recording, the TRTC backend will record the audio and video data of anchor 1 and anchor 2 separately. The recording results will include:

1. An M3U8 index file of anchor 1's video
2. Multiple TS segment files of anchor 1's video
3. An M3U8 index file of anchor 1's audio
4. Multiple TS segment files of anchor 1's audio
5. An M3U8 index file of anchor 2's video
6. Multiple TS segment files of anchor 2's video
7. An M3U8 index file of anchor 2's audio
8. Multiple TS segment files of anchor 2's audio

The backend will then upload the files to the cloud storage server you specify. You need to download the files and merge/transcode them. We offer a [script for merging audio and video streams](#).

## Mixed-stream recording



The above shows the workflow of mixed-stream recording. In room 1234, anchor 1 and anchor 2 are publishing streams. If you subscribe to their streams and enable mixed-stream recording, the TRTC backend will mix the streams of anchor 1 and anchor 2 according to the layout template you specify and then record them into one result, which will include:

1. An M3U8 index file of the mixed video
2. Multiple TS segment files of the mixed video

The backend will then upload the files to the cloud storage server you specify. You need to download the files and merge/transcode them. We offer a [script for merging audio and video streams](#).

Note :

- The rate limit for the recording API is 20 queries per second.
- The timeout period for a query is 6 seconds.
- We allow up to 100 ongoing recording tasks at the same time. If you need to record more, please [submit a ticket](#).
- In the single-stream recording mode, you can record up to 25 streams in a room at the same time.

## Directions

### 1. Start recording

Call the RESTful API `CreateCloudRecording` from your server to start on-cloud recording. Pay attention to the following parameters:

#### TaskId

This parameter uniquely identifies a recording task. Note it as you will need to provide it for other actions on the same task later.

#### RecordMode

- Single-stream recording separately records the audios and videos of individual anchors whose streams you receive before uploading the results (including M3U8 and TS segment files) to the cloud.
- Mixed-stream recording records all the audios and videos of anchors whose streams you receive into one result (including M3U8 and TS segment files) before uploading it to the cloud.

## SubscribeStreamUserIds

By default, on-cloud recording records all the streams (max 25) you receive in a room. You can use this parameter to specify whose streams you want to record and can change its value during recording.

## StorageParams

You can use this parameter to specify the cloud storage/video-on-demand service you want to save recording files to. Make sure you use a valid value and that the cloud storage/video-on-demand service you use is available. Below are the naming conventions of recording files:

### Naming of recording files

- M3U8 file in the single-stream recording mode:  
`<prefix>/<taskid>/<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>.m3u8`
- TS segment file in the single-stream recording mode:  
`<prefix>/<taskid>/<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>_<utc>.ts`
- MP4 file in the single-stream recording mode:  
`<prefix>/<taskid>/<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<index>.mp4`
- M3U8 file in the mixed-stream recording mode:  
`<prefix>/<taskid>/<sdkappid>_<roomid>.m3u8`
- TS segment file in the mixed-stream recording mode:  
`<prefix>/<taskid>/<sdkappid>_<roomid>_<utc>.ts`
- MP4 file in the mixed-stream recording mode:  
`<prefix>/<taskid>/<sdkappid>_<roomid>_<index>.mp4`

- Naming of recovered files

The on-cloud recording feature has a high availability scheme that can recover recording files if the server fails. To prevent the recovered files from replacing the original files, we add a prefix `ha<1/2/3>` to the names of recovered files. The numbers indicate the times (max 3) the high availability scheme is used.

- M3U8 file in the single-stream recording mode:  
`<prefix>/<taskid>/ha&lt;1/2/3>_<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>.m3u8`
- TS segment file in the single-stream recording mode:  
`<prefix>/<taskid>/ha&lt;1/2/3>_<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>_<utc>.ts`
- M3U8 file in the mixed-stream recording mode:  
`<prefix>/<taskid>/ha&lt;1/2/3>_<sdkappid>_<roomid>.m3u8`
- TS segment file in the mixed-stream recording mode:  
`<prefix>/<taskid>/ha&lt;1/2/3>_<sdkappid>_<roomid>_<utc>.ts`

**Field description**

`<prefix>`: filename prefix, which is not used if not specified

`<taskid>`: task ID, which is unique and is returned by the start recording API

`<sdkappid>`: application ID

`<roomid>`: room ID

`<userid>`: Base64-encoded ID of a user whose stream is recorded

`<mediaid>`: indicates whether the primary stream ( `main` ) or substream ( `aux` ) is recorded

`<type>`: the type of stream that is recorded (audio or video)

`<utc>`: recording start time (UTC+0) (year, month, day, hour, minute, second, millisecond)

`<index>`: index of a segment, which starts from 1. This field is used only if the size of an MP4 file reaches 2 GB or its length reaches 24 hours and needs to be segmented.

`ha&lt;1/2/3>`: prefix for a file recovered by the high availability scheme. For example, if the scheme is used for the first time, the recovered file is named

`<prefix>/<taskid>/ha1_<sdkappid>_<roomid>.m3u8`.

**Note :**

If `<roomid>` is a string, it will be encoded into Base64. In the result, `/` is replaced with `-` and `=` is replaced with `.`

`<userid>` is encoded into Base64. In the result, `/` is replaced with `-` and `=` is replaced with `.`

**Recording start time**

Recording starts when you start receiving data from an anchor. Recording start time is the Unix time on the server when recording starts.

You can query the start time of a recording task in three ways:

- Using the `DescribeCloudRecording` API. `BeginTimeStamp` in the response parameters indicates the recording start time (ms).

Below is a response of the `DescribeCloudRecording` API, in which `BeginTimeStamp` is `1622186279144`.

```
{
  "Response": {
    "Status": "xx",
    "StorageFileList": [
      {
        "TrackType": "xx",
        "BeginTimeStamp": 1622186279144,
        "UserId": "xx",
        "FileName": "xx"
      }
    ],
    "RequestId": "xx",
    "TaskId": "xx"
  }
}
```

- Viewing the M3U8 file (the `#EXT-X-TRTC-START-REC-TIME` directive)

According to the M3U8 file below, the Unix time (ms) when recording started was `1622425551884`.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:70
#EXT-X-TRTC-START-REC-TIME:1622425551884
#EXT-X-TRTC-VIDEO-METADATA:WIDTH:1920 HEIGHT:1080
#EXTINF:12.074
1400123456_12345_UserId_s_MTY4NjExOQ.._UserId_e_main_video_20330531094551825.ts
#EXTINF:11.901
1400123456_12345_UserId_s_MTY4NjExOQ.._UserId_e_main_video_20330531094603825.ts
#EXTINF:12.076
1400123456_12345_UserId_s_MTY4NjExOQ.._UserId_e_main_video_20330531094615764.ts
#EXT-X-ENDLIST
```

- Via a recording callback

If you have registered recording callbacks, you will receive a callback for the generation of recording files (event type 307), in which `BeginTimeStamp` indicates the recording start time.

According to the callback below, the Unix time (ms) when recording started was 1622186279144.

```
{
  "EventGroupId": 3,
  "EventType": 307,
  "CallbackTs": 1622186289148,
  "EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186289",
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "FileName": "xx.m3u8",
      "UserId": "xx",
      "TrackType": "audio",
      "BeginTimeStamp": 1622186279144
    }
  }
}
```

## MixWatermark

TRTC allows you to watermark videos during mixed-stream recording. You can add up to 25 marks to a video in your desired positions.

Field	Description
Top	Vertical offset of the watermark from the top-left corner of the video
Left	Horizontal offset of the watermark from the top-left of the video
Width	Watermark width
Height	Watermark height
url	URL of the watermark file

## MixLayoutMode

TRTC supports the grid layout (default), floating layout, screen sharing layout, and custom layout.

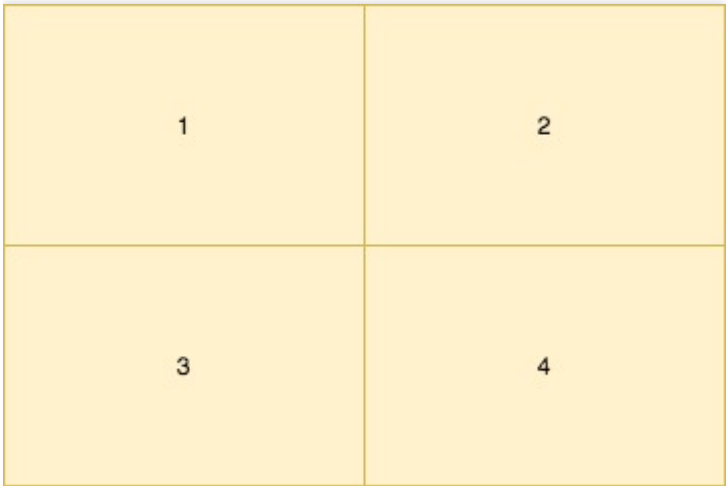
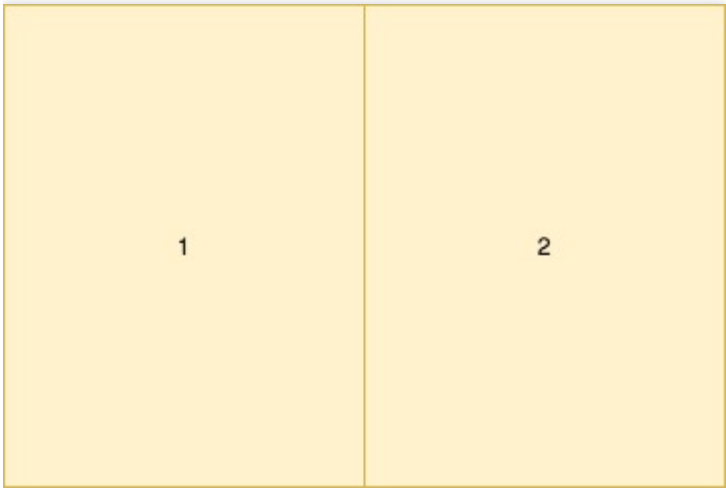
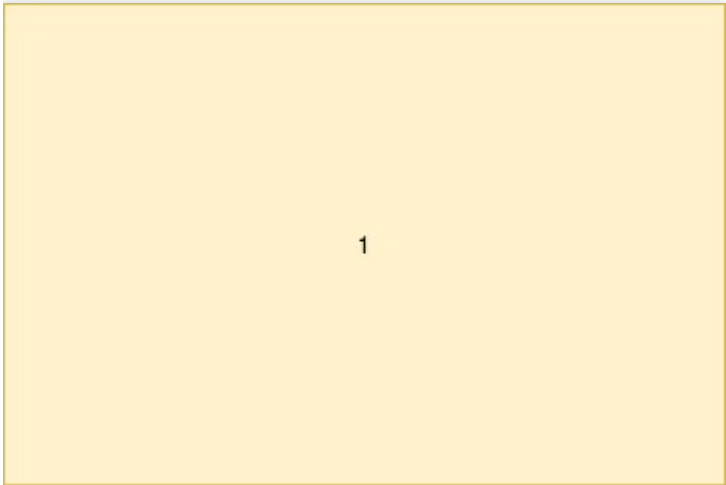
### Grid layout

The videos of anchors are scaled and positioned automatically according to the total number of anchors in a room. Each video has the same size. Up to 25 videos can be displayed.

- When there is only one video:  
The video is scaled to fill the canvas.
- When there are two videos:  
The width of each video is half of the canvas width.  
The height of each video is the same as the canvas height.
- When there are three or four videos:  
The canvas is split evenly into four windows and each video is displayed in one window.
- When the number of videos is between four and nine:  
The canvas is split evenly into nine windows and each video is displayed in one window.
- When the number of videos is between 10 and 16:  
The canvas is split evenly into 16 windows and each video is displayed in one window.
- When there are more than 16 videos:  
The canvas is split evenly into 25 windows and each video is displayed in one window.

As shown below:





1	2	3
4	5	6
7	8	9

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

### Floating layout

By default, the video of the first anchor (you can also specify an anchor) who enters the room is scaled to fill the screen. When other anchors enter the room, their videos appear smaller and are

superimposed over the large video from left to right starting from the bottom of the canvas according to their room entry sequence. If the total number of videos is 17 or less, there will be four windows in each row ( $4 \times 4$ ); if it is greater than 17, there will be five windows in each row ( $5 \times 5$ ). Up to 25 videos can be displayed. A user who publishes only audio will still be displayed in one window.

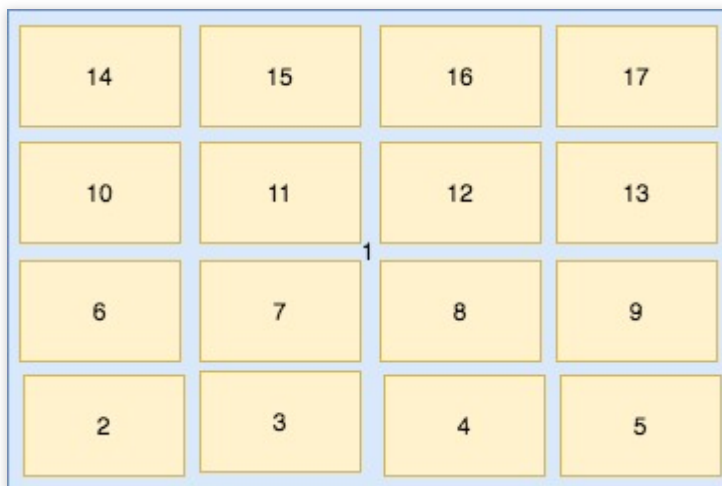
- When there are 17 or fewer videos:

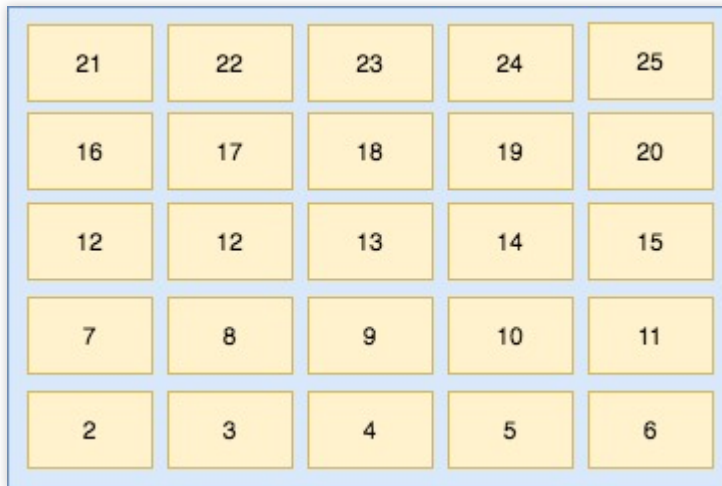
The **width** and **height** of each **video** are 23.5% of the **canvas width** and **height**.  
The horizontal space and vertical space between two neighboring videos are 1.2% of the **canvas width** and **height**.  
The **right/left** and **top/bottom margin** are 1.2% of the **canvas width** and **height**.

- When there are more than 17 videos:

The **width** and **height** of each **video** are 18.8% of the **canvas width** and **height**.  
The horizontal space and vertical space between two neighboring videos are 1% of the **canvas width** and **height**.  
The **right/left** and **top/bottom margin** are 1% of the **canvas width** and **height**.

As shown below:





### Screen sharing layout

The video of a specified anchor occupies a larger part of the canvas on the left side (if you do not specify an anchor, the left window will display the canvas background). The videos of other anchors are smaller and are positioned on the right side. If the total number of videos is 17 or less, the small videos are positioned from top to bottom in up to two columns on the right side, with eight videos per column at most. If there are more than 17 videos, the additional videos are positioned at the bottom of the canvas from left to right. Up to 24 videos can be displayed. A user who publishes only audio will still be displayed in one window.

- When there are five or fewer videos:  
The size of each small video on the right is  $\frac{1}{5}$  the canvas width and  $\frac{1}{4}$  the canvas height.  
The width of the large video on the left is  $\frac{4}{5}$  the canvas width, and its height is the same as the canvas height.
- When the number of videos is between 5 and 7:  
The size of each small video on the right are  $\frac{1}{7}$  the canvas width and  $\frac{1}{6}$  the canvas height.  
The width of the large video on the left is  $\frac{6}{7}$  the canvas width, and its height is the same as the canvas height.
- When there are eight or nine videos:  
The size of each small video on the right is  $\frac{1}{9}$  the canvas width and  $\frac{1}{8}$  the canvas height.  
The width of the large video on the left is  $\frac{8}{9}$  the canvas width, and its height is the same as the canvas height.
- When the number of videos is between 10 and 17:  
The size of each small video on the right side is  $\frac{1}{10}$  the canvas width and  $\frac{1}{8}$  the canvas height.

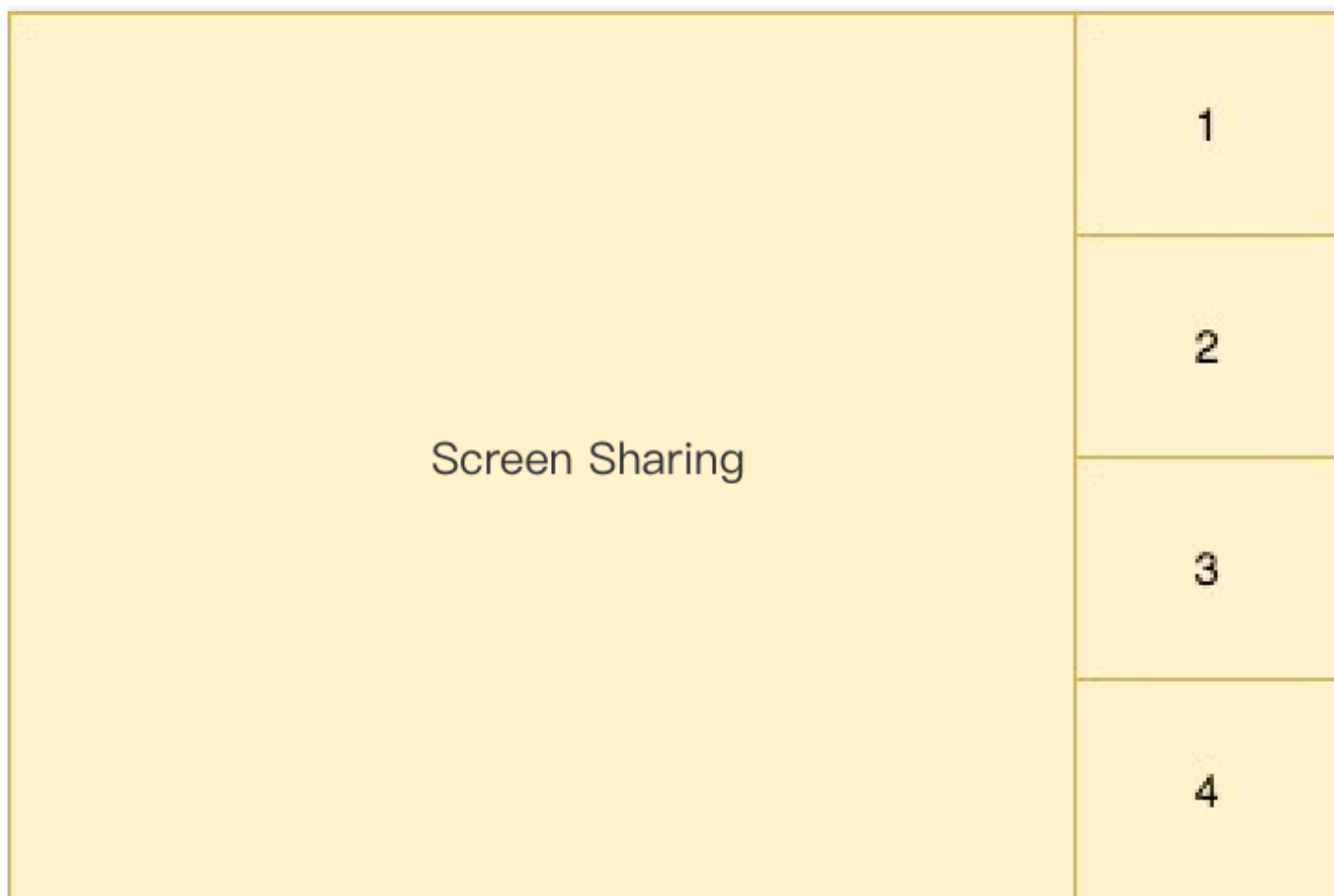
The width of the large video on the left side is  $\frac{4}{5}$  the canvas width, and its height is the same as the canvas height.

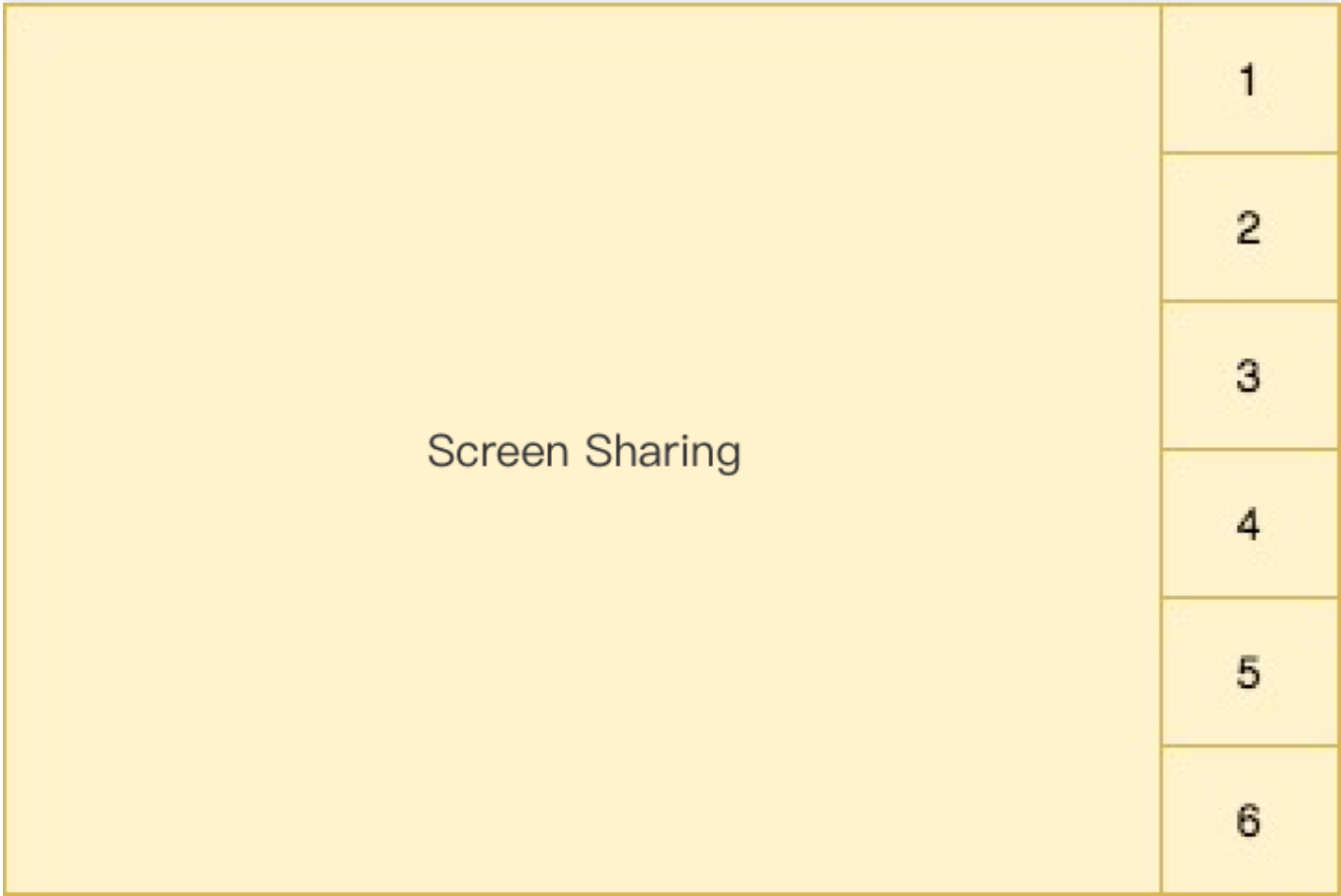
- When there are more than 17 videos:

The size of each small video on the right and bottom is  $\frac{1}{10}$  the canvas width and  $\frac{1}{8}$  the canvas height.

The width of the large video on the left is  $\frac{4}{5}$  the canvas width, and its height is  $\frac{7}{8}$  the canvas height.

As shown below:





Screen Sharing	1
	2
	3
	4
	5
	6
	7
	8

Screen Sharing	1	9
	2	10
	3	11
	4	12
	5	13
	6	14
	7	15
	8	16



Screen Sharing								1	9
								2	10
								3	11
								4	12
								5	13
								6	14
								7	15
17	18	19	20	21	22	23	24	8	16

### Custom layout

You can also use `MixLayoutList` to customize a layout for anchor videos.

## 2. Query the recording task

You can use the `DescribeCloudRecording` API to query the status of an ongoing recording task. If the queried task has already ended, an error will be returned.

The file list ( `StorageFile` ) that is returned will include all the M3U8 files of the recording as well as the Unix time when recording started. If the task queried is a recording to VOD task, the `StorageFile` returned will be empty.

## 3. Modify recording parameters

You can use the `ModifyCloudRecording` API to modify recording parameters, including `SubscribeStreamUserIds` and `MixLayoutParams` (valid only for mixed-stream recording). Note that to modify parameters, you need to respecify all the parameters, including `MixLayoutParams` and `SubscribeStreamUserIds` , and not just the one you want to modify. As a result, we recommend you note all the values before modifying a parameter, or you will need to calculate them again.

## 4. Stop recording

You can call the `DeleteCloudRecording` API to stop recording. A recording task will also end automatically if there are no anchors in a room for longer than the specified time period ( `MaxIdleTime` ). We recommend that you call the API to stop recording when you no longer need the service.

# Advanced Features

## Recording in MP4 format

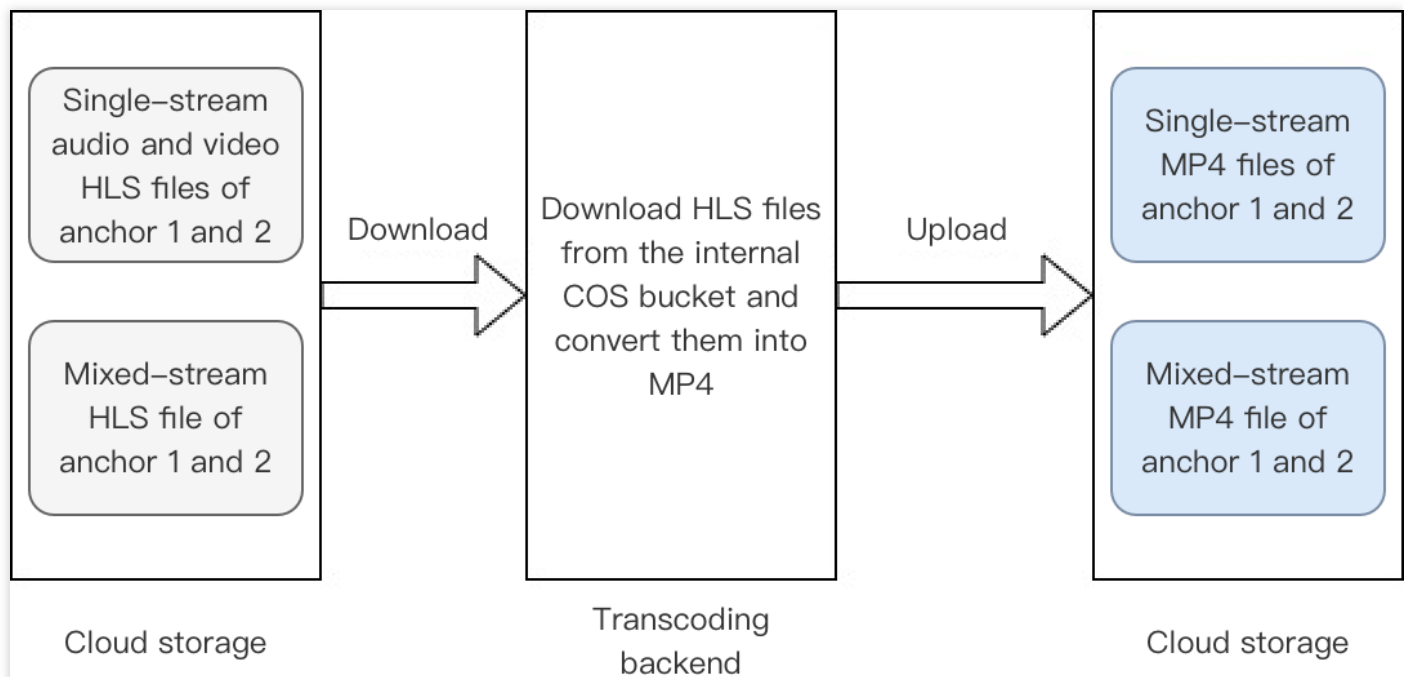
To record in MP4 format, set `OutputFormat` to `hls+mp4` when calling the `CreateCloudRecording` API. TRTC will still record in HLS format, but once recording ends, it will download the HLS file from its COS bucket, convert it into MP4 format, and upload the MP4 file to the COS bucket.

Please note that COS download access is required for the above to work.

An MP4 file will be segmented if one of the following conditions is met.

1. The recording lasts for 24 hours or longer.
2. The MP4 file is 2 GB or larger.

The figure below shows the workflow of recording in MP4 format.

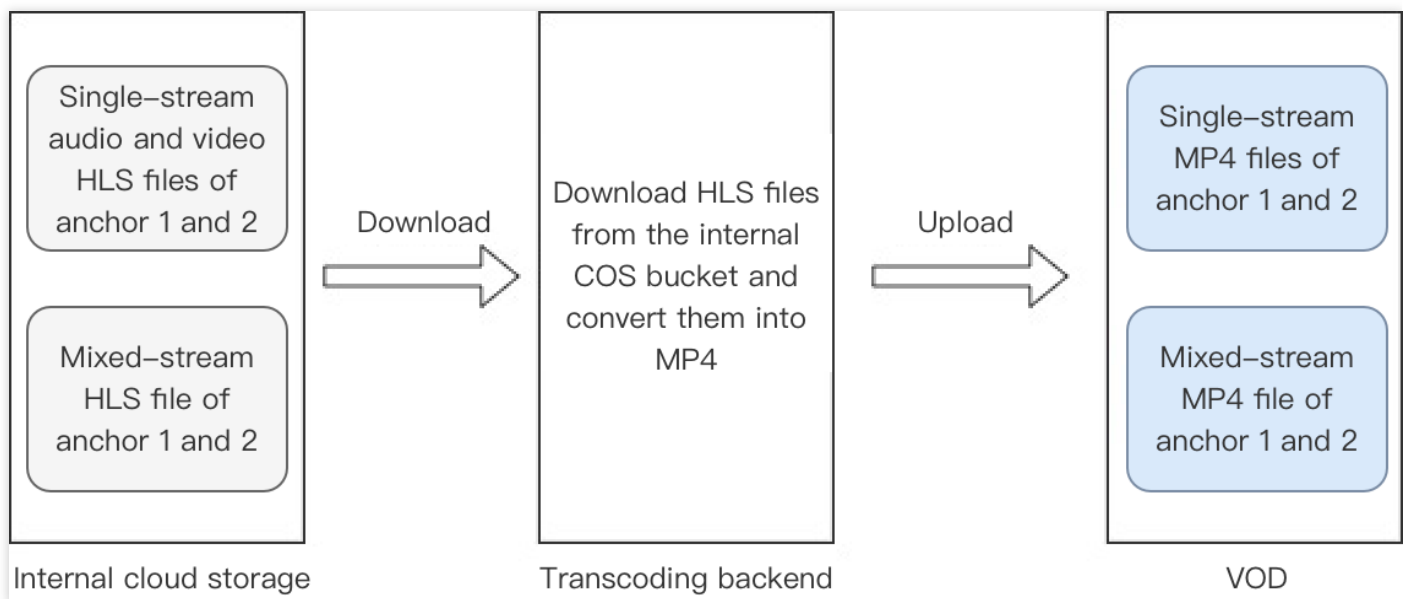


## Recording to VOD

To record to VOD, specify the `CloudVod` parameter in `StorageParams` when calling the `CreateCloudRecording` API. After recording ends, the backend will save the file in MP4 format to VOD using the method you specify and send you a playback URL via a callback. In the single-stream recording mode, there will be a playback URL for each anchor whose stream is recorded; in the mixed-stream recording mode, there will be only one playback URL. When you record to VOD, pay attention to the following:

1. `CloudVod` and `CloudStorage` are mutually exclusive. If you specify both, the recording task will fail to start.
2. If you use `DescribeCloudRecording` to query a recording to VOD task, the `StorageFile` returned will be empty.

The figure below shows the workflow of recording to VOD, in which "cloud storage" refers to the internal storage of the recording backend. You don't need to specify a COS bucket for it.



## Script for merging single-stream recording files

We offer a [script](#) for merging single-stream audio and video files into MP4 files.

### Note :

If two segment files are more than 15 seconds apart, during which no audio or video data is recorded (if the substream is disabled, its data will be ignored), the two segments will be considered to belong to different sections, one being the ending segment of the previous section, and the other the starting segment of the next section.

- Section-based merge ( `-m 0` )

In this mode, the recording files of each user ( `UserId` ) are merged by section. An MP4 file is generated for each section.

- User-based merge ( `-m 1` )

In this mode, the recording files of each user ( `UserId` ) are merged into one MP4 file. You can use the `-s` option to specify whether to fill in the blanks between sections.

## Environment requirements

### Python 3

- Centos: `sudo yum install python3`
- Ubuntu: `sudo apt-get install python3`

### Python 3 dependency

-sortedcontainers : `pip3 install sortedcontainers`

## Directions

1. Run the merge script: `python3 TRTC_Merge.py [option]`
2. An MP4 file will be generated in the directory of the recording files.  
E.g.: `python3 TRTC_Merge.py -f /xxx/file -m 0`

Below is a list of the options:

Option	Description
<code>-f</code>	Directory of the recording files. If there are multiple users ( <code>UserId</code> ), their recording files will be merged separately.
<code>-m</code>	<code>0</code> : section-based merge (default). In this mode, the recording files of each user ( <code>UserId</code> ) are merged by section. Multiple MP4 files may be generated for each user. <code>1</code> : user-based merge. In this mode, the recording files of each user ( <code>UserId</code> ) are merged into one MP4 file.
<code>-s</code>	Indicates whether to delete the blanks between sections in the user-based merge mode, in which case the files generated will be shorter than the recording duration.

Option	Description
-a	<p>0 : primary stream merge (default). The audio of a user ( <code>UserId</code> ) is merged with the user's primary stream, not the substream.</p> <p>1 : automatic merge. If a user ( <code>UserId</code> ) has a primary stream, the user's audio is merged with the primary stream; if not, it is merged with the substream.</p> <p>2 : substream merge. The audio of a user ( <code>UserId</code> ) is merged with the user's substream, not the primary stream.</p>
-p	Frame rate (fps) of the output video, which is 15 by default. Value range: 5-120. If you enter a value smaller than 5, 5 will be used; if you enter a value greater than 120, 120 will be used.
-r	Resolution of the output video. For example, <code>-r 640 360</code> indicates that the resolution of the output video is 640 × 360.

## File Naming

- Audio-video file: `UserId_timestamp_av.mp4`
- Audio-only file: `UserId_timestamp.m4a`

Note :

`UserId` is the Base64-encoded ID of a user whose stream is recorded. For details, see the naming of recording files. `timestamp` is the starting time of the first TS segment of a section.

## Callback APIs

You can register callbacks by providing an HTTP/HTTPS gateway to receive callbacks. When an on-cloud recording event occurs, the system will send a callback notification to your callback server.

### Callback format

Callbacks are sent to your server in the form of HTTP/HTTPS POST requests, which consist of the following parts:

Character encoding: UTF-8

Request: The request body is in JSON format.

Response: HTTP STATUS CODE = 200. The server ignores the content of the response packet. For protocol-friendliness, we recommend adding `JSON: {"code":0}``` to the response.

### Field description

The header of a callback contains the following fields.

Field	Description
Content-Type	application/json
Sign	Signature value
SdkAppId	SDK application ID

The body of a callback contains the following fields.

Field	Type	Description
EventGroupId	Number	Event group ID, which is 3 for on-cloud recording
EventType	Number	Event type
CallbackTs	Number	Unix time (ms) when the callback was sent to your server
EventInfo	JSON Object	Event information

#### Event types

Field	Type	Description
EVENT_TYPE_CLOUD_RECORDING_RECORDER_START	301	On-cloud recording - The recording module was started.
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	On-cloud recording - The recording module was stopped.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	On-cloud recording - The upload module was started.
EVENT_TYPE_CLOUD_RECORDING_FILE_INFO	304	On-cloud recording - The first M3U8 file was generated and uploaded successfully.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	On-cloud recording - The upload finished.

Field	Type	Description
EVENT_TYPE_CLOUD_RECORDING_FAILOVER	306	On-cloud recording - The recording task was migrated.
EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE	307	On-cloud recording - The first TS segment was available and an M3U8 file was generated.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR	308	On-cloud recording - The upload module encountered an error.
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	On-cloud recording - An error occurred during the download of the image decoding file.
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	On-cloud recording - The task of recording in MP4 format ended and the name of the MP4 file generated was returned.
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	On-cloud recording - Upload was completed for the recording to VOD task.
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	On-cloud recording - The recording to VOD task ended.

## Event information

Field	Type	Description
RoomId	String/Number	Room ID (same type as Room ID on the client)
EventTs	Number	Unix timestamp (s) when the event occurred

Field	Type	Description
UserId	String	User ID of the recording robot
TaskId	String	Recording ID, which uniquely identifies a recording task
Payload	JsonObject	The content of this field varies with event type.

If the event type is `301` (`EVENT_TYPE_CLOUD_RECORDING_RECORDER_START`):

Field	Type	Description
Status	Number	<code>0</code> : The recording module was started successfully. <code>1</code> : The recording module failed to be started.

```
{
  "EventGroupId": 3,
  "EventType": 301,
  "CallbackTs": 1622186275913,
  "EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186275",
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

If the event type is `302` (`EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP`):

Field	Type	Description
LeaveCode	Number	<code>0</code> : The recording module ended the recording normally. <code>1</code> : The recording robot was removed from the room. <code>2</code> : The room was closed. <code>3</code> : The server removed the recording robot from the room. <code>4</code> : The server closed the room. <code>99</code> : There were no anchors in the room except the recording robot, which quit after the specified time period elapsed. <code>100</code> : The recording robot quit after timeout. <code>101</code> : The recording robot was removed due to repeat entry by the same user.



```
{
  "EventGroupId": 3,
  "EventType": 302,
  "CallbackTs": 1622186354806,
  "EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186354",
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "LeaveCode": 0
    }
  }
}
```

If the event type is 303 (EVENT\_TYPE\_CLOUD\_RECORDING\_UPLOAD\_START):

Field	Type	Description
Status	Number	0 : The upload module was started successfully. 1 : The upload module failed to be started.

```
{
  "EventGroupId": 3,
  "EventType": 303,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

If the event type is 304 (EVENT\_TYPE\_CLOUD\_RECORDING\_FILE\_INFO):

Field	Type	Description
FileList	String	Name of the M3U8 file generated

```
{
  "EventGroupId": 3,
  "EventType": 304,
  "CallbackTs": 1622191965350,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "FileList": "xx.m3u8"
    }
  }
}
```

If the event type is 305 (EVENT\_TYPE\_CLOUD\_RECORDING\_UPLOAD\_STOP):

Field	Type	Description
Status	Number	<p>0 : The recording task ended and all the files were uploaded to the specified third-party cloud storage service.</p> <p>1 : The recording task ended, but at least one file on the server or in backup storage failed to be uploaded.</p> <p>2 : The files that failed to be uploaded earlier were uploaded to the specified third-party cloud storage service.</p>

```
{
  "EventGroupId": 3,
  "EventType": 305,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

If the event type is 306 (EVENT\_TYPE\_CLOUD\_RECORDING\_FAILOVER):

Field	Type	Description
-------	------	-------------

Field	Type	Description
Status	Number	0 : The migration was completed.

```
{
  "EventGroupId": 3,
  "EventType": 306,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

If the event type is 307 (EVENT\_TYPE\_CLOUD\_RECORDING\_FILE\_SLICE):

Field	Type	Description
FileName	String	Name of the M3U8 file generated
UserId	String	ID of the user whose stream was recorded in the file
TrackType	String	Valid values: audio/video/audio_video
BeginTimeStamp	Number	Unix timestamp on the server when recording started

```
{
  "EventGroupId": 3,
  "EventType": 307,
  "CallbackTs": 1622186289148,
  "EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186289",
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "FileName": "xx.m3u8",
      "UserId": "xx",
      "TrackType": "audio",
      "BeginTimeStamp": 1622186279144
    }
  }
}
```

```
}  
}  
}
```

If the event type is **308** (EVENT\_TYPE\_CLOUD\_RECORDING\_UPLOAD\_ERROR):

Field	Type	Description
Code	String	Error code returned by the third-party cloud storage service
Message	String	Error message returned by the third-party cloud storage service

```
{  
  "Code": "InvalidParameter",  
  "Message": "AccessKey invalid"  
}  
{  
  "EventGroupId": 3,  
  "EventType": 308,  
  "CallbackTs": 1622191989674,  
  "EventInfo": {  
    "RoomId": "20015",  
    "EventTs": 1622191989,  
    "UserId": "xx",  
    "TaskId": "xx",  
    "Payload": {  
      "Code": "xx",  
      "Message": "xx"  
    }  
  }  
}
```

If the event type is **309** (EVENT\_TYPE\_CLOUD\_RECORDING\_DOWNLOAD\_IMAGE\_ERROR):

Field	Type	Description
Url	String	The URL from which the file failed to be downloaded

```
{  
  "EventGroupId": 3,  
  "EventType": 309,  
  "CallbackTs": 1622191989674,  
  "EventInfo": {  
    "RoomId": "20015",  
    "EventTs": 1622191989,  

```

```
"UserId": "xx",
"TaskId": "xx",
"Payload": {
  "Url": "http://xx",
}
}
```

If the event type is 310 (EVENT\_TYPE\_CLOUD\_RECORDING\_MP4\_STOP):

Field	Type	Description
Status	Number	<p>0 : The task ended and all the files were uploaded to the specified third-party cloud storage service.</p> <p>1 : The task ended, but at least one file on the server or in backup storage failed to be uploaded.</p> <p>2 : The task stopped due to an error, for example, failure to download HLS files from COS.</p>
FileList	String	Name of the MP4 file generated

```
{
  "EventGroupId": 3,
  "EventType": 310,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "FileList": ["xxx1.mp4", "xxx2.mp4"]
    }
  }
}
```

If the event type is 311 (EVENT\_TYPE\_CLOUD\_RECORDING\_VOD\_COMMIT):

Field	Type	Description
Status	Number	<p>0 : The recording file was successfully uploaded to VOD.</p> <p>1 : The recording file failed to be uploaded.</p> <p>2 : An error occurred.</p>

Field	Type	Description
UserId	String	ID of the user whose stream is recorded. In the mixed-stream recording mode, this field is empty.
TrackType	String	Valid values: audio/video/audio_video
MediaId	String	Valid values: main/aux
FileId	String	Unique ID of the recording file in VOD
VideoUrl	String	Playback URL of the recording file in VOD
CacheFile	String	Name of the MP4 file converted from the recording file (before it was uploaded to VOD)
Errmsg	String	Error message when <code>Status</code> is not <code>0</code>

Callback for successful upload:

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "TencentVod": {
        "UserId": "xx",
        "TrackType": "audio_video",
        "MediaId": "main",
        "FileId": "xxxx",
        "VideoUrl": "http://xxxx"
      }
    }
  }
}
```

Callback for failed upload:

```
{
  "EventGroupId": 3,
```

```
"EventType": 311,
"CallbackTs": 1622191965320,
"EventInfo": {
  "RoomId": "20015",
  "EventTs": 1622191965,
  "UserId": "xx",
  "TaskId": "xx",
  "Payload": {
    "Status": 1,
    "ErrMsg": "xxx",
    "TencentVod": {
      "UserId": "123",
      "TrackType": "audio_video",
      "CacheFile": "xxx.mp4"
    }
  }
}
```

If the event type is 312 (EVENT\_TYPE\_CLOUD\_RECORDING\_VOD\_STOP):

Field	Type	Description
Status	Number	0 : The task ended after upload was completed. 1 : The task ended due to an error.

```
{
  "EventGroupId": 3,
  "EventType": 312,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

## Best Practices

To ensure the high availability of the recording service, we recommend the following practices when you use the RESTful APIs:

1. Pay attention to the HTTP response after you call `CreateCloudRecording` . If your request fails, fix the problem according to the status code and try again.

The status code consists of two parts, for example `InvalidParameter.SdkAppId` .

- `InternalServerError.xxxxx` indicates that a server error occurred. You can retry until the request succeeds and `TaskId` is returned. We recommend you use the exponential backup algorithm for retry. For example, you can wait for three seconds for the first retry, six seconds for the second, 12 seconds for the third, and so on.
  - `InValidParameter.xxxxx` indicates that a parameter value entered was invalid. Please check the parameter.
  - `FailedOperation.RestrictedConcurrency` indicates that you reached the maximum number (100 by default) of ongoing recording tasks allowed. To raise the limit, please contact technical support.
2. The `UserId` and `UserSig` you pass in when calling `CreateCloudRecording` are for the recording robot. Please make sure that they are different from those of other users in the room. In addition, the room users join from the TRTC client must be of the same type as the room you specify when calling the API. For example, if the room created in the TRTC SDK is a string, the room specified for on-cloud recording must also be a string.
  3. You can obtain the information of a recording file in the following ways.
    - Call `DescribeCloudRecording` 15 seconds after a `CreateCloudRecording` request succeeds. If the status returned is `idle` , it indicates that no audio or video data is available for recording. Please check whether there are anchors publishing data in the room.
    - After a `CreateCloudRecording` request succeeds, if there are anchors publishing data in the room, you can splice the name of the recording file according to the naming conventions.
    - If you have registered on-cloud recording callbacks, the information of recording files will be sent to your server via callbacks.
    - You can specify a COS bucket to save recording files when calling the `CreateCloudRecording` API, so after a recording task ends, you can find the recording file in the COS bucket.
  4. Make sure the validity period of the recording user's `UserSig` is set longer than the duration of the recording task. This is to avoid cases where the high availability scheme fails to resume a recording task after an internet disconnection because the `UserSig` is already expired.